CISC 7610 Lecture 5
Distributed multimedia databases

Topics:

Scaling up vs out

Replication

Partitioning

CAP Theorem

NoSQL

NewSQL

Motivation

- YouTube receives 400 hours of video per minute
- That is 200M hours per year
- At 12 GB/hour (for H.264 HD quality)
- That is 2.6 Exabytes (2,600,000 TB) per year

- Which is more than one machine can handle
  - So how do we spread this across multiple machines?

http://www.reelseo.com/vidcon-2015-strategic-insights-tactical-advice/

Scaling up vs scaling out

- Scaling up: buy a bigger server
  - Pro: Code stays pretty much the same
  - Con: Expensive, hard limits
- Scaling out: buy more servers
  - Pro: Much cheaper, soft limits
  - Con: Much more complicated code (distributed systems)

Example: Plenty of fish

- PlentyOfFish.com: ad-funded dating site
- 1.2 billion page views per month
- 500,000 average unique logins per day
- 30+ million hits per day (500-600 per second)
- Top 30 site in the US, top 10 in Canada
- 5 servers (2 application, 3 DB)
- $10M in yearly revenue
- 1 employee

http://blog.codinghorror.com/my-scaling-hero/

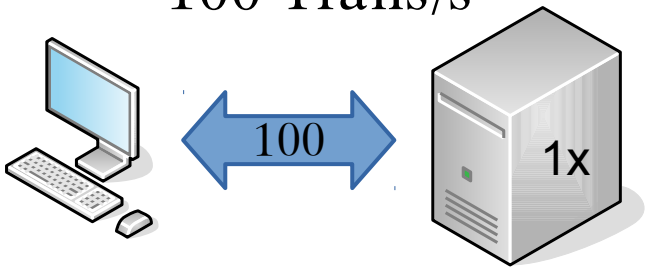
New database server

- Scaling up: buy a bigger server
  - HP ProLiant DL785: $100,000
  - CPU: 32, RAM: 512 GB, Disk: 4 TB
  - Space: 7 rack units, Power: $1600/yr, MS Licenses: $10k
- Scaling out: buy more servers
  - Lenovo ThinkServer RS110, 8 GB RAM, quad-core CPU: $1200
  - Can get 83 of these for $100,000
  - CPU: 332, RAM: 664 GB, Disk: 40 TB
  - Space: 83U, Power: $22k/yr, MS Licenses: $80k

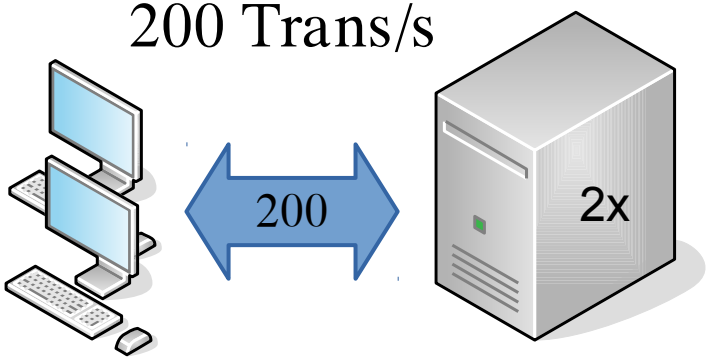
http://blog.codinghorror.com/scaling-up-vs-scaling-out-hidden-costs/

