BACH KHOA UNIVERSITY

FACULTY of COMPUTER SCIENCE AND ENGINEERING

DATABASE MANAGEMENT SYSTEMS

Query processing

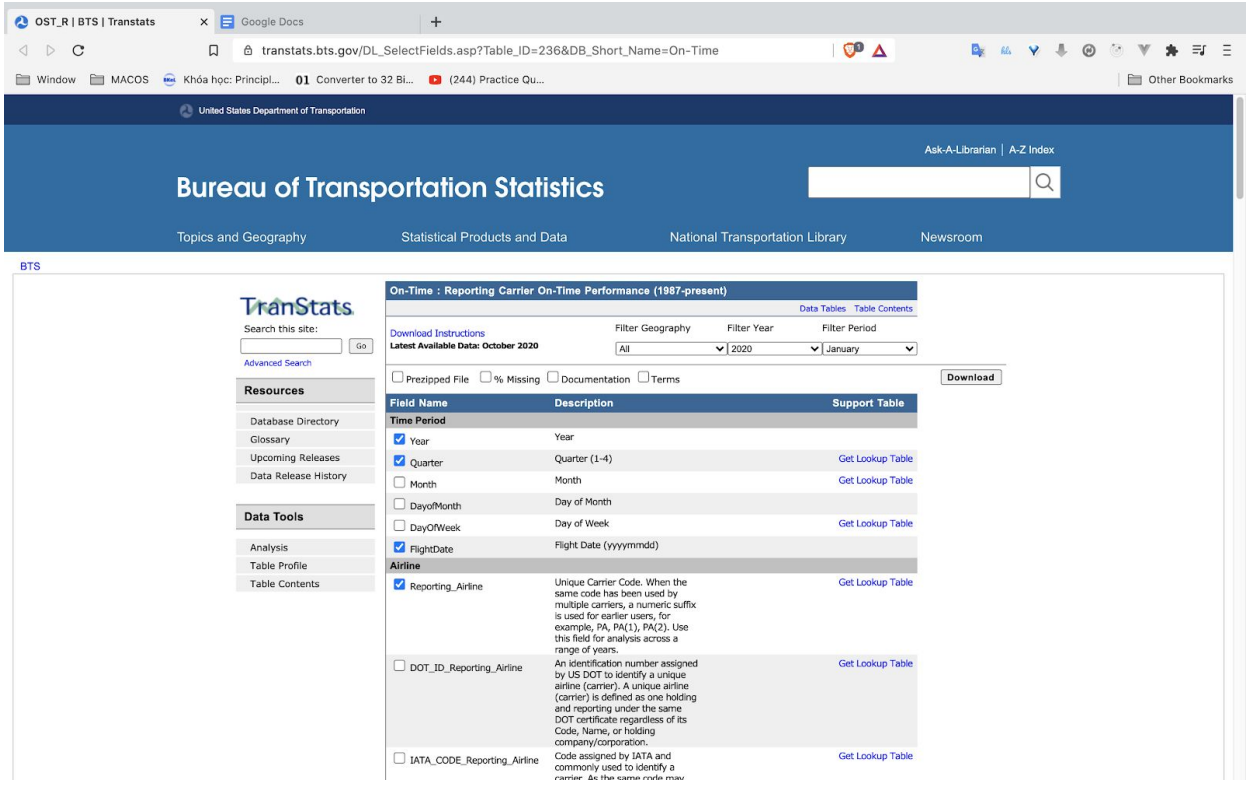
Demo

Students:	*Student ID:*
Vo Minh Huy	1552150
Le Ba Thanh	1552340

Contents

1 parallel Algorithms for Query Processing Approach

Source of the Database:



To demonstrate the *Parallel Algorithm* in Query Processing i had reference to this website to retrieve enormous dataset of air plane flight.

