facebook

facebook

Incremental Backup using Row based Replication

Santosh Banda (santoshb@fb.com)
Database Engineering Team, Facebook

Agenda

1. Evolution of MySQL Backups at Facebook
2. Incremental Backup using Row based Replication
3. Benefits and challenges with RBR
4. Facebook MySQL Server Improvements

Evolution of MySQL Backups at Facebook

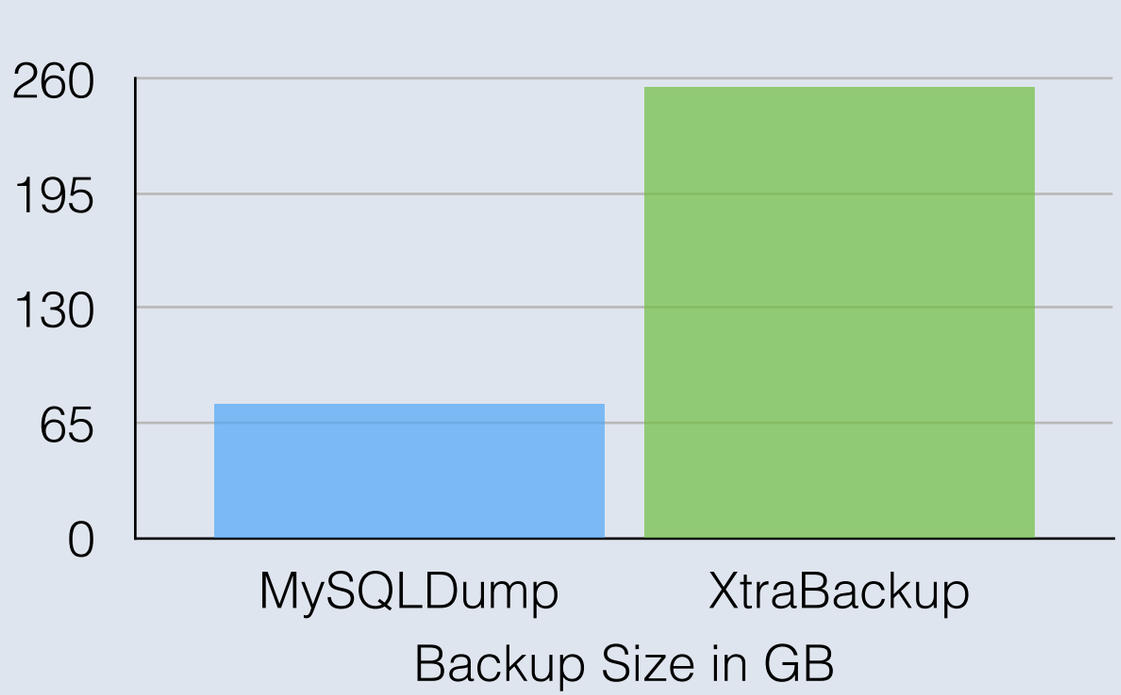
Logical vs Physical

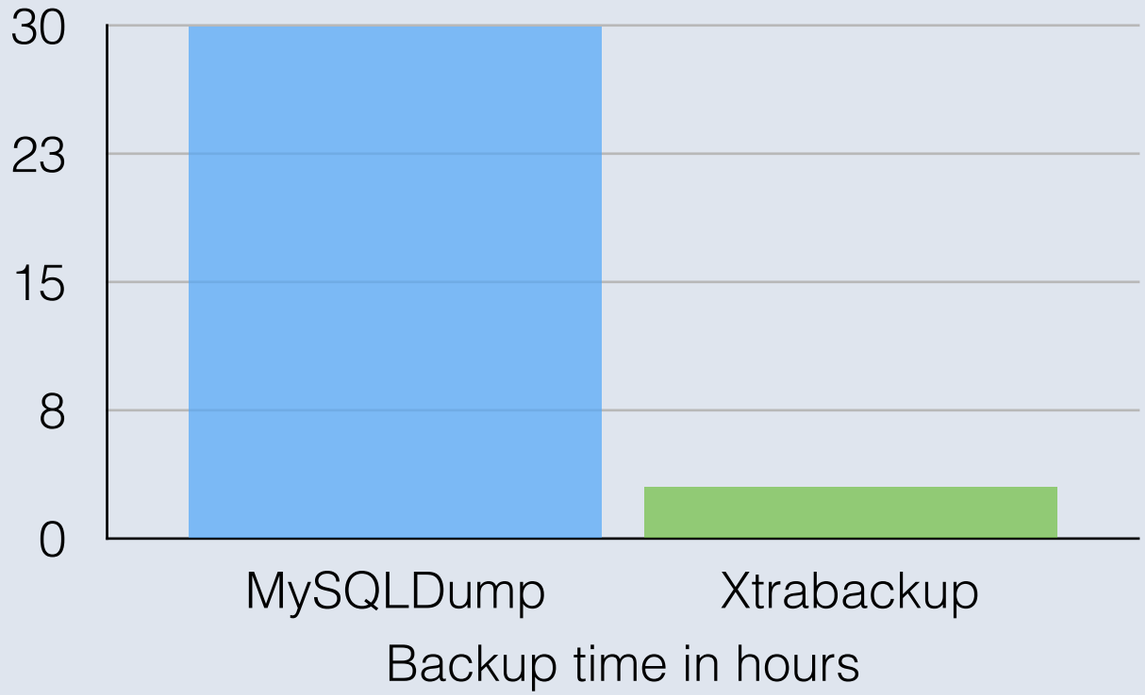
Which one to choose?