Scaling out: Replication vs partitioning

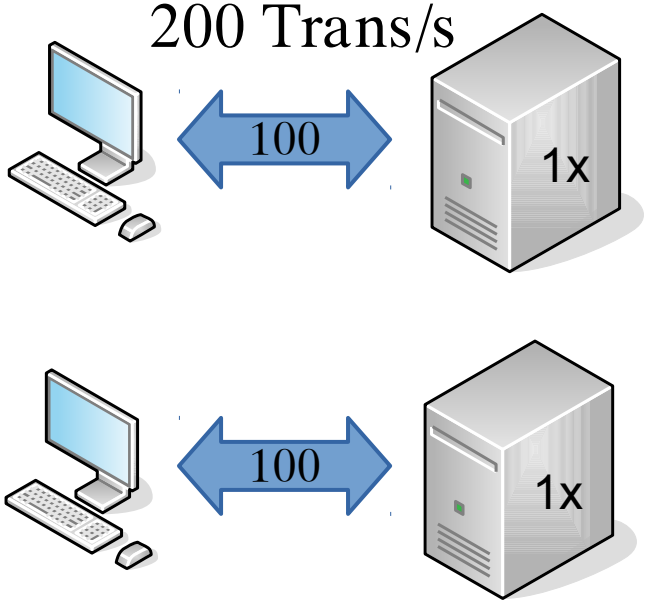
Initial system
100 Trans/s



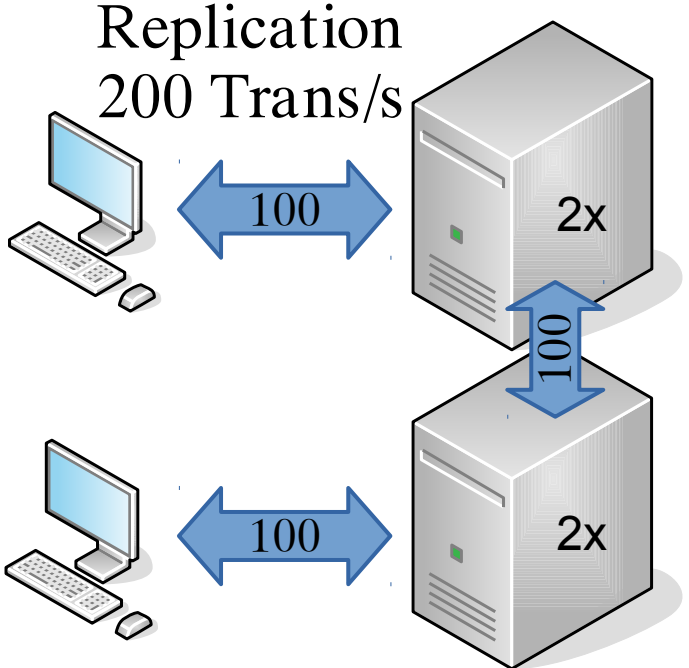
Scale up
200 Trans/s



Scale out
Partitioning
200 Trans/s

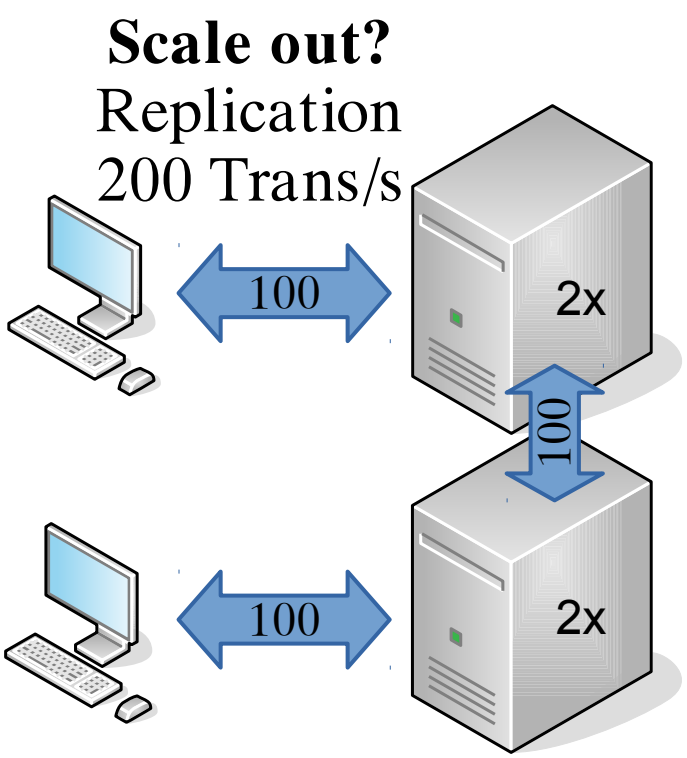


Scale out?
Replication
200 Trans/s



Replication: copy and coordinate

- Replication maintains identical copies of data on multiple machines
- Reads can come from any machine
- Writes must be applied to all machines



