Introducing Postgres Table Partitioning to dCache

Mwai Karimi
IT-SCI,
HEPix Spring 2022, 27.4.2022

HELMHOLTZ

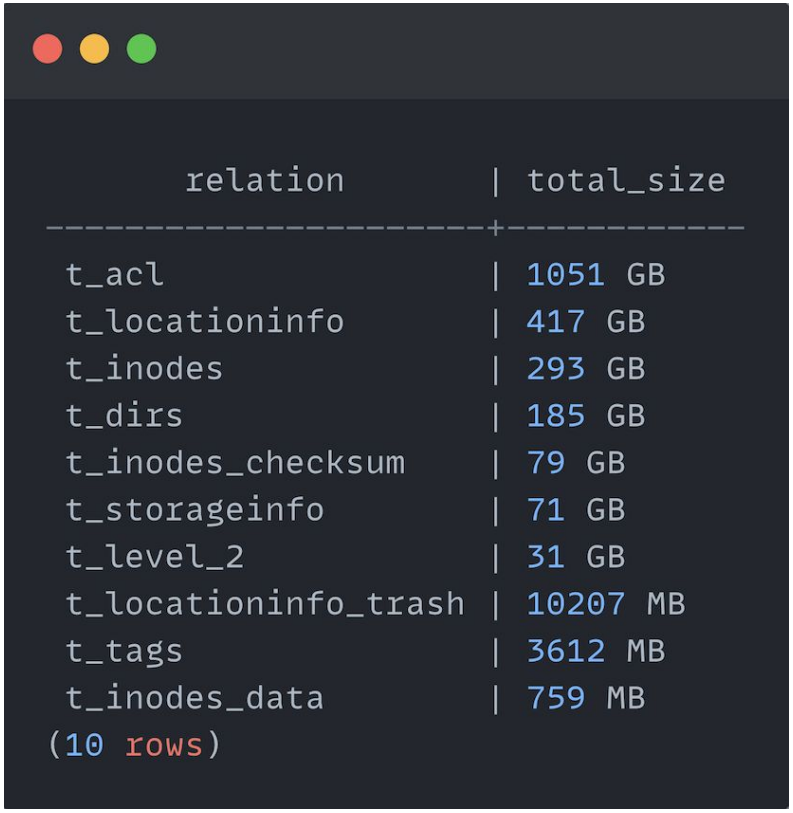
DESY.

dCache PgSQL instances

- 6 Production Instances
  - CMS - 56G
  - ATLAS - 69G
  - DESY - 82G
  - XFEL - 126G
  - CLOUD - 269G
  - PHOTON - 2.2T

Motivation

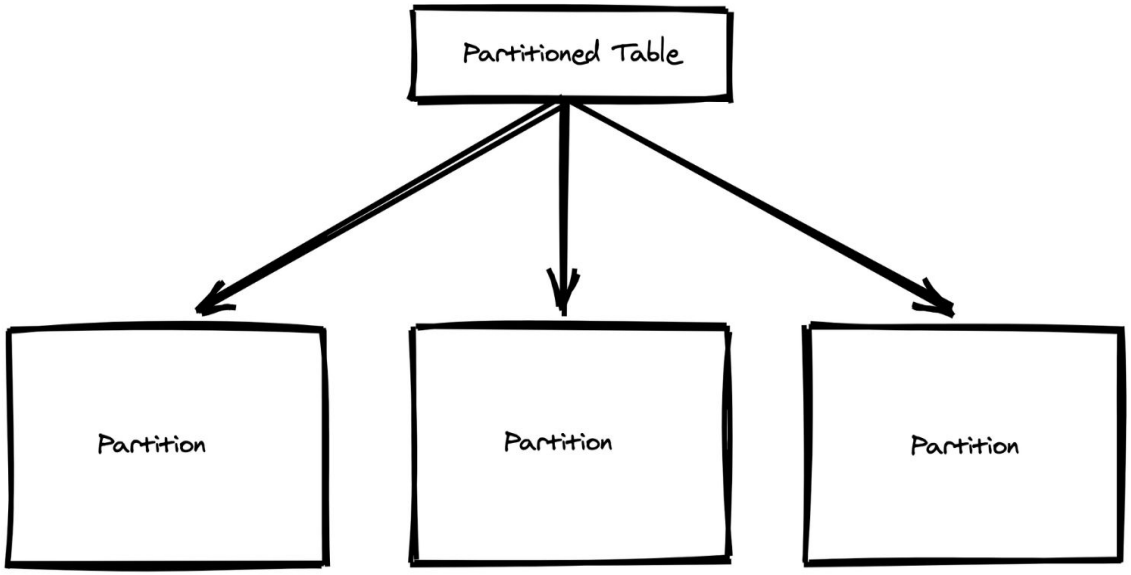
- Growing table sizes
  - Candidate table - ~960m rows
  - Slow queries
  - Index maintenance overhead



```
        relation         | total_size
-------------------------+------------
 t_acl                   | 1051 GB
 t_locationinfo          | 417 GB
 t_inodes                | 293 GB
 t_dirs                  | 185 GB
 t_inodes_checksum       | 79 GB
 t_storageinfo           | 71 GB
 t_level_2               | 31 GB
 t_locationinfo_trash    | 10207 MB
 t_tags                  | 3612 MB
 t_inodes_data           | 759 MB
(10 rows)
```

Intro to Partitioning

- Organising data into logical chunks
- Each chunk goes to it's own table



Why?

- Optimisation of queries - partition pruning, tuple routing
- Bulk loads and deletes
- Optimising access to data
  - Improved scan performance
  - Avoid cache churn

Inheritance based Partitioning

- Manual
- Slow
- Hard to maintain
- Error prone
- Hard to develop features on top of

Declarative Partitioning

- Simpler syntax
- Properly optimised
- Easy to manage

Options for Partitioning Strategy

```
[ PARTITION BY { RANGE | LIST | HASH }
( { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ]
```

- Range
- List
- Hash

Limitations

- Unique constraints
- Exclusion constraints

Implementation Strategy

- Partition by hash on `inumber` column

```
CREATE TABLE t_inodes (
        ...
) PARTITION BY HASH(inumber);
```

  - Create 20 Partitions

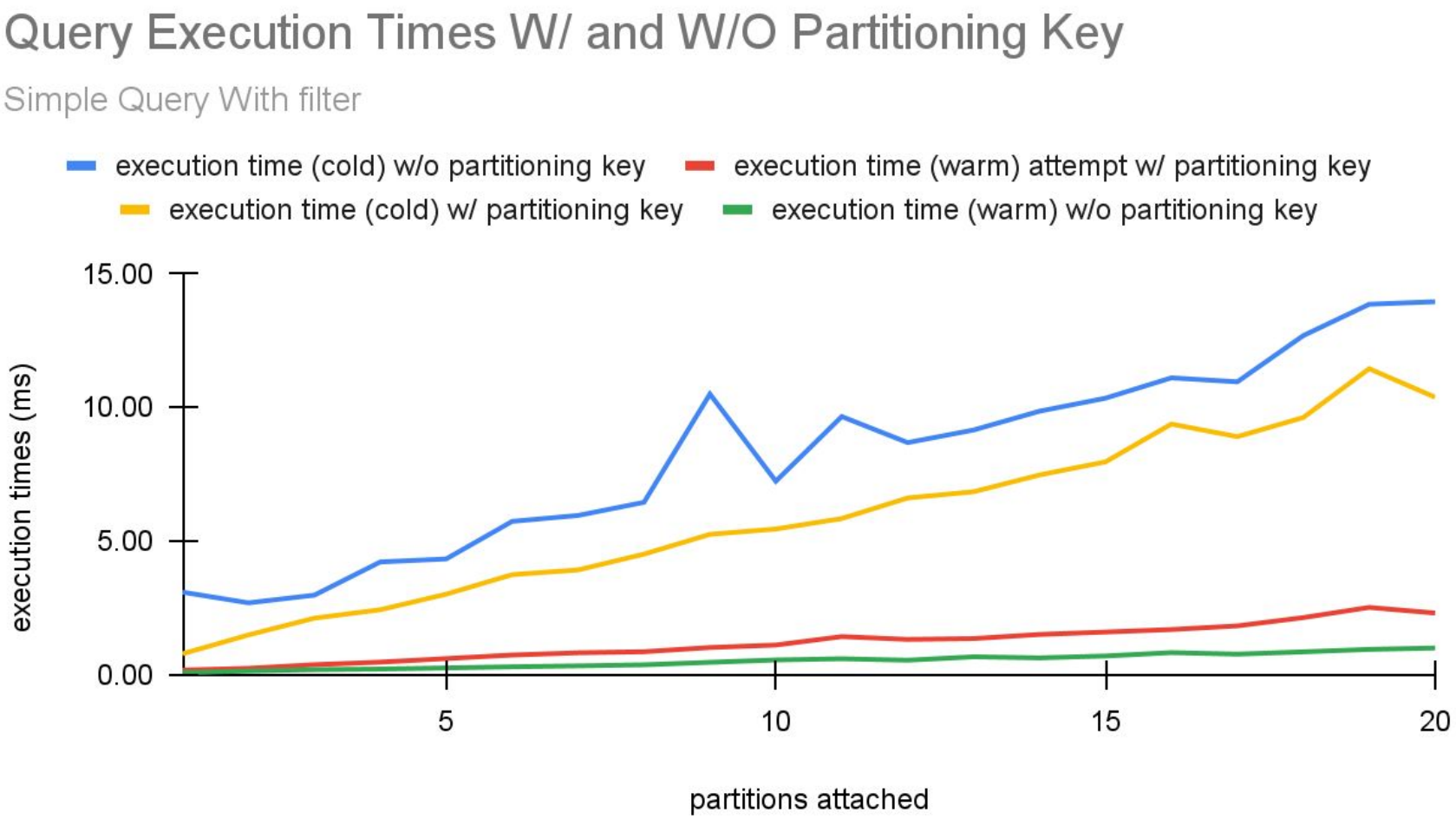
```
-- call function create_hash_parts(int, regclass)
SELECT FROM create_hash_parts(20, 't_inodes');
```

Distribution

20 Partitions - ~48m rows each

```
chimera=# select tableoid::regclass, count(*) from t_inodes group by 1 order by 1;
         relname          |     count
--------------------------+---------------
 t_inodes_p01             | 4.7640808e+07
 t_inodes_p02             | 4.7638512e+07
 t_inodes_p03             | 4.7640232e+07
 ...
 t_inodes_p18             | 4.7642164e+07
 t_inodes_p19             | 4.7643148e+07
 t_inodes_p20             | 4.7639996e+07
(20 rows)
```

Query Analysis: Select..where inumber=?



Query analysis: Select…inner join…where?

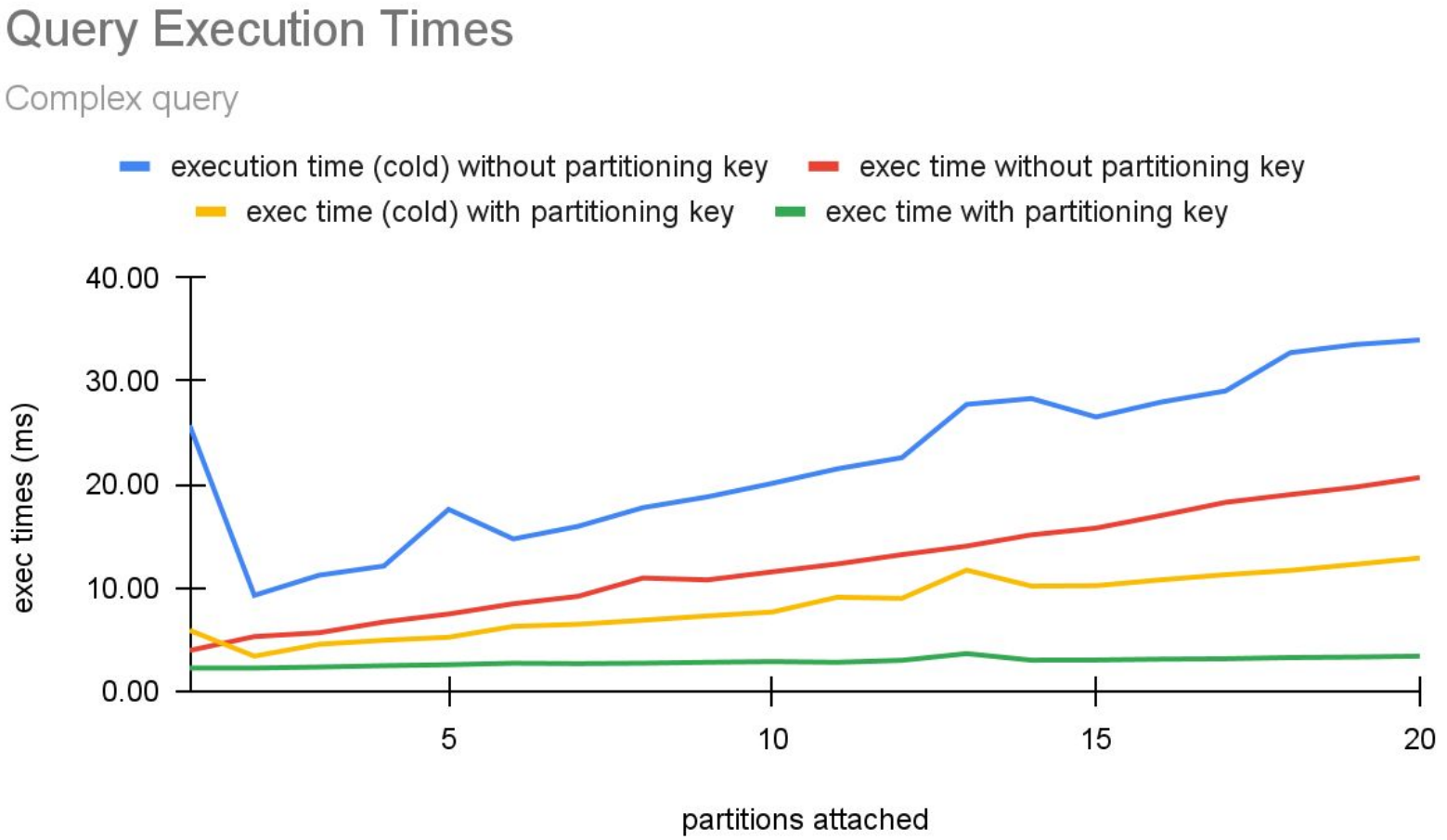
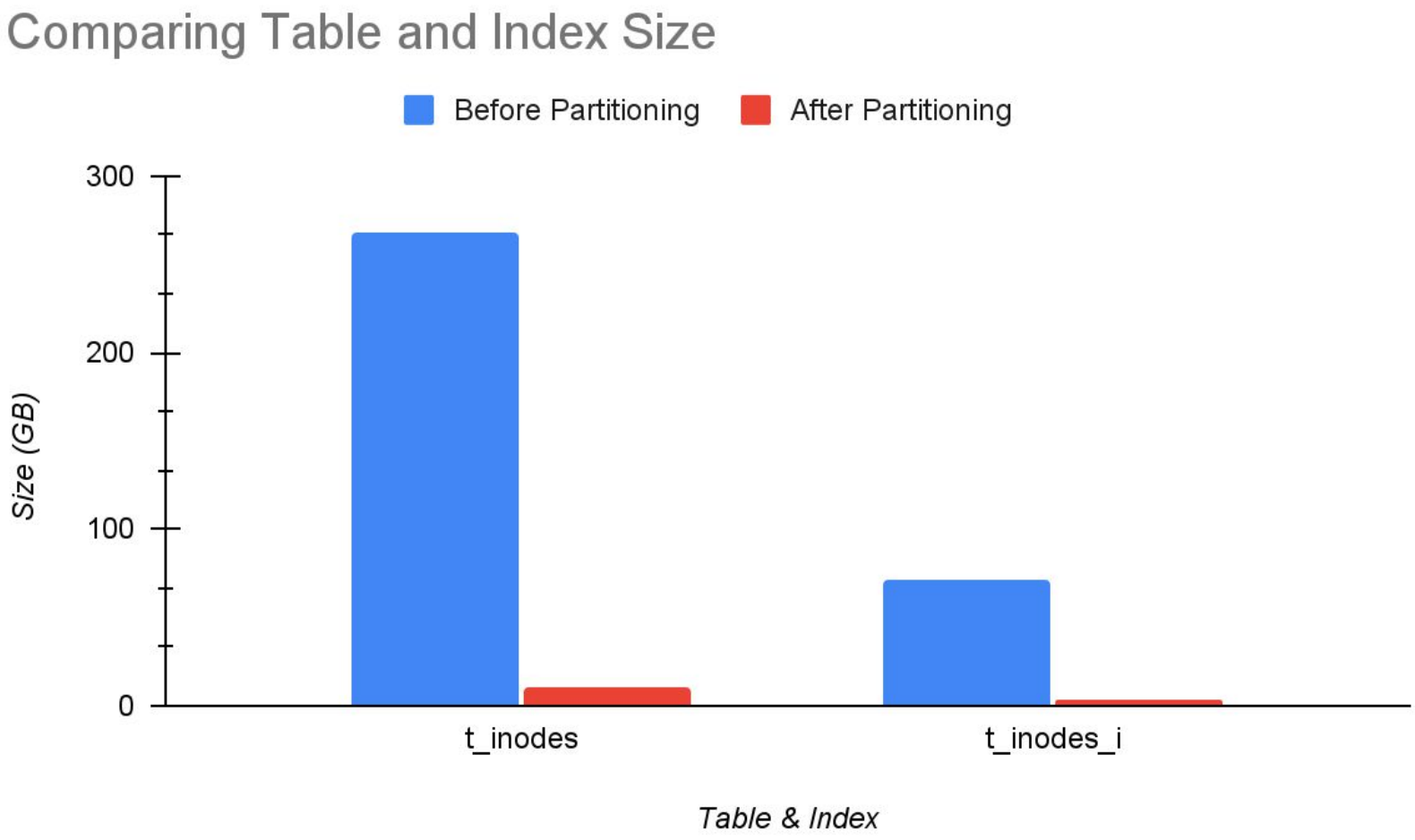


Table and Indices



Takeaways

- Partitioning helps reduce execution times
- Planning times increase due to pruning
- Reduced table and index size

Thank you