Trading latency for freshness in storage systems

Thesis proposal

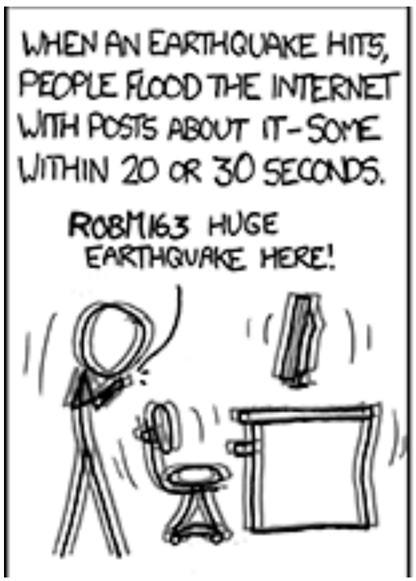
Jim Cipar

PARALLEL DATA LABORATORY

Carnegie Mellon University

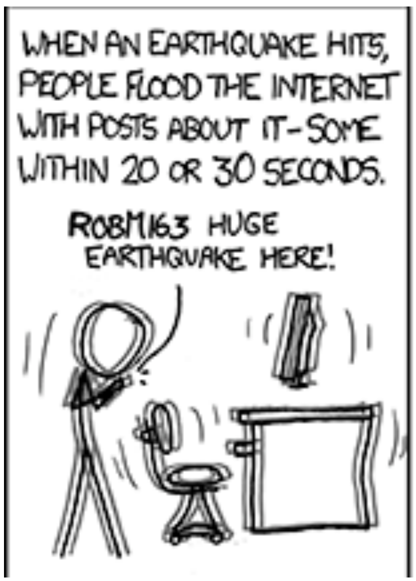
Example application

- High bandwidth stream of Tweets
  - Many thousands per second
  - 200 million per day, up to 20k per second



Example application

- High bandwidth stream of Tweets
  - Many thousands per second
  - 200 million per day, up to 20k per second



- Queries accept different freshness levels
  - Freshest: USGS Twitter Earthquake Detector
  - Fresh: Hot news in last 10 minutes
  - Stale: social network graph analysis
- Freshness depends on *query* not *data*

Applications and freshness

Freshness / Domain	Seconds	Minutes	Hours+
Retail	Real-time coupons, targeted ads	Just-in-time inventory	Product search, earnings reports
Enterprise information management	Infected machine identification	File-based policy validation	E-discovery requests, search
Transportation	Emergency response	Real-time traffic maps	Traffic engineering, route planning

Class of analytical applications

- **Performance**
  - Continuous high-throughput updates
  - “Big” analytical queries
- **Freshness**
  - Varying freshness requirements for queries
  - Freshness varies by query, not data set

Thesis statement

Storage systems can and should provide for per-query configuration of the tradeoff between data freshness and query efficiency and latency.

Overview

- Introduction and thesis statement
- Two examples of freshness vs. performance
  - LazyBase - data collection and analytics
  - LazyTables – shared data for machine learning
- Related work
- Status and plan

LazyBase design goals

- Distributed database
- Continuous, high-throughput ingest
- Queries specify freshness requirement

LazyBase limitations

- Only supports **observational data**
  - Transactions are read-only or write-only
  - No read-modify-write
  - Not online transaction processing
- Not (currently) targeting really huge scale
  - 10s of servers, not 1000s
  - Not everyone is a Google (or a Facebook…)

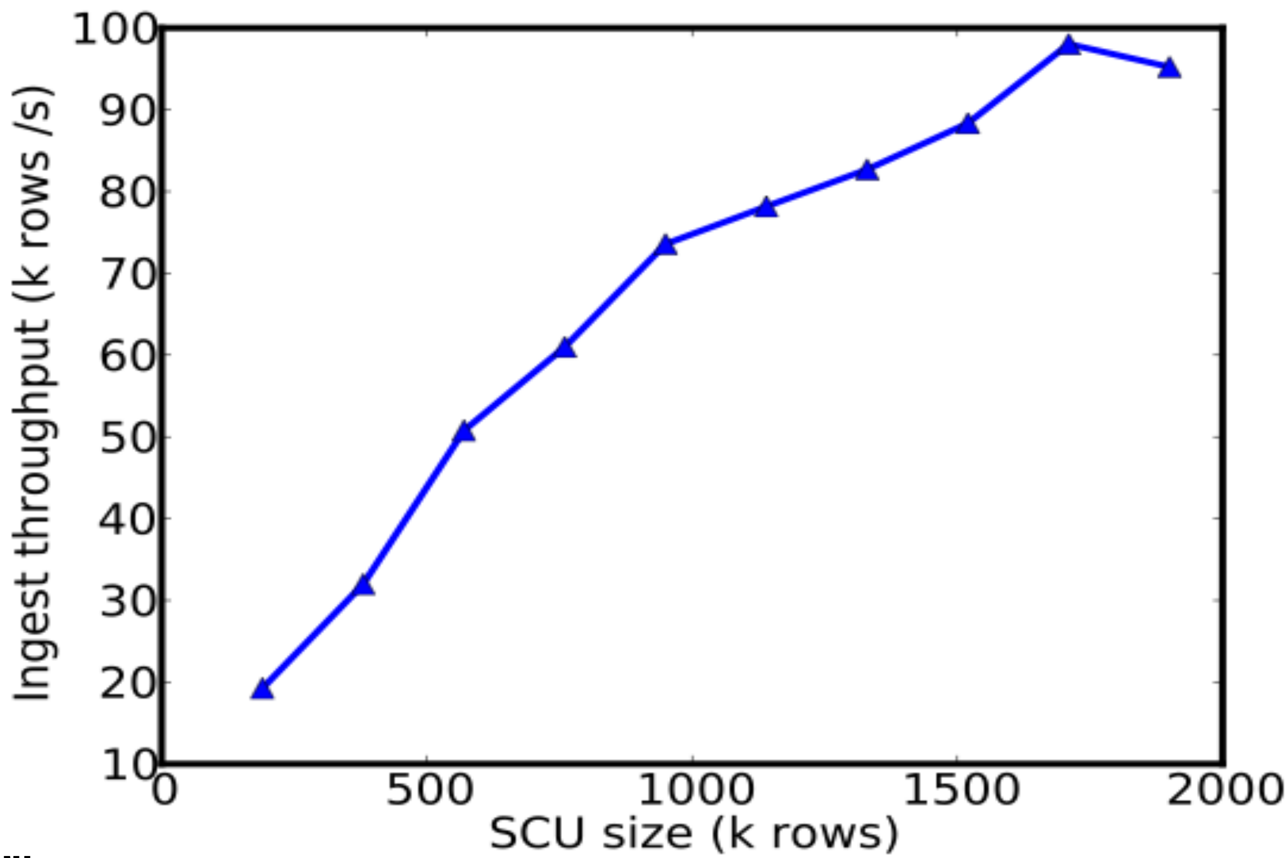
High throughput ingest

- All servers receive updates continously
  - Server receiving data creates large batches

- **Batching provides high performance updates**
  - Common technique for throughput
  - E.g. bulk loading of data in data warehouse

Batching for performance

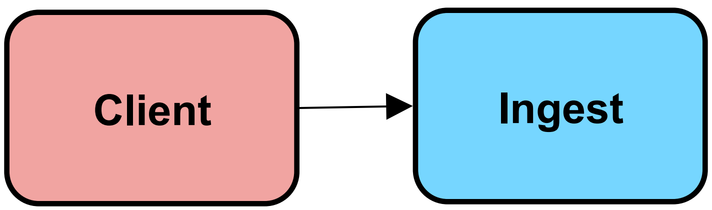
Large batches of updates increase throughput