Purpose of replication

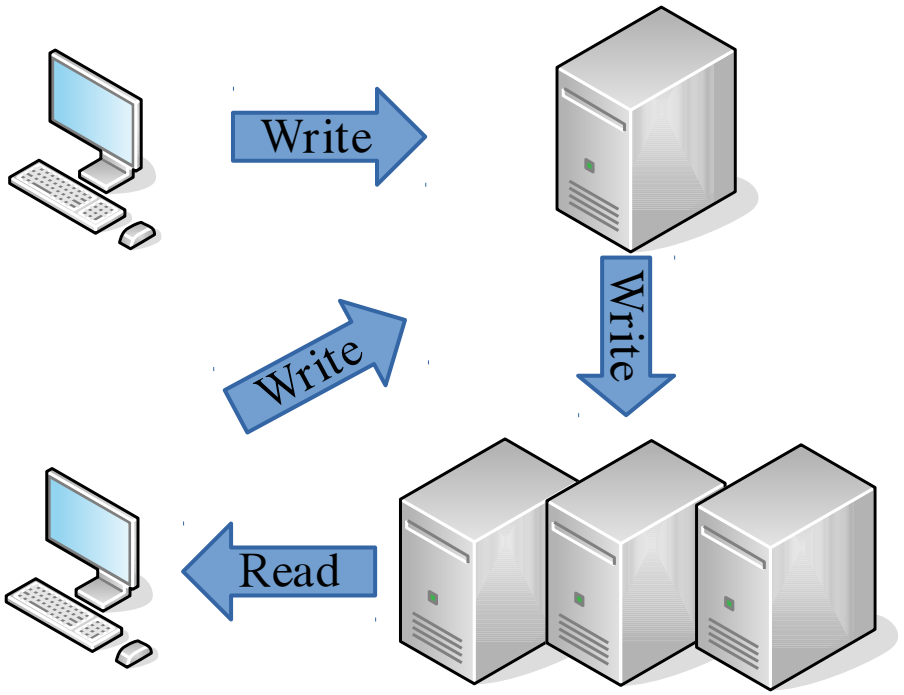
- Data distribution
  - Geographical diversity for lower latency and redundancy
- Load balancing
  - Spread requests among multiple servers
- Backups and recovery
  - Make and restore copies without downtime
- High availability
  - Hot spare for fast recovery

Types of replication

- Eager / synchronous
  - Transaction waits for all replicas to be updated
  - Maintains consistency, but can cause delays
- Lazy / eventual / asynchronous
  - Transaction waits for one replica to be updated
  - Changes propagated to other replicas eventually
  - Faster, but can lead to conflicts between replicas modified in different ways

Master/slave replication

- Only “master” replica accepts writes
  - Avoid consistency issues
  - Sends updates to others
  - Single point of failure
- All replicas serve reads
- If master goes down, another replica can be promoted



Partitioning

- Sometimes, data can be divided into uncoupled or loosely coupled partitions
  - Then scaling to more machines just requires dividing into more partitions
- Horizontal partitioning: divide relations by ID
  - Also known as sharding
- Vertical partitioning: divide relations by attributes
  - Compare to database normalization

