The Next 50 Years of Databases

Andy Pavlo
Computer Science Department

The Past

The first database management system (DBMS) came on-line in 1968. IBM's IMS was built to keep track of the supplies and parts inventory for the Saturn V and Apollo space exploration projects. It introduced the idea that an application's code should be separate from the data that it operates on. This allows developers to write applications that only focus on the access and manipulation of data, and not the complications and overhead associated with performing these operations and ensuring that data is safe. IMS was later followed by the pioneering work in the early 1970s on the first relational DBMSs, IBM's System R and the University of California's INGRES.

The database workloads for these first systems were less complex and diverse than they are today. In these earlier applications, a human operator started a transaction through a terminal and then entered new data into the system manually. Back then, the expected peak throughput of a DBMSs was only tens to hundreds transactions per second and the response times were measured in seconds. The architecture of these early DBMSs was also based on the computing hardware that was prevalent at the time they were invented. They were typically deployed on computers that had a single CPU core and a small amount of main memory. For these systems, disks were the primary storage location of databases because they were able to store more data than could fit in memory and were less expensive.

The Present

Although a lot has changed 50 years later in terms of how we use databases, the relational model and SQL are still predominant way to organize a database and interact with it. Many Internet applications need to support hundreds of thousands or even millions of transactions per second, and with each transaction's processing latency is in the order of milliseconds. This is because they are connected to millions of users and other computer systems at the same time. And now that businesses and organization are able collect a large amount of data from these applications, they want to analyze it to extrapolate new information to guide their decision making. For this reason, in recent years we have seen the rise of specialized systems that target specific application scenarios that are able to perform much better than the general purpose DBMSs based on the 1970s architectures. There are now DBMSs that are designed to ingest new information quickly for on-line transaction processing (OLTP) applications and other DBMSs that are designed to store large amount of data for complex on-line analytical processing (OLAP) programs.

These newer DBMSs also take advantage of the two major hardware trends that have emerged in recent years. The first is the advent of large-memory computers, which makes it now affordable to deploy a small number of machines that have enough DRAM to store all but the largest OLTP databases. Storing data in memory ensures that the DBMS can process many transactions simultaneously with low latencies. In our experience, the databases' for modern OLTP applications are typically several hundred gigabytes in size. Contrast this with an OLAP data warehouse, where the DBMS could be managing databases that are several petabytes in size. This difference is because an OLTP database stores the current state of an application (e.g., the orders from the last 90 days), whereas an OLAP database stores all of the historical information for an organization (e.g., all orders ever placed). Thus, OLAP DBMSs are still primarily stored on disks and employ several optimizations like compression or columnar storage to overcome their slower access times.

The second hardware trend is the shift from increasing single-core CPU clock speeds to multi-core CPUs. Clock frequencies have increased for decades, but now the growth has stopped because of hard power constraints and complexity issues. Aggressive, out-of-order, super-scalar processors are being replaced with simple, in-order, single issue cores. Exploiting this increased parallelism is difficult in a DBMS because of the complexity of coordinating competing accesses to shared data for hundreds of threads. Modern DBMSs are employing low-overhead concurrency control and other lock-free techniques to improve the scalability of the system.

Despite these advancements, there are still significant barriers that impede many from deploying data intensive applications. One overarching theme in all of this is that databases are still a human-intensive component of computing systems (e.g., deployment, configuration, administration). Using two independent DBMSs separates the OLTP and OLAP workloads to avoid one from slowing down the other, but it requires additional processes to transfer data from

system to the other. In addition to this, tuning a DBMS to get the best performance for a particular application is notoriously difficult. Many organizations resort to hiring experts to configure the system for the expected workload. But as databases grow in both size and complexity, optimizing a DBMS to meet the needs of these applications has surpassed the abilities of humans.

The Future

Over the next 50 years, like with the previous, we will see significant changes to the database landscape. Beyond obvious things like the volume and velocity of the data being stored being much greater, there will be major changes in how databases are used in applications and the type of hardware that they will deployed on. It is difficult to predict what the major paradigm shift will be in the field. Hence, we opine on several themes.

The relational model will still dominant for most applications, but developers will no longer need to explicitly worry about data model their application uses. There will be a tighter coupling of programming frameworks and DBMSs such that all database interactions will be transparent (and optimal). Likewise, SQL (or some dialect of it) will remain the de facto language for interacting with a DBMS, but humans will never actually write SQL. They will instead ask questions about data in a natural language. Such changes will cause a major shift in how we write programs; the developer models their data in a way that is best understood by humans and then the framework (in conjunction with the DBMS) will automatically generate the optimal storage scheme for it. All programs will execute using strongly consistent ACID transactions. That is, the eventual consistency methods that are used in today's Web-based applications will be avoided do to the management complexity. There will be major advancements in network communication, concurrency control, and resource management that makes using ACID transactions preferable and scalable.

There will be an increasing number of applications where it is more natural to store data in arrays or matrices. This is because organizations will want to analyze large corpus of unstructured information, especially video. We will have mastered the ability convert all unstructured data into semi-structured formats that are more easily organized and indexed in a DBMS. As part of this, temporality will become important as well because it matters how information changes over time. Current systems are not able to account for this because of the large overhead of storing extracted information about each video frame in a time series.

The omnipresent "Internet of Things" will mean that every device is able to collect data about its environment. This will range from small, embedded sensors to larger, autonomous robots. Smaller devices will use an on-chip DBMS in the same way that cellphones now contain on-chip video decoders. The databases for all of these systems will be completely composable and easily federated through some standard API (possibly SQL). This means that DBMSs will communicate with each other with zero configuration. You just point two DBMSs at each other and they will immediately transfer their information and ensure that they are synchronized. Some manager service will be able to distribute query execution across devices as needed. Humans will not need to manually configure extract-transform-load utilities or other tools to keep the data on disparate systems consistent. It will be a significant engineering job to make all of the various DBMSs composable and inter-operable in this manner. As such, there will be a toolkit that uses artificial intelligence and/or machine learning to automatically map the different variations of the DBMSs to each other for the same operation.

For new hardware, more flexible and programmable processing fabrics will be more prevalent. DBMSs will compile the critical sections of their program (e.g., the lock manager) into a hardware accelerator. We will also see the disappearance of the dichotomy between volatile and non-volatile memory. The DBMSs will assume that all memory is fast and durable; the need to maintain caches in volatile will be unnecessary. This new memory will be orders of magnitude larger than what is available today. Thus, the DBMS will store multiple copies of its data in pre-computed materialized views in order to quickly respond to any possible query.

The role of humans as database administrators will cease to exist. These future systems will be too complex for a human to reason about. DBMSs will finally be completely autonomous and self-healing. Again, the tighter coupling between programming frameworks and DBMSs will allow the system to make better decisions on how to organize data, provision resources, and optimize execution than human-generated planning.

Lastly, we will see the rise of database transactions for inter-planetary devices (e.g., space probes). In this scenario the DBMSs running on these vessels will be at greater distances from each other than Earth-bound systems and incur significantly longer latencies (i.e., minutes or hours). This means that the weak consistency techniques and practices that are used in today's Web-based applications will then be applied to these interstellar systems.