Project Voldemort

Jay Kreps

The Plan

LinkedIn

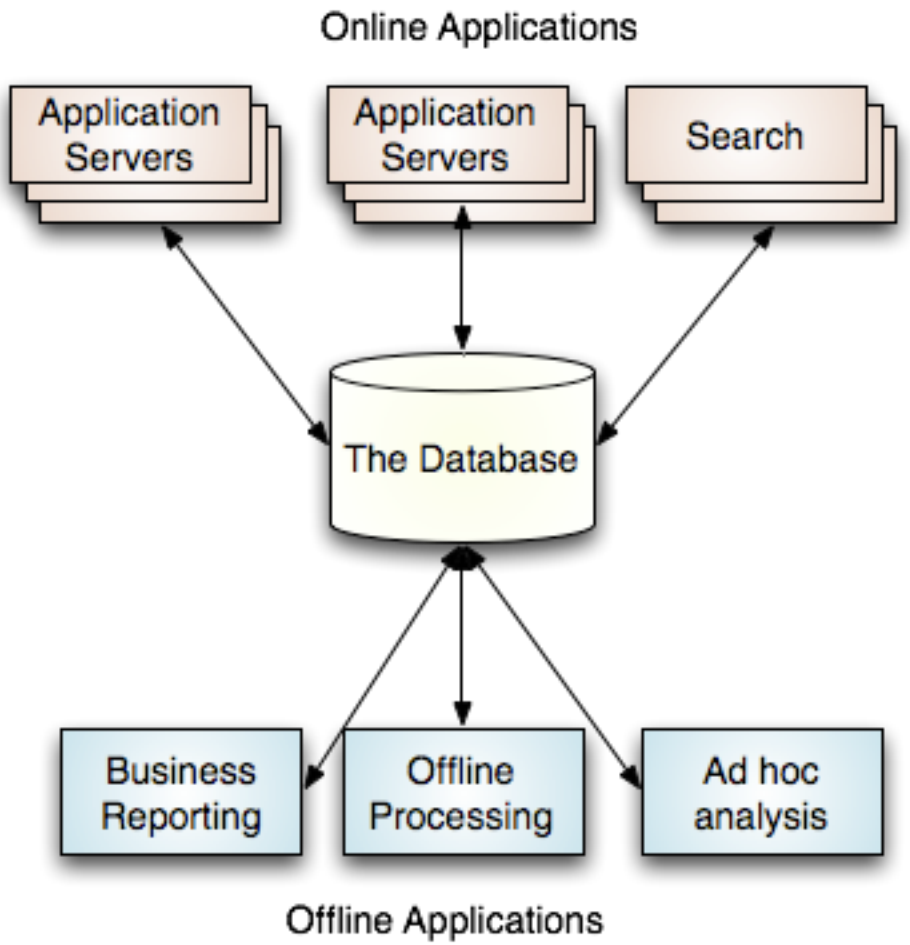
1. Motivation
2. Core Concepts
3. Implementation
4. In Practice
5. Results