Partitioning example

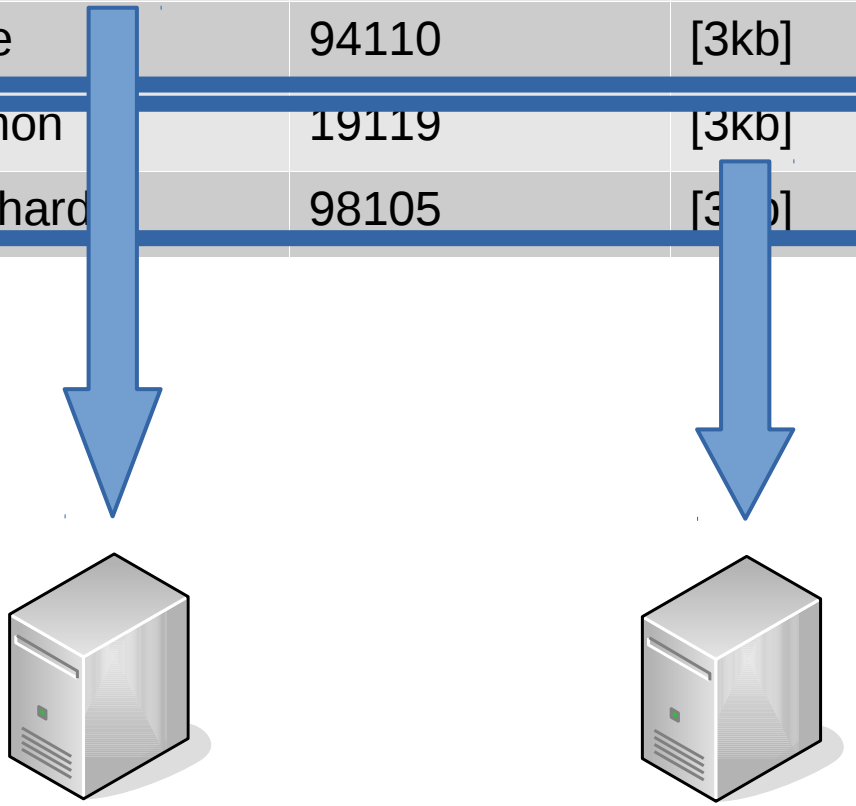
ID	Name	Zipcode	Thumbnail	Photo
1	David	02138	[3kb]	[2MB]
2	Jared	43201	[3kb]	[2MB]
3	Sue	94110	[3kb]	[2MB]
4	Simon	19119	[3kb]	[2MB]
5	Richard	98105	[3kb]	[2MB]

Vertical Partitioning example

ID	Name	Zipcode	Thumbnail	Photo
1	David	02138	[3kb]	[2MB]
2	Jared	43201	[3kb]	[2MB]
3	Sue	94110	[3kb]	[2MB]
4	Simon	19119	[3kb]	[2MB]
5	Richard	98105	[3kb]	[2MB]

Horizontal Partitioning example

ID	Name	Zipcode	Thumbnail	Photo
1	David	02138	[3kb]	[2MB]
2	Jared	43201	[3kb]	[2MB]
3	Sue	94110	[3kb]	[2MB]
4	Simon	19119	[3kb]	[2MB]
5	Richard	98105	[3kb]	[2MB]



