University of Melbourne

Advanced Database Systems

COMP90050

Main Memory Database Management Systems

Authors:
Hammad Javed
Ivn Patricio Valarezo
Saman Bonab

ID: 601099

May 16, 2013

Contents

List of Figures

Abstract

Main Memory Databases (MMDB), which are also referred to as In Memory Databases (IMDB), store their complete data inside RAM which adds to performance of the applications linked to the database. These special type of databases are often misinterpreted as the conventional on disk database, which is optimized for disk I/O, residing in the physical memory. However in contrast, the MMDBs use other techniques for performance in data structures, access methods and durability. These types of databases are best suited for real time embedded applications as well as Business Enterprise solutions which process massive amounts of data. This report discusses briefly the limitations of traditional on disk or Disk Resident Databases (DRDB), optimization techniques used by MMDBs to overcome these limitations including data organization along with methods to ensure reliability within MMDBs.

1 Introduction

Since the inception of computing there are mainly two trends which has significant effects on IT (King, 2011).According to Moores law (Gray & Reuter, 1993) the processing power doubles every 18 to 24 months, this trend has held for almost half a century with no sign of fading.

As the increase in the processing power came with decrease in cost, the number of people adapting to computing has soared with leaps. This necessitates the evolution in the field and resulted in the creation of a whole set of smart and intelligent devices to perform the computations.

More computations and involvements in almost every strata of life from medical to military and space means more data to be processed and its exponential increase. Which tends to be the other driving trend in IT. This implies the processing of larger data by applications and in some cases where speed and time is crucial (real time systems).

Unfortunately the ever increase in processing speed is not linear with respect to other computing components especially storage systems (Boncz, Manegold, & Kersten, 1999). In fact the storage is the bottleneck for their high latency for high volume data processing real time applications.

Hard Disk drives are the cheaper, scalable and reliable solution for data storage but due to mechanical parts involved in the data access it has considerable latency issues as compared to other storage media and thus lies at the bottom of the storage hierarchy in terms of performance (Figure 1).

RAM or DRAM lies below high speed L1/L2 cache or SRAM but above any other storage medium in memory hierarchy system categorized as increase in performance from bottom to top as shown in figure 1. Since the SRAM cannot be extended in size due to hardware limitations therefore DRAM or RAM is the ideal candidate for data storage.

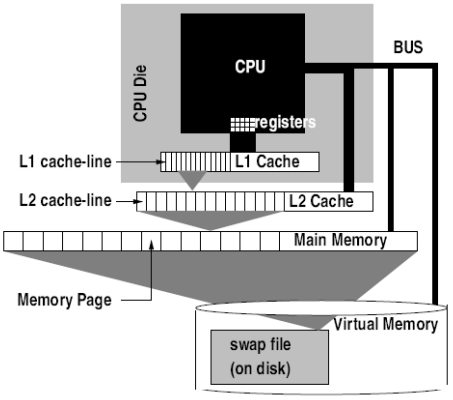


Figure 1: Hierarchical Memory System

Definition As the name implies Main Memory Databases MMDB or In Memory Databases (IMDB) is a database management system which stores the complete database on physical memory or RAM.

1.1 History

Most people argue in memory computing to be a relatively new and unproven technology. This is a myth rather than fact. For ages caching mechanisms is being used in traditional DRDBs which involves the most frequent use data inside RAM for fast access (McObject, 2009) . In addition in-memory tables feature have also been part of some DBMS to hold data inside Main Memory. Also the use of DRDBs in main memory was a practice carried out to get better response times.

This entails that MMDB are not new and considerable work has been done in this regard in 1980s. However they were not popular as the cost of memory and other limitations related to persistence of data were considered a problem. Many experimental MMDBs were developed then like MARS, HALO

and MM-DBMS as part of research along with commercial systems like IBM Fastpath (Garcia-Molina & Salem, 1992).

As of today IBM, Oracle and SAP being the top database vendors in the market have acquired the companies which were developing and marketing in memory database solutions (Metz, 2011). Gartner[1] has placed In Memory computing as part of *Top ten technology trends for 2013* (Penchikala, 2013). This clearly indicates the increase in the demand of MMDBs in market and their adaptation to the solutions in coming years.

