CarbonData : An Indexed Columnar File Format For Interactive Query

www.huawei.com

HUAWEI TECHNOLOGIES CO., LTD.

HUAWEI

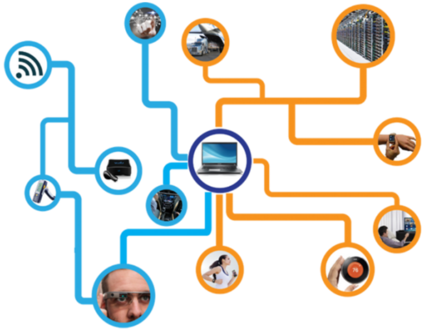
Outline

- Motivation : Why introducing a new file format?
- CarbonData Deep Dive
- Tuning Hint

Big Data

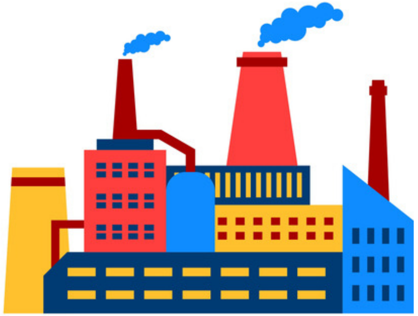
Network

- 54B records per day
- 750TB per month
- Complex correlated data



Consumer

- 100 thousands of sensors
- >2 million events per second
- Time series, geospatial data



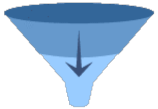
Enterprise

- 100 GB to TB per day
- Data across different domains

Typical Scenario

Report & Dashboard

OLAP & Ad-hoc



Batch processing

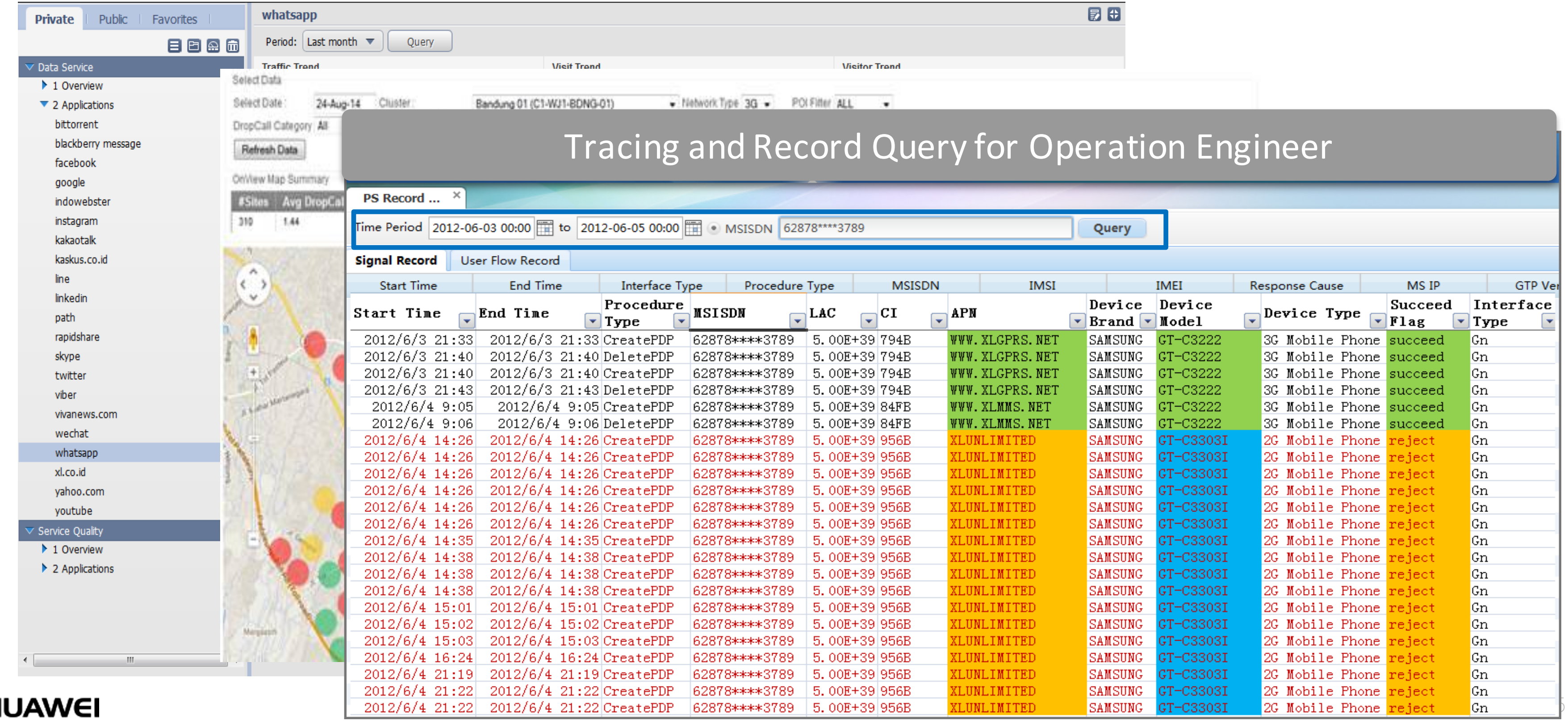
Machine learning

Realtime Decision

Text analysis

data

Analytic Examples



Start Time	End Time	Procedure Type	MSISDN	LAC	CI	APN	Device Brand	Device Model	Device Type	Succeed Flag	Interface Type
2012/6/3 21:33	2012/6/3 21:33	CreatePDP	62878****3789	5.00E+39	794B	WWW.XLGPRS.NET	SAMSUNG	GT-C3222	3G Mobile Phone	succeed	Gn
2012/6/3 21:40	2012/6/3 21:40	DeletePDP	62878****3789	5.00E+39	794B	WWW.XLGPRS.NET	SAMSUNG	GT-C3222	3G Mobile Phone	succeed	Gn
2012/6/3 21:40	2012/6/3 21:40	CreatePDP	62878****3789	5.00E+39	794B	WWW.XLGPRS.NET	SAMSUNG	GT-C3222	3G Mobile Phone	succeed	Gn
2012/6/3 21:43	2012/6/3 21:43	DeletePDP	62878****3789	5.00E+39	794B	WWW.XLGPRS.NET	SAMSUNG	GT-C3222	3G Mobile Phone	succeed	Gn
2012/6/4 9:05	2012/6/4 9:05	CreatePDP	62878****3789	5.00E+39	84FB	WWW.XLMMS.NET	SAMSUNG	GT-C3222	3G Mobile Phone	succeed	Gn
2012/6/4 9:06	2012/6/4 9:06	DeletePDP	62878****3789	5.00E+39	84FB	WWW.XLMMS.NET	SAMSUNG	GT-C3222	3G Mobile Phone	succeed	Gn
2012/6/4 14:26	2012/6/4 14:26	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 14:26	2012/6/4 14:26	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 14:26	2012/6/4 14:26	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 14:26	2012/6/4 14:26	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 14:26	2012/6/4 14:26	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 14:26	2012/6/4 14:26	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 14:35	2012/6/4 14:35	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 14:38	2012/6/4 14:38	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 14:38	2012/6/4 14:38	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 14:38	2012/6/4 14:38	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 15:01	2012/6/4 15:01	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 15:02	2012/6/4 15:02	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 15:03	2012/6/4 15:03	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 16:24	2012/6/4 16:24	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 21:19	2012/6/4 21:19	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 21:22	2012/6/4 21:22	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn
2012/6/4 21:22	2012/6/4 21:22	CreatePDP	62878****3789	5.00E+39	956B	XLUNLIMITED	SAMSUNG	GT-C3303I	2G Mobile Phone	reject	Gn

HUAWEI

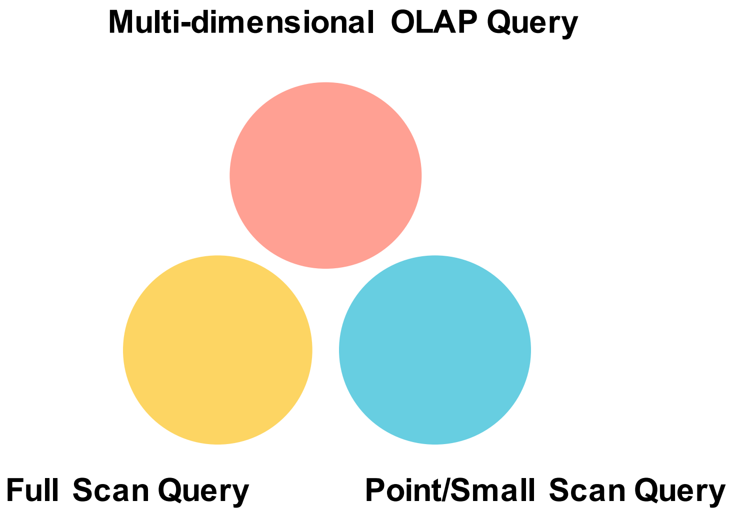
Challenge - Data

- Data Size
  - Single Table >10 B
  - Fast growing
- Multi-dimensional
  - Every record > 100 dimension
  - Add new dimension occasionally
- Rich of Detail
  - Billion level high cardinality
  - 1B terminal * 200K cell * 1440 minutes = 28800 (万亿)
  - Nested data structure for complex object

Challenge - Application

- Enterprise Integration
  - SQL 2003 Standard Syntax
  - BI integration, JDBC/ODBC

- Flexible Query
  - Any combination of dimensions
  - OLAP Vs Detail Record
  - Full scan Vs Small scan
  - Precise search & Fuzzy search



How to choose storage?

