LazyBase

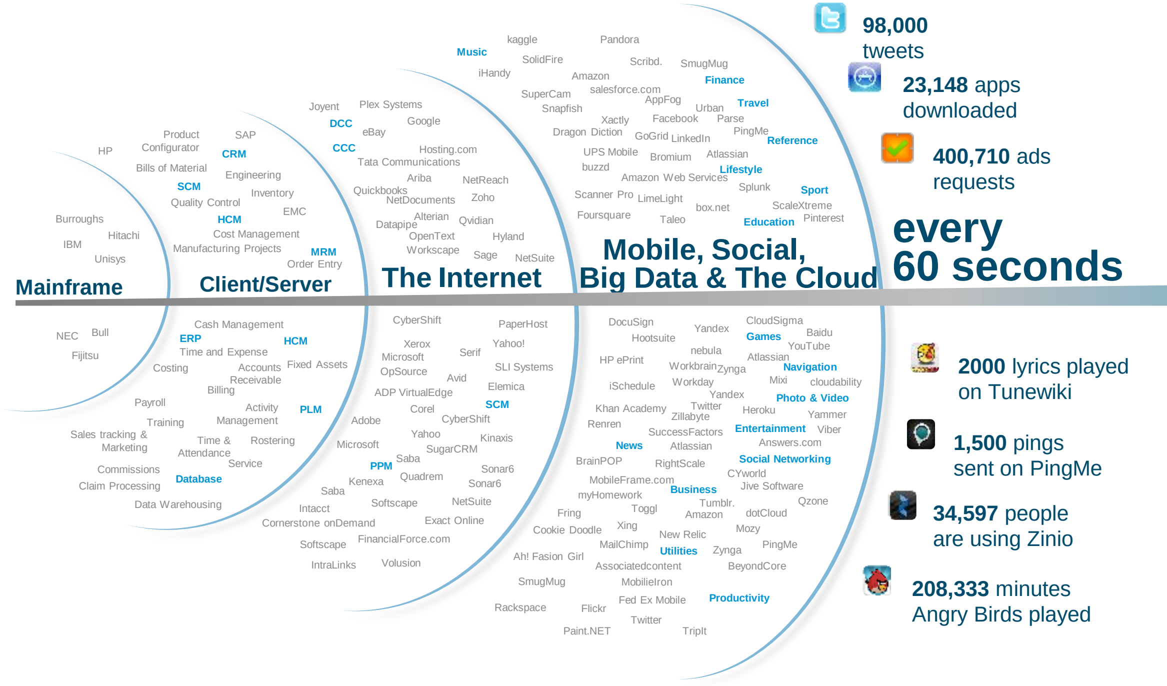
Trading Freshness and Performance in a Scalable Database

Brad Morrey / September 18, 2013

Joint work with:
Jim Cipar and Greg Ganger
Carnegie Mellon University

Kimberly Keeton, Craig Soules, and Alistair Veitch
HP Labs

Big Data Has Arrived…



- **Unstructured data** accounts for **90%** of the information in the world today and growing **62%** CAGR

Even *You* Have Big Data

File Systems contain more and more data in them

Have you ever tried to run “find” over a file system with 500 million files?

Archiving has arrived

Must maintain your data for compliance reasons, but can you ***find*** what you need when you get sued?

You may want to track what is happening to your data

Auditing allows forensic analysis after intrusions

You may want to leverage data feeds like Twitter

Managing that Deluge

Big File Systems/Archive/Content Depot

Scalable “find” – file system metadata index

Activity or change log

Audit log

Custom attribute index

Data feeds like Twitter

If you have a feed, you might want to look at what is “trending” right now

You might also want to do deep data mining on what has occurred over the last week/month/year

Metadata storage background

Conceptually, want a single, large database containing all of the metadata we collect. It should:

- Scale well
- Handle a high update rate
- Isolate query performance from ongoing updates
- Allow correlations across tables (I.e. Queries supporting joins)

Leverage the fact that many enterprise applications can work on slightly stale data

- E.g., Deep dive analysis looking at last 5 months might not need updates from last day

Application freshness requirements

Freshness / Domain	Seconds	Minutes	Hours+
Big File System	Very selective “find” command	Activity log, Audit log	Storage trends, E-discovery requests
Twitter Feeds	USGS earthquake detector	Trending world news	Social network graph analysis

“Freshness ” – how long after insertion can it be queried

Existing solutions fall short

OLTP databases

Don’t provide good isolation between bulk loads of updates and queries.

Additionally, transactional update rates are low unless you pay $$$$

Data warehouses

ETL implies a delay before newly ingested data can be queried

NoSQL databases

Provide hard-to-understand eventual consistency and inefficient update throughput

Outline

- Motivation
- LazyBase
  - Description
  - Research Results
- LazyBase -> Express Query
  - HP Product with LazyBase embedded
- Conclusions

LazyBase Organizes the Deluge

Provides Efficient Scalable Data Managment

Efficient – highly optimized per-node performance

Scalable - linear scalability with node count

Data - can be events, metadata, or actual data organized into a typed schema

Management - make durable, index, and allow queries over

Supports continuous high-throughput updates and big analytical queries

LazyBase Gives You Control
Lets You Reason about the Results

Allows per query *freshness* selection

The user is able to trade query response time for freshness on a per query basis

Users might want:

