Step Description

1. Build distributed clickhouse database cluster.
2. Create a local database and a local table on one onde.
3. Create distributed database and table.
4. Download and load example data to local table and distributed table.
5. Use insert into select query testing insert performance.
6. Query local table and distributed table test query performancel.
7. Analyze and document performance differences.

1. build a distributed clickhouse database cluster

We build a distributed clickhouse database server named jk_testcluster. Including three sharding nodes.

cluster	sharding_num	shard_weight	replica_num
jk_testcluster	1	1	1
jk_testcluster	2	1	1
jk_testcluster	3	1	1

2. Create local database and table on one node.

```
CREATE DATABASE local_db

USE local_db

CREATE TABLE ontime
(
    `Year` UInt16,
    `Quarter` UInt8,
    `Month` UInt8,
    `DayofMonth` UInt8,
    `DayOfWeek` UInt8,
    `FlightDate` Date,
    `UniqueCarrier` FixedString(7),
    `AirlineID` Int32,
    `Carrier` FixedString(2),
    `TailNum` String,
    `FlightNum` String,
    `OriginAirportID` Int32,
```

`OriginAirportSeqID` Int32,
`OriginCityMarketID` Int32,
`Origin` FixedString(5),
`OriginCityName` String,
`OriginState` FixedString(2),
`OriginStateFips` String,
`OriginStateName` String,
`OriginWac` Int32,
`DestAirportID` Int32,
`DestAirportSeqID` Int32,
`DestCityMarketID` Int32,
`Dest` FixedString(5),
`DestCityName` String,
`DestState` FixedString(2),
`DestStateFips` String,
`DestStateName` String,
`DestWac` Int32,
`CRSDepTime` Int32,
`DepTime` Int32,
`DepDelay` Int32,
`DepDelayMinutes` Int32,
`DepDel15` Int32,
`DepartureDelayGroups` String,
`DepTimeBlk` String,
`TaxiOut` Int32,
`WheelsOff` Int32,
`WheelsOn` Int32,
`TaxiIn` Int32,
`CRSArrTime` Int32,
`ArrTime` Int32,
`ArrDelay` Int32,
`ArrDelayMinutes` Int32,
`ArrDel15` Int32,
`ArrivalDelayGroups` Int32,
`ArrTimeBlk` String,
`Cancelled` UInt8,
`CancellationCode` FixedString(1),
`Diverted` UInt8,
`CRSElapsedTime` Int32,
`ActualElapsedTime` Int32,
`AirTime` Int32,
`Flights` Int32,
`Distance` Int32,
`DistanceGroup` UInt8,

`CarrierDelay` Int32,
`WeatherDelay` Int32,
`NASDelay` Int32,
`SecurityDelay` Int32,
`LateAircraftDelay` Int32,
`FirstDepTime` String,
`TotalAddGTime` String,
`LongestAddGTime` String,
`DivAirportLandings` String,
`DivReachedDest` String,
`DivActualElapsedTime` String,
`DivArrDelay` String,
`DivDistance` String,
`Div1Airport` String,
`Div1AirportID` Int32,
`Div1AirportSeqID` Int32,
`Div1WheelsOn` String,
`Div1TotalGTime` String,
`Div1LongestGTime` String,
`Div1WheelsOff` String,
`Div1TailNum` String,
`Div2Airport` String,
`Div2AirportID` Int32,
`Div2AirportSeqID` Int32,
`Div2WheelsOn` String,
`Div2TotalGTime` String,
`Div2LongestGTime` String,
`Div2WheelsOff` String,
`Div2TailNum` String,
`Div3Airport` String,
`Div3AirportID` Int32,
`Div3AirportSeqID` Int32,
`Div3WheelsOn` String,
`Div3TotalGTime` String,
`Div3LongestGTime` String,
`Div3WheelsOff` String,
`Div3TailNum` String,
`Div4Airport` String,
`Div4AirportID` Int32,
`Div4AirportSeqID` Int32,
`Div4WheelsOn` String,
`Div4TotalGTime` String,
`Div4LongestGTime` String,
`Div4WheelsOff` String,

```
    `Div4TailNum` String,
    `Div5Airport` String,
    `Div5AirportID` Int32,
    `Div5AirportSeqID` Int32,
    `Div5WheelsOn` String,
    `Div5TotalGTime` String,
    `Div5LongestGTime` String,
    `Div5WheelsOff` String,
    `Div5TailNum` String
)
ENGINE = MergeTree
PARTITION BY Year
ORDER BY (Carrier, FlightDate)
SETTINGS index_granularity = 8192
```

