WEBINAR

ClickHouse and Apache Parquet

Past, Present, and Future

Alexander Zaitsev
Altinity co-founder and CTO
ClickHouse expert since 2016

7:00 AM PT Tuesday 30 January

A brief message from our sponsor…

Altinity Engineering

ClickHouse evangelists, experts and contributors

Alexander Zaitsev

Expert in high scale analytics systems design and implementation. Altinity CTO

ClickHouse support and services: Altinity.Cloud and Altinity Stable Builds
Authors of Altinity Kubernetes Operator for ClickHouse

Introducing ClickHouse and Parquet

Meet ClickHouse. It's a real-time analytic database

Understands SQL

Runs on bare metal to cloud

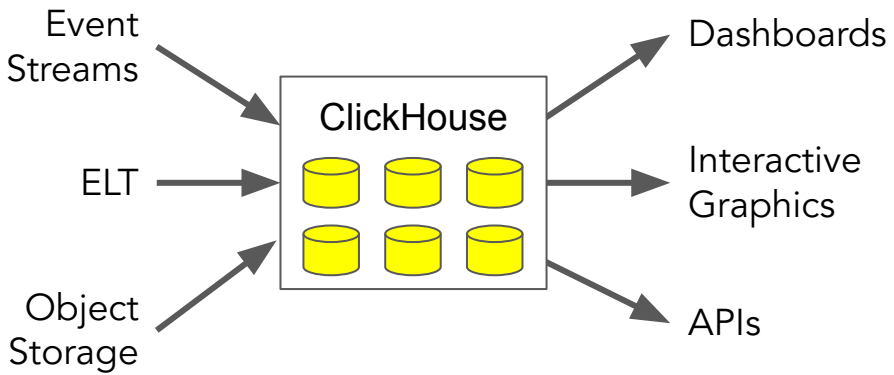
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)



It's the core engine for low-latency analytics

ClickHouse main engine = MergeTree

```
CREATE TABLE test (
    `A` Int64,
    `S` String,
    `D` Date
)
ENGINE = MergeTree
PARTITION BY D
ORDER BY A;

INSERT INTO test
SELECT number, number,
'2023-01-01' FROM numbers(1e8);
```

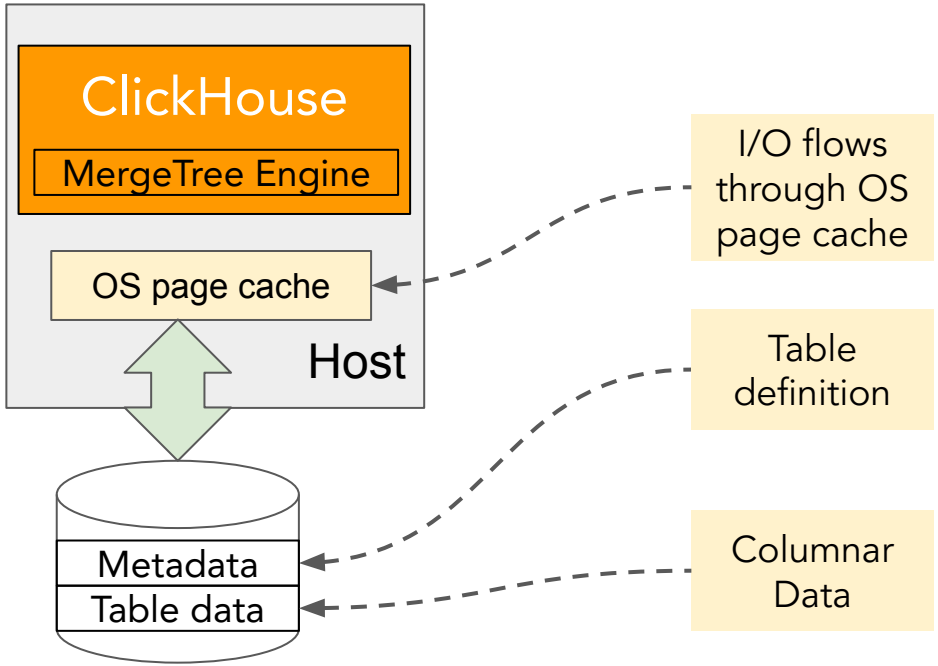
Use MergeTree for “big data”

Divide table into parts by day

Order data in parts by A value

Load 100M rows of test data

ClickHouse main engine = MergeTree



- ✓ Columnar
- ✓ Column encodings
- ✓ Compression
- ✓ ORDER + sparse index for efficient queries

