Database Architecture 2 & Storage

Instructor: Matei Zaharia

cs245.stanford.edu
Outline

Relational DBMS architecture

Alternative architectures & tradeoffs

Storage hardware
Summary from Last Time

System R mostly matched the architecture of a modern RDBMS
  » SQL
  » Many storage & access methods
  » Cost-based optimizer
  » Lock manager
  » Recovery
  » View-based access control
Differentiating by Workload

Two big classes of commercial RDBMS today

**Transactional DBMS:** focus on concurrent, small, low-latency transactions (e.g. MySQL, Postgres, Oracle, DB2) → **real-time apps**

**Analytical DBMS:** focus on large, parallel but mostly read-only analytics (e.g. Teradata, Redshift, Vertica) → “data warehouses”
## How To Design Components for Transactional vs Analytical DBMS?

<table>
<thead>
<tr>
<th>Component</th>
<th>Transactional DBMS</th>
<th>Analytical DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data storage</td>
<td>B-trees, row oriented storage</td>
<td>Column-oriented storage</td>
</tr>
<tr>
<td>Locking</td>
<td>Fine-grained, very optimized</td>
<td>Coarse-grained (few writes)</td>
</tr>
<tr>
<td>Recovery</td>
<td>Log data writes, minimize latency</td>
<td>Log queries</td>
</tr>
</tbody>
</table>
Outline

Relational DBMS architecture

Alternative architectures & tradeoffs

Storage hardware
How Can We Change the DBMS Architecture?
Decouple Query Processing from Storage Management

Example: “data lake” architecture (Hadoop, S3, etc)

<table>
<thead>
<tr>
<th>Processing engines</th>
<th>File formats &amp; metadata</th>
<th>Large-scale file systems or blob stores</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapReduce</td>
<td>Parquet</td>
<td>GFS</td>
</tr>
<tr>
<td>Apache Giraph</td>
<td>Apache Storm</td>
<td>HDFS</td>
</tr>
<tr>
<td>Spark</td>
<td>protoBuf</td>
<td>Amazon S3</td>
</tr>
<tr>
<td>presto</td>
<td>JSON</td>
<td></td>
</tr>
</tbody>
</table>
Decouple Query Processing from Storage Management

Pros:
» Can scale compute independently of storage (e.g. in datacenter or public cloud)
» Let different orgs develop different engines
» Your data is “open” by default to new tech

Cons:
» Harder to guarantee isolation, reliability, etc
» Harder to co-optimize compute and storage
» Can’t optimize across many compute engines
» Harder to manage if too many engines!
Change the Data Model

**Key-value stores:** data is just key-value pairs, don’t worry about record internals

**Message queues:** data is only accessed in a specific FIFO order; limited operations

**ML frameworks:** data is tensors, models, etc
Change the Compute Model

**Stream processing:** Apps run continuously and system can manage upgrades, scaleup, recovery, etc

**Eventual consistency:** handle it at app level
Different Hardware Setting

**Distributed databases:** need to distribute your lock manager, storage manager, etc, or find system designs that eliminate them

**Public cloud:** “serverless” databases that can scale compute independently of storage (e.g. AWS Aurora, Google BigQuery)
Example: AWS Aurora Serverless

Diagram showing the integration of Applications with a Proxy fleet, Aurora Database Storage components, and a warm pool of DB capacity.
Outline

Relational DBMS architecture

Alternative architectures & tradeoffs

Storage hardware
Typical Server

- CPU
- DRAM
- I/O Controller
- Network Card
- Storage Devices
Storage Performance Metrics

- latency (s)
- throughput (bytes/s)
- storage capacity (bytes, bytes/$)
"Numbers Everyone Should Know" from Jeff Dean

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
<td>0.0005</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
<td>0.005</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
<td>0.007</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>100</td>
<td>0.10</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
<td>0.10</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>10,000</td>
<td>0.01</td>
</tr>
<tr>
<td>Send 1K bytes over 1 Gbps network</td>
<td>10,000</td>
<td>0.01</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
<td>0.25</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
<td>0.50</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
<td>10</td>
</tr>
<tr>
<td>Read 1 MB sequentially from network</td>
<td>10,000,000</td>
<td>10</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>30,000,000</td>
<td>30</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000</td>
<td>150</td>
</tr>
</tbody>
</table>
Storage Latency

- \(10^9\) Tape / Optical Robot
- \(10^6\) Disk
- Memory: 150
- L2 Cache: 10
- L1 Cache: 2
- Registers: 1

Latency:
- My Head: 1 min
- This Room: 10 min
- This Campus: 2 hr
- Sacramento: 2,000 Years
- Andromeda: 2 Years
- Pluto: 2 Years
- Tape / Optical Robot: 2,000 Years
Max Attainable Throughput

Varies significantly by device
  » 100 GB/s for RAM
  » 2 GB/s for NVMe SSD
  » 130 MB/s for hard disk

Assumes large reads (≫1 block)!
Storage Cost

$1000 at NewEgg today buys:
» 0.25 TB of RAM
» 9 TB of NVMe SSD
» 50 TB of magnetic disk
Hardware Trends over Time