So I load a total of 24 columns with ~152MB size per each row in sum up, and the total dataset was 56GB.initial the table by SQL followed below.

```
CREATE TABLE `ontime` (
`YearD` year(4) NOT NULL,
`Quarter` tinyint(4) DEFAULT NULL,
`MonthD` tinyint(4) DEFAULT NULL,
`DayofMonth` tinyint(4) DEFAULT NULL,
`DayOfWeek` tinyint(4) DEFAULT NULL,
`FlightDate` date DEFAULT NULL,
`UniqueCarrier` char(7) DEFAULT NULL,
`AirlineID` int(11) DEFAULT NULL,
`Carrier` char(2) DEFAULT NULL,
`TailNum` varchar(50) DEFAULT NULL,
`FlightNum` varchar(10) DEFAULT NULL,
`OriginAirportID` int(11) DEFAULT NULL,
`OriginAirportSeqID` int(11) DEFAULT NULL,
`OriginCityMarketID` int(11) DEFAULT NULL,
`Origin` char(5) DEFAULT NULL,
`OriginCityName` varchar(100) DEFAULT NULL,
`OriginState` char(2) DEFAULT NULL,
`OriginStateFips` varchar(10) DEFAULT NULL,
`OriginStateName` varchar(100) DEFAULT NULL,
`OriginWac` int(11) DEFAULT NULL,
`DestAirportID` int(11) DEFAULT NULL,
`DestAirportSeqID` int(11) DEFAULT NULL,
`DestCityMarketID` int(11) DEFAULT NULL,
`Dest` char(5) DEFAULT NULL,
-- ... (removed number of fields)
`id` int(11) NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`),
KEY `YearD` (`YearD`),
KEY `Carrier` (`Carrier`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

thanhle@Erics-Mac ~
```

- Simple Query:

So at the beginning of the demo stage i run a simple query to retrieve all the flights per year in the US. However each row is 150MB so it load on the Query(cached):

```
mysql> select yeard, count(*) from ontime group by yeard;
+-------+----------+
| yeard | count(*) |
+-------+----------+
| 1988  | 5202096  |
| 1989  | 5041200  |
| 1990  | 5270893  |
| 1991  | 5076925  |
| 1992  | 5092157  |
| 1993  | 5070501  |
| 1994  | 5180048  |
| 1995  | 5327435  |
| 1996  | 5351983  |
| 1997  | 5411843  |
| 1998  | 5384721  |
| 1999  | 5527884  |
| 2000  | 5683047  |
| 2001  | 5967780  |
| 2002  | 5271359  |
| 2003  | 6488540  |
| 2004  | 7129270  |
| 2005  | 7140596  |
| 2006  | 7141922  |
| 2007  | 7455458  |
| 2008  | 7009726  |
| 2009  | 6450285  |
| 2010  | 6450117  |
| 2011  | 6085281  |
| 2012  | 6096762  |
| 2013  | 5349447  |
+-------+----------+
26 rows in set (54.10 sec)
```

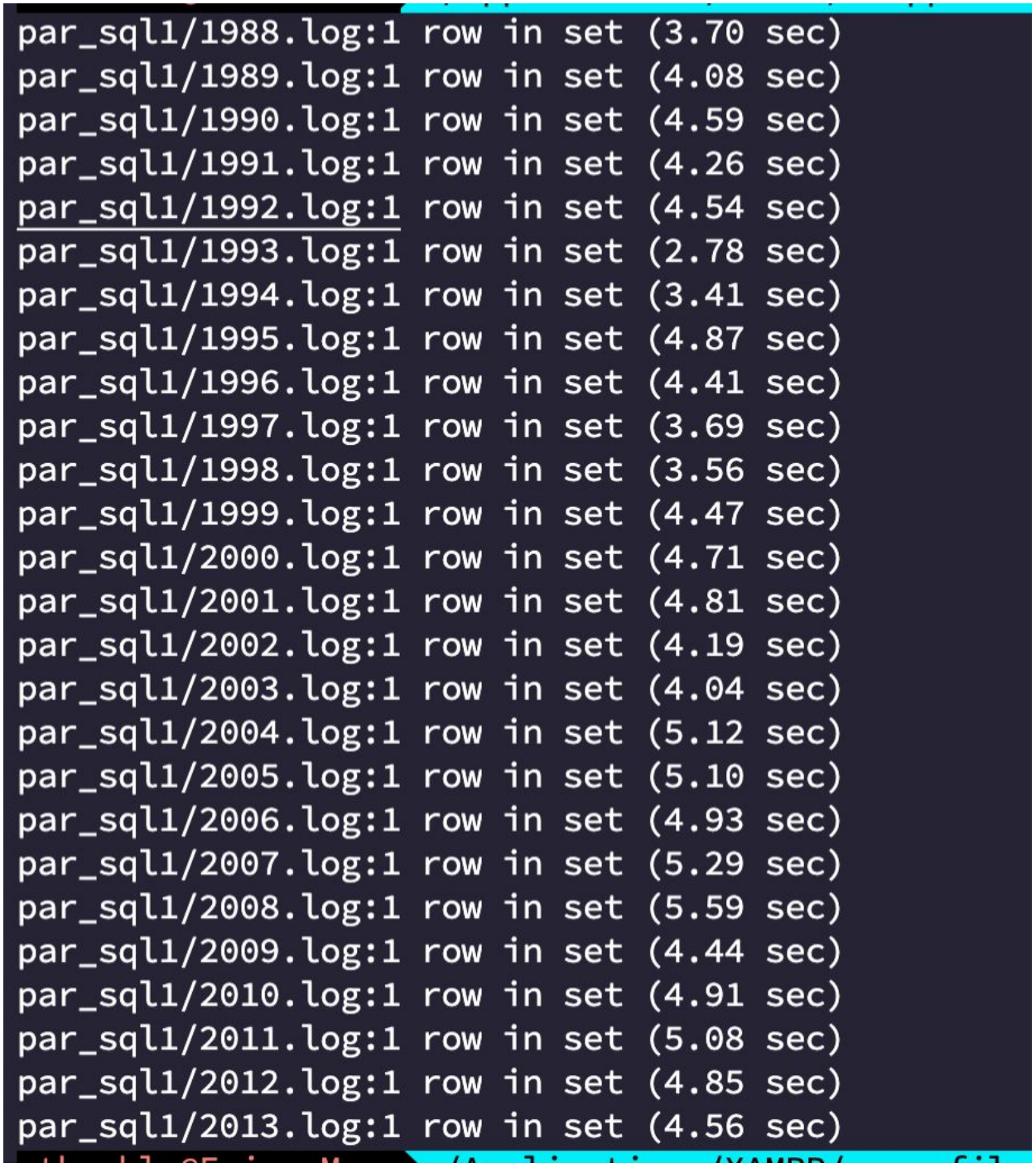
Totally it takes **54.10 secs** to execute the query above to scan flight data from 1988 to 2013. To explain more in detail what actually the sql has done i used the **EXPLAIN** for more detail.

```
mysql> explain select yeard, count(*) from ontime group by yeardG
*************************** 1. row ***************************
id: 1
select_type: SIMPLE
table: ontime
type: index
possible_keys: YearD,comb1
key: YearD
key_len: 1
ref: NULL
rows: 148046200
Extra: Using index
1 row in set (0.00 sec)
```