1.2 Bounds of MMDBs

MMDBs with their high performance mechanisms are the perfect contenders for embedded systems (Graves, 2002). These systems with the evolution of smarter devices and expanding feature set needs to manage complex data structures which while written from ground up will require extra effort. With MMDBs pointer style low latency data access and simpler optimization techniques the embedded system developers have less things to worry about, stepping MMDBs adaptation to these types of systems. These systems include set-top boxes, network switches and consumer electronics.

The real-time systems require time-critical access to and processing of massive amounts data. Example applications that handle large amounts of data and have stringent timing requirements include telephone switching, radar tracking and others. Furthermore as stated by (Kao & Garcia-Molina, 1993), the real time database system design must not contain latency components like disk I/O operations, massage passing or garbage collection. Thus MMDBs architecture fits perfectly into the real-time application scenario.

The role of the MMDBs has greatly expanded over the last couple of years as high-end 64-bit servers can now accommodate databases that are quite large even terabytes in size. Therefore the applications for large IMDBs further include many popular large-scale Web applications and social networking sites.

Nowadays, the data within an enterprise is distributed throughout a wide range of applications and stored in disjoint locations. Analytical reporting with amalgamated view on the data as a whole is a cumbersome and time-consuming process. Besides these reports are not entirely based on the operational data, but on combined data from a data warehouse by transforming it through ETL[2]process. Hasso has pointed out in (Plattner & Zeier, 2011) that In-Memory data management assists the analytical operations resulting in planning, forecasting and faster decision making by separating it from operational data processing. Thus resulting in innovative and hybrid style of enterprise applications residing partially or fully in memory.

2 MMDB vs. DRDB : A Comparison

In MMDBs, since the complete database physically resides inside Main Memory, a Main Memory Database Management System eliminates the most expensive Disk I/O operations which are the performance bottle neck in traditional DRDBs[3]. Before delving deeper into variances it is pertinent to mention few important common questions and myths about MMDBs.

Putting a DRDB in main memory will deliver us the same performance results as a MMDB or *MMDBs are the same traditional databases inside RAM.* This notion is not true at all. Although the traditional DRDB inside main memory out performs its instance on disk but while its comparison with pure MMDB the performance gap is far too much (McObject, 2012). Linux systems and now windows as well have the capability to create a RAM disk (File system inside the main memory). But a traditional DRDB deployed on such a virtually fast hard drive doesn't provide the same benefits of a pure MMDB. In-memory databases are less complex than their on disk counterparts fully deployed in physical memory and thus require minor usage of processing cycles and RAM.

Another popular feature which most database providers provide in their DBMSs are the creation of memory tables, through which certain tables can be designated for all-in-memory handling (McObject, 2009). So does this mean that *with these memory features, DRDBs can compete with MMDBs performance?*. Not really as the problem remains the same as memory tables not only don't change the

[1]Gartner, Inc. is an American information technology research and advisory firm head-quartered in Stamford, Connecticut, United States

[2]ETL (Extract, Transform and Load) is a process in data warehousing responsible for pulling data out of the source systems and placing it into a data warehouse

[3]Disk Resident Databases. Throughout this report we will use the terms Disk-Resident, on-disk and disk-based interchangeably to refer to traditional DBMSs that are hard-wired to store records to a file system on persistent media. While this media is usually a hard disk, it can also be a flash memory stick, solid state drive (SSD) or some other storage.

database design assumptions but also have certain additional restrictions. For example, in MySQL, memory tables cannot contain BLOB or TEXT columns and the maximum table size is fixed. Furthermore in contrast to MMDBs, the space that is freed up by deletion of tuples in an in-memory relation can only be used by the same relation, which is wastage of precious space inside RAM.

IMDS Performance Can Be Obtained By Deploying a Traditional DBMS on a Solid-State Drive (Flash). Solid State Drives (SSDs) are nowadays another popular and widely used data storage devices in systems all over the world. Embedded and real time systems and even personal computers and laptops utilize their low latency data access features for specific applications. Since no moving part is included in SSDs, they outperform the hard disks in terms of disk I/O operations resulting in better responsiveness for database operations (McObject, 2009) but other performance drains in the form of caching, disk I/O, data transfer etc will remain there. Moreover, even though these NAND based devices are faster and better in performance than hard disks but they are nowhere near to DRAMs amazing speeds (Research, 2010).