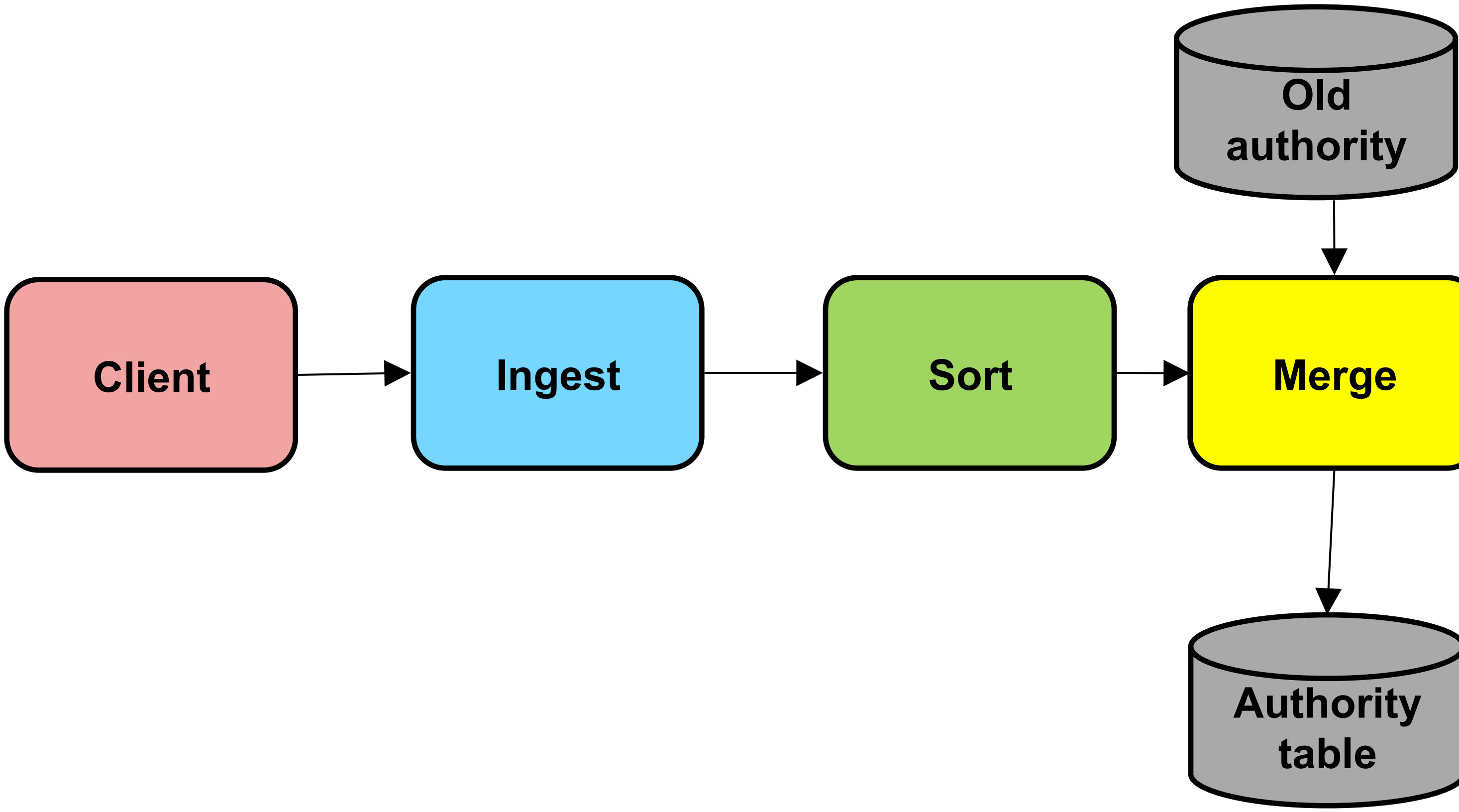
LazyBase design

- LazyBase is a distributed database
  - Commodity servers (e.g. 8 CPU cores, 16 GB RAM)
  - Can use direct attached storage
- Each server runs:
  - General purpose worker process
  - Ingest server that accepts client requests
  - Query agent that processes query requests
- Logically LazyBase is a pipeline
  - Each pipeline stage can be run on any worker
  - Single stage may be parallelized on multiple workers

Pipelined data flow



Pipelined data flow



Problem with batching: latency

- Batching trades update latency for throughput
  - Large batches → database is very stale
  - Very large batches/busy system → could be hours old

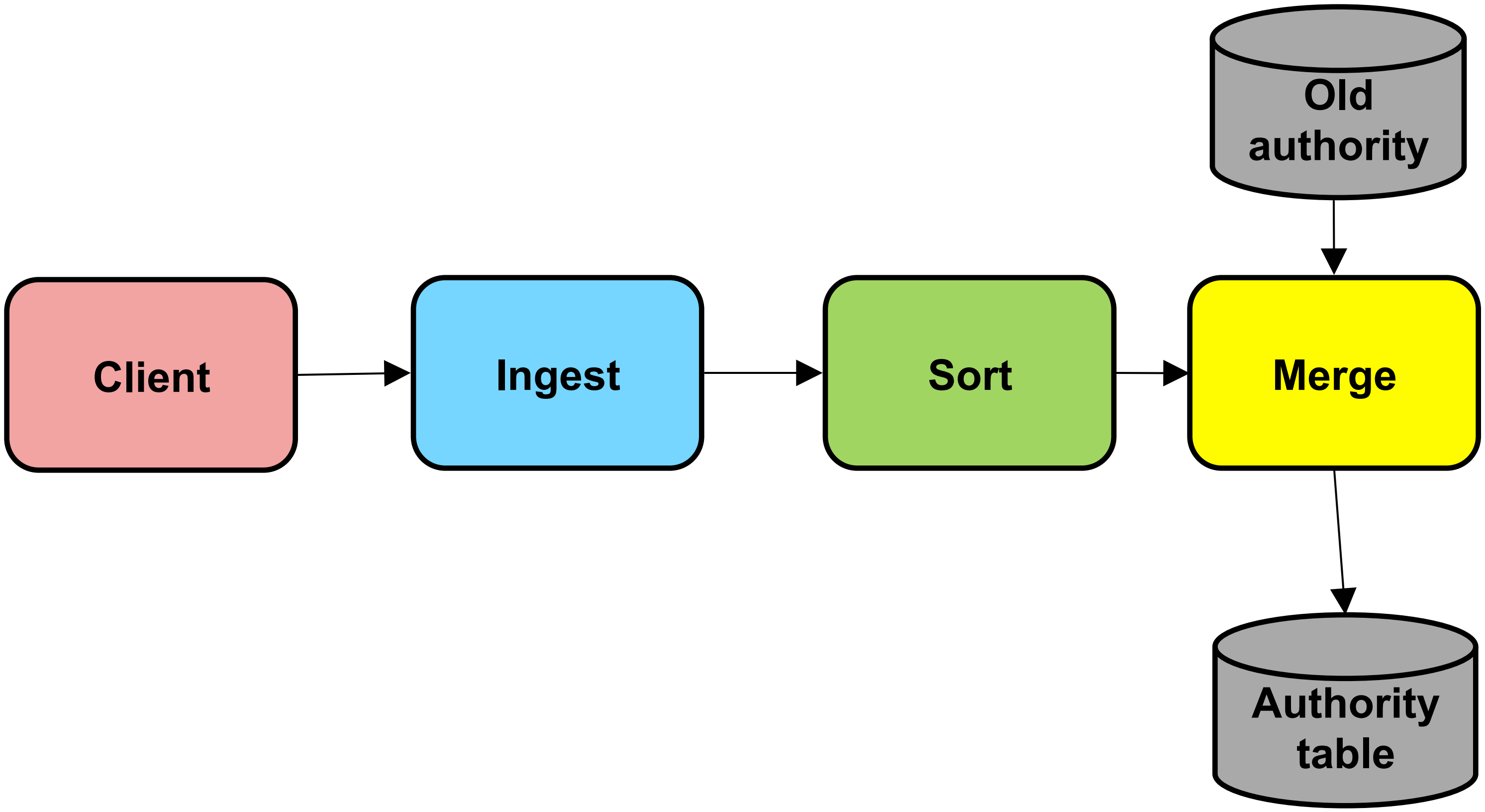
- OK for some queries, bad for others

Queries look at intermediate data

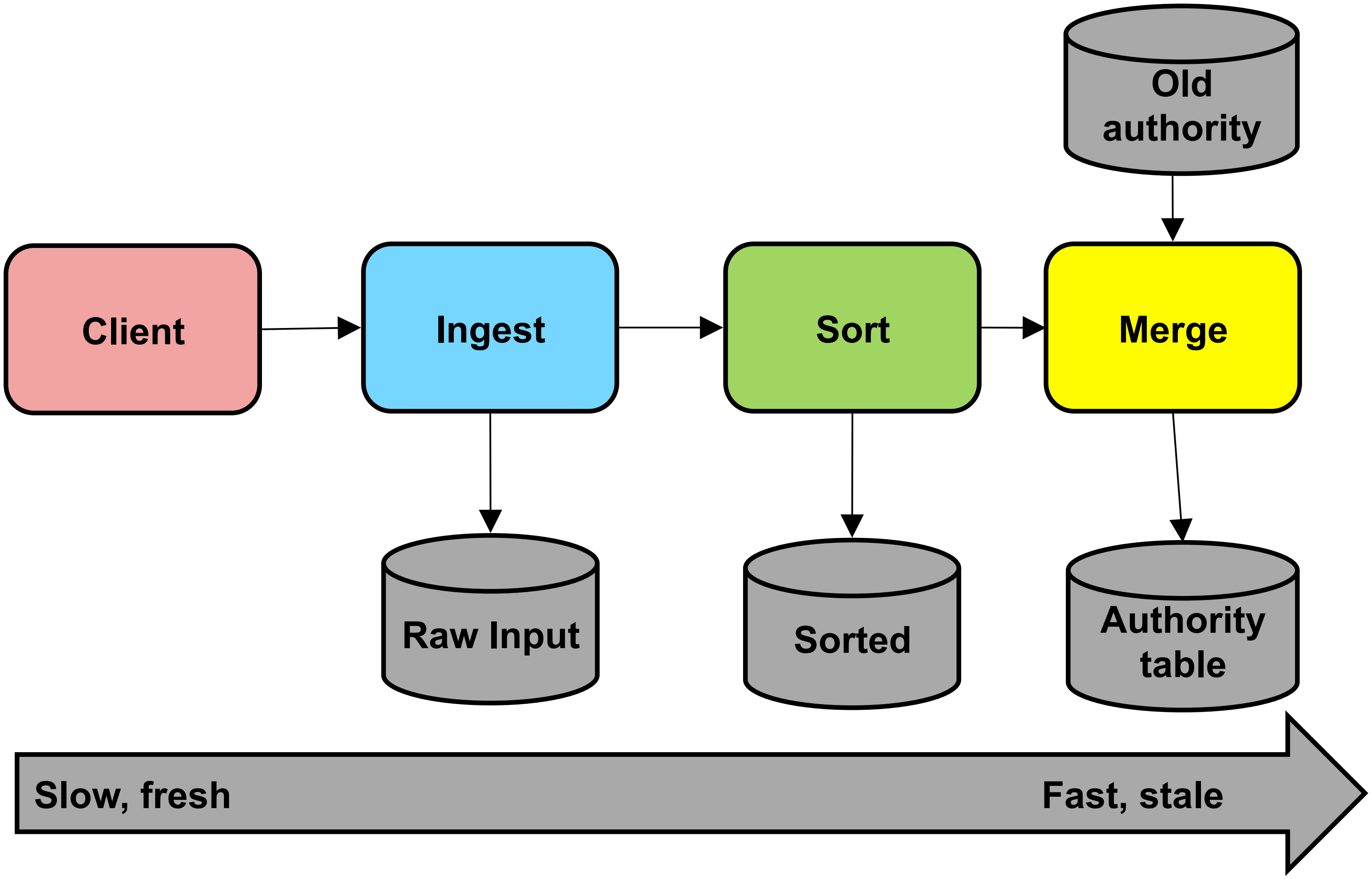
As updates are processed through pipeline, they become progressively “easier” to query.

