

2012

# 阿里技术沙龙

享技术·聚朋友

D2



## 淘宝MySQL十大经典案例

阿里集团-淘宝网DBA

@杨德华Devin 2012-09

aDev

iDevOps

iData TCon

# 个人简介

- 负责淘宝用户中心从IOE迁移到MySQL集群的性能测试,高可用运维,DB可用率达到100%
- 管理数据魔方,SNS,淘宝评价,webww,notify等MySQL集群
- 服务器优化,成本节省,把某个业务线的MySQL机器数量从100台减少到70台。
- SAS->SSD/FIO的MySQL服务器升级过程
- 负责MySQL5.1.45->5.1.48-> Percona 5.5.18的工作

# 淘宝线上十大MySQL经典案例

- 数据库设计相关:

- (一) InnoDB表如何设计主键索引

- SQL相关:

- (二) 字符串索引隐式转换
  - (三) 表数据被莫名清空
  - (四) InnoDB表更新锁问题

- 客户端相关:

- (五) 客户端连接被中断

# 淘宝线上十大MySQL经典案例

- “灵异事件”相关:

- (六) 核心数据库被同时关闭

- Slave相关:

- (七) Slave 事件Loop
- (八) Slave 更新操作找不到对应记录
- (九) 备库设置read\_only被堵塞

- Swap相关:

- (十) 数据库服务器Swap

# 一 InnoDB表如何设计主键索引

```
CREATE TABLE `a` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `message_id` int(11) NOT NULL,  
  `user_id` int(11) NOT NULL,  
  `msg` varchar(1024) DEFAULT NULL,  
  `gmt_create` datetime NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `user_id` (`user_id`,`message_id`),  
  KEY `idx_gmt_create` (`gmt_create`)  
  ) ENGINE=InnoDB DEFAULT CHARSET=gbk;
```

```
CREATE TABLE `b` (  
  `user_id` int(11) NOT NULL,  
  `message_id` int(11) NOT NULL,  
  `msg` varchar(1024) DEFAULT NULL,  
  `gmt_create` datetime NOT NULL,  
  PRIMARY KEY (`user_id`,`message_id`),  
  KEY `idx_gmt_create` (`gmt_create`)  
  ) ENGINE=InnoDB DEFAULT CHARSET=gbk;
```

# 一 InnoDB表如何设计主键索引

```
CREATE TABLE `a` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `message_id` int(11) NOT NULL,  
  `user_id` int(11) NOT NULL,  
  `msg` varchar(1024) DEFAULT NULL,  
  `gmt_create` datetime NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `user_id` (`user_id`,`message_id`),  
  KEY `idx_gmt_create` (`gmt_create`)  
) ENGINE=InnoDB DEFAULT CHARSET=gbk;
```

```
CREATE TABLE `b` (  
  `user_id` int(11) NOT NULL,  
  `message_id` int(11) NOT NULL,  
  `msg` varchar(1024) DEFAULT NULL,  
  `gmt_create` datetime NOT NULL,  
  PRIMARY KEY (`user_id`,`message_id`),  
  KEY `idx_gmt_create` (`gmt_create`)  
) ENGINE=InnoDB DEFAULT CHARSET=gbk;
```

大多数互联网业务(用户,消息), 都可以选择a或者b来满足业务需求, 但a表和b表有何区别?



# 一 InnoDB表如何设计主键索引

	记录	空间	优点	缺点
A表	500万 (顺序)	509M	主键ID自增,在写入数据的时候,Btree分裂成本低,写性能高	?
B表	500万 (随机)	361M	?	?

# 一 InnoDB表如何设计主键索引

	记录	空间	优点	缺点
A表	500万 (顺序)	509M	主键ID自增,在写入数据的时候,Btree分裂成本低,写性能高	?
B表	500万 (随机)	361M	1.物理空间相对减少 2.根据user_id查数据,直接走主键拿到数据,无需回表	?



# 一 InnoDB表如何设计主键索引

	记录	空间	优点	缺点
A表	500万	509M	主键ID自增,在写入数据的时候,Btree分裂成本低,写性能高	物理空间相对较多 如果根据user_id来查记录,需要走两次IO
B表	500万	361M	1.物理空间相对减少 2.根据user_id查数据,直接走主键拿到数据,无需回表	?

