The advent of cloud computing platforms with massive user bases and high transaction-throughput requirements has made it necessary for enterprises to find ways to scale their services in a quick and cost-effective manner. This puts pressure on system architects to cost-effectively design larger and improved systems. In the era of big data, enterprises are increasingly looking inward at huge caches of under-processed or throw-away data as resources to be mined.

Processing voluminous amounts of data requires a fast and scalable platform. In the past, deployments of these types of platforms were limited to a few large enterprises that could afford such costly data mining solutions. Nowadays, enterprises have more options. This article provides an overview of one of the options available—the in-memory database (IMDB)\(^1\)—its evolution and the risk involved in its adoption.

IMDB technology has been touted as the cure for database performance problems—a key factor is its ability to load and execute all data in memory. This removes a substantial amount of input/output (I/O)-related performance problems associated with database systems. However, IMDB technologies introduce fundamental risk that must be considered in their deployment: durability of data, looser security controls (compared to its full database counterparts) and migration concerns. It is critical that the risk be considered when exploring the use of IMDB technology.

WAYS OF SCALING

There are two ways of scaling applications: horizontally and vertically. Horizontal scaling allows the enterprise to create applications that can take advantage by simply adding computing nodes when they need more capacity. In general, applications that require a large amount of atomic working data or perform a large amount of mutually exclusive/heavily pipe-lined transactions are suitable for horizontal parallelization. Not so long ago, this was called parallel or supercomputing.\(^2\) Large web applications in which each web transaction is atomic and does not depend on other concurrent transactions is an example of horizontal scaling. Thus, each transaction can be routed to separate computing nodes for processing. Horizontal scaling allows Facebook, LinkedIn and Twitter to handle millions of users. However, not all applications are easily portable to horizontally scaled platforms. One of the main challenges of horizontal scaling is that applications have normally not been built with horizontal scalability/concurrency in mind. Even typical desktop applications are not built to utilize the multiple central processing unit (CPU) cores available in modern commodity computing platforms. In these and similar cases, enterprises may opt to use vertical scaling.

Vertical scaling involves increasing the internal capacity of a system so it can handle more transactions. This is normally the fastest way to increase capacity without substantially changing the operating environment or the system architecture. Increasing the memory or disk storage of a computing system to handle more transactions is an example of vertical scaling. Vertical scaling is not limited to adding hardware, but can also apply to enhancing the application to get the most out of the existing resources. However, vertical scalability is generally more costly.

IT IS ALL IN RAM

There are also other ways of increasing the scalability of systems vertically. One of these is the use of in-memory computing technology. The art of scaling systems involves identifying bottlenecks when performing transactions. By
determining the key areas of slowdown, system architects can work on optimizing those areas without the need to buy more hardware. Different applications will need different levels of a particular resource and will have different bottlenecks. For data-driven applications, the bottleneck is most likely disk storage or I/O. A key bottleneck exists when the application requires a lot of data interaction and subsequently disk access. A great deal of complex database applications are I/O bound.

On the other hand, memory access is normally measured in nanoseconds while disk storage access is measured in milliseconds. This shows that memory access is orders of magnitude faster than disk storage access. Therefore, a possible solution to I/O-bound applications is the use of in-memory computing. All data are loaded into memory, and all transactions are executed in memory. The most tangible manifestation of in-memory computing is the IMDB. IMDBs provide substantial performance gains by storing all data in the main memory instead of disks. This provides the benefit of being able to execute I/O transactions entirely in memory. A person who memorizes the dictionary can respond faster to a word definition query than a person who did not memorize the entire dictionary and has to look up the word in a printed book.

WHO CAN BENEFIT FROM IN-MEMORY COMPUTING?
The first step in determining the need for in-memory computing is to determine if the application requires a lot of data access and manipulation. Normally, database applications can benefit from IMDB technology. Generally, any type of database transaction will be slower on a disk-based database as opposed to an IMDB. Enterprises are attracted to IMDBs because they allow easy porting of applications from disk-based database systems. Not all specifications and the aspects relating to them will be considered at the outset and used for preplanning the need for, and deployment of, IMDB technology. Sometimes, bottlenecks can be determined during the course of development, user-acceptance testing or even during actual production.

Two common ways to determine I/O bottlenecks are:

1. **I/O issues that manifest as high CPU utilization**—For example, if disk I/O is busy in a system, the I/O wait process can take a substantial amount of CPU time. In some cases, the database process shows high CPU utilization; hence, some people think that it is the CPU (processing power) that needs upgrading. In reality, it is the storage subsystem that is the bottleneck and needs to be addressed.

2. **Operating systems with I/O monitoring tools**—Linux- and UNIX-derived systems come with the highly functional iostat tool. MS Windows-based systems come with perfmon. Administrators must be on the lookout for parameters such as average queue length, average transfer time and percentage disk time. If these values are elevated, there is possible I/O contention.

The best way to determine if an application can benefit from IMBD technology is to try the solutions. There are a number of commercial (Oracle TimesTen, SAP HANA, IBM solidDB, VMWare Gemfire and open-source (MySQL cluster, SQLite, VoltDB, Druid) solutions available in the market.

**IS RAM VOLATILE? ARE MY DATA SAFE?**

There are a number of factors that must be considered with any new technology introduced into the market, the first of which is durability. It is the first thing that generally comes to mind when using and selecting in-memory computing technology. Main memory is volatile, so when the power is cut, systems will lose data in memory. Such data loss is particularly damaging for data-driven applications.

Nevertheless, the majority of in-memory solutions do have a mechanism for ensuring that data are preserved. The most common mechanism is to write back to persistent storage. However, this requires depending on (slow) disks. However, the majority of solutions on the market use something called “lazy” or “fuzzy” write-through. This means that the transaction execution is done entirely on data stored in memory. The transactions are then stored in the form of a log buffer that is also in memory. The system will then write the data into disk for persistence. In the event of an outage, there is a chance of data loss if the log buffer was not able to complete its disk write. However, most of the database will be intact. Some IMDB solutions (e.g., Oracle TimesTen) allow one to vary the “laziness” of the write-through depending on the importance of the transactions. Low-value writes (i.e., transaction logging) defer the writes to disk over a longer period and reduce the I/O load compared to high-value writes (i.e., Airtime top-up), which synchronously write to disk for persistence all the time. This allows users to vary the “laziness” to adapt to the application requirements.
This limitation is the reason why IMDB high-availability deployments normally call for the use of replication. Network throughput is still generally faster than disk throughput. It allows multiple instances of the IMDB to synchronize the data contained in the system. The most common setup is to have a single, active database replicated with a standby or read-only database. The probability of all these systems going down simultaneously is far less than the probability of a single one failing.

On the other hand, some in-memory database solutions utilize a shared-nothing technology for replication. This means that data in these databases are distributed across a cluster of computing nodes for both load balancing and high availability. Shared-nothing technology has the additional benefit of scaling the load onto multiple computing nodes and is an example of horizontal scalability at work. Thus, shared-nothing in-memory computing technology is both vertically and horizontally scaled.

MOVING DATABASE APPLICATIONS TO IMDB

In general, most database applications can benefit from IMDB technology, largely because most applications only use a simple subset of the Structured Query Language (SQL) language. However, IMDB solutions generally do not have the full set of functionality available to disk-based relational database management systems (RDBMSs). For example, some IMDBs do not support database triggers and would not have the same level of granularity for field constraints. Limitations on field constraints (i.e., unicode characters, numeric formats) are particularly important as applications might be written to depend on enforcing field constraints at the database level. If moving to IMDB loosens the previously expected constraints, this opens up a number of field validation-related issues such as injection-type attacks.

Some IMDB platforms do not provide the same level of user and rights management that is common in disk-based relational databases. In some cases, access to a database instance provides access to all the data contained in that instance. In such cases, administrators are required to create separate instances of the database for separate applications. This requires a different user management paradigm.

