In Memory Database - Optimal Performance Solutions

Laxmi Ghate[1] Vaibhav Dhage[2]
[1]M.E. 1st Year [2]Assistant Professor
[1,2]Department of Computer Science & Engineering
[1,2]Dr. Sau. Kamaltai Gawai Institute of Engineering & Technology, Darapur, Sant Gadge Baba Amravati University, Amravati, Maharashtra, India

Abstract— for several decades there have been two major trends that have had a critical impact on IT systems. The first is the ever increasing speed of CPU and second trend is the exponential growth of data size due to increased number of users, devices and IT applications in almost all areas of life. This leads to the problem of capturing, storing, managing and analyzing terabytes or petabytes of data, stored in multiple formats, from different internal and external sources. Moreover, new applications scenarios like weather forecasting, trading, artificial intelligence etc. need huge data processing in real time. Also users are now demanding more decision-enabling information on the tip of the finger. Hence, organizations are adapting to new data management solutions such as In Memory Database System (IMDS) or Main Memory Database systems (MMDB) which store their data in main physical memory and provide very high-speed access compared to traditional On Disk DBMS. This phase of the work provides overview of MMDB system, its characteristic, different available products and challenges associated with choosing and implementing MMDBs. Overview and features of Oracle Times Ten Database are mentioned as an example to explain the concept. In the next phase of the work, experimental study is proposed to arrive at optimal software, hardware and database configurations to ensure optimal performance for most widely used applications and user scenarios.

Key words: In-memory; MMDB, IMDS, Times Ten, Data replication, Caching, Indexing, Buffer, NUMA; NVRM

I. INTRODUCTION

With the increasing demand of real time data processing, traditional (on-disk) database management systems are in tremendous pressure to improve the performance. With the increasing amount of data, which is expected to touch 40ZB (1ZB = 1 billion terabytes) by 2020, means 5247 GB of data per person [1], and with traditional DBMS architecture, it is becoming more and more challenging to process the data and to produce analytical results in almost real time. For on-disk databases, disk I/O operations are the main bottleneck, which are very slow operations and can't be optimized beyond a limit being mechanical in nature. Although traditional on-disk DBMS have tried to improve on this by introducing various caching techniques to cache the frequently accessed data, however, it comes at the cost of synchronization of cache with disk and vice versa and to implement various complex logic to manage transaction and resources, which itself pose as a limitation to performance. So what is the way forward?

Here comes the in-memory database system concept, which actually changed the whole architecture paradigm for the database management system. An in-memory database system or main-memory database system is a breed of database management system that stores data entirely in main memory instead of keeping it on disk [2]. With decreasing cost of main memory, and advance technological innovations, it becomes quite feasible to store large amount of data in main memory.

An in-memory database (IMDB, also known as a main memory database or MMDB) is a database whose data is stored in main memory to facilitate faster response times. Source Data is loaded into system memory in a compressed, non-relational format. In-memory databases streamline the work involved in processing Queries. Traditional DBMSes move data from disk to memory in a cache or buffer pool when it is accessed. Moving the data to memory makes re-accessing the data more efficient, but the constant need to move data can cause performance issues. Because data in an IMDBMS already resides in memory and doesn't have to be moved, application and query performance can be significantly improved.

In-memory database systems have broad uses but are primarily used for real-time applications that require high performance and very low latency. Use cases [3] for IMDBMSes include applications with real-time data management requirements such as telecom, finance, defense, intelligence, weather forecasting, trading application, social networking websites, artificial intelligence etc. Applications that require real-time data access, including call center apps, travel and reservations apps and streaming apps are also good candidates for IMDBMS. Another important use for in-memory database systems is in real-time embedded systems such as IP network routing, telecom switching, industrial control etc.

II. LITERATURE REVIEW

A. Architecture of MMDB

A general MMDB architecture consists of a main memory implemented via standard RAM and an optional non-volatile (stable) memory. The main memory holds the primary copy of the database, while the stable memory holds the log and a backup copy of the database. A logger process flushes the logs asynchronously to the disk. Alternatively, an additional nonvolatile memory may be used as a shadow memory. This memory is intended to hold updates performed by active transactions and the tail end of the log that contains information about those updates [4]. Fig. 1 shows a very simply high level architecture of MMDB. This is a major paradigm shift in DBMS design approach, which triggers many other design optimizations. Now DBMS needs not to be worried about optimizing the disk I/O operations, and about caching like techniques for these optimizations. Whole designed will be geared to have high performance data access, manipulation and analysis relying on main memory data store.