3. Create distributed database and table

Create cluster database and table on all node, and create a distributed table as proxy table.

```
CREATE DATABASE sharding_db ON CLUSTER jk_testcluster

USE sharding_db

CREATE TABLE ontime ON CLUSTER jk_testcluster
(
    `Year` UInt16,
    `Quarter` UInt8,
    `Month` UInt8,
    `DayofMonth` UInt8,
    `DayOfWeek` UInt8,
    `FlightDate` Date,
    `UniqueCarrier` FixedString(7),
    `AirlineID` Int32,
    `Carrier` FixedString(2),
    `TailNum` String,
    `FlightNum` String,
    `OriginAirportID` Int32,
    `OriginAirportSeqID` Int32,
    `OriginCityMarketID` Int32,
    `Origin` FixedString(5),
    `OriginCityName` String,
```

`OriginState` FixedString(2),
`OriginStateFips` String,
`OriginStateName` String,
`OriginWac` Int32,
`DestAirportID` Int32,
`DestAirportSeqID` Int32,
`DestCityMarketID` Int32,
`Dest` FixedString(5),
`DestCityName` String,
`DestState` FixedString(2),
`DestStateFips` String,
`DestStateName` String,
`DestWac` Int32,
`CRSDepTime` Int32,
`DepTime` Int32,
`DepDelay` Int32,
`DepDelayMinutes` Int32,
`DepDel15` Int32,
`DepartureDelayGroups` String,
`DepTimeBlk` String,
`TaxiOut` Int32,
`WheelsOff` Int32,
`WheelsOn` Int32,
`TaxiIn` Int32,
`CRSArrTime` Int32,
`ArrTime` Int32,
`ArrDelay` Int32,
`ArrDelayMinutes` Int32,
`ArrDel15` Int32,
`ArrivalDelayGroups` Int32,
`ArrTimeBlk` String,
`Cancelled` UInt8,
`CancellationCode` FixedString(1),
`Diverted` UInt8,
`CRSElapsedTime` Int32,
`ActualElapsedTime` Int32,
`AirTime` Int32,
`Flights` Int32,
`Distance` Int32,
`DistanceGroup` UInt8,
`CarrierDelay` Int32,
`WeatherDelay` Int32,
`NASDelay` Int32,
`SecurityDelay` Int32,

`LateAircraftDelay` Int32,
`FirstDepTime` String,
`TotalAddGTime` String,
`LongestAddGTime` String,
`DivAirportLandings` String,
`DivReachedDest` String,
`DivActualElapsedTime` String,
`DivArrDelay` String,
`DivDistance` String,
`Div1Airport` String,
`Div1AirportID` Int32,
`Div1AirportSeqID` Int32,
`Div1WheelsOn` String,
`Div1TotalGTime` String,
`Div1LongestGTime` String,
`Div1WheelsOff` String,
`Div1TailNum` String,
`Div2Airport` String,
`Div2AirportID` Int32,
`Div2AirportSeqID` Int32,
`Div2WheelsOn` String,
`Div2TotalGTime` String,
`Div2LongestGTime` String,
`Div2WheelsOff` String,
`Div2TailNum` String,
`Div3Airport` String,
`Div3AirportID` Int32,
`Div3AirportSeqID` Int32,
`Div3WheelsOn` String,
`Div3TotalGTime` String,
`Div3LongestGTime` String,
`Div3WheelsOff` String,
`Div3TailNum` String,
`Div4Airport` String,
`Div4AirportID` Int32,
`Div4AirportSeqID` Int32,
`Div4WheelsOn` String,
`Div4TotalGTime` String,
`Div4LongestGTime` String,
`Div4WheelsOff` String,
`Div4TailNum` String,
`Div5Airport` String,
`Div5AirportID` Int32,
`Div5AirportSeqID` Int32,

```
    `Div5WheelsOn` String,
    `Div5TotalGTime` String,
    `Div5LongestGTime` String,
    `Div5WheelsOff` String,
    `Div5TailNum` String
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-{shard}/ontime', '{replica}')
PARTITION BY Year
ORDER BY (Carrier, FlightDate)
SETTINGS index_granularity = 8192
```

```
CREATE TABLE ontime_proxy AS ontime ENGINE = Distributed(jk_testcluster,sharding_db, ontime, rand());
```

4. Download and load example data to local table and distributed table

Download example test data use this script.

```
for s in `seq 1987 2018`
do
for m in `seq 1 12`
do
wget https://transtats.bts.gov/PREZIP/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_${s}_${m}.zip
done
done
```

Load test data into local table

```
#for i in *.zip; do echo $i; unzip -cq $i '*.csv' | sed 's/\.00//g' | clickhouse-client --host=127.0.0.1 --query="INSERT
INTO local_db.ontime FORMAT CSVWithNames"; done
```

```
SELECT count(*)
FROM local_db.ontime
```

```
┌──────count()─┐
│  163896350 │
└──────────────┘
```

1 rows in set. Elapsed: 0.002 sec.