NoSQL Database

Key-Value store: low latency, <5ms

Type	Examples
Key-Value Store	redis, riak
Wide Column Store	Apache HBase, Cassandra

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Multi-dimensional problem

Pre-compute all aggregation combinations

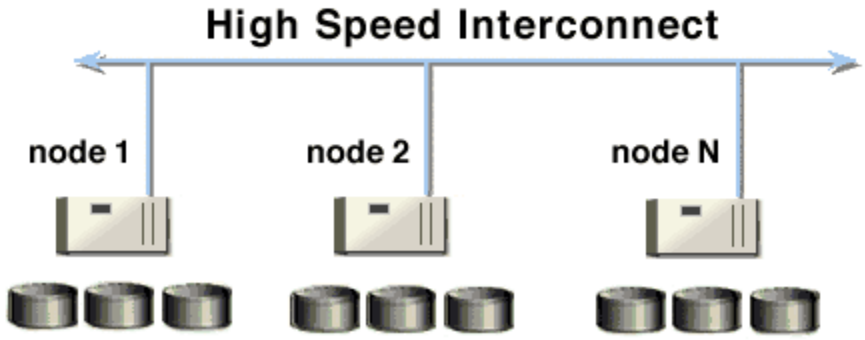
Complexity: O(2^n)

- Dimension < 10
- Too much space
- Slow loading speed

- Row Keys: identify the rows in an HBase table.

	Row Key	CF1			CF2				...
		colA	colB	colC	colA	colB	colC	colD	
R1	axxx	val		val	val			val	
	...								
	gxxx	val			val	val	val		
R2	hxxx	val	val	val	val	val	val	val	
	...								
	jxxx	val							
R3	kxxx	val		val	val			val	
	...								
	rxxx	val	val	val	val	val	val		
...	sxxx	val						val	

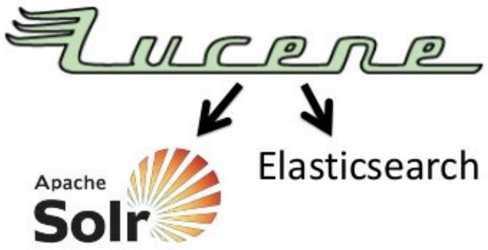
Shared nothing database



- Parallel scan + distributed compute
- Multi-dimensional OLAP
  - Index management problem
- Questionable scalability and fault-tolerance
  - Cluster size < 100 data node
  - Not suitable for big batch job

Search engine

- All column indexed
- Fast searching
- Simple aggregation



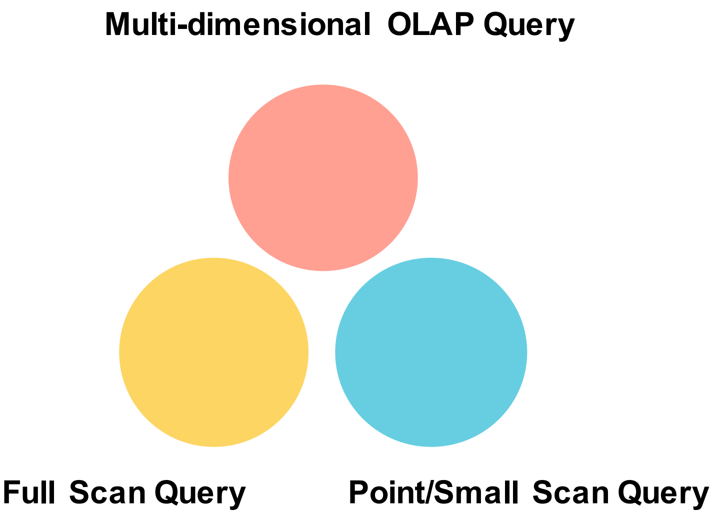
- Designed for search but not OLAP
  - complex computation: TopN, join, multi-level aggregation
- 3~4X data expansion in size
- No SQL support

SQL on Hadoop

- Modern distributed architecture, scale well in computation.
  - Pipeline based: Impala, Drill, Flink, ...
  - BSP based: Hive, SparkSQL

- BUT, still using file format designed for batch job
  - Focus on scan only
  - No index support, not suitable for point or small scan queries

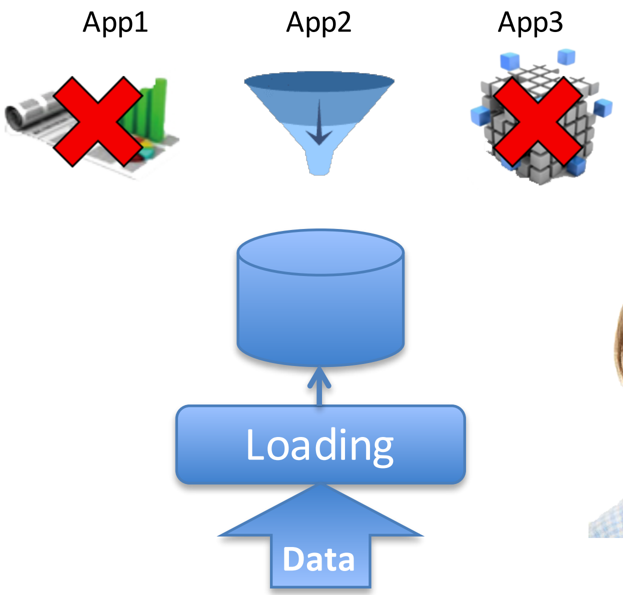
Capability Matrix

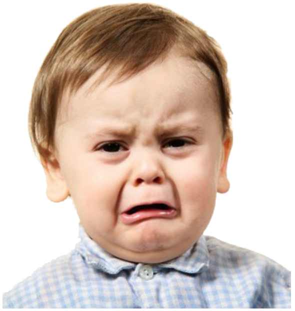


Type	Store	Good	Bad
KV Store	HBase, Cassandra, …		
Parallel database	Greenplum, Vertica, …		
Search engine	Solr, ElasticSearch, …		
SQL on Hadoop - Pipeline	Impala, HAWQ, Drill, …		
SQL on Hadoop - BSP	Hive, SparkSQL		