# 一 InnoDB表如何设计主键索引

	记录	空间	优点	缺点
A表	500万	509M	主键ID自增,在写入数据的时候,Btree分裂成本低,写性能高	物理空间相对较多 如果根据user_id来查记录,需要走两次IO
B表	500万	361M	1.物理空间相对减少 2.根据user_id查数据,直接走主键拿到数据,无需回表	(user_id,message_id)为随机写入,Btree分裂成本高,写性能低

# InnoDB表如何设计主键索引

	优点	适用场景?
A表	主键ID自增,在写入数据的时候,Btree分裂成本低,写性能高	写操作较多的场景
B表	1.物理空间相对减少 2.根据user_id查数据,直接走主键拿到数据,无需回表	

# InnoDB表如何设计主键索引

	优点	适用场景?
A表	主键ID自增,在写入数据的时候,Btree分裂成本低,写性能高	写操作较多的场景
B表	1.物理空间相对减少 2.根据user_id查数据,直接走主键拿到数据,无需回表	写少读多的场景,例如从hadoop回流到MySQL的统计结果表,这种统计结果一般数据较多,但主要是读

# SQL相关-案例二字符串索引隐式转换

```
Create Table: CREATE TABLE `index_str` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `user_id` varchar(30) NOT NULL,  
  `name` varchar(30) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `idx_user_id` (`user_id`)  
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT  
CHARSET=gbk
```

# SQL相关-案例二字符串索引隐式转换

```
Create Table: CREATE TABLE `index_str` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `user_id` varchar(30) NOT NULL,  
  `name` varchar(30) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `idx_user_id` (`user_id`)
```

Id一般为数字，但这个表把id定义为了varchar类型。

```
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT  
CHARSET=gbk
```

# SQL相关-案例二字符串索引隐式转换

```
root@test 11:39:38>select * from index_str;
```

```
+-----+-----+-----+
| id | user_id | name |
```

```
+-----+-----+-----+
```

```
| 1 | 1111 | NULL |
```

```
| 2 | 0001 | NULL |
```

```
| 3 | 1 | NULL |
```

```
+-----+-----+-----+
```

```
3 rows in set (0.00 sec)
```

恩...有三条记录



# SQL相关-案例二字符串索引隐式

```
root@test 11:51:14>explain select * from index_str where user_id=1\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: index_str
         type: ALL
possible_keys: idx_user_id
          key: NULL
        key_len: NULL
         ref: NULL
         rows: 3
      Extra: Using where
1 row in set (0.00 sec)
```


当user\_id=1的时候,查询分析器表示没用到idx\_user\_id索引

# SQL相关-案例二字符串索引隐式转换

```
root@test 11:50:32>explain select * from index_str where user_id='1'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: index_str
         type: ref
possible_keys: idx_user_id
          key: idx_user_id
        key_len: 62
         ref: const
        rows: 1
   Extra: Using where
1 row in set (0.00 sec)
```

当user\_id='1'的时候,查询分析器用到了idx\_user\_id的索引

# SQL相关案例二 字符串索引隐式转换 原因分析

- 数字类型的0001等价于1
-  字符串的0001和1不等值
- 当字符串的列有对应的索引，而在where条件里面不指定为字符串，index无法确认最终的记录

# SQL相关-案例二字符串索引隐式转换

```
root@test 11:58:42> select * from index_str where user_id=0001\G
***** 1. row *****
      id: 2
user_id: 0001
      name: NULL
***** 2. row *****
      id: 3
user_id: 1
      name: NULL
2 rows in set (0.00 sec)
```

user\_id=0001的时候,MySQL返回两条记录(被转换成1)

```
root@test 11:58:48> select * from index_str where user_id='0001'\G
***** 1. row *****
      id: 2
user_id: 0001
      name: NULL
1 row in set (0.00 sec)
```

user\_id='0001'的时候,MySQL返回一条记录

# SQL相关-案例二字符串索引隐式转换

```
root@test 11:59:03> select * from index_str where user_id=1\G
***** 1. row *****
      id: 2
user_id: 0001
      name: NULL
***** 2. row *****
      id: 3
user_id: 1
      name: NULL
2 rows in set (0.00 sec)
```

**user\_id=1**的时候,MySQL返回两条记录(被转换成数字值)

```
root@test 11:59:21> select * from index_str where user_id='1'\G
***** 1. row *****
      id: 3
user_id: 1
      name: NULL
1 row in set (0.00 sec)
```

**user\_id='1'**的时候,MySQL返回两条记录

# SQL相关-案例二字符串索引隐式转换

- 被隐式转换后，会进行全表遍历



- 建表需要注意对应好字段类型



# SQL相关 案例三 表被莫名清空

- root@test 12:34:22>select \* from test\_delete;

- +-----+

- | id |

- +-----+

- | 1 |

- | 2 |

- | 3 |

- | 4 |

- | 5 |

- +-----+

- 5 rows in set (0.00 sec)

全表清理的普通写法

1.delete from test\_delete;

2.delete from test\_delete  
where id in(1,2,3,4,5);

3.truncate table test\_delete;

4.delete from test\_delete  
where id >0;

还有什么二逼语句可以删除全表数据?