We can use this to trade query latency for freshness.

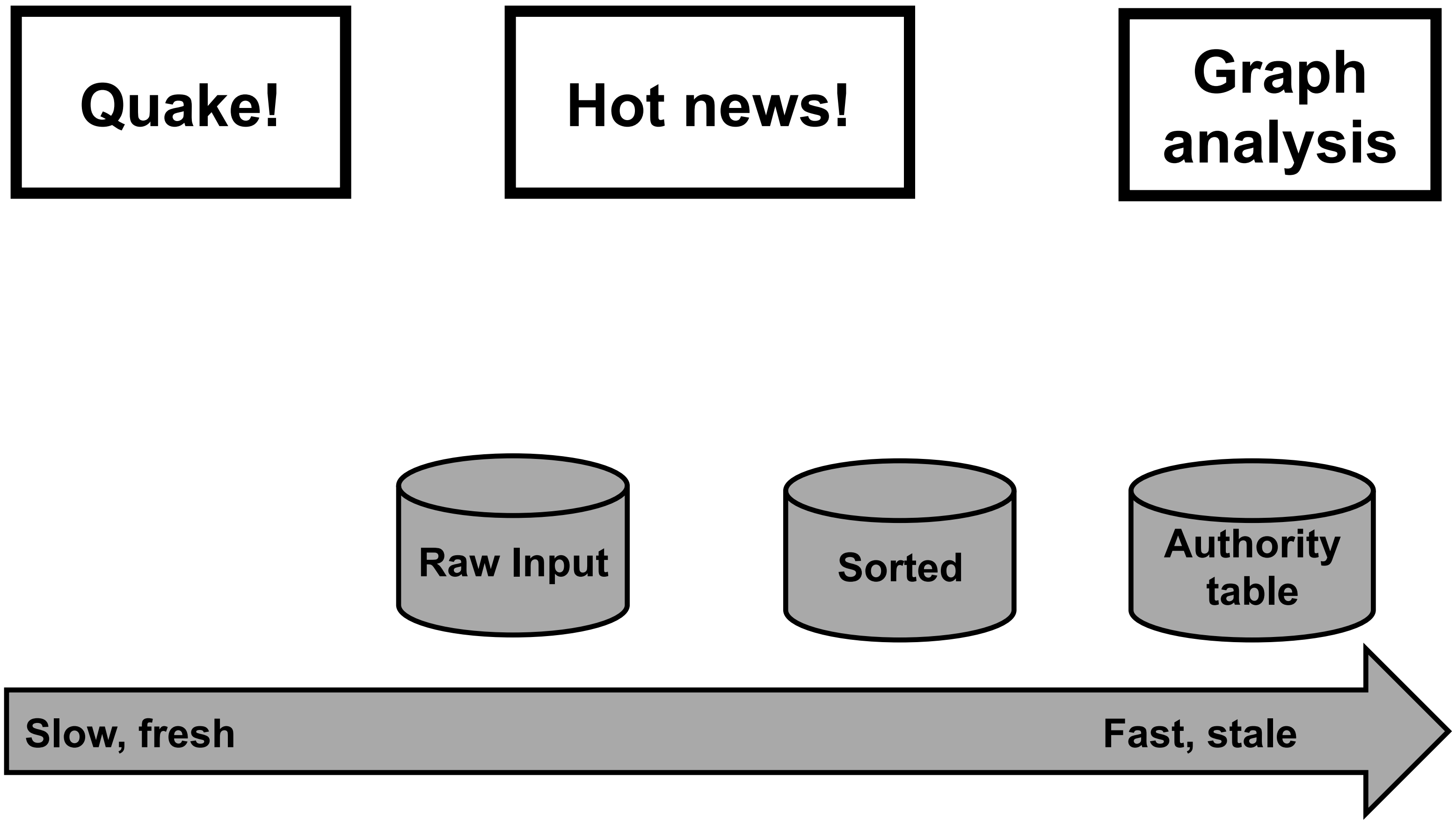
Query freshness



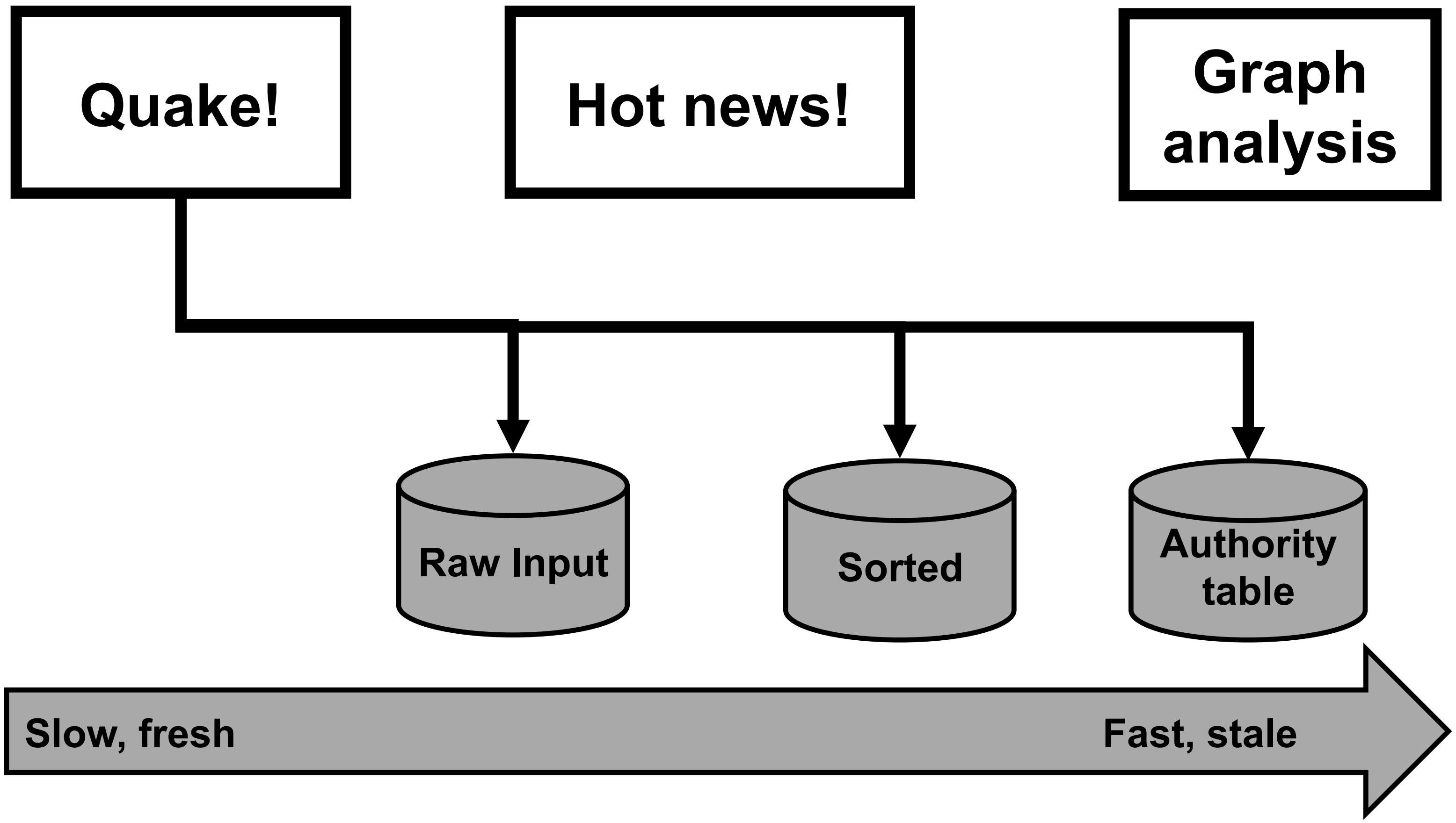
Query freshness



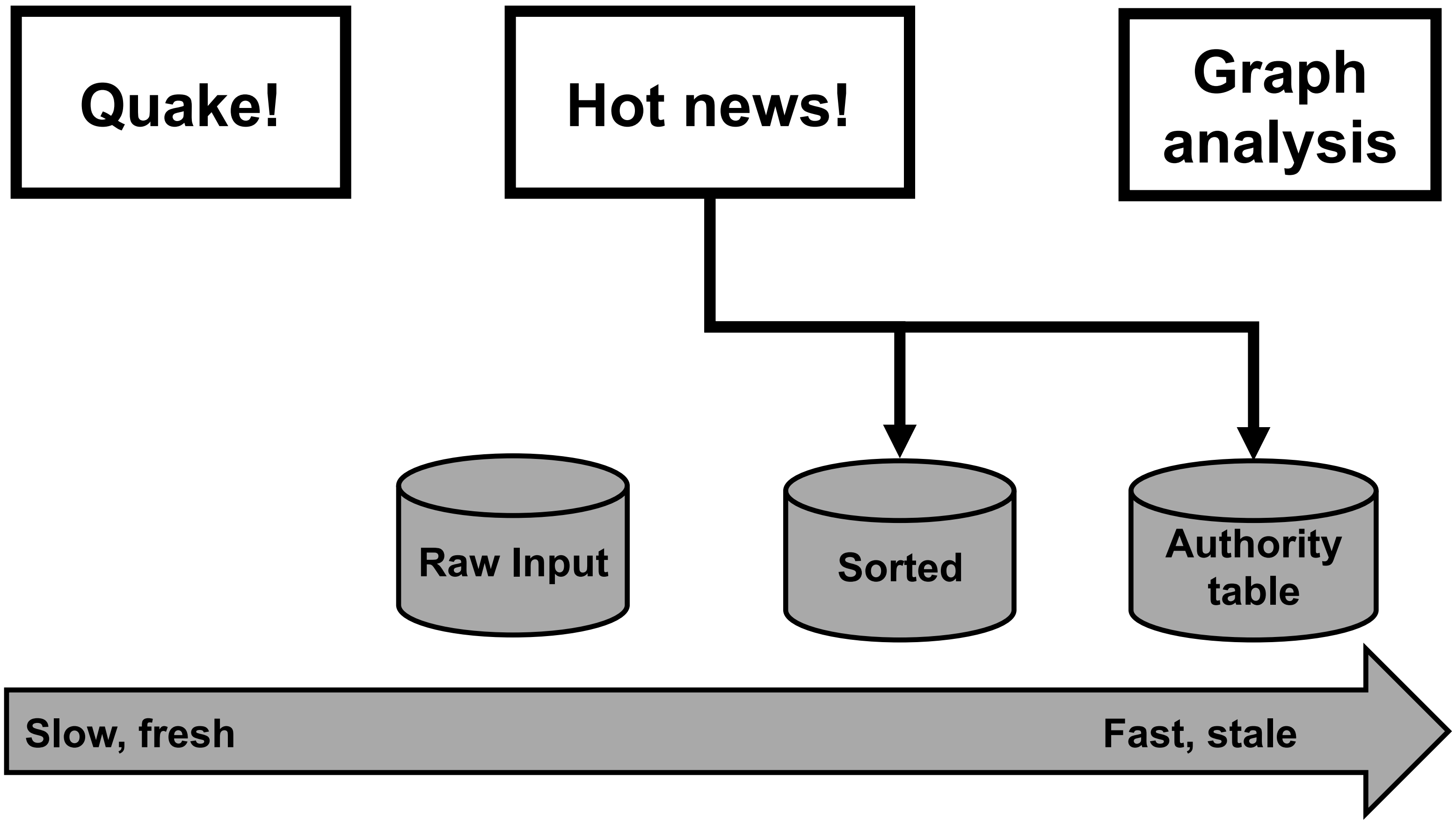
Query freshness



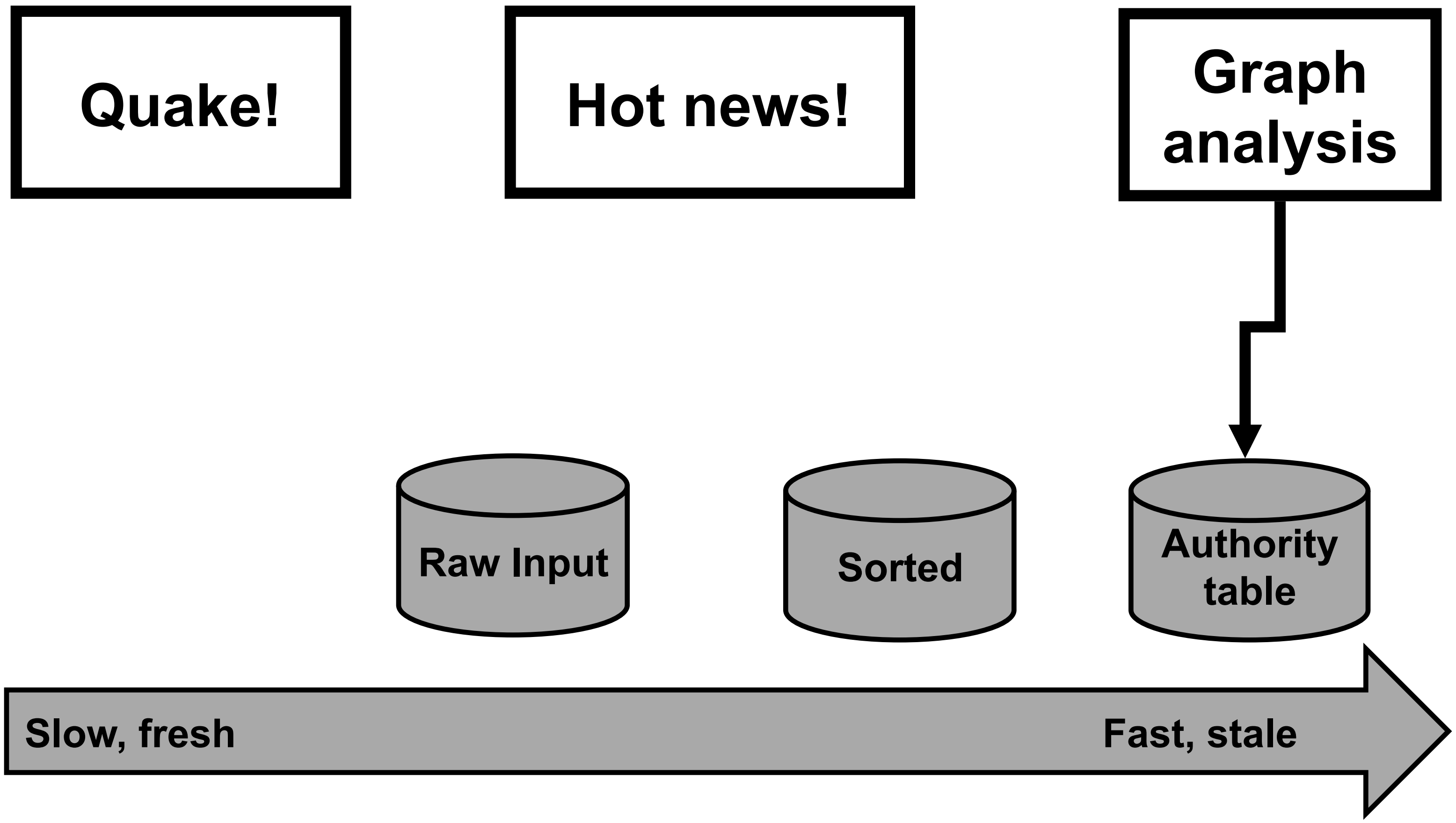
Query freshness



Query freshness



Query freshness



Query interface

- User issues high-level queries
  - Programatically or like a limited subset of SQL
  - **Specifies freshness**

```
SELECT COUNT(*) FROM tweets
WHERE user = “jcipar”
FRESHNESS 30;
```

