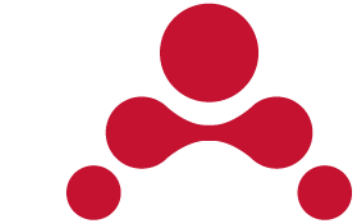


ASTER

Do more.

Optimizing Hierarchy Joins

Herodotos Herodotou
Nedyalko Borisov

Real Life Scenario

- Brewery Inc.
  - Daily reports for production and sales
  - Need to organize data in data warehouse
- Options:
  - Single table per production/sales
  - Table per production/sales per day

Options Discussion

- Single table per production/sales
  - + Simple queries for yearly/quarter reports
  - - Performance issues with daily reports
- Table per production/sales per day
  - + Good performance for daily reports
  - - Complicated queries for yearly/quarter reports
- Want: benefits from both

PostgreSQL Support

- Inheritance
  - Way to organize tables
  - Retain the abstraction of a single table
- Example

  production (id, date, quantity, …)

  production_jan_1 (id, date, quantity, …)

  production_jan_2 (id, date, quantity, …)

  . . .

- Query over “production” => combines all data
- Query over “production_jan_1” => only Jan 1st

Specifics

- Parent table

```
create table production (
    int             “id”,
    timestamp  “date”,
    . . . );
```

- Child tables

```
create table production_jan_1 (
    check (‘2009-01-01’ <= “date” and “date” < ‘2009-01-02’)
) inherits (production);
```

Sample Reports

- Beer quantities produced on Jan 1st

```
select id, sum(quantity)
from production
where '2009-01-01' <= "date" and "date" < '2009-01-02'
group by id;
```

- Execution plan scans ONLY "production_jan_1"
- How?
  - Filters tables based on check constraints

Sample reports cont.

- Sales per beer kind per day

```
select p.id, p.date, sum(s.price)
from production p, sales s
where p.date = s.date and p.id = s.id
group by p.id, p.date;
```

- Execution plan treats each hierarchy as single table
  - next slide

Execution Plan

```
Group agrregate
-> merge join
    -> sort
        -> Append
            -> Seq Scan on production
            -> Seq Scan on production_jan_1
            -> . . .
    -> sort
        -> Append
            -> Seq Scan on sales
            -> Seq Scan on sales_jan_1
            -> . . .
```

Execution Plan - Discussion

- Extra work
  - matching beer produced on Jan 1 with sales from Jan 2
- Can we avoid it?
  - Utilize check constraints
  - Join child tables directly

New Execution Plan

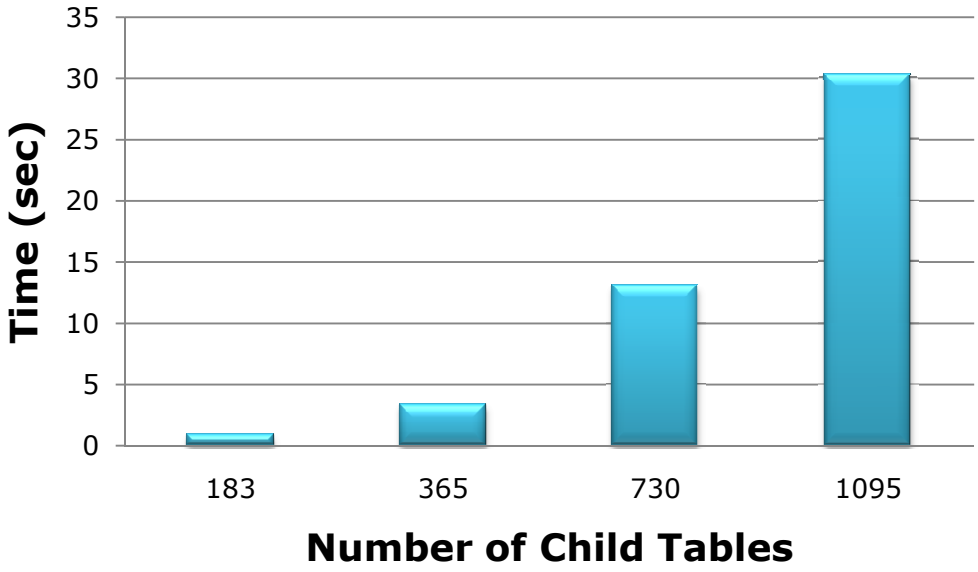
```
Hash aggregate
-> Append
    -> Hash Join
        -> Seq Scan on production_jan_1
        -> Seq Scan on sales_jan_1
    -> Hash Join
        -> Seq Scan on production_jan_2
        -> Seq Scan on sales_jan_2
    -> . . .
```

Algorithm Overview

1. Check if we can join child tables directly
2. Get child table constraints
3. Find possible child joins
4. Generate plans for each child join
5. Combine results from child joins

Find possible child joins

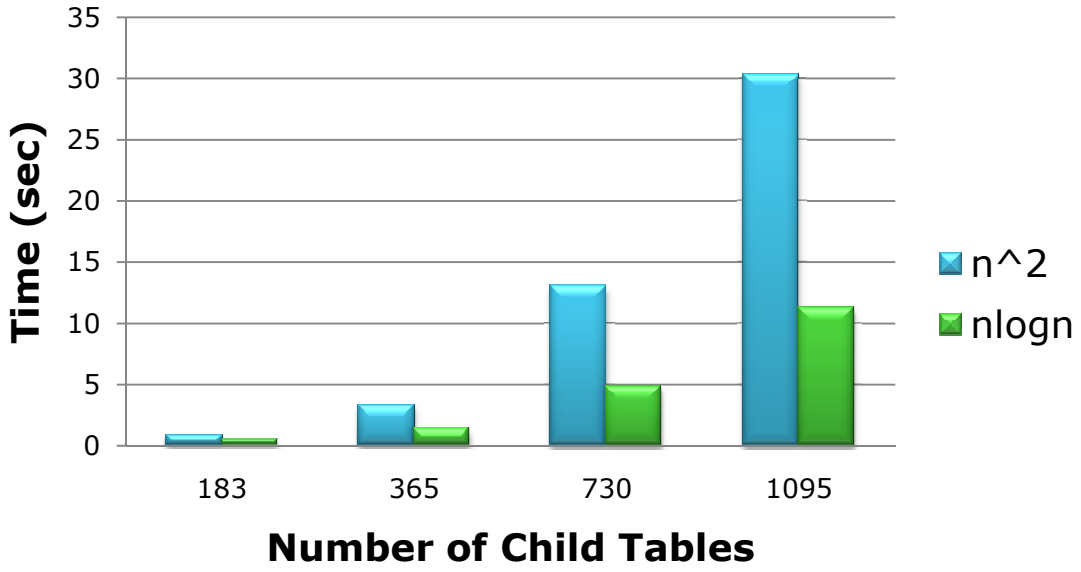
- First implementation: naïve n^2
  - Examine constraints from each possible pair of child tables
  - Expensive



ASTER

Find possible child joins cont.

- Another approach:
  - Treat constraints as intervals
  - Use an interval tree for the matching of child tables
  - Complexity: n*logn