As fresh as possible - willing to accept slower query response time

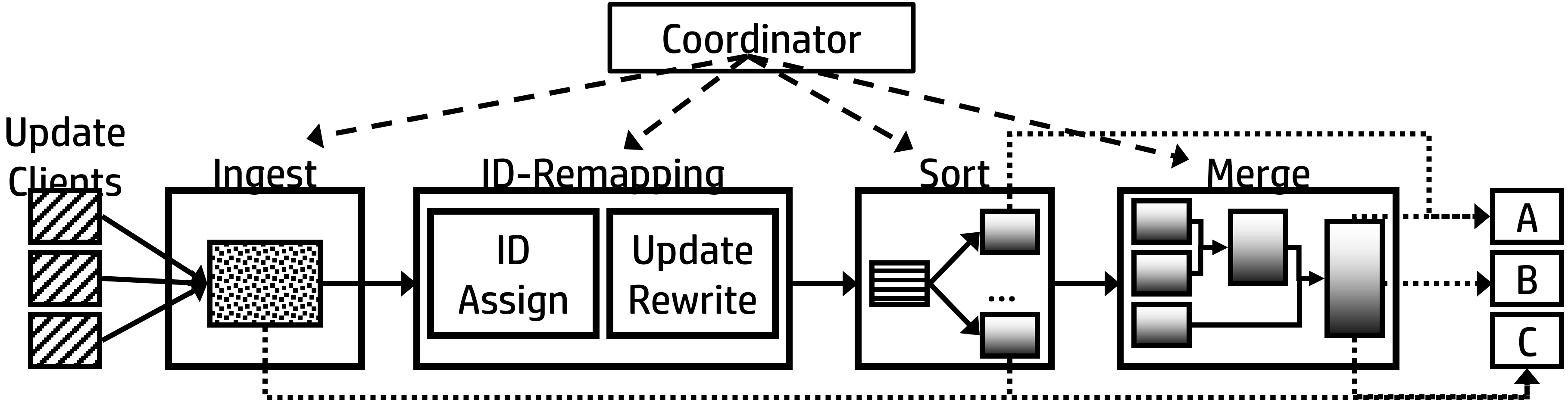
More efficient queries without the freshest data

Queries are always run over a consistent snapshot

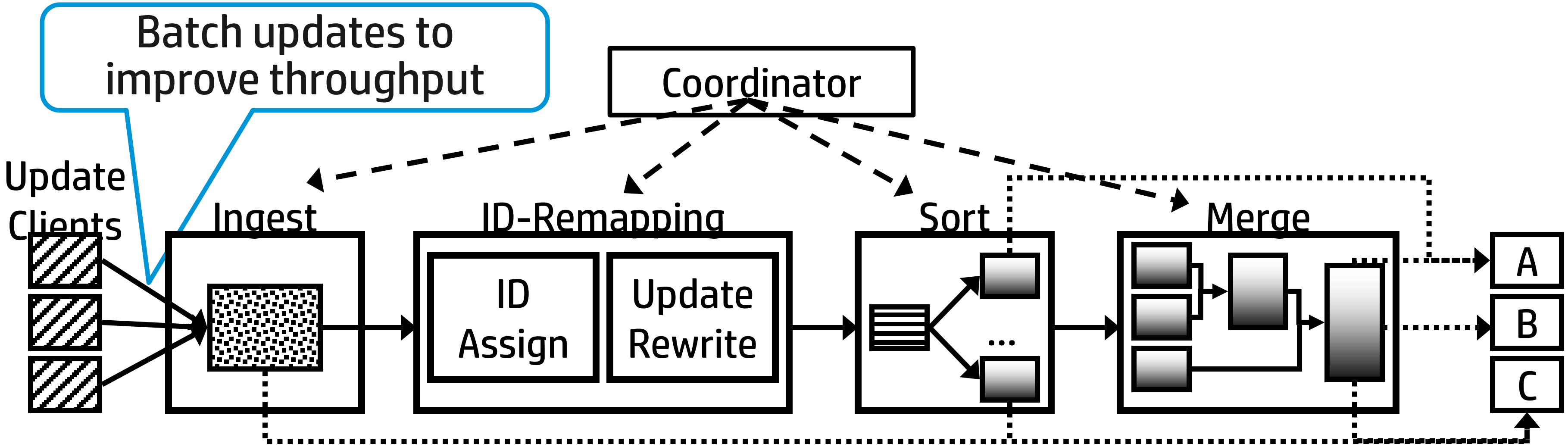
Multi-table joins will always return consistent results in LazyBase