Meet Apache Parquet

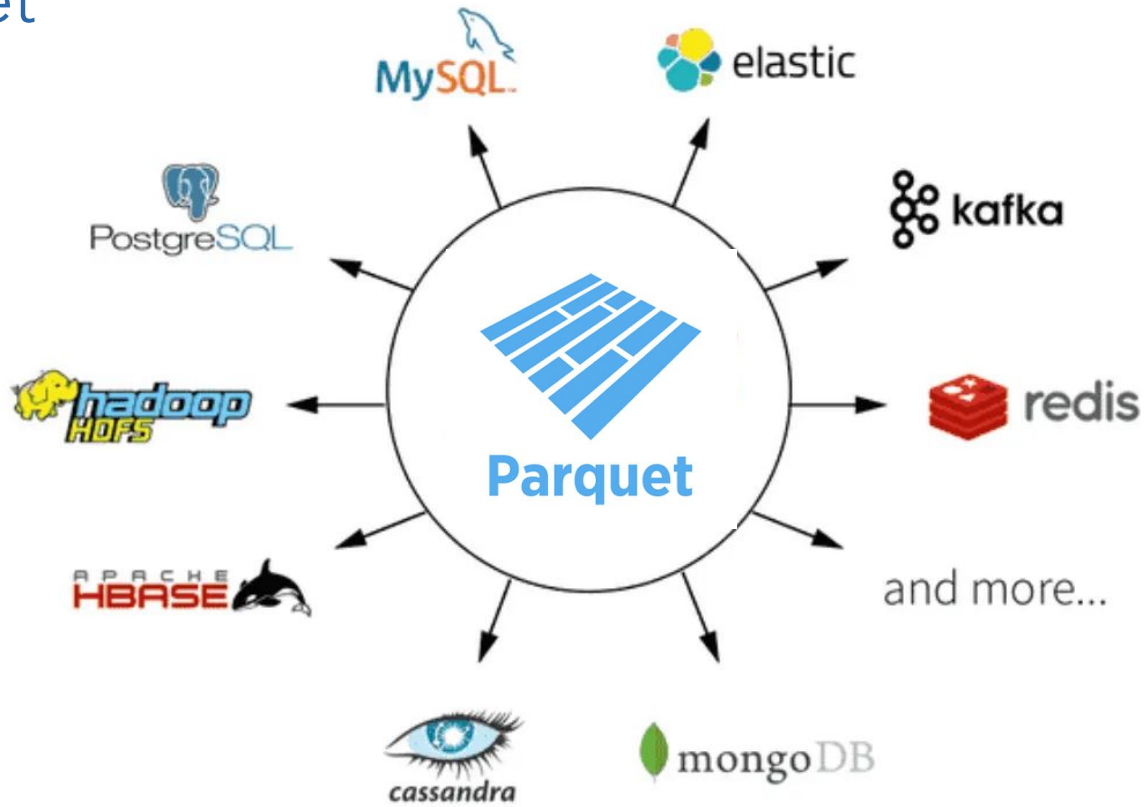
This is a file format

Designed for Hadoop Ecosystem

Widely used in data lakes

Can be consumed by many tools

Extensible



Parquet format

- ✓ Columnar
- ✓ Column encodings
- ✓ Compression
- ✓ Column metadata for efficient queries

MergeTree

- ✓ Columnar
- ✓ Column encodings
- ✓ Compression
- ✓ ORDER + sparse index for efficient queries

Looks similar, doesn't it?

Reading and Writing Parquet Files

Writing Parquet files

```
INSERT INTO FUNCTION file('ontime.parquet')
SELECT * FROM ontime
```

```
/var/lib/clickhouse/user_files/
total 3010580
-rw-r-----  1 clickhouse clickhouse 5366985658 Jan 29 18:10
ontime.parquet
```

Location is configured at
<user_files_path>/var/lib/clickhouse/user_files/</user_files_path>

Writing Parquet files

```
INSERT INTO FUNCTION file('ontime_{_partition_id}.parquet')
PARTITION BY Year
SELECT * FROM ontime
```

_partition_id macro allows to generate multiple files

```
-rw-r----- 1 clickhouse clickhouse  42991616 Jan 29 18:22
/var/lib/clickhouse/user_files/ontime_1988.parquet
-rw-r----- 1 clickhouse clickhouse  30408704 Jan 29 18:23
/var/lib/clickhouse/user_files/ontime_1989.parquet
-rw-r----- 1 clickhouse clickhouse  30408704 Jan 29 18:21
/var/lib/clickhouse/user_files/ontime_1992.parquet
. . .
```

Other Ways Writing Parquet files

CH ver 23.5+

```
SELECT * FROM ontime INTO OUTFILE 'ontime.parquet'
```

A lot of flexibility for data exports:

SELECT <expr_list> INTO OUTFILE file_name
[AND STDOUT] [APPEND | TRUNCATE] [COMPRESSION type [LEVEL level]]

WARNING !!! Writes to the client file system!