	MySQLDump	Xtrabackup
Can external tools easily understand the backup?	**Yes**	No
Size of the backup	**Small**	Big
Is it easy to restore just one table?	**Yes**	No
Debugging corrupt backups	**Easy**	Difficult
Backup and restore time	High	**Low**

Logical backup

MySQLDump

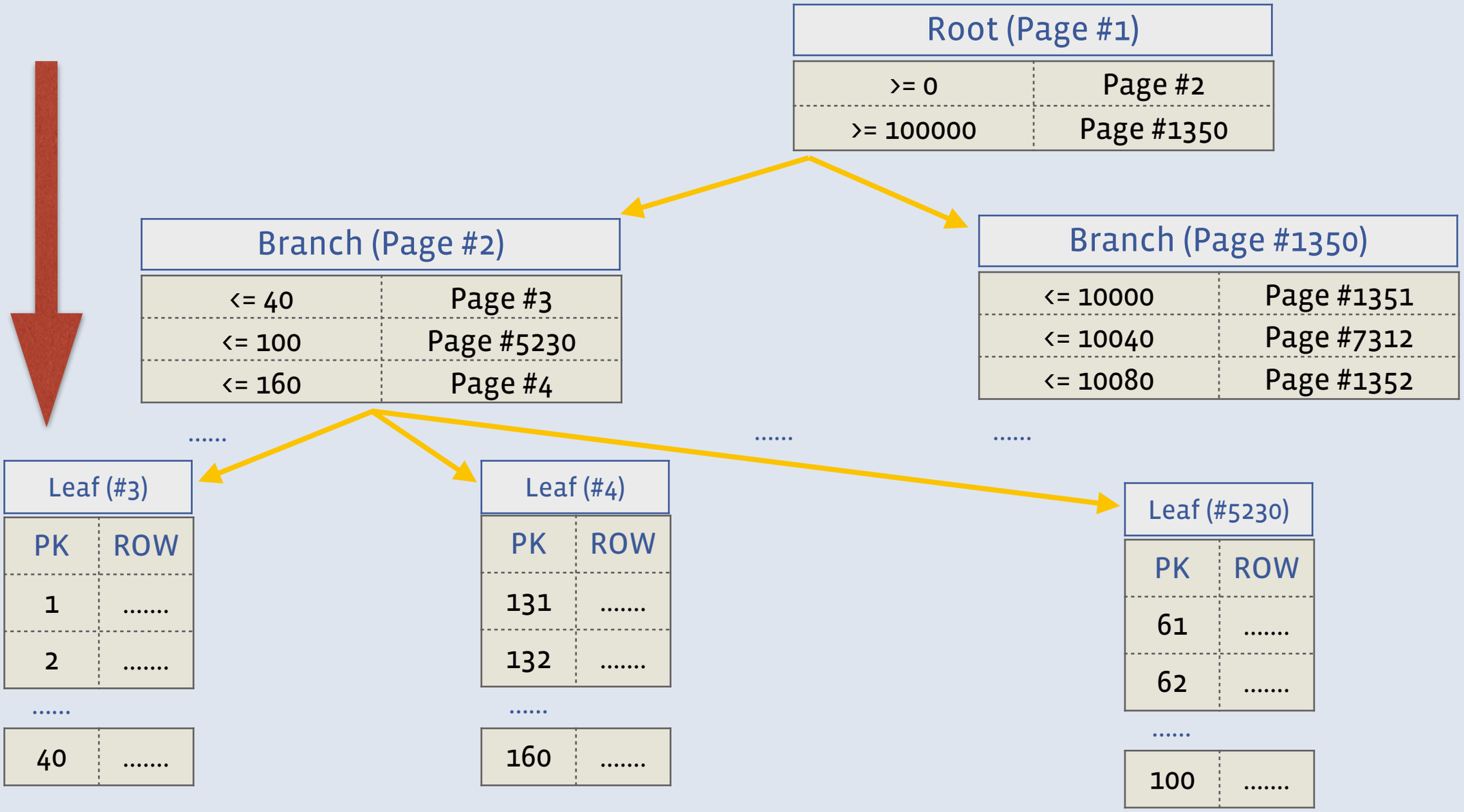




MySQLDump backups are smaller in size but MySQLDump is 10X slow

Logical backup

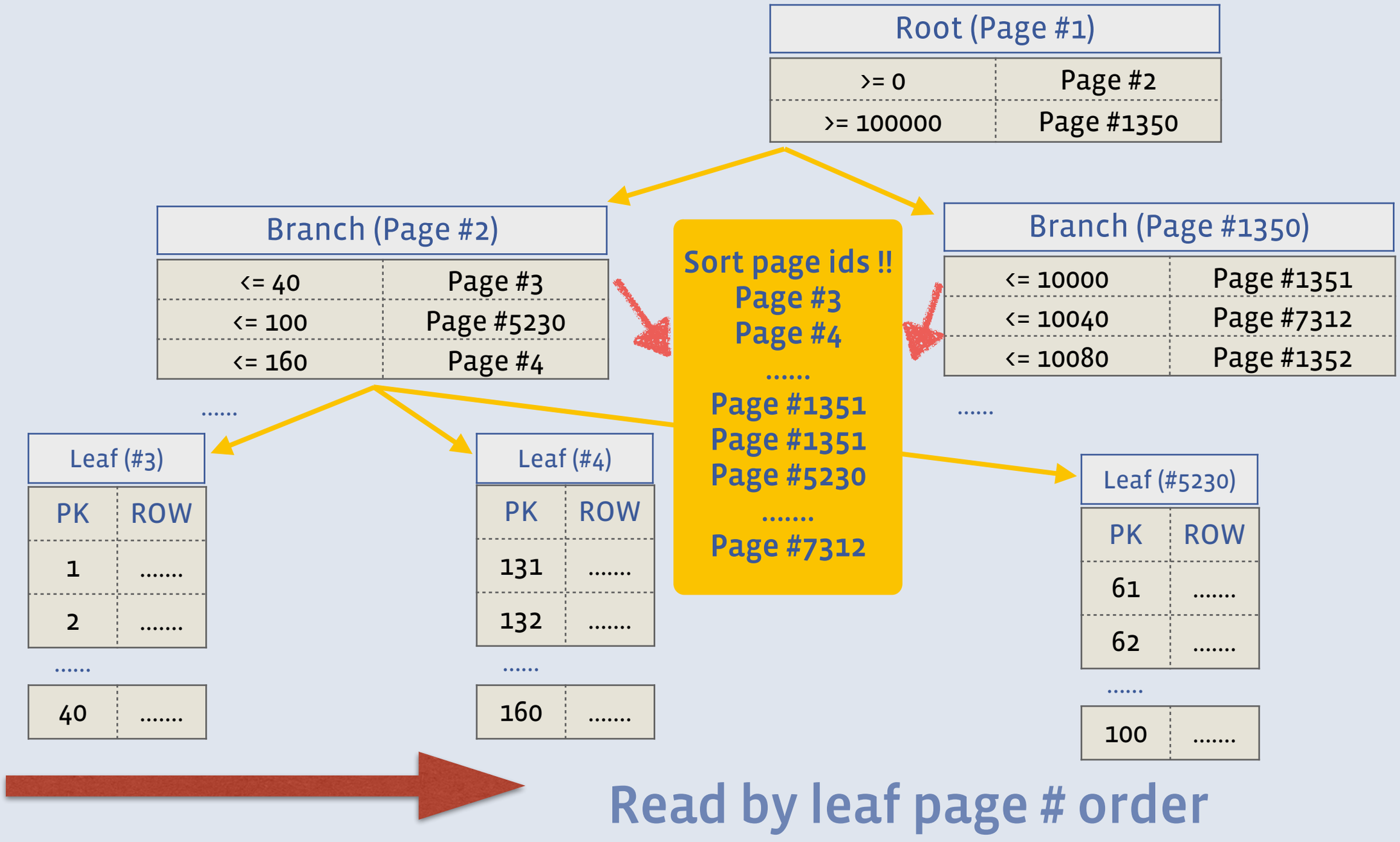
Full table scan



Random read order. Bad for HDD.