More details on consistency a bit later

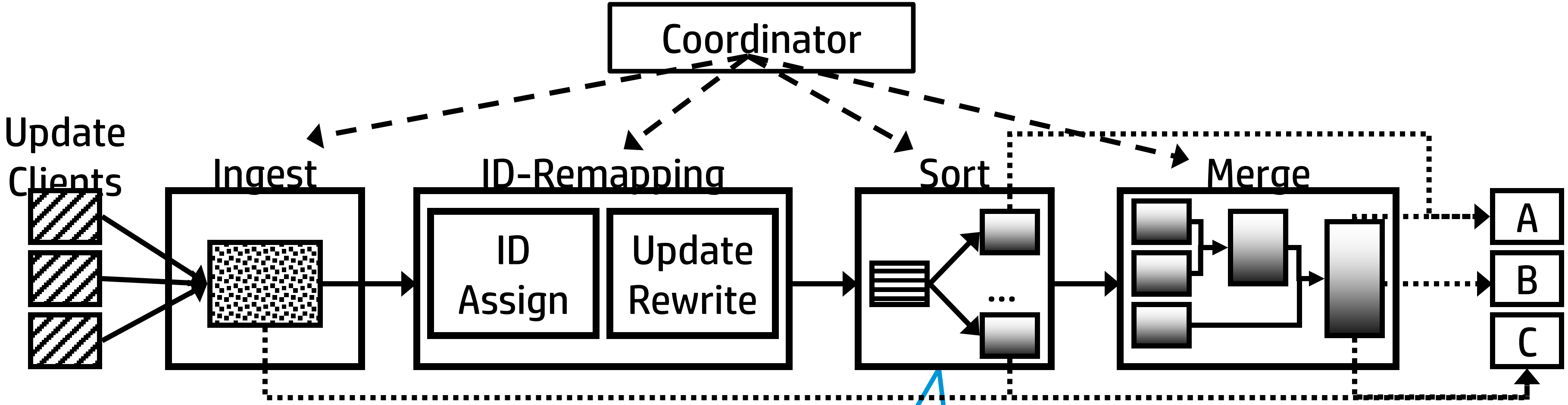
LazyBase architecture



LazyBase architecture

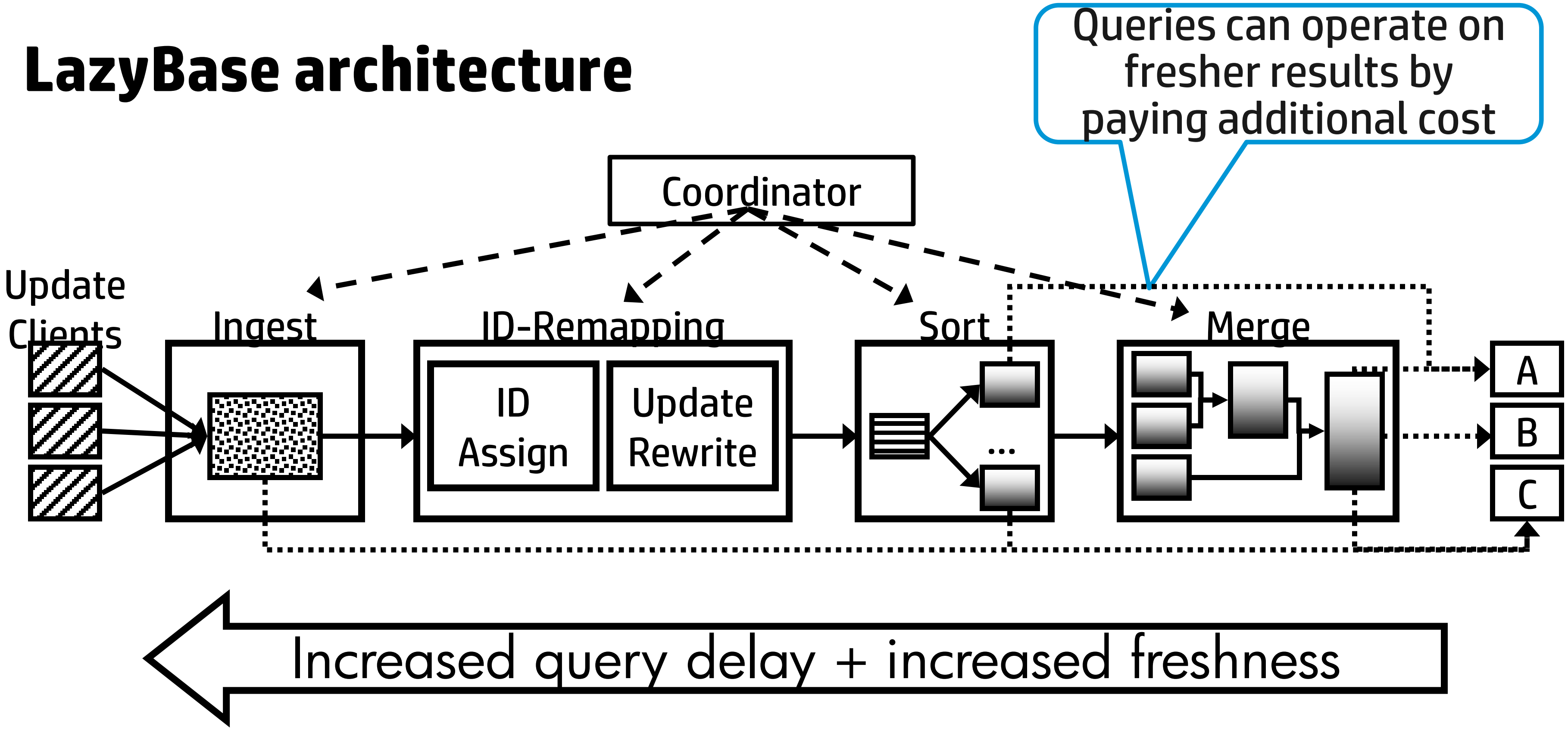


LazyBase architecture



Each stage is individually parallelizable to provide scalability

LazyBase architecture



Transactional Model

Updates batched into *self-consistent updates* (SCUs)

Each SCU contains inserts, updates and deletes

Represents a consistent state of all tables affected by that set of updates

Atomic *multi-row* updates

SCUs are made durable immediately (at ingest)

SCUs are applied atomically and in order at each stage of the pipeline

Only supports *observational data*

Inserts, updates and deletes must contain primary key of table

Therefore no read-modify-write transactions

Consistency for queries in LazyBase

Similar to snapshot consistency

Monotonicity:

If a query sees update A, all subsequent queries will see update A

Consistent Prefix:

If a query sees update number X, it will also see updates 1…(X-1)

Query model

Queries are read-only lookups

Updates handled through ingestion pipeline

Queries may request data from increasingly earlier stages to achieve the desired freshness

Queries are specified programmatically or using a limited subset of SQL that specifies freshness (seconds out-of-date)

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

Query execution

Query clients send parallel requests to pipeline stage workers

Workers perform filtering and projection locally

Query client combines results

DataSeries: LazyBase Storage Layer

***Open source* storage layer originally designed for log storage and analysis**

Arbitrary relational tables with named columns

Variety of compression techniques to trade CPU utilization for I/O bandwidth

- Delta-coding, duplicate string elimination, generic compression, etc.