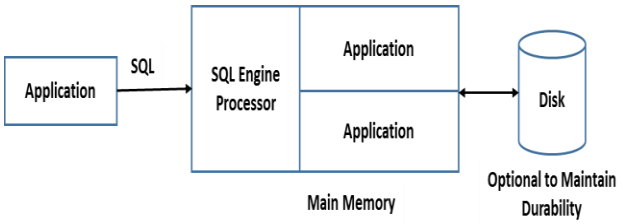

Fig. 1: High Level Architecture of MMDB

To ensure the durability of the data in an IMDBMS, it must be moved from memory to persistent, non-volatile storage periodically. This is important because data stored in memory will not survive an outage. There are various ways to achieve data persistence. One option is transaction logging, in which periodic snapshots of the in-memory database get written to non-volatile storage media. If the system fails and must be restarted, the database can then be rolled back or forward to the last completed transaction.

Another option for maintaining data persistence is to create additional copies of the database on non-volatile media. Yet another option is to utilize non-volatile RAM (NVRAM), such as battery RAM that is backed up by a battery or ferroelectric RAM (FeRAM) that can maintain data when the power is turned off. Hybrid IMDBMSes, which store data on hard disk drives as well as on memory chips, are also an option

The below section describes the system components of Oracle TimesTen MMDB as an example. TimesTen consists of

- Shared libraries,
- Memory-resident data structures,
- System processes,
- Administrative programs,
- Checkpoint files and Log files on disk

Application-Tier Shared Libraries: The components that implement the functionality of TimesTen are embodied in a set of shared libraries that developers link to their applications and execute as a part of the application process. This shared library approach is in contrast to a more conventional RDBMS, which is implemented as a collection of executable programs that applications connect to, typically over a client/server network.

Memory-Resident Data Structures: In-memory databases are maintained in an operating system's shared memory segments, and contain all user data, indexes, system catalogs, log buffers, lock tables, and temp space. Multiple applications can share a TimesTen database, and a single application can access multiple TimesTen databases on the same system.

System Processes: Background processes provide services for startup, shutdown, and application failure detection at the system level, and provide loading, check pointing, and deadlock handling at the database level. There is one instance-wide TimesTen daemon (a system may have multiple instances, i.e., multiple installations of the TimesTen software), and a separate sub daemon for each database.

Administrative Programs: Utility programs are explicitly invoked by users, scripts, and applications to perform services such as interactive SQL, bulk copy, backup/restore, database migration, and system monitoring.

Checkpoint and Log Files: Changes to the database and transaction logs are written to disk periodically. Should a database need to be recovered, TimesTen merges the database checkpoint on disk with the completed transactions that are still in the log files. Normal disk file systems are used for checkpoints and log files.

B. Characteristics of MMDB

In order to provide efficiency and enforce ACID properties (Atomicity, Consistency, Isolation, and Durability), In-memory data management and processing focuses on following aspects as summarized in Table 1 below.

Aspect	Optimization Area
Index	cache consciousness, time/space efficiency
Data Layout	cache consciousness, space efficiency
Parallelism	linear scaling, partitioning
Concurrency Control/ Transaction Management	overhead, correctness
Query Processing	code locality, register temporal locality, time efficiency
Fault Tolerance	durability, correlated failures, availability
Data Overflow	locality, paging strategy, hot/cold classification

Table 1: Optimization Aspects on In-memory Data Management and Processing

Indexes: Although in-memory data access is extremely fast compared to disk access, an efficient index is still required for supporting point queries in order to avoid memory-intensive scan. Indexes designed for in-memory databases are quite different from traditional indexes designed for disk-based databases such as the B+-tree, because traditional indexes mainly care about the I/O efficiency instead of memory and cache utilization. Hash-based indexes are commonly used in key-value stores [6] and can be further optimized for better cache utilization by reducing pointer chasing. However hash based indexes do not support range queries, which are crucial for data analytics and thus, tree-based indexes have also been proposed, such as T-Tree, Cache Sensitive Search Trees, Fast Architecture Sensitive Tree (FAST) etc., some of which also consider pointer reduction.

Data Layouts: In-memory data layouts have a significant impact on the memory usage and cache utilization. Columnar layout of relational table facilitates scan-like queries/analytics as it can achieve good cache locality [7] and can achieve better data compression, but is not optimal for OLTP queries that need to operate on the row level. It is also possible to have a hybrid of row and column layouts, such as PAX which organizes data by columns only within a page, and SAP HANA with multi-layer stores consisting of several delta row/column stores and a main column store, which are merged periodically. In addition, there are also proposals on handling the memory fragmentation problem, such as the slab-based allocator in Memcached, and log-structured data organization with periodical cleaning in RAMCloud, and better utilization of some hardware features (e.g., bit-level parallelism, SIMD), such as BitWeaving and ByteSlice.