Local test data into distributed table.

#for i in *.zip; do echo $i; unzip -cq $i '*.csv' | sed 's/\.00//g' | clickhouse-client --host=127.0.0.1 --query="INSERT
INTO sharding_db.ontime_proxy FORMAT CSVWithNames"; done

SELECT count(*)
FROM sharding_db.ontime_proxy

```
┌────────count()─┐
│  163896350  │
└──────────────────────┘
```

1 rows in set. Elapsed: 0.005 sec.

5. Use insert into select query testing insert performance.

Local table:

INSERT INTO local_db.ontime SELECT *
FROM local_db.ontime

0 rows in set. Elapsed: 987.407 sec. Processed 327.79 million rows, 238.09 GB (331.97 thousand rows/s., 241.12 MB/s.)

Distributed table:

INSERT INTO sharding_db.ontime_proxy SELECT *
FROM local_db.ontime

0 rows in set. Elapsed: 1198.381 sec. Processed 327.79 million rows, 238.09 GB (273.53 thousand rows/s., 198.67 MB/s.)

6. Query local table and distributed table test query performancel.

SQL1-Query local table

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

┌──────────avg(c1)─┐
│ 3449508.656716418 │
└──────────────────┘
```

1 rows in set. Elapsed: 0.860 sec. Processed 1.16 billion rows, 3.47 GB (1.34 billion rows/s., 4.03 GB/s.)

SQL1-Query distributed table

```
SELECT avg(c1)
FROM
(
    SELECT
        Year,
        Month,
        count(*) AS c1
    FROM sharding_db.ontime_proxy
    GROUP BY
        Year,
        Month
)

┌──────────avg(c1)─┐
│ 3424699.850746269 │
└──────────────────┘
```

1 rows in set. Elapsed: 2.542 sec. Processed 1.15 billion rows, 3.44 GB (451.34 million rows/s., 1.35 GB/s.)

SQL2-Query local table

```
SELECT
    DayOfWeek,
    count(*) AS c
FROM local_db.ontime
WHERE (Year >= 1980) AND (Year <= 2000)
GROUP BY DayOfWeek
ORDER BY c DESC
```

```
┌─DayOfWeek─┬────────c─┐
│         2 │ 72359764 │
│         3 │ 72350933 │
│         1 │ 72237834 │
│         4 │ 72020086 │
│         5 │ 71928627 │
│         7 │ 68348074 │
│         6 │ 64795230 │
└───────────┴──────────┘
```

7 rows in set. Elapsed: 0.160 sec. Processed 494.04 million rows, 1.48 GB (3.10 billion rows/s., 9.29 GB/s.)

SQL1-Query distributed table

```
SELECT
    DayOfWeek,
    count(*) AS c
FROM sharding_db.ontime_proxy
WHERE (Year >= 1980) AND (Year <= 2000)
GROUP BY DayOfWeek
ORDER BY c DESC
```

```
┌─DayOfWeek─┬────────c─┐
│         2 │ 71755803 │
│         3 │ 71738429 │
│         1 │ 71617896 │
│         4 │ 71404879 │
│         5 │ 71319010 │
│         7 │ 67687641 │
│         6 │ 64004255 │
└───────────┴──────────┘
```

7 rows in set. Elapsed: 0.477 sec. Processed 489.53 million rows, 1.47 GB (1.03 billion rows/s., 3.08 GB/s.)

SQL3-Query local table