Performance close to speed of a type-specific binary format

Excellent streaming performance

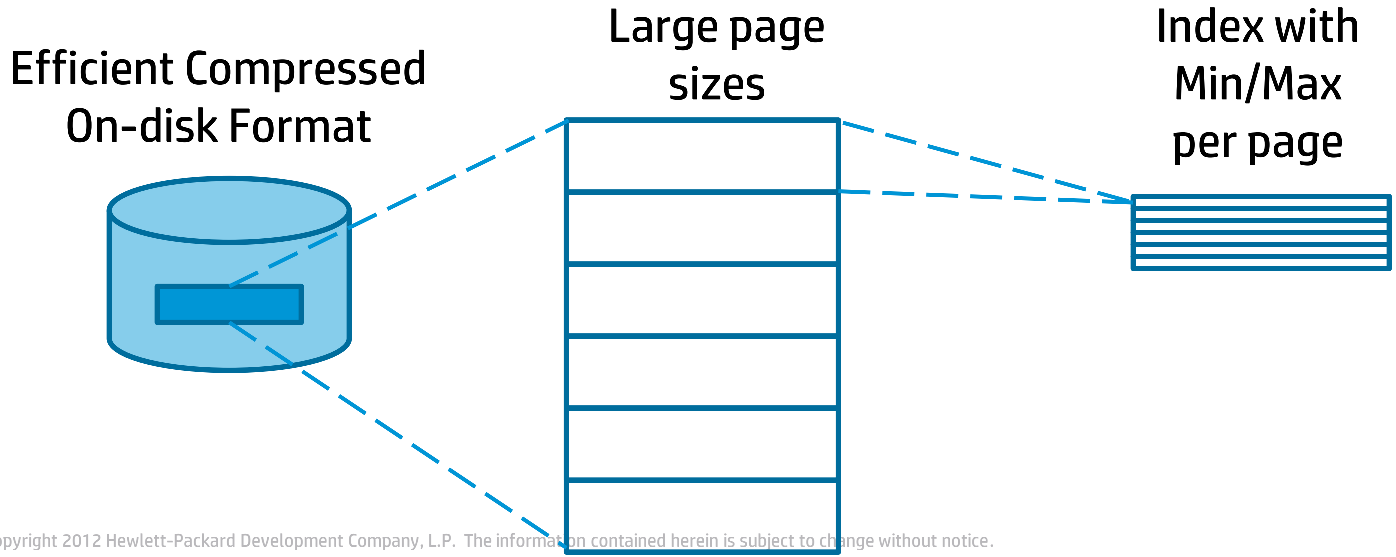
Benefits from DataSeries storage

Large pages increase streaming throughput

Very useful for data mining analyses

Range-based indexes fit in memory

Large pages provide a ~1000:1 table to index ratio



Outline

Motivation

LazyBase

- Description
- Research Results

LazyBase -> Express Query

- HP Product with LazyBase embedded

Conclusions

Experiments

Efficiency of batching

Ingest scalability

Query scalability

Point queries

Range queries

Query freshness vs. performance tradeoff

Consistency

Experimental setup

Platform: Linux kernel virtual machines on OpenCirrus cluster

6 dedicated cores, 12GB RAM, local storage

Workload: ingest data set of ~38 million tweets

50 GB uncompressed

Comparison point: Cassandra

Reputation as “write-optimized” NoSQL database

Default configuration:

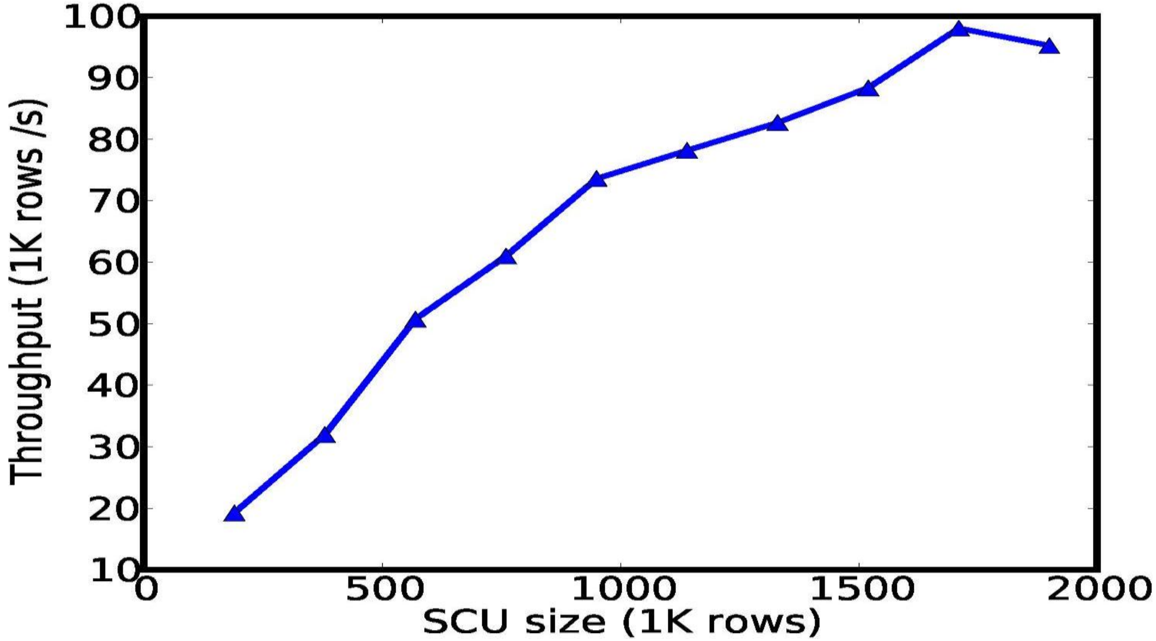
LazyBase: 9 dynamically assigned pipeline worker+ingest, 1 ID remap+coordinator

Cassandra: 9 nodes (equivalent to 9 workers for LazyBase)

