ETH zürich

Pay One, Get Hundreds for Free:
Reducing Cloud Costs through Shared Query Execution

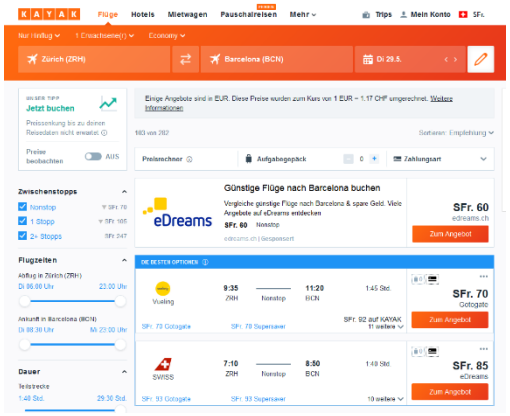
Renato Marroquín, Ingo Müller, Darko Makreshanski, Gustavo Alonso

Motivation: Query-as-a-service

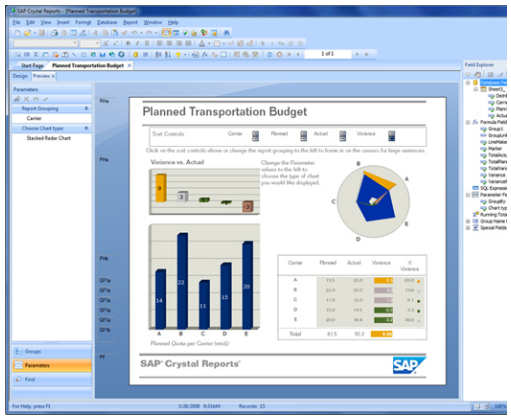
- Pros
  - No need to move data outside the cloud
  - No infrastructure deployment
  - No database maintenance
- Pricing model
  - Pay-per-byte-processed
- Popular Systems

- Disadvantages
  - Expensive with frequent usage
  - No intuitive way to optimize
    - Cost
    - Throughput

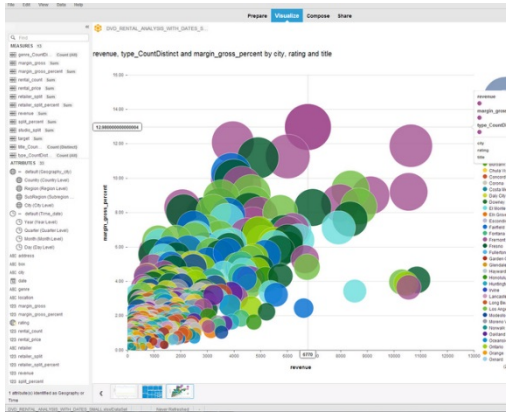
Motivation: Use cases



Parameter exploration



BI reporting



Ad-hoc analytics

- Observations:
  - High overlap among concurrent queries
  - Query burst from single user
  - Similar workload from multiple users

Work sharing: carry out redundant work only once.

