Low-Latency Geo-Distributed Data Analytics (Iridium)

...

Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik,
Srikanth Kandula, Aditya Akella, Paramvir Bahl, Ion Stoica

SIGCOMM ‘15 (Aug 17-21)

Nicholas Turner

Problem Statement

- Many industries have use cases where they would like to execute a query across data which is distributed over a large geographical area

Problem Statement

- Many industries have use cases where they would like to execute a query across data which is distributed over a large geographical area
  - some examples include
    - determining global properties of a service across users - median response time
    - whole cluster system log queries

Problem Statement

- Many industries have use cases where they would like to execute a query across data which is distributed over a large geographical area

- The common approach to doing this involves copying ALL of the relevant data to a single data center, or to spread the work ‘evenly’ across datacenters

Problem Statement

- Many industries have use cases where they would like to execute a query across data which is distributed over a large geographical area

- The common approach to doing this involves copying ALL of the relevant data to a single data center, or to spread the work ‘evenly’ across datacenters
  - but these methods can be time inefficient, or have high ***query response time*** when datacenter qualities are variable
  - This is particularly undesirable when the results of these queries are used directly by operators (analysts), or fuel interactive services

Core Ideas

- Can we instead process the data in a globally distributed way, assigning the heavy lifting of analytics queries intelligently?

Core Ideas

- Can we instead process the data in a globally distributed way, assigning the heavy lifting of analytics queries intelligently?

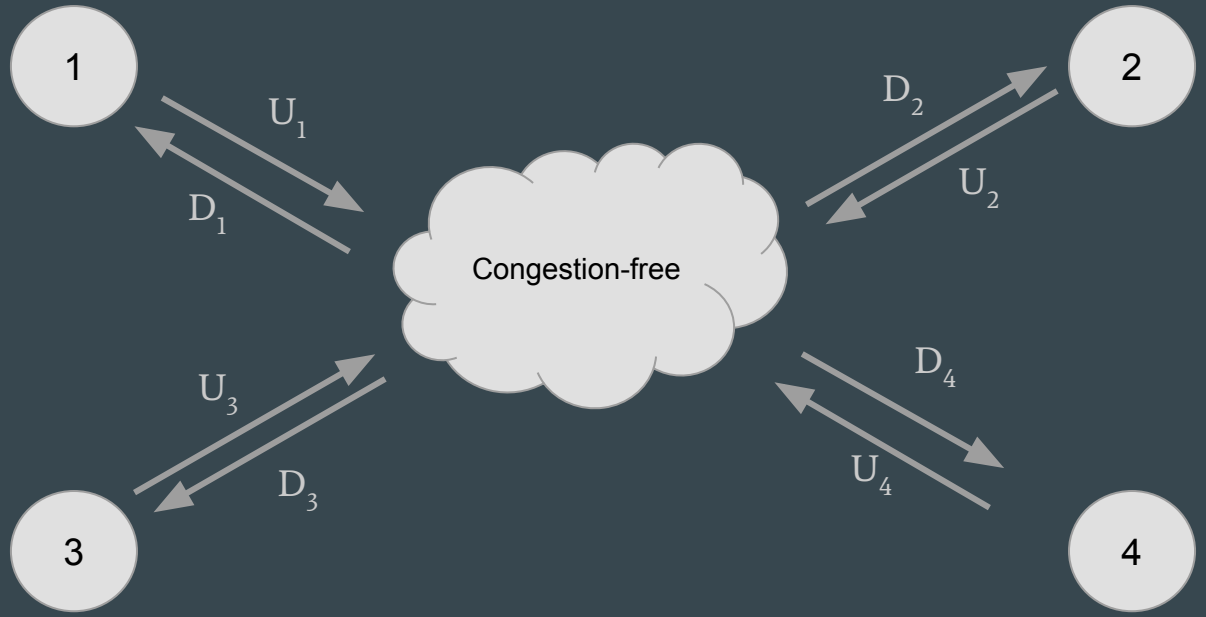
- Can we shuffle data around preemptively so that future queries are faster?