Comparing MMDBs with DRDBs we can find at least 4 key differences:

- Cache Maintenance
- Data Transfer
- Transaction Processing
- Concurrency Control

2.1 Cache Maintenance

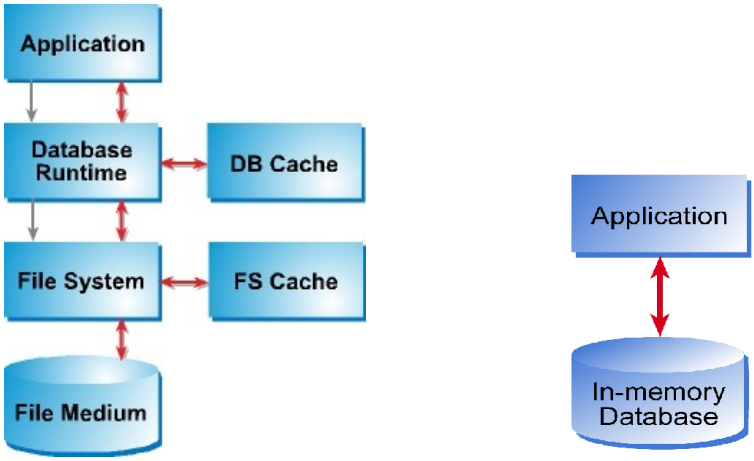
Caching is a sophisticated mechanism that is used by all on disk traditional database management systems to improve performance due to high latency issues in hard disk. This incorporates the most accessed records to be placed inside main memory (database cache) for quick access. Sufficient overhead is involved in management of this mechanism which in case of MMDB is not required since all the data in readily available in RAM and can be accessed directly. Removal of caching logic removes the two expensive operations:

- Cache Synchronization : This ensures that database image in memory is synched with its physical contents on disk. Thus preventing the application to read inconsistent data.
- Cache lookup : is another process that determines if data that is requested by the application is in cache or not. If yes then it is fetched, otherwise a cache miss occurs and the corresponding page has to be read from the disk and load in memory. Furthermore it also takes responsibility and additional logic for protection of dirty data to be read by other applications before flushing it back to the disk after the transaction commits.

2.2 Data Transfer

Main performance degradation in DRDBs comes with the necessary file I/O operations that are performed on data as requested by the transactions from the application. This data undergoes extensive handling and unnecessary transfers before the application can use it as shown in Figure2a. The chronological steps involved in the request can be summarized as follows by Graves in (Graves, 2002):

- application requests the data item from the database runtime through the database API.
- The database runtime instructs the filesystem to retrieve the data from the physical media.
- The filesystem makes a copy of the data for its cache and passes another copy to the database.
- The database keeps one copy in its cache and passes another copy to the application.
- The application modifies its copy and passes it back to the database through the database API.
- The database runtime copies the modified data item back to database cache.
- The copy in the database cache is eventually written to the filesystem, where it is updated in the filesystem cache.



(a) On Disk Database (b) In-Memory Database

Figure 2: Data Transfer

- Finally, the data is written back to the physical media.

In contrast the MMDB system entails little or no data transfer. The application may make copies of data in local variables but even this is not required as the MMDB system provides a direct pointer to the application, enabling it to deal with the data directly. Thus eliminating the multiple data transfer and streamlining processing . Besides cutting these extra copies saves memory and processor consumption and simplifies the design of MMDB systems. Data is copied directly from the IMDS to the application, and back from the application to the database, as shown in Figure2b. Database API ensures that the data remains secure as the pointer is accessed through it. Moreover, in case of traditional databases, the cache flushed to disk require random access as data might be stored at different locations and thus adds up to the latency by disk seek, which in case of MMDBs is not a problem.

2.3 Transaction Processing

Transaction logging is a permanent feature of traditional on disk DBMS to ensure the ACID property of the database. We examine this memory-intensive processing in terms of atomicity and durability.

Atomicity In order to achieve atomicity, DRDBs use their hard-wired logging feature. The recovery process comprises of committing or rolling back transactions from log files. The IMDB should be able to handle the effects of unsuccessful transactions. In general terms, the problem has to explicitly with the data existent in the volatile memory, since all successful transaction are in a successful state only after been committed to the logging infrastructure. In order to comply with this transactional integrity, MMDBs use before and after-images of the database. When the application commits the transaction, the memory for before images are freed and are returned to memory pool. If it decides to abort due to any reason, the before-images are restored and newly inserted images are freed to be returned to the memory pool in a very fast and efficient way as compared to DRDBs.