Let's query

```
SELECT count() FROM file('ontime.parquet')

┌──count()─┐
│ 201575308 │
└──────────┘

1 row in set. Elapsed: 0.051 sec.
```

Standard globs syntax

```
SELECT count() FROM file('ontime_*.parquet')

1 row in set. Elapsed: 0.013 sec.
```

Note that it is faster!

Let's query

```
SELECT DayOfWeek, count(*)
FROM file('ontime.parquet')
WHERE Year>=2000 AND
Year<=2008
GROUP BY 1 ORDER BY 1

7 rows in set. Elapsed:
0.674 sec.
```

```
SELECT DayOfWeek, count(*)
FROM file('ontime_*.parquet')
WHERE Year>=2000 AND
Year<=2008
GROUP BY 1 ORDER BY 1

7 rows in set. Elapsed: 0.139
sec.
```

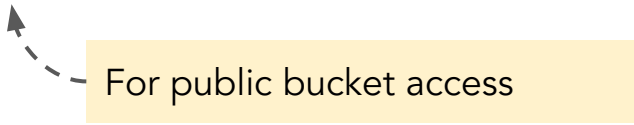
Partitioned files work faster here as well!

Reading and Writing to S3

Using S3 table function is the same as file

```
INSERT INTO FUNCTION s3('<s3
url>/ontime_{_partition_id}.parquet', <credentials>,
'Parquet')
PARTITION BY Year
SELECT * FROM ontime

SELECT count() FROM
s3('https://altinity-clickhouse-data.s3.amazonaws.com/air
line/data/ontime_parquet/*.parquet', NOSIGN)
```



Ways to pass S3 credentials to ClickHouse

```
<clickhouse>
  <s3>
    <data-lake>
      <endpoint from_env="AWS_S3_DATALAKE_URL"/>
      <use_environment_credentials>1</use_environment_credentials>
    </data-lake>
  </s3>
</clickhouse>
```

Method 1: pass values as environment variables using <s3> configuration tag

Method 2: Pass keys as strings in <s3> configuration tag

Method 3: Grant cloud IAM role to ClickHouse VM

Method 4: Use a named collection with keys

Explore Parquet on the Web

Read from GitHub

```
SELECT * FROM
url('https://github.com/plotly/datasets/raw/master/2015_flights.parquet')
LIMIT 1
SETTINGS max_http_get_redirects=1
```

```
┌─DEPARTURE_DELAY─┬─ARRIVAL_DELAY─┬─DISTANCE─┬─SCHEDULED_DEPARTURE─┐
│             -11 │           -22 │     1448 │ 0.08333333333333333 │
└─────────────────┴───────────────┴──────────┴─────────────────────┘
```

Read Public Datasets

https://platform.opentargets.org/downloads

Dataset	Description	Format(s)	Schema
Target	Core annotation for targets	JSON Parquet	
Disease/Phenotype	Core annotation for diseases and phenotypes	JSON Parquet	
Drug	Core annotation for drug molecules	JSON Parquet	
Target - Disease evidence	Integrated list of target - disease evidence from all datasources	JSON Parquet	
Associations - direct (overall score)	Overall metrics for direct target-disease associations	JSON Parquet	
Associations - indirect (overall score)	Overall metrics for indirect target-disease associations	JSON Parquet	
Associations - direct (by data source)	Data-source metrics for direct target-disease associations	JSON Parquet	
Associations - indrect (by data source)	Data-source metrics for indirect target-disease associations	JSON Parquet	
Associations - direct (by data type)	Data-type metrics for direct target-disease associations	JSON Parquet	
Associations - indirect (by data type)	Data-type metrics for indirect target-disease associations	JSON Parquet	