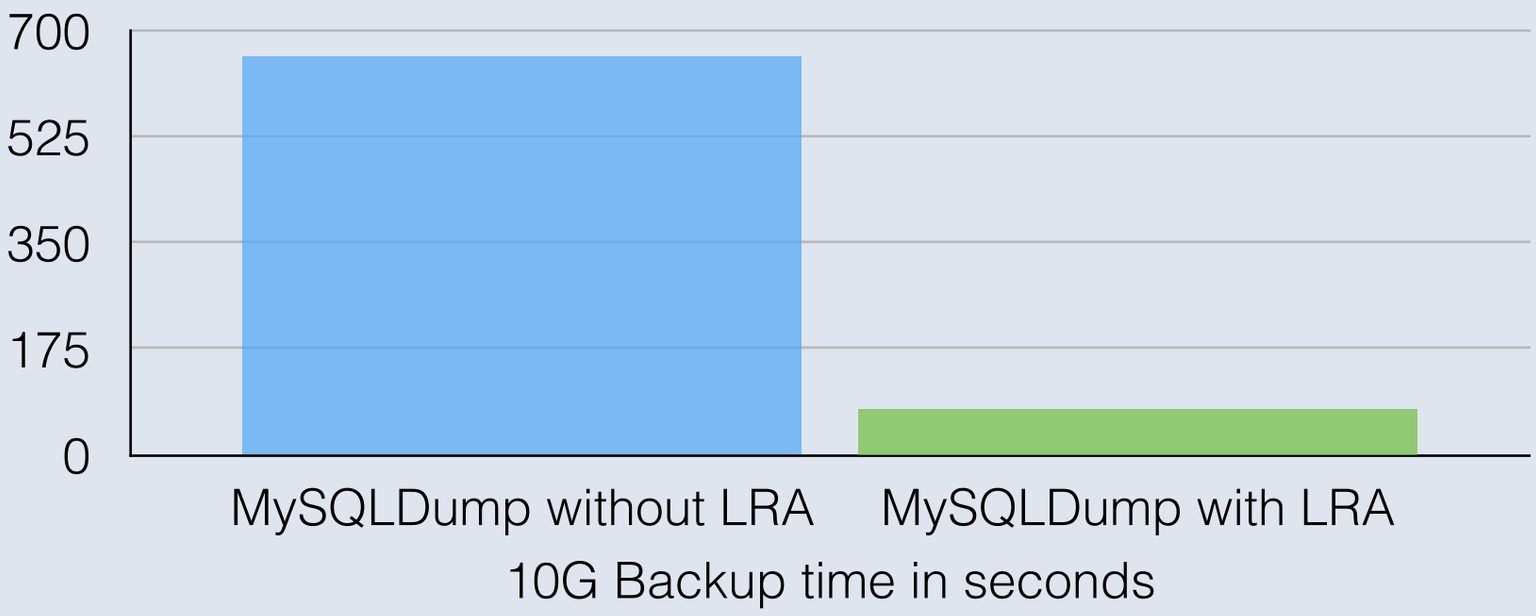
Improve backup speed

MySQLDump with Logical Read Ahead (LRA)



Improve backup speed

Logical Read Ahead (LRA)



MySQLDump with LRA is 10X faster on HDD

Space efficient backups

Differential Backup

Store only the data that has changed by comparing two full MySQLDump backups !!

Differential backup

How does it work?

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(1, 'San Fransisco'),
(2, 'San Jose`),
(3, 'Mountain View'),
.....
.....

Full backup1

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(1, 'San Fransisco'),
(2, 'Santa clara`),
(400, 'Los Angeles'),
.....
.....
.....

Full backup2

No Change

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES

Deleted rows

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES

Inserted rows

Differential backup

How does it work?

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(1, 'San Fransisco'),
(2, 'San Jose`),
(3, 'Mountain View'),
.....
.....

Full backup1

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(1, 'San Fransisco'),
(2, 'Santa clara`),
(400, 'Los Angeles'),
.....
.....
.....

Full backup2

Row updated

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(2, `San Jose`)

Deleted rows

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(2, `Santa Clara`)

Inserted rows

Differential backup

How does it work?

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(1, 'San Fransisco'),
(2, 'San Jose`),
(3, 'Mountain View'),
.....
.....

Full backup1

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(1, 'San Fransisco'),
(2, 'Santa clara`),
(400, 'Los Angeles'),
.....
.....
.....

Full backup2

Row deleted

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(2, `San Jose`)
(3, 'Mountain View')

Deleted rows

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(2, `Santa Clara`)

Inserted rows

Differential backup

How does it work?

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(1, 'San Fransisco'),
(2, 'San Jose`),
(3, 'Mountain View'),
.....
.....

Full backup1

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(1, 'San Fransisco'),
(2, 'Santa clara`),
(400, 'Los Angeles'),
.....
.....
.....