Parallelism: In general, there are three levels of parallelism, i.e., data-level parallelism (e.g., bit-level parallelism, SIMD), shared-memory scale-up parallelism (thread/process), and shared-nothing scale-out parallelism (distributed computation). The bit parallel algorithms fully unleash the intra-cycle parallelism of modern CPUs, by packing multiple data values into one CPU word, which can be processed in one single cycle [8]. Intra-cycle parallelism performance can be proportional to the packing ratio, since it does not require any concurrency control protocol. SIMD instructions can improve vector style computations greatly, which are extensively used in high-performance computing, and also in the database systems. Scale-up parallelism can take advantage of the multi-core architecture of supercomputers or even commodity computers, while scale-out parallelism is highly utilized in cloud/distributed computing. Both scale-up and scale-out parallelisms require a good data partitioning strategy in order to achieve load balancing and minimize cross-partition coordination.

Concurrency control/transaction management: Concurrency control/transaction management becomes an extremely important performance issue in in-memory data management with the many-core systems. Heavy-weight mechanisms based on lock/semaphore greatly degrade the performance, due to its blocking-style scheme and the overhead caused by centralized lock manager and deadlock detection [9]. Lightweight Intent Lock (LIL) was proposed to maintain a set of lightweight counters in a global lock table instead of lock queues for intent locks. Very Lightweight Locking (VLL). Further simplifies the data structure by compressing all the lock states of one record into a pair of integers for partitioned databases. Another class of concurrency control is based on timestamp, where a predefined order is used to guarantee transactions' serializability, such as Optimistic Concurrency Control (OCC) and Multi-Version Concurrency Control (MVCC). Furthermore, H-Store seeks to eliminate concurrency control in single-partition transactions by partitioning the database beforehand based on a priori workload and providing one thread for each partition.

HyPer isolates OLTP and OLAP by forking a child process (via fork () system call) for OLAP jobs based on the hardware-assisted virtual snapshot, which will never be modified. DGCC is proposed to reduce the overhead of concurrency control by separating concurrency control from execution based on a dependency graph. Hekaton utilizes optimistic MVCC and lock-free data structures to achieve high concurrency efficiently. Besides, hardware transactional memory (HTM) is being increasingly exploited in concurrency control for OLTP.

Query processing: Query processing is going through an evolution in in-memory databases. While the traditional Iterator- /Volcano-style model facilitates easy combination of arbitrary operators, it generates a huge number of function calls (e.g., next ()) which results in evicting the register contents. The poor code locality and frequent instruction miss predictions further add to the overhead [10]. Coarse grained stored procedures (e.g., transaction-level) can be used to alleviate the problem, and dynamic compiling (Justin-Time) is another approach to achieve better code and data locality [10]. Performance gain can also be achieved by optimizing specific query operation such as join.

Fault tolerance: DRAM is volatile, and fault-tolerance mechanisms are thus crucial to guarantee the data durability to avoid data loss and to ensure transactional consistency when there is a failure (e.g., power, software or hardware failure). Traditional write-ahead logging (WAL) is also the de facto approach used in in-memory database systems [11]. But the data volatility of the in-memory storage invalids the necessaries of the persistent undo log or completely disables it in some scenarios. To eliminate the potential I/O bottleneck caused by logging, group commit and log coalescing, and remote logging are adopted to optimize the logging efficiency. New hardware technologies such as SSD and PCM are utilized to increase the I/O performance. Recent studies proposed to use command logging, which logs only operations instead of the updated data, which is used in traditional ARIES logging studies how to alternate between these two strategies adaptively. To speed up the recovery process, a consistent snapshot has to be check pointed periodically, and replicas should be dispersed in anticipation of correlated failures. High availability is usually achieved by maintaining multiple replicas and stand-by servers [11], or relying on fast recovery upon failure Data can be further backed up onto a more stable storage such as GPFS, HDFS and NAS to further secure the data.

Data overflow: In spite of significant increase in memory size and sharp drop in its price, it still cannot keep pace with the rapid growth of data in the Big Data era, which makes it essential to deal with data overflow where the size of the data exceeds the size of main memory. With the advancement of hardware, hybrid systems which incorporate Non- Volatile Memories (NVMs) (e.g., SCM, PCM, SSD, Flash memory) become a natural solution for achieving the speed. Alternatively, as in the traditional database systems, effective eviction mechanisms could be adopted to replace the in memory data when the main memory is not sufficient. propose to move cold data to disks, and re-organizes the data in memory and relies on OS to do the paging, while [12] introduces pointer swizzling in database buffer pool management to alleviate the overhead caused by traditional databases in order to compete with the completely re-designed in-memory databases. UVMM taps onto a hybrid of hardware-assisted and semantics-aware access tracking, and non-blocking kernel I/O scheduler, to facilitate efficient memory management. Data compression has also been used to alleviate the memory usage pressure.

C. Core Technologies for In-Memory Systems

In this section, we shall introduce some concepts and techniques that are important for efficient in-memory data management, including memory hierarchy, non-uniform memory access (NUMA), transactional memory, non-volatile random access memory (NVRAM), data replication, caching, query optimization & processing, and buffer pool management. These are the basics on which the performance of in-memory data management systems heavily rely.