Rows per page: 10 1–10 of 26

Read Public Datasets – FTP Server

http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targets/

Index of /pub/databases/opentargets/platform/23.12/output/etl/parquet/targets

Name	Last modified	Size	Description
Parent Directory		-	
_SUCCESS	2023-10-31 14:28	0	
part-00000-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet	2023-10-31 14:28	462K	
part-00001-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet	2023-10-31 14:28	443K	
part-00002-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet	2023-10-31 14:28	390K	
part-00003-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet	2023-10-31 14:28	424K	
part-00004-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet	2023-10-31 14:28	477K	
part-00005-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet	2023-10-31 14:28	483K	
part-00006-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet	2023-10-31 14:28	509K	
part-00007-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet	2023-10-31 14:28	478K	
part-00008-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet	2023-10-31 14:28	493K	
part-00009-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet	2023-10-31 14:28	429K	

Read Public Datasets – Checking Data

```
DESCRIBE
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targets/part-00000-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet')
```

DESCRIBE allows to inspect remote table structure

```
id      Nullable(String)
approvedSymbol  Nullable(String)
biotype Nullable(String)
transcriptIds   Array(Nullable(String))
canonicalTranscript     Tuple(id Nullable(String), chromosome Nullable(String), start
Nullable(Int64), end Nullable(Int64), strand Nullable(String))
canonicalExons  Array(Nullable(String))
...
```

Read Public Datasets – Checking Data

```
SELECT * FROM
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/
targets/part-00000-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet') LIMIT 1
FORMAT Vertical

Row 1:
──────
id:                 ENSG00000020219
approvedSymbol:     CCT8L1P
biotype:            processed_pseudogene
transcriptIds:      ['ENST00000465400']
canonicalTranscript: ('ENST00000465400','7',152445477,152447150,'+')
canonicalExons:     ['152445477','152447150']
genomicLocation:    ('7',152445477,152447150,1)
alternativeGenes:   []
approvedName:       chaperonin containing TCP1 subunit 8 like 1, pseudogene
...
```

url() table function works similar to file and s3. It may accept HTTP headers

Read Public Datasets – CREATE TABLE for import

```
CREATE TABLE target Engine = MergeTree ORDER BY (biotype,
approvedSymbol) SETTINGS allow_nullable_key=1
AS SELECT * FROM
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform
/23.12/output/etl/parquet/targets/part-00000-70041866-e16c-4
2ce-b724-442f59bf453a-c000.snappy.parquet') NODATA
```

NODATA only reads structure for CREATE TABLE

Read Public Datasets – How to Read All Files?

```
SELECT * from
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform
/23.12/output/etl/parquet/targets/', TSVRaw)
LIMIT 100
```

TSVRaw breaks content in rows by line ends

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
 <head>
  <title>Index of /pub/databases/opentargets/platform/23.12/output/etl/parquet/targets</title>
 </head>
 <body>
<h1>Index of /pub/databases/opentargets/platform/23.12/output/etl/parquet/targets</h1>
  <table>
   <tr><th valign="top"><img src="/icons/blank.gif" alt="[ICO]"></th><th><a href="?C=N;O=D">Name</a></th><th><a href="?C=M;O=A">Last modified</a></t
   <tr><th colspan="5"><hr></th></tr>
<tr><td valign="top"><img src="/icons/back.g                                    atabases/opentargets/platform/23.12/output/etl/parquet/">P
<tr><td valign="top"><img src="/icons/unknow                                    >_SUCCESS</a></td><td align="right">2023-10-31 14:28  </t
<tr><td valign="top"><img src="/icons/unknow                                    0-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet
<tr><td valign="top"><img src="/icons/unknow                                    1-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet
<tr><td valign="top"><img src="/icons/unknown.gif" alt="[    ]"></td><td><a href="part-00002-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet
<tr><td valign="top"><img src="/icons/unknown.gif" alt="[    ]"></td><td><a href="part-00003-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet
<tr><td valign="top"><img src="/icons/unknown.gif" alt="[    ]"></td><td><a href="part-00004-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet
```

ClickHouse can read HTML!!!

Read Public Datasets – How to Read All Files?

```
SELECT DISTINCT extract(r, 'href="([^"]+\\.parquet)"') AS file_parquet
FROM
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/outp
ut/etl/parquet/targets/', TSVRaw, 'r String') WHERE file_parquet != ''
```

We can parse HTML with regular expressions

```
┌─file_parquet──────────────────────────────────────────────────────┐
│ part-00000-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │
│ part-00001-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │
│ part-00002-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │
│ part-00003-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │
│ part-00004-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │
```

Read Public Datasets – How to Read All Files?