Further Illustration - Model Queries

- Consider a MapReduce query where
  - nodes communicate via a ***congestion-free core*** (communication bottlenecks between site and core)

Further Illustration - Model Queries

- Consider a MapReduce query where
  - nodes communicate via a ***congestion-free core*** (communication bottlenecks lie between site and core)



Further Illustration - Model Queries

- Consider a MapReduce query where
  - nodes communicate via a ***congestion-free core***_ (communication bottlenecks lie between site and core)
  - IO and CPU tasks take 0 time to execute (or at least that their time cost is negligible compared to communication latencies

Further Illustration - Model Queries

- Consider a MapReduce query where
  - nodes communicate via a ***congestion-free core*** (communication bottlenecks lie between site and core)
  - IO and CPU tasks take 0 time to execute (or at least that their time cost is negligible compared to communication latencies
  - Computation is split across three sites with the following characteristics

	Site-1	Site-2	Site-3
Input Data (MB), *I*	300	240	240
Intermediate Data (MB), *S*	150	120	120
Uplink (MB/s), *U*	10	10	10
Downlink (MB/s), *D*	1	10	10

Further Illustration - Model Queries

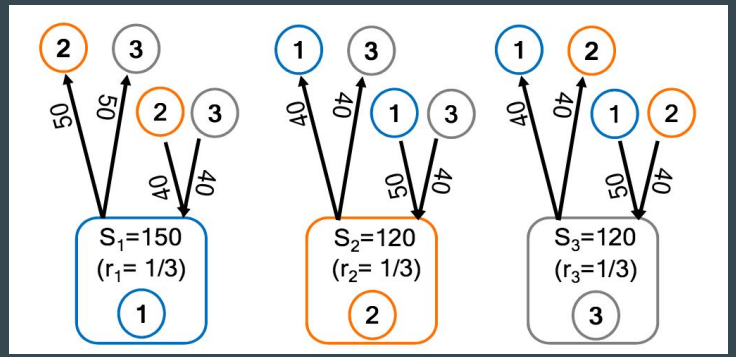
- Consider a MapReduce query where
  - nodes communicate via a ***congestion-free core*** (communication bottlenecks lie between site and core)
  - IO and CPU tasks take 0 time to execute (or at least that their time cost is negligible compared to communication latencies
  - Computation is split across three sites with the following characteristics

	Site-1	Site-2	Site-3
Input Data (MB), *I*	300	240	240
Intermediate Data (MB), *S*	150	120	120
Uplink (MB/s), *U*	10	10	10
Downlink (MB/s), *D*	1	10	10

Further Illustration - Model Queries

- Consider a MapReduce query where
  - nodes communicate via a ***congestion-free core*** (communication bottlenecks lie between site and core)
  - IO and CPU tasks take 0 time to execute (or at least that their time cost is negligible compared to communication latencies
  - Computation is split across three sites with the following characteristics
  - Sites shuffle data in an 'all-to-all' fashion - always sending a percentage of their data proportional to the number of reduce tasks on the destination site

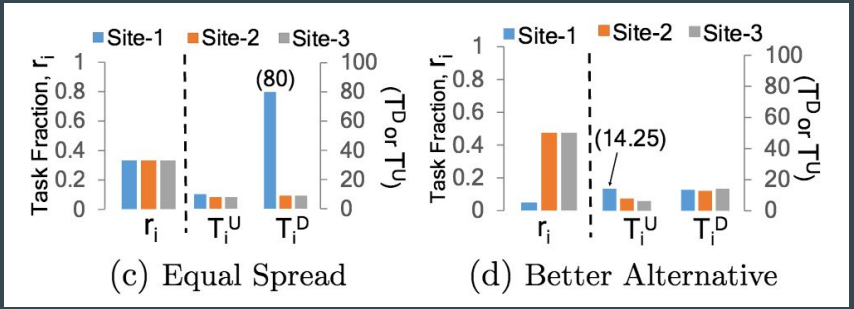
	Site-1	Site-2	Site-3
Input Data (MB), *I*	300	240	240
Intermediate Data (MB), *S*	150	120	120
Uplink (MB/s), *U*	10	10	10
Downlink (MB/s), *D*	1	10	10