Data Replication Technology: When high availability or workload distribution is desired, data replication can be configured to send updates between two or more servers. A master server is configured to send

updates, and a subscriber server is configured to receive them, and a server can be both a master and a subscriber for bidirectional replication. Time-based conflict detection and resolution is used to establish precedence in the rare event of the same data being updated in multiple locations at the same time. When replication is configured, a replication agent process is started for each database. If multiple databases on the same server are configured for replication, there is a separate replication agent for each database.

Each replication agent can send updates to one or more subscribers, and receive updates from one or more masters. Each of these connections is implemented as a separate thread of execution inside the replication agent process. Replication agents communicate through TCP/IP stream sockets. For maximum performance, the replication agent detects updates to a database by monitoring the existing transaction log, and sends updates to the subscribers in batches if possible. Only committed transactions are replicated. On the subscriber node, the replication agent updates the database though an efficient low-level interface, avoiding the overhead of the SQL layer.

Caching Technology: When TimesTen Application-Tier Database Cache (TimesTen Cache) is used to cache portions of an Oracle Database, one or more cache groups are created to hold the cached data, and a system agent (the cache agent) performs all asynchronous data transfers between the cache and Oracle Database.

A cache group is a collection of one or more tables arranged in a logical hierarchy via primary/foreign key relationships. Each table in a cache group is related to an Oracle Database table. A cache group table can contain all or a subset of the rows and columns in the related Oracle Database table. Cache groups can be created and modified via SQL and PL/SQL statements. Cache groups support the following features:

- Applications can both read from and write to tables in the cache groups.
- Cache groups can be refreshed from an Oracle database automatically or manually.
- Updates to cache groups can be propagated to an Oracle database automatically or manually.
- Changes to either Oracle Database tables or cache groups can be tracked automatically.

When rows in a cache group are updated by applications, the corresponding rows in Oracle Database tables are updated synchronously as part of the same transaction, or asynchronously immediately afterward depending on the type of cache group that was created. Changes that originate in Oracle Database are refreshed into the cache via the cache agent.

Query Optimization: Query optimization algorithms are different for disk-based systems than for memory-based systems. RDBMS optimization decisions are based on the assumption that data resides primarily on disk. In a dynamic runtime environment, data might be on disk or cached in main-memory at any given moment. Because disk input/output (I/O) is far more expensive than memory access, disk-based RDBMSs have to reduce the probability of access to disk as much as possible. They are optimized to reduce the point of performance bottleneck, so a disk-based optimizer will not always produce the optimal plan for data that resides—primarily or fully—in main memory. IMDB technology, on the other hand, knows that the data resides in main memory and optimizes its queries under simpler assumptions. It does not need to make a worst-case scenario based on disk residency, and therefore, its cost estimates can be simpler and more consistently accurate.

Indexing: Indexes are as important to the performance of memory-based systems as they are for disk-based systems. Indexes are used for a variety of purposes during query processing, such as quickly qualifying rows in a SQL SELECT, UPDATE or DELETE statement, finding the minimum or maximum value of a given column or set of columns, or speeding up join processing. Indexes can also provide an ordered stream of records during query processing. Such ordered streams can eliminate the need for additional sort operations, or allow for fast duplicate elimination.

For all of these similarities with indexes in disk-based RDBMs, indexes can have significant differences with IMDB technology. Where each index entry in a disk-based RDBMS index usually contains an index key and a record identifier, which enables the RDBMS to locate the record on disk, with IMDB technology, keys do not need to be stored in the indexes, and record identifiers can be implemented as record pointers. A record pointer in an index entry points to the corresponding record that contains the key. By avoiding the duplication of key values in the index structure, IMDB technology greatly reduces the size of each index. The absence of key values can also lead to a simpler index implementation because it avoids the need to manage variable-size keys within the index nodes.

Query Processing: IMDB technology takes advantage of memory residency of all data. For example, the fastest way to execute a query with an ORDER BY clause in a disk-based RDBMS is if the base table happens to be stored in sorted order using the same column as in the ORDER BY clause. That is of course very unlikely. The next best case is if the order can be satisfied using an index scan. However, retrieving the data via an index scan results in random access to the records, and potentially results in very high I/O cost. It may even require that the same data block be read from disk multiple times. Since the base table can only be stored in sorted order via one set of columns, the order in which the records are stored can be only used to satisfy ORDER BY queries where the ORDER BY columns are a prefix of the sorting columns. This is not an issue with IMDB technology because index entries point directly to the memory address where the data resides, so random scan via an index is as cheap as a sequential scan.

Similarly, when implementing a sort function in a disk-based system, one has to decide whether the sort structure will contain entire or partial records, key/record identifier pairs, or only record identifiers. This is a space/performance trade-off. If the attributes of interest are not stored in the sort structure, random access with potential I/O overhead will be needed to retrieve the data. By contrast, memory-based systems do not have this issue because data can be accessed directly via tuple pointers so the sort structure needs to contain only the tuple pointers.

Buffer Pool Management: In conventional RDBMS architectures, buffer pools must be maintained for data that has been cached in main memory. When the query processor requires a page of data, it must first search the buffer pool

for that data, and even if the data is there, in many cases it must be copied out of the pool for processing. This buffer pool maintenance and management, coupled with additional data copies, add significantly to the original burden of making the data available to an application.

Although necessary for disk-based databases, buffer pools are unnecessary for in-memory databases. The data already resides within main memory, so TimesTen has no buffer pool. Therefore, code path length and engine footprint are reduced, copying is avoided, algorithms are simplified, and data is delivered to the application more quickly.

Non-uniform Memory Access: Non-uniform Memory Access (NUMA) is an architecture of the main memory subsystem where the latency of a memory operation depends on the relative location of the processor that is performing memory operations. Broadly, each processor in a NUMA system has a local memory that can be accessed with minimal latency, but can also access at least one remote memory with longer latency. The main reason for employing NUMA architecture is to improve the main memory bandwidth and total memory size that can be deployed in a server node. NUMA allows the clustering of several memory controllers into a single server node, creating several memory domains. In the context of data management systems, current research directions on NUMA-awareness can be broadly classified into three categories:

- Partitioning the data such that memory accesses to remote NUMA domains are minimized [13];
- Managing NUMA effects on latency-sensitive workloads such as OLTP transactions
- Efficient data shuffling across NUMA domains

Transactional Memory: Transactional memory is a concurrency control mechanism for shared memory access, which is analogous to atomic database transactions. The two types of transactional memory, i.e., software transactional memory (STM) and hardware transactional memory (HTM), STM causes a significant slowdown during execution and thus has limited practical application [14], while HTM has attracted new attention for its efficient hardware-assisted atomic operations/transactions, since Intel introduced it in its mainstream Haswell microarchitecture CPU This HTM design incurs almost no overhead for transaction execution, but has the following drawbacks, which make HTM only suitable for small and short transactions.

The transaction size is limited to the size of L1 data cache, which is usually 32 KB. Thus it is not possible to simply execute a database transaction as one monolithic HTM transaction. Cache associativity makes it more prone to false conflicts, because some cache lines are likely to go to the same cache set, and an eviction of a cache line leads to abort of the transaction, which cannot be resolved by restarting the transaction due to the determinism of the cache mapping strategy.

NVRAM: Newly-emerging Non Volatile Memory (NVM) raises the prospect of persistent high-speed memory with large capacity. Examples of NVM include both NAND/NOR flash memory with block - granularity addressability, and non-volatile random access memory (NVRAM) with byte-granularity addressability. Flash memory/SSD has been widely used in practice, and attracted a significant amount of attention in both academia and industry, but its block-granularity interface, and expensive "erase" operation make it only suitable to act as the lower-level storage, such as replacement of hard disk or disk cache. Advanced NVRAM technologies, such as Phase Change Memory (PCM), Spin-Transfer Torque Magnetic RAM can provide orders of magnitude better performance than either conventional hard disk or flash memory, and deliver excellent performance on the same order of magnitude as DRAM, but with persistent writes. It has been shown that simply replacing disk with NVRAM is not optimal, due to the high overhead from the cumbersome file system interface (e.g., file system cache and costly system calls), block-granularity access and high economic cost, etc. Instead, NVRAM has been proposed to be placed side-by-side with DRAM on the memory bus, available to ordinary CPU loads and stores, such that the physical address space can be divided between volatile and non-volatile memory, or be constituted completely by non-volatile memory equipped with fine-tuned OS support.

III. Analysis of Problem

Despite the promising design attributes for high performance, IMDS face some challenges in compare to traditional on-disk databases. While the robust traditional database systems, as a matter of design based on hard disk, guarantee atomic, consistent, isolated and durable (ACID) nature of the transactions.

A. Challenges with IMDB

IMDSs have various challenges with assuring the durable characteristic due to its RAM-based design. Different products are offering diverse ways to cope with this issue. To achieve durability, in-memory database systems are applying the following techniques:

1) Persistent Main Memory

Latest technological developments in main memory arena like NVRAM (Non- Volatile RAM) or battery powered main memory enable the scenario where data in main memory will not be lost.

2) Transaction logging

With transaction logging, every transaction will be logged to a persistent store, which will enable the roll-forward mechanism to restore the database in case of power failure. To avoid making transaction logging the bottleneck in overall performance, different design strategies are to write the logs first in stable memory (faster than disk write) and then to disk in asynchronous processes and hence the main database operations will not wait for the logging to be completed.

3) High availability implementation

High availability implementation is another design approach to address this problem. High availability means replicating the data to other available nodes in real time. That will enable to create the copies of data in almost real time and hence will reduce the probability of complete data failure. In case, if any one node goes down, other node will take over the request and will process it.