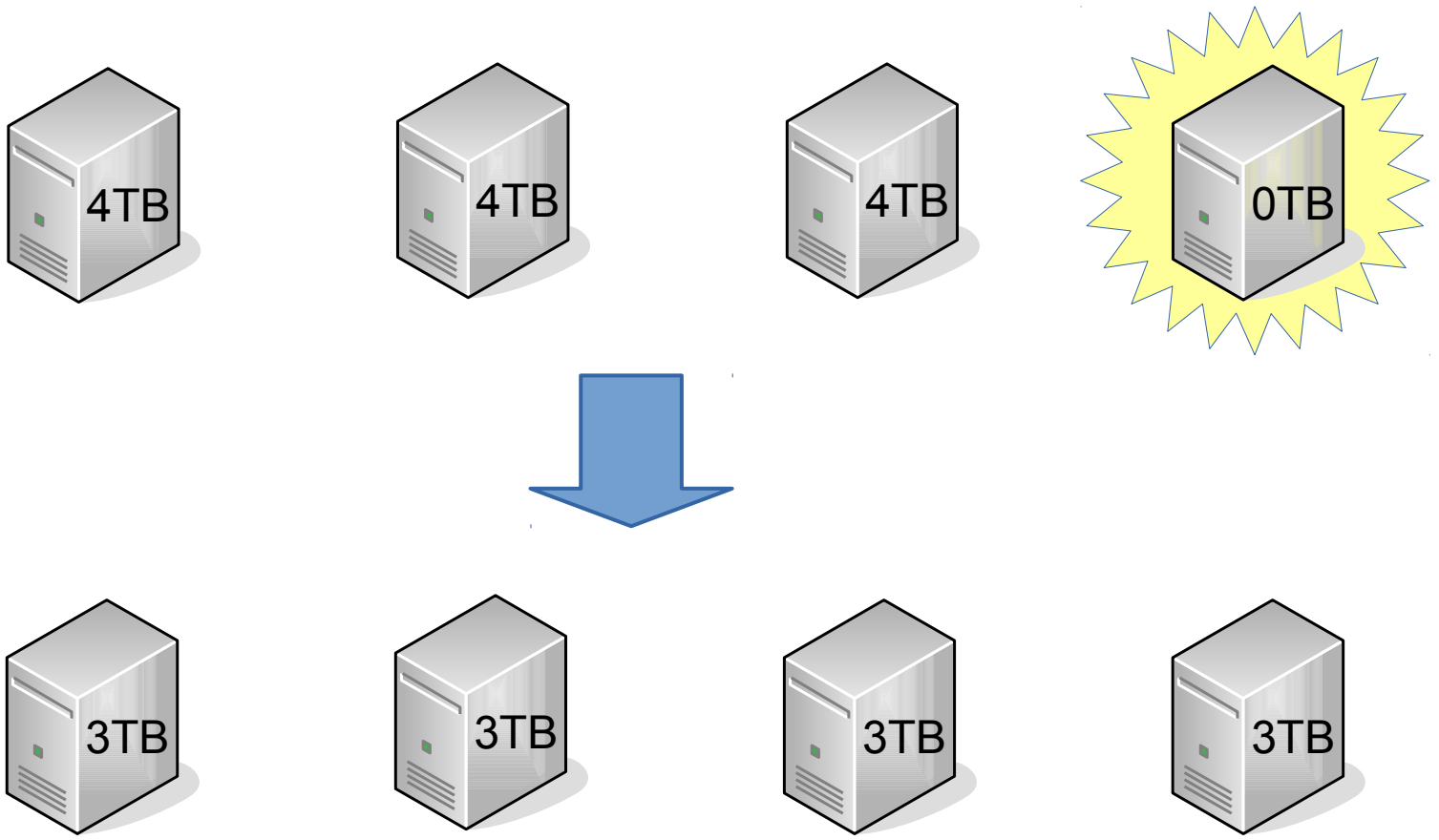
Horizontal Partitioning advantages

- Higher write bandwidth than replication
  - All shards accept writes
  - So N times higher capacity
- Higher scalability than replication
  - Continue to sub-divide shards to scale up without limit

Horizontal Partitioning disadvantages

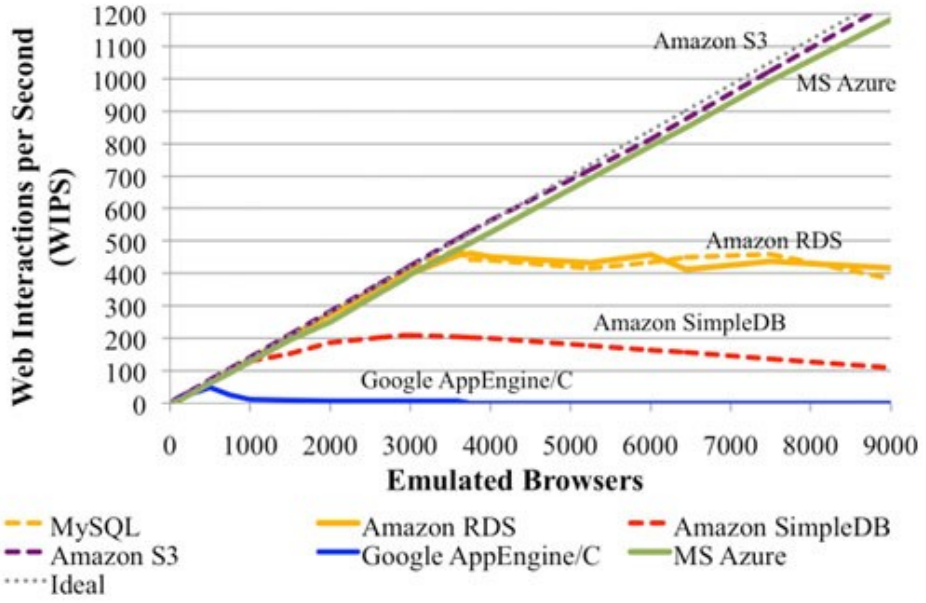
- Re-balancing is necessary and costly
  - When one shard grows too large
  - When adding new machines
- Cross-shard joins are slow or impossible
  - Additional reliance on network
  - Shards could be in separate data centers (US vs Europe)

Re-balancing shards



Scalability

- How hard is it to double your current capacity?
  - Capacity for traffic, storage, transactions
  - Capacity for read or write operations
- Independent of current processing speed



http://highscalability.com/blog/2010/5/26/end-to-end-performance-study-of-cloud-services.html

Brewer's CAP theorem

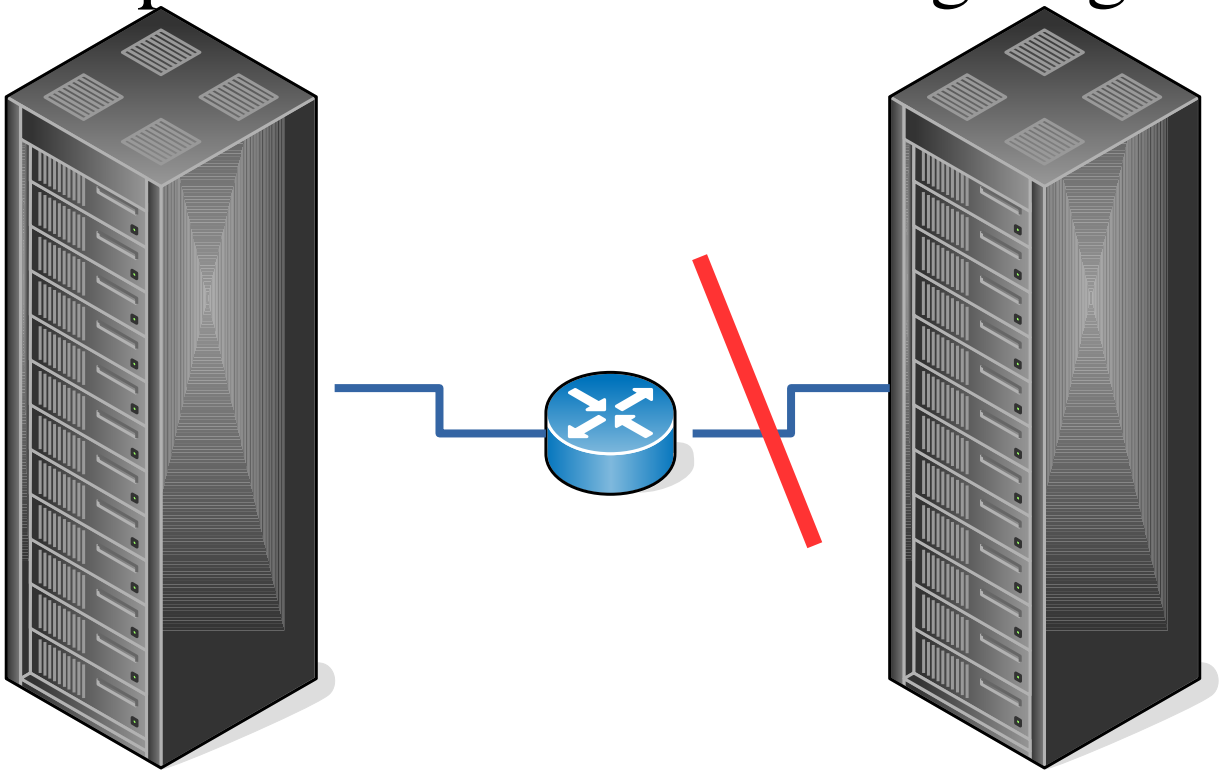
- These three properties cannot be achieved by a distributed system simultaneously
  - Strong Consistency: All clients see the same data at the same time
  - Availability: All requests receive a response as to their success or failure
  - Partition tolerance: The system continues to function in the event of network failures

Brewer's real CAP theorem

- A node fails to communicate with another node to keep it in sync (P)
- It can decide to
  - Go ahead anyway, sacrificing consistency (C)
  - Wait for the other node, sacrificing availability (A)
- Business considerations: availability > consistency
  - Take the customer's order and sort it out later if necessary

Partitions are inevitable

- A network is partitioned when a component fails and a cluster is divided in two
- Want applications to keep operating
- Can't tell a partition from a node going down