1.95M row SCUs, up to 8-SCU merges

20 upload nodes

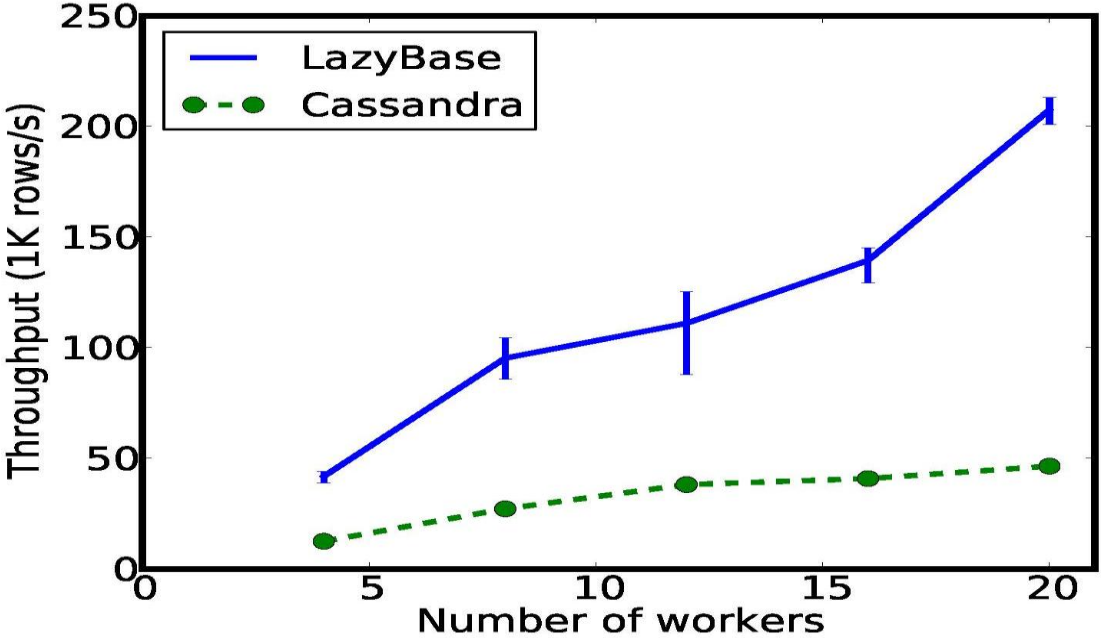
Efficiency of batching



Larger update batches improve ingest throughput

Also increase ingest latency

Ingest scalability

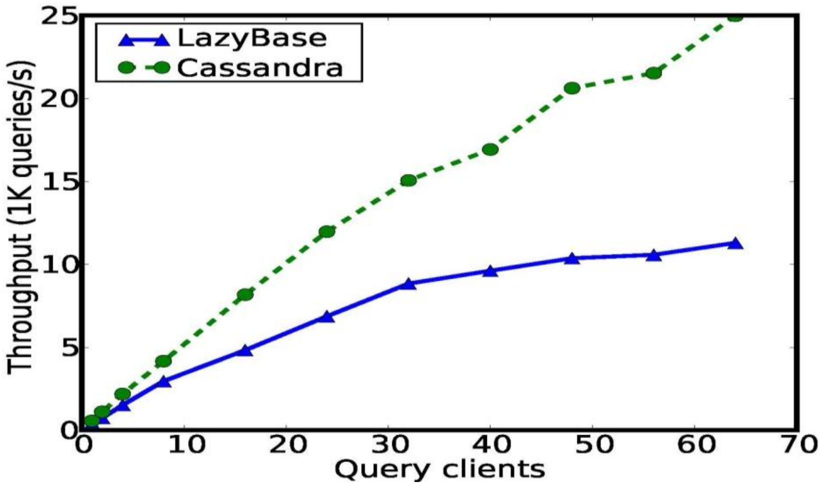


LazyBase scales effectively up to 20 servers

Efficiency is ~4x better than Cassandra

Query throughput

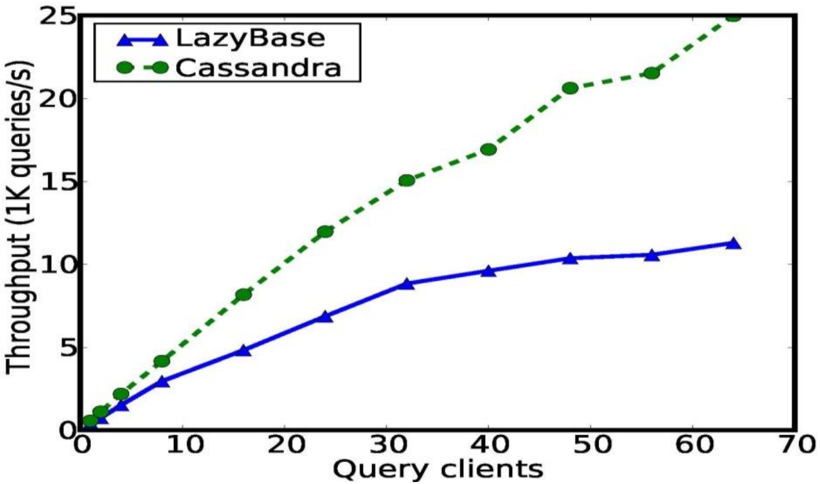
Point queries



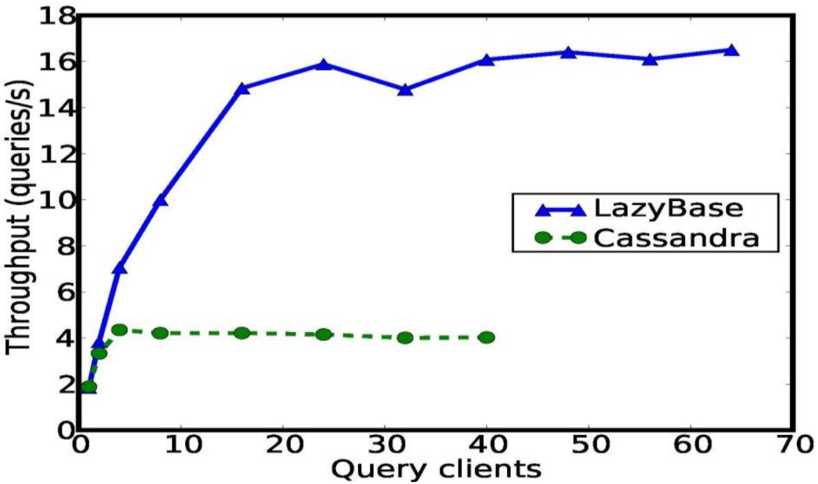
LazyBase point query throughput limited by DataSeries page decompression

