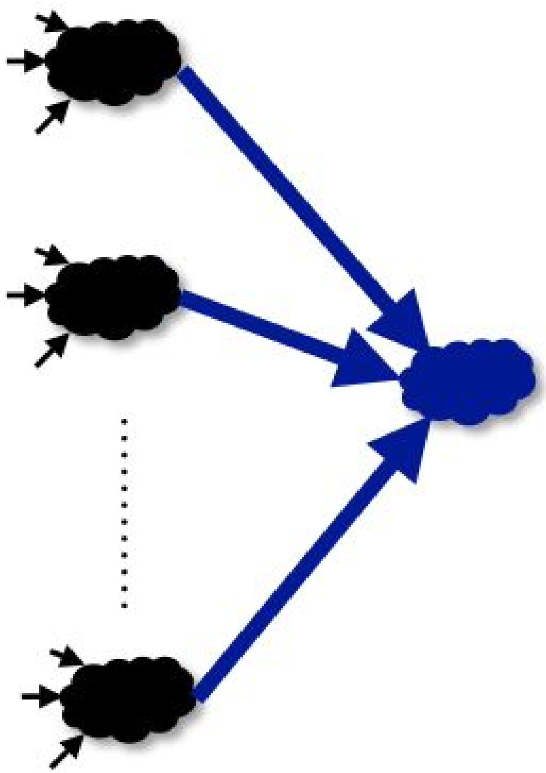
Global Analytics in the Face of Bandwidth and Regulatory Constraints

Ashish Vulimiri, Carlo Curino, Brighten Godfrey, Thomas Jungblut, Jitu Padhye, George Varghese

NSDI ’15

Presenter: Sarthak Grover

Motivation



~ 10 TB/day

- Current centralized approach inadequate
  - Scarce, expensive cross-DC bandwidth
  - Incompatible with sovereignty concerns

SQL analytics across geo-distributed data to extract insights

Problem Statement:
Geo-Distributed SQL Analysis

- Given:
  - Data born distributed across DCs
- Goal: support SQL analytics on this data
  - Minimize **bandwidth** cost
  - Handle:
    - **fault-tolerance**
    - **sovereignty** constraints

Example

Data Collected:

- ClickLog(sourceIP,destURL,visitDate,adRevenue,...)
- PageInfo(pageURL,pageSize,pageRank,...)

Q: SELECT sourceIP, sum(adRevenue), avg(pageRank)

FROM ClickLog cl JOIN PageInfo pi

ON cl.destURL = pi.pageURL

WHERE pi.pageCategory = ’Entertainment’