Motivation

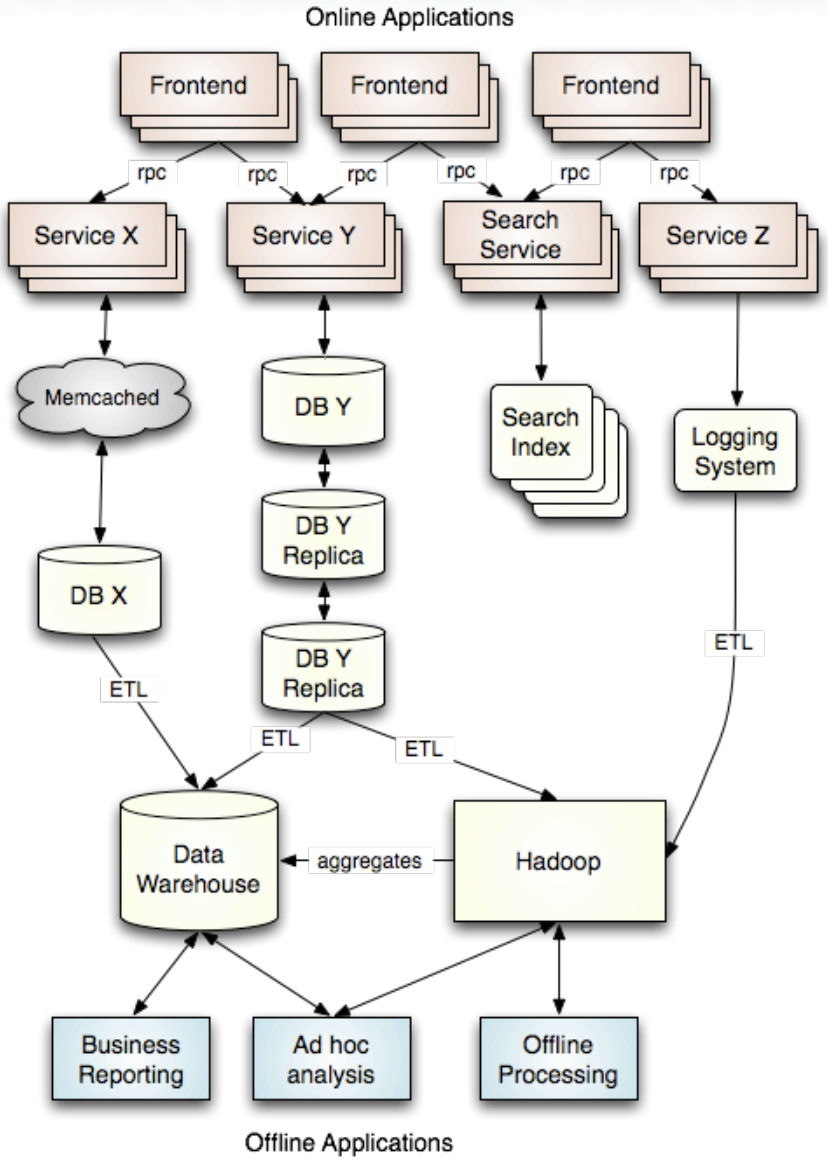
The Team

- LinkedIn's Search, Network, and Analytics Team
  - Project Voldemort
  - Search Infrastructure: Zoie, Bobo, etc
  - LinkedIn's Hadoop system
  - Recommendation Engine
  - Data intensive features
    - People you may know
    - Who's viewed my profile
    - User history service

The Idea of the Relational Database



The Reality of a Modern Web Site



Why did this happen?

- The internet centralizes computation
- Specialized systems are efficient (10-100x)
  - Search: Inverted index
  - Offline: Hadoop, Terradata, Oracle DWH
  - Memcached
  - In memory systems (social graph)
- Specialized system are scalable
- New data and problems
  - Graphs, sequences, and text

Services and Scale Break Relational DBs

LinkedIn

- No joins
- Lots of denormalization
- ORM is less helpful
- No constraints, triggers, etc
- Caching => key/value model
- Latency is key

Two Cheers For Relational Databases

LinkedIn

- The relational model is a triumph of computer science:
  - General
  - Concise
  - Well understood
- But then again:
  - SQL is a pain
  - Hard to build re-usable data structures
  - Don’t hide the memory hierarchy!
    Good: Filesystem API
    Bad: SQL, some RPCs

Other Considerations

LinkedIn

- Who is responsible for performance (engineers? DBA? site operations?)
- Can you do capacity planning?
- Can you simulate the problem early in the design phase?
- How do you do upgrades?
- Can you mock your database?

Some motivating factors

- This is a latency-oriented system
- Data set is large and persistent
    - Cannot be all in memory
- Performance considerations
    - Partition data
    - Delay writes
    - Eliminate network hops
- 80% of caching tiers are fixing problems that shouldn't exist
- Need control over system availability and data durability
    - Must replicate data on multiple machines
- Cost of scalability can't be too high

Inspired By Amazon Dynamo & Memcached

LinkedIn

- Amazon's Dynamo storage system
  - Works across data centers
  - Eventual consistency
  - Commodity hardware
  - Not too hard to build
- Memcached
  - Actually works
  - Really fast
  - Really simple
- Decisions:
  - Multiple reads/writes
  - Consistent hashing for data distribution
  - Key-Value model
  - Data versioning

Priorities

1. Performance and scalability
2. Actually works
3. Community
4. Data consistency
5. Flexible & Extensible
6. Everything else

Why Is This Hard?

LinkedIn

- Failures in a distributed system are much more complicated
  - A can talk to B does not imply B can talk to A
  - A can talk to B does not imply C can talk to B
- Getting a consistent view of the cluster is as hard as getting a consistent view of the data
- Nodes will fail and come back to life with stale data
- I/O has high request latency variance
- I/O on commodity disks is even worse
- Intermittent failures are common
- User must be isolated from these problems
- There are fundamental trade-offs between availability and consistency

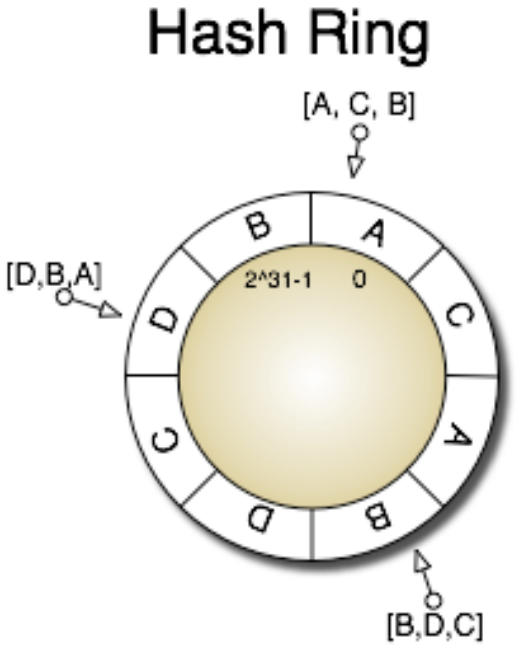
Core Concepts

Core Concepts - I

- ACID
  - Great for single centralized server.
- CAP Theorem
  - Consistency (Strict), Availability , Partition Tolerance
  - Impossible to achieve all three at same time in distributed platform
  - Can choose 2 out of 3
  - Dynamo chooses High Availability and Partition Tolerance
    - by sacrificing Strict Consistency to Eventual consistency
- Consistency Models
  - Strict consistency
    - 2 Phase Commits
    - PAXOS : distributed algorithm to ensure quorum for consistency
  - Eventual consistency
    - Different nodes can have different views of value
    - In a steady state system will return last written value.
    - BUT Can have much strong guarantees.

Core Concept - II

- Consistent Hashing
- Key space is Partitioned
  - Many small partitions
- Partitions never change
  - Partitions ownership can change
- Replication
  - Each partition is stored by ‘N’ nodes
- Node Failures
  - Transient (short term)
  - Long term
    - Needs faster bootstrapping



Core Concept - III

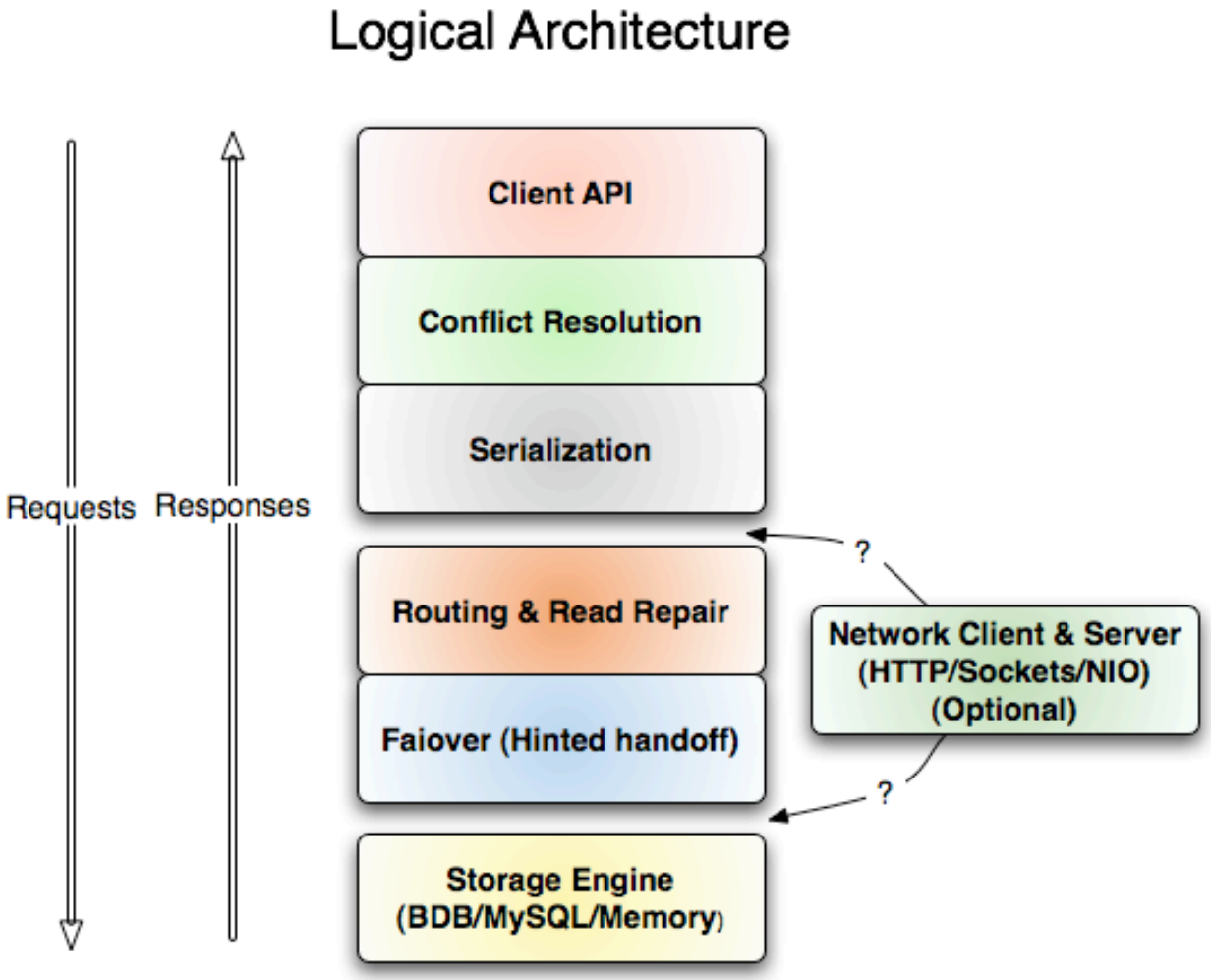
- N - The replication factor
- R - The number of blocking reads
- W - The number of blocking writes
- If $R+W > N$
  - then we have a quorum-like algorithm
  - Guarantees that we will read latest writes OR fail
- R, W, N can be tuned for different use cases
  - W = 1, Highly available writes
  - R = 1, Read intensive workloads
  - Knobs to tune performance, durability and availability

