Global analytics in the face of bandwidth and regulatory constraints

Ashish Vulimiri\textsuperscript{u}
Thomas Jungblut\textsuperscript{m}

Carlo Curino\textsuperscript{m}
Jitu Padhye\textsuperscript{m}

Brighten Godfrey\textsuperscript{u}
George Varghese\textsuperscript{m}

\textsuperscript{u}UIUC
\textsuperscript{m}Microsoft
## Massive data volumes

<table>
<thead>
<tr>
<th>Platform</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook</td>
<td>600 TB/day</td>
</tr>
<tr>
<td>Twitter</td>
<td>100 TB/day</td>
</tr>
<tr>
<td>Microsoft</td>
<td>10s TB/day</td>
</tr>
<tr>
<td>LinkedIn</td>
<td>10 TB/day</td>
</tr>
<tr>
<td>Yahoo!</td>
<td>10 TB/day</td>
</tr>
</tbody>
</table>
# Massive data volumes

**Use cases:**
- User activity logs
- Monitoring remote infrastructures
- ...

<table>
<thead>
<tr>
<th>Platform</th>
<th>Daily Data Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook</td>
<td>600 TB/day</td>
</tr>
<tr>
<td>Twitter</td>
<td>100 TB/day</td>
</tr>
<tr>
<td>Microsoft</td>
<td>10s TB/day</td>
</tr>
<tr>
<td>LinkedIn</td>
<td>10 TB/day</td>
</tr>
<tr>
<td>Yahoo!</td>
<td>10 TB/day</td>
</tr>
</tbody>
</table>

Collected across several data centers for low user latency
SQL analytics across geo-distributed data to extract insight

current solution: centralize
- copy all data to central data center
- run all queries there

10s-100s TB/day
up to 10s of DCs
Centralized approach is inadequate

1. Consumes scarce, expensive cross-DC bandwidth
Centralized approach is inadequate

1. Consumes scarce, expensive cross-DC bandwidth

- **rising costs**
  - external network is fastest rising DC cost

- **scarce capacity**
  - total Internet capacity
    - 100 Tb/s
  - some DCs’ internal bisection b/w
    - 1000Tb/s

- **slowing growth**
  - Internet capacity growth (%)
  - 2009: 80%
  - 2014: 20%

- **recognized concern**
  - several other efforts to reduce wide-area traffic
    - e.g. SWAN, B4
Centralized approach is inadequate

1. Consumes scarce, expensive cross-DC bandwidth

2. Incompatible with sovereignty concerns
   - Many countries considering restricting moving citizens’ data
   - Could render centralization impossible
   - Speculation: derived information might still be acceptable

current solution: copy all data to central DC, run all analytics there
Centralized approach is inadequate

1. Consumes scarce, expensive cross-DC bandwidth
2. Incompatible with sovereignty concerns
Geo-distributed SQL analytics

Centralized execution: 10 TB/day

SQL query:

adserve_log → preprocess adserve_log

click_log → preprocess click_log

join → k-means clustering
Geo-distributed SQL analytics

SQL query:

```
adserve_log
```

```
click_log
```

```
preprocess
```

```
join
```

```
k-means clustering
```

Centralized execution: 10 TB/day

Distributed execution: 0.03 TB/day

<table>
<thead>
<tr>
<th>t = 0</th>
<th>t = 1</th>
<th>t = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>push down preprocess</td>
<td>distributed semi-join</td>
<td>centralized k-means</td>
</tr>
</tbody>
</table>
Geo-distributed SQL analytics

Centralized execution: 10 TB/day

Distributed execution: 0.03 TB/day

333x cost reduction
Geo-distributed SQL analytics

Optimizations: synthesize and extend ideas from
- Parallel and distributed databases
- Distributed systems
... as well as novel techniques of our own

Common thread: revisit classical database problems from networking perspective
PROBLEM DEFINITION
Requirements

Possible challenges to address

- **Bandwidth**
- Sovereignty
- Fault-tolerance
- Latency
- Consistency

We target the **batch analytics** dominant in organizations today.
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
Given: data born distributed across DCs a certain way

Goal: support SQL analytics on this data
  - Minimize bandwidth cost
  - Handle fault-tolerance, sovereignty constraints

System will handle arbitrary queries at runtime
  - But will be tuned to optimize known ~stable recurring workload
OUR APPROACH
Basic Architecture

Coordinator

Queries -> Coordinator -> Results

End-user facing DB
(handles OLTP)

Single-DC SQL stack
[Hive]

Local ETL

End-user facing DB
(handles OLTP)