Durability The logs are flushed to disk after transaction commits in DRDBs. The recovery process comprises of committing or rolling back transactions from log files in the event of a restart. In the event of a power or any other failure, the in-memory database image is lost forever as the memory is volatile and needs to be built again through some mechanism. This disadvantage of MMDBs suits most embedded systems like program guide application in set-top boxes which requires a connection to a satellite or cable head-end for downloading, a network switch that discovers network topology on startup, or a wireless access point that is provisioned by a server upstream. However in case of enterprise applications durability factor has to be addressed in an efficient way, as described later in this report, but it still contributes to performance degradation in MMDBs.

2.4 Query Optimization

Traditional on Disk Database query optimizer output results which aimed at minimizing the disk I/O operations while a MMDB query optimizer has a diverse goal for reducing CPU cycles and precious main memory.

3 Characteristics/Strong points of MMDB

Pure In Memory Database The differences between MMDB and traditional disk-based DBMS entails that DRDBs cannot be used as MMDBs and therefore a true In-Memory DBMS has to be written from bottom to top (Mcobject RealImmitation). The underlying goal of DRDBs is the minimizing of disk I/O even at the cost of CPU cycles in contrast to MMDBs goal for minimizing CPU cycles and Memory utilization. This implies that MMDBs can use the spared CPU cycles elsewhere to address durability and in some cases to create hybrid databases.

No Cache Maintenance As already discussed in previous section, cache maintenance and management requires considerable resources which is eliminated in case of MMDBs. Cache synchronization and lookup processes adds to the function calls by processor and thus are expensive if aggregated together. MMDBs are indifferent to these mechanisms.

Lightweight Being lightweight is one of the common advantages IMDBs share and are helpful in embedded systems. This kind of databases is really simple: they don't implement any of the complex mechanisms used by traditional database to speed up the disk I/O therefore a small footprint. Besides, as the data is readily available in memory and can be accessed by use of pointers, there is no need for storing redundant data in indexes (discussed later in data organization)

High Performance is another common feature for every IMDB, because they all store the whole database in main memory, avoiding disk access and minimizing CPU cycles and memory.

Sound Memory Management Another key characteristic of Main Memory Databases are their sound memory management which derives that memory is a precious resource and needs to be looked after. This means that data alignment issues and sharing of heap memory through serialization be included in considerations for a MMDB memory manager as discussed in (?, ?).

New Breed of Application MMDBs can be used as cache managers for large DBMS instead of being the primary database of application. This resulted in new breed of OLAP and OLTP applications which enables faster ETL processes and in-memory analytics.

4 Issues in MMDBs

Although MMDBs are currently the new hype but before jumping on the bandwagon few issues and limitations needs to be discussed. There are methods which address almost all of these issues. Most in-memory database vendors provide the facility to the application developer to choose required quality over others depending upon the logic and need.

Scalability . According to Independent oracle users group data growth survey(?, ?), the data growth of surveyed organizations is estimated around 25% per year of their existing data. Is this a bad news for in-memory databases?.To overcome this problem there are two approaches

- Either get a better machine with more main memory or
- Use distributed architecture by adding cheap commodity machines.

Problem with first one is the cost involved with the new technology and second what is the guarantee that it will keep up with the growth rate? In case of distributed architecture, there are issues pertaining to the coordination and overhead between the nodes as well as the network latency but the system can scale indefinitely. A concept of RAMCloud as discussed in (?, ?) is a viable solution but it comes with disadvantages of energy and space along with latency in cross-datacenters replication. However according to this paper, the failure of servers will be handled by the RAMCloud cluster within the recovery time objective of 1-2 seconds.

Durability & High Availability As RAM is a volatile memory by design and in-memory databases keep the primary copy of records in Main Memory, all the stored data is lost whenever the system is powered off due to any reason. Therefore additional measures like check-pointing, shadow-pages along with logging needs to be built in to achieve durability. Similarly to achieve high availability, replication mechanisms on some persistent medium requires further implementation. These durability means are discussed later in this report.