Full backup2

Row inserted

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(2, `San Jose`)
(3, 'Mountain View')

Deleted rows

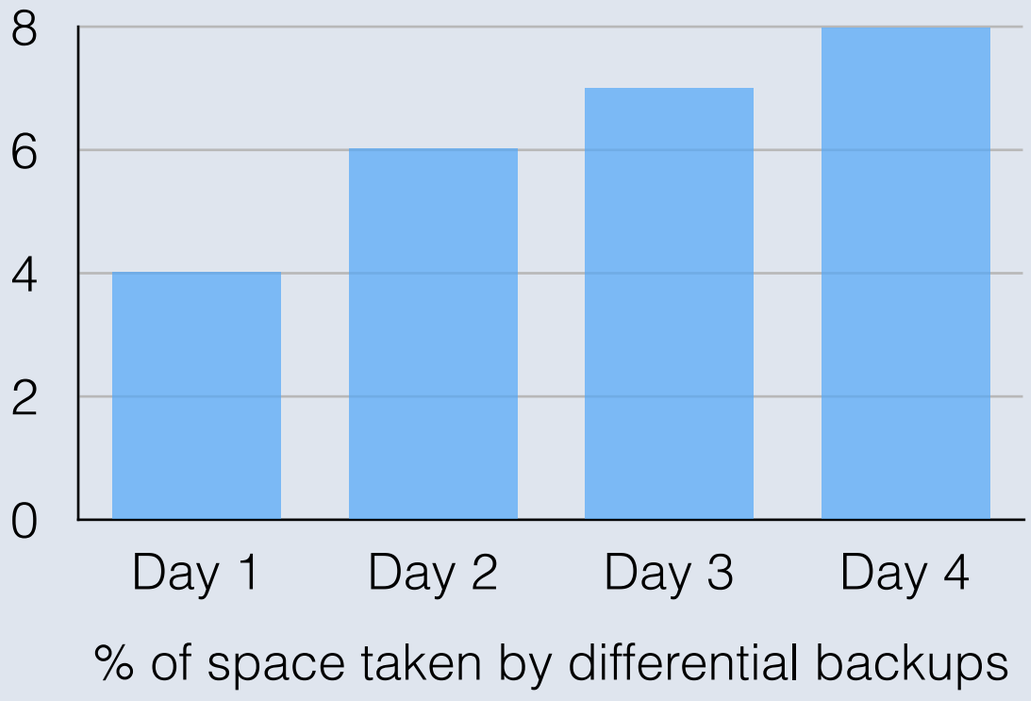
Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(2, `Santa Clara`),
(400, 'Los Angeles');

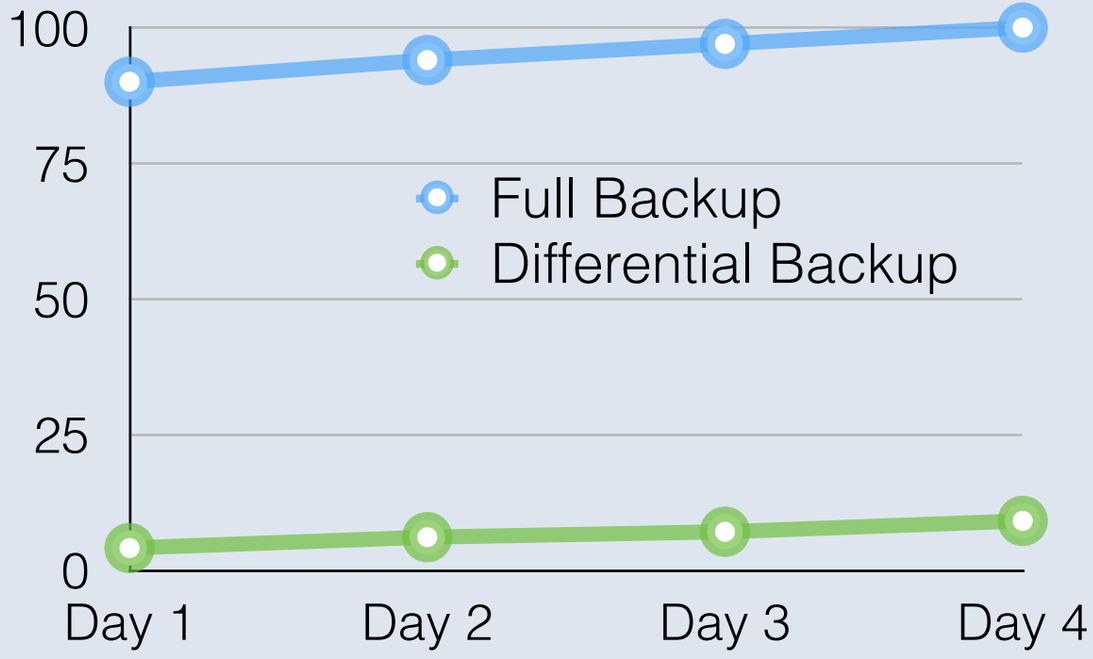
Inserted rows

Space efficient backups

Differential Backup



% of space taken by differential backups



Relative backup space usage

3X-4X total space savings

Incremental Backup using Row based Replication

I/O efficient Backup

Incremental backup

- Goals
  - Remove full table scan
  - Same space usage as differential backup

I/O efficient Backup

Incremental backup

- Why not just use binlog backups ?
  - Not space efficient
  - Huge restore time
  - Need backups per shard

Row based binary logs have row modifications. Use them to create a logical backup !!