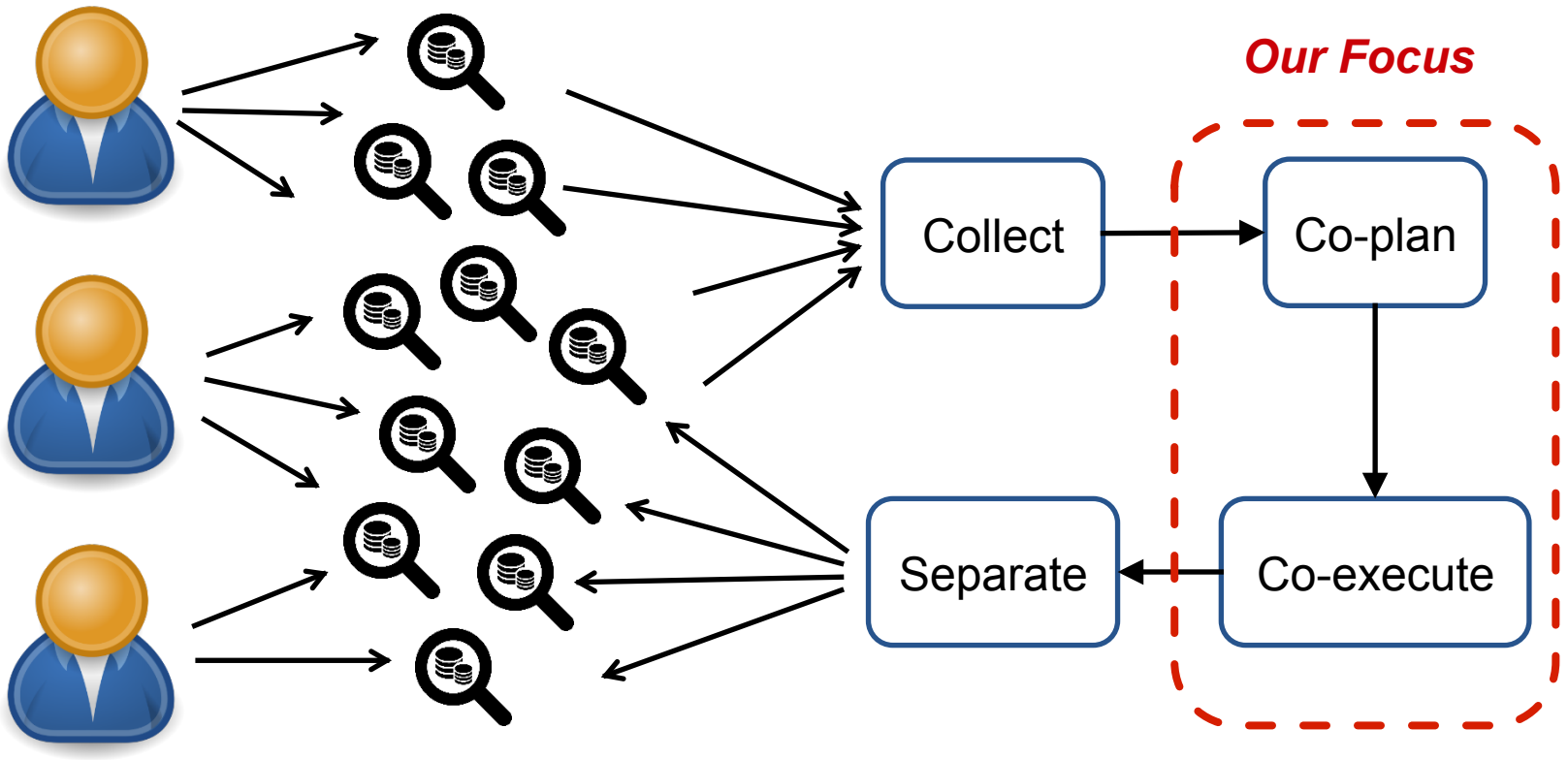
Our Goal

Reduce query-as-a-service ***costs*** using
work sharing techniques through ***query rewriting***

Outline

- Related work

Multi Query Execution: General sharing workflow



Shared execution techniques

1st generation: Crescando

2nd generation: SharedDB, MQJoin, BatchDB

- Annotate tuples with `query_id` attribute
- Supports all relational operators

Name	Age	Sex	query_id
Renato	30	m	Q1,Q3
Darko	30	m	Q1,Q2,Q3
Ingo	32	m	Q1

Shared execution techniques

- Previous approaches need dedicated engine
  - High implementation effort
  - Needs vendor support

This work: rewrite shared query plans as SQL.