Query throughput

Point queries



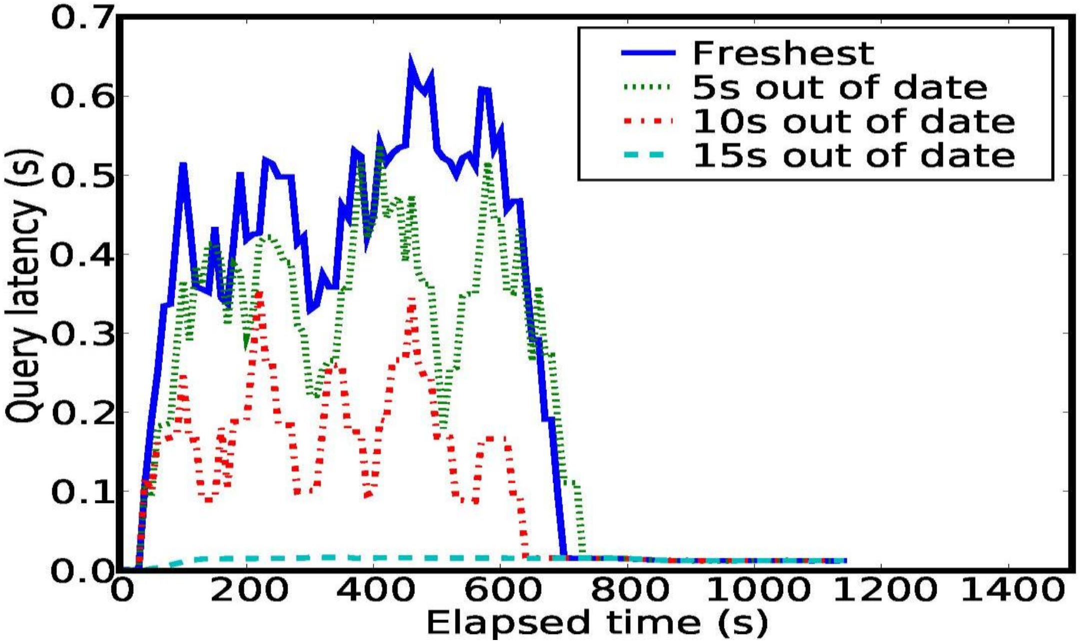
Range queries



LazyBase point query throughput limited by DataSeries page decompression

LazyBase sorts data, providing ~4X improvement over Cassandra's hash distribution

Query freshness/performance tradeoff



Increasing freshness also increases query latency

In practice, “stale” data often suffices for application queries

Consistency experiment

Workload

Single integer column table with two rows (A, B)

Sequence of update pairs (increment A, decrement B)

Goal: invariant of A + B = 0

Background Twitter workload to keep servers busy

LazyBase

Issue update pair, commit

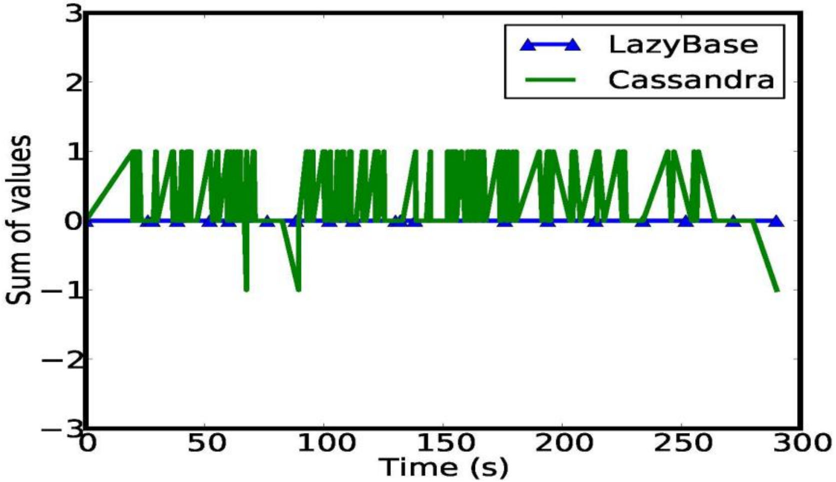
Cassandra

Update pair uses “batch update” call

Quorum consistency model: updates sent to majority of nodes before returning from update call

Consistency

Sum = A + B

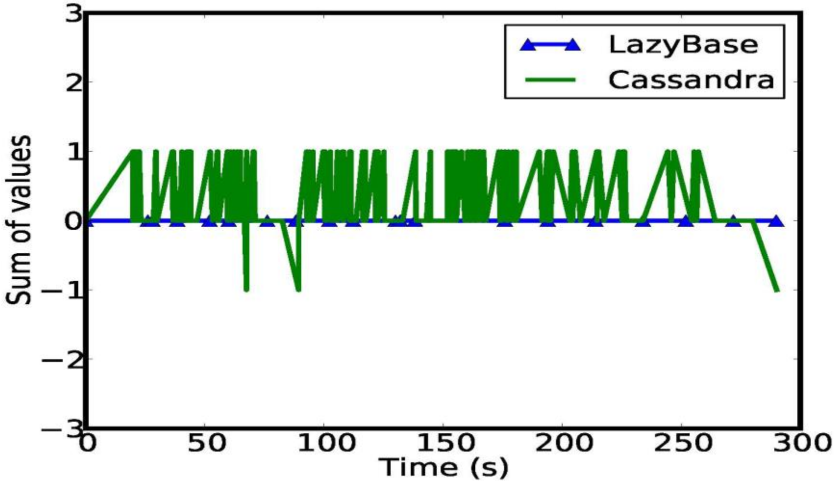


Cassandra observes inconsistencies (non-zero values)

LazyBase always provides consistent results

Consistency

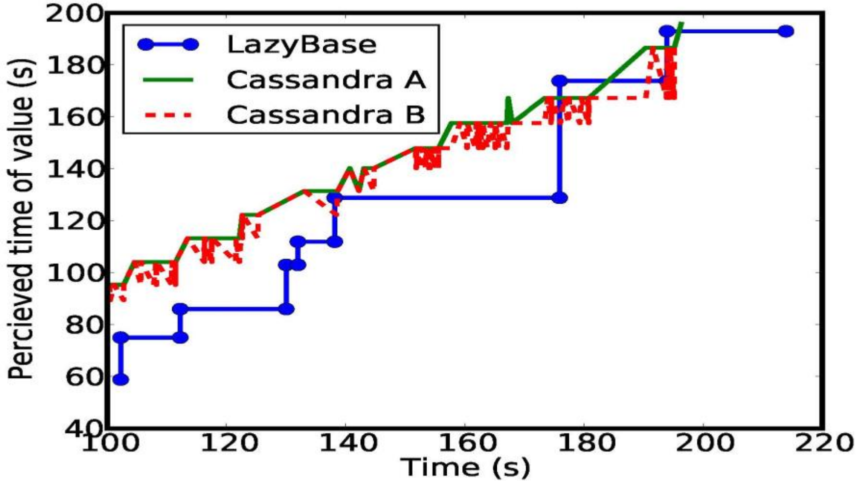
Sum = A + B



Cassandra observes inconsistencies (non-zero values)

LazyBase always provides consistent results

Result freshness



Cassandra A/B differences are inconsistencies between rows

Single-row dips indicate violations of read monotonicity

LazyBase may be less fresh, but always provides read monotonicity

LazyBase Research Summary

High-throughput insert/update rates

Batching and task specialization using pipelining

Scalability

Each stage of pipeline parallelizable across many nodes

Understandable consistency model

Atomic update batches with snapshot isolation on lookups

Pay only for desired query result freshness

Lookups may see slightly out-of-date results

Can pay more at query time to get more up-to-date results

Jim Cipar, Greg Ganger, Kimberly Keeton, Charles B. Morrey III, Craig Soules, Alistair Veitch, "LazyBase: Trading freshness for performance in a scalable database," *Proc. of EuroSys*, April 2012.

