NoSQL and Big Data Systems

NoSQL and Big Data Systems: Motivation

- **Book Chapters**
  - 10.1, 10.2 *(7TH EDITION)*
- **Key topics:**
  - Big data motivating scenarios
  - Why systems so far (relational databases, data warehouses, parallel databases) don’t *always* work
RDBMS Evolution

- Original database systems aimed to support both use cases
- Slowly, specialized systems were built, starting late 80’s-early 90’s, especially for decision support (Data Warehouses)
- Today, different RDBMSs systems for different use cases, e.g.,:
  - VoltDB for OLTP – fully in-memory, very fast transactions, but no complex queries
  - Teradata, Aster Data, Snowflake, AWS Redshift – handle PBs of data, but batch updates only – many indexes and summary structures (cubes) for queries – typically “parallel” (i.e., use many machines)
- Fundamental and wide differences in the technology
- But both still support SQL as the primary interface (with visualizations, exploration, and other tools on top)

NoSQL + Big Data Systems: Motivation

- Very large volumes of data being collected
  - Driven by growth of web, social media, and more recently internet-of-things
  - Web logs were an early source of data
    - Analytics on web logs has great value for advertisements, web site structuring, what posts to show to a user, etc
- Big Data: differentiated from data handled by earlier generation databases
  - **Volume**: much larger amounts of data stored
  - **Velocity**: much higher rates of insertions
  - **Variety**: many types of data, beyond relational data
  - **Veracity**: different levels of truth
Some motivating scenarios

- Deciding what to show a user in a social network, or news aggregator
  - Advertising on the Web or Mobile

- Analyzing user behavior on web sites to optimize or increase engagement

- Analyzing large numbers of images and building search indexes on them

- Text analytics for topic modelling, summarization, ...

- Internet of things…

- And many many others…
Two Primary Use Cases

- **OLTP-like**
  - Simple queries, but lots of updates
  - Need to support distributed users
  - Need to support non-relational data (e.g., graphs, JSONs)
  - Need to scale fast (10 users to 10s of Millions of Users)
  - Need to work well in 3-tier Web Apps
  - Need to support fast schema changes

- **OLAP-like**
  - Complex analysis on large volumes of data
  - Often no “real-time” component, and no updates
  - Mostly non-relational data (images, webpages, text, etc)
  - Tasks often procedural in nature (analyse webpages for searching, data cleaning, ML)

Why (Parallel) Databases Don't Work

- The data is often not relational in nature
  - E.g., images, text, graphs

- The analysis/queries are not relational in nature
  - E.g., Image Analysis, Text Analytics, Natural Language Processing, Web Analytics, Social Network Analysis, Machine Learning, etc.
  - Databases don’t really have constructs to support this
    - User-defined functions can help to some extent
  - Need to interleave relational-like operations with non-relational (e.g., data cleaning, etc.)
  - Domain users are more used to procedural languages

- The operations are often one-time
  - Only need to analyse images once in a while to create a “deep learning” model
  - Databases are really better suited for repeated analysis of the data

- Much of the analysis not time-sensitive

- Parallel databases too expensive given the data volumes
  - Designed for large enterprises, with typically big budgets
Parallel and Distributed Architectures

- Ability to scale “up” a computer is limited ➔ Use many computers together
  - Called cluster or network of computers (and today, just a “data center”)

- Also need to "meet" where the users are
  - To minimize interactive latencies (e.g., social networks)

- Has made parallel and distributed architectures very common today

Parallel Architectures

- Shared-nothing vs. shared-memory vs. shared-disk

![ diagrams of different parallel architectures ]
All the cores on a single processor typically access a shared memory. Further, a system can have multiple processors which can share memory. Another effect of the increasing number of gates has been the steady increase in the size of main memory as well as a decrease in cost, per-byte, of main memory.

Given the availability of multicore processors at a low cost, as well as the concurrent availability of very large amounts of memory at a low cost, shared-memory parallel processing has become increasingly important in recent years.

20.4.5.1 Shared-Memory Architectures

In earlier generation architectures, processors were connected to memory via a bus, with all processor cores and memory banks sharing a single bus. A downside of shared-memory accessed via a common bus is that the bus or the interconnection network becomes a bottleneck, since it is shared by all processors. Adding more processors does not help after a point, since the processors will spend most of their time waiting for their turn on the bus to access memory.

As a result, modern shared-memory architectures associate memory directly with processors; each processor has locally connected memory, which can be accessed very quickly; however, each processor can also access memory associated with other processors; a fast interprocessor communication network ensures that data are fetched with relatively low overhead. Since there is a difference in memory access speed depending on which part of memory is accessed, such an architecture is often referred to as non-uniform memory architecture (NUMA).

Figure 20.6 shows a conceptual architecture of a modern shared-memory system with multiple processors; note that each processor has a bank of memory directly connected to it, and the processors are linked by a fast interconnect system; processors are also connected to I/O controllers which interface with external storage.

A storage-area network (SAN) is a high-speed local-area network designed to connect large banks of storage devices (disks) to nodes that use the data (see Figure 20.8). The storage devices physically consist of an array of multiple disks but provide a view of a logical disk, or set of disks, that hides the details of the underlying disks. For example, a logical disk may be much larger than any of the physical disks, and a logical disk’s size can be increased by adding more physical disks. The processing nodes can access disks as if they are local disks, even though they are physically separate.

Storage-area networks are usually built with redundancy, such as multiple paths between nodes, so if a component such as a link or a connection to the network fails, the network continues to function.

Storage-area networks are well suited for building shared-disk systems. The shared-disk architecture with storage-area networks has found acceptance in applications that do not need a very high degree of parallelism but do require high availability.

Compared to shared-memory systems, shared-disk systems can scale to a larger number of processors, but communication across nodes is slower (up to a few milliseconds in the absence of special-purpose hardware for communication), since it has to go through a communication network.

One limitation of shared-disk systems is that the bandwidth of the network connection to storage in a shared-disk system is usually less than the bandwidth available to access local storage. Thus, storage access can become a bottleneck, limiting scalability.

20.4.7 Shared Nothing

In a shared-nothing system, each node consists of a processor, memory, and one or more disks. The nodes communicate by a high-speed interconnection network. A node

<table>
<thead>
<tr>
<th></th>
<th>Shared Memory</th>
<th>Shared Disk</th>
<th>Shared Nothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication between processors</td>
<td>Extremely fast</td>
<td>Disk interconnect is very fast</td>
<td>Over a LAN, so slowest</td>
</tr>
<tr>
<td>Scalability?</td>
<td>Not beyond 32 or 64 or so (memory bus is the bottleneck)</td>
<td>Not very scalable (disk interconnect is the bottleneck)</td>
<td>Very very scalable</td>
</tr>
<tr>
<td>Notes</td>
<td>Cache-coherency an issue</td>
<td>Transactions complicated; natural fault-tolerance.</td>
<td>Distributed transactions are complicated (deadlock detection etc);</td>
</tr>
<tr>
<td>Main use</td>
<td>Low degrees of parallelism</td>
<td>Not used very often</td>
<td>Everywhere</td>
</tr>
</tbody>
</table>