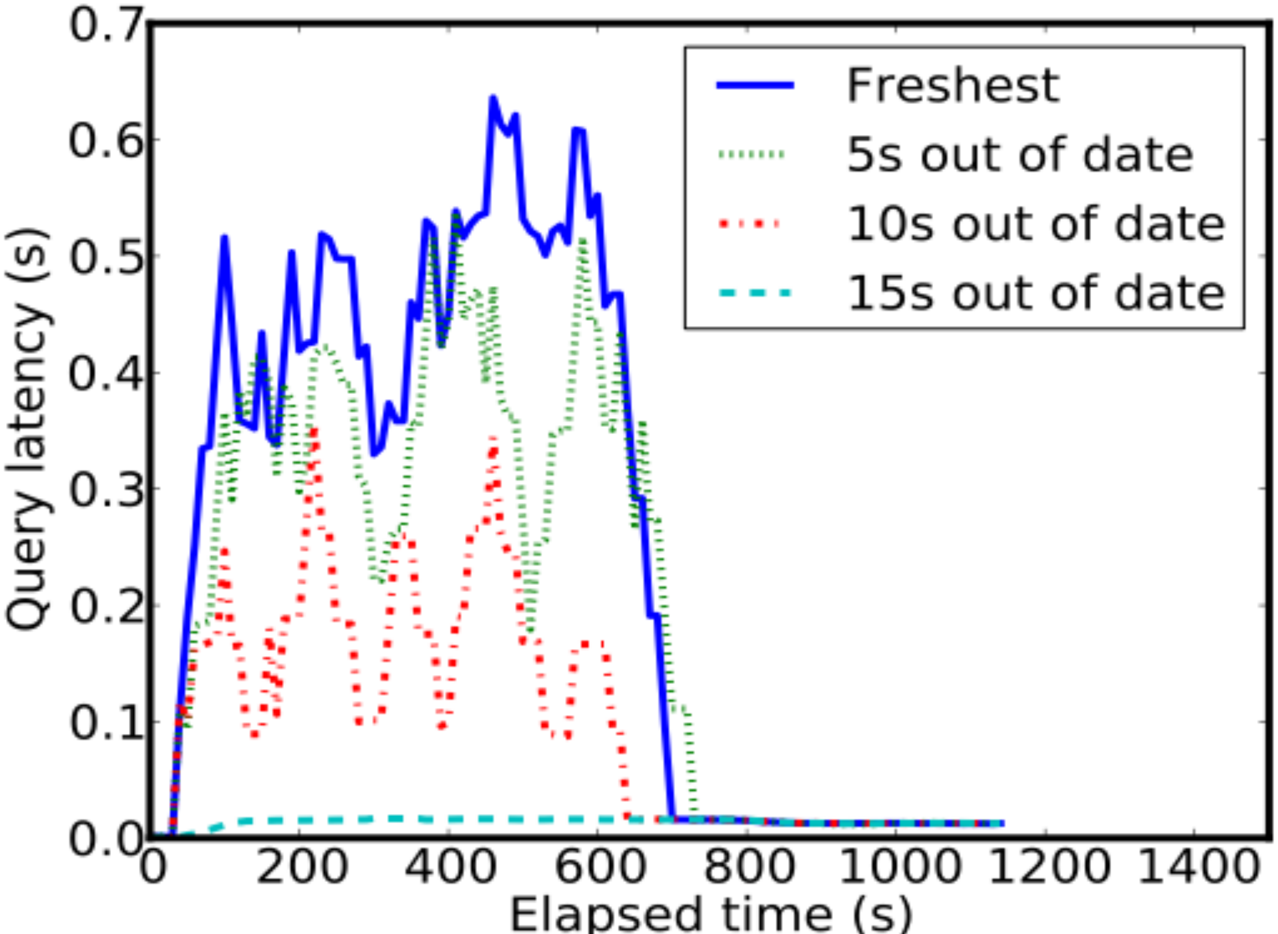
- Client library handles all the “dirty work”

Testing freshness requirements

- Upload data continuously for 10 min
  - Uploaded slowly, difficult to back up pipeline
- During upload, test query latency
  - 4 different freshness requirements

Query latency/freshness

Queries allowing staler results return faster



Experiments to show…

- Importance of batching
- Freshness/performance tradeoff
- Throughput and scalability of updates
- Performance for queries
  - Both “small” and “big” queries

Experiments to show…

- Importance of batching
- Freshness/performance tradeoff
- Throughput and scalability of updates
- Performance for queries
  - Both "small" and "big" queries

Experiments to show…

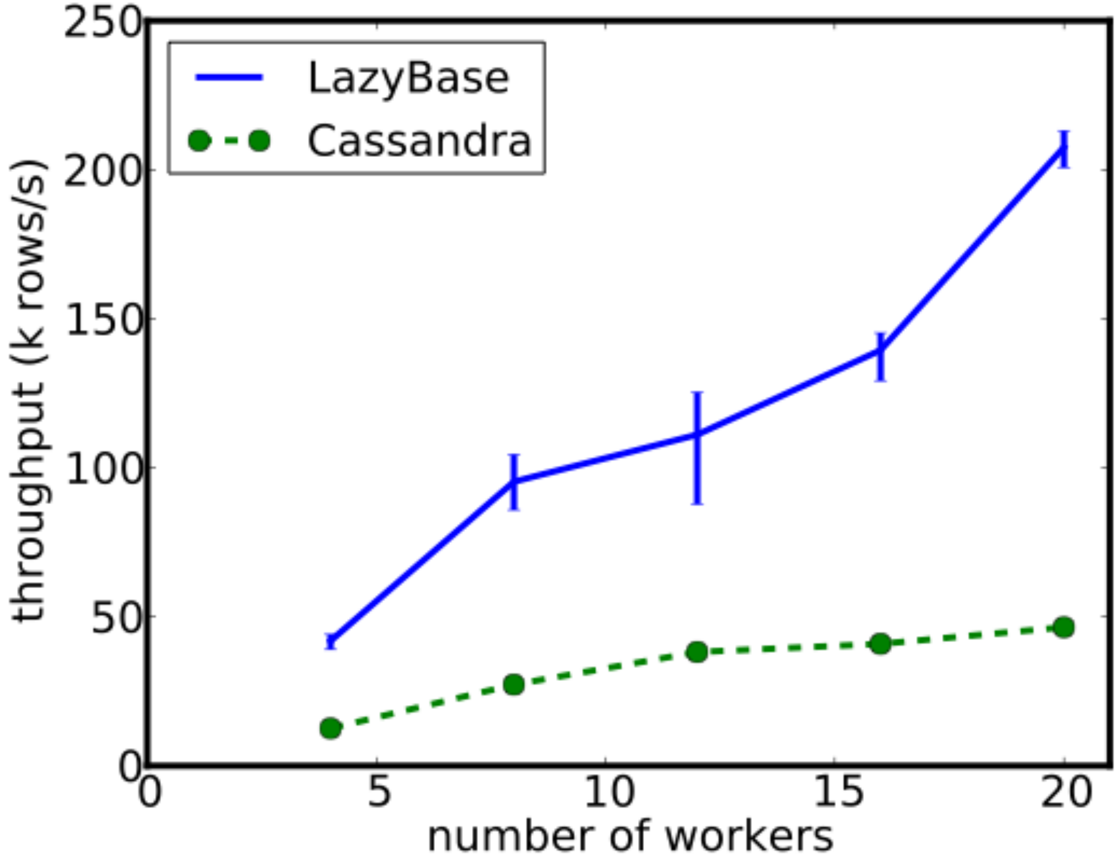
- Importance of batching
- Freshness/performance tradeoff
- **Throughput and scalability of ingest**
- Performance for queries
  - Both “small” and “big” queries
- Consistency relative to Cassandra
- Freshness relative to Cassandra

Ingest scalability experiment

- Measured time to ingest entire data set
- Uploaded in parallel from 20 servers
- Varied number of worker processes

Ingest scalability results

LazyBase scales effectively up to 20 servers
Efficiency is ~4x better than Cassandra



Experiments to show…

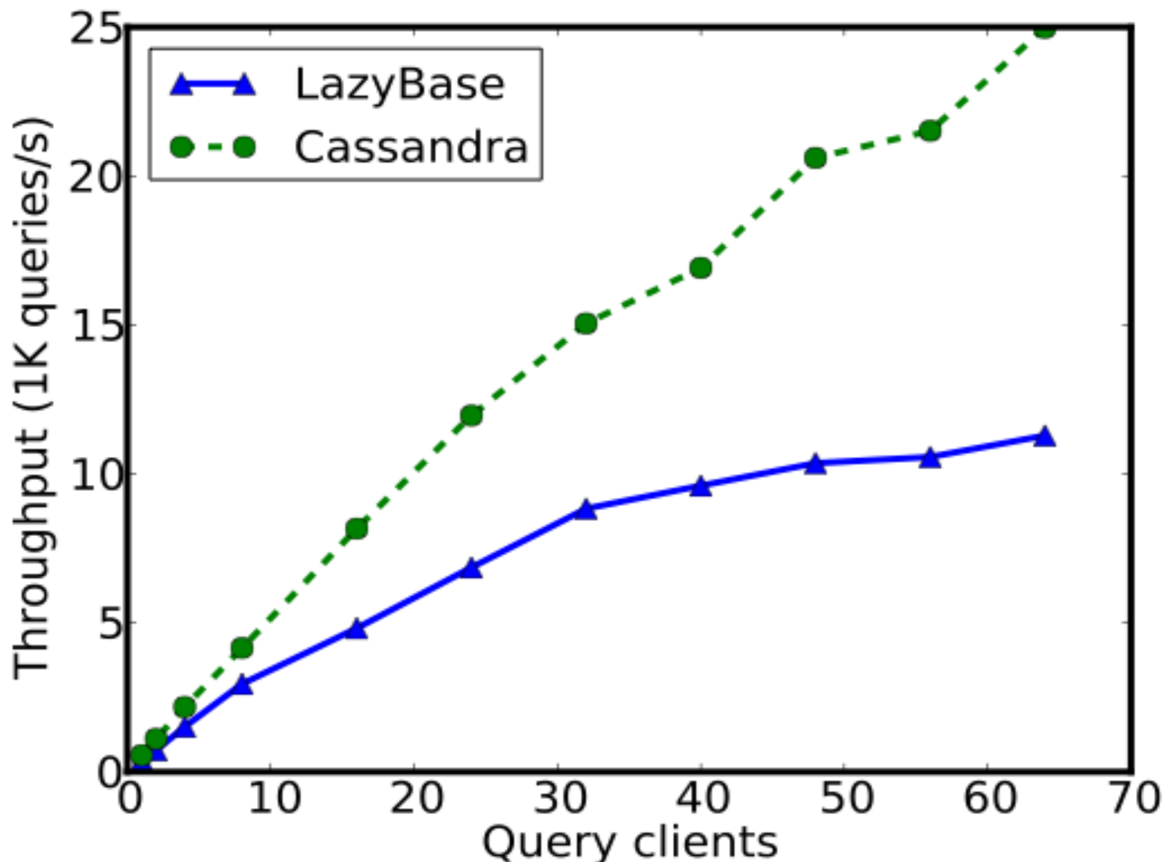
- Importance of batching
- Freshness/performance tradeoff
- Throughput and scalability of ingest
- **Performance for queries**
  - **Both “small” and “big” queries**