# SQL相关 案例三 表被莫名清空

1. delete from test\_delete where 'a'='a';

Query OK, 5 rows affected (0.00 sec)

2. delete from test\_delete where id=1 or 'a'='a';

Query OK, 5 rows affected (0.00 sec)

3. delete from test\_delete where id;

Query OK, 5 rows affected (0.00 sec)

# SQL相关 案例三 表被莫名清空

```
DELETE FROM test_delete
```

```
WHERE EXISTS (SELECT *
```

```
FROM (SELECT *
```



```
FROM test_delete
```

```
WHERE id = 5) AS b);
```

Query OK, 5 rows affected (0.00 sec)

# SQL相关 案例三 表被莫名清空

```
DELETE FROM test_delete
WHERE EXISTS (SELECT *
FROM (SELECT *
FROM test_delete
WHERE id = 5) AS b);
```



Query OK, 5 rows affected (0.00 sec)

exists后面的子查询是有返回值,恒真.  
导致前面的delete语句被触发

# SQL相关 案例四 InnoDB表更新锁问题

```
CREATE TABLE `a` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `message_id` int(11) NOT NULL,  
  `user_id` int(11) NOT NULL,  
  `msg` varchar(1024) DEFAULT NULL,  
  `gmt_create` datetime NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `user_id` (`user_id`,`message_id`),  
  KEY `idx_gmt_create` (`gmt_create`)  
) ENGINE=InnoDB DEFAULT CHARSET=gbk;
```

业务需要订正数据

tx\_isolation=REPEATABLE-READ

```
UPDATE message_auto  
SET   gmt_create = Now()  
WHERE msg IS NOT NULL;
```

msg is not null 用不到索引,也没有索引  
对于其他insert,delete,update,select会有何影响?

```
root@test 01:21:21>update message_auto set  
gmt_create=now() where mtext is not null;
```

Query OK, 5000002 rows affected (1 min 21.88 sec) Rows matched: 5000002 Changed:  
5000002 Warnings: 0

```
root@test 01:21:23>update message_auto set  
gmt_create=now() where id=1;
```

Query OK, 1 row affected (1 min 16.96 sec) Rows matched: 1 Changed: 1 Warnings: 0



```
root@test 01:22:08>insert into message_auto  
value(100000001,1,1,'hello','hello',now());
```

Query OK, 1 row affected (0.00 sec)

```
root@test 01:22:12>update message_auto set  
gmt_create=now() where id=100000001;
```

Query OK, 1 row affected (0.01 sec) Rows matched: 1 Changed: 1 Warnings: 0

```
root@test 01:21:21>update message_auto set  
gmt_create=now() where mtext is not null;
```

Query OK, 5000002 rows affected (1 min 21.88 sec) Rows matched: 5000002 Changed:  
5000002 Warnings: 0

```
root@test 01:21:23>update message_auto set  
gmt_create=now() where id=1;
```

Query OK, 1 row affected (1 min 16.96 sec) Rows matched: 1 Changed: 1 Warnings: 0

id=1的记录update操作需要和update全表差不多的时间

```
root@test 01:22:08>insert into message_auto  
value(100000001,1,1,'hello','hello',now());
```

Query OK, 1 row affected (0.00 sec)

```
root@test 01:22:12>update message_auto set  
gmt_create=now() where id=100000001;
```

Query OK, 1 row affected (0.01 sec) Rows matched: 1 Changed: 1 Warnings: 0

insert新的记录和update最新的记录，时间非常短



# SQL相关 案例四 InnoDB表更新锁问题

```
show engine innodb status\G
```

mysql tables in use 1, locked 1

649 lock struct(s), heap size 80312, **146316 row lock(s)**, undo log entries 145669

MySQL thread id 2675649, query id 610234950

localhost root Updating

update message\_auto set gmt\_create=now()  
where mtext is not null



# 案例五 客户端连接被中断

- PHP Warning: mysql\_connect() [

# 案例五 客户端连接被中断

- PHP Warning: mysql\_connect() [

pererror 99

OS error code 99: Cannot assign requested address

# 案例五 客户端连接被中断

- 客户端的TCP连接相关统计信息

80端口连接数(CurrentConnection): 16

nginx进程数量: 10

php进程数量: 130

TCP连接状态:

TIME\_WAIT 26258

FIN\_WAIT1 1

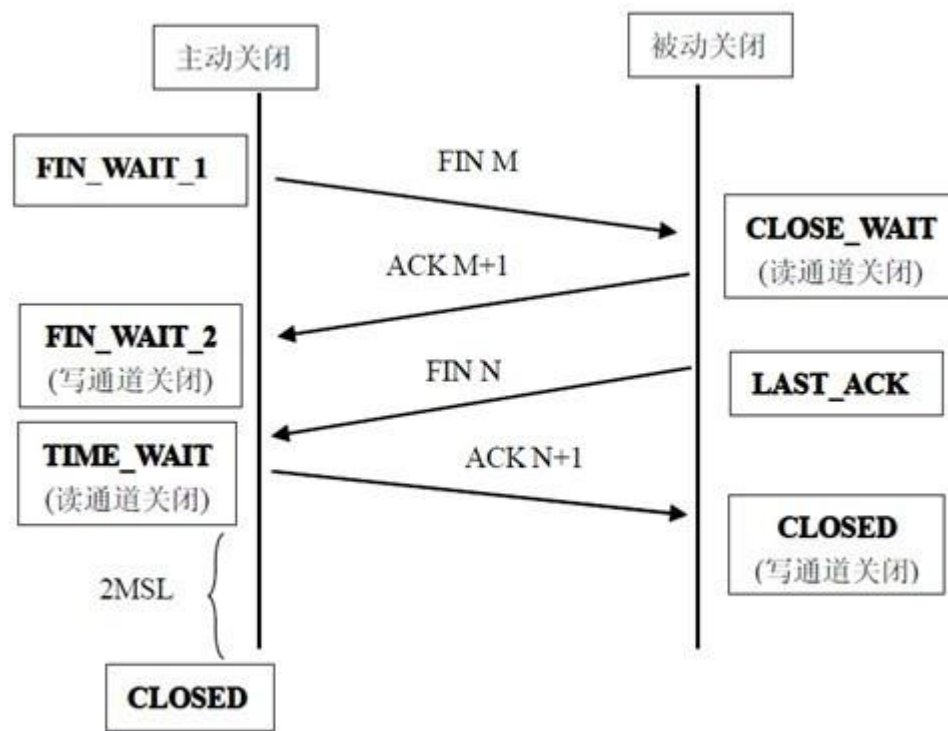
FIN\_WAIT2 6

ESTABLISHED 893

TIME\_WAIT的数量非常多,也可以通过netstat -ant | grep TIME\_WAIT来排查

# 案例五 客户端连接被中断

- TIME\_WAIT过多的原因
- nginx使用了短连接方式，会造成大量处于TIME\_WAIT状态的连接



# 案例五 客户端连接被中断

- 解决办法

- 让TIME\_WAIT状态可以重用，这样即使TIME\_WAIT占满了所有端口，也不会拒绝新的请求造成障碍

`net.ipv4.tcp_tw_recycle = 1`

`net.ipv4.tcp_tw_reuse = 1` 让TIME\_WAIT尽快回收

# 案例五 客户端连接被中断

## 其他案例

Cause: org.jboss.util.NestedSQLException: No ManagedConnections available within configured blocking timeout ( 1000 [ms] ); - nested throwable: (javax.resource.ResourceException: No ManagedConnections available within configured blocking timeout ( 1000 [ms] ))

Java服务器频繁GC导致没有可用连接池



# 案例五 客户端连接被中断

## 其他案例

- 2011-02-23 11:47:16 WARNING  
CONNECT\_ERROR -  
{“host”:“172.19.70.59”,“port”:3306,“user”:“USER”,“pass”:“3\*\*\*\*\*2”,“error”:“Can’t connect to MySQL server on ‘客户端IP’ (4)”}

错误代码4是表示系统中断  
PHP服务器文件句柄不足,连接被中断



# 案例六 两台核心数据库被同时关闭

昨天下午14点44分

两台核心MySQL数据库在同时升级一个任务调度程序

120921 14:44:54 [Note] /u01/mysql/libexec/mysqld:  
Normal shutdown

A服务器

120921 14:44:54 [Note] /u01/mysql/libexec/mysqld:  
Normal shutdown

B服务器

# 案例六 两台核心数据库被同时关闭

120921 14:44:54 [Note] /u01/mysql/libexec/mysqld:  
Normal shutdown

共同点 都是同一个时间点开始  
Normal shutdown

120921 14:44:54 [Note] /u01/mysql/libexec/mysqld:  
Normal shutdown

Kill -9 mysqld\_pