Easy-to-use data schema management scheme for RDBMS that includes the utilization of the column-store features

Tadashi Murakami

High Energy Accelerator Research Organization (KEK)
Computing Research Center

Outline

- Column-store database
  - introduction, pros/cons, usefulness
  - comparison of data compression, performance
- Easy-to-use data schema management scheme for RDBMS
  - In progress
- Summary

Outline

- Column-store database
  - introduction, pros/cons, usefulness
  - comparison of data compression, performance
- Easy-to-use data schema management scheme for RDBMS
  - In progress
- Summary

Column-store database

- Column-store database stores data by column
  - Ordinal database stores data by rows
- Recently, some of Relational DBMS (RDBMS) supports column-store features

Ordinal database

	col1	col2	col3
row1	r1c1	r1c2	r1c3
row2	r2c1	r2c2	r2c3
row3	r3c1	r3c2	r3c3

Column-store database

	col1	col2	col3
row1	r1c1	r1c2	r1c3
row2	r2c1	r2c2	r2c3
row3	r3c1	r3c2	r3c3

Column-store database: pros and cons

- Merits
  - Stored columns itself become a sequential index
  - Efficient data compression can be achieved
    - Similar-trended data are gathered in each column
  - Length of rows does not effect search performance
- Demerits
  - Inefficiency in insert data
  - Difficulty in update and delete data
    - Many of column-store DBs do not support

Ordinal database

	col1	col2	col3
row1	r1c1	r1c2	r1c3
row2	r2c1	r2c2	r2c3
row3	r3c1	r3c2	r3c3

Column-store database

	col1	col2	col3
row1	r1c1	r1c2	r1c3
row2	r2c1	r2c2	r2c3
row3	r3c1	r3c2	r3c3

Usefulness of column-store

- Strength of column-store
  - Lossless data compression
  - with a high level of both data size and query performance
- Suitable for time-series data
  - in many cases, no need to update or delete logged data
  - data change might be written in another tables
- Suitable for HEP experiments
  - bothering for both high performance and storage capacity in handling time-series data

Outline

- Column-store database
  - introduction, pros/cons, usefulness
  - comparison of data compression, performance
- Easy-to-use data schema management scheme for RDBMS
  - In progress
- Summary

Comparison: table size and query performance

- Measured table size and query performance
- Using time-series data of a firewall's connection log
  - 114 million rows, 49 columns
    - extract 15 columns in 'core' tables
- Prepared indexed and non-indexed tables
  - indexed in 8 columns and used for the queries
- Three of query pattern:
  - Simple (find 15258 rows) : src-IP address
  - Complex (find 6974 rows) : src-IP address, 9am-5pm
  - Complex (find 6878 rows) : src-IP address, 9am-5pm, allowed communications

Comparison: environments

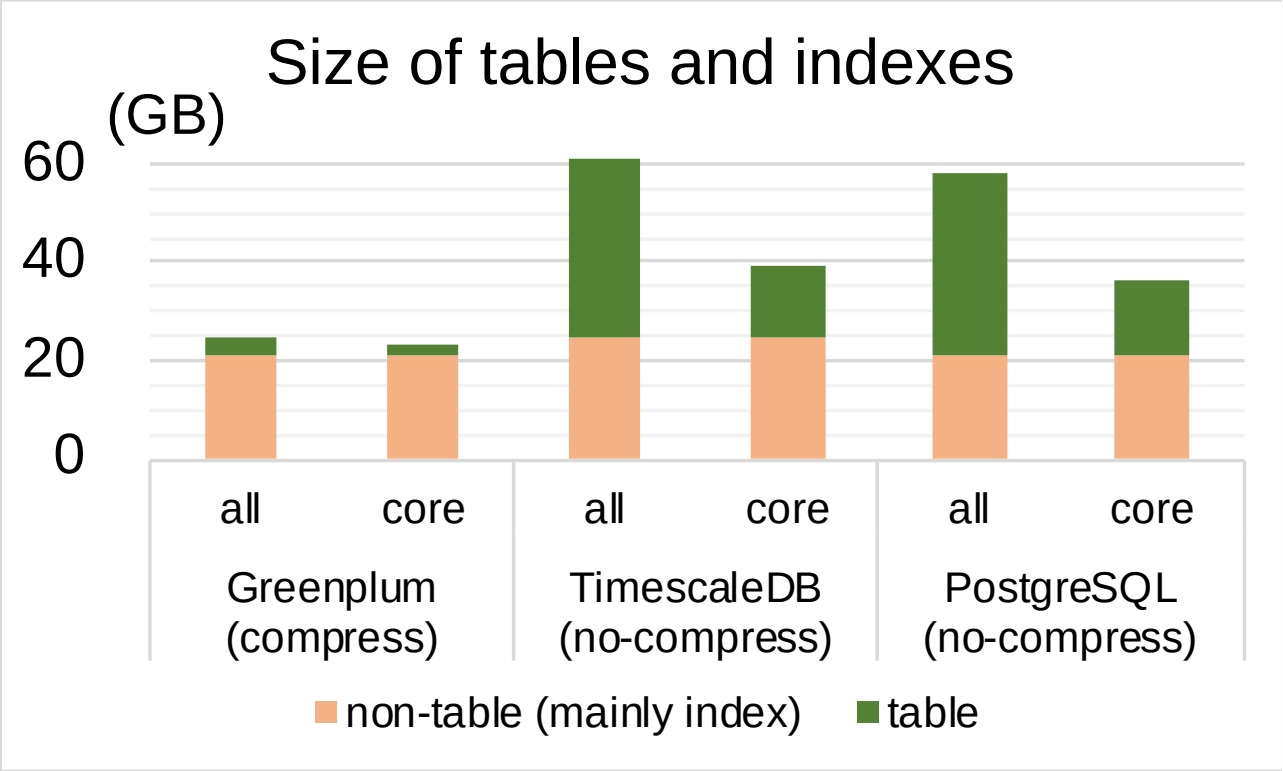
- Server spec:
  - Dual Xeon E5-2603v3 Dual, 32GB Memory
  - RAID6 (3TB SATA-HDD x 7)
  - Oracle Linux 7.7
- Database environments and tune-up
  - PostgreSQL 8.3.23-based GreenplumDB
    - Follow the recommendations of PGTune (for ver 9.2)
    - on 2019-09-03, newer version was released and target version was changed to pgsql 9.4
  - PostgreSQL 11.5 with extensions of cstore_fdw, TimescaleDB
    - Follow the recommendations of timescaledb-tune

Comparison: database environments

- Options
  - (i) No-indexing data / (I) indexing data
  - (c) No-compress data / (C) compress data
- Postgres-based column-store databases:
  - cstore_fdw (extension of pgsql 11.5: 11-1.6.2-1) [ic] I: no support
  - Greenplum (5.3.0 ... based on PostgreSQL 8.3.23) [ic, Ic, IC, iC]
- Postgres and Postgres-based (row-store) databases:
  - Native postgreSQL (11.5) [ic, Ic] C: no support
  - TimescaleDB (extension of pgsql 11.5: 11-1.4.2) [ic, Ic] C: no support

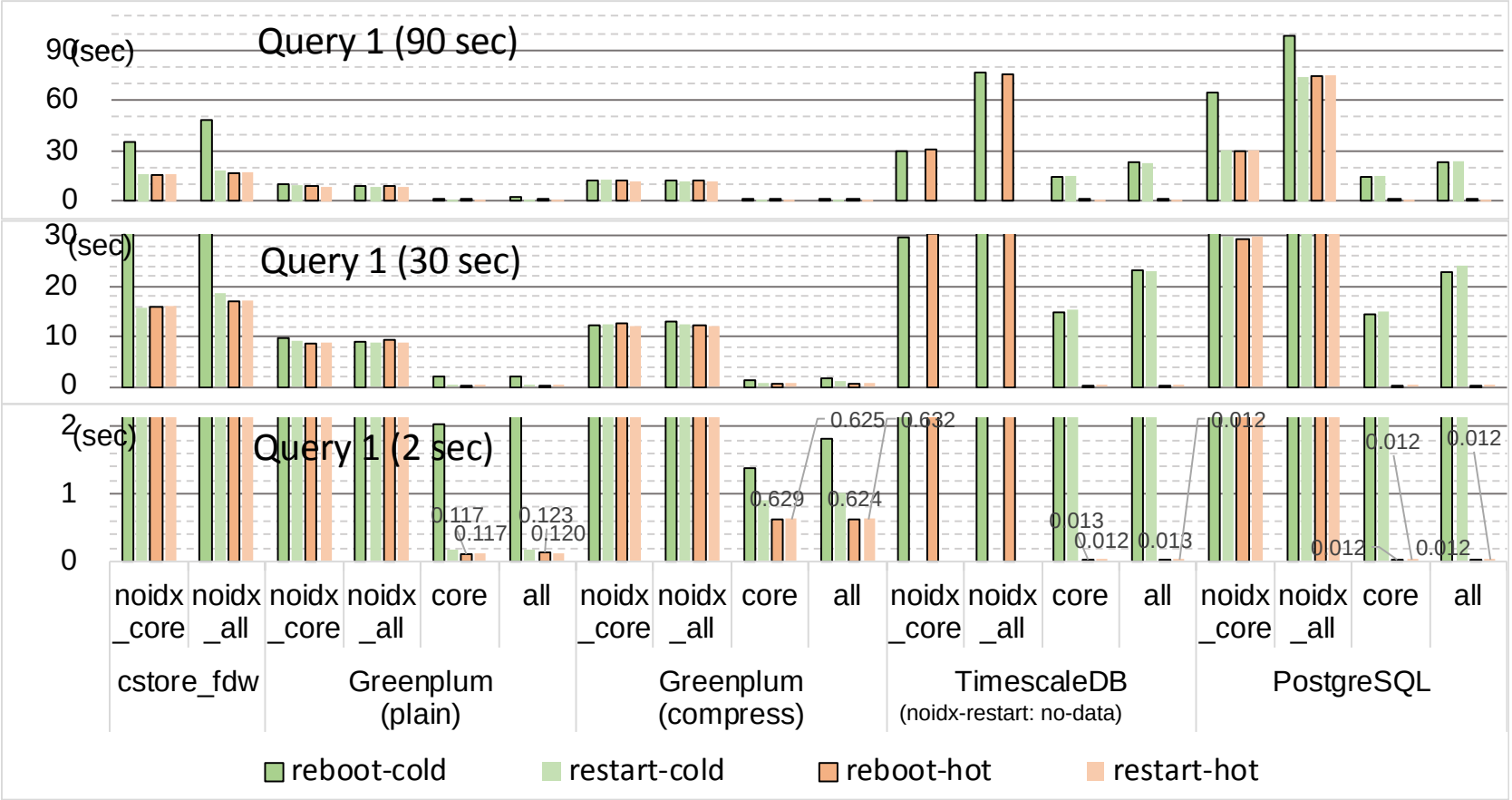
Comparison: table and index size

- About 90% compression rate of tables
- Indexes cannot be compressed



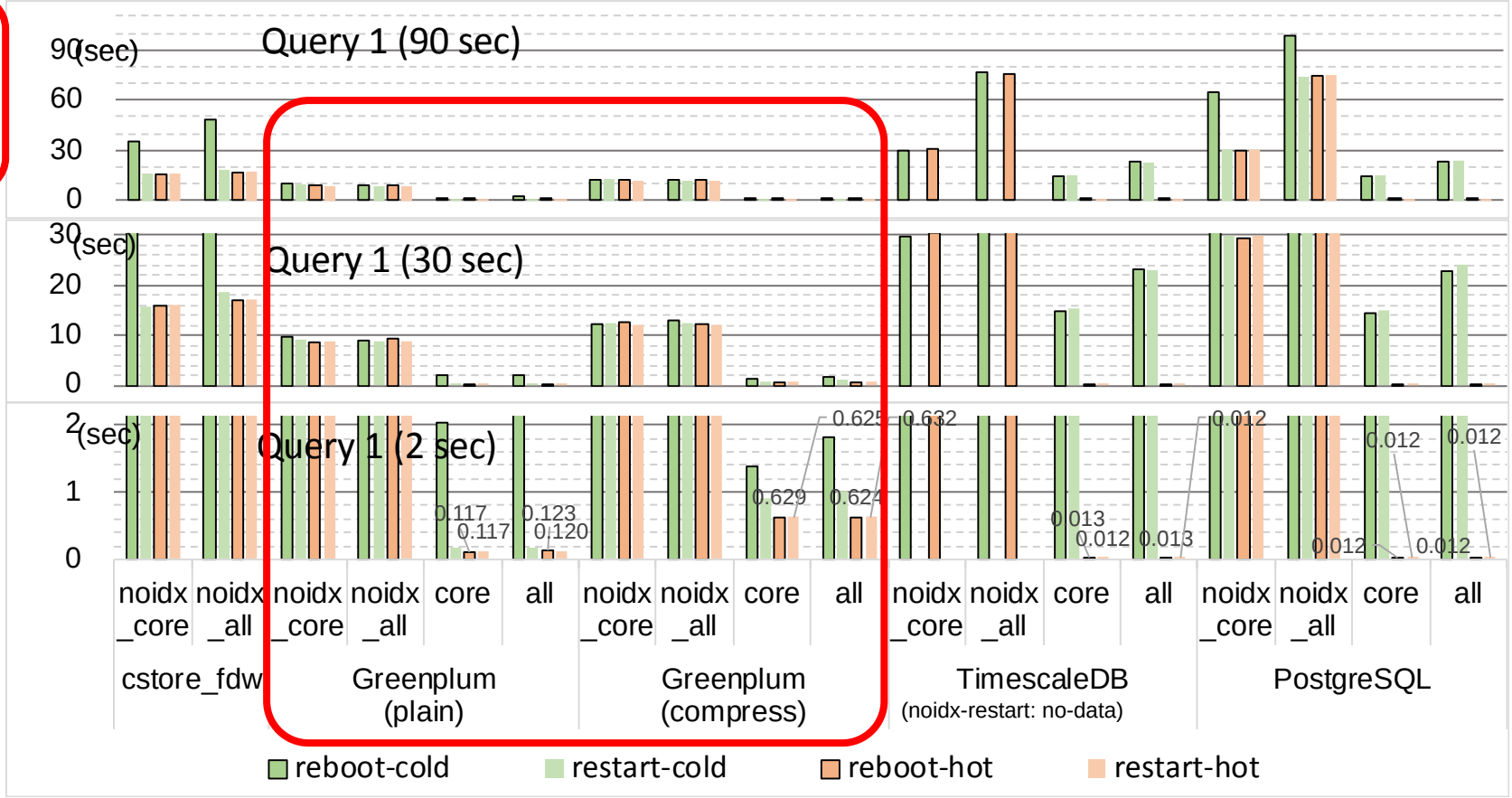
Comparison: performance

- The three graphs in the right is plotted from same data
  - 90 sec
  - 30 sec
  - 2or5 sec
- Green: cold search
  - In a first search from reboot (PC) or restart (DB)
  - Cache is not active
- Red: hot search
  - Same search is issued
  - Cache is active



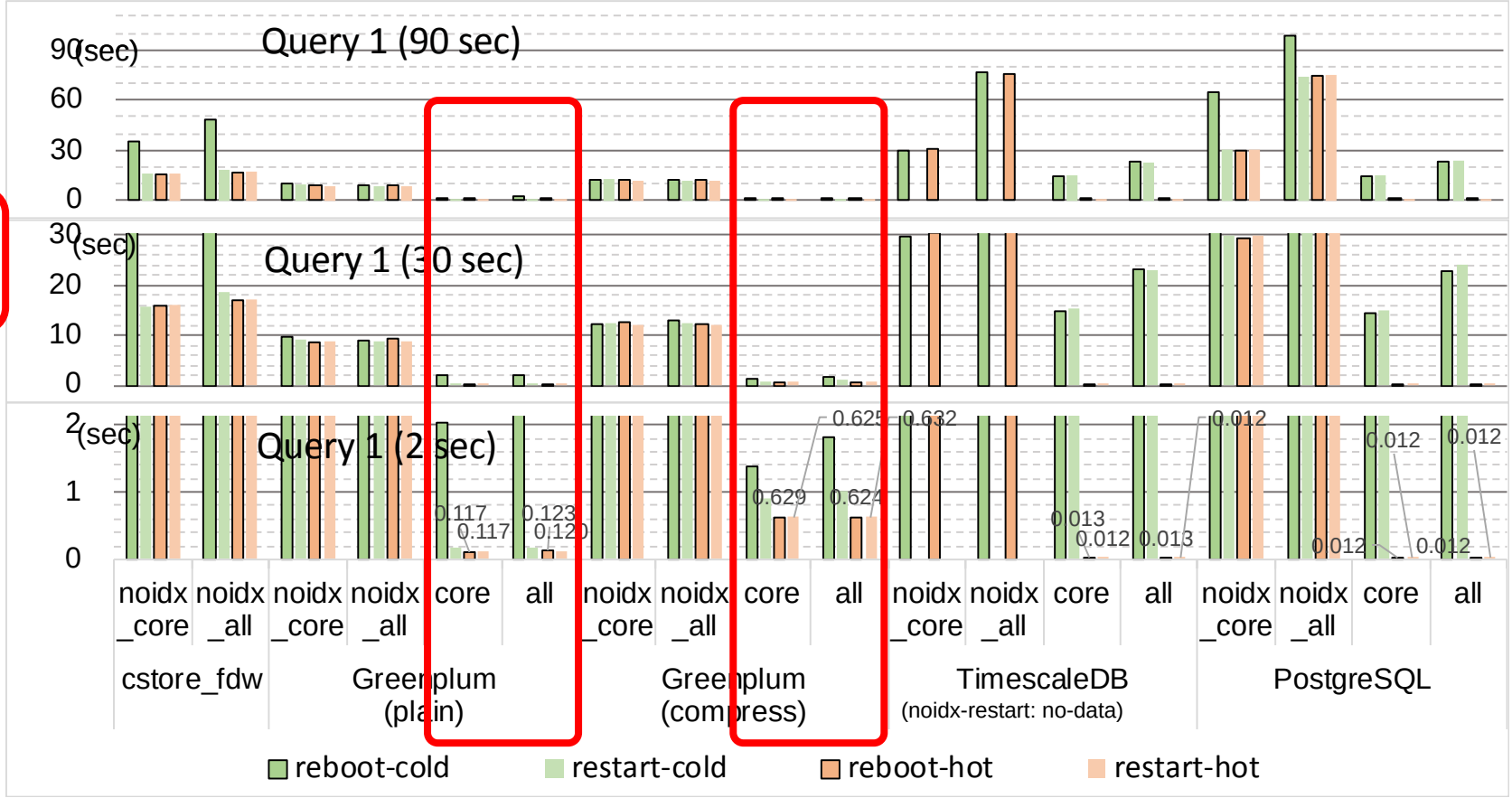
Comparison: performance

- Green: cold search
  - Without cache, Greenplum overwhelms others
  - Especially in indexed data
- Red: hot search
  - When a same search is issued, PostgreSQL itself returns a good performance
    - Base version of Greenplum is old



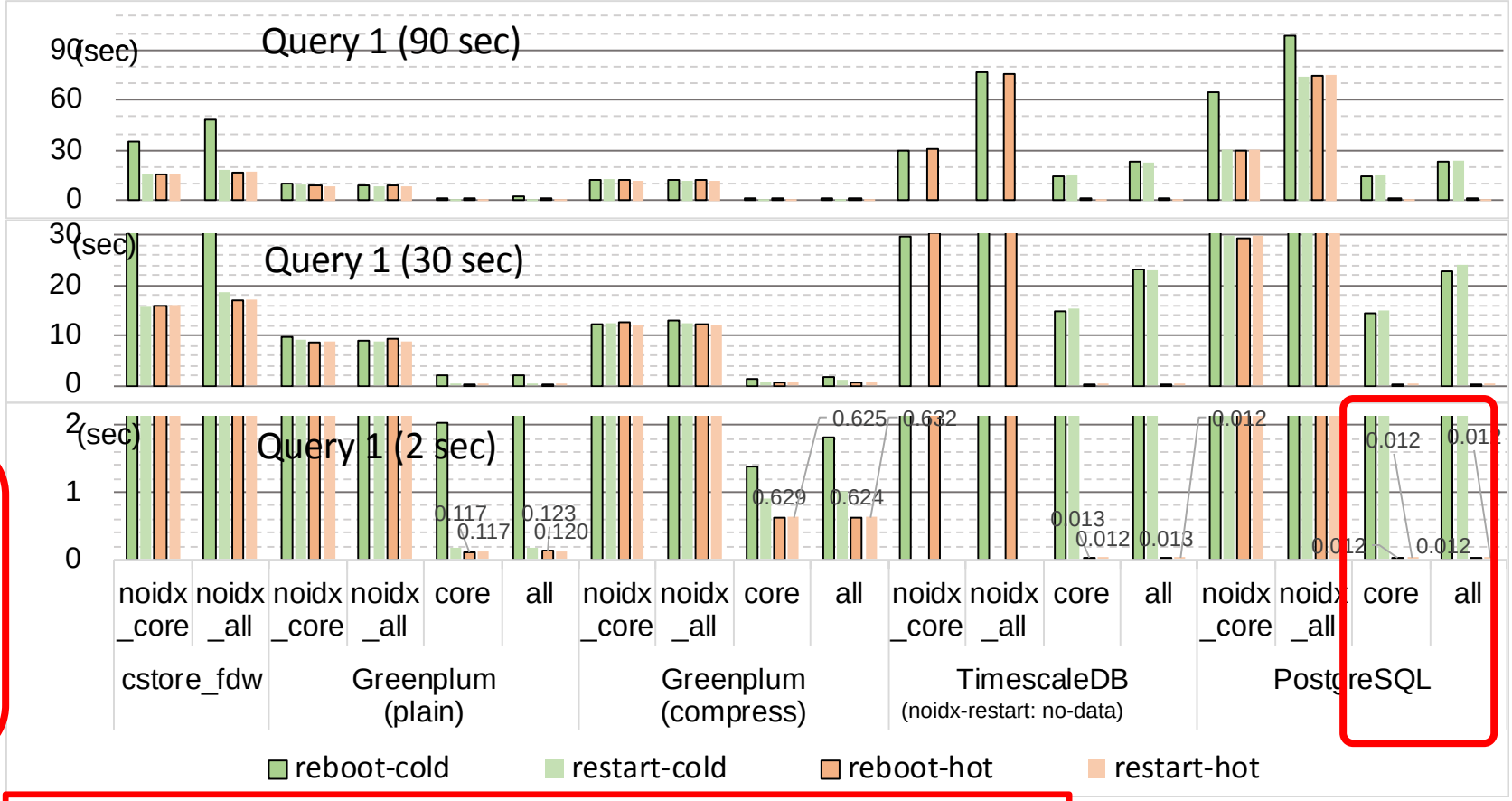
Comparison: performance

- Green: cold search
  - Without cache, Greenplum overwhelms others
  - Especially in indexed data
- Red: hot search
  - When a same search is issued, PostgreSQL itself returns a good performance
    - Base version of Greenplum is old



Comparison: performance

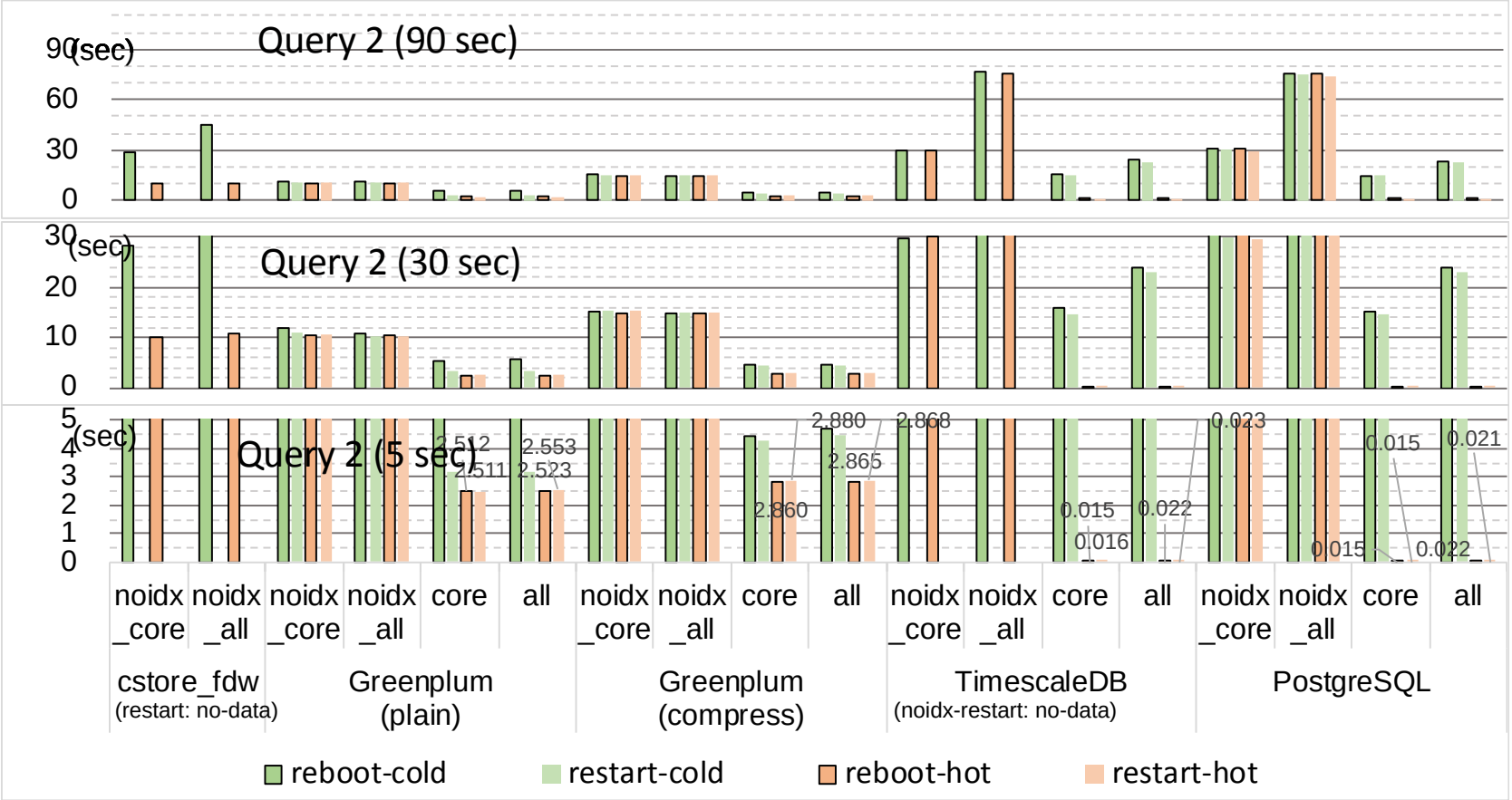
- Green: cold search
  - Without cache, Greenplum overwhelms others
  - Especially in indexed data
- Red: hot search
  - When a same search is issued, PostgreSQL itself returns a good performance
    - Base version of Greenplum is old



In an analysis of data, searching for a new data (cold search) is more important

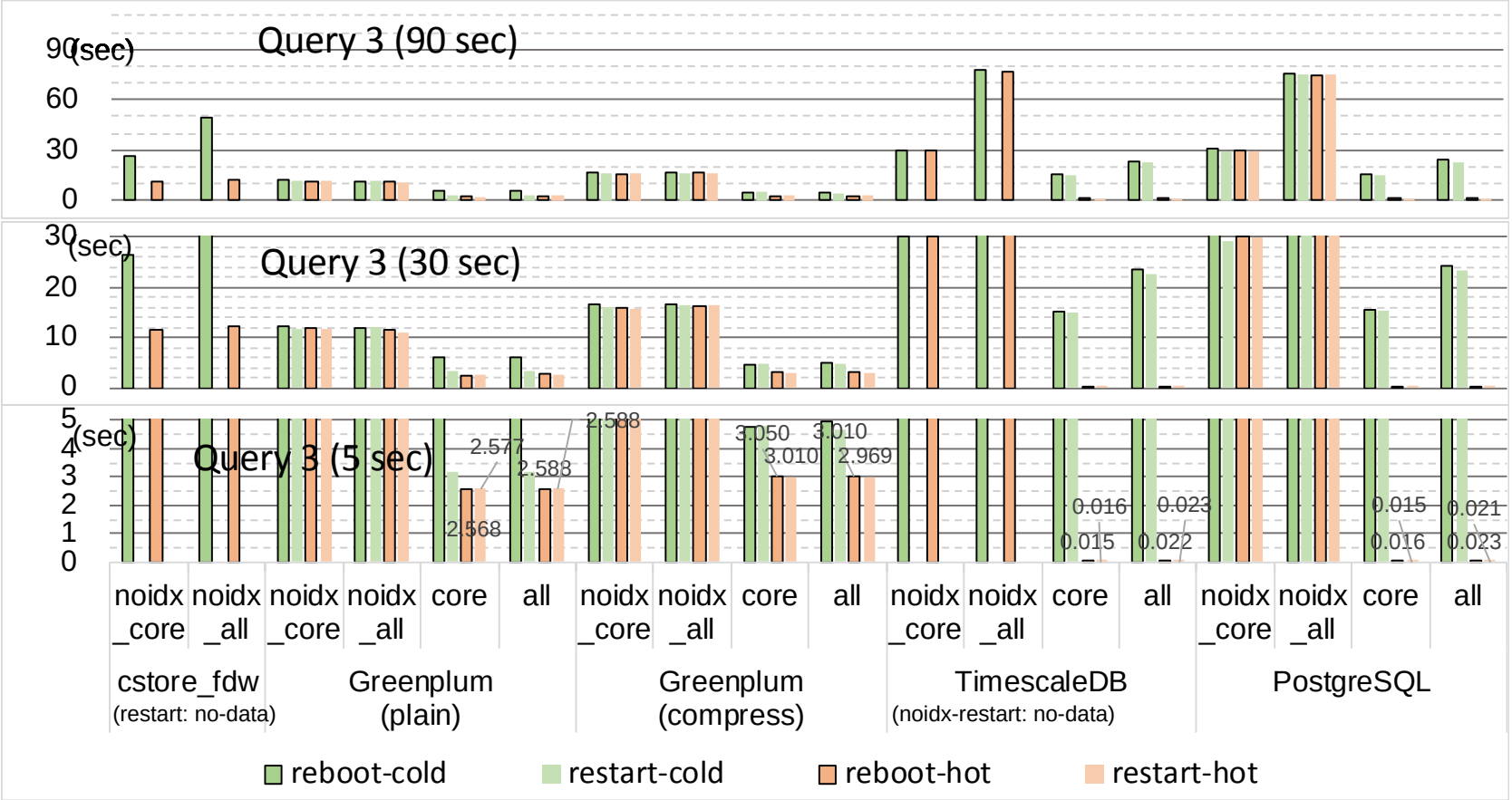
Comparison: performance

- In complex queries, the results show the same tendency



Comparison: performance

- In complex queries, the results show the same tendency



Outline

- Column-store database
  - introduction, pros/cons, usefulness
  - comparison of data compression, performance
- Easy-to-use data schema management scheme for RDBMS
  - In progress
- Summary

Easy-to-use data schema management scheme for RDBMS

- (Finished) Develop a simple language
  - create tables
  - monitor data
  - import data
    - indexing
- (In progress) Support options
  - partition
  - star-schema
  - other options to optimize the tables

```
# (1) variable definition
  $db_conn_str = jdbc:postgresql://localhost/test?user=fred&password=secret

  $time = \d{2}:\d{2}:\d{2}  ;  $date = \d{4}/\d{2}/\d{2}  ;  $dtime = ${date} ${time}
  # ...


# (2) record definition
  [dev_kekfw_test_1]      # assign table name
    input  = file:///path/to/kekfw1/kekfw1.local0.log.20160619.gz
    output = rdb: $db_conn_str

    filter = RegExp: (\w+ +\d+ ${time}) (\w+ (?:\d+,)?)((?:${date} )?${time}),(${id}),  \\
                     (TRAFFIC),(\w+),(\d+),(${dtime}),                   \\
                     (${ip4_6}),(${ip4_6}),(${ip4_6}),(${ip4_6}),        \\
    # ...
    filter = ColName: (kekfw_id), domain_time, domain, receive_time<timestamp>, serial_number,\\
                      type, log_subtype, config_version<smallint>, generated_time<timestamp>, \\
                      src_ip<inet>, dst_ip<inet>, nat_src_ip<inet>, nat_dst_ip<inet>,         \\
    # ...

    filter = partition: date("start_time", "day")        # setup partition table
    filter = toStar: rule_name, application              # setup star schema
    # ...
```

Easy-to-use data schema management scheme for RDBMS (cont'd)

- Support options: partition
  - Split into sub-tables using key columns of sequential data (e.g., time, serial_id)
  - Current version supports only fixed options (e.g., split per one day)
    - It is desirable if a user do not have to pay attention on it
- Support options: star-schema
  - Extract dimensions from a fact
  - fact: data obtained from a event
  - dimension: labeled structured data obtained from a fact
- Support options: join other the tables

Summary

- Column-store database stores data by column
- Merits
  - Efficient data compression can be achieved
  - Length of rows does not effect search performance
- Demerits
  - Inefficiency in insert data, Difficulty in update and delete data
- Easy-to-use data schema management scheme for RDBMS
  - Simple language
  - Support options