Further Illustration - Model Queries

- Consider a MapReduce query where
  - nodes communicate via a ***congestion-free core*** (communication bottlenecks lie between site and core)
  - IO and CPU tasks take 0 time to execute (or at least that their time cost is negligible compared to communication latencies
  - Computation is split across three sites with the following characteristics
  - Sites shuffle data in an 'all-to-all' fashion - always sending a percentage of their data proportional to the number of reduce tasks on the destination site

	Site-1	Site-2	Site-3
Input Data (MB), *I*	300	240	240
Intermediate Data (MB), *S*	150	120	120
Uplink (MB/s), *U*	10	10	10
Downlink (MB/s), *D*	1	10	10



Technical Design - Task Placement

- Assuming the model above (and that tasks are infinitesimally divisible), you can find an optimal solution for a single reduce task by the following a linear program

$$S = \sum_i S_i$$

$$T_i^U(r_i) = \frac{(1 - r_i)S_i}{U_i}$$

$$T_i^D(r_i) = \frac{r_i(S - S_i)}{D_i}$$

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & \forall i : r_i \geq 0 \\ & \textstyle\sum_i r_i = 1 \\ & \forall i : T_i^U(r_i) \leq z, \quad T_i^D(r_i) \leq z \end{aligned}$$

Technical Design - Task Placement

- Assuming the model above (and that tasks are infinitesimally divisible), you can find an optimal solution for a single reduce task by the following a linear program
- In practice, they relax the divisibility assumption above by using a Mixed Integer Program instead (which takes a bit more time)

$$S = \sum_i S_i$$

$$T_i^U(r_i) = \frac{(1 - r_i)S_i}{U_i}$$

$$T_i^D(r_i) = \frac{r_i(S - S_i)}{D_i}$$

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & \forall i : r_i \geq 0 \\ & \textstyle\sum_i r_i = 1 \\ & \forall i : T_i^U(r_i) \leq z, \quad T_i^D(r_i) \leq z \end{aligned}$$

Technical Design - Task Placement

- Assuming the model above (and that tasks are infinitesimally divisible), you can find an optimal solution for a single reduce task by the following a linear program
- In practice, they relax the divisibility assumption above by using a Mixed Integer Program instead (which takes a bit more time)
- They also approximate the solution for multiple linked tasks (DAGs) greedily

$$S = \sum_i S_i$$

$$T_i^U(r_i) = \frac{(1 - r_i)S_i}{U_i}$$

$$T_i^D(r_i) = \frac{r_i(S - S_i)}{D_i}$$

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & \forall i : r_i \geq 0 \\ & \textstyle\sum_i r_i = 1 \\ & \forall i : T_i^U(r_i) \leq z, \quad T_i^D(r_i) \leq z \end{aligned}$$

Technical Design - Data Placement

- The terms in the last equations depended heavily upon S_i (which in turn depend on I_i), if we knew which queries were coming, we could optimize these terms to find better solutions (though the theoretical formulation isn't tractable here).

$$S = \sum_i S_i$$

$$T_i^U(r_i) = \frac{(1 - r_i)S_i}{U_i}$$

$$T_i^D(r_i) = \frac{r_i(S - S_i)}{D_i}$$

Technical Design - Data Placement

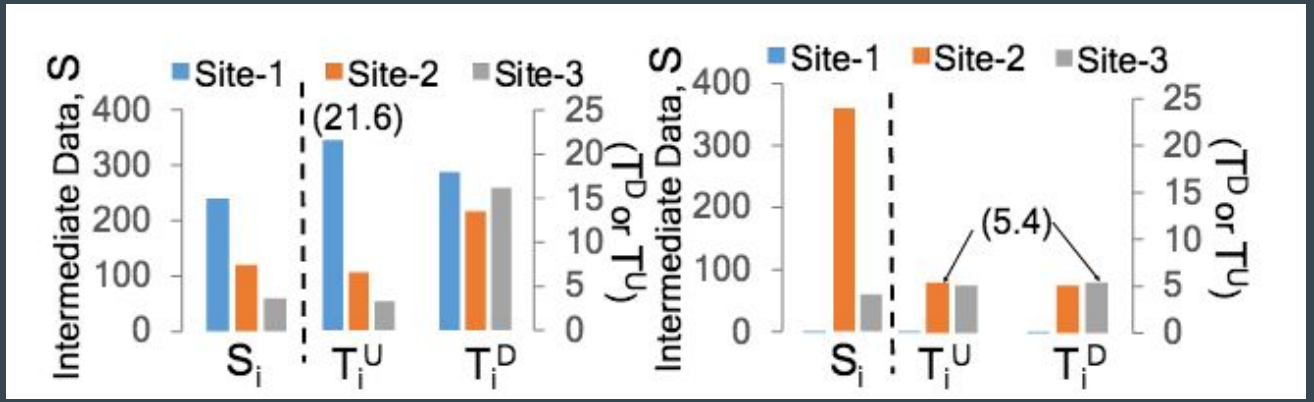
- The terms in the last equations depended heavily upon S_i, if we knew which queries were coming, we could optimize these terms to find better solutions (though the theoretical formulation isn't tractable here).