Outline

- Related work
- Multi-query execution

Multi Query Execution: Data-query model

```
SELECT * FROM Emp WHERE …

Q1: … age < 40
Q2: … name LIKE '%k%'
Q3: … age <= 30 AND age > 20
```

Name	Age	Sex
Renato	30	m
Darko	30	m
Ingo	32	m
Gustavo	50	m

Name	Age	Sex	query_id
Renato	30	m	{ 1, 3 }
Darko	30	m	{ 1, 2, 3 }
Ingo	32	m	{ 1 }
~~Gustavo~~	~~50~~	~~m~~	~~{ }~~

Multi Query Execution: The `query_id` attribute

Name:	Age:	Sex:	query_id:
Renato	30	m	{ 1, 3 }
Darko	30	m	{ 1, 2, 3 }
Ingo	32	m	{ 1 }

- Result of Qx is given by

```
SELECT * FROM R WHERE x = ANY(query_id)
```

- This is standard SQL!

Shared operators: Shared scan

```
SELECT * FROM Emp WHERE …

Q1: … age < 40
Q2: … name LIKE ‘%k%’
Q3: … age <= 30 AND age > 20
```

Name	Age	Sex
Renato	30	m
Darko	30	m
Ingo	32	m
Gustavo	50	m

```
SELECT *,
   ARRAY_REMOVE(
      ARRAY[
         CASE WHEN age < 40                    THEN 1 ELSE 0 END
         CASE WHEN name LIKE ‘%k%’             THEN 2 ELSE 0 END
         CASE WHEN age <= 30 AND age > 20 THEN 3 ELSE 0 END
      ]
      , 0)
FROM Emp
WHERE (age < 40) OR (name LIKE ‘%k%’) OR (age <= 30 AND age > 20)
```

Shared operators: Shared join

Employees

Name	Age	Sex	D_ID	query_id
Renato	30	m	1	{ 1, 3 }
Darko	30	m	1	{ 1, 2, 3 }
Ingo	32	m	1	{ 1 }

Departments

D_ID	D_Name	query_id
1	Systems	{ 2, 3 }
2	Algorithms	{ 2 }
3	ML	{ 1, 2, 3 }

```
SELECT * FROM Emp e
JOIN Dep d ON e.did = d.did
WHERE …
```

Name	Age	Sex	D_ID	D_Name	query_id
Renato	30	m	1	Systems	{ 3 }
Darko	30	m	1	Systems	{ 2, 3 }

Shared operators: Shared join

Employees

Name	Age	Sex	D_ID	query_id
Renato	30	m	1	{ 1, 3 }
Darko	30	m	1	{ 1, 2, 3 }
Ingo	32	m	1	{ 1 }

Departments

D_ID	D_Name	query_id
1	Systems	{ 2, 3 }
2	Algorithms	{ 2 }
3	ML	{ 1, 2, 3 }

```
WITH shared_emp AS (…),    -- shared scan on emp
     shared_dep AS (…),    -- shared scan on dep
shared_join_helper AS (
   SELECT
      R.name, R.age, R.sex, S.d_id, S.d_name
      ARRAY_INTERSECT(R.query_id, S.query_id) AS query_id
   FROM shared_emp e JOIN shared_dep d ON e.d_id = d.d_id
SELECT * FROM shared_join_helper
WHERE CARDINALITY(query_id) > 0
```

Shared operators: Other shared operators

- `GROUP BY`
  - Use *UNNEST* to duplicate tuples for each query
  - Then group by `query_id`
- `LIMIT` / `TOP K`
  - `PARTITION BY query_id`
  - Then filter by `RANK()`

Enough to express all TPC-H queries!