```
WITH files AS (select distinct extract(r, 'href="([^"]+\.parquet)"') file_parquet FROM
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targe
ts/', TSVRaw, 'r String')
WHERE file_parquet != '')
SELECT 'INSERT INTO target SELECT * FROM
url(\'http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targ
ets/' || file_parquet || '\')'
FROM files FORMAT TSVRaw
```

Now we can generate INSERT statements to load all files into ClickHouse table

```
INSERT INTO target SELECT * FROM
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targe
ts/part-00000-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet')
INSERT INTO target SELECT * FROM
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targe
ts/part-00001-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet')
...
```

Read Public Datasets – How to Read All Files?

```
WITH files AS (select distinct extract(r, 'href="([^"]+\.parquet)"') file_parquet
FROM
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targets/',
TSVRaw, 'r String') WHERE file_parquet != ''),
urls AS (SELECT 'SELECT * FROM
url(\'http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targets/' ||
file_parquet || '\')' url FROM files)
SELECT 'CREATE TABLE target_all AS ' || arrayStringConcat(groupArray(url), ' UNION ALL ')
FROM urls
```

Or create a HUGE table with 200 UNION ALL.

```
CREATE TABLE target_all AS SELECT * FROM
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targets/part-00000-70041866-e16c-42
ce-b724-442f59bf453a-c000.snappy.parquet') UNION ALL SELECT * FROM
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targets/part-00001-70041866-e16c-42
ce-b724-442f59bf453a-c000.snappy.parquet') UNION ALL SELECT * FROM
url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targets/part-00002-70041866-e16c-42
ce-b724-442f59b..
```

Not very fancy. Is there a better way?

Read Public Datasets – How to Read All Files?

```
WITH files AS (select distinct extract(r, 'href="([^"]+\.parquet)"') file_parquet
FROM url('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targets/', TSVRaw,
'r String')
WHERE file_parquet != '')
SELECT 'CREATE TABLE target_all."' || file_parquet || '" Engine =
URL(\'http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targets/' ||
file_parquet || '\');'
FROM files
```

```
CREATE TABLE target_all."part-00000-70041866-e16c-42ce-b724-442f59
URL('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12
1866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet');

CREATE TABLE target_all."part-00001-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet" Engine =
URL('http://ftp.ebi.ac.uk/pub/databases/opentargets/platform/23.12/output/etl/parquet/targets/part-00001-7004
1866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet');
```

Instead, for every file we create a table with URL engine that maps to a file

Read Public Datasets – How to Read All Files? FINAL

```
SHOW TABLES FROM target_all

┌─name──────────────────────────────────────────────────────────────────────┐
│ part-00000-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │
│ part-00001-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │
│ part-00002-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │
└───────────────────────────────────────────────────────────────────────┘

SELECT count() FROM merge('target_all', '.*parquet')

┌─count()─┐
│   62733 │
└─────────┘

SELECT _file, count() FROM merge('target_all', '.*parquet') group by 1;

┌─_file──────────────────────────────────────────────────────────────────┬─count()─┐
│ part-00123-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │     318 │
│ part-00147-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │     320 │
│ part-00067-70041866-e16c-42ce-b724-442f59bf453a-c000.snappy.parquet │     311 │
```

merge() table function allows to query many tables at once, and in parallel

'_file' is a virtual column from URL engine or url() table function

Overlay Databases – the easier way to access many files

CH ver 23.7+

```
CREATE DATABASE user_files
Filesystem('/var/lib/clickhouse/user_files/')

USE user_files

SELECT count() FROM "ontime.parquet"
```

All files in the folder are accessible as tables

https://github.com/ClickHouse/ClickHouse/pull/48821

S3 and HDFS are also supported, but URL is not

Performance

Compare ontime dataset queries

- Original ontime MergeTree ORDER BY (Carrier, FlightDate) table
- file('ontime.parquet')
- file('ontime_*.parquet)
- s3('https://altinity-clickhouse-data.s3.amazonaws.com/airline/data/ontime_parquet/*.parquet')

Example queries:

Query 1: Simple grouping

```
SELECT avg(c1)
FROM (
    SELECT Year, Month, count(*) AS c1
    FROM ontime
    GROUP BY Year, Month)
```

Query 2. Grouping with filter on partition column