Core Concepts - IV

- Vector Clock [Lamport] provides way to order events in a distributed system.
- A vector clock is a tuple {t1 , t2 , ..., tn } of counters.
- Each value update has a master node
  - When data is written with master node i, it increments ti.
  - All the replicas will receive the same version
  - Helps resolving consistency between writes on multiple replicas
- If you get network partitions
  - You can have a case where two vector clocks are not comparable.
  - In this case Voldemort returns both values to clients for conflict resolution

Implementation

Voldemort Design



Client API

- Data is organized into “stores”, i.e. tables
- Key-value only
  - But values can be arbitrarily rich or complex
    - Maps, lists, nested combinations …
- Four operations
  - PUT (K, V)
  - GET (K)
  - MULTI-GET (Keys),
  - DELETE (K, Version)
  - No Range Scans

Versioning & Conflict Resolution

- Eventual Consistency allows multiple versions of value
  - Need a way to understand which value is latest
  - Need a way to say values are not comparable
- Solutions
  - Timestamp
  - Vector clocks
    - Provides global ordering.
    - No locking or blocking necessary

Serialization

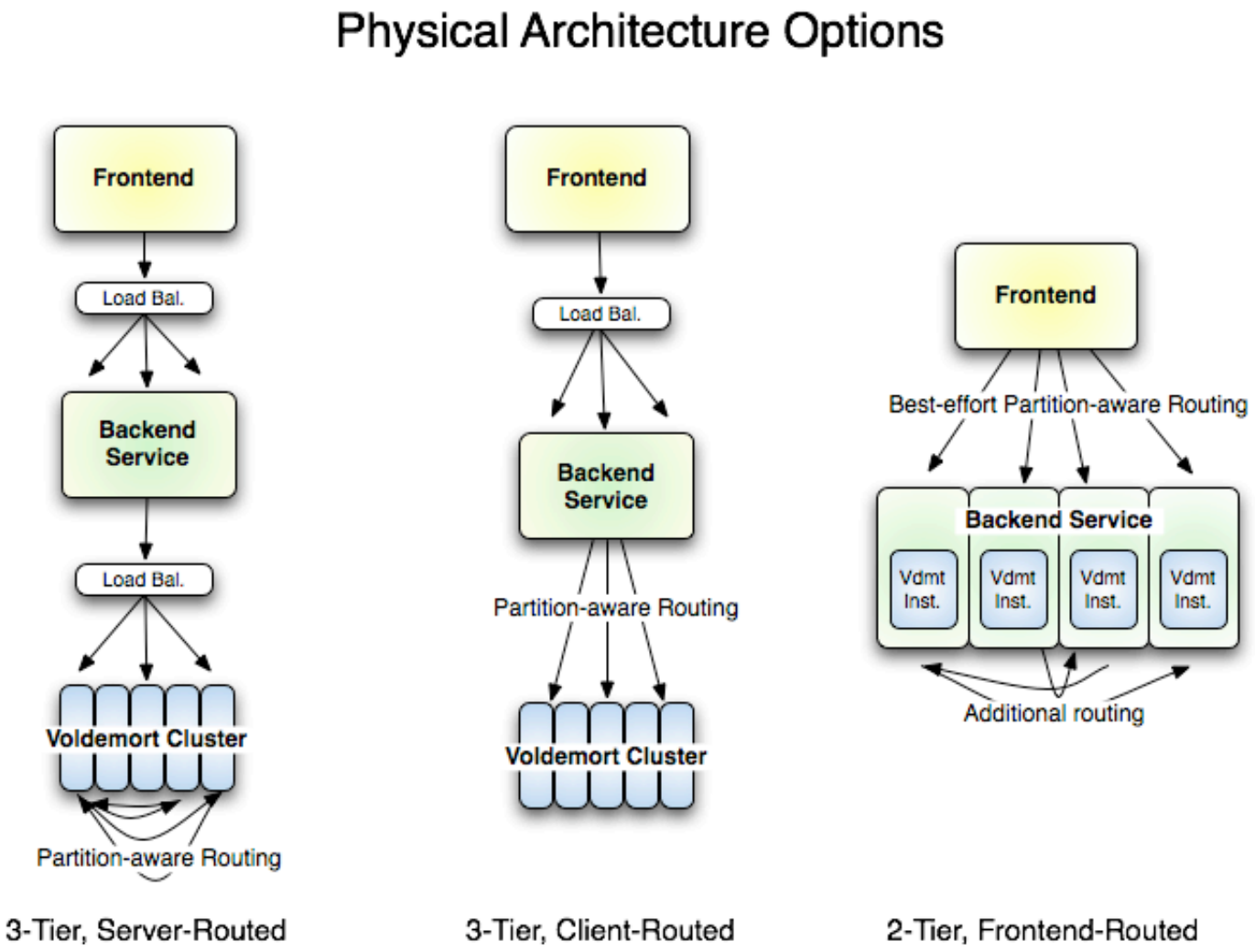
LinkedIn

- Really important
  - Few Considerations
    - Schema free?
    - Backward/Forward compatible
    - Real life data structures
    - Bytes <=> objects <=> strings?
    - Size (No XML)
- Many ways to do it -- we allow anything
  - Compressed JSON, Protocol Buffers, Thrift, Voldemort custom serialization

Routing

- Routing layer hides lot of complexity
  - Hashing schema
  - Replication (N, R , W)
  - Failures
  - Read-Repair (online repair mechanism)
  - Hinted Handoff (Long term recovery mechanism)
- Easy to add domain specific strategies
  - E.g. only do synchronous operations on nodes in the local data center
- Client Side / Server Side / Hybrid

Voldemort Physical Deployment



Routing With Failures

- Failure Detection
  - Requirements
    - Need to be very very fast