**Capacity/$** grows exponentially at a fast rate (e.g. double every 2 years)

**Throughput** grows at a slower rate (e.g. 5% per year), but new interconnects help

**Latency** does not improve much over time
Most Common Permanent Storage: Hard Disks

Terms: Platter, Head, Actuator Cylinder, Track Sector (physical), Block (logical), Gap
Top View

tracks

sector

gap
Disk Access Time

I want block X

? in memory block x
Disk Access Time

Time = Seek Time + Rotational Delay + Transfer Time + Other
Seek Time

Time

3-5X

X

1

Cylinders Traveled

X

N

3

5X

1
Typical Seek Time

Ranges from
  » 4 ms for high end drives
  » 15 ms for mobile devices

In contrast, SSD access time ranges from
  » 0.02 ms: NVMe
  » 0.16 ms: SATA
Rotational Delay

Head Here

Block I Want
Average Rotational Delay

R = 1/2 revolution \hspace{1cm} R=0 \text{ for SSDs}

Typical HDD figures

<table>
<thead>
<tr>
<th>HDD Spindle [rpm]</th>
<th>Average rotational latency [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,200</td>
<td>7.14</td>
</tr>
<tr>
<td>5,400</td>
<td>5.56</td>
</tr>
<tr>
<td>7,200</td>
<td>4.17</td>
</tr>
<tr>
<td>10,000</td>
<td>3.00</td>
</tr>
<tr>
<td>15,000</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Transfer Rate

Transfer rate $T$ is around 50-130 MB/s

Transfer time: size / $T$ for contiguous read

Block size: usually 512-4096 bytes
So Far: Random Block Access

What about reading the “next” block?
If We Do Things Right (Double Buffer, etc)

Time to get = \frac{\text{block size}}{t} + \text{negligible}

Potential slowdowns:
» Skip gap
» Next track
» Discontinuous block placement

Sequential access generally much faster than random access
Cost of Writing: Similar to Reading

…. unless we want to verify!
need to add (full) rotation + block size / t
Cost To Modify a Block?

To Modify Block:

(a) Read Block
(b) Modify in Memory
(c) Write Block
[(d) Verify?]
Performance of DRAM

The same basic issues with “lookup time” vs throughput apply to DRAM

Min read from DRAM is a cache line (64 bytes)

Even 64-byte random reads may not be as fast as sequential ones due to prefetching, page table, controllers, etc

Place co-accessed data together!
Suppose we’re accessing 8-byte records in a DRAM with 64-byte cache line sizes.

How much slower is random vs sequential?

In the random case, we are reading 64 bytes for every 8 bytes we need, so we expect to max out the throughput at least 8x sooner.
Storage Hierarchy

Typically want to **cache** frequently accessed data at a high level of the storage hierarchy to improve performance.
Sizing Storage Tiers

How much high-tier storage should we have?

Can determine based on workload & cost

The 5 Minute Rule for Trading Memory Accesses for Disc Accesses
Jim Gray & Franco Putzolu
May 1985
The Five Minute Rule

Say a page is accessed every $X$ seconds

Assume a disk costs $D$ dollars and can do $I$ operations/sec; cost of keeping this page on disk is

$$C_{disk} = C_{iop} / X = D / (IX)$$

Assume 1 MB of RAM costs $M$ dollars and holds $P$ pages; then the cost of keeping it in DRAM is:

$$C_{mem} = M / P$$
Five Minute Rule

This tells us that the page is worth caching when $C_{mem} < C_{disk}$, i.e.

$$X < \frac{PagesPerMBofDRAM}{AccessesPerSecondPerDisk} \times \frac{PricePerDiskDrive}{PricePerMBofDRAM}$$

<table>
<thead>
<tr>
<th>Tier</th>
<th>1987</th>
<th>1997</th>
<th>2007</th>
<th>2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM–HDD</td>
<td>5m</td>
<td>5m</td>
<td>1.5h</td>
<td>4h</td>
</tr>
<tr>
<td>DRAM–SSD</td>
<td>-</td>
<td>-</td>
<td>15m</td>
<td>7m (r) / 24m (w)</td>
</tr>
<tr>
<td>SSD–HDD</td>
<td>-</td>
<td>-</td>
<td>2.25h</td>
<td>1d</td>
</tr>
</tbody>
</table>

Source: The Five-minute Rule Thirty Years Later and its Impact on the Storage Hierarchy
Disk Arrays

Many flavors of “RAID”: striping, mirroring, etc to increase performance and reliability

logically one disk
Common RAID Levels

**RAID 0**

- Striping across 2 disks: adds performance but not reliability

**RAID 1**

- Mirroring across 2 disks: adds reliability but not performance (except for reads)

**RAID 5**

- Striping + 1 parity disk: adds performance and reliability at lower storage cost

Coping with Disk Failures

Detection
  » E.g. checksum

Correction
  » Requires redundancy
At What Level Do We Cope?

Single Disk
  » E.g., error-correcting codes on read

Disk Array

Logical  ←  Physical
Operating System

E.g., network-replicated storage

Logical Block  Copy A  Copy B
Database System

E.g.,

Current DB → Log → Last week’s DB
Summary

Storage devices offer various tradeoffs in terms of latency, throughput and cost.

In all cases, data layout and access pattern matter because random ≪ sequential access.

Most systems will combine multiple devices.
Assignment 1

Explores the effect of data layout for a simple in-memory database

» Fixed set of supported queries
» Implement a row store, column store, indexed store, and your own custom store!

Now posted on website!