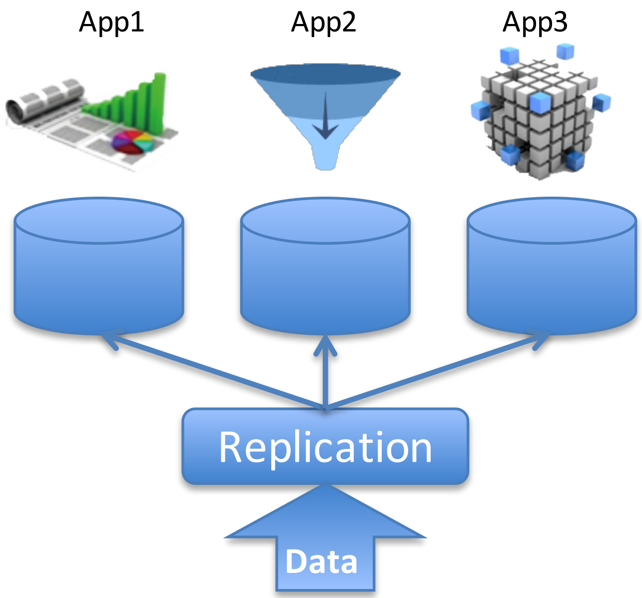
Architect’ s choice

Choice 1: Compromising



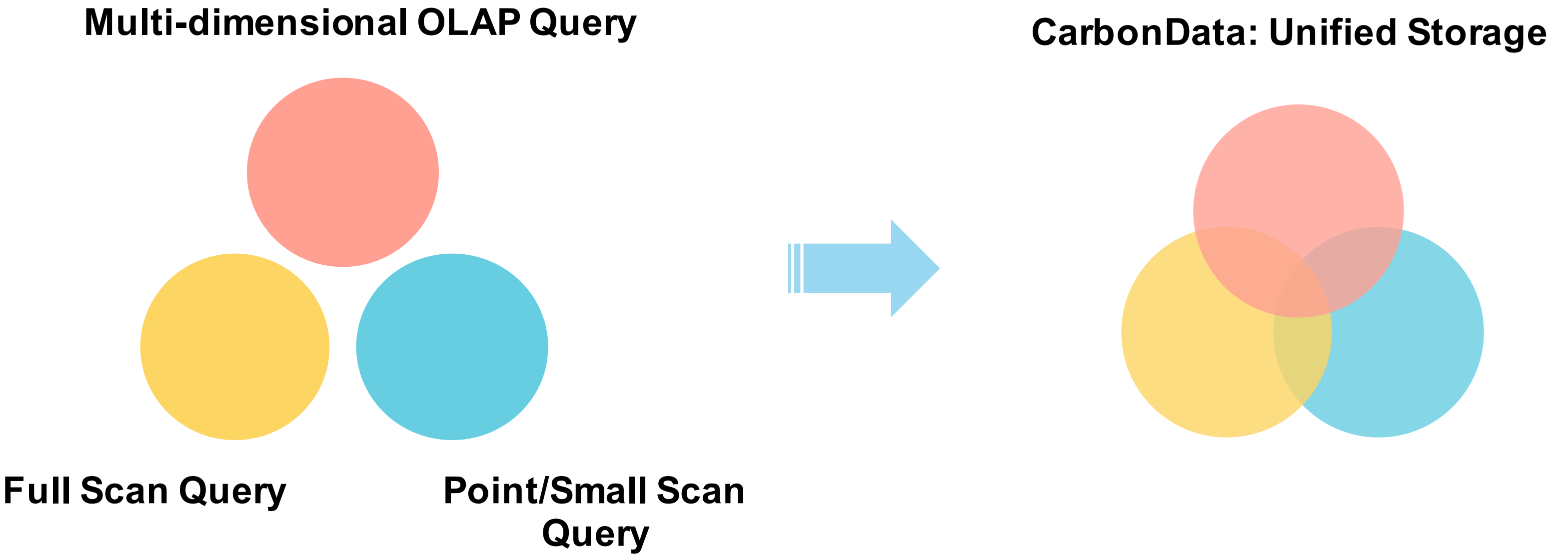


Choice 2: Replicating of data



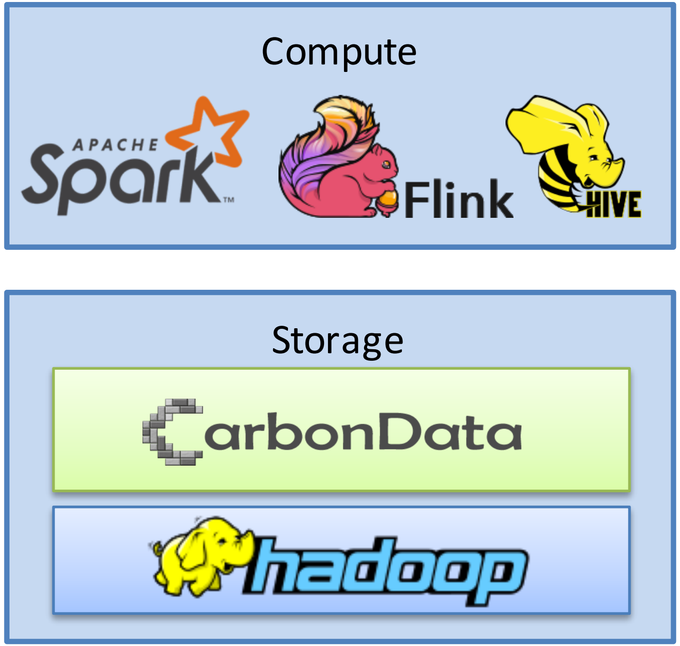
CarbonData: An Unified Data Storage in Hadoop Ecosystem

Motivation


Multi-dimensional OLAP Query
Full Scan Query
Point/Small Scan Query
CarbonData: Unified Storage

Apache CarbonData

- An Apache Incubating Project
  - Incubation start in June, 2016
- Goal：
  - Make big data simple
  - High performance
- Current Status：
  - First stable version released
  - Focus on indexed columnar file format
  - Deep query optimization with Apache Spark



Community

- First Stable Version Released in Aug, 2016!

- Welcome contribution:
  - Code: https://github.com/apache/incubator-carbondata
  - JIRA: https://issues.apache.org/jira/browse/CARBONDATA
  - Maillist: dev@carbondata.incubator.apache.org

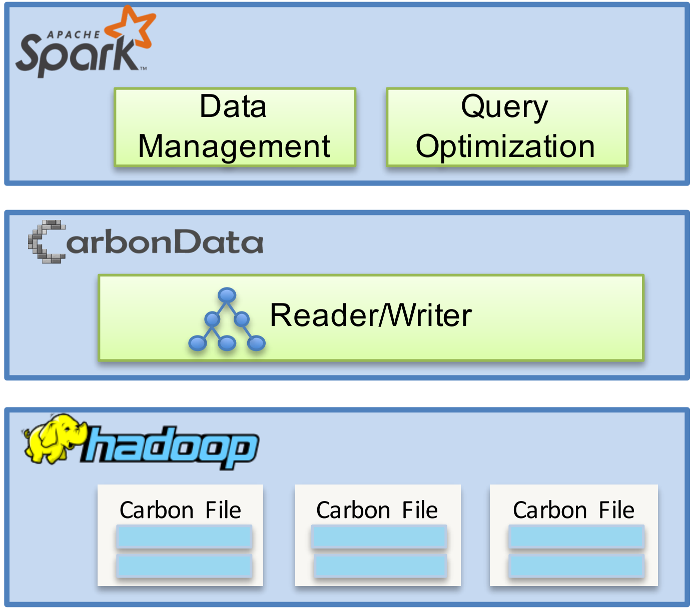
- Contributor from: Huawei, Talend, Intel, eBay, Inmobi, MeiTuan(美团)

Feature Introduction

- Table level feature
- File level feature

Carbon-Spark Integration

- Built-in Spark integration
  - Spark 1.5, 1.6
- Interface
  - SQL
  - DataFrame API
  - Query Optimization
- Data Management
  - Bulk load/Incremental load
  - Delete load
  - Compaction



Integration with Spark

- Query CarbonData Table
  - DataFrame API

```
carbonContext.read
             .format("carbondata")
             .option("tableName", "table1")
             .load()
```

With late decode optimization and carbon-specific SQL command support

```
sqlContext.read
          .format("carbondata")
          .load("path_to_carbon_file")
```

  - Spark SQL Statement

```
CREATE TABLE IF NOT EXISTS T1 (name String, PhoneNumber String) STORED BY
"carbondata"

LOAD DATA LOCAL INPATH 'path/to/data' INTO TABLE T1
```

- Support schema evolution of Carbon table via *ALTER TABLE*
  - Add, Delete or Rename Column

HUAWEI

Data Ingestion

- Bulk Data Ingestion
  - CSV file conversion
  - MDK clustering level: load level vs. node level