Query experiments

- Test performance of fastest queries
  - Access only authority table
- Two types of queries: point and range
  - Point queries get single tweet by ID
  - Range queries get list of valid tweet IDs in range
    - Range size chosen to return ~0.1% of all IDs
- Cassandra range queries used get_slice
  - Actual range queries discouraged

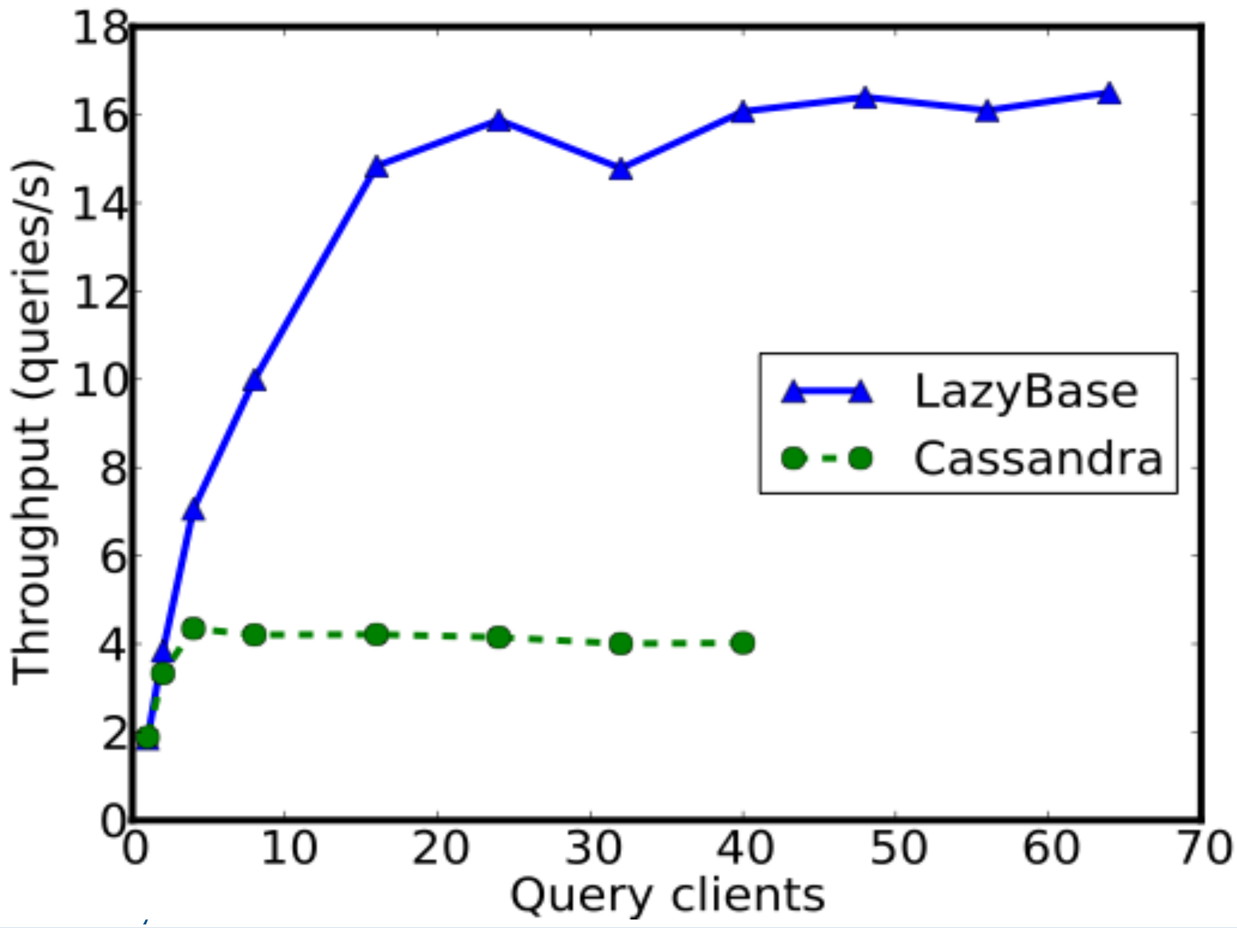
Point query throughput

Queries scale to multiple clients
Raw performance suffers due to on-disk format



Range query throughput

Range query performance ~4x Cassandra



Overview

- Introduction and thesis statement
- Two examples of freshness vs. performance
  - LazyBase - data collection and analytics
  - LazyTables – shared data for machine learning
- Related work
- Status and plan

ML as optimization

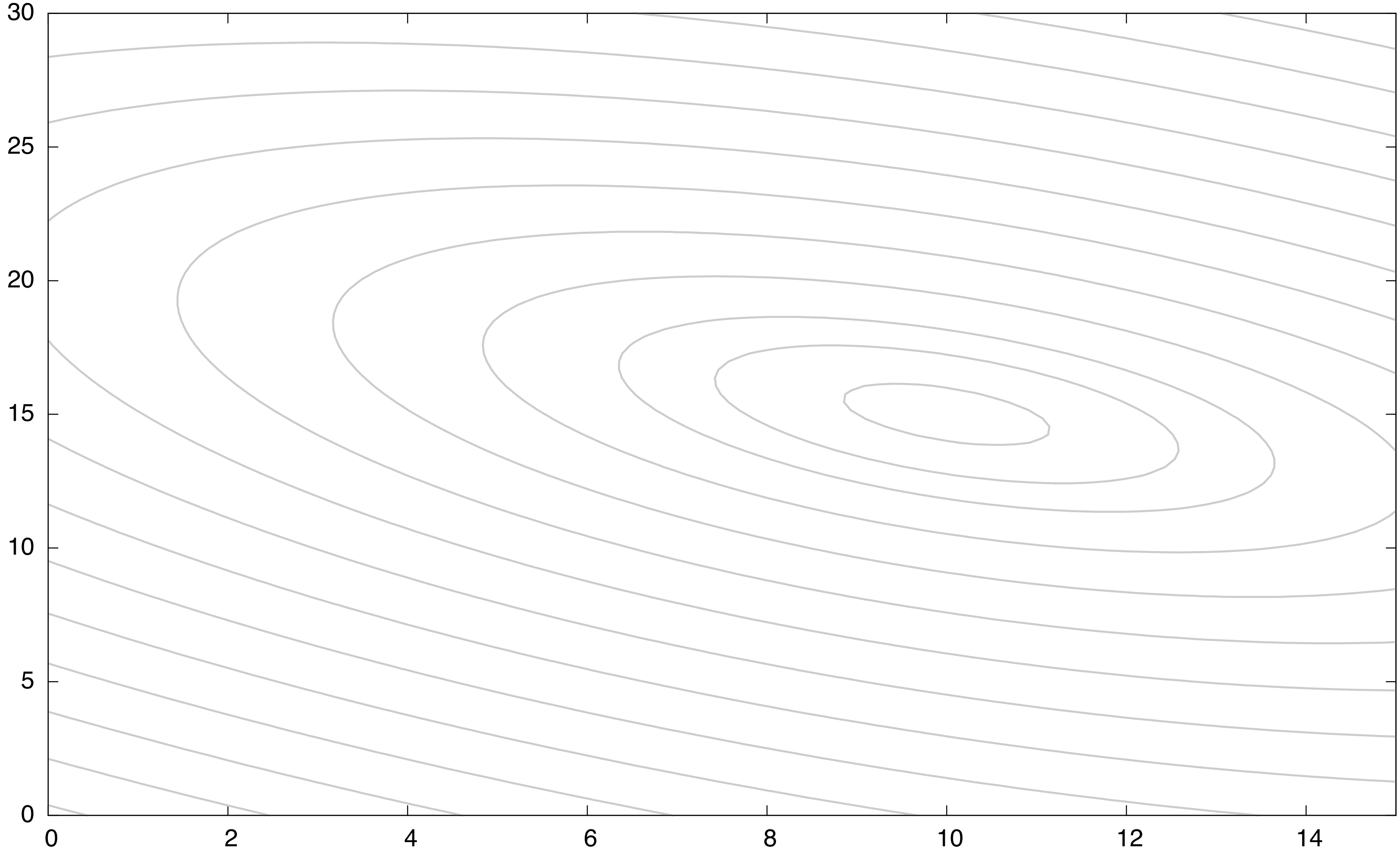
- Many ML algorithms are function optimization
  - Trying to find the X that minimizes f(X)
  - f(X) is a complex function that depends on data
- E.g. document classification
  - Finding topics that documents are about
  - X is the classification of the documents
  - Function f(X) is a penalty function
  - f(X) depends on content of documents