	Site-1	Site-2	Site-3
Input Data (MB), *I*	240	120	60
Intermediate Data (MB), *S*	240	120	60
Uplink (MB/s), *U*	10	10	10
Downlink (MB/s), *D*	1	10	10

$$S = \sum_i S_i$$

$$T_i^U(r_i) = \frac{(1 - r_i)S_i}{U_i}$$

$$T_i^D(r_i) = \frac{r_i(S - S_i)}{D_i}$$



Technical Design - Data Placement

- The conceptual lesson from the last example is that data shouldn’t reside on ‘bottleneck’ sites when possible

Technical Design - Data Placement

- The conceptual lesson from the last example is that data shouldn't reside on 'bottleneck' sites when possible

- Luckily, the LP is efficient, and it tells us which sites are bottleneck nodes

$$
\begin{aligned}
\min \quad & z \\
\text{s.t.} \quad & \forall i : r_i \geq 0 \\
& \textstyle\sum_i r_i = 1 \\
& \forall i : T_i^U(r_i) \leq z, \quad T_i^D(r_i) \leq z
\end{aligned}
$$

Technical Design - Data Placement

- The conceptual lesson from the last example is that data shouldn't reside on 'bottleneck' sites when possible

- Luckily, the LP is efficient, and it tells us which sites are bottleneck nodes

- So, although we can't find an exact solution, we can take a greedy approach by considering data movements in small increments - frequently running the LP to see the effects on the eventual query response time

Technical Design - Multiple Datasets

- None of this is useful in real data centers unless you're able to prioritize which datasets to optimize at a given point in time

Technical Design - Multiple Datasets

- None of this is useful in real data centers unless you're able to prioritize which datasets to optimize at a given point in time
- Fortunately, we can do this by iterating over all datasets, and consolidating the query frequency of each query for a particular dataset

```
1: procedure AllocateMoves(List⟨Dataset⟩ D)
2:     for each Dataset d in D do
3:         Move d.m ← FindMove(d)
4:         lag ← ∑_{q∈d.Queries} q.lag / d.Queries.Count
5:         d.value ← ∑_{q∈d.Queries} d.m.timeReduction[q ] /
   lag
6:         d.score ← d.value / d.m.cost
7:     for each Dataset d in D.SortedByDesc(d.score) do
8:         if d.m.bottleneck.canMove() then
9:             execute d.m
```

Technical Design - Other Features

- Can impose a tunable ***WAN-usage Budget*** by first running a WAN-optimized strategy, and then only moving data if it fits under some multiplicative factor of the usage of that strategy (notably, 1.3 factor yields 90% of the performance gains)

- Using ***query/data contention*** they only move a data packet if it won't upset currently running queries too much (also estimated using the same greedy methods)

Implementation

- Written as a modified version of Spark (~MapReduce successor) using Hadoop Distributed File System
  - Measure bandwidth of each site and intermediate data size by empirical estimation from data throughput and task flows

Evaluations

- 8 Globally distributed EC2 instances
  - in Tokyo, Singapore, Sydney, Frankfurt, Ireland, Sao Paulo, Virginia, and California
  - also use 30 instances within one region with tuned bandwidth distributions

