Distributed Data, Replication and NoSQL

Administrivia

- **Final Exam**
  - Monday, May 11, 11:30-2:30
  - RSF Fieldhouse
  - Cumulative, stress end-of-semester
  - 3 cribsheets
- **Review Sessions**
  - At usual classtime, will be webcasted
  - TAs and Prof H holding OH and sections next week
    - TA Hours on Piazza

Outline

- **Basics**
  - Traditional DB replication
  - Replica consistency
  - Where do updates happen?
- **Considerations from Massive Scale**
  - Performance requirements
  - NoSQL & Eventual Consistency
- **Programming with Inconsistent data**
  - Approaches
  - Expected costs
  - CALM
- **NoSQL today**

Why Replicate Data?

- **Increase availability**
  - Survive catastrophic failures
    - Avoid correlation: different rack, different datacenter
  - Bypass transient failures
- **Reduce latency**
  - Choose a "nearby" server
    - Particularly for geo-distributed DBs
  - Ask many servers and take the 1st response
- **Load balancing**
- **Other reasons**
  - facilitate sharing, security, autonomy, system management, etc.

A Definition: Replica Consistency

- **Replica Consistency**
  - Typically *linearizability* of single writes
  - Illusion: all copies of data item X updated atomically
  - Really: readers see values of X that they might see in a single-node setting
- **Not to be confused with the C in ACID!**
  - This has caused endless headaches
- **Not to be confused with serializablity**
  - Actions only; not transactions. Single-object writes
  - You can layer serializability on top of linearizable stores

Traditional DB Replication Mechanisms

- **Small number of big databases (say one per continent)**
  - Replicated for failure recovery, mostly
- **Data-shipping**
  - Interruptive at both ends
  - Difficult to get transactional guarantees
    - Even single-site transactions would require 2PC!
  - Relatively easy interoperability across different systems
- **Log-shipping**
  - Cheap/free at the source node
  - Modest bandwidth: diffs
  - Single-node transactional replication comes built-in
    - in a snapshot sense at least
  - Tricky to do across different systems
    - Replaying Oracle logs to an MS SQL Server DB? Ick.

Single- vs. Multi-Master Writes

- **Single-master**
  - Every data item has one appointed Master node
  - Writes are first performed at the Master
  - Replication propagates the writes to others
  - 2PL/2PC can be used for transactions
  - Easy to understand, but defeats many benefits for writers
    - Esp. reducing latency, also load balancing
- **Multi-master**
  - Writes can happen anywhere
  - Replication among all copies
  - Allows even writers to enjoy lower latency, load balancing
  - Fast, but hard to make sense of things
    - Same item can be updated in two places; which update "wins"?
    - 2PL/2PC defeat the positive effects of multi-master

A Compromise: Quorums

- **Suppose you have N nodes replicating each item**
- **Send each write to W<N nodes**
  - sender timestamps the write
- **Send each read to R<N nodes**
  - reader uses timestamps to choose result
- **How to ensure readers see latest data:**
  - ensure W+R >= N ("strict" quorum)
  - E.g. W = R > N/2 (majority quorum)
  - E.g. W=N, R=1 (writer broadcasts)
- **Advantages**
  - Not at the mercy of a single slow "master"
  - Relatively easy to recover from a failed node
- **But…**
  - Transactional semantics still require 2PC

Is 2PC really so expensive?

- **How expensive is it to hold a unanimous vote?**
  - Raw latencies depend on your network
    - Geo-replication and speed of light
    - Vs. modern datacenter switches
  - But best-case behavior is not a good metric!
- **Much depends on your machines' delay distribution**
  - Hardware: NW switches, mag disk vs. Flash, etc.
  - Software: GC in the JVM, carefully-crafted event handlers…

Google does it; it must be good?!

- **Google Spanner uses 2PC and a host of other stuff**
- **But the latencies! Speed of light.**
  - 7 times round the world per second

10 TPS!

operation	latency (ms) mean	std dev	count
all reads	8.7	376.4	21.5B
single-site commit	72.3	112.8	31.2M
multi-site commit	103.0	52.2	32.1M

Hamilton Quote on Coordination

"The first principle of successful scalability is to batter the consistency mechanisms down to a minimum, move them off the critical path, hide them in a rarely visited corner of the system, and then make it as hard as possible for application developers to get permission to use them"

—James Hamilton (IBM, MS, Amazon)

But only relevant to massive scalability!