Gradient descent

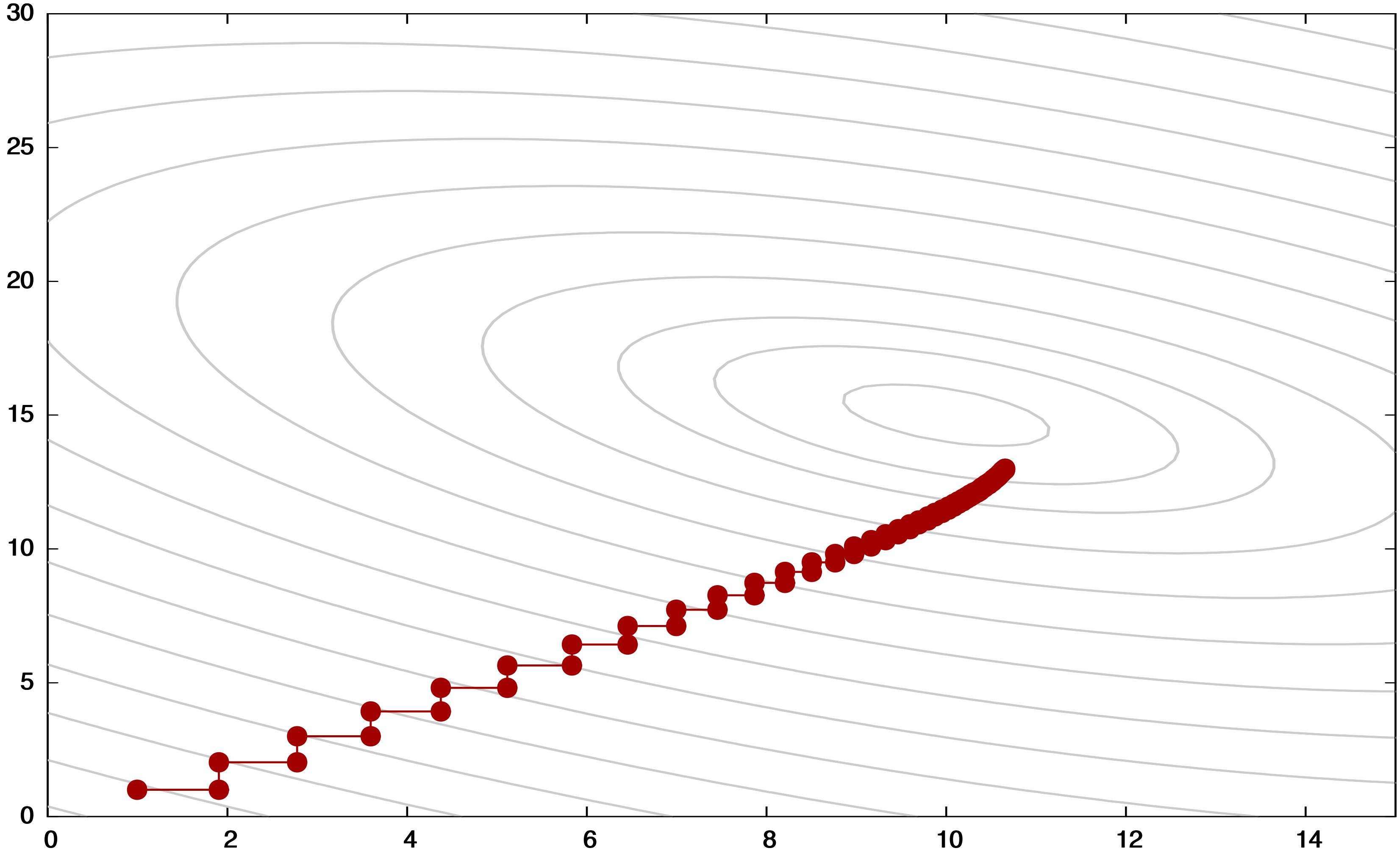
- Basic algorithm
  1. Pick initial guess for X
  2. Calculate gradient at X
  3. Set X := X - grad(f(X))

- Coordinate descent
  - Similar to gradient descent
  - Operates on only one axis at a time

Coordinate descent example



Coordinate descent example



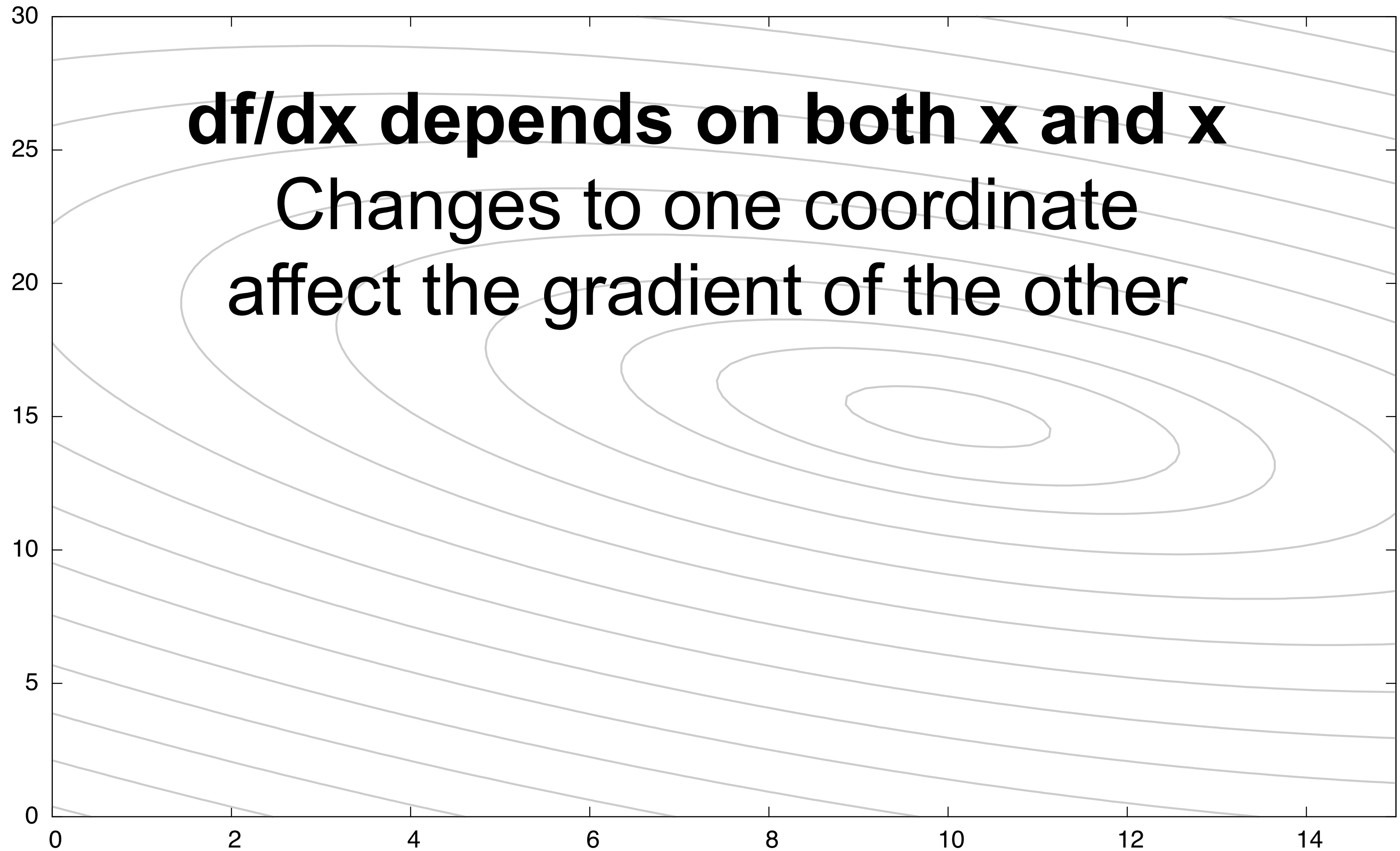
Parallel coordinate descent

- Multiple threads update different axes
- Updates depend on values from other threads
  - Requires synchronization on shared data: X
- What if threads operate on stale data?
  - Updates not available to other threads immediately
  - Do we still find a solution?
  - Can we improve performance?