GROUP BY sourceIP

HAVING sum(adRevenue) >= 100

Example

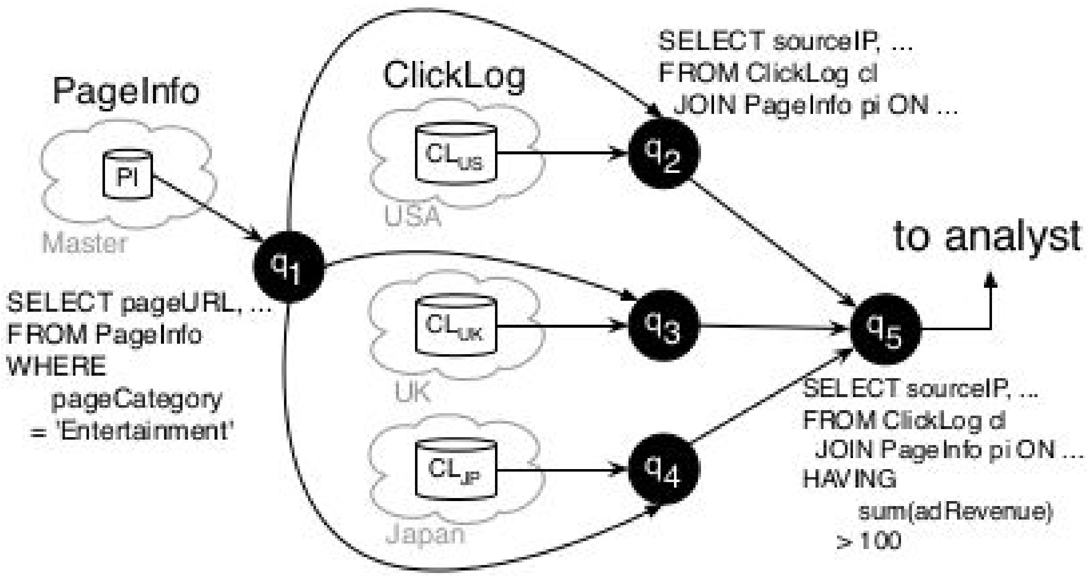
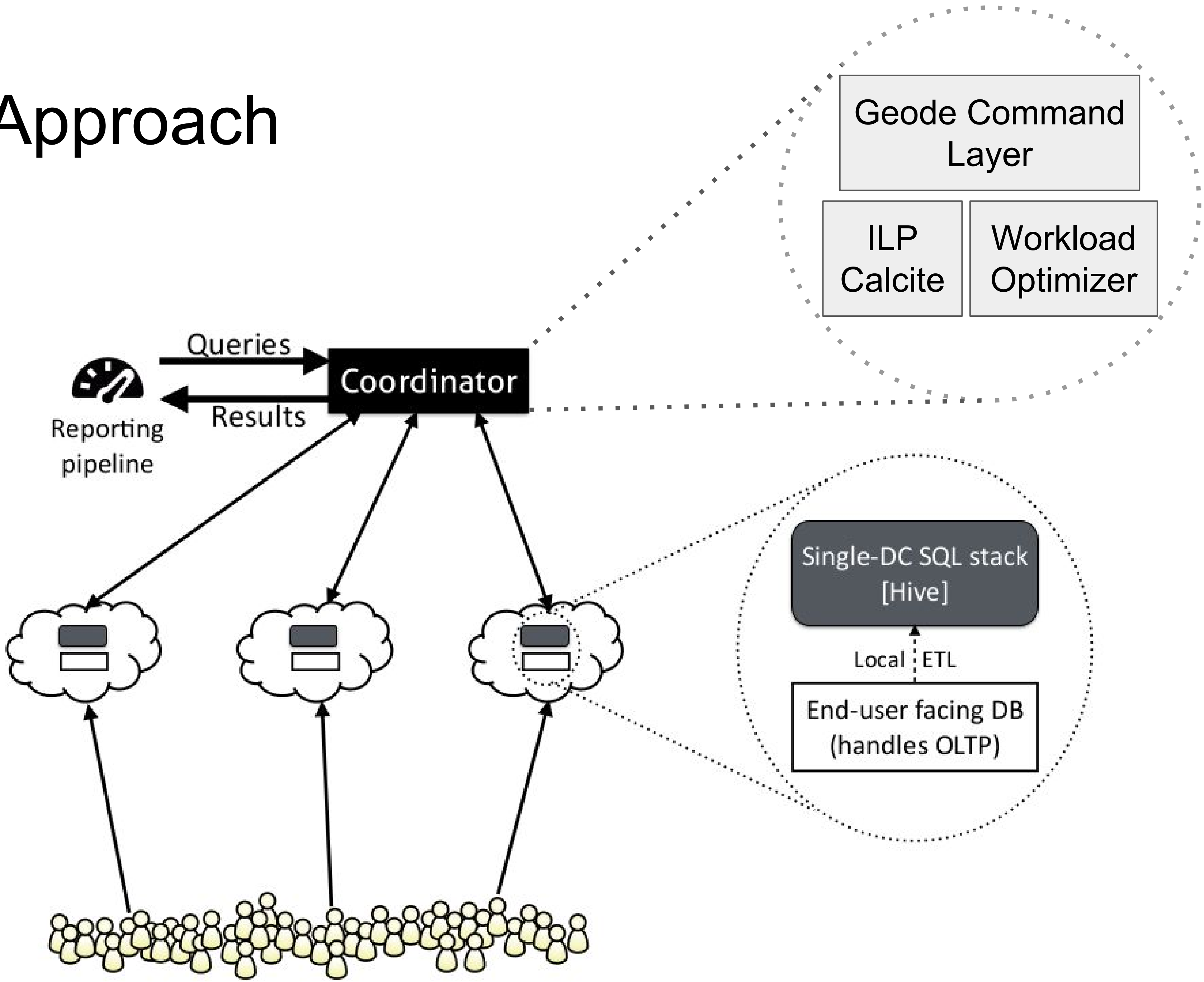


Figure 2: DAG corresponding to Q_{opt}

- Replicate smaller table
- Broadcast joins
- Schedule q to minimize BW

Approach



Geode Command Layer
ILP Calcite
Workload Optimizer
Queries
Coordinator
Results
Reporting pipeline
Single-DC SQL stack [Hive]
Local ETL
End-user facing DB (handles OLTP)