Outline

- **Basics**
  - Traditional DB replication
  - Replica consistency
  - Where do updates happen?
- **Considerations from Massive Scale**
  - Performance requirements
  - NoSQL & Eventual Consistency
- **Programming with Inconsistent data**
  - Approaches
  - Expected costs
  - CALM
- **NoSQL today**

Update-heavy Global-Scale Services

- **E.g. Amazon, Facebook, LinkedIn, Twitter**
  - Shopping, Posting and Connecting
- **Latency and Availability both paramount**
  - Favors multi-master solutions, or loose quora
  - Replica consistency becomes frustrating
    - See Hamilton
- **Becomes quite natural to ditch Consistency!**
  - The rise of NoSQL

NoSQL

- **Born in Amazon Dynamo**
- **Typical NoSQL approach**
  - Simple key/value data model.
    - `put(key, val). get(key, val)`
  - Replicated data (think 3+ replicas)
  - Multi-master: write anywhere, with timestamp
  - Update is acked by a single node
    - No 2PC!
  - Update "gossiped" lazily among the replicas in background
    - A.k.a "anti-entropy", "hinted handoff" and other fancy names
  - Reads can consult 1 or more replicas
    - Quorums can be used to achieve linearizability

NoSQL ambiguities

- **Atypical to bother with linearizability**
- **Per-object ambiguities**
  - You may not read the latest version
  - Writers can conflict
    - Write the same key in different places
    - 'Conflict Resolution' or 'Merge' rules apply:
      - E.g. Last/First writer wins
        - But what clock do you use?
      - E.g. Semantic merge (e.g. Increment)
- **Across objects**
  - Typically no guarantees
  - No notion of "trans"-actional semantics

"Eventual Consistency"

"if no new updates are made to the object, eventually all accesses will return the last updated value"

-- Werner Vogels, "Eventually Consistent", CACM 2009

- Which in practice means...??

Problems with E.C.

- **Two kinds of properties people discuss in distributed systems:**

1. **Safety:** *nothing bad ever happens*
   - False! At any given time, the state of the DB may be "bad"
2. **Liveness:** *a good thing eventually happens*
   - Sort of! Only in a "quiescent" eventuality.

Outline

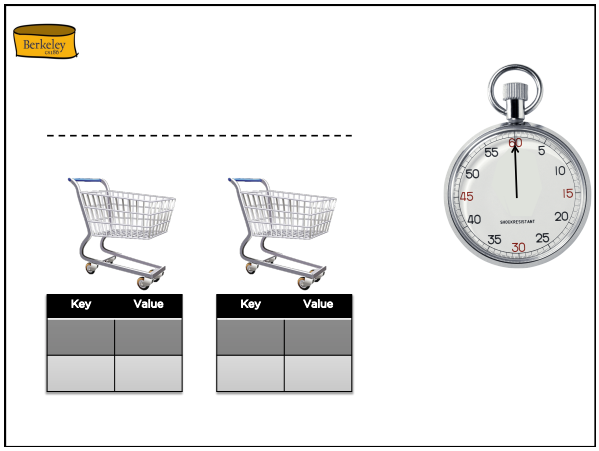
- **Basics**
  - Traditional DB replication
  - Replica consistency
  - Where do updates happen?
- **Considerations from Massive Scale**
  - Performance requirements
  - NoSQL & Eventual Consistency
- **Programming with Inconsistent data**
  - Approaches
  - Expected costs
  - CALM
- **NoSQL today**

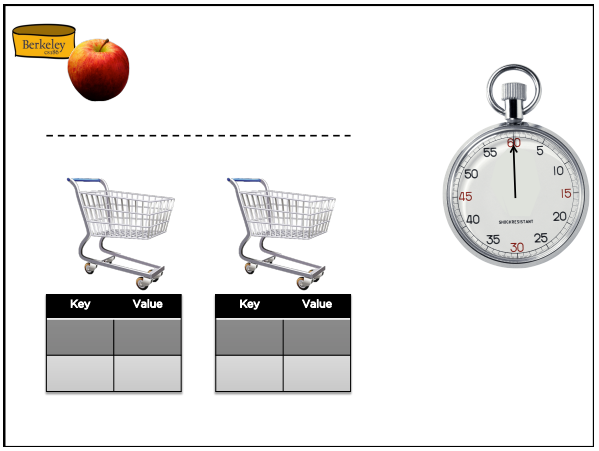
How do programmers deal with EC?

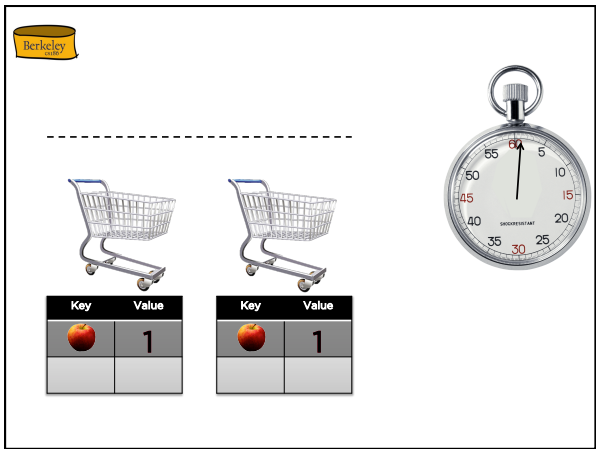
1. **What, me worry?**
2. **Application-level coordination**
3. **Provably consistent code: monotonicity**

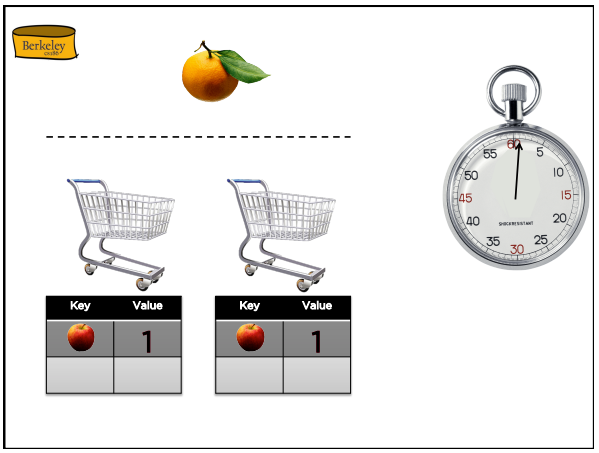
Case Study: The Shopping Cart

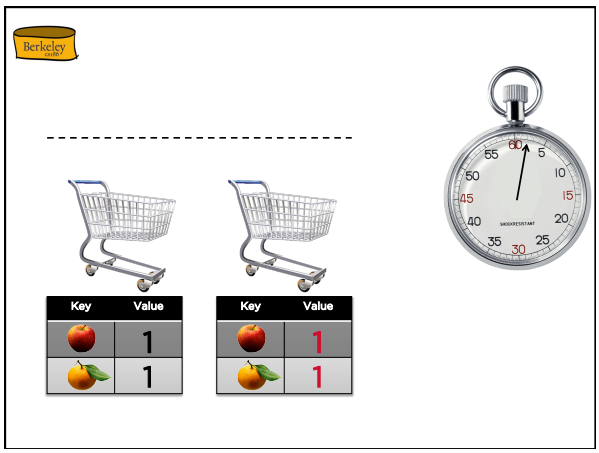
- **Based on Amazon Dynamo**
- **With the global 2PC commit order "clock"**