```
SELECT
    min(Year),
    max(Year),
    Carrier,
    count(*) AS cnt,
    sum(ArrDelayMinutes > 30) AS flights_delayed,
    round(sum(ArrDelayMinutes > 30) / count(*), 2) AS rate
FROM local_db.ontime
WHERE (DayOfWeek NOT IN (6, 7)) AND (OriginState NOT IN ('AK', 'HI', 'PR', 'VI')) AND (DestState NOT IN ('AK', 'HI', 'PR', 'VI')) AND (FlightDate < '2010-01-01')
GROUP BY Carrier
HAVING (cnt > 100000) AND (max(Year) > 1990)
ORDER BY rate DESC
LIMIT 1000
```

min(Year)	max(Year)	Carrier	cnt	flights_delayed	rate
2003	2009	EV	9245879	1470287	0.16
2003	2009	FL	6606861	1005069	0.15
2003	2009	B6	4296927	634408	0.15
2006	2009	YV	3583359	540107	0.15
2006	2009	XE	4492979	674072	0.15
2003	2005	DH	4008448	558664	0.14
2001	2009	MQ	21268743	2854270	0.13
2003	2006	RU	7005914	889798	0.13
2004	2009	OH	7141593	916197	0.13
2003	2006	TZ	936773	113361	0.12
1987	2009	UA	61674786	7652254	0.12
1987	2009	AA	81941399	8899129	0.11
1987	2009	CO	46408618	5110734	0.11
1987	2001	TW	18514194	1962164	0.11
1988	2009	US	62420781	6019908	0.1
1987	2009	AS	11341683	1095019	0.1
1987	2009	DL	89527333	8677613	0.1
2007	2009	9E	3063399	311598	0.1
1987	2009	WN	69927025	6060131	0.09
1987	1991	PA	1559235	141927	0.09
1987	2005	HP	19475953	1745861	0.09
2005	2009	F9	1692296	157018	0.09
1987	2009	NW	52797934	4965057	0.09
2003	2009	OO	15275994	1411647	0.09
1991	1991	ML	414936	28864	0.07

```
──────────────────────────────────────┴──────────────┘
```

25 rows in set. Elapsed: 2.177 sec. Processed 870.99 million rows, 13.01 GB (400.17 million rows/s., 5.98 GB/s.)

SQL2-Query distributed table

```
SELECT
    min(Year),
    max(Year),
    Carrier,
    count(*) AS cnt,
    sum(ArrDelayMinutes > 30) AS flights_delayed,
    round(sum(ArrDelayMinutes > 30) / count(*), 2) AS rate
FROM sharding_db.ontime_proxy
WHERE (DayOfWeek NOT IN (6, 7)) AND (OriginState NOT IN ('AK', 'HI', 'PR', 'VI')) AND (DestState NOT IN ('AK', 'HI', 'PR', 'VI')) AND (FlightDate < '2010-01-01')
GROUP BY Carrier
HAVING (cnt > 100000) AND (max(Year) > 1990)
ORDER BY rate DESC
LIMIT 1000
```

```
┌─min(Year)─┬─max(Year)─┬─Carrier─┬──────cnt─┬─flights_delayed─┬─rate─┐
│      2003 │      2009 │ EV      │  8718493 │         1412404 │ 0.16 │
│      2003 │      2009 │ FL      │  6300196 │          958279 │ 0.15 │
│      2003 │      2009 │ B6      │  3869460 │          572166 │ 0.15 │
│      2006 │      2009 │ YV      │  3922569 │          591626 │ 0.15 │
│      2006 │      2009 │ XE      │  5038012 │          757575 │ 0.15 │
│      2001 │      2009 │ MQ      │ 20070323 │         2726885 │ 0.14 │
│      2003 │      2005 │ DH      │  3507392 │          488831 │ 0.14 │
│      2003 │      2006 │ RU      │  7050736 │          887131 │ 0.13 │
│      2004 │      2009 │ OH      │  7270935 │          939841 │ 0.13 │
│      2003 │      2006 │ TZ      │   957145 │          115472 │ 0.12 │
│      1987 │      2009 │ UA      │ 65674035 │         8105307 │ 0.12 │
│      1987 │      2001 │ TW      │ 18972485 │         1999949 │ 0.11 │
│      1987 │      2009 │ AA      │ 72111704 │         7844823 │ 0.11 │
│      1987 │      2009 │ CO      │ 41330625 │         4565015 │ 0.11 │
│      1988 │      2009 │ US      │ 69600678 │         6654151 │  0.1 │
│      1987 │      2009 │ AS      │ 10097584 │          977522 │  0.1 │
│      1987 │      2009 │ DL      │ 81731468 │         7935431 │  0.1 │
│      2007 │      2009 │ 9E      │  2711093 │          275891 │  0.1 │
│      2003 │      2009 │ OO      │ 15611785 │         1453137 │ 0.09 │
│      1987 │      2005 │ HP      │ 18483465 │         1665272 │ 0.09 │
│      2005 │      2009 │ F9      │  1672783 │          157850 │ 0.09 │
```

```
|      1987 |      2009 | NW |  51915045 |  4885097 | 0.09 |
|      1987 |      1991 | PA |   1532566 |   143479 | 0.09 |
|      1987 |      2009 | WN |  83320930 |  7199808 | 0.09 |
|      1991 |      1991 | ML |    363069 |    25256 | 0.07 |
└─────────────┴─────────────┴────┴──────────┴──────────┴──────┘
```

25 rows in set. Elapsed: 7.665 sec. Processed 869.77 million rows, 12.94 GB (113.48 million rows/s., 1.69 GB/s.)

8. Analyze and document performance differences.