Extreme Computing Revision

Emphasis on solving problems

Design a system to store YouTube videos.
You're Twitter. Make a low-latency feed system.
Show the most frequently visited pages. Live.

Reason About a Problem

- What's the bottleneck? Compute, storage, bandwidth, latency, ...
- Which data is big? Which is small? $\rightarrow$ Join strategies
- Is it balanced?
- What if that machine fails?

Reason About a Problem

- What's the bottleneck? Compute, storage, bandwidth, latency, ...
- Which data is big? Which is small? $\rightarrow$ Join strategies
- Is it balanced?
- What if that machine fails?

More fun: how will this system fail?

Bag of Tricks

- Sharding aka partitioning: divide the work across machines
- Replication for speed and fault tolerance
- “Cold” large read-only store, “hot” small mutable store
- Approximations: Bloom filters, streaming counts, resevoir sampling

Systems

MapReduce	Parallel batch processing
BitTorrent	File sharing
HDFS	File storage
Chord	Divide responsibility as machines join and leave
SSTable	Large read-only key-value store
BigTable	Large mutable key-value store

Systems

MapReduce	Parallel batch processing
BitTorrent	File sharing
HDFS	File storage
Chord	Divide responsibility as machines join and leave
SSTable	Large read-only key-value store
BigTable	Large mutable key-value store

Given a problem, name a system and apply it.
Take inspiration from the design of these systems.

Linearisable versus Sequential

Alice and Bob write checks to each other for the same amount.

Alice's Statement

-10	Check Alice $\rightarrow$ Bob
0	Check Bob $\rightarrow$ Alice

Bob's Statement

-10	Check Bob $\rightarrow$ Alice
0	Check Alice $\rightarrow$ Bob

Both overdraft.

✓ Sequential: each client sees a consistent order

✗ Linearizable: no globally linear story