Outline

Motivation

LazyBase

- Description
- Research Results

LazyBase -> Express Query

- HP Product with LazyBase embedded

Conclusions

StoreAll Archiving with Express Query

Integrated LazyBase with the HP StoreAll scalable file system product line to provide new features or improve existing ones

Metadata indexing, activity/change log, reporting, auditing, validation, user-defined metadata

Aim is "intelligent" storage that integrates structured metadata with unstructured data and analytics

NEW: HP StoreAll

Hyperscale storage to tame and mine your content explosion

Hyperscale: Massive scalability without complexity
Scale to over 1000 nodes,16PB, and billions of objects and files in a single namespace

Harnessed: Structure for unstructured data
Custom meta tagging, retention policies and WORM, and autonomic protection for data durability

Instant: Ultra-fast search and value extraction at petabyte scale
Discovery, compliance, and analytics with Express Query and Autonomy IDOL integration

Economic: Scale down costs of storing data over time
Policy-based tiering and cost efficient capacity with a scale-out pay as you grow architecture

100,000x faster search at petabyte scale

NEW: HP StoreAll Express Query For Instant Value Extraction And Search Of Big Data

Filter: Reduce compute overhead to maximize efficiency

Offload parametric queries to HP StoreAll with manual or API interface

Find: 100,000X faster search increases business agility

Eliminate filesystem scans when a race against the clock matters

Freshen: Pinpoint accuracy for business insight

Autonomic indexing and custom tagging for up to date search without performance impact

NEW: HP StoreAll Express Query For Instant Value Extraction And Search Of Big Data

Developed with HP Labs

Filter: Reduce compute overhead to maximize efficiency

Offload parametric queries to HP StoreAll with manual or API interface

Find: 100,000X faster search increases business agility

Eliminate filesystem scans when a race against the clock matters

Freshen: Pinpoint accuracy for business insight

Autonomic indexing and custom tagging for up to date search without performance impact

DreamWorks Animation SKG

symantec.

GE Healthcare

Autonomy an HP company

Express Query performance

Experiment:

Find recently modified files in a ~500 million file system

StoreAll 9320 scale-out storage configuration

Traditional "find" command: 151,278 seconds

Express Query: 1.434 seconds

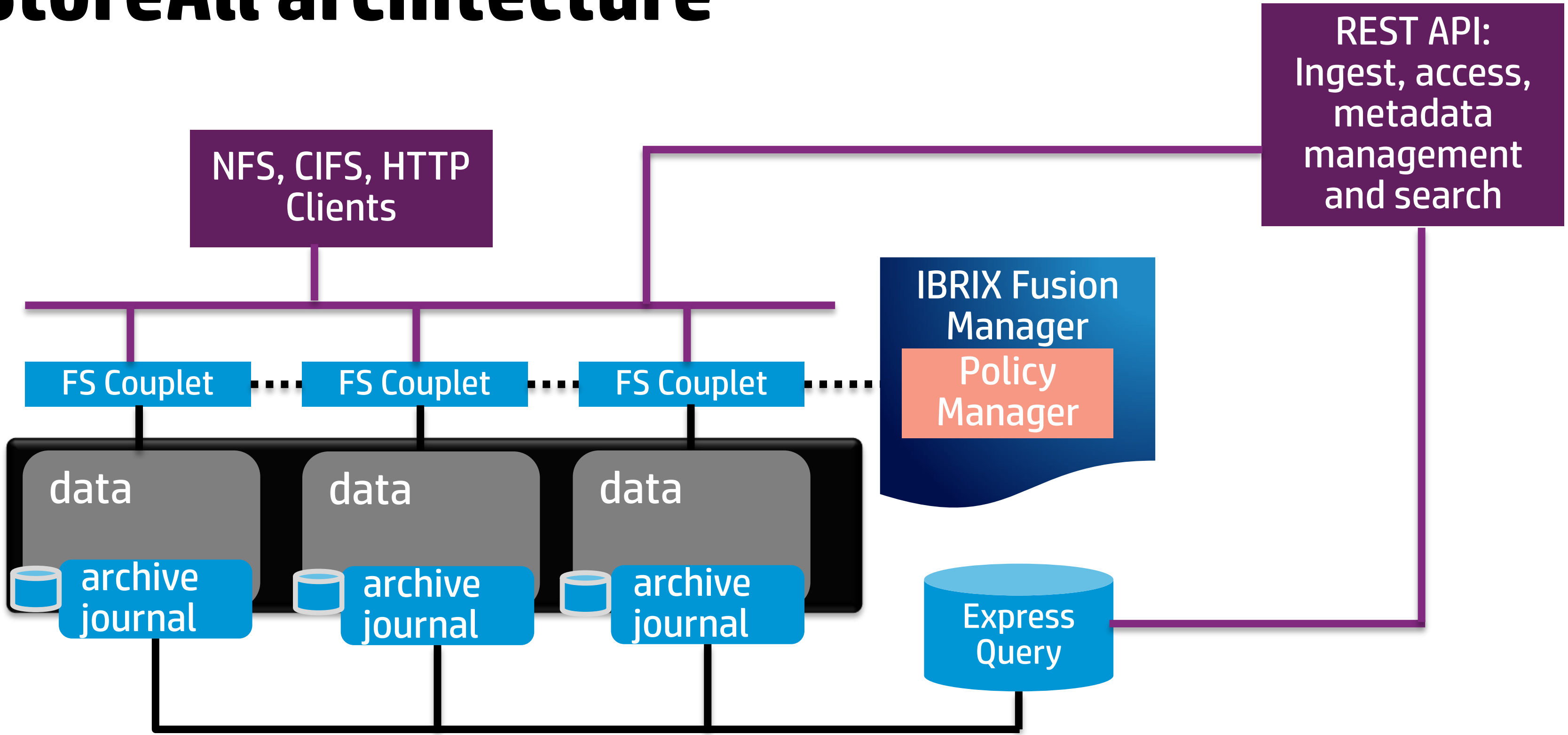
Express Query was 105,500 times faster!