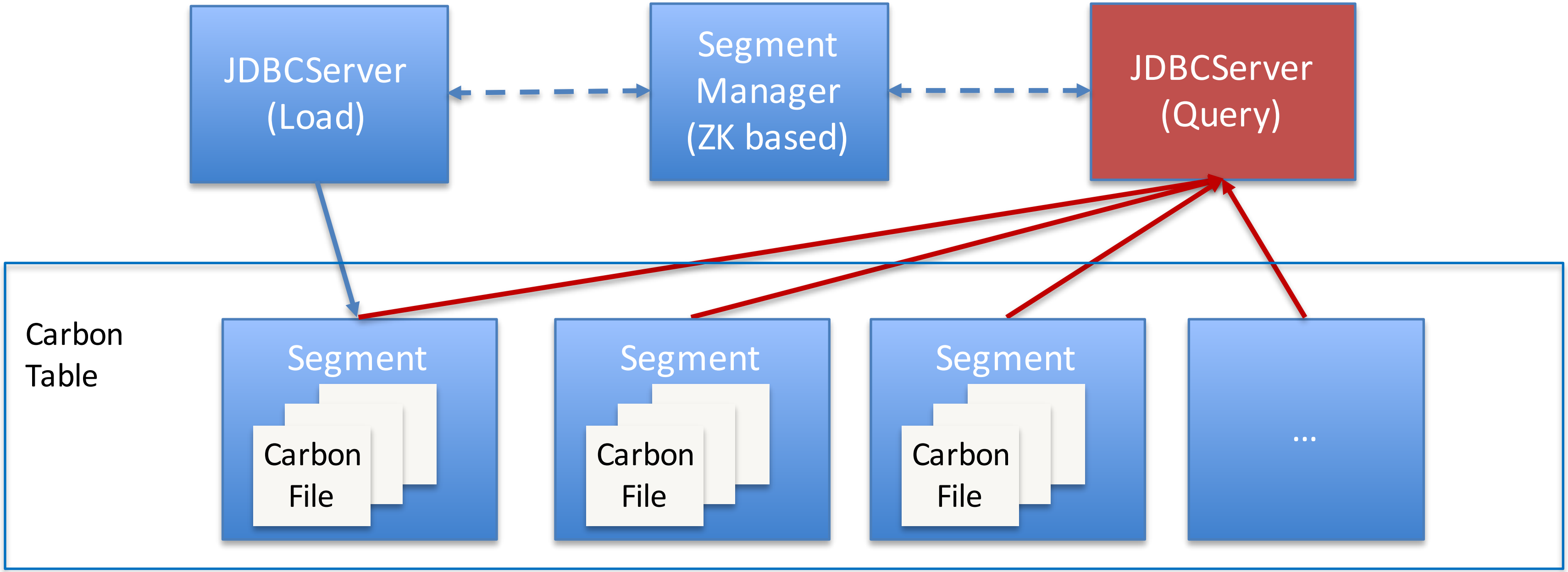
```
LOAD DATA [LOCAL] INPATH 'folder path'
[OVERWRITE] INTO TABLE tablename
OPTIONS(property_name=property_value, ...)

INSERT INTO TABLE tablennme AS
select_statement1 FROM table1;
```

- Save Spark dataframe as Carbon data file

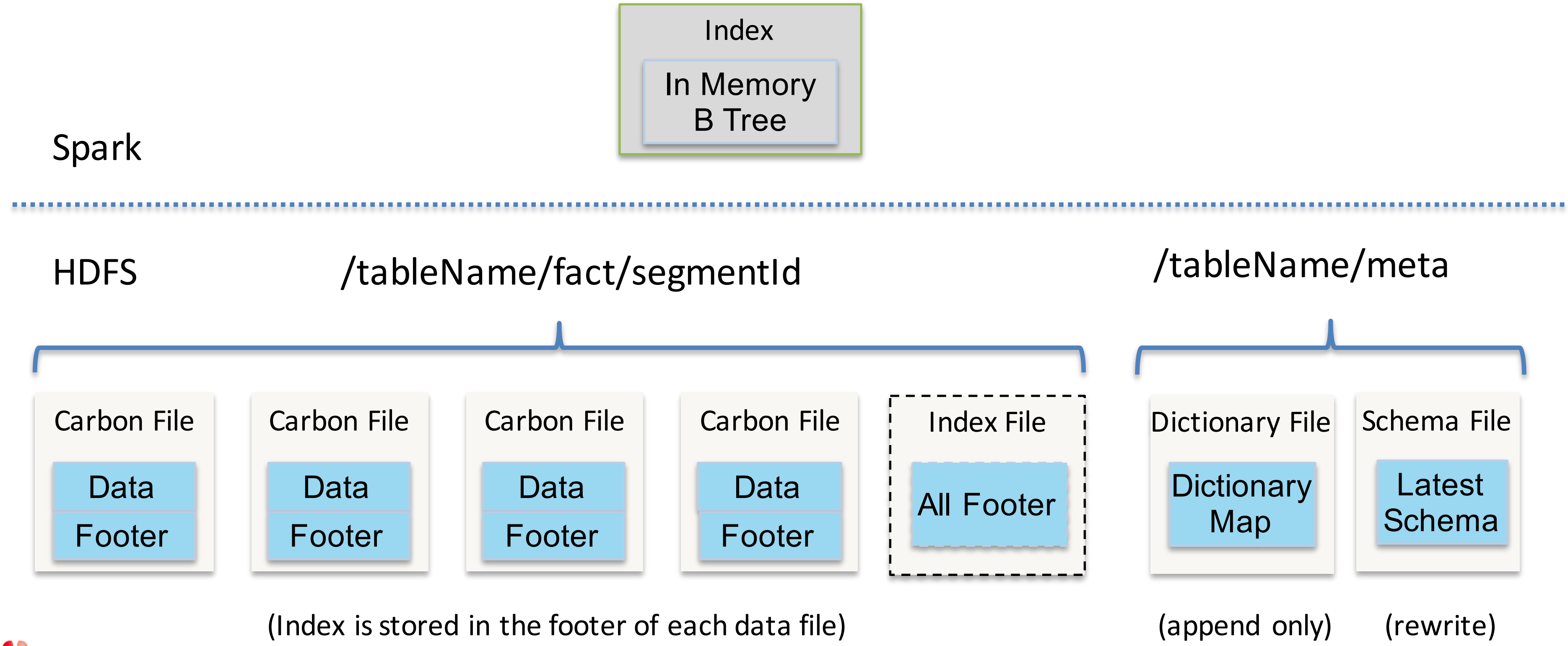
```
df.write
    .format("carbondata")
    .options("tableName", "tbl1"))
    .mode(SaveMode.Overwrite)
    .save()
```

Segment

Every data load becomes one segment in CarbonData table, data is **sorted** within one segment.



CarbonData Table Organization



Data Compaction

- Data compaction is used to merge small files
  - Re-clustering across loads for better performance

- Two types of compactions supported
  - Minor compaction
    - Compact adjacent segment based on number of segment
  - Major compaction
    - Compact segments based on size

```
ALTER TABLE [db_name.]table_name COMPACT 'MINOR/MAJOR'
```

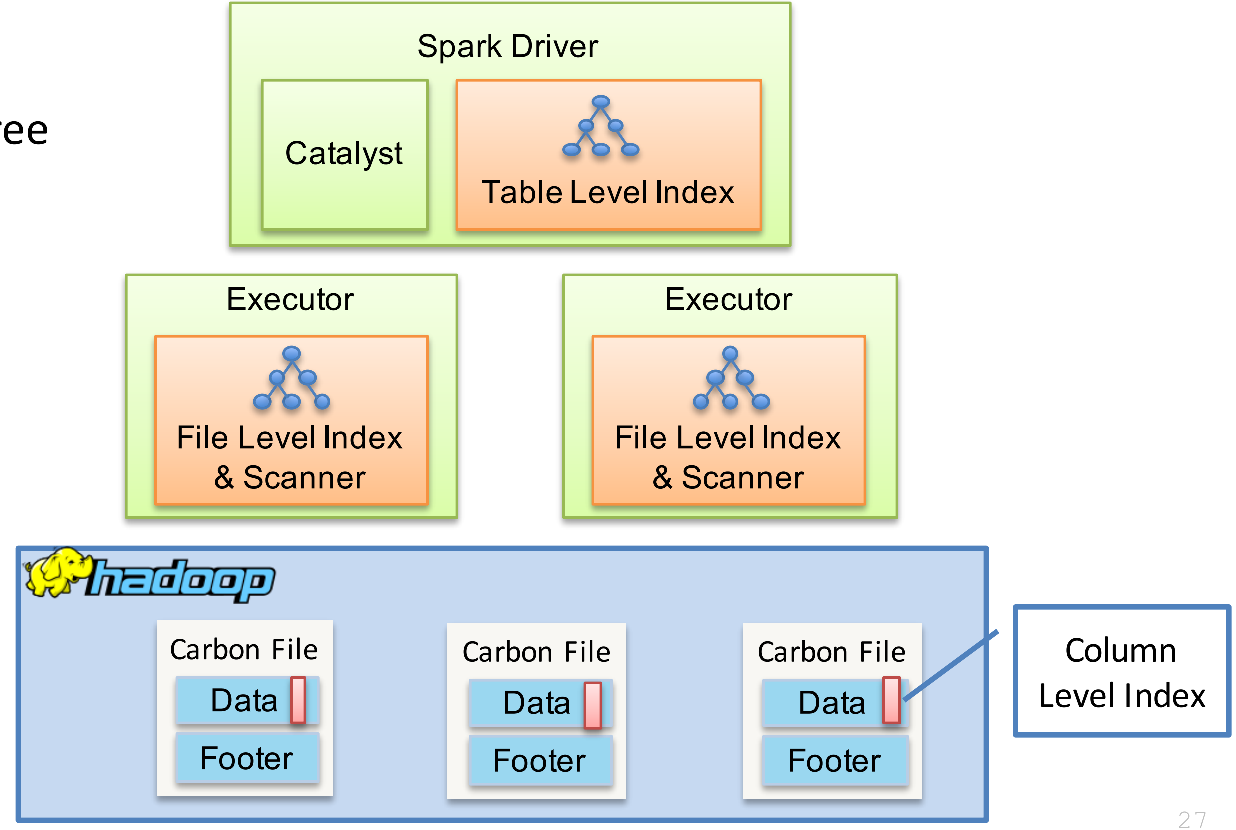
Query Optimization: Index

Multi-level indexes:

•Table level index: global B+ tree index, used to filter blocks

•File level index: local B+ tree index, used to filter blocklet

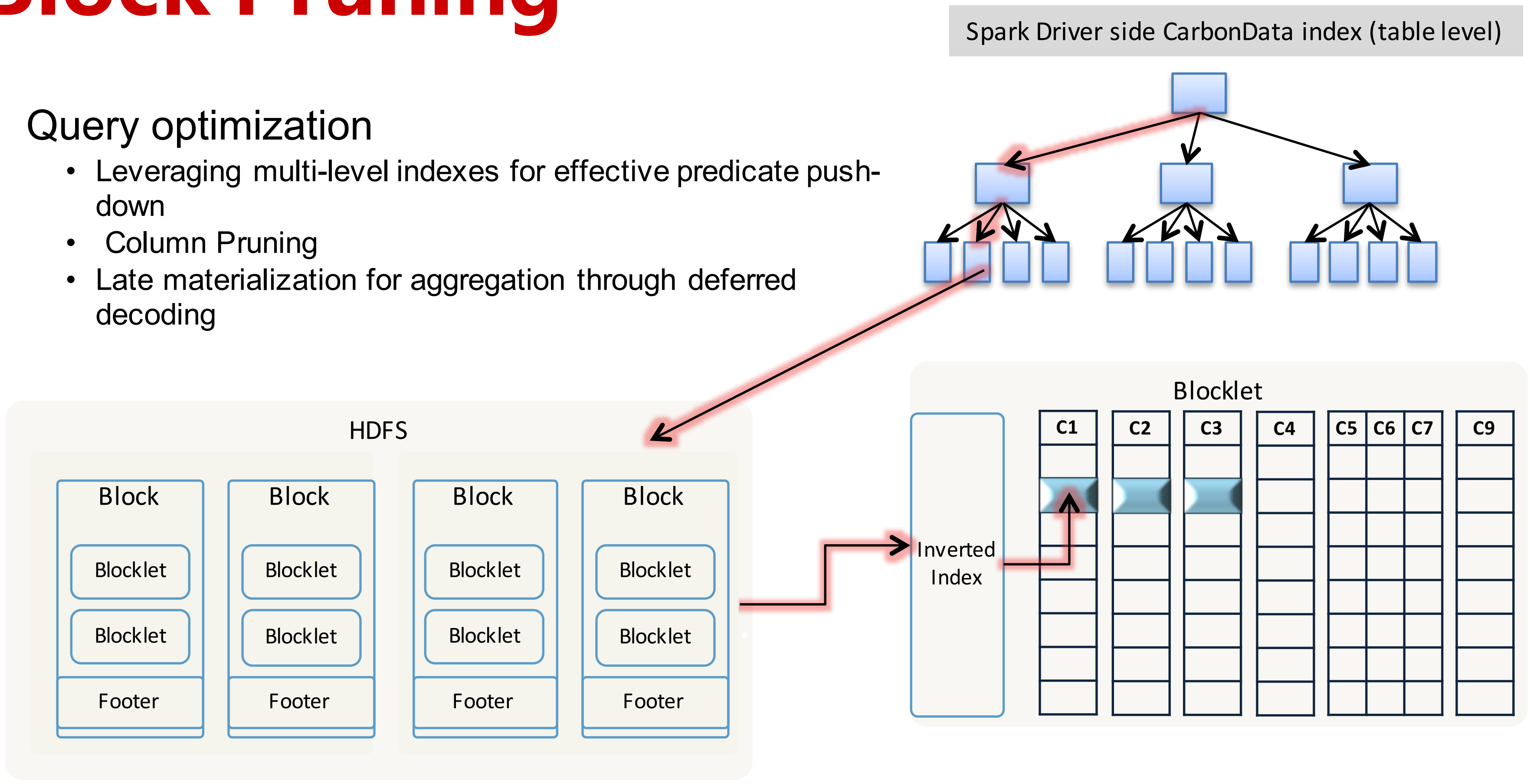
•Column level index: inverted index within column chunk



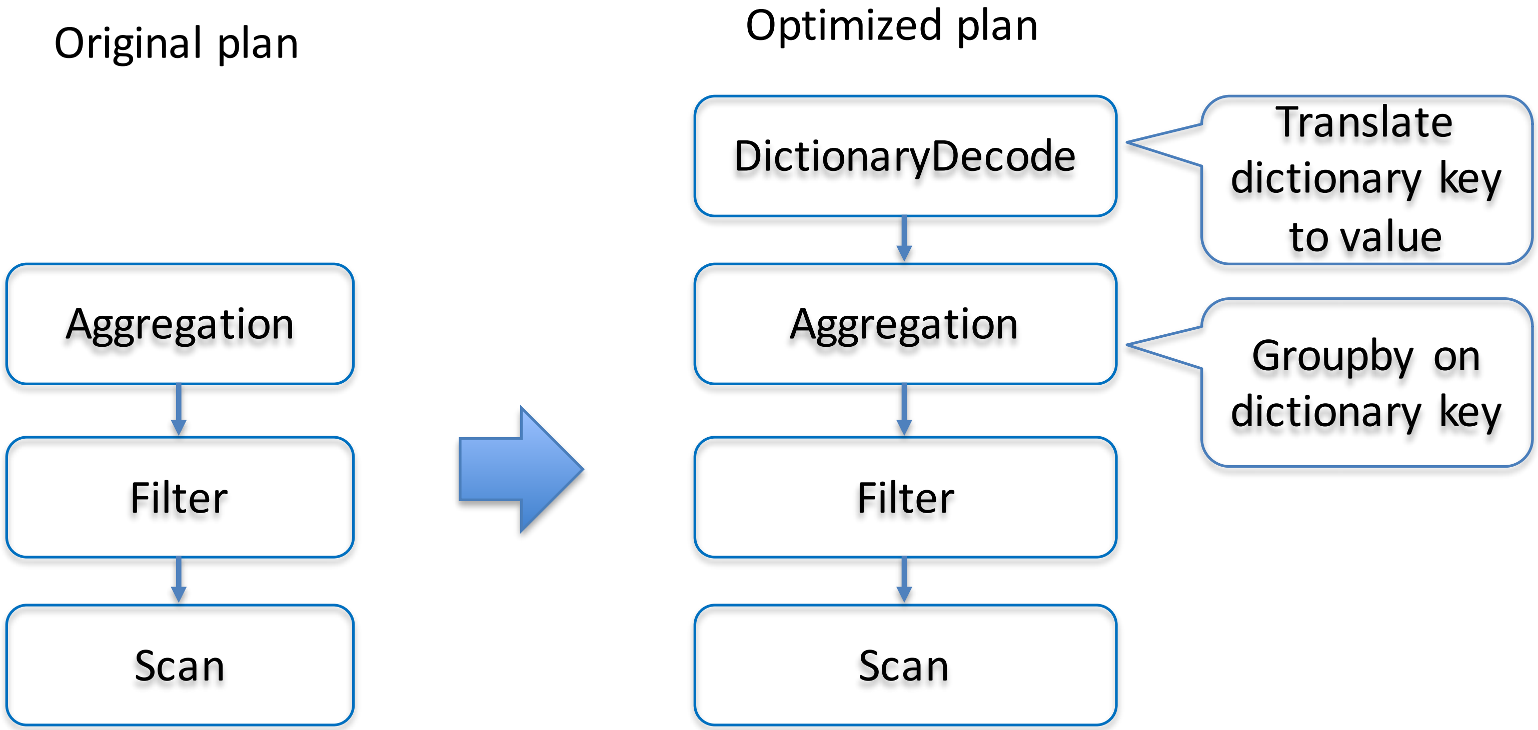
HUAWEI

Block Pruning

- Query optimization
  - Leveraging multi-level indexes for effective predicate push-down
  - Column Pruning
  - Late materialization for aggregation through deferred decoding

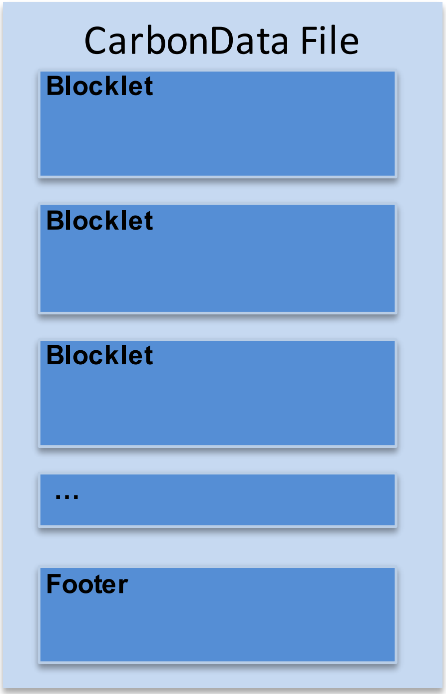


Query Optimization: Late Decode

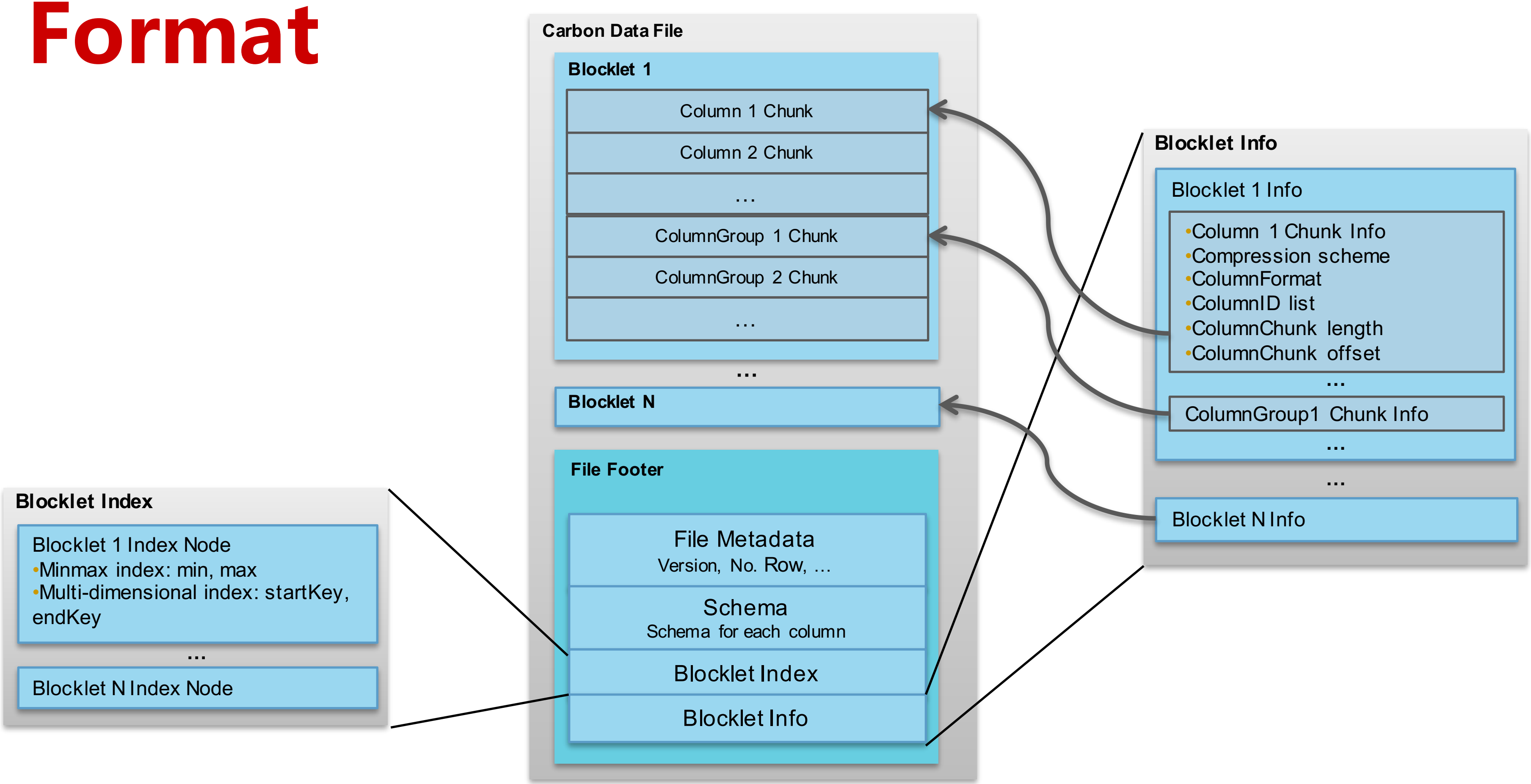


CarbonData File Structure

- **Built-in Columnar & Index**
  - Store index and data in the same file, co-located in HDFS
  - Balance between batch and point query
- **Index support:**
  - Multi-dimensional Index (B+ Tree)
  - Min/Max index
  - Inverted index
- **Encoding:**
  - Dictionary, RLE, Delta
  - Snappy for compression
- **Data Type:**
  - Primitive type and nested type
- **Schema Evolution:**
  - Add, Remove, Rename columns



Format



HUAWEI

Encoding Example

- Data are sorted along MDK (multi-dimensional keys)
- Data stored as index in columnar format

Years	Quarters	Months	Territory	Country	Quantity	Sales
2003	QTR1	Jan	EMEA	Germany	142	11,432
2003	QTR1	Jan	APAC	China	541	54,702
2003	QTR1	Jan	EMEA	Spain	443	44,622
2003	QTR1	Feb	EMEA	Denmark	545	58,871
2003	QTR1	Feb	EMEA	Italy	675	56,181
2003	QTR1	Mar	APAC	India	52	9,749
2003	QTR1	Mar	EMEA	UK	570	51,018
2003	QTR1	Mar	Japan	Japan	561	55,245
2003	QTR2	Apr	APAC	Australia	525	50,398
2003	QTR2	Apr	EMEA	Germany	144	11,532

Encoding

[1,1,1,1,1] : [142,11432]
[1,1,1,3,2] : [541,54702]
[1,1,1,1,3] : [443,44622]
[1,1,2,1,4] : [545,58871]
[1,1,2,1,5] : [675,56181]
[1,1,3,3,6] : [52,9749]
[1,1,3,1,7] : [570,51018]
[1,1,3,2,8] : [561,55245]
[1,2,4,3,9] : [525,50398]
[1,2,4,1,1] : [144,11532]

Sort (MDK Index)

Sorted MDK Index

[1,1,1,1,1] : [142,11432]
[1,1,1,1,3] : [443,44622]
[1,1,1,3,2] : [541,54702]
[1,1,2,1,4] : [545,58871]
[1,1,2,1,5] : [675,56181]
[1,1,3,1,7] : [570,51018]
[1,1,3,2,8] : [561,55245]
[1,1,3,3,6] : [52,9749]
[1,2,4,1,1] : [144,11532]
[1,2,4,3,9] : [525,50398]

Column split

Blocklet Logical View

C1	C2	C3	C4	C5	C6	C7
1	1	1	1	1	142	11432
1	1	1	1	3	443	44622
1	1	1	3	2	541	54702
1	1	2	1	4	545	58871
1	1	2	1	5	675	56181
1	1	3	1	7	570	51018
1	1	3	2	8	561	55245
1	1	3	3	6	52	9749
1	2	4	1	1	144	11532
1	2	4	3	9	525	50398