Incremental backup

Row based Replication

```
BEGIN;
insert into t1 values(1), (10);
COMMIT;
....
update t1 set a=2 where a=1;
....
delete from t1;
```

binlog_format = STATEMENT

```
Table_map: `test`.`t1` mapped to number 103
Write_rows: table id 103 flags: STMT_END_F

Table_map: `test`.`t1` mapped to number 103
Update_rows: table id 103 flags: STMT_END_F

Table_map: `test`.`t1` mapped to number 103
Delete_rows: table id 103 flags: STMT_END_F
```

binlog_format = ROW

Row modifications

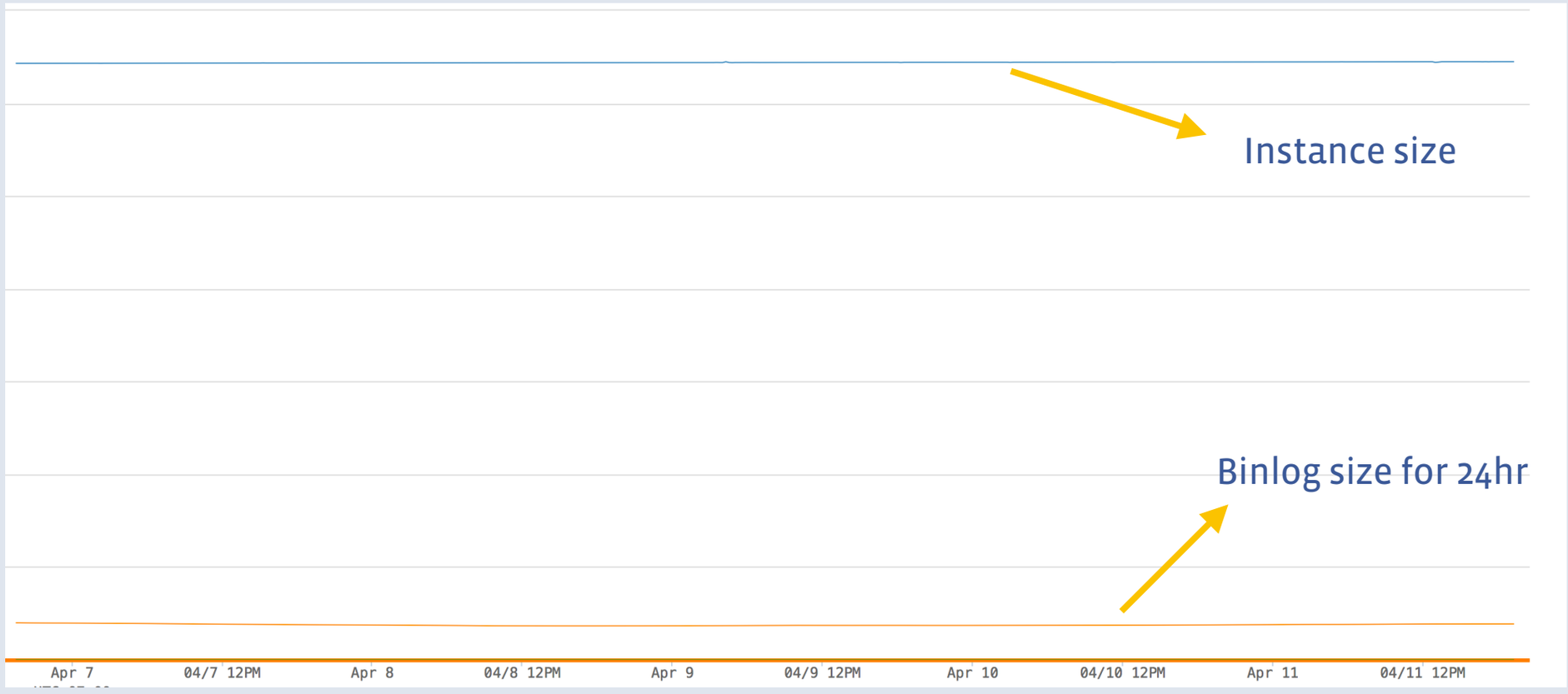
 # Insert (a = 1), (a = 10)

 # Update (a =1) -> (a = 2)

 # Delete (a = 2), (a = 10)

Incremental Backup

Performance ?



Binlog size is **10X** less than instance size

10X I/O improvement

Incremental backup

How does it work?