Evaluations

- 8 Globally distributed EC2 instances
  - in Tokyo, Singapore, Sydney, Frankfurt, Ireland, Sao Paulo, Virginia, and California
  - also use 30 instances within one region with tuned bandwidth distributions
- Run query workload benchmarks from 4 different applications
  - ***Conviva Video Analytics*** - queries from a video delivery and monitoring company
  - ***Microsoft Bing Edge Distribution*** - Edge server streaming queries
  - ***TPC-DS Benchmark*** - Decision support queries modelled after Amazon
  - ***AMPLab Big-Data*** - combination of Hive and Spark queries using the same schema throughout

Evaluations

- 8 Globally distributed EC2 instances
  - in Tokyo, Singapore, Sydney, Frankfurt, Ireland, Sao Paulo, Virginia, and California
  - also use 30 instances within one region with tuned bandwidth distributions
- Run query workload benchmarks from 4 different applications
  - ***Conviva Video Analytics*** - queries from a video delivery and monitoring company
  - ***Microsoft Bing Edge Distribution*** - Edge server streaming queries
  - ***TPC-DS Benchmark*** - Decision support queries modelled after Amazon
  - ***AMPLab Big-Data*** - combination of Hive and Spark queries using the same schema throughout
- Also perform a trace-driven simulation of Facebook's Hadoop cluster

Evaluations

- 8 Globally distributed EC2 instances
  - in Tokyo, Singapore, Sydney, Frankfurt, Ireland, Sao Paulo, Virginia, and California
  - also use 30 instances within one region with tuned bandwidth distributions
- Run query workload benchmarks from 4 different applications
  - ***Conviva Video Analytics*** - queries from a video delivery and monitoring company
  - ***Microsoft Bing Edge Distribution*** - Edge server streaming queries
  - ***TPC-DS Benchmark*** - Decision support queries modelled after Amazon
  - ***AMPLab Big-Data*** - combination of Hive and Spark queries using the same schema throughout
- Also perform a trace-driven simulation of Facebook's Hadoop cluster
- Compare against two baseline strategies:
  - leaving data 'in-place' & centralization strategy (with very high in-DC bandwidth)
  - both use Spark's standard scheduling routines

Evaluations

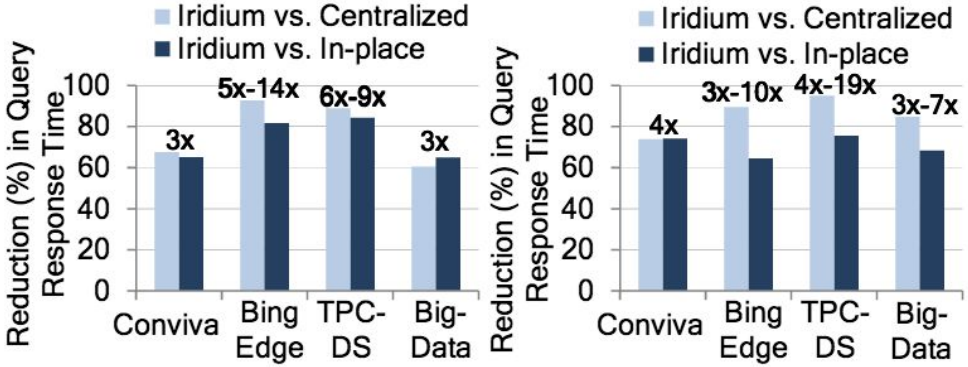


Figure 5: **EC2 Results across eight worldwide regions (a): Tokyo, Singapore, Sydney, Frankfurt, Ireland, Sao Paulo, Virginia (US) and California (US). The figure on the right (b) is on a larger 30-site setup. Iridium is $3\times - 19\times$ better compared to the two baselines.**

Evaluations

	Iridium vs. In-place	Iridium vs. Centralized
Core	26%	32%
Core + Query Lag	41%	46%
Core + Query Lag + Contention	59%	74%
Core + Contention	45%	53%

Table 2: **Progression of Iridium's gains as additional features of considering query lag and contention between query/data movements are added to the basic heuristic. (Facebook workload)**