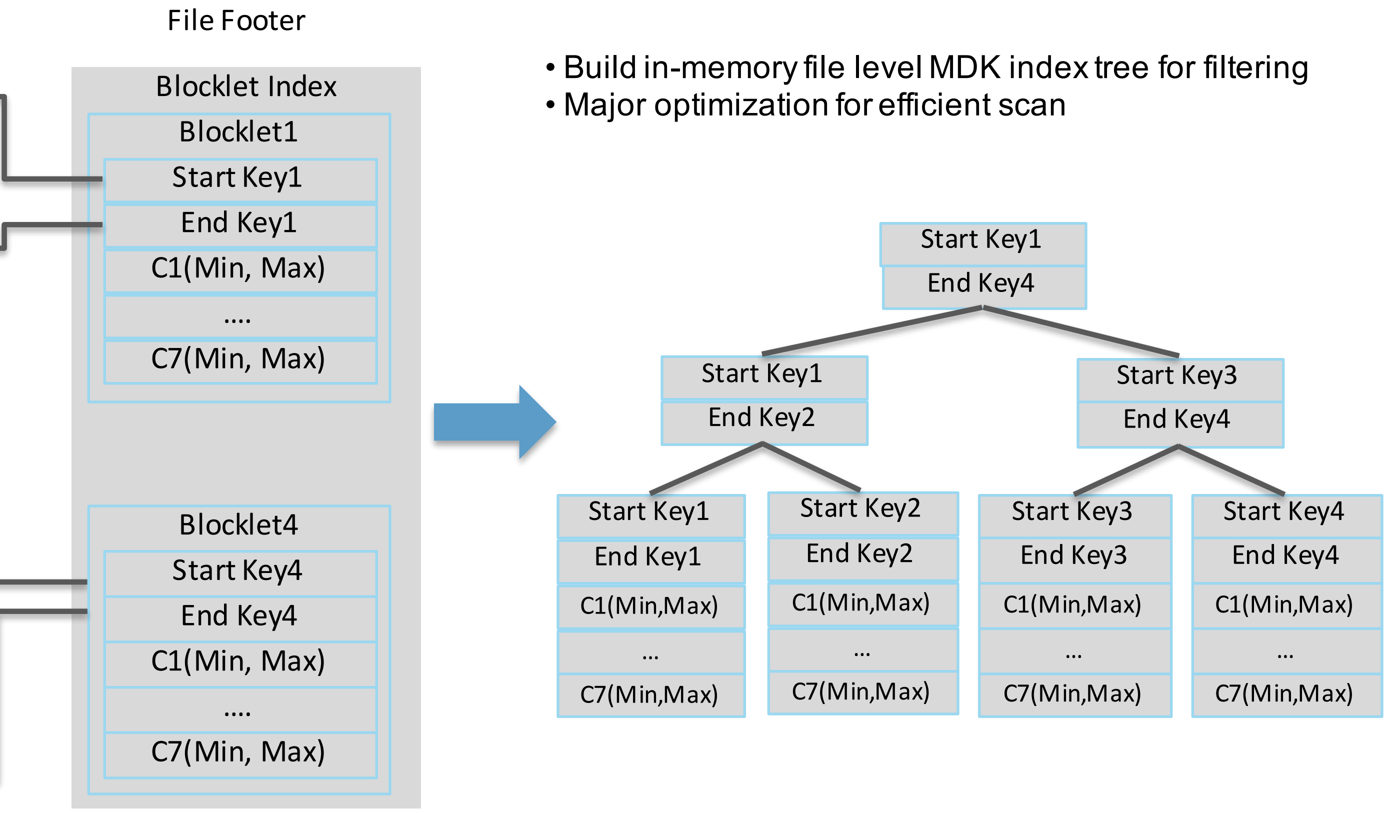
HUAWEI

File Level Blocklet Index

- Build in-memory file level MDK index tree for filtering
- Major optimization for efficient scan

Blocklet

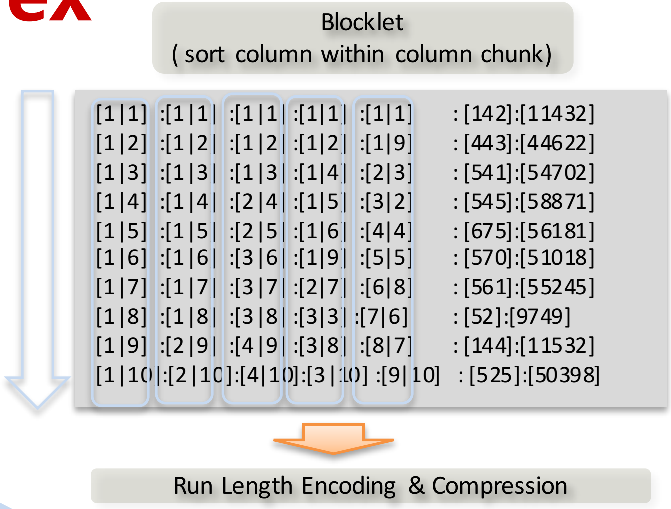
Blocklet 1						
1	1	1	1	1	1	12000
1	1	1	2	1	2	5000
1	1	2	1	1	1	12000
1	1	2	2	1	2	5000
1	1	3	1	1	1	12000
1	1	3	2	1	2	5000
Blocklet 2						
1	2	1	3	2	3	11000
1	2	2	3	2	3	11000
1	2	3	3	2	3	11000
1	3	1	4	3	4	2000
1	3	1	5	3	4	1000
1	3	2	4	3	4	2000
Blocklet 3						
1	3	2	5	3	4	1000
1	3	3	4	3	4	2000
1	3	3	5	3	4	1000
1	4	1	4	1	1	20000
1	4	2	4	1	1	20000
1	4	3	4	1	1	20000
Blocklet 4						
2	1	1	1	1	1	12000
2	1	1	2	1	2	5000
2	1	2	1	1	1	12000
2	1	2	2	1	2	5000
2	1	3	1	1	1	12000
2	1	3	2	1	2	5000

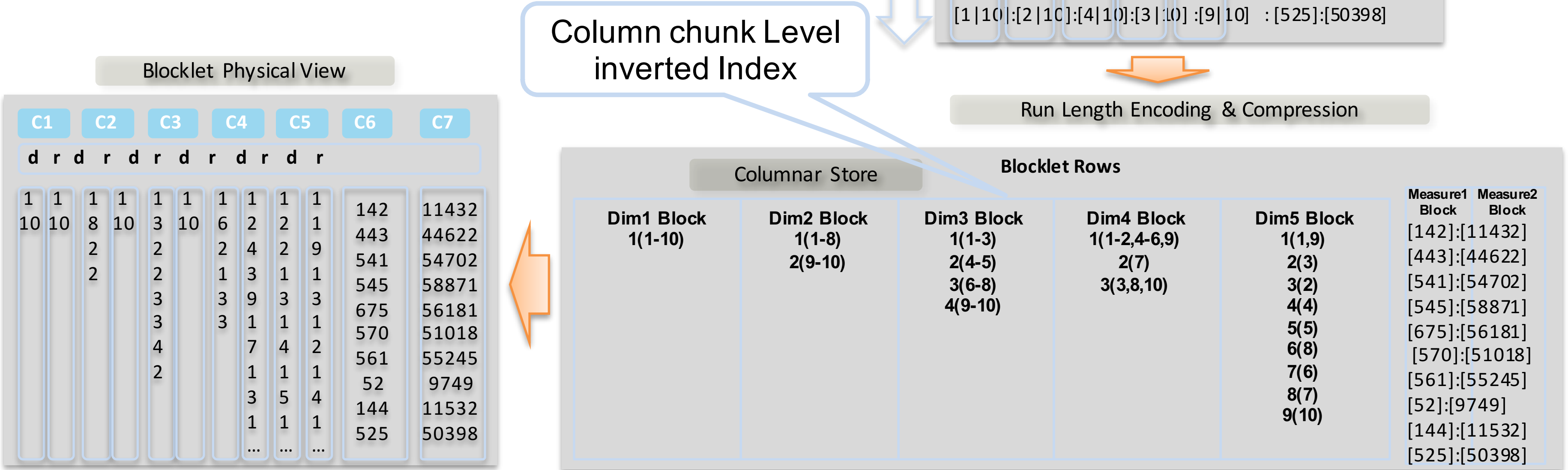


HUAWEI

Column Chunk Inverted Index

- Optionally store column data as inverted index within column chunk
  - suitable to low cardinality column
  - better compression & fast predicate filtering





HUAWEI

Column Group

- Allow multiple columns form a column group
  - stored as a single column chunk in row-based format
  - suitable to set of columns frequently fetched together
  - saving stitching cost for reconstructing row