```
SELECT DayOfWeek, count(*) AS c
FROM ontime
WHERE Year>=2000 AND Year<=2008
GROUP BY DayOfWeek ORDER BY c DESC
```

Query 3. Grouping with filter on non-partition column

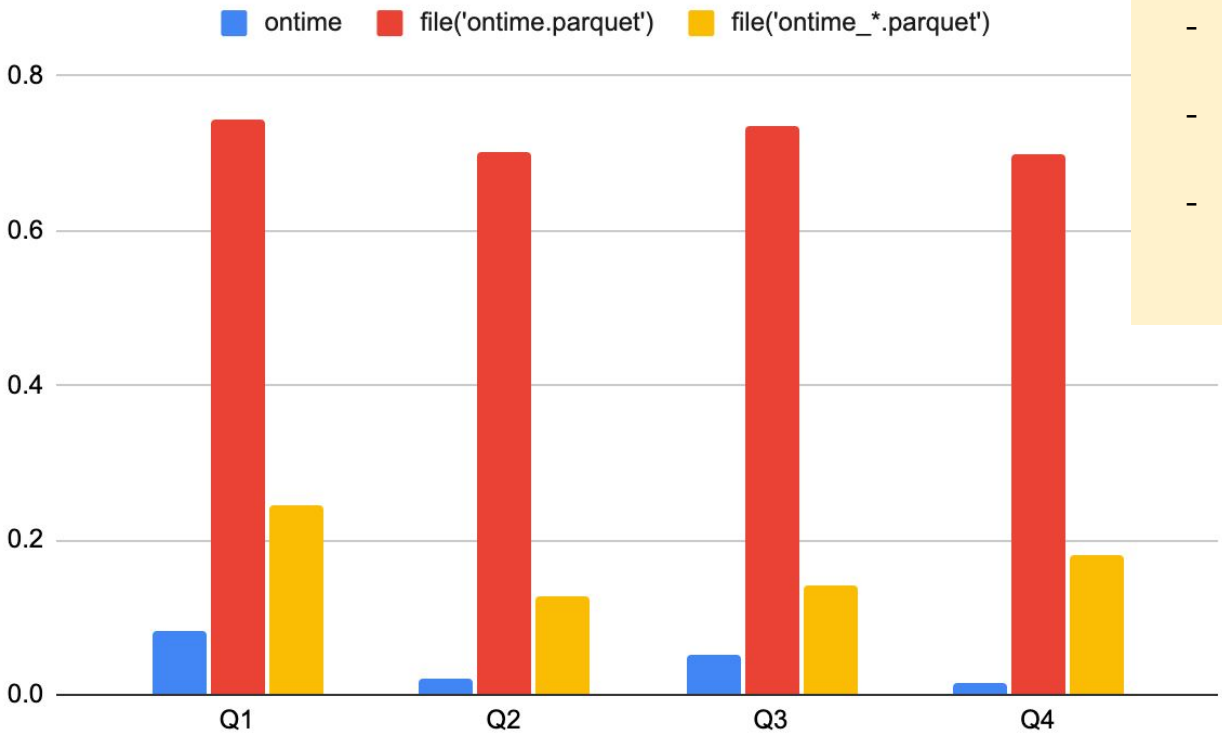
```
SELECT DayOfWeek, count(*) AS c
FROM ontime
WHERE DepDelay>10 AND Year>=2000 AND Year<=2008
GROUP BY DayOfWeek ORDER BY c DESC
```

Query 4. Filter and grouping on PK columns

```
SELECT Carrier, count(*) AS c
FROM ontime
WHERE FlightDate = '2020-01-01'
GROUP BY Carrier ORDER BY c DESC
```

Note, Parquet does not have Date data type, so FlightDate is stored as Int, and toDate(FlightDate) conversion is needed

Results



Notes:

- S3 shows very bad results, with 2-7 seconds per query
- Partitioned Parquet files benefit from mutli-threading
- Sub-second response time from Parquet is impressive!

What if we sort Parquet as a MergeTree?

```
INSERT INTO FUNCTION
file('mt_ontime_{_partition_id}.parquet')
PARTITION BY Year
SELECT * FROM ontime ORDER BY Carrier, FlightDate
```

No performance difference :(

Let's Check Compression

```
SELECT
    sum(bytes) AS c,
    sum(data_uncompressed_bytes) AS u,
    u / c AS ratio
FROM system.parts
WHERE ((database, table) = ('default',
'ontime')) AND active
```

```
┌──────────c─┬──────────u─┬─ratio─┐
│ 14453668443 │ 47759251760 │ 3.30 │
└─────────────┴─────────────┴───────┘
```

```
# ls -la ontime.parquet

-rw-r----- 1 clickhouse clickhouse
5366985658 Jan 29 18:16
ontime.parquet

# du -c -b ontime_*.parquet | grep
total

4909213868 total
```

Default Parquet compression is 3 times better than ClickHouse!

Digging Into Columns – ParquetMetadata

```
SELECT * FROM file('ontime_*.parquet', 'ParquetMetadata') format PrettyJSONEachRow
```