    - View of server state may be inconsistent
      - A can talk to B but C cannot
      - A can talk to C , B can talk to A but not to C
  - Currently done by routing layer (request timeouts)
    - Periodically retries failed nodes.
    - All requests must have hard SLAs
  - Other possible solutions
    - Central server
    - Gossip protocol
    - Need to look more into this.

Repair Mechanism

- Read Repair
  - Online repair mechanism
    - Routing client receives values from multiple node
    - Notify a node if you see an old value
    - Only works for keys which are read after failures
- Hinted Handoff
  - If a write fails write it to any random node
  - Just mark the write as a special write
  - Each node periodically tries to get rid of all special entries
- Bootstrapping mechanism (We don't have it yet)
  - If a node was down for long time
    - Hinted handoff can generate ton of traffic
    - Need a better way to bootstrap and clear hinted handoff tables

Network Layer

LinkedIn

- Network is the major bottleneck in many uses
- Client performance turns out to be harder than server (client must wait!)
    - Lots of issue with socket buffer size/socket pool
- Server is also a Client
- Two implementations
    - HTTP + servlet container
    - Simple socket protocol + custom server
- HTTP server is great, but http client is 5-10X slower
- Socket protocol is what we use in production
- Recently added a non-blocking version of the server

Persistence

- Single machine key-value storage is a commodity
- Plugins are better than tying yourself to a single strategy
  - Different use cases
    - optimize reads
    - optimize writes
    - large vs small values
  - SSDs may completely change this layer
  - Better filesystems may completely change this layer
- Couple of different options
  - BDB, MySQL and mmap'd file implementations
  - Berkeley DBs most popular
  - In memory plugin for testing
- Btrees are still the best all-purpose structure
- No flush on write is a huge, huge win