Geode Command Layer

- **Logically centralized view** over data partitioned and/or replicated across Hive instances in multiple data centers.
- Each table contains **partition** column
- Supports **joins** and **nested queries**

Design Goal: BW optimization

Given an SQL query:

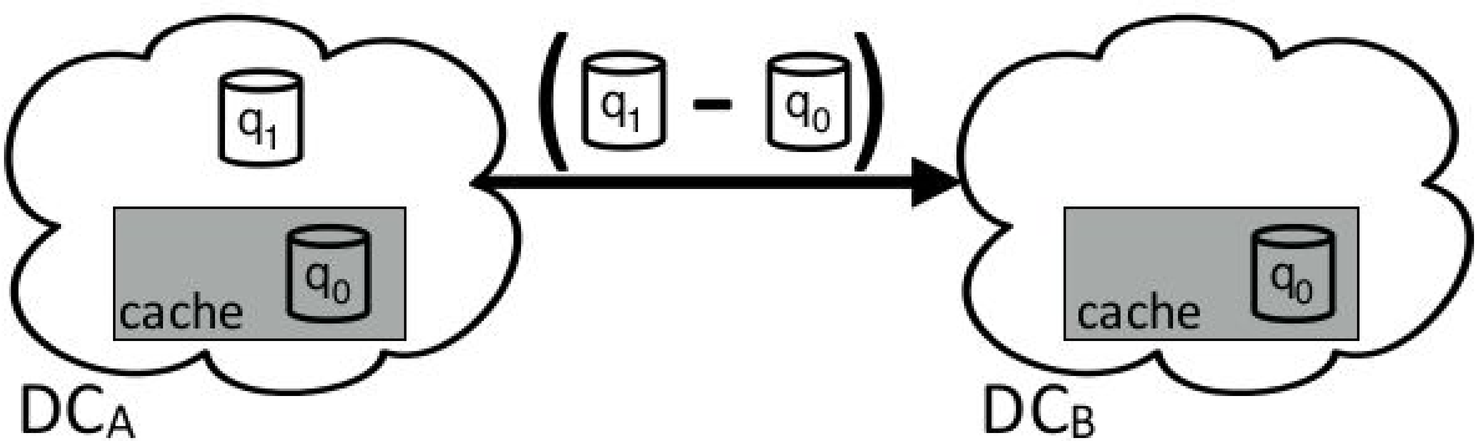
- Choose join order and strategies
- Schedule tasks

Optimizations:

1. Minimize Cross-DC bandwidth (S3)
2. Plan SQL query and schedule tasks given sovereignty, fault tolerance constraints to minimize transfer costs (S4)
3. Extended optimization for specific functions (S5)

Minimize Cross-DC Bandwidth

- Geode is meant for repeated queries over a changing database
- Each DC
  - Cache subquery intermediate results
  - Transfer deltas



Optimizations

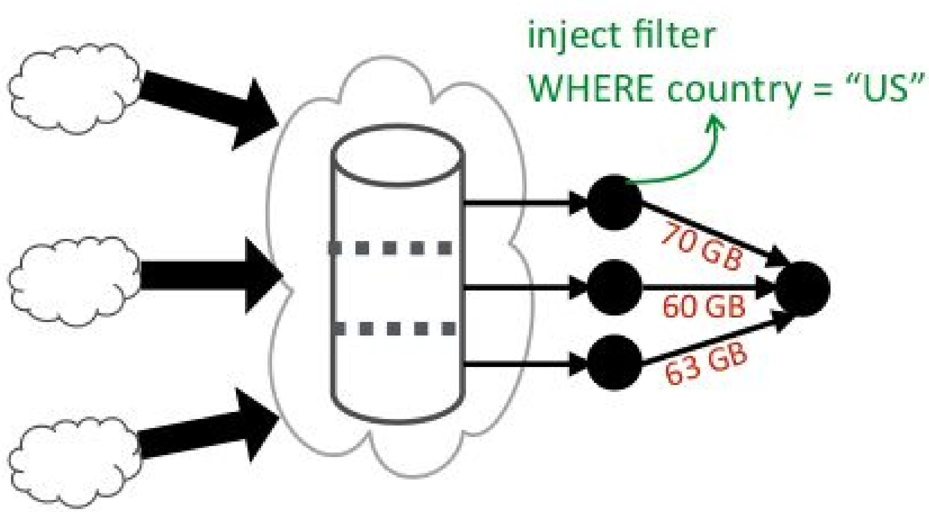
1. Minimize Cross-DC bandwidth
2. **Plan SQL query and schedule tasks given sovereignty, fault tolerance constraints to minimize transfer costs**
3. Extended optimization for specific functions