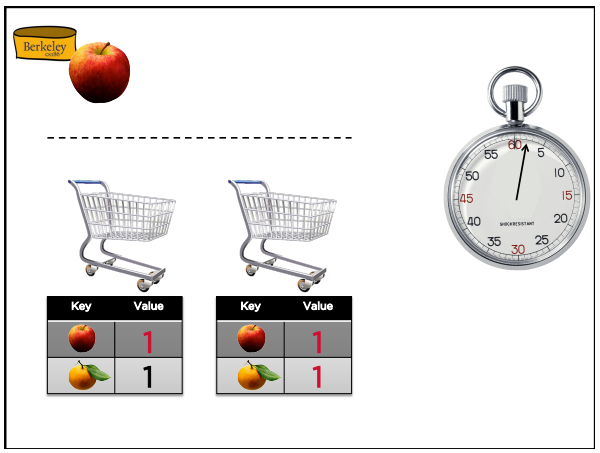


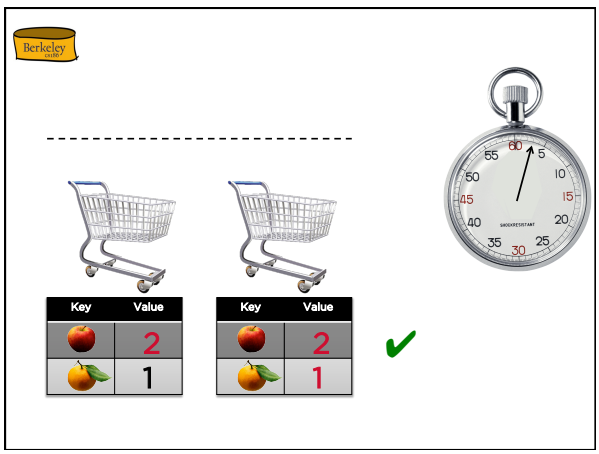






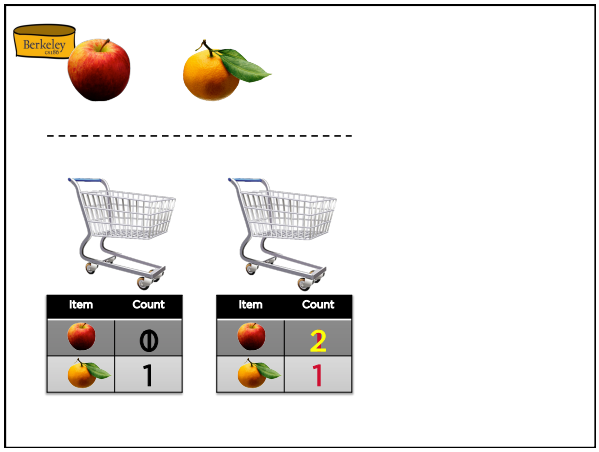






Eventually Consistent Version

- **What could go wrong?**

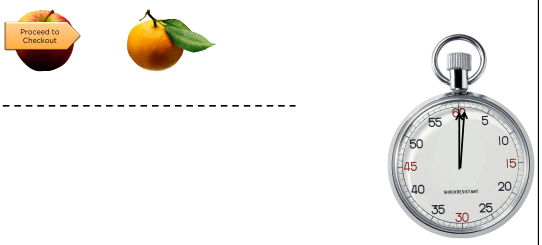


What, me worry?

- **How often does inconsistency occur?**
  - E.g. What are the odds of a "stale read"?
  - P.B.S. Study at Berkeley (pbs.cs.berkeley.edu)
    - Based on LinkedIn and Yammer traces
    - Stale reads are pretty rare
      - Esp in a single datacenter, using flash disks
    - But they happen!
- **How bad are the effects of inconsistency?**
  - In terms of application semantics
  - In terms of exception detection/handling ("apologies")

Application-level Reasoning

- **The most interesting part of the Dynamo paper is its application-level smarts!**
- **Basic idea:**
  - Don't mutate values in a key-value store
  - Accumulate logs at each node!
    - Union up a set of items
    - Union is commutative/associative!
  - Only coordinate for checkout



Monotonic Code

- **Merge things "upward"**
  - sets grow bigger (merge: Union)
  - counters go up (merge: MAX)
  - booleans go from false to true (merge: OR)

Intuition from the Integers

VON NEUMANN

```
int ctr;

operator:= (x) {
      // assign
      ctr = x;
}
```

MONOTONIC

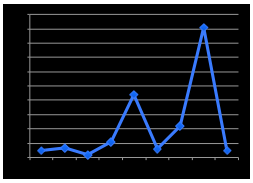
```
int ctr;

operator<= (x) {
  // merge
  ctr = MAX(ctr, x);
}
```

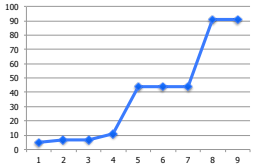
DISORDERLY INPUT STREAMS:
2, 5, 6, 7, 11, 22, 44, 91
5, 7, 2, 11, 44, 6, 22, 91, 5

Intuition from the Integers

VON NEUMANN



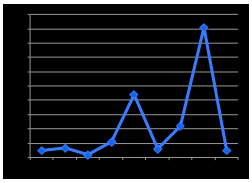
MONOTONIC



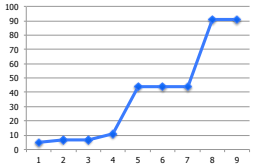
DISORDERLY INPUT STREAMS:
2, 5, 6, 7, 11, 22, 44, 91
5, 7, 2, 11, 44, 6, 22, 91, 5