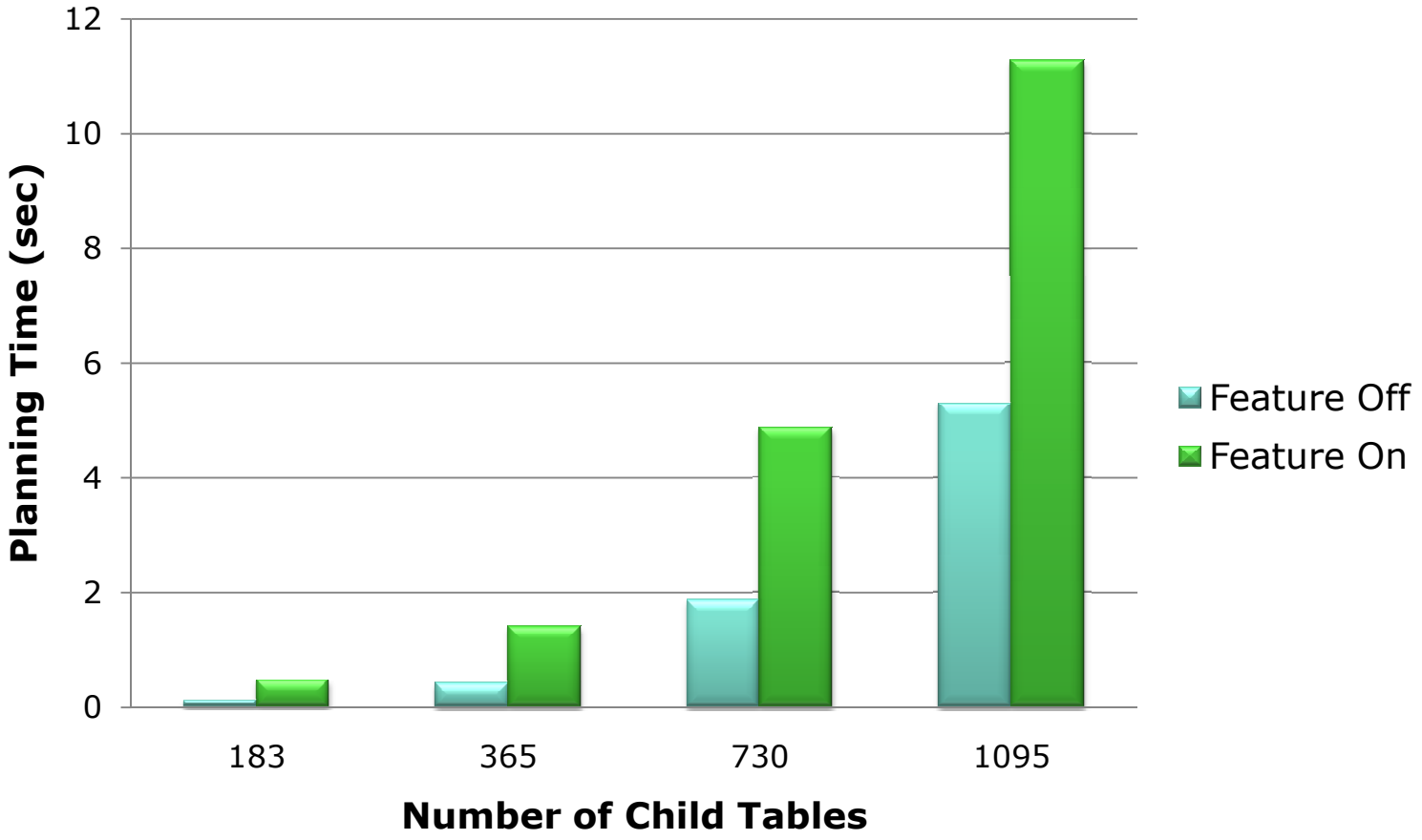
ASTER

Experimental Evaluation

- Goals:
  - Examine overhead on planning time
  - Examine effects on execution time
- Factors
  - Number of child tables
  - Number of records in hierarchies
  - Number of joins in query