Now I will use the Parallel Algorithm to process the query retrieval process more faster

```
date
for y in {1988..2013}
do
  sql="select yeard, count(*) from ontime where yeard=$y"
  mysql -vvv ontime -e "$sql" &>par_sql1/$y.log &
done
wait
date
```

And the results it takes 10 times (~ 5.12 seconds) faster compared to the original simple query.



- Complex Query:

The Basic idea of this complex query is to look up the number of flights that are delayed in the domestic US only from Monday to Friday, and the conditioning is more strict make the query more complex .

```
mysql> select
   min(yeard), max(yeard), Carrier, count(*) as cnt,
   sum(ArrDelayMinutes>30) as flights_delayed,
   round(sum(ArrDelayMinutes>30)/count(*),2) as rate
FROM 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(yeard) > 1990
ORDER by rate DESC
```

And here is the result of the above query:

```
+------------+------------+---------+----------+-----------------+------+
| min(yeard) | max(yeard) | Carrier | cnt      | flights_delayed | rate |
+------------+------------+---------+----------+-----------------+------+
|       2003 |       2009 | EV      |  1454777 |          237698 | 0.16 |
|       2006 |       2009 | XE      |  1016010 |          152431 | 0.15 |
|       2006 |       2009 | YV      |   740608 |          110389 | 0.15 |
|       2003 |       2009 | B6      |   683874 |          103677 | 0.15 |
|       2003 |       2009 | FL      |  1082489 |          158748 | 0.15 |
|       2003 |       2005 | DH      |   501056 |           69833 | 0.14 |
|       2001 |       2009 | MQ      |  3238137 |          448037 | 0.14 |
|       2003 |       2006 | RU      |  1007248 |          126733 | 0.13 |
|       2004 |       2009 | OH      |  1195868 |          160071 | 0.13 |
|       2003 |       2006 | TZ      |   136735 |           16496 | 0.12 |
|       1988 |       2009 | UA      |  9593284 |         1197053 | 0.12 |
|       1988 |       2009 | AA      | 10600509 |         1185343 | 0.11 |
|       1988 |       2001 | TW      |  2659963 |          280741 | 0.11 |
|       1988 |       2009 | CO      |  6029149 |          673863 | 0.11 |
|       2007 |       2009 | 9E      |   577244 |           59440 | 0.10 |
|       1988 |       2009 | DL      | 11869471 |         1156267 | 0.10 |
|       1988 |       2009 | NW      |  7601727 |          725460 | 0.10 |
|       1988 |       2009 | AS      |  1506003 |          146920 | 0.10 |
|       2003 |       2009 | OO      |  2654259 |          257069 | 0.10 |
|       1988 |       2009 | US      | 10276941 |          991016 | 0.10 |
|       1988 |       1991 | PA      |   206841 |           19465 | 0.09 |
|       1988 |       2005 | HP      |  2607603 |          235675 | 0.09 |
|       1988 |       2009 | WN      | 12722174 |         1107840 | 0.09 |
|       2005 |       2009 | F9      |   307569 |           28679 | 0.09 |
+------------+------------+---------+----------+-----------------+------+
24 rows in set (15 min 56.40 sec)
```