Intuition from the Integers

VON NEUMANN



MONOTONIC

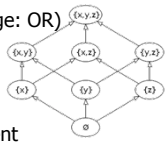


DISORDERLY INPUT STREAMS:
2, 5, 6, 7, 11, 22, 44, 91
5, 7, 2, 11, 44, 6, 22, 91, 5

+ monotonic "progress"
+ order-insensitive outcome

General Monotonicity

- **Merge things "upward"**
  - sets grow bigger (merge: Union)
  - counters go up (merge: MAX)
  - booleans go from false to true (merge: OR)
- **Can be partially ordered**
  - E.g. sets growing
- **Note why this works!**
  - Associative, Commutative, Idempotent
- **Core mathematical objects**
  - Lattices & Monotonic logic
    - E.g. Select/project/join/union. But not set-difference, negation



CALM Theorem

- **When can you have consistency without coordination?**
  - Really.
  - I mean really. Like "complexity theory" really.
  - I.e. given application X, is there *any* way to code it up to run correctly without coordination?
- **CALM: Consistency As Logical Monotonicity**
  - Programs are eventually consistent (without coordination) iff they are monotonic.
    - Monotonicity => coordination can be avoided (somehow)
    - Non-monotonicity => coordination required

Hellerstein: The Declarative Imperative, 2010

Example

- **The fully monotone shopping cart**


✔ ✔ A "seal" or "manifest"

The Burden on App Developers

- **Given the tools you have**
  - Java/Eclipse, C++/gdb, etc.
- **Convince yourself your app is correct**
  - No race conditions? Fault tolerant?
- **Convince yourself your app will *remain* correct**
  - Even after you are replaced
- **Convince yourself the app plays well with others**
  - Does your eventual consistency taint somebody else's serializable data?
- **Wow. OK. Do you miss transactions yet?**
  - Amazon reportedly replaced Dynamo
  - Transactions could be practical in a datacenter
    - MS Azure makes use of this
  - But what about global, high-performance systems?

Shameless plug: BOOM

- **Really good programmers avoid coordination**
  - Maybe
- **Everyone else needs a better programming model**
  - One that encourages monotonicity, and analyzes for it.
  - E.g. Bloom (http://bloom-lang.net)
    - + Confluence analysis (Blazes)
    - + Fault Tolerance analysis (Molly)
    - Etc.
  - See http://boom.cs.berkeley.edu

Outline

- **Basics**
  - Traditional DB replication
  - Replica consistency
  - Where do updates happen?
- **Considerations from Massive Scale**
  - Performance requirements
  - CAP, NoSQL, Eventual Consistency
- **Programming with Inconsistent data**
  - Approaches
  - Expected costs
  - CALM
- **NoSQL today**

Data Models

- **Key/Value Stores**
  - Opaque values
- **"Document" Stores**
  - Transparent values: JSON or XML
  - Amenable to indexing by tags
  - Query support still usually pretty simple
- **Think about these in terms of schema design (e.g. ER diagrams)!**

Popular NoSQL Systems

- **Hbase, Cassandra, Voldemort, Riak, Couchbase, CouchDB, MongoDB, etc. etc.**
- **Why so many?**
  - Not all that hard to build
  - Lack of standards, design alternatives
- **Interesting to compare to Hadoop**
  - Why no NoSQL core? Cause or effect?

Should you be using NoSQL?

- **Data model and query support**
  - Do you want/need the power of something like SQL?
    - And are your queries canned, or ad-hoc?
  - Do you want/need fixed or flexible schemas
    - Note that you can put flexible data into a SQL database
      - See for example PostgreSQL's JSON support
- **Scale**
  - Do you want/need massive scalability and high availability?
    - What's your data volume? Update/query workload?
    - Will you need geo-replication
  - Are you willing to sacrifice replica consistency?
- **Agility and growth**
  - Are you building a service that could grow exponentially?
  - Optimizing for quick, simple coding?
    - Or maintainability?

Data & Lessons to Live By

Topics

- **Everything in the slides *except:***
  - FDs: What does 3NF achieve, Minimal Covers, Decomp into 3NF
    - But you *are* responsible for the definition of 3NF
  - Text Search: Crawlers
  - Data Vis: Vega
  - Guest Lectures
    - Big Data
    - Data Science
  - Data replication and NoSQL
  - And...

Topics, cont.