In Practice

LinkedIn problems we wanted to solve

- Application Examples
  - People You May Know
  - Item-Item Recommendations
  - Member and Company Derived Data
  - User's network statistics
  - Who Viewed My Profile?
  - Abuse detection
  - User's History Service
  - Relevance data
  - Crawler detection
  - Many others have come up since
- Some data is batch computed and served as read only
- Some data is very high write load
- Latency is key

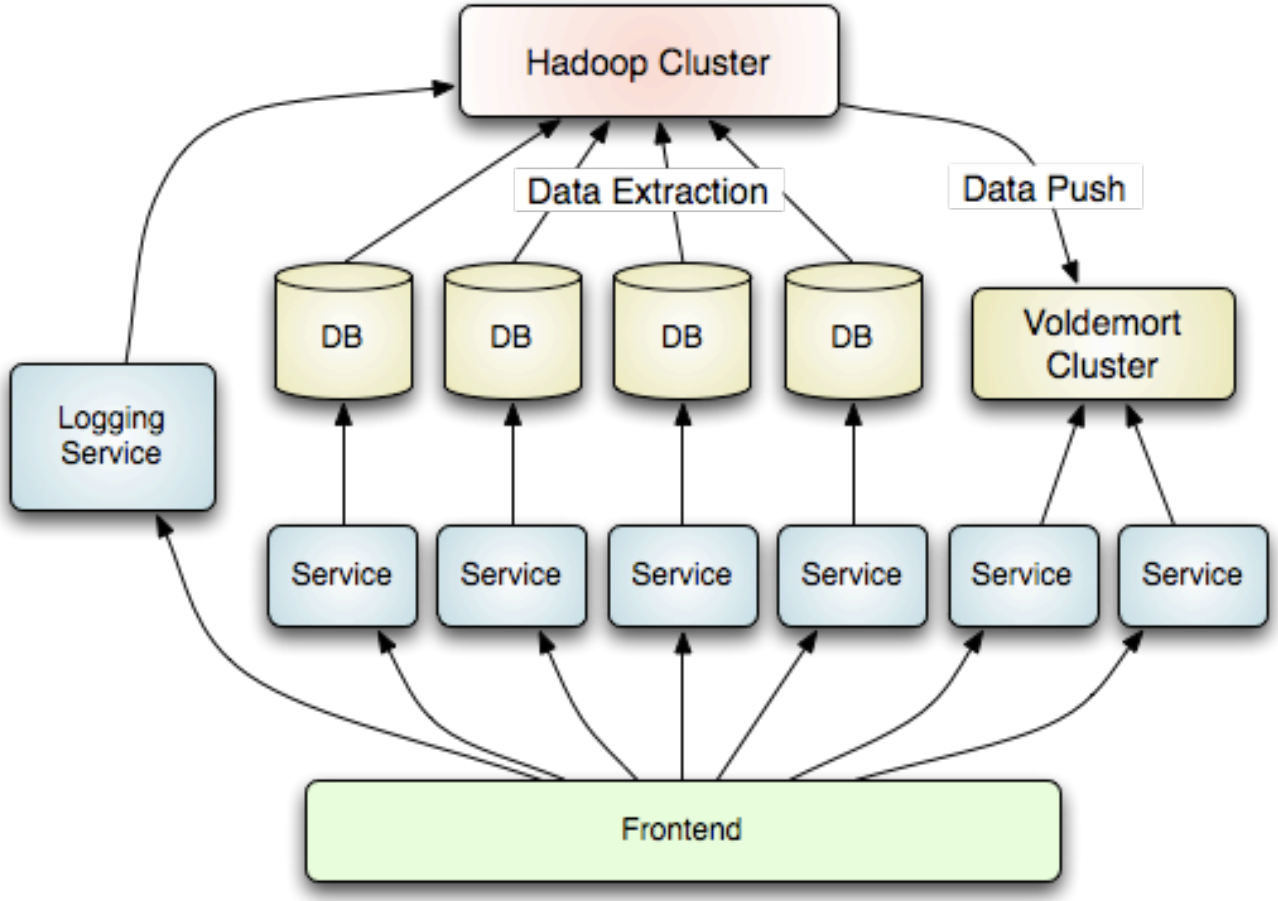
Key-Value Design Example

- How to build a fast, scalable comment system?
- One approach
  - (post_id, page) => [comment_id_1, comment_id_2, …]
  - comment_id => comment_body
- GET comment_ids by post and page
- MULTIGET comment bodies
- Threaded, paginated comments left as an exercise ☺