Users must also consider the resources required to support IMDBs. The main resource required is memory. In particular, extremely large databases may not fit in commercially available quantities of RAM. Disk space is typically measured in tens of terabytes now. Memory, on the other hand, is measured in tens of gigabytes. Some IMDB solutions (e.g., solidDB) allow spanning between memory and disk; this limits the amount of main memory and performance that will degrade if the disk is hit. Thus, shared-nothing in-memory systems (i.e., VoltDB/HANA) outweigh those that are not shared-nothing.

Finally, it is important to remember that an application will have many different components and subsystems. Optimizing only the database will yield performance gains, but that may not be the only bottleneck present in the system. It is important to take into account outside considerations. Examples of these database-related bottlenecks outside the IMDB include connection pooling and interface conversions. In some cases, the number of database connections in the connection pool is limited, causing a transaction bottleneck. Another common problem is when an interface to the database, such as a blocking synchronous transaction or processing heavy data transformation (i.e., computations and conversions), creates a scenario where interface limitations throttle transactions and limit potential top performance. Finally, some transactions do not make it on time to the database because of application-level queuing issues (i.e., some real-time and voluminous non-real-time transactions in the same queue can starve real-time transactions). These are examples of performance issues that involve moving data into the database as opposed to performance of the database itself. It is important not to overoptimize in one area.

Choosing an IMDB solution

The following are key factors to consider when choosing an IMDB solution:

- **ACID compliance/data durability**—Atomicity, consistency, isolation and durability (ACID) are compliance properties that assume that database transactions perform reliably. In particular, durability tends to vary in IMDB
implementations. Most IMDB solutions (e.g., SAP HANA, Oracle TimesTen, VMware Gemfire, MySQL Cluster, VoltDB, SQLite) are ACID-compliant. However, they typically vary when it comes to durability on disk. The “laziness” of the write-through will determine this. Some solutions (e.g., Oracle TimesTen) allow developers to adjust the “laziness” of this write-through, and others (e.g., SQLite) do not support disk write-through.

- **Data volume and scale-out requirements**—How scalable does the application need to be? A number of IMDB solutions support shared-nothing architectures that allow developers to easily create applications that horizontally scale by adding computing/storage nodes. Shared-nothing architectures (i.e., VMware Gemfire, SAP HANA, VoltDB) allow arbitrary scaling by simply adding notes. Their most important feature is that they provide resiliency by not having a single point of data failure (i.e., N+1, mirrored configuration). Some architectures (e.g., Oracle TimesTen) only support aggregate scaling where scaling is also done by adding nodes with a well-partitioned subset of data in them. Therefore, architectures that support shared-nothing can be designed to support horizontally scalable general data warehousing requirements—architectures that do not require developers to design applications for aggregate scaling.

- **SQL compatibility/SQL dialect**—Not all IMDBs are the same when it comes to SQL support. Some provide a basic set of SQL primitives (i.e., Create, Select, Insert, Delete, Update), and others provide a broader set (e.g., foreign key constraints, stored procedures). Simpler packages such as SQLite tend to have simpler SQL primitives’ support but also are easier to implement. Packages with support for complex SQL primitives allow easier migration for applications that already use these primitives. This is a key reason why IMDB technology is attractive. The ease of porting depends on how broad a set of SQL primitives the application requires. This is the major reason that enterprises with existing RDBMS-backed applications prefer IMDB over NoSQL.15

- **Compression**—Using main memory to process transactions does put a constraint on the absolute size of data that can be processed at any given time and node. This can be circumvented by using compression at the expense of CPU processing time. Some databases (e.g., Oracle TimesTen) support this. However, the whole reason for using IMDBs is to remove a performance bottleneck (I/O). It would be counterproductive to replace it with another (CPU). Therefore, careful planning is necessary.