Row modifications

 # Insert (4, ‘Los Angeles)

 # Update (2, ‘San Jose’) -> (2, ‘New york’)

 # Delete (3, ‘Mountain View’)

.....
.....
.....

 # Update (2, ‘New York’) -> (2, ‘Santa Clara’)

.....

 # Delete (4, ‘Los Angeles’)

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

Deleted rows

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(4, ‘Los Angeles’)

Inserted rows

Incremental backup

How does it work?

Row modifications

 # Insert (4, ‘Los Angeles)

 # Update (2, ‘San Jose’) -> (2, ‘New york’)

 # Delete (3, ‘Mountain View’)

.....
.....
.....

 # Update (2, ‘New York’) -> (2, ‘Santa Clara’)

.....

 # Delete (4, ‘Los Angeles’)

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */
INSERT INTO t1 VALUES
(2, ‘San Jose’)

Deleted rows

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(4, ‘Los Angeles’),
(2, ‘New York’)

Inserted rows

Incremental backup

How does it work?

Row modifications

 # Insert (4, ‘Los Angeles)

 # Update (2, ‘San Jose’) -> (2, ‘New york’)

 # Delete (3, ‘Mountain View’)

.....
.....
.....

 # Update (2, ‘New York’) -> (2, ‘Santa Clara’)

.....

 # Delete (4, ‘Los Angeles’)

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */
INSERT INTO t1 VALUES
(2, ‘San Jose’),
(3, ‘Mountain View’)

Deleted rows

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(4, ‘Los Angeles’),
(2, ‘New York’)

Inserted rows

Incremental backup

De-duplication

Row modifications

 # Insert (4, ‘Los Angeles)

 # Update (2, ‘San Jose’) -> (2, ‘New york’)

 # Delete (3, ‘Mountain View’)

.....
.....
.....

 # Update (2, ‘New York’) -> (2, ‘Santa Clara’)

.....

 # Delete (4, ‘Los Angeles’)

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */
INSERT INTO t1 VALUES
(2, ‘San Jose’)
(3, ‘Mountain View’)

Deleted rows

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(4, ‘Los Angeles’),
(2, ‘Santa Clara’)

Inserted rows

Remove intermediate rows !!

Incremental backup

De-duplication

Row modifications

 # Insert (4, ‘Los Angeles)

 # Update (2, ‘San Jose’) -> (2, ‘New york’)

 # Delete (3, ‘Mountain View’)

.....
.....
.....

 # Update (2, ‘New York’) -> (2, ‘Santa Clara’)

.....

 # Delete (4, ‘Los Angeles’)

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */
INSERT INTO t1 VALUES
(2, ‘San Jose’)
(3, ‘Mountain View’)

Deleted rows

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(4, ‘Los Angeles’),
(2, ‘Santa Clara’)

Inserted rows

Remove intermediate rows !!

Incremental backup

De-duplication

Row modifications

 # Insert (4, ‘Los Angeles)

 # Update (2, ‘San Jose’) -> (2, ‘New york’)

 # Delete (3, ‘Mountain View’)

.....
.....
.....

 # Update (2, ‘New York’) -> (2, ‘Santa Clara’)

.....

 # Delete (4, ‘Los Angeles’)

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */
INSERT INTO t1 VALUES
(2, ‘San Jose’)
(3, ‘Mountain View’)

Deleted rows

Create TABLE t1 (user_id int, city char(50);
/* ORDERING KEY : (user_id) */

INSERT INTO t1 VALUES
(2, ‘Santa Clara’)

Inserted rows

Same backup size as differential backup

Incremental Backup

Performance?

Peak RAM	4 GB (increases further)
CPU util	~8%
Time	20 seconds per 1GB binlog
Backup size	1.8 GB

In memory de-duplication on 40GB binlog

Peak RAM	120MB
CPU util	~2.5%
Time	32 seconds per 1GB binlog
Backup size	1.8 GB

Disk based de-duplication on 40GB binlog

Incremental backup

Benefits

Removes full table scan

No need to interact with MySQL server for backups

10X improvements in speed, I/O and network usage

Incremental backup

How to make backup consistent?

GTID to the rescue

```
--
-- GTID state extracted from the output of START TRANSACTION
--


/*SET @@GLOBAL.GTID_PURGED='1bcab207-a709-11e4-8967-f45214bfc262:1-6815437,
1f40cf35-a704-11e4-8947-f45214bd996a:1-1578 ;*/

--
-- Position to start replication or point-in-time recovery from
--

-- CHANGE MASTER TO MASTER_LOG_FILE='binary-logs.000836', MASTER_LOG_POS=60593170;
```

mysqldump --single-transaction --gtid-mode=COMMENTED

Incremental backup

How to make backup consistent?