IMDSs also have challenges as a result of some of their design attributes for high performance. Some these challenges are described as below -

4) Parallelism

In general, there are three levels of parallelism, i.e., data-level parallelism (e.g., bit-parallel, SIMD), shared-memory scale-up parallelism (e.g., thread/process) and shared-nothing scale-out parallelism (e.g., distributed computation). Ideally, we would like to achieve linear scalability as the computation resources increase. This, however, is non-trivial and requires considerable tuning and well-designed algorithms. The fact that all these three levels of parallelism have been deployed in a wide variety of combinations further compounds the problem.

5) Concurrency Control

An efficient concurrency control protocol is necessary and important in order to ensure the atomicity and isolation properties, and not to offset the benefit derived from parallelism. With the increasing number of machines that can be deployed in a cluster and the increasing number of CPU cores in each machine, it is not uncommon that more threads/processes will run in parallel, which dramatically increase the complexity for concurrency control. Surprisingly, current concurrency control algorithms fail to scale beyond 1024 cores [15].

6) Communication

Network communication is incurred for a variety of critical operations: data replication for fault tolerance, information exchange for coordination, data transmission for data sharing or load balancing, and so on. The limited size of main memory of a single server, in contrast to the big volume of data, exacerbates the network communication requirement. However, the data access latency gap between main memory and network is huge, making communication efficiency important to the overall performance.

7) Storage

Even though in-memory databases store all the data in the main memory, the data should also be persisted to non-volatile storage for durability and fault tolerance. In traditional disk-based databases, this is achieved by logging each data update to a disk-resident write-ahead log. Logging to disk, however, is prohibitively expensive in the context of in-memory databases due to the extremely slow disk access, in contrast to the fast memory access. Can we scale in-memory databases to exploit expanding NVM capacities effectively?

B. IMD Solutions Available In Market

In recent years, variety of IMDS solutions (both commercial and open source) made available in market from giant players of database market such as IBM, Oracle, SAP, VMWare etc. Although all of them share the capability to maintain the database in main memory and supports industry standards such as SQL for data processing, they offer different set of features. This section investigates some famous commercial and open-source in-memory databases

Commercial IMDS:

1) TimesTen

TimesTen is in-memory relational database system from Oracle with features like durability, query optimization, recoverability etc. It offers instant responsiveness and very high throughput required by today's real time applications such as telecom, capital markets and defence. It also provides multiple interfaces such as JDBC, ODBC and other SQL APIs.

2) SolidDB

SolidDB is a hybrid disk/ in-memory relational database system from IBM. It offers extreme speed, availability and adaptability required for mission-critical applications. There are a number of deployments of solidDB in telecommunication networks, enterprise applications and embedded software and systems.

3) ExtremeDB

ExtremeDB from McObject is an extremely fast in-memory database system. It is designed explicitly for real-time applications and for embedded systems such as set-up boxes, telecom/ networking devices, industrial control system etc. It states to offers unmatched performance, reliability and development flexibility.

4) SQL Fire

SQLFire is an in-memory distributed SQL database from VMware vFabric. It states to offer high throughput, dynamic & linear scalability, and continuous availability of data. It is designed by utilizing the research and development done on in-memory cache 'GemFire' by same vendor and hence is quoted to be very mature and feature rich.

5) HANA

HANA [16] is distributed in-memory relational database system from SAP. It supports features like column based storage and queries, data compression and parallel processing which makes possible forecasting, planning, analysis and simulation in real or near to real time. It states to offer support for complex queries and high performance for complex queries.

C. Open-Source IMDS

1) SQ Lite

SQLite is an open-source relational in memory database engine. It is small, fast and reliable DBMS suited for embedded systems such as cell phones, PDAs, set-up boxes etc. It is also used for local/ client storage in web browsers. However, it has limited support for complex SQL queries, triggers and views.

2) CSQL

CSQL is another open-source main memory relational database system developed at sourceforge.net. It is designed to provide high performance on simple SQL queries and DML statements that involve only one table. It supports to work in embedded as well as client/server mode. Apart from acting as relational storage engine, it can also be used as a cache for existing disk-based commercial databases.

3) Monet DB

MonetDB is an open-source column oriented main-memory database management system developed at the National Research Institute for Mathematics and Computer Science in the Netherlands. It was designed to provide high performance on complex queries in large databases, such as combining tables with hundreds of columns and multi-million rows. MonetDB is one of the first database systems to focus its query optimization. It has been successfully applied in high-performance applications for data mining, OLAP, XML Query, GIS etc.

IV. PROPOSED WORK

The challenges faced by in-memory databases can be addressed from both software and hardware perspectives. Software solutions are constrained by the underlying

software stack, and increasingly hitting the performance bottleneck, while hardware solutions can boost the performance from the lowest underlying layers (i.e., transistors, circuits), which usually does not introduce much overhead.