Outline

- Related work
- Multi-query execution
- Evaluation
- Summary

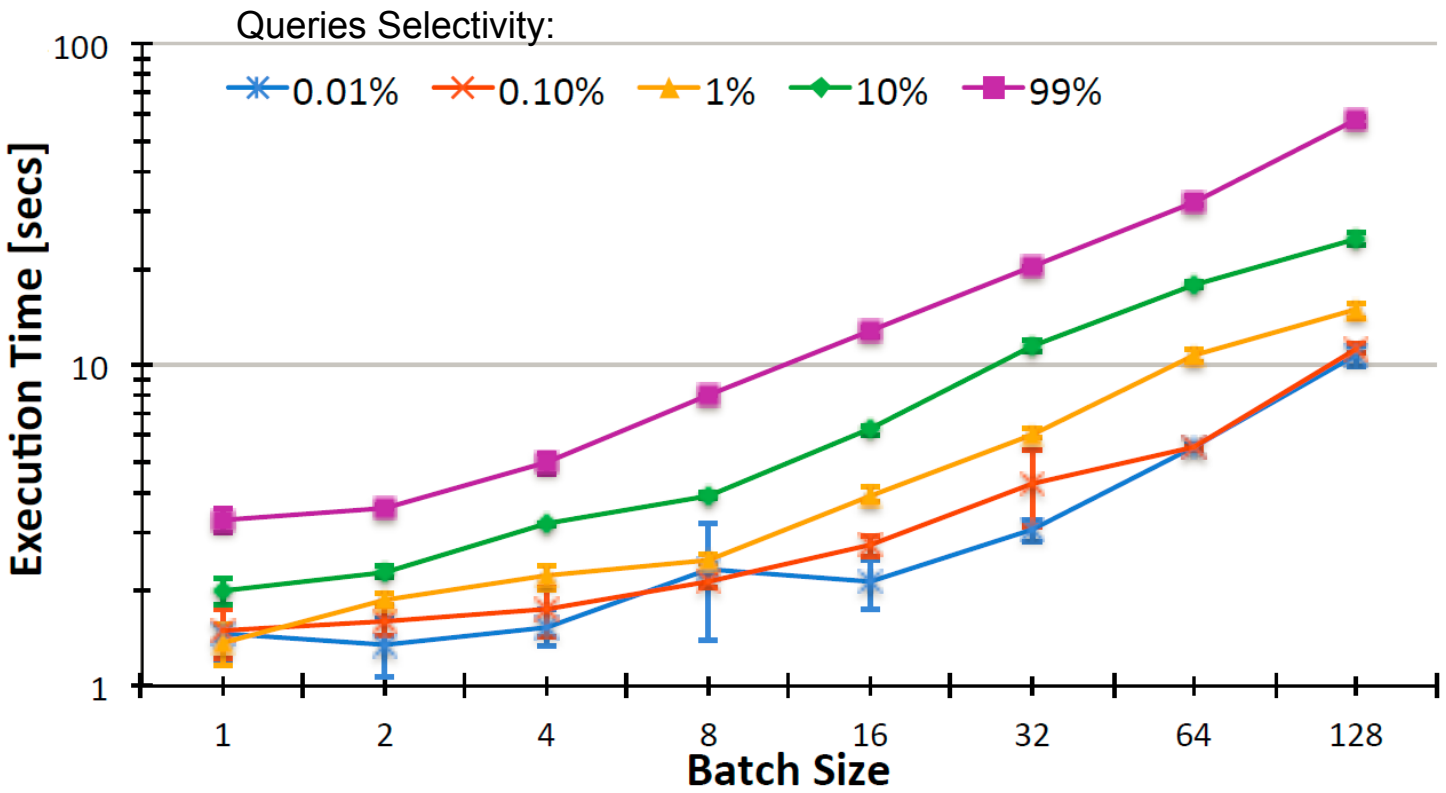
Evaluation: Query-as-a-Service systems

- Systems-under-test
  - Amazon Athena
  - Google BigQuery
- Run SQL queries against files in cloud storage
  - Apache Parquet
  - Google internal columnar storage
- Micro-benchmarks
- End-to-end query execution