Evaluations

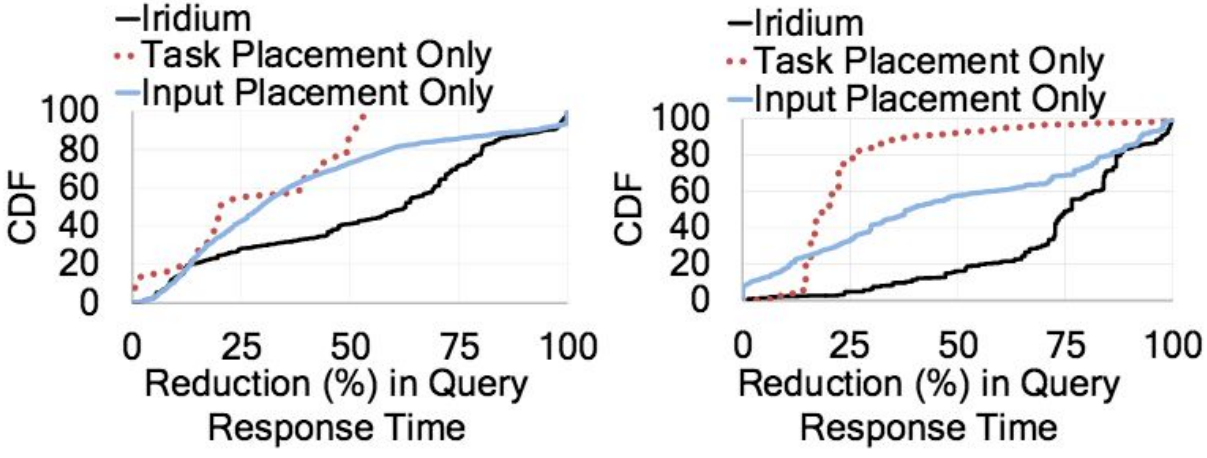


Figure 6: **CDF of Iridium's gains with the Facebook workload. We also compare our two techniques—task placement and data placement—standalone.**

Evaluations

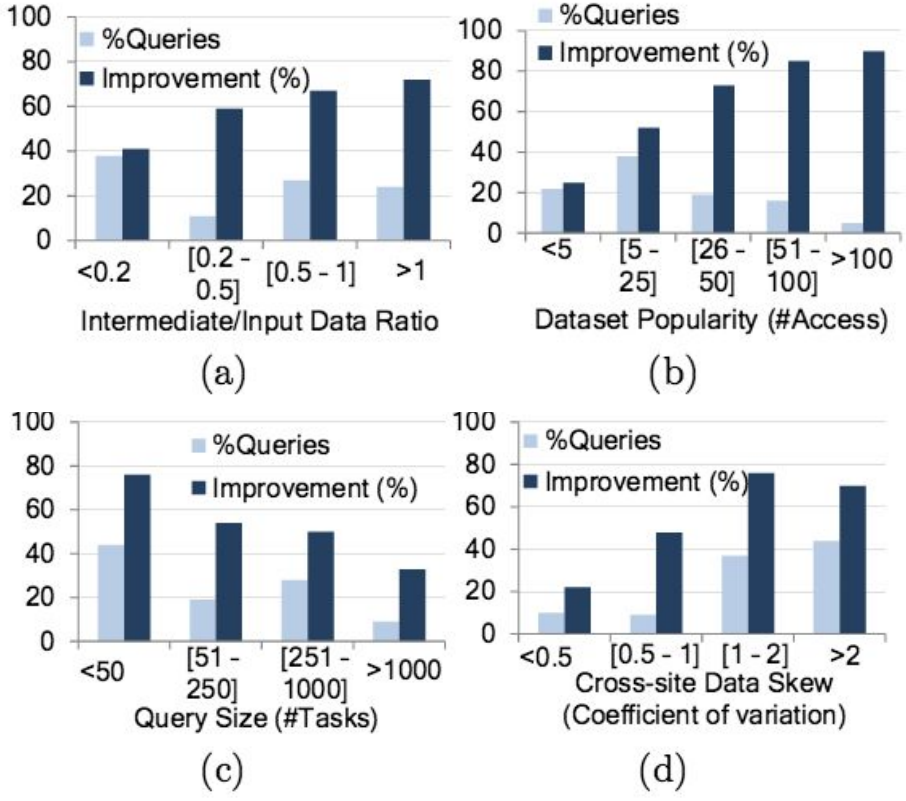


Figure 7: Iridium's improvements (and % queries), bucketed by various query characteristics: (a) intermediate/input data ratio, (b) dataset access count, (c) query size (# tasks), and (d) cross-site skew in intermediate data.