A. Software and Hardware Solutions

The potential solutions to the challenges described in above section, from both software and hardware perspectives are as following:

1) Parallelism

On the software level, we can impose different computation models or techniques to realize parallelism of different granularities, namely, fine-grained, coarse-grained and embarrassing parallelism. Multithreaded programs and distributed computing helps scale up and scale out the computing/storage capability. NUMA architecture is proposed to solve the data starvation problem in modern CPUs, by eliminating the coordination among different processors when accessing its local memory, and thus fully exerting the accumulated power from multiple processors. SIMD provides an easier alternative to achieve data-level parallelism, for its capability to operate on multiple data objects in one instruction.

2) Concurrency Control

Concurrency control protocol can either be lock-based or timestamp-based, from the software perspective. Very Lightweight Locking (VLL) is a lock-based approach, and some MVCC protocols are timestamp-based approaches. The ideal case for concurrency control is that all transactions are executed in parallel without any concurrency control protocol overhead, which is usually very hard to achieve in practice. It has been shown that HTM-based timestamp-based concurrency control performs quite close to the ideal and hence simply cannot be ignored.

3) Communication

To improve the network performance, data locality is key to minimizing communication overhead. With good data locality, there will be less frequent access to remote data. This can be achieved by a good partitioning algorithm, or an efficient query routing mechanism to push computation close to the data. The data locality can only be considered from the software perspective, since the strategy is usually application-specific.

4) Storage

To alleviate durability overhead for in-memory databases, recent studies [17] proposed to use command logging, which logs only operations instead of the updated data, and combines with group commit to further reduce the number of loggings. However, there still exists a fundamental design trade-off between the high durability and logging overhead. If only a small number of transactions are committed each time, then the logging overhead may still substantially affect the transaction throughput due to its orders of magnitude higher latency than the execution time of transactions. On the other hand, accumulating a large number of transactions for group commit can lead to more data lost upon failure. Potential Problems with Hardware Solutions. The adoption of hardware solutions does not guarantee superior performance over software solutions. For example, simply using RDMA does not necessarily improve the performance greatly even though RDMA throughput is 40 times faster than Ethernet. Some issues that arise from hardware solutions are as below:

1) Mismatch with traditional software/OS stack. Newly-emerged hardware sometimes does not match with the traditional software/OS stack, which will cause unexpected behaviors or performance degradation.
2) Scalability. The scale of some new hardware cannot catch up with advances of other parts of the system, and some new hardware cannot easily scale up/out without significant performance degradation. Another scalability issue is that the current hardware support for virtual memory does not scale to terabytes (and beyond) of physical memory. The use of small page sizes and fine protection boundaries will require significant space overhead in the corresponding page tables.
3) Generality/Compatibility. Hardware solutions are usually architecture specific, and not general enough to satisfy the different requirements from a variety of applications. For example, RDMA cannot easily communicate with the traditional Ethernet network directly. In addition, not all database transaction semantics can be expressed using HTM, since there are some other factors restricting its usage, e.g., limited HTM transaction size that is restricted by L1 cache, unexpected transaction abort due to cache pollution problems. Moreover, data alignment requirement for SIMD makes SIMD-based implementation architecture specific.
4) Extra overhead. In order to utilize some hardware, there are extra preparation work to do, which may offset the performance gain from the hardware.
5) Bottleneck shift. Even though the use of new hardware may tackle one bottleneck, the contribution to the system's overall performance may be restricted. E.g fast networking moves the bottleneck from network I/O to CPU.

B. Proposed Integrated Software and Hardware Solutions

We believe that hardware solutions, when combined with software solutions, would be able to fully exploit the potentials of in-memory databases. Atomic primitives can be used for single object synchronization, and virtual snapshot by forking facilitates a hardware-assisted isolation among processes. HTM combined with other concurrency control mechanisms (e.g., timestamp-based) can be an alternative to the lock-based mechanism, but its special features (e.g., limited transaction size, unexpected aborts under certain conditions) should be taken into consideration. A mix of these protection mechanisms should enable a more efficient concurrency control model. Since the bottleneck for in-memory databases shifts from disk to memory, a good concurrency control protocol also needs to consider the underlying memory hierarchy, such as NUMA architecture and caches, whose performance highly depends on the data locality.

Nevertheless, robust data structures that are parallelism conscious, memory-economical, and access-efficient form the foundation for further parallelism exploration. A good partition strategy (i.e., to achieve data locality), and efficient communication model (e.g., batch or coalescing transmission), the communication performance can be significantly enhanced.

For NVM-based in-memory databases, we believe a unified space management is required to effectively exploit its features (e.g., byte addressability and durable write). The manipulation of hardware can be achieved either through syscalls or kernelbypass methods. Some new hardware already provides direct kernel-bypass interfaces. But with kernel bypass, the mature functionalities of the OS, such as memory management, concurrency control, and buffer management are no longer available.