Evaluation: Shared scan

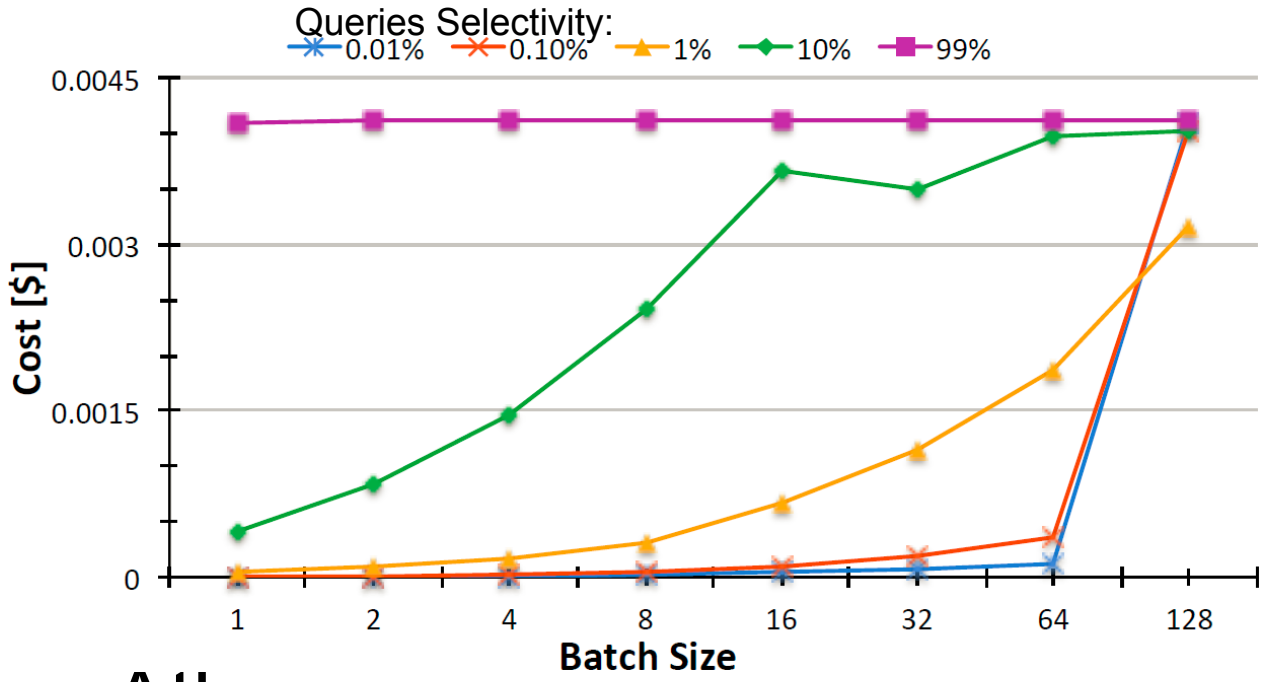
- Data
  - Lineitem from TPC-H Scale Factor 100
- Workload
  - Parameterized queries
  - Queries in a group/batch have equal selectivity

Evaluation: Shared scan



Execution time increases sublinearly with query count.