Blocklet 1

C1	C2	C3	C4	C5	C6
Col Chunk	Col Chunk	Col Chunk	Col Group Chunk		Col Chunk
10	2	23	23	38	15.2
10	2	50	15	29	18.5
10	3	51	18	52	22.8
11	6	60	29	16	32.9
12	8	68	32	18	21.6

HUAWEI

Nested Data Type Representation

Arrays

- Represented as a composite of two columns
- One column for the element value
- One column for start index & length of Array

Struts

- Represented as a composite of finite number of columns
- Each struct element is a separate column

Name	Array<Ph_Number>
John	[192,191]
Sam	[121,345,333]
Bob	[198,787]

Name	Array [start,len]	Ph_Number
John	0,2	192
Sam	2,3	191
Bob	5,2	121
		345
		333
		198
		787

Name	Info Strut<age,gender>
John	[31,M]
Sam	[45,F]
Bob	[16,M]

Name	Info.age	Info.gender
John	31	M
Sam	45	F
Bob	16	M

Encoding & Compression

- Efficient encoding scheme supported:
  - DELTA, RLE, BIT_PACKED
  - Dictionary:
    - medium high cardinality: file level dictionary
    - low cardinality: table level global dictionary
  - CUSTOM
- Compression Scheme: Snappy

Big Win:

- Speedup Aggregation
- Reduce run-time memory footprint
- Enable deferred decoding
- Enable fast distinct count

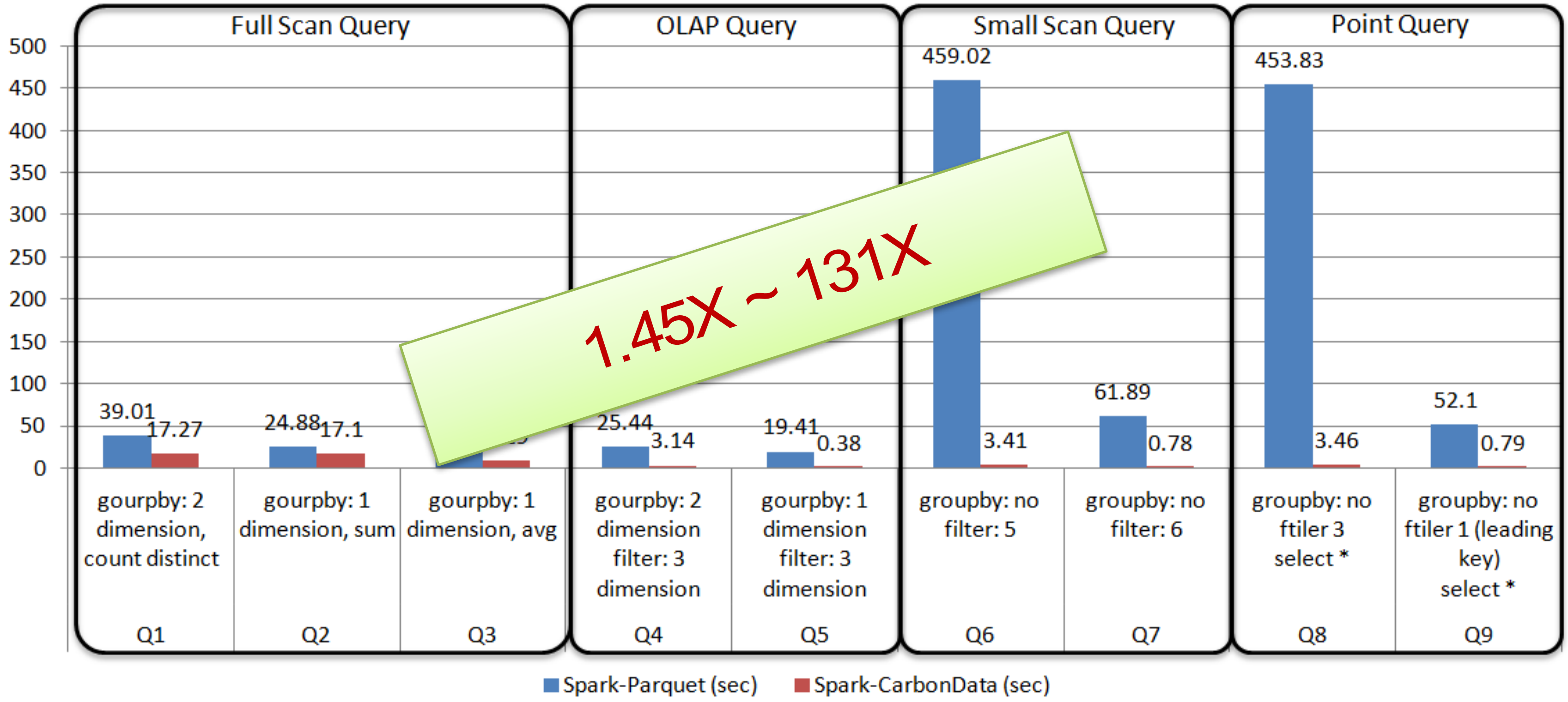
HUAWEI

Performance Test

- Real world test cases from Telecom, Finance, Government domain
  1. Point query: filter with leading key
  2. Small scan query: filter with multiple columns
  3. Full scan query: complex aggregation & join, no filter
  4. OLAP query: some filter, some aggregation

- Test environment
  - Cluster: 3 workers + 1 master, 40 cores, 384GB, 10GE network
  - Software: Hadoop 2.7.2, Spark 1.5.2
  - Data: 1 Billion record * 300 columns, totally 1.9TB original data

Performance comparison



HUAWEI

What’s coming next:

- Streaming Ingest:
  - Introduce row-based format for fast ingestion
  - Gradually compact row-based to column-based for analytic workload
  - Integrate with Kafka Connect, Spark Streaming
- Batch Update:
  - Support daily update scenario for OLAP, Not OLTP!
  - Base+Delta file based design
- Broader Integration across Hadoop-ecosystem: Flink, Hive, Presto

What’s coming Next:

- Support pre-computed aggregation table
  - Speed up OLAP query, eliminating aggregate and join in query time
  - Enhance Optimizer to make smart choice based on workload

- Upgrade to Apache Spark 2.0
  - Embrace powerful features introduced in Spark 2.0
  - Challenges encountered:
    - Requires parser extension interface
    - Advanced Carbondata ‘s optimization tightly coupled with Catalyst

Tuning Hint

Index Tuning

- Default Behavior:
  - column order in CREATE TABLE is the MDK order
- Rule 1: make column order as cardinality order
  - Table schema: from low cardinality to high cardinality
  - Can achieve very good compression ratio
  - Example: col1(low), col2(low), col3(median), col4(high), …
- Rule 2: move frequently filter column to left
  - In MDK, left side dimension’ s filter efficiency is better
  - Example: col1(high), col2(low), col3(median), …

Dictionary Tuning

- Default Behavior:
  - Carbon will do cardinality discovery for the first data load.
  - Will do dictionary if cardinality < 1 million

- Rule 1: use DICTIONARY_INCLUDE and DICTIONARY_EXCLUDE option in CREATE TABLE
  - If not enough memory, use it to not doing dictionary for some columns

Data Load Tuning

- One executor per node:
  - Sort all files within one node will have better query performance
  - More memory for sorting

- IO:
  - Set **carbon.use.local.dir** = true, Carbon will use YARN local directories for multi-table load disk load balance
  - Load from compressed CSV (gzip, bz2)

- CPU:
  - **carbon.number.of.cores.while.loading**: thread pool size for loading

- Memory:
  - **carbon.sort.file.buffer.size**: sort file size
  - **carbon.sort.intermediate.files.limit**: number of files before multi-level merge sort

HUAWEI

Query Tuning

- One executor per node:
  - More memory for index and dictionary.
  - After block/blocklet pruning by index, data read is less than Parquet. GC is controllable.

- IO :
  - HDFS block size: 256MB ~ 1 GB
  - **carbon.detail.batch.size:** This is the minimum rows to read, set it according to LIMIT query

- CPU :
  - **spark.sql.shuffle.partitions:** set it to 1~2X of number of executors, to reduce the number of reducer

- Do compaction:
  - If there are many segments, query performance may slow down
  - Do compaction to make data sorted: minor compaction is triggered automatically, major compaction is trigger manually by ALTER TABLE

More

Configurations For Optimizing CarbonData Performance

https://cwiki.apache.org/confluence/display/CARBONDATA/Configurations+For+Optimizing+CarbonData+Performance

Thank you

www.huawei.com