Every operational overhead that is considered negligible in disk-based systems, may become the new bottleneck in memory-based systems. Thus the removal of these legacy bottlenecks such as system calls, network stack, and cross-cache-line data layout, would contribute to a significant performance boost for in-memory systems

V. APPLICATION

The proposed work will arrive at integrated software and hardware solution for optimal performance of in-memory database management systems. The plan is also find out optimal configurations for different types of workload e.g. Data Warehousing Workloads which are I/O intensive and predominantly read based with low hit ratios on buffer pools, OLTP Workloads are strongly random I/O intensive, Batch Workloads are more write intensive, Backup operations drive high level of sequential IO and Recovery operation drives high levels of random I/O.

The proposed work will help to arrive optimal solutions for some of following use cases -

- Enterprise OLTP & Periodic Reporting systems (packaged ERP, CRM, HCM such as Siebel, PeopleSoft, JD Edwards, etc.) typically include a mixture of both OLTP transactions and periodic analytic reporting
- Data Warehouse Systems, IoT Applications
- Real-Time Enterprises - Business activity monitoring, complex event processing, RFID/sensor-based applications, Web portals, and Web services
- Real-Time Data Management Software
- Mobile apps that need integrated data, rich interactions and advanced analytics using devices such as tablets, smartphones, and wearables
- Real-time apps such as stock trading, fraud detection, counter terrorism, patient health monitoring, machine analysis, or earthquake monitoring.
- Customer analytics retail stores, retailers, eCommerce sites

VI. CONCLUSION

In this work, we studied In-Memory database for its Architecture, components, characteristics and core technologies involved. Also summarized the challenges of in-memory databases, and their solutions from both the software and hardware perspectives. While hardware solutions are known for its efficiency with less overhead, as shown in this paper, they do not always outperform software solutions. In order to fully exploit the potentials of in-memory systems, we believe that a combined hardware-software solution is needed which the plan for next phase of work. This solution also to be analyzed and experimented for some mostly used enterprise applications to arrive at optimal configurations. This in turn will also help to outline the process for selecting suitable available solution in the market as well as optimal configurations of the solution to achieve enterprise objectives.

ACKNOWLEDGMENT

The author is thankful to Department of Computer Science & Engineering, Dr. Sau. Kamaltai Gawai Institute of Engineering & Technology, Darapur, affiliated to Sant Gadge Baba Amravati University, Amravati, Maharashtra, India, for guidance and support during this study and for proposed future work.

REFERENCES

[1] IDC Digital Universe Study. (2012) The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. [Online]. Available: http://idcdocserv.com/1414.
[2] Hector Garcia-Molina, and Kenneth Salem, "Main Memory Database Systems: An Overview", IEEE Transactions on Knowledge and Data Engineering, Vol. 4, No. 6, pp. 509-516, Dec. 1992.
[3] Ram Babu, Nirmal Lodhi et al., "Performance Management of In Memory Databases", International Journal of Advancements in Research & Technology, Volume 2, Issue 4, pp.245-248, Apr. 2013.
[4] Fremont, "TPC Benchmark B Standard Specification ", Waterside Associates, Transaction Processing Performance Council, CA, August 1990.
[5] Oracle TimesTen In-memory database. [Online]. Available: http://www.oracle.com/ technetwork/database/databasetechnologies/timesten/overview/timesten-imdb-086887.html
[6] B. Fitzpatrick and A. Vorobey, "Memcached: a distributed memory object caching system," 2003. [Online]. Available: http: //memcached.org/
[7] H. Plattner, "A common database approach for oltp and olap using an in-memory column database," in SIGMOD'09, 2009.
[8] Y. Li and J. M. Patel, "Bitweaving: Fast scans for main memory data processing," in SIGMOD '13, 2013.
[9] X. Yu, G. Bezerra, A. Pavlo et al., "Staring into the abyss: An evaluation of concurrency control with one thousand cores," in PVLDB '15, 2014.
[10] H. Pirk, F. Funke, M. Grund et al., "CPU and cache efficient management of memory-resident databases," in ICDE '13, 2013.
[11] C. Diaconu, C. Freedman, E. Ismert et al., "Hekaton: Sql server's memory-optimized OLTP engine," in SIGMOD '13, 2013.
[12] G. Graefe, H. Volos, H. Kimura et al., "In-memory performance for big data," in PVLDB '15, 2014.
[13] V. Leis, P. Boncz, A. Kemper, and T. Neumann, "Morsel-driven parallelism: A numa-aware query evaluation framework for the many-core age," in SIGMOD '14, 2014.
[14] C. Cascaval, C. Blundell, M. Michael et al., "Software transactional memory: Why is it only a research toy?" Queue, 2008.

[15] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang. In-memory big data management and processing: A survey. TKDE, 27(7):1920{1947, July 2015.
[16] SAP HANA. [Online]. Available: http://www.saphana.com.
[17] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker. Rethinking main memory oltp recovery. In ICDE '14, pages 604 - 615, 2014.