Evaluation: Shared scan



Amazon Athena

27GB, Parquet format

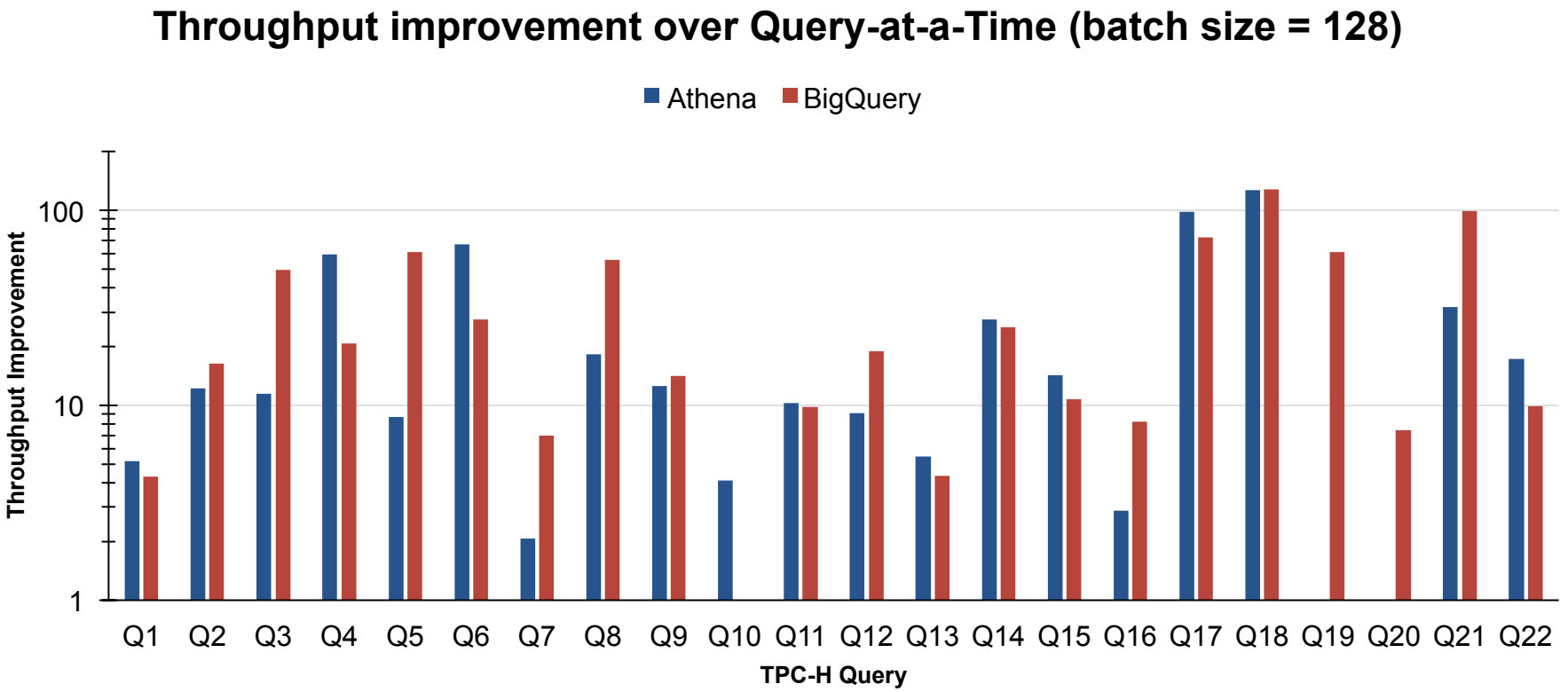
- Pay per processed byte

Monetary cost (almost) independent of batch size.