Explain SQL:

```
*************************** 1. row ***************************
id: 1
select_type: SIMPLE
table: ontime
type: index
possible_keys: comb1
key: comb1
key_len: 9
ref: NULL
rows: 148046200
Extra: Using where; Using temporary; Using filesort
```

It took a lot of time to finish the simple query (~ 15 min 56.40 sec).

Now, I will apply the Parallel Processing approach by splitting the process based on the brand of the airlines (In here we have totally 31 distinguished airplanes type).

```
mysql>date
for c in '9E' 'AA' 'AL' 'AQ' 'AS' 'B6' 'CO' 'DH' 'DL' 'EA' 'EV' 'F9' 'FL' 'HA' 'HP' 'ML' 'MQ' 'NW' 'OH' 'OO' 'PA'
'PI' 'PS' 'RU' 'TW' 'TZ' 'UA' 'US' 'WN' 'XE' 'YV'
do
   sql=" select min(yeard), max(yeard), Carrier, count(*) as cnt, sum(ArrDelayMinutes>30) as flights_delayed, roun
d(sum(ArrDelayMinutes>30)/count(*),2) as rate FROM 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' and carrier = '$c
'"
   mysql -uroot -vvv ontime -e "$sql" &>par_sql_complex/$c.log &
done
wait
date
```

The process is significantly improved by the Parallel Algorithm base, below is the result I got.

```
par_sql_complex/9E.log:1 row in set (44.47 sec)
par_sql_complex/AA.log:1 row in set (5 min 41.13 sec)
par_sql_complex/AL.log:1 row in set (15.81 sec)
par_sql_complex/AQ.log:1 row in set (14.52 sec)
par_sql_complex/AS.log:1 row in set (2 min 43.01 sec)
par_sql_complex/B6.log:1 row in set (1 min 26.06 sec)
par_sql_complex/CO.log:1 row in set (3 min 58.07 sec)
par_sql_complex/DH.log:1 row in set (31.30 sec)
par_sql_complex/DL.log:1 row in set (5 min 47.07 sec)
par_sql_complex/EA.log:1 row in set (28.58 sec)
par_sql_complex/EV.log:1 row in set (2 min 6.87 sec)
par_sql_complex/F9.log:1 row in set (46.18 sec)
par_sql_complex/FL.log:1 row in set (1 min 30.83 sec)
par_sql_complex/HA.log:1 row in set (39.42 sec)
par_sql_complex/HP.log:1 row in set (2 min 45.57 sec)
par_sql_complex/ML.log:1 row in set (4.64 sec)
par_sql_complex/MQ.log:1 row in set (2 min 22.55 sec)
par_sql_complex/NW.log:1 row in set (4 min 26.67 sec)
par_sql_complex/OH.log:1 row in set (1 min 9.67 sec)
par_sql_complex/OO.log:1 row in set (2 min 14.97 sec)
par_sql_complex/PA.log:1 row in set (17.62 sec)
par_sql_complex/PI.log:1 row in set (14.52 sec)
par_sql_complex/PS.log:1 row in set (3.46 sec)
par_sql_complex/RU.log:1 row in set (40.14 sec)
par_sql_complex/TW.log:1 row in set (2 min 32.32 sec)
par_sql_complex/TZ.log:1 row in set (14.16 sec)
par_sql_complex/UA.log:1 row in set (4 min 55.18 sec)
par_sql_complex/US.log:1 row in set (4 min 38.08 sec)
par_sql_complex/WN.log:1 row in set (4 min 56.12 sec)
par_sql_complex/XE.log:1 row in set (24.21 sec)
par_sql_complex/YV.log:1 row in set (20.82 sec)
```

The longest process took nearly 6 mins to process because of the data evenly distributed procedure caused the gap of time processing between each process .