- **In the Advanced Concurrency Control Lecture*:***
  - You are *not* responsible for:
    - Index concurrency (B-link trees, phantoms)
    - Distributed concurrency (2PC)
    - Weak Isolation
  - You *are* responsible for:
    - Multigranularity locking
    - MVCC

As you study…

- "Reading maketh a full man; conference a ready man; and writing an exact man."
  -Francis Bacon
- "If you want truly to understand something, try to change it."
  -Kurt Lewin
- "I hear and I forget. I see and I remember. I do and I understand."
  -Chinese Proverb.
- "Knowledge is a process of piling up facts; wisdom lies in their simplification."
  -Martin H. Fischer

Data Systems are More Similar than Different

- **Relational DBs**
- **Big Data Analytics**
- **NoSQL Key-Value Stores**
- **Search engines**
- **File Systems**

- **Don't learn a system — learn the principles!**
  - examine how tricks vary across these use cases
  - draw connections
  - Innovate!

Things are Changing

- **Cloud-scale**
- **Multicore**
- **Flash**
- **Memory-resident databases**

- **Opportunity to revisit, refactor**
  - Not just the monolithic relational DBMS
  - Maybe disk seek is free
  - Maybe on-chip cache locality is important
  - Maybe there's lots of room for multiple versions of data
  - Etc. Etc.
- **You've been trained to roll with—and drive—the change!**
  - Well-grounded and open-minded.

Note: All of CS Moves This Way

- **Data grows exponentially.**
- **The number of cores grows exponentially.**
- **Programming *is* (parallel) data management.**
- **Data-centric methods unlock parallelism.**
  - declarative, disorderly

Note: All of human endeavour too

- **Data-centric, measurement-driven thinking across society**
  - Government, policy
  - Economics
  - Medicine
  - Science
  - Business
  - Etc.
- **There's more to data than advertising!**

You Have Enormous Value

- **Job market**
  - McKinsey Big Data Report, 2011:
    - "The US alone faces a shortage of 140,000 to 190,000 people with deep analytical skills as well as 1.5 million managers and analysts to analyze big data and make decisions based on their findings."
  - Choose your own adventure!
    - engineering, sales, marketing, R&D, mgmt, etc
    - systems, applications, data science, …
    - big SW firms, web services, startups, application domains… it's all Data!

You Have Enormous Potential

- **Graduate school**
  - DBs & Data Science now at many top PhD programs
  - Exciting crossover to other areas
- **Save the world!**
  - DataKind.org
  - CodeForAmerica.org
  - Data in the First Mile
    - http://db.cs.berkeley.edu/papers/cidr11-firstmile.pdf

"More, more I'm still not satisfied"

- **Grad classes @ Berkeley**
  - CS262A: grad level intro to DBMS and OS
  - CS286A: grad DB class (not offered next yr?)
  - DB, SysML, AMPLab seminars
- **Watch for 194's and MOOCs**
  - Data Science
  - Programming the Cloud

Parting Thoughts

- *Education is the ability to listen to almost anything without losing your temper or your self-confidence.*
  -Robert Frost
- *It is a miracle that curiosity survives formal education.*
  -Albert Einstein
- *Humility…yet pride and scorn;*
  Instinct and study; love and hate;
  Audacity…reverence. These must mate
  -Herman Melville, *Art*
- *The only thing one can do with good advice is to pass it on. It is never of any use to oneself.*
  -Oscar Wilde

Bye!

Backup material

Brewer's CAP Theorem

1. **(replica) Consistency**
2. **Availability**
3. **(tolerance of network) Partitions**

- **Basic formulation**
  - A system can only provide 2 of {C, A, P}
- **More practical formulation**
  - Partition ~= high latency (the CAL theorem?)
  - If you want High Availability and Low Latency:
    - Give up on Consistency.
  - If you want Consistency:
    - Give up on High Availability and Low Latency

Additional concern: finding replicas

- **Centralized directory**
  - A single node that maps keys to machines
- **Hierarchical directory**
  - Namespace is partitioned and sub-partitioned
  - Ask "up the tree"
  - DNS works this way
- **Hashing**
  - Static, well-known hash function is not very flexible.
    - More adaptive schemes based on "consistent hashing"
  - Truly distributed hashing: e.g. Chord
- **All of these should be performant and fault-tolerant too!**
  - Caching of locations is important throughout
  - Variety of trickiness here

Fault Tolerance and Load Balancing

- **Nodes may fail**
  - Can be recovered from replicas
  - In the interim, the replica nodes may be overloaded
    - Interesting questions on how to get *graceful degradation*
  - Some process needs to oversee this
    - Various solutions here, both centralized and distributed
- **Load balancing**
  - Even without failures, keys may have skewed access
  - Consistent hashing helps adapt placement while still providing directories