Sacrificing availability

- Wait until all nodes can synchronize
- “Amazon claim that just an extra one tenth of a second on their response times will cost them 1% in sales. Google said they noticed that just a half a second increase in latency caused traffic to drop by a fifth.”
- Examples: Multi-master DBs, Neo4j, Google BigTable

http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

Sacrificing consistency

- Go ahead with update (Eventual consistency)
- Problems
  - Pushes complexity from database into application
  - When is “eventually”?
- Examples
  - Domain name service
  - Facebook’s Cassandra and Voldemort
  - CouchDB

NoSQL vs NewSQL

- NoSQL: restrictive interpretation of CAP theorem
  - Throw out ACID transactions with SQL
  - In order to increase scalability
- NewSQL: more nuanced interpretation of CAP
  - Regain ACID transactions and SQL
  - Maintain scalability by optimizing for quick, localized transactions
- SQL is just the query language, independent of CAP
  - And still need a way to query

Types of NoSQL Databases

- Key-value store
- Key-document store
- Column-family stores
- Graph databases (again)

Key-value store: Memcached and Redis

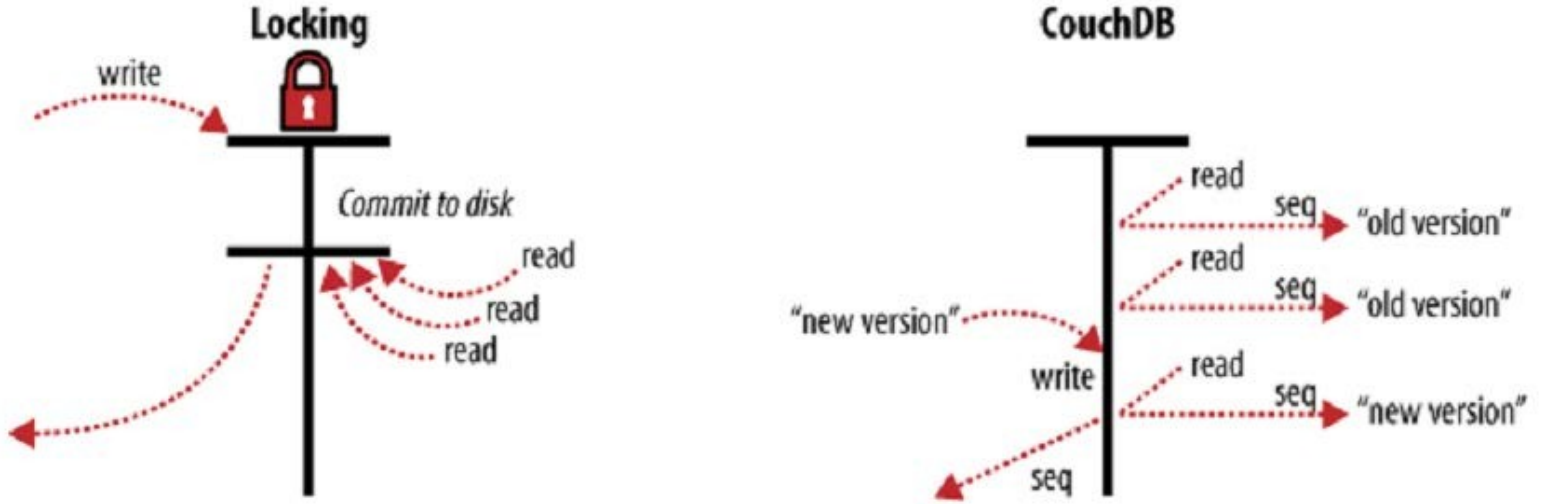
Memcached: key-value store	
Key1	Value1
Key2	Value2
Key3	Value3
Key4	Value4

Redis: key-structure store	
Key1	[1,2,5,2,1]
Key2	{'a':1, 'b':2}
Key3	7
Key4	(1,2,5)