Evaluation: TPC-H

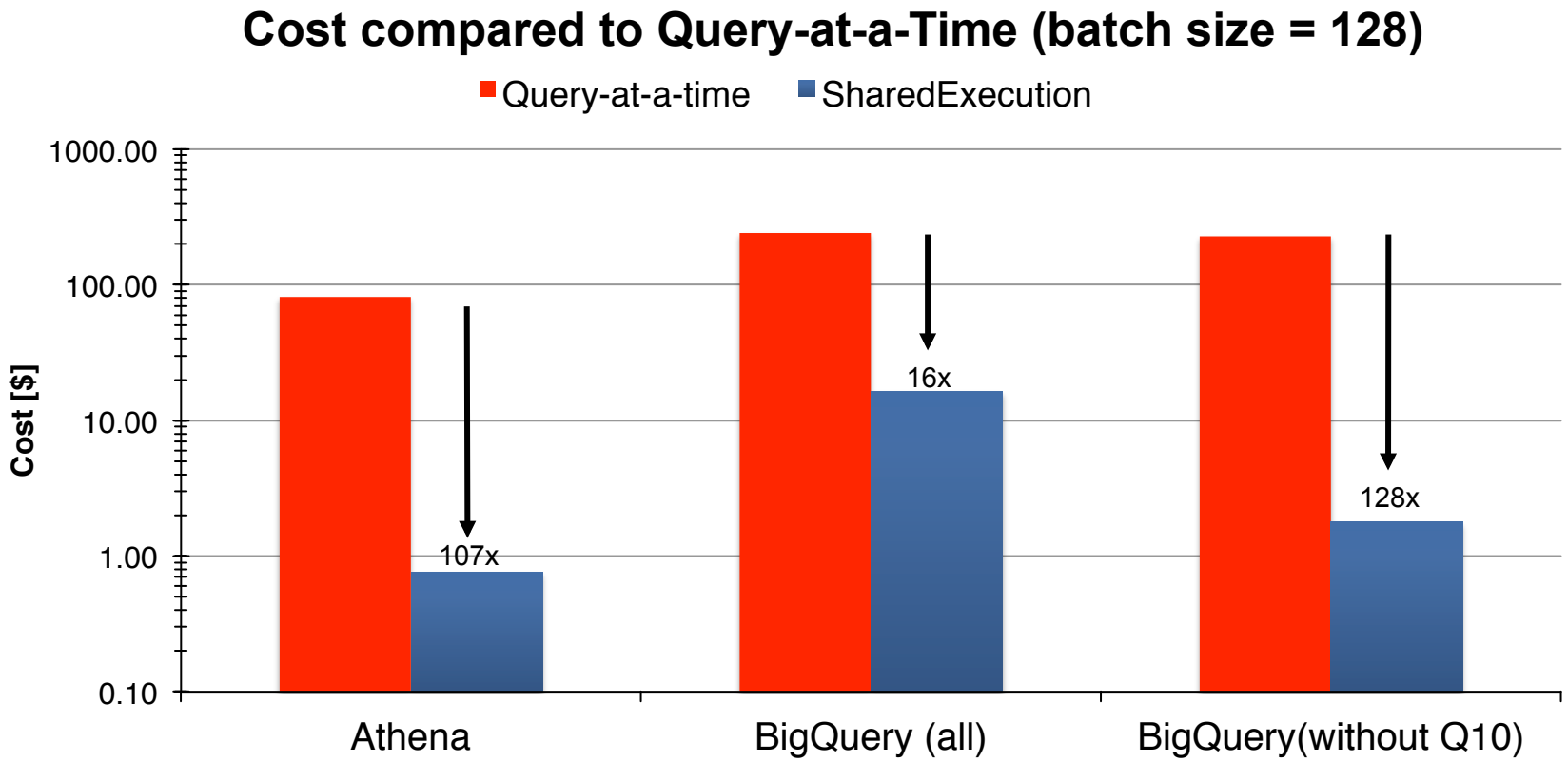
- Data
  - TPC-H Scale Factor 100
- Simulate multiple apps interacting with QaaS
  - Each app emitting different TPC-H queries
- Workload
  - 128 instantiations of each TPC-H query type

Evaluation: TPC-H



Considerable speedup: up to ~128x.

Evaluation: TPC-H



Again: cost (almost) independent of batch size.

Outline

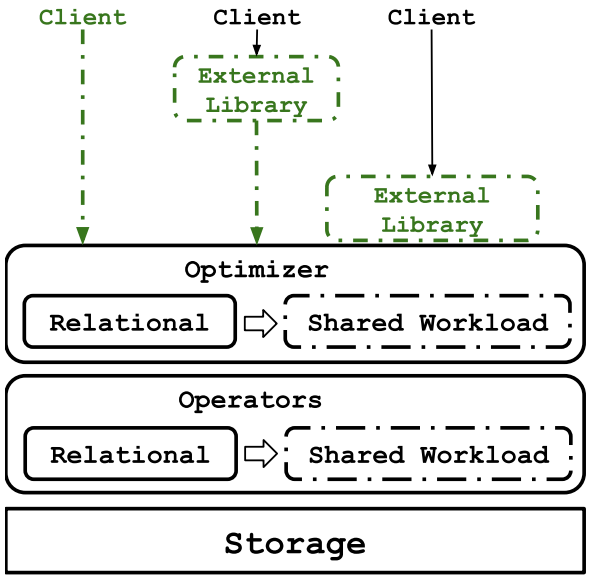
- Related work
- Multi-query execution
- Evaluation
- Summary
- Conclusions

Summary

- Shared query execution through query rewriting
  - Rewriting can be done with standard SQL
  - Evaluate our approach by executing queries end-to-end
- Show improvements in execution cost and throughput

Conclusion

- Sharing can be implemented in:
  - Client
  - Library
  - Middleware



Thanks!