In fb-mysql binlog index file has GTID executed along with the binary log file names

binary-logs. 000800’ 1bcab207-a709-11e4-8967-f45214bfc262:1-5815437,
1f40cf35-a704-11e4-8947-f45214bd996a:1-1578
……
……
…..
binary-logs. 000836’ 1bcab207-a709-11e4-8967-f45214bfc262:1-6815437,
1f40cf35-a704-11e4-8947-f45214bd996a:1-1578
…..
…..
….
….
binary-logs. 000900’ 1bcab207-a709-11e4-8967-f45214bfc262:1-8815437,
1f40cf35-a704-11e4-8947-f45214bd996a:1-1578

binary-logs.index

Incremental backup

How to make backup consistent?

In fb-mysql binlog index file has GTID executed along with the binary log file names

binary-logs. 000800’ 1bcab207-a709-11e4-8967-f45214bfc262:1-5815437,
1f40cf35-a704-11e4-8947-f45214bd996a:1-1578
……
……
…..
binary-logs. 000836’ 1bcab207-a709-11e4-8967-f45214bfc262:1-6815437,
1f40cf35-a704-11e4-8947-f45214bd996a:1-1578
…..
…..
….
….
binary-logs. 000900’ 1bcab207-a709-11e4-8967-f45214bfc262:1-8815437,
1f40cf35-a704-11e4-8947-f45214bd996a:1-1578

binary-logs.index

Find starting position by using GTID in full backup

Incremental backup

Looks good. Ship it !!

Incremental backup

Row based Replication

Row based replication

Benefits

- Non deterministic queries are safe with RBR
- Better performance than SBR
  - Fewer locks required
  - Slave doesn't need to understand huge context
- Easy for external customers to consume data changes
- Better handling of data drift

Facebook MySQL Server Improvements

Row based Replication

```
# at 289
#010908 18:46:40 server id 1  end_log_pos 411   Rows_query
# /* Need query comments */ insert into t1 values (101,'{ "id":101, "name":"Alex", "phone":6507770001}')
# at 411
#010908 18:46:40 server id 1  end_log_pos 456   Table_map: `test`.`t1` mapped to number 30
# at 456
```

Use binlog_rows_query_log_events=ON

```
# at 677
#010908 18:46:40 server id 1  end_log_pos 748   Rows_query
# /* Need query comments */ /* but don\'t need SQL */
# at 748
#010908 18:46:40 server id 1  end_log_pos 793   Table_map: `test`.`t1` mapped to number 30
# at 793
```

Use binlog_rows_query_log_events=ON and log_only_query_comments=TRUE

Row based Replication

Execution state

```
mysql> show processlist;
+--------+------------+-----------------------------------+------+---------+------+-----------------------------------------+
| Id     | User       | Host                              | db   | Command | Time | State                                   |
+--------+------------+-----------------------------------+------+---------+------+-----------------------------------------+
| 455635 | system user |                                  | NULL | Connect |   13 | Waiting for master to send event
| 455636 | system user |                                  | NULL | Connect |  267 | Slave has read all relay log;
                                                                                  waiting for the slave I/O thread to update it
```

```
mysql> show slave status\G
*************************** 1. row ***************************

        Retrieved_Gtid_Set: fa5cece9-c262-11e4-bbc0-0002c9d31542:5-7057111
        Executed_Gtid_Set: fa5cece9-c262-11e4-bbc0-0002c9d31542:1-950694
```

Row based Replication

Execution state

```
mysql> show processlist;
+--------+------------+-----------------------------------+------+---------+------+-----------------------------------------+
| Id     | User       | Host                              | db   | Command | Time | State                                   |
+--------+------------+-----------------------------------+------+---------+------+-----------------------------------------+
| 455635 | system user |                                  | NULL | Connect |   13 | Waiting for master to send event
| 455636 | system user |                                  | NULL | Connect |  267 | Executing Write_rows event
                                                                                  at position 25652671
```

Ported from twitter's MySQL 5.5

Row based Replication

Slave side triggers

- Triggers are not fired when executing RBR events

Port mariadb's slave side trigger feature to Fb-MySQL

Row based Replication

Master side triggers

- Triggers modifications on master are logged breaking slaves

New option sql_log_bin_triggers added to Fb-MySQL

Row based Replication

Schema differences

- RBR is not compatible with schema changes
  - new columns added in the middle of a slave's table schema
  - dropped columns in slave's table schema
  - different column order
  - data type change

New option log_column_names added to Fb-MySQL

Alter table t1 rbr_column_names=ON

Row based Replication

Some more issues

- Row modifications don't have all the schema information
- Watch out for `delete from t1` on a 700G table !!

References

- Logical Read Ahead
- GTID in start transaction
- Online Schema Change
- RBR execution state
- MariaDB slave side triggers
- Sql_log_bin_triggers
- RBR column names
- Differential and incremental backup tools. Not yet open sourced

Questions?

facebook