Experimental Evaluation

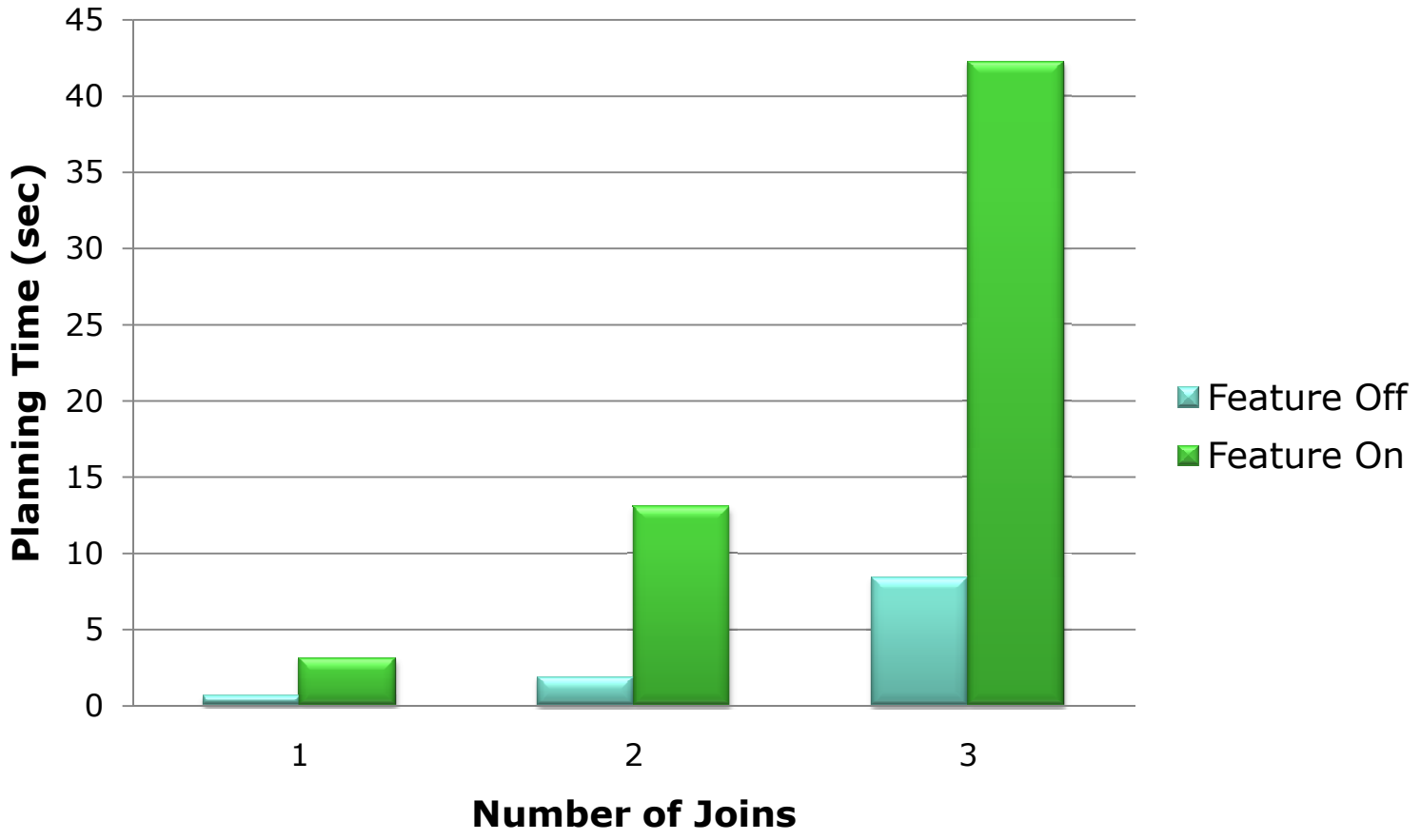
- Number of joins = 2



ASTER

Experimental Evaluation

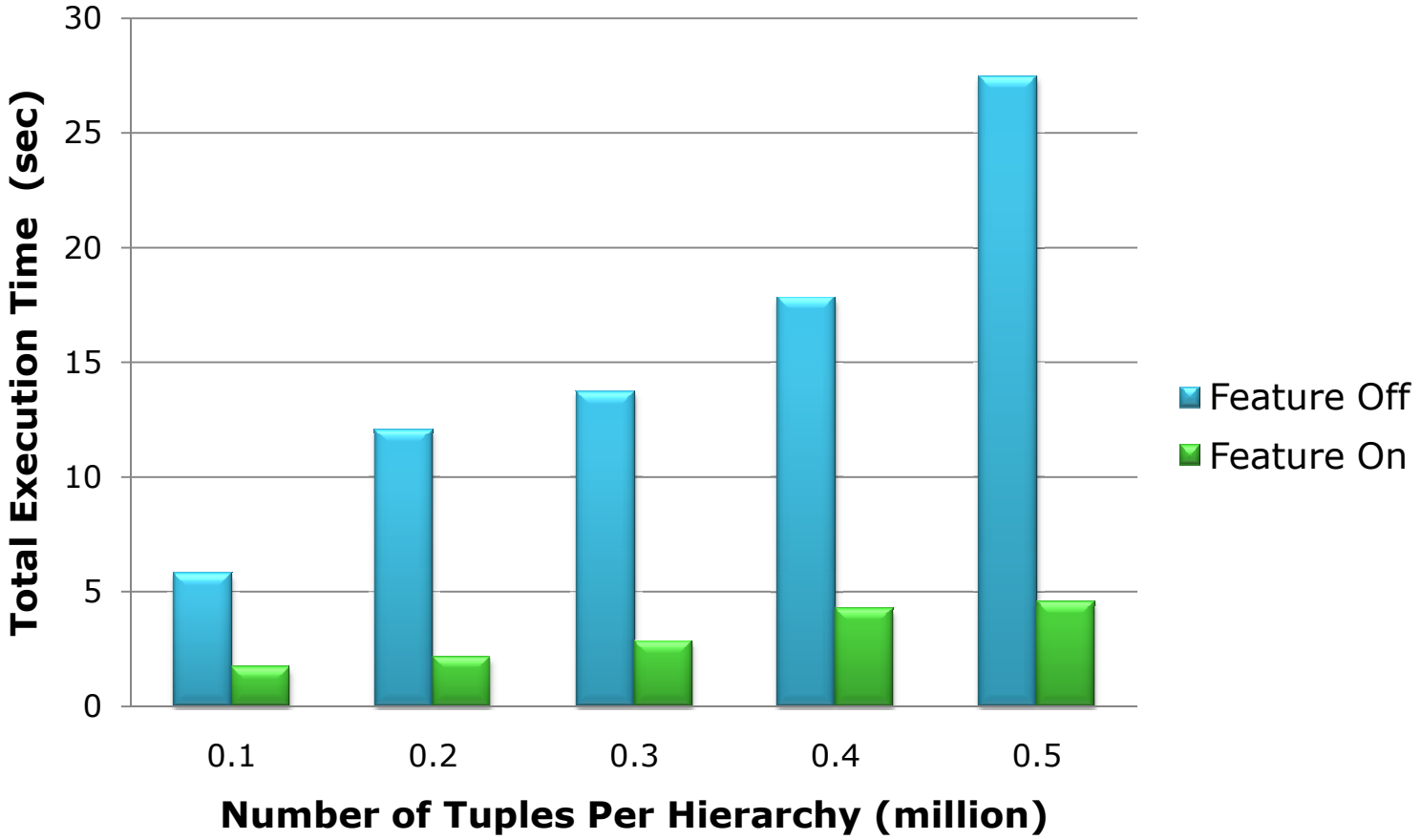
- Number of child tables = 730



ASTER

Experimental Evaluation

- Number of child tables = 365
- Number of joins = 2



ASTER