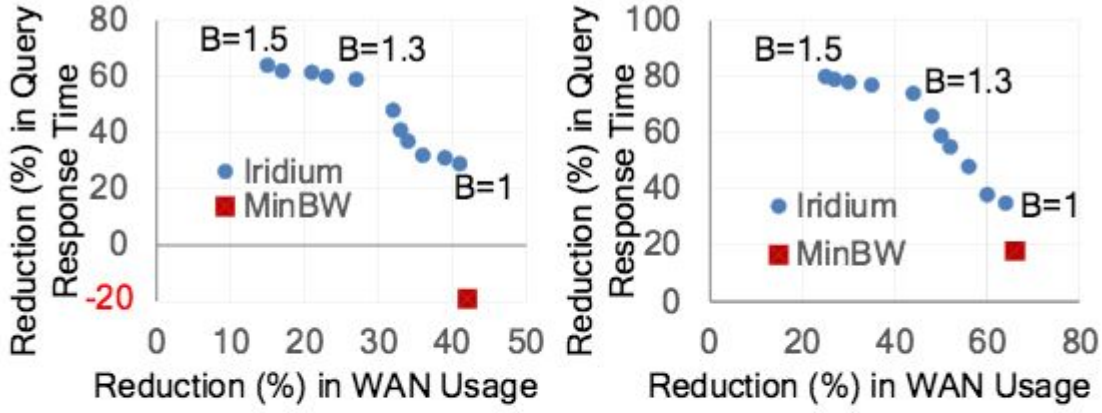
Evaluations

Lag Metric	Vs. In-place	Vs. Centralized
Iridium **(Avg.)**	59%	74%
Iridium (Median)	56%	75%
Iridium (Earliest)	38%	42%
Iridium (Latest)	24%	40%
Oracle	66%	81%

Table 3: **Effectiveness of estimating query lag. Iridium's approach of using the average lag outperforms other options and crucially, has gains of $\sim 90\%$ of an oracle that has full knowledge about query arrivals.**

```
    class Move
        double cost
        ⟨QueryID, double⟩ timeReduction
        Site bottleneck
1:  procedure AllocateMoves(List⟨Dataset⟩ D)
2:      for each Dataset d in D do
3:          Move d.m ← FindMove(d)
4:          lag ← Σ_{q∈d.Queries} q.lag / d.Queries.Count
5:          d.value ← Σ_{q∈d.Queries} d.m.timeReduction[q ] /
    lag
6:          d.score ← d.value / d.m.cost
7:      for each Dataset d in D.SortedByDesc(d.score) do
8:          if d.m.bottleneck.canMove() then
9:              execute d.m
```

Evaluations



(a) In-place baseline (b) Centralized baseline

Figure 8: **WAN Bandwidth Usage knob, B. MinBW is the scheme that optimizes for WAN bandwidth usage. Even with same WAN usage as MinBW ($B = 1$), Iridium's gains in query response time are significantly higher. MinBW slows down queries against the in-place baseline.**

Strengths

- Provide significant performance gains in a zone which was seemingly poorly optimized beforehand
- Avoids blindly optimizing one metric, and allows the user to tune their preferences (WAN knob)

- Very comprehensive and relevant evaluations, which effectively probe how the system works
- Intuition models were effective ways to communicate their ideas

Places for Improvement

- Greedy strategy seems to have clear negative effects on their current performance in some places

- Haven't properly modelled tasks which take a significant amount of time

- Assumes very large storage and compute capacities, which limit their relevant applications

Low-Latency Geo-Distributed Data Analytics

...

Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik,
Srikanth Kandula, Aditya Akella, Paramvir Bahl, Ion Stoica

SIGCOMM ‘15 (Aug 17-21)

Nicholas Turner