Hadoop and Voldemort sitting in a tree…

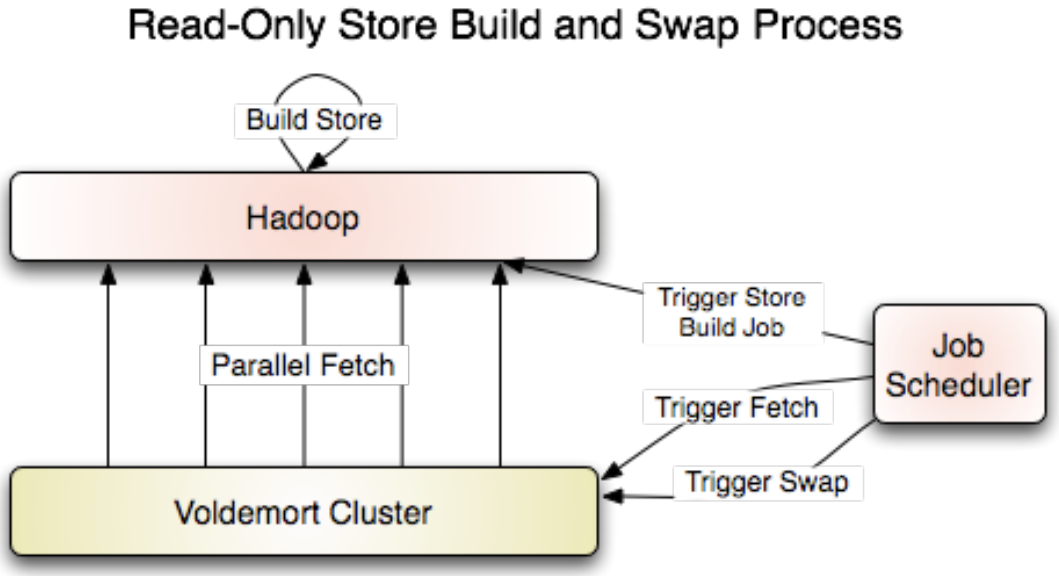
- Hadoop can generate a lot of data
- Bottleneck 1: Getting the data out of hadoop
- Bottleneck 2: Transfer to DB
- Bottleneck 3: Index building
- We had a critical process where this process took a DBA a week to run!
- Index building is a batch operation

20,000 Foot View Of The Data Cycle



Read-only storage engine

- Throughput vs. Latency
- Index building done in Hadoop
- Fully parallel transfer
- Very efficient on-disk structure
- Heavy reliance on OS pagecache
- Rollback!



Voldemort At LinkedIn

- 4 Clusters, 4 teams
  - Wide variety of data sizes, clients, needs
- My team:
  - 12 machines
  - Nice servers
  - 500M operations/day
  - ~4 billion events in 10 stores (one per event type)
  - Peak load > 10k operations / second
- Other teams: news article data, email related data, UI settings

Results

Some performance numbers

- Production stats
  - Median: 0.1 ms
  - 99.9 percentile GET: 3 ms
- Single node max throughput (1 client node, 1 server node):
  - 19,384 reads/sec
  - 16,559 writes/sec
- These numbers are for mostly in-memory problems

Glaring Weaknesses

LinkedIn

- Not nearly enough documentation
- No online cluster expansion (without reduced guarantees)
- Need more clients in other languages (Java, Python, Ruby, and C++ currently)
- Better tools for cluster-wide control and monitoring

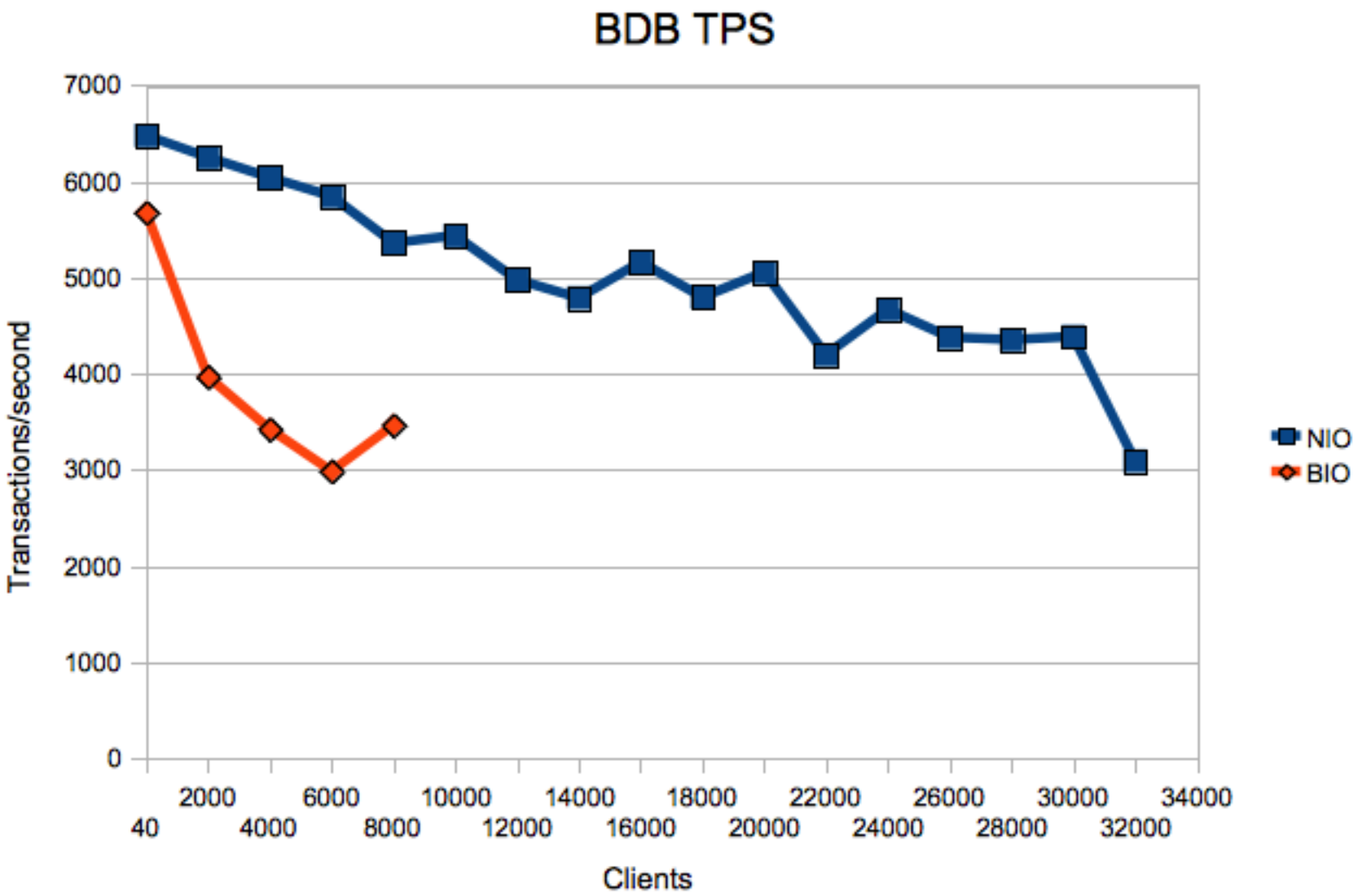
State of the Project

- Active mailing list
- 4-5 regular committers outside LinkedIn
- Lots of contributors
- Equal contribution from in and out of LinkedIn
- Project basics
  - IRC
  - Some documentation
  - Lots more to do
- > 300 unit tests that run on every checkin (and pass)
- Pretty clean code
- Moved to GitHub (by popular demand)
- Production usage at a half dozen companies
- Not just a LinkedIn project anymore
- But LinkedIn is really committed to it (and we are hiring to work on it)

Some new & upcoming things

- New
  - Python, Ruby clients
  - Non-blocking socket server
  - Alpha round on online cluster expansion
  - Read-only store and Hadoop integration
  - Improved monitoring stats
  - Distributed testing infrastructure
  - Compression
- Future
  - Publish/Subscribe model to track changes
  - Improved failure detection

Socket Server Scalability



Testing and releases

- Testing “in the cloud”
  - Distributed systems have complex failure scenarios
  - A storage system, above all, must be stable
  - Automated testing allows rapid iteration while maintaining confidence in systems’ correctness and stability
- EC2-based testing framework
  - Tests are invoked programmatically
  - Contributed by Kirk True
  - Adaptable to other cloud hosting providers
- Regular releases for new features and bugs
- Trunk stays stable

Shameless promotion

- Check it out: project-voldemort.com
- We love getting patches.
- We kind of love getting bug reports.
- LinkedIn is hiring, so you can work on this full time.
    - Email me if interested
    - jkreps@linkedin.com

The End