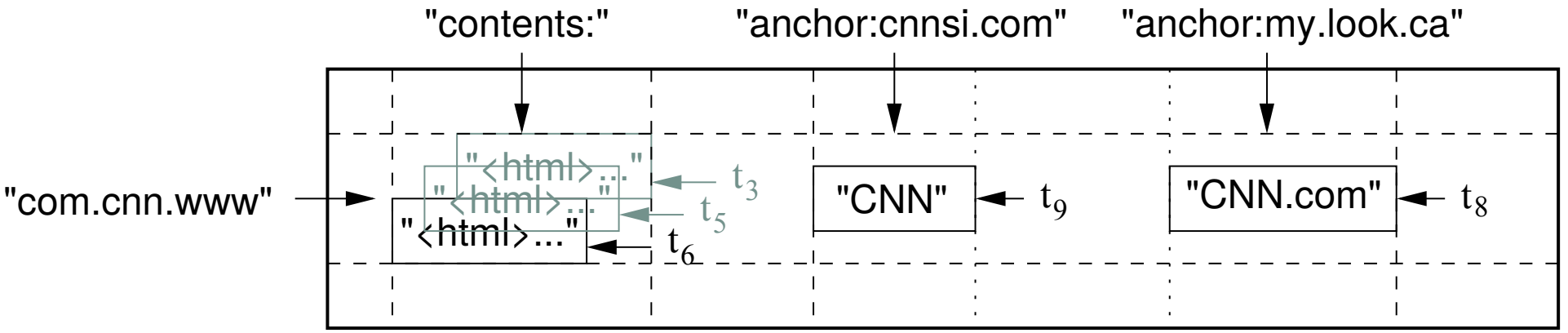
- Can only “query” on one attribute (key)
  - But that is sufficient for many applications
  - Especially for de-normalized data
- Redis allows additional atomic operations on values
- Very easy to scale

Document store: CouchDB

- Stores arbitrary “documents” (schema-less)
  - Indexes various aspects of each document for querying
- Eventual consistency through multi-version concurrency control
  - Writes create new versions of documents instead of waiting for locks



Column-family store: BigTable



- Maps (rowId, columnId, timestamp) to data
- Stores keys in sorted order to allow range queries
- Heavy use of sharding through “tablets”
- No transactions

Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." ACM Transactions on Computer Systems (TOCS) 26.2 (2008): 4.

NewSQL: Google F1

- Adds transactions to BigTable. Lets application developers deal with transaction bottlenecks
- Key ideas
  - Scalability: auto-sharded
  - Availability & Consistency: synchronous replication
  - High commit latency that can be hidden
- Single unified database running AdWords: 10s of TBs, 1000s of machines

No Compromise Storage Google

	Sharded MySQL	NoSQL (Bigtable)	F1 & Spanner
Consistent reads and ACID	✔ (per shard)		✔ (global)
SQL Query	✔ (per shard)		✔ (global)
Schema mgmt.	✔ (downtime required)		✔ (nonblocking)
Indexes	✔		✔
"Infinite" scaling		✔	✔
MapReduce		✔	✔
High Availability	Mostly		✔

http://www.slideshare.net/dmconf/f1-the-distributed-sql-database-supporting-googles-ad-business-bart-samwell

NewSQL: VoltDB

- Uses relational data model
- Uses SQL as its primary interface
- Targeted at applications with transactions that are
  - Short-lived (i.e., no user stalls)
  - Touch a small subset of data using index lookups (i.e., no full table scans or large distributed joins)
  - Are repetitive (i.e. executing the same queries with different inputs)
- In-memory
- Based on stored procedures (Java + SQL)

Summary

- Scaling up is easier, but scaling out is more sustainable in the long term
- Replication copies data, allowing parallel reads
- Partitioning divides up data to allow parallel writes
- CAP theorem says that you must choose between availability and consistency
- NoSQL sacrifices ACID transactions for scalability
- NewSQL maintains ACID transactions with scalability

Further reading

- http://faculty.cs.nku.edu/~waldenj/classes/2014/spring/cit668/
- http://www.aosabook.org/en/nosql.html
- http://www.slideshare.net/GrishaWeintraub/cap-28353551
- Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." ACM Transactions on Computer Systems (TOCS) 26.2 (2008): 4.
- http://www.slideshare.net/dmconf/f1-the-distributed-sql-database-supporting-googles-ad-business-bart-samwell