```
{
    "num_columns": "109",
    "num_rows": "1311826",
    "num_row_groups": "2",
    "format_version": "2.6",
    "metadata_size": "21271",
    "total_uncompressed_size": "44715050",
    "total_compressed_size": "25245952",
    "columns": [
        {
            "name": "Year",
            "path": "Year",
            "max_definition_level": "0",
            "max_repetition_level": "0",
            "physical_type": "INT32",
            "logical_type": "Int(bitWidth=16, isSigned=false)",
            "compression": "LZ4",
            "total_uncompressed_size": "116",
            "total_compressed_size": "120",
            "space_saved": "-3.448%",
            "encodings": [
                "PLAIN",
                "RLE",
                "RLE_DICTIONARY"
            ]
        },
```


Columns, types, encodings and much more

Digging Into Columns – ParquetMetadata

```
SELECT column, type, parquet_compressed, ch_compressed, round(ch_compressed / parquet_compressed,
2) as delta_pct
FROM
(
SELECT column, sum(num_rows), sum(column_uncompressed_size) parquet_uncompressed,
sum(column_compressed_size) as parquet_compressed FROM
(WITH arrayJoin(columns) as c
SELECT _file, num_rows, metadata_size, total_uncompressed_size, total_compressed_size,
tupleElement(c, 'name') as column, tupleElement(c, 'total_uncompressed_size') as
column_uncompressed_size, tupleElement(c, 'total_compressed_size') as column_compressed_size
FROM file('ontime_*.parquet', 'ParquetMetadata') t
GROUP BY column
) a
LEFT JOIN (SELECT name as column, type, data_compressed_bytes ch_compressed,
data_uncompressed_bytes ch_uncompressed FROM system.columns WHERE table='ontime') b
USING (column)
ORDER BY ch_compressed desc
```

Digging into columns – ParquetMetadata

```
SELECT column, type, parquet_compressed, ch_compressed, round(ch_compressed / parquet_compressed,
2) as delta_pct
FROM
(
SELE
sum(
(WIT
SELE
tupl
colu
FROM
GROU
) a
LEFT JOIN (SELECT name as column, type, data_compressed_bytes ch_compressed,
data_uncompressed_bytes ch_uncompressed FROM system.columns WHERE table='ontime') b
USING (column)
ORDER BY ch_compressed desc
```

column	type	parquet_compressed	ch_compressed	delta_pct
ActualElapsedTime	Int32	247332821	584065800	2.36
ArrDelay	Int32	250383972	578060989	2.31
ArrTime	Int32	279906384	577328389	2.06
DepTime	Int32	279086189	572803383	2.05
ArrTimeBlk	**String**	**72625419**	**560357061**	**7.72**
DepTimeBlk	**String**	**71994506**	**545390708**	**7.58**
DepDelay	Int32	237203517	523892164	2.21
TailNum	String	219532396	490496727	2.23

Outliers – let's check if we can compress them better in ClickHouse

Optimizing a column in 3 simple steps

1. Checking data

```
SELECT ArrTimeBlk
FROM ontime
LIMIT 5
```

```
┌─ArrTimeBlk─┐
│ 1100-1159  │
│ 1500-1559  │
│ 1600-1659  │
│ 2100-2159  │
│ 1400-1459  │
└────────────┘
```

2. Modifying a column

```
ALTER TABLE ontime MODIFY
COLUMN
ArrTimeBlk
LowCardinality(String);
```

3. Confirming results

```
SELECT
sum(column_data_compressed_bytes)
FROM system.parts_columns
WHERE (database, table, column) =
('default','ontime','ArrTimeBlk')
AND active
```

```
┌─sum(column_data_compressed_bytes)─┐
│                          49610756 │
└───────────────────────────────────┘
```

560 357 061 -> 49 610 756 => Improved 11 times!

Let's convert all String columns!

```
# clickhouse-client --query="select 'ALTER TABLE default.ontime MODIFY COLUMN ' || name
|| ' LowCardinality(String);' from system.columns where table = 'ontime' and type =
'String';" | xargs -I {} clickhouse-client --echo --query="{}"

ALTER TABLE default.ontime MODIFY COLUMN TailNum LowCardinality(String);
ALTER TABLE default.ontime MODIFY COLUMN FlightNum LowCardinality(String);
...
```

14 453 668 443 -> 11 784 090 971

Good improvement but still much worse than Parquet

And apply ZSTD(1) compression to integer columns

```
# clickhouse-client --query="select 'ALTER TABLE default.ontime MODIFY COLUMN ' || name
|| ' CODEC(ZSTD(1));' from system.columns where table = 'ontime' and type like 'Int%';"
| xargs -I {} clickhouse-client --echo --query="{}"

ALTER TABLE default.ontime MODIFY COLUMN AirlineID CODEC(ZSTD(1));
ALTER TABLE default.ontime MODIFY COLUMN OriginAirportID CODEC(ZSTD(1));

...