Slow Start-up All durability and High Availability mechanisms, to an in-memory database system, make the database's startup terribly slow, because the database image is not stored on the hard disk, or on another persistent memory, and it needs to be reconstruct at every reboot, for example by reading the transaction log file and repeat all the operations executed before the power off. This issue can be mitigated by snapshots. But snapshot decreases performance of database once the system is executing the snapshot (McObject, 2009). Built-up of hybrid systems can also address this issue along with durability and High Availability.

Application Implementation As discussed in section 2 there are considerable differences between DRDB and pure In-Memory database, therefore the applications built for use with traditional DRDBs has to be modified to be used with an MMDB. This makes sense as the query optimization, memory management and the trade-off in performance by durability and high availability along with no disk I/O latency has to be taken into account while developing an application for MMDB.

Shared Memory for processing Main memory's primary usage lies with the processor for fast access and processing. This implies that it has to share the memory with the database and thus even the database API protects the data from being accessed but still the data is vulnerable to software errors.

5 Data Organization

In this section we will discuss the common data organization and access methods used by MMDBs. Relational Databases have been the backbone of business applications for more than 20 years, trying to provide companies with a management information system for a set of applications. During all this time, we have dreamed with the possibility of having all the information at our fingertip, we even have sold this idea(Plattner, 2009).

But due to many reasons, we have not been able to offer this. And our systems have been separated in two main different groups: The online transaction processing (OLTP) and on line analytical processing (OLAP).

5.1 OLTP and OLAP

Under the OLTP classification, could be grouped systems with a high rate of day to day transactions, most of the database systems (as we know them) are mainly used for transaction processing. Some examples of OLTP systems are financial, sales, Manufacturing, order entry, banking transaction processing, human resources. All this systems performs relatively well mainly because they work on a small portion of the data. The TCP-C benchmark publish an average processing of 100.000 such sales transactions per second on a powerful system (Kemper & Neumann, 2011).

On the other hand, analytical, business intelligence and financial planning application were moved out to separated systems (for more flexibility and better performance). The OLAP Systems could be aimed toward specific task related to the company data warehousing.

Even though, this solutions are different in context, both are still based on *Relational theory* but with different technical approaches. The main purposes of this separation could be generalized into *performance* and maybe *technical* issues. Even though both systems keep the essence in terms of relational theory, there are some important differences between them (Plattner, 2009).

Moreover, recent development in the field of OLAP and the increased availability of main memory (sufficient enough to hold a completed compressed database) have enabled the processing of complex analytical requests in a fraction of a second and thus ease the development of new business processes and applications. The next step seems obviously to undo the separation between OLTP and OLAP and all requests be handled on a combined data set.

Now that the main memory is abundant, we have gone back to see the possibility of having a all-in-one MMDB. SAP[4] is one of the most enthusiastic companies pushing the use of Main Memory Databases for a mixed OLTP & OLAP environment (Plattner, 2009), the company advertises SAP HANA as a generic MMDB solutions.

In this direction, it is suitable to estimate the challenges toward this join in technical perspectives, specifically, we have defined two main critical points related to data organization: The improved MMDB indexing system and the data Storage.

5.2 MMDB Indexing

It is mandatory to talk first about the differences between MMDB and on-disk DB, at first instance: In a MMDB Index, the main goal is to redesign the data structure and algorithm to make efficient use of *CPU* and *memory space* rather than minimize disk access, since we don't have disk of course. On the other hand, disk-oriented index structure are designed to minimize disk access and minimize disk space, this mainly because traditional database systems are CPU bounded because they spend considerable effort to avoid disk accesses.

A Main Memory oriented index structure is designed to reduce overall computation while using as little memory as possible (Lehman & Carey, 1986), since everything is in memory, MMDB Index could store only pointers to tuples or structures in the main memory (the actual data) instead of storing attribute values as on-disk DB usually does, decreasing efficiently the index footprint.

Although a MMDB system could be arranged in different ways, two main types of *Main Memory Index Structure* could be specified: Order Preserving index structures and randomized (Lehman & Carey, 1986). Under the order preserving group could be summarized: arrays, AVL Trees, B Trees and T-Trees. T-Tree is one of the most important and well know structure, named T-Tree after its 'T' shape. On the other randomized group falls: Chained Bucket Hashing, Linear Hashing and Extendible Hashing.