- **Cost**—There are a number of IMDB solutions that are open-source and commercial. The choice will primarily depend on the previously listed requirements. If the remaining candidates provide an open-source and commercial option, factors such as support and maintenance requirements come into play. Commercial and open-source with paid commercial solutions are recommended options. Open-source solutions are viable when no commercial support is necessary and the open-source package has a robust developer community.

**CONCLUSION**

IMDB technology is not new. It has been around for specialized high-throughput (e.g., telecommunications) use cases or caching requirements (e.g., network and authentication proxies) for quite some time. Today, the big data trend is compelling enterprises to mine their large internal hoard of data. The additional insight provided by mining this information can be invaluable for creating an enhanced user experience. The use cases, which require fast processing turnaround times, can benefit from in-memory technology. Fortunately, the industry has also adopted offerings that make it easier to consider in-memory technology, such as the introduction of SQL interfaces, shared-nothing replication and fuzzy write-through for durability.

In terms of cost, IMDB technology requires a substantial amount of memory since all data must fit into memory. Memory speeds are 100,000 to one million times faster than mechanical hard disks in terms of access times. The cost of memory is roughly 100 times that of mechanical hard disks. There certainty (1,000 to 10,000 times) is a substantial

------------

**ALTERNATIVE IN-MEMORY DATABASE ARCHITECTURE**

An alternative to using a dedicated in-memory computing system such as an IMDB would be to use a regular RDBMS on a computing platform that makes exclusive use of memory-based storage devices such as solid-state drives (SSD). Of course, modern computing architecture still treats the SSD disks as I/O devices even if they are made out of internal memory. Therefore, there is still some benefit to a pure RAM implementation. However, as technology gets better, there could be solutions where flash-storage access times become comparable to RAM access times.

---
performance gain when switching to memory-based solutions. The potential challenge is getting enough memory modules into a machine since most computing hardware accepts only a limited amount of RAM (e.g., dmidecode -t 16). Another option is to utilize solid state disk (SSD) technology with regular RDBMs technology (refer to the Alternative In-memory Database Architecture sidebar).

IMDBs provide an easy path toward reaping the benefits of in-memory computing. The use of the SQL interface has provided a quick option for most enterprises to migrate their existing applications. Write-through and replication can address concerns with respect to load balancing and high availability. The obvious “memory is faster than disk” thinking allows for justification of such an initiative. However, care must be taken to ensure that applications truly benefit from the use of in-memory technology. System designers must ask themselves a few basic questions to determine solution fit (refer to the Questions to Ask When Considering an IMDB sidebar). Once the decision to use in-memory computing is made, additional work must be done to ensure that considerations have been deliberated. In particular, the areas of resources required, functionality and security requirements (confidentiality, integrity and availability) must be reviewed. Most important, enterprises must make an effort to try the technology first.

As more and more people interact on the web, service and application providers have more data and tools in their possession—one of which is IMDB technology—to know their customers better. The proliferation of various solutions—commercial and free—puts traditional high-performance data applications in everybody’s hands.

ENDNOTES

2. Kumar, V; A. Grama; A. Gupta; G. Karypis; Introduction to Parallel Computing, vol. 110, Benjamin/Cummings, 1994
5. Godard, Sebastien; “iostat,” Man Page, http://linux.die.net/man/1/iostat/
12. SQLite, SQLite In-Memory Database, www.sqlite.org/inmemorydb.html

QUESTIONS TO ASK WHEN CONSIDERING AN IMDB

- Will the application benefit from in-memory computing technology? Is it I/O-bound?
- Can the data fit in commercially available amounts of RAM?
- Does the application require an SQL interface? Does the IMDB provide it?
- Does the choice of IMDB support the subset of SQL that the application requires?
- Are there security assumptions that change because of the limits of functionality?
- Are persistence and durability necessary? Does the IMDB support disk-based persistence?
- Is there a chance of data loss when solely depending on disk-based persistence? Is that OK?
- Is load-balancing necessary? Does the IMDB support shared-nothing replication? Can it support this for both load balancing and high availability?