Actually Express Query is arbitrarily faster for this type of query

Query is relatively constant time

Search grows with size of file system

StoreAll architecture



Turning LazyBase into Express Query

“Hardening” prototype code

Flexible query interface: SQL query front end

Working with the StoreAll ecosystem

Single node LazyBase pipeline

Archive journal (AJ) design

Identifying file system events and designing schema

AJ scanner design

RESTful API design and implementation

Request parser and SQL query generator

Scalability testing to 1 billion files

Querying Express Query

Supports querying authority tables at end of pipeline

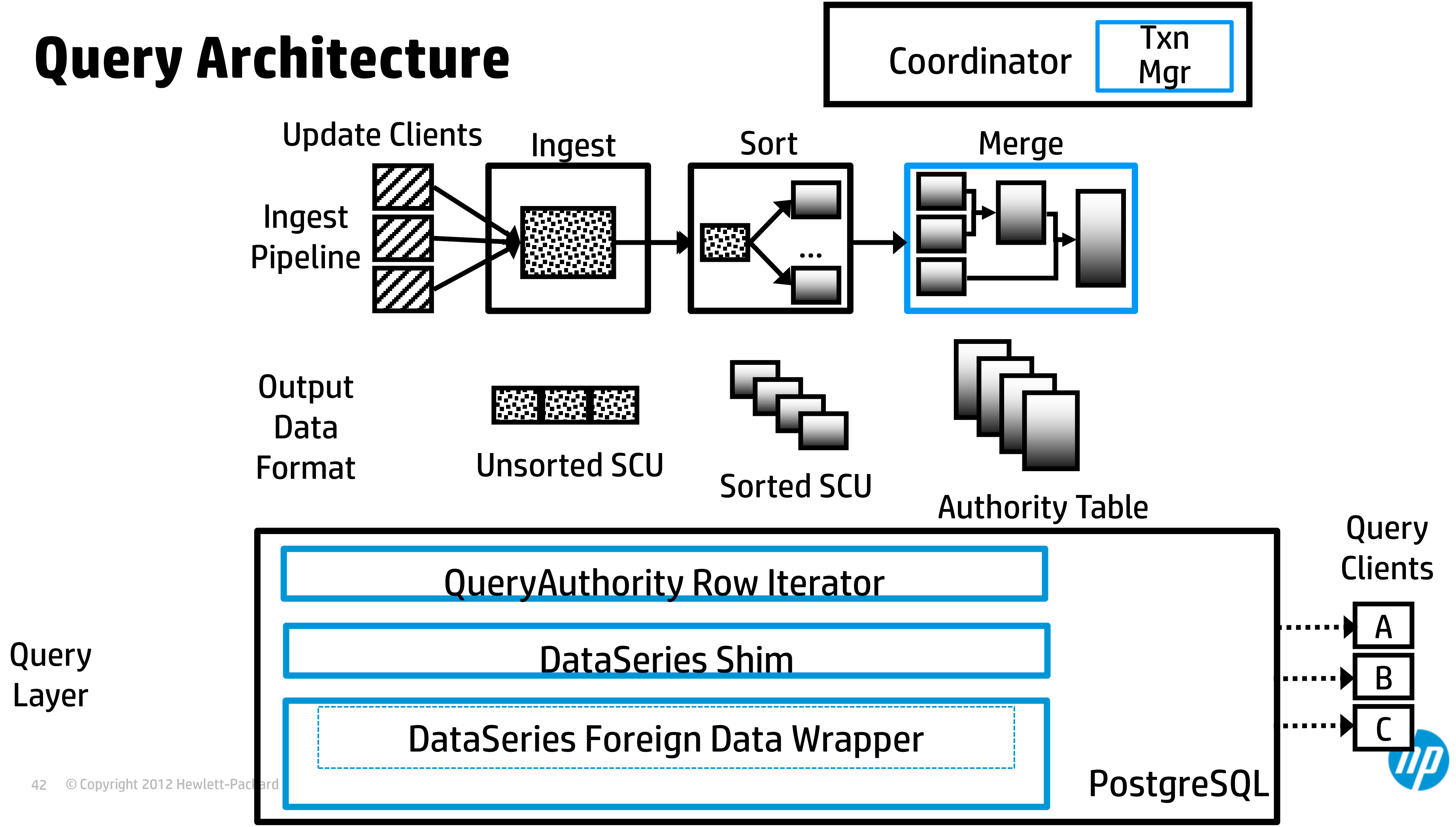
Queries are specified in SQL using PostgreSQL

PostgreSQL Foreign Data Wrapper for DataSeries

Basic flow is similar to any relational model

Lookups on indexes to retrieve matching rows, join results

Query Architecture



Summary

Metadata store for analytics on observational data

High-throughput insert/update rates

Scalability

Understandable consistency model

Per-query result freshness vs. performance tradeoffs

StoreAll with Express Query: StoreAll scalable file system + LazyBase

An HP product with HP Labs technology inside

For more information

Publication:

J. Cipar, G. Ganger, K. Keeton, C. Morrey, C. Soules, A. Veitch, "LazyBase: Trading freshness for performance in a scalable database," *Proc. EuroSys*, April 2012.

Email us:

Brad.Morrey@hp.com

Kimberly.Keeton@hp.com

Alistair.Veitch@hp.com

URLs:

www.hp.com/go/storeall

https://github.com/dataseries

Thank you

Backup slides

Example JSON output

[root@ib77s8 ~]# curl http://192.168.77.8/api/file1?attributes=*


```
[
 {
   "file1" :
   {
     "system::ownerUserId" : 0,
     "system::size" : 0,
     "system::ownerGroupId" : 0,
     "system::onDiskAtime" : 1347366778.000000000,
     "system::lastModifiedTime" : 1347280908.000000000,
     "system::createTime" : 1347366778.095327000,
     "system::retentionState" : 0,
    "tag2" : “myTagHere",
   }
 }
]
```