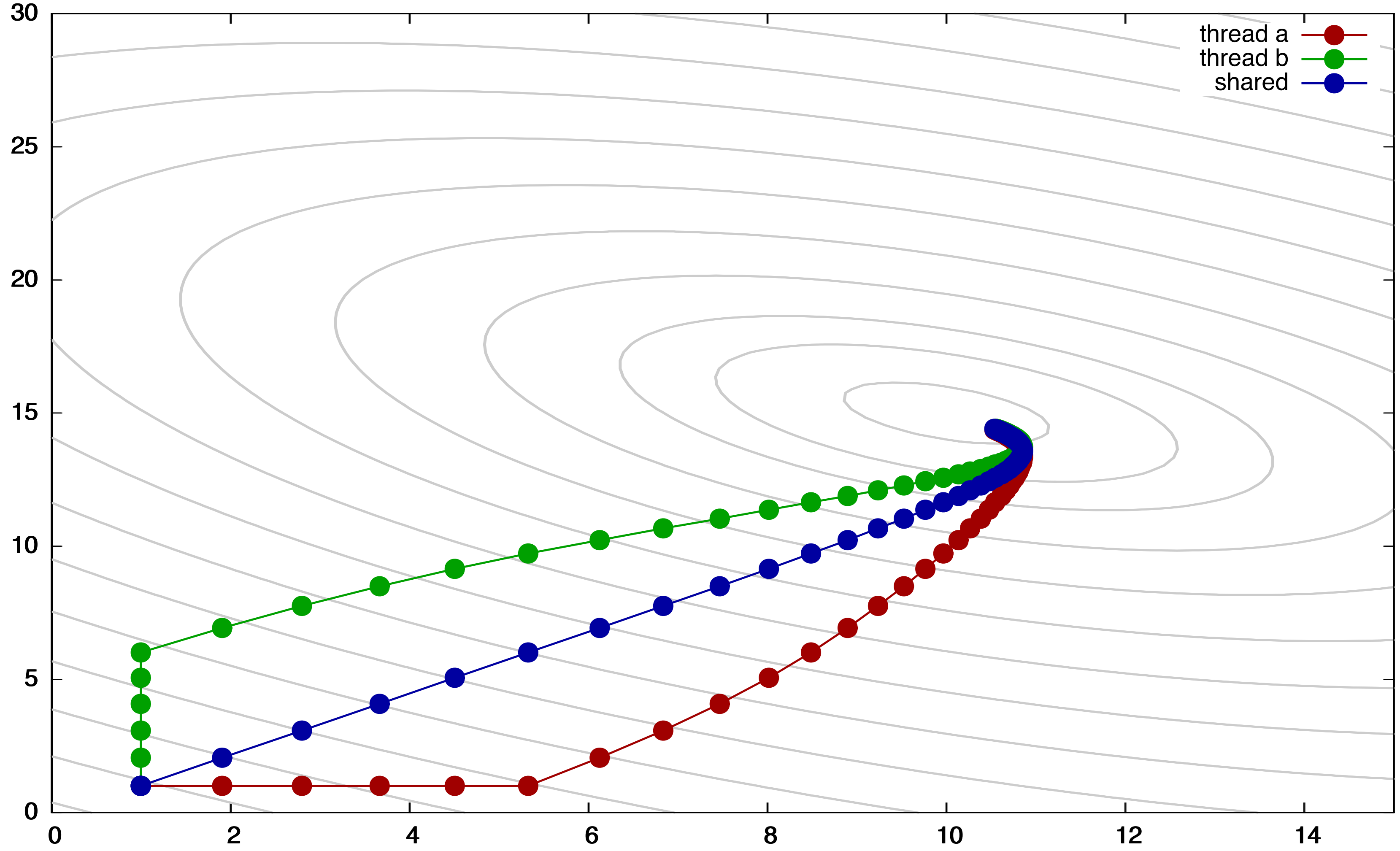
Lazy coordinate descent

- Shared value X
- Each thread also keeps update log
- Thread computes on X and its own update log
  - Apparent value of X is modified by updates in log
- Changes appended to log, not X
- After 5 iterations, update X

Coordinate descent example



Lazy Coordinate Descent



LazyTables design goals

- Shared data structure for machine learning
  - 2 dimensional table of values (floats and ints)
- Extremely high update rate
  - Hundreds of thousands per second per thread
- Reads infrequent, often tolerate stale data
  - Staleness more tolerable at start of algorithm
  - When fine-tuning solution, accurate reads important
- Scalable to different problem sizes
  - From single-machine in-core to distributed out-of-core

Motivating experiments

- Simple C++ table implementation
  - Based on STL `map<>` data structure
  - Get/put, increment/decrement, multiply
- Basic implementation: reader/writer locks
- Lazy implementation
  - Queue updates in thread-local storage
  - After 1k updates - or `flush()` - perform bulk update
- Used actual document classification code
  - Latent Dirichlet Allocation algorithm
  - Similar in behavior to coordinate descent

LDA experiments

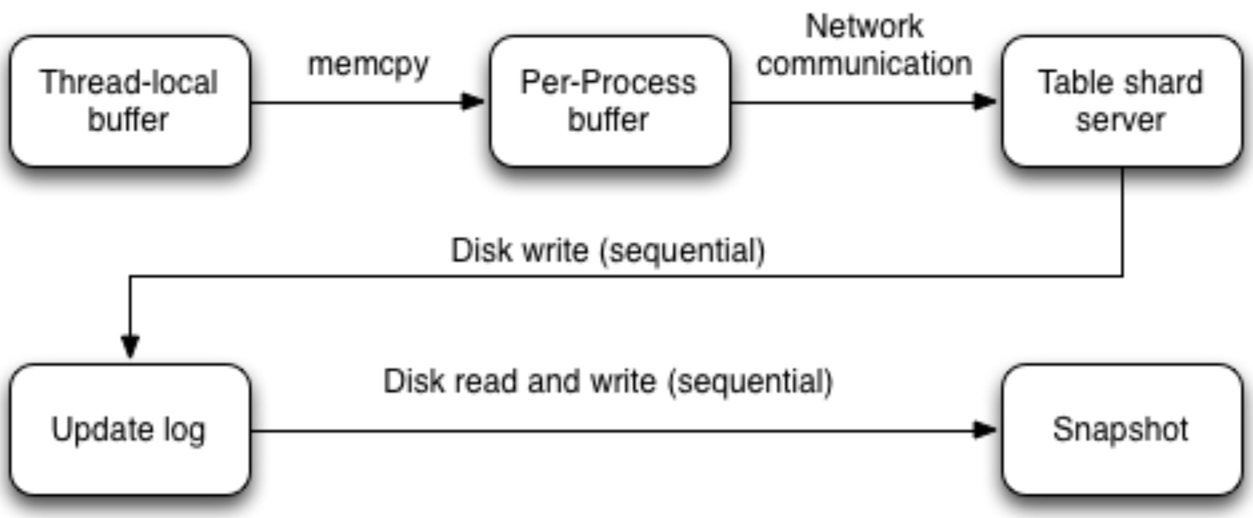
Synchronization method	Threads	Runtime (s)	log-likelihood
Single-threaded	1	62	--1.06015e7
Locking	1	75	--1.06015e7
Batching (1024)	1	66	--1.06015e7
Locking	2	94	--1.07928e7
Batching (1024)	2	36	--1.07981e7
Locking	4	841	--1.08774e7
Batching (1024)	4	20	--1.0868e7
Locking	6	1774	--1.08961e7
Batching (1024)	6	16	--1.08937e7

Batching performance improves with more threads, while locking gets worse

LazyTables design

- Update operations batched at different stages
  - Provides high throughput, low latency update
- Make intermediate data available for query
  - Allow reads to specify what data to look at
- Rows of table grouped into *shards*
  - Shards can be distributed to different servers
- Update written to on-disk log
  - Avoids read-modify-write
  - Reads may specify to only read snapshot, or also log

Potential architecture



Like in LazyBase, queries can access intermediate data from any stage

Future work overview

- Effect of staleness on ML algorithms
- Exploiting staleness for performance

Effect of staleness

- Examine effect of staleness in detail
  - Working with ML researchers on multiple algorithms
- How do we measure freshness and time?
  - Iteration number?
  - Update count?
- How do freshness requirements change?
  - Based on input data?
  - As algorithm progresses?
- What are the consistency requirements?
  - Read-my-writes?

Systems techniques

- How can we exploit tolerance of staleness
- Batching: Thread-local, per-machine, in memory
- Logging: avoid read-modify-write
- Eventually write to a snapshot (i.e. Authority)

Overview

- Introduction and thesis statement
- Two examples of freshness vs. performance
  - LazyBase - data collection and analytics
  - LazyTables – shared data for machine learning
- Related work
- Status and plan

Web view materialization

- Describes freshness/performance tradeoff in serving dynamic web content
- Advocates decoupling content updates, view materialization, and serving clients

- Single freshness parameter for whole system
- Only pre-defined views

- Labrinidis, 2004

Statistical query processing

One can often make reasonable decisions in the absence of perfect answers

-Agarwal et al., BlinkDB

- Speed up analytical queries by sampling data
- Provide accuracy/latency tradeoff

Parallel machine learning

- Much work in distributed/parallel ML
  - Mahout, GraphLab, Spark, Piccolo
- Designing algorithms resistant to inconsistency
  - Shotgun (Bradley et al.) shows that coordinate descent often converges without explicit synchronization

Status and plan

- First case study: LazyBase
  - Started during HP internship
  - Published in EuroSys 2012
- Second case study: LazyTables
  - Prototype and motivating experiments done
  - Expand prototype to test effects of staleness on ML
    - ICML 2013, deadline Dec 15
  - Experiments testing individual components of pipeline
    - HotOS? Other deadline early next year?

Conclusions

Storage systems can and should provide for per-query configuration of the tradeoff between data freshness and query efficiency and staleness.

- LazyUpdates allow us to exploit tradeoff
- 2 example systems
  - LazyBase: data collection and analytics
  - LazyTables: large-scale machine learning