Large Systems:

Design + Implementation:

➢ Google Search

Image (c) Facebook
Case Study: Google Evolution

- Jeff Dean, “Building Software Systems at Google and Lessons Learned”, Stanford Computer Science Department Distinguished Computer Scientist Lecture, November, 2010
- https://research.google.com/pubs/jeff.html
Dealing with Growth

Ad System

Frontend Web Server

query

Cache servers

Doc Servers

Index servers

Index shards

Replicas
Dealing with Growth

Eventually have enough replicas so that total memory across all index machines can hold ONE entire copy of index in memory.
Early 2001: In-Memory Index

- Ad System
- Frontend Web Server
- Cache Servers
- Doc Servers
- Balancers
- Index servers
- Index shards

Query flow:
- From Ad System to Frontend Web Server
- From Frontend Web Server to Cache Servers
- From Cache Servers to Doc Servers
- From Doc Servers to Balancers
- From Balancers to Index servers
- From Index servers to Index shards

Shards:
- Shard 0
- Shard 1
- Shard 2
- Shard N
In-Memory Indexing Systems

- Many positives:
  - big increase in throughput
  - big decrease in query latency
    - especially at the tail: expensive queries that previously needed GBs of disk I/O became much faster and cheaper
      e.g. [ “circle of life” ]
In-Memory Indexing Systems

• Many positives:
  – big increase in throughput
  – big decrease in query latency
    • especially at the tail: expensive queries that previously needed
      GBs of disk I/O became much faster and cheaper
      e.g. [ “circle of life” ]

• Some issues:
  – Variance: query touches 1000s of machines, not dozens
    • e.g. randomized cron jobs caused us trouble for a while
  – Availability: 1 or few replicas of each doc’s index data
    • Availability of index data when machine failed (esp for important
      docs): replicate important docs
    • Queries of death that kill all the backends at once: very bad
Canary Requests

- Problem: requests sometimes cause server process to crash
  - testing can help reduce probability, but can’t eliminate

- If sending same or similar request to 1000s of machines:
  - they all might crash!
  - recovery time for 1000s of processes pretty slow

- Solution: send canary request first to one machine
  - if RPC finishes successfully, go ahead and send to all the rest
  - if RPC fails unexpectedly, try another machine
    (might have just been coincidence)
  - if fails $K$ times, reject request

- Crash only a few servers, not 1000s
Query Serving System, 2004 edition

- Cache servers
- Requests
- Root
- Parent Servers
- Repository Manager
- Leaf Servers
- Repository Shards
- File Loaders
- GFS
Leaf servers handle both index & doc requests from in-memory data structures.
Leaf servers handle both index & doc requests from in-memory data structures.
New Problems

- More collections to search besides Web
  - More structured: Maps
- Need more real-time results
More Real-Time

- Creating Index was batch process via MapReduce
  - Store all documents in GFS (==HDFS)
  - Run several MapReduce jobs to create index
  - Upload index to Leaf servers
- New documents would not show up in search results for 2-3 days [Peng and Dadek, 2010]
- Needed lower “time from crawl-to-search-hit”
- Solution:
  - New data storage system: Colossus / BigTable
  - Event-driven, incremental processing: Caffeine / Percolator
**BigTable**: Basic Data Model

- Distributed multi-dimensional sparse map
  
  \[(\text{row}, \text{column}, \text{timestamp}) \rightarrow \text{cell contents}\]

- Rows are ordered lexicographically
- Good match for most of our applications
BigTable: Tablets & Splitting

“language:”

“contents:”

“aaa.com”
“cnn.com”
“cnn.com/sports.html”

Tablets

“website.com”

“zuppa.com/menu.html”
Caffeine / Percolator

- Crawler uploads new version of page in BigTable
- Updates to BigTable can trigger code
- E.g. code to create index
- Push index update to Leafs

```
<table>
<thead>
<tr>
<th>Rows</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;www.cnn.com&quot;</td>
<td>&quot;&lt;html&gt;...&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;contents:&quot;</td>
</tr>
<tr>
<td></td>
<td>Timestamps</td>
</tr>
</tbody>
</table>
```