OPTIMIZE TABLE default.ontime FINAL
```

This is required to re-compress data

```
11 784 090 971 -> 5 892 513 899
```

x2 improvement, but still behind Parquet

Parquet in a Cluster – fileCluster(), urlCluster(), s3Cluster()

- Easy way to distribute reading Parquet files across multiple ClickHouse nodes
- Scales linearly
  - https://altinity.com/blog/tips-for-high-performance-clickhouse-clusters-with-s3-object-storage
- fileCluster() requires a shared file system
- BUT: No query rewrite, performance of aggregation queries is slower compared to Distributed tables

Write performance

```
:) INSERT INTO function file('perf3.parquet') SELECT * FROM ontime;

Elapsed: 954.394 sec. Processed 201.58 million rows, 41.47 GB
(211.21 thousand rows/s., 43.45 MB/s.)
```

This looks slow, doesn't it?

Write performance

```
:) INSERT INTO function file('perf3.parquet') SELECT * FROM ontime;

Elapsed: 954.394 sec. Processed 201.58 million rows, 41.47 GB
(211.21 thousand rows/s., 43.45 MB/s.)

:) INSERT INTO function file('perf2.parquet') SELECT * FROM ontime
settings output_format_parquet_use_custom_encoder=1;

Elapsed: 25.963 sec. Processed 201.58 million rows, 41.47 GB (7.76
million rows/s., 1.60 GB/s.)
```

x40 times faster!!!

Write performance

```
:) INSERT INTO function file('perf2.parquet') SELECT * FROM ontime
settings output_format_parquet_use_custom_encoder=1;

Elapsed: 25.963 sec. Processed 201.58 million rows, 41.47 GB (7.76
million rows/s., 1.60 GB/s.)

:) INSERT INTO ontime2 SELECT * FROM ontime;

Elapsed: 37.800 sec. Processed 201.58 million rows, 41.48 GB (5.33
million rows/s., 1.10 GB/s.)
```

Even 50% faster than MergeTree!

Future Developments

ClickHouse Roadmap

- Support for Iceberg Data Catalog (2024 Roadmap)
- Support for Hive-style partitioning (2024 Roadmap)
- Enable multi-threaded writes (experimental feature) https://github.com/ClickHouse/ClickHouse/pull/53168
- Improvements for complex data types support:
  - https://github.com/ClickHouse/ClickHouse/pull/56156
  - https://github.com/ClickHouse/ClickHouse/pull/53443

RFC: Parquet part format for MergeTree

- Similar to compact data parts but with a Parquet as a file format:

```
./2024_1_1_0/primary.idx
./2024_1_1_0/columns.txt
./2024_1_1_0/data.parquet
```

github.com/ClickHouse/ClickHouse/issues/57799

Want to assist with or sponsor Parquet improvements?
Contact Altinity at https://altinity.com

Wrap-Up

Parquet and ClickHouse

Good

- Easy to work with files, S3, and hdfs (presumably)
- Parquet default compression is outstanding!
- Parquet performance from local files is very good
- Easy to explore remote Parquet datasets

Not so good

- Hard to work with web server file collections
- Does not use sort order and indices

Best practices for Parquet with ClickHouse

- Use latest ClickHouse version for latest features
  - But stick with LTS or Altinity.Builds for production
- Use multiple Parquet files better query performance
- Use cluster functions in ClickHouse clusters

References

- ClickHouse Inc documentation
- ClickHouse Roadmap 2024:
  - https://github.com/ClickHouse/ClickHouse/issues/58392
- A Deep Dive into Apache Parquet with ClickHouse:
  - https://clickhouse.com/blog/apache-parquet-clickhouse-local-querying-writing
  - https://clickhouse.com/blog/apache-parquet-clickhouse-local-querying-writing-internals-row-groups
- What's Up with Parquet in ClickHouse:
  - https://altinity.com/blog/whats-up-with-parquet-performance-in-clickhouse

Thank you!

Visit us: https://altinity.com
Altinity Slack (Invite Link)

Altinity.Cloud
Altinity Stable Builds for ClickHouse
Altinity Kubernetes Operator for ClickHouse