Reporting pipeline
Optimizations

Function-specific

SQL-aware
  workload planning

Runtime
  data transfer reduction

semantic level
Optimizations

3. Function-specific

2. SQL-aware

1. Runtime
Optimizations

1. Runtime

2. SQL-aware

3. Function-specific
In our setting
- CPU, storage, ... within data centers is cheap
- Cross-DC bandwidth is the expensive resource

Trade off CPU, storage for bandwidth reduction
aggressively cache all intermediate output

t = 0 \quad \text{DC}_B \text{ asks } \text{DC}_A \text{ for results of subquery } q
1. Runtime data transfer optimization

aggressively cache all intermediate output
1. Runtime data transfer optimization

aggressively cache all intermediate output

t = 1  \( \text{DC}_B \) asks \( \text{DC}_A \) for results of subquery \( q \) again
1. Runtime data transfer optimization

aggressively cache all intermediate output

t = 1  DC_B asks DC_A for results of subquery q again
1. Runtime data transfer optimization

aggressively cache all intermediate output

recompute $q_1$ from scratch
- not using caching to save latency, CPU
- only bandwidth
1. Runtime data transfer optimization

aggressively cache **all intermediate output**

Caching helps not only when same query arrives repeatedly

... but also when different queries have common sub-operations

e.g. 6x data transfer reduction in TPC-CH
aggressively cache all intermediate output

Caching helps not only when same query arrives repeatedly

... but also when different queries have common sub-operations

e.g. 6x data transfer reduction in TPC-CH

**Database parallel:** caching $\approx$ view materialization

- Caching is a low-level, mechanical form of view maintenance
- Works for arbitrary computations, including arbitrary UDFs
  - Uses more CPU, storage
  - Can miss opportunities
Optimizations

3. Function-specific

2. SQL-aware

1. Runtime
Optimizations

1. Runtime
2. SQL-aware
3. Function-specific
2. SQL-aware workload planning

Given
- Stable workload (set of queries)
- Fault-tolerance and sovereignty constraints

Jointly optimize
- Query plan
- Site selection (task scheduling)
- Data replication
  ‣ Replicate data for performance and/or fault-tolerance
to minimize data transfer cost

Challenge: optimization search space is exponentially large

Approach: simplify search space
2. SQL-aware workload planning

Simplification

Computation: copy both tables to one DC, then join them

Decision 1: do we copy the big table or the small table?
2. SQL-aware workload planning

Simplification

Computation: copy both tables to one DC, then join them

Decision 1: do we copy the big table or the small table?
Computation: copy both tables to one DC, then join them

Decision 1: do we copy the big table or the small table?

Decision 2: which copy of the small table do we use?
Had two kinds of decisions to make:

1. **Logical plan**
   - Do we copy the big table or the small table?

2. **Physical plan**
   - Which copy of the small table do we use?
Had two kinds of decisions to make:

1. **Logical plan**
   - Do we copy the big table or the small table?
   - Choice was clear, strategies were orders of magnitude apart

2. **Physical plan**
   - Which copy of the small table do we use?
   - Choice wasn’t as obvious, had to know precise costs
Simplification: Two-phase approach

1. **Logical plan**
   - Choose based on simple statistics on each table

2. **Physical plan**
   - Profile logical plan, collecting precise measurements
   - Use to optimize physical plan

**Key insight**
- “Logical” choices: simple statistics usually suffice
- “Physical” choices: need more careful cost estimates
- Only an **empirical** insight
  ‣ But worked well in all our experimental workloads
2. SQL-aware workload planning

workload

logical plan

query 1

DAG 1

profiled DAG 1

query n

DAG n

profiled DAG n

schedule tasks to DCs, decide data replication policy

physical plan
2. SQL-aware workload planning

workload

logical plan

physical plan

1. Query planner
2. Profiler
3. Integer Linear Program
2. SQL-aware workload planning

workload

logical plan

query 1

DAG 1

profiled DAG 1

Schedule tasks to DCs, decide data replication policy

query n

DAG n

profiled DAG n

physical plan

1. Query planner

2. Profiler

3. Integer Linear Program
2. SQL-aware workload planning

Profiling task graphs

```
SELECT city, SUM(orderValue)
FROM sales
WHERE category = 'Electronics'
GROUP BY city
```

Distributed deployment:

```
70 GB
60 GB
63 GB
```

want to measure
2. SQL-aware workload planning

**Profiling task graphs**

```sql
SELECT city, SUM(orderValue)
FROM sales
WHERE category = 'Electronics'
GROUP BY city
```

**Distributed deployment:**

- US DC
  - US
  - 70 GB
- UK DC
  - UK
  - 60 GB
- Japan DC
  - JP
  - 63 GB

**Centralized deployment:**

- Inject filter
  - WHERE country = "US"
- Partial aggregate
  - merge
  - US
    - 70 GB
  - UK
    - 60 GB
  - JP
    - 63 GB
Profiling task graphs

**Pseudo-distributed execution**

Rewrite query DAGs to simulate alternate configurations

Fully general what-if analysis. Use cases:
- Bootstrap: centralized -> distributed
- Test alternate data replication strategies
- Simulate adding/removing data centers
2. SQL-aware workload planning

1. Query Planner
2. Profiler
3. Integer Linear Program

schedule tasks to DCs, decide data replication policy

see paper
Optimizations

1. Runtime

2. SQL-aware

3. Function-specific
Optimizations

1. Runtime

2. SQL-aware

3. Function-specific
3. Function-specific optimizations

Past work: large number of distributed algorithms targeting specific problems

Support via extensible user-defined function interface
- Allows registering multiple implementations
- Optimizer will automatically choose best, based on profiling

As examples, implemented
- Top-k \(^1\)
- Approximate count-distinct \(^2\)

\(^1\) “Efficient top-k query computation in distributed networks”
P. Cao, Z. Wang, PODC 2004

\(^2\) “HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm”
P. Flajolet, E. Fusy, O. Gandouet, F. Meunier, AOFA 2007
EVALUATION
Implemented Hadoop-stack prototype
  - Prototype multi-DC replacement for Apache Hive

Experiments up to 10s of TBs scale
  - Real Microsoft production workload
  - Several synthetic benchmarks:
    ‣ TPC-CH
    ‣ BigBench-SQL
    ‣ Berkeley Big-Data
    ‣ YCSB
BigBench-SQL

Data transfer

GB (compressed)

GB (raw, uncompressed)

Size of updates to DB since last analytics run

Centralized
Distributed: no caching
Distributed: with caching

330x
TPC-CH

Data transfer

Size of updates to DB since last analytics run

Centralized
Distributed: no caching
Distributed: with caching

360x
Microsoft production workload

Data transfer (compressed)

Centralized
Distributed: no caching
Distributed: with caching

Size of OLTP updates since last OLAP run (raw, uncompressed)

257x
Berkeley Big-Data

Data transfer

Size of updates to DB since last analytics run

Centralized
Distributed: no caching
Distributed: with caching
Distributed: caching + top-k

3.5x
Berkeley Big-Data

Data transfer

GB (compressed)

GB (raw, uncompressed)

Size of updates to DB since last analytics run

- Centralized
- Distributed: no caching
- Distributed: with caching
- Distributed: caching + top-k

27x
Beyond SQL: DAG workflows

Computational model: directed acyclic task graphs, each node = arbitrary computation

Significantly more challenging setting

Initial results encouraging
  - Same level of improvement as SQL

More details: [CIDR 2015]

RELATED WORK

Distributed and parallel databases

Single-DC frameworks (Hadoop/Spark/...)

Data warehouses

Scientific workflow systems

Sensor networks

Stream processing systems (e.g. JetStream)

...
Key characteristics

1. Support full relational model at 100s TBs/day scale

2. No control over data partitioning

3. Focus on cross-DC bandwidth

4. Unique constraints
   - Heterogeneous bandwidth costs/capacities
   - Sovereignty

5. Assumption of ~stable recurring workload
   - Enables highly tuned optimization
Centralized analytics becoming unsustainable

Geo-distributed analytics: SQL and DAG workflows

Several novel techniques
- Redundancy elimination via caching
- Pseudo-distributed measurement
- [SQL query planner + ILP] optimizer

Up to $360x$ less bandwidth on real & synthetic workloads
THANK YOU!
SUMMARY

Centralized analytics becoming unsustainable

Geo-distributed analytics: SQL and DAG workflows

Several novel techniques
- Redundancy elimination via caching
- Pseudo-distributed measurement
- [SQL query planner + ILP] optimizer

Up to 360x less bandwidth on real & synthetic workloads
BACKUP SLIDES
Consider \( \text{SELECT } \text{val} - \text{avg(val)} \text{ FROM table} \)

Cutpoint selection problem: do we cache
- Base \([\text{val}], \text{or}\)
- Results after average has been subtracted

Akin to view selection problem in SQL databases

Current implementation makes wrong choice
Sovereignty: Partial support

Our system respects data-at-rest regulations (e.g. German data should not leave Germany)

But we allow arbitrary queries on the data

Limitation: we don’t differentiate between

- Acceptable queries, e.g.
  “what’s the total revenue from each city”
- Problematic queries, e.g.
  SELECT * FROM Germany
Sovereignty: Partial support

Solution: either
- Legally vet the core workload of queries
- Use differential privacy mechanism

Open problem
Past work