Workload Optimizer

- Maximize performance
- Jointly optimize:
  - Query plan
  - Site selection
  - Data replication
- Steps:
  - Find the best centralized plan (Calcite++)
  - Decompose centralized to distributed using heuristics
    - Pseudo-distributed execution
    - ILP

Pseudo-distributed Execution

- Calcite++ gives optimum JOIN strategy for tables
- Assume centralized execution, form partitions, measure data transfer for different strategies
- Only execute whenever re-evaluation is needed (eg: initialization, new DC added, …)



- Centralized bootstrapping
- SELECT … WHERE country=’US’
- Measure transfer costs

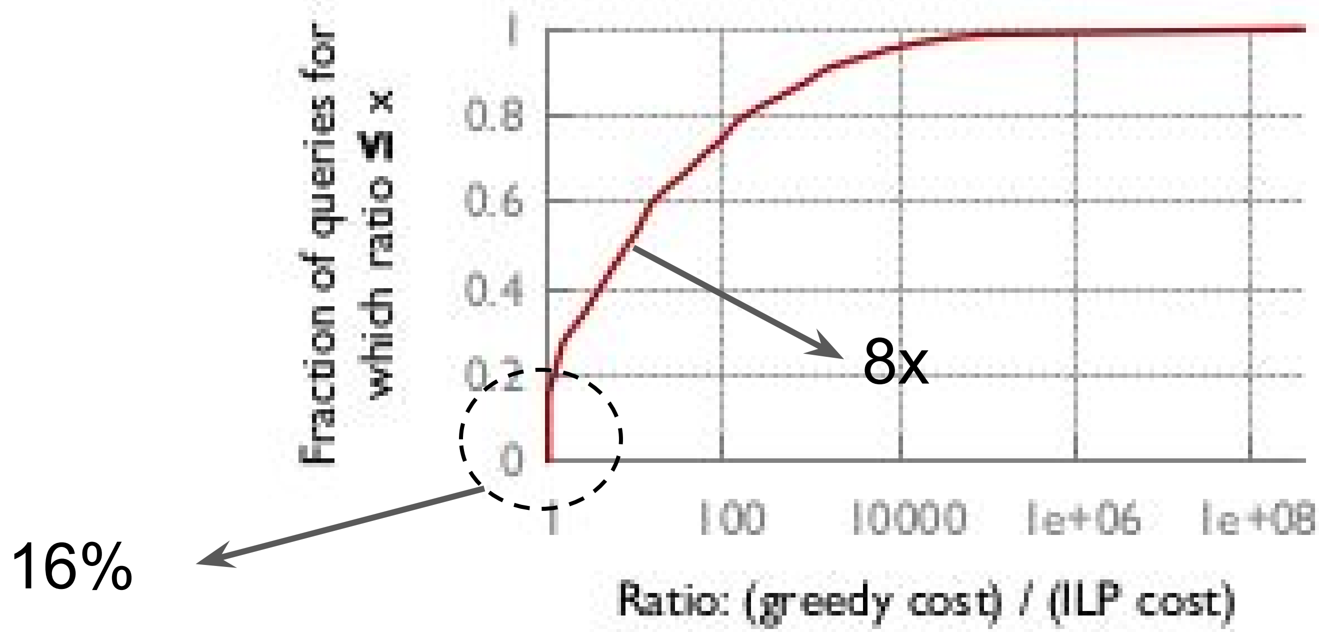
Site Selection and Data Replication

- Given:
  - Logical plan of tasks for each query (DAG)
  - Data transfer costs along each edge
  - Sovereignty and recovery requirements
  - Update rate
- Minimize total bandwidth costs
- Solve:
  - **Site selection**: which data centers should tasks run on and which copy of data should be accessible
  - **Data replication**: which data centers each base data partition should be replicated to (for performance and/or fault tolerance)

ILP vs Greedy Heuristic

- ILP is highly optimized but may be unscalable and slow
- Isolate both problems
  - Site selection
    - Natural greedy task placement
    - Assign **tasks to lowest cost data centers** where possible
  - Data replication
    - Independent and **solvable ILP**
    - Check if replicating would further reduce cost

Evaluation: ILP vs Greedy

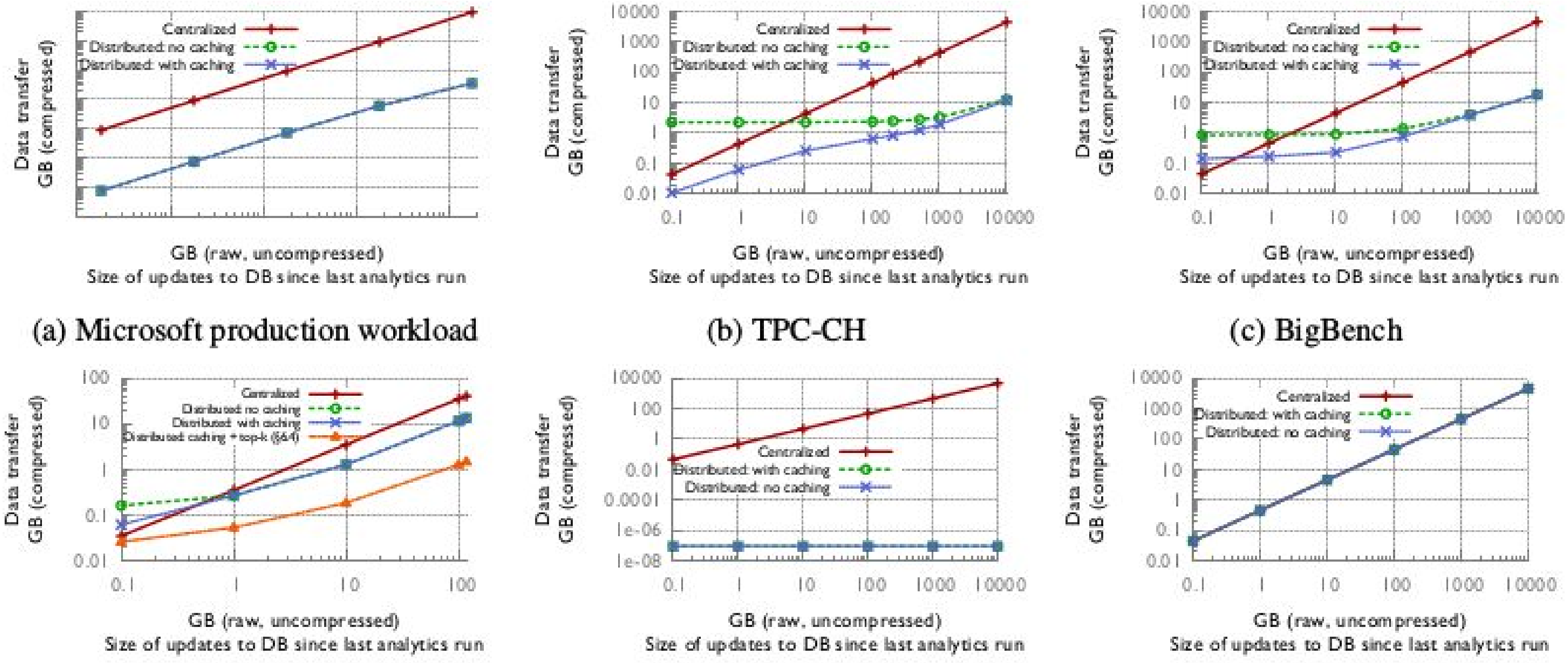


(a) Bandwidth cost ratio on 10k randomly generated queries

- Synthetic query patterns
- ILP scalable to 10 DCs, Greedy scalable to 100
- Real benchmarks: 98% were same

Large Scale Evaluation

257x 360x 330x



(a) Microsoft production workload (b) TPC-CH (c) BigBench

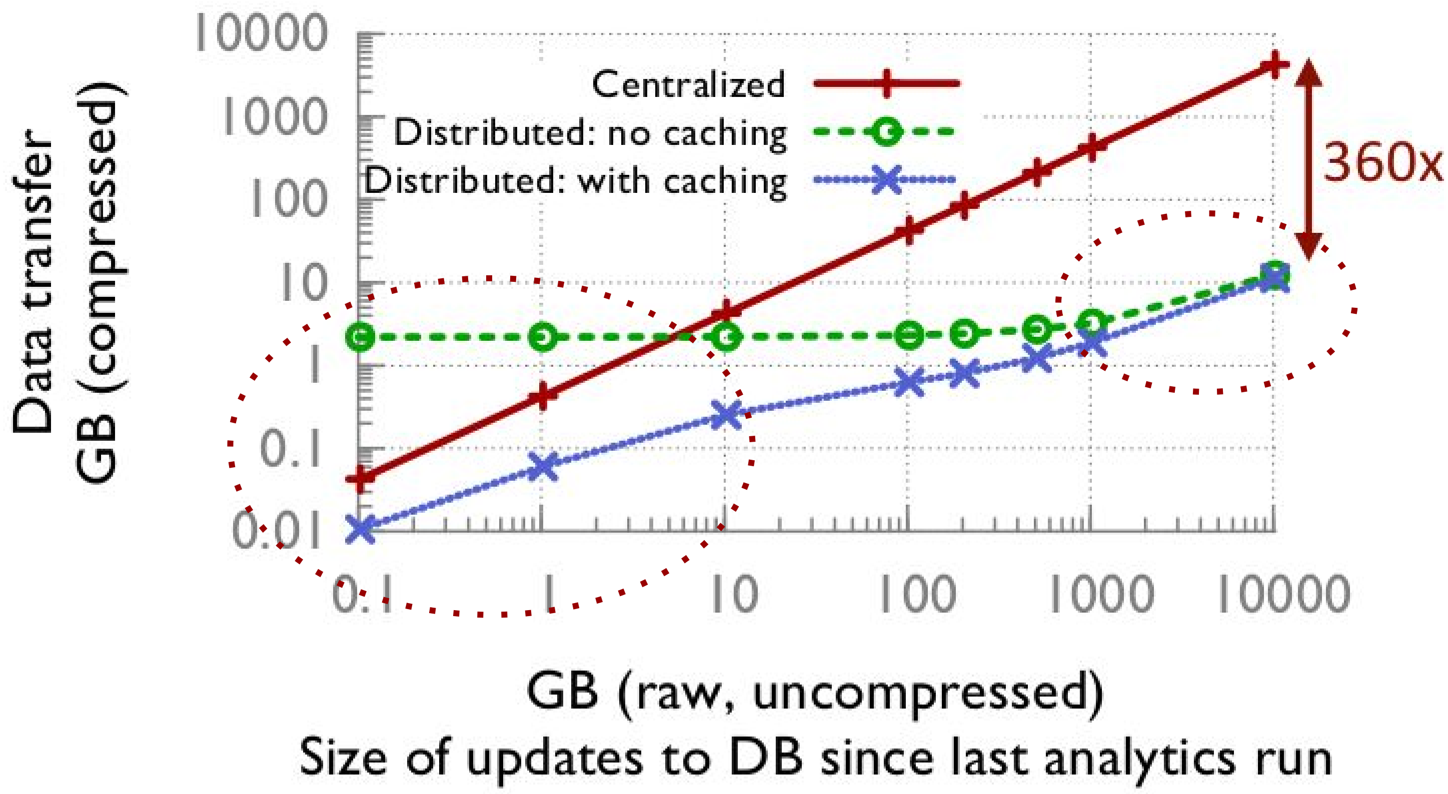
(d) Berkeley big-data (e) YCSB-aggr (f) YCSB-getall

Figure 8: End-to-end evaluation of all six workloads

x-axis: update to database between subsequent queries; y-axis: transfer costs

evaluate: centralized, distributed, distributed+caching

Evaluation: TCP-CH (from slides)



- **centralized better than distributed for low churn**
- **cache is less effective for v. high churn**

Strengths

- Works on relational databases (SQL-like model)
- Extensible to user defined optimizations
- Intermediate caching might result in unexpected gains during cross-DC task assignments
- Profiling latency overhead turns out to be small (<20%)

Weaknesses

- Solves only for relational data model - not extendible to MapReduce type
- Very simplistic uniform bandwidth cost model is assumed
- Only optimizes for bandwidth constraints, not latency
- Relaxed eventual consistency model
- No attempt to preserve privacy as arbitrary queries are allowed as long as sovereignty constraints regarding base data are met

Thanks!

Design: Key Characteristics

1. Support full relational model
2. No control over data partitioning
   - Dictated by external factors, typically end user latency
3. Cross-DC bandwidth is scarcest resource by far
   - CPU, storage etc within data centers are relatively cheap
4. Unique constraints
   - Heterogeneous bandwidth costs/capacities
   - Sovereignty
5. Bulk of load comes from ~stable recurring workload
   - Consistent with production logs