Although, the performance observed in a MMDB could be outstanding compared to a on disk DB, the index structure is a critical bottleneck (Leis, Kemper, & Neumann, n.d.). The T-Tree (and one of the most developed) index technology involved in MMDB was proposed 25 years ago, and was designed based on the AVL and B Trees. Acording to Leis et al. (n.d.), "the dramatic processor architecture changes have rendered T-trees, like all traditional binary search trees, inefficient on modern hardware".

TODO: maybe talk about details of t-trees...

5.2.1 Column-Store Database System

Column-oriented databases are based on vertical partitioning by columns, this is not a new idea, Column-oriented store where considered around 70s. The core of the functioning is based in the fact that each column is stored separately, having the logical scheme built around unique positioning keys (Krueger et al., 2011). This vertical data organization offers particular advantages to reading fewer attributes. According to (Plattner, 2009), in real world applications, "only 10% of the attributes of a single table are typically used in one SQL statement", this could imply that: if during requests no unnecessary columns must be read, we could just use the columns independently of the related column attributes. Row oriented structures on hard drives normally can't perform this independence.

As a first premise, recent improvements in parallel processing of modern CPUs, could lead to think that a column oriented approach today seems to be ideal. The Column oriented storage has the following advantages:

- Column Store performs outstanding on modern CPUS.
- The Storage model is based on vertical fragmentation.
- Column Store Allow data compression
- Better memory consumption performance
- Data arranged in columns are better suited for parallel processing because of its independence.

[4]SAP AG: A German multinational software corporation that makes enterprise software to manage business operations and customer relations.

Although, column oriented storage presents a lot of interesting features, it's worth to mention that there are also some drawbacks that kept this technology behind row oriented storage: for example: Column stores are expensive to update, but also having all data in main memory overcomes this limitation. Also column store increases seek time, but this is a concern only for on-disk DB systems.

From (Plattner, 2009), it seems that although more memory has been always useful, the database systems for OLTP where not well adopted in parallel environments, one of the reasons of this were problems like deadlocks and temporary locking in parallel transactions. So in general terms has not been a good idea. Really? review this since Plattner is one of the pro Column Oriented store.

Also from (Plattner, 2009), initial tests regarding in-memory databases using relational type based on *row storage* has not shown notable performance over RDBMSs. The opportunity for *column based storage* now raises due the abundant availability of main memory.

Compression is another characteristic that Column-oriented ease. The redundancy of column store and the homogeneous domain is an advantage and a convenience for compression techniques. All the data within a column belongs to the same type, and the entropy[5] is relatively low.

TODO: talk more about column oriented, put some graphics

6 Concurrency Control

6.1 IMDB Data Recovery

7 In Memory database solutions on market

In recent years, a number of database vendors have marketed in-memory database solutions targeting different application developers. Some aim at real time embedded systems like McObjects eXtreme DB (?, ?). This versatile product is available in different flavors, each addressing a particular quality of traditional DBMS e.g. transaction logging edition, High Availability edition and hybrid Fusion edition which combines all the traditional DBMS properties with performance of a true in-memory database. eXtreme DB uses R-trees for geospatial data, Patricia tries for IP/telecom, KD-trees for multi-dimensional data and Query-by-Example (QBE), B-trees and hash indexes as access methods.

The big database giant SAP has introduced its HANA database which is positioned as the core of the SAP HANA Appliance to support complex business analytical processes(OLAP) in combination with transactionally consistent operational workloads(OLTP) (?, ?). The in-memory computing engine allows HANA to process data stored in RAM as column and row stores according to the access patterns, allowing faster and better performance and aiding in decision making.

Oracle is offering Times Ten as In-Memory database, which can be used as a preprocessing cache for its traditional flagship RDBMS or as a stand-alone database product. TimesTen uses Hash & T-tree indices for data representation along with transactional logging and database checkpointing as a measure to provide durability (?, ?).

IBM has acquired SolidDB in 2008 and markets it as IMDB which maintains durability by keeping two separate but synchronized copies of database all the times as well as permanent log file stored on disk. It uses VTrie Indexes for Memory Tables and B+Tree for Disk tables. Recovery happens in less than one second through Hot Standby support (?, ?).

ALTIBASE HDB is an in-memory database that features a hybrid (memory plus disk) architecture. It is not open source, but it is free for non-commercial use and free for small business (?, ?).

Table 1: Summary of MMDBs

	Times Ten	SAP HANA	eXtreme DB	IBM Solid DB	Altibase HDB

[5]The entropy could be defined as the similarities of data

Data Organization	Hash &; T tree	Row Store, Column Store	R-trees for geospatial data, Patricia tries for IP/telecom, KD-trees for multi-dimensional data and Query-by-Example (QBE), B-trees, hash indexes	VTrie Indexes for Memory tables and B+tree for Disk tables	Spatial Indexes
Recovery / Durability & High Availability	Blocking &; Fuzzy Check pointing with WAL logging Synchronous &; Asynchronous Replication	WAL, shadow Paging and save points	synchronous (2-safe) and asynchronous (1-safe) replication, with automatic failover &; Logging	Checkpointing, synchronous and asynchronous, Hot standby replication	checkpointing and transaction logging.
Concurrency Control	Locks &; latches with two isolation levels (Read Committed &; Serialization)	MVCC in addition with a time-travel mechanism	MVCC	Pessimistic &; optimistic concurrency control	MVCC
Support Hybrid	Yes	Yes	Yes	Yes	Yes
Cache support	Yes	Yes	Yes	Yes	Yes
OLAP Support	No	Yes	No	No	No

8 Applications

9 Conclusions

9.1 Conclusion regarding OLTP and OLAP merging

We could talk about: *MonetDB* , *VoltDB*

References

Boncz, P., Manegold, S., & Kersten, M. (1999). Database architecture optimized for the new bottleneck: Memory access. In *Proceedings of the international conference on very large data bases* (pp. 54–65).

Garcia-Molina, H., & Salem, K. (1992). Main memory database systems: An overview. *Knowledge and Data Engineering, IEEE Transactions on*, *4*(6), 509–516.

Graves, S. (2002). In-memory database systems. *Linux Journal*, *2002*(101), 10.

Gray, J., & Reuter, A. (1993). *Transaction processing*. Address: Kaufmann.

Kao, B., & Garcia-Molina, H. (1993). *An overview of real-time database systems* (Technical Report No. 1993-6). Stanford University. Available from `http://ilpubs.stanford.edu:8090/39/`

Kemper, A., & Neumann, T. (2011). Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. *2011 IEEE 27th International Conference on Data Engineering (ICDE)*, 195. Available from `https://ezp.lib.unimelb.edu.au/login?url=https://`

search-ebscohost-com.ezp.lib.unimelb.edu.au/login.aspx?direct=true\&db=edb\&AN=80280785\&site=eds-live

King, E. (2011). *The growth and expanding application of in-memory databases.* www.loyola.edu/department/lattanze/working-papers.aspx.

Krueger, J., Huebner, F., Wust, J., Boissier, M., Zeier, A., & Plattner, H. (2011). Main memory databases for enterprise applications. In *Industrial engineering and engineering management (ie&em), 2011 ieee 18th international conference on* (pp. 547–557). Potsdam Germany: University of Potsdam.

Lehman, T. J., & Carey, M. J. (1986). A study of index structures for main memory database management systems. In *Conference on very large data bases* (Vol. 294). University of Wisconsin, Madison, WI 53706: Computer Science Department.

Leis, V., Kemper, A., & Neumann, T. (n.d.). The adaptive radix tree: Artful indexing for main-memory databases. *Technische Universitat Munchen*.

McObject. (2009). *In-memory database systems:myths and facts.*

McObject. (2012). *In-memory vs. ram-disk database systems:a linux-based benchmark.*

Metz, C. (2011). *Say hello to memory. its the new disk.* http://www.wired.com/wiredenterprise/2011/12/memory-is-the-new-disk.

Penchikala, S. (2013). *Gartner's technology trends for information infrastructure: Big data, nosql and in-memory computing.* http://www.infoq.com/news/2013/04/gartner-technology-trends.

Plattner, H. (2009). A common database approach for oltp and olap using an in-memory column database. In *Proceedings of the 2009 acm sigmod international conference on management of data* (pp. 1–2). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1559845.1559846

Plattner, H., & Zeier, A. (2011). *In-memory data management: an inflection point for enterprise applications.* Springerverlag Berlin Heidelberg.

Research, I. (2010). *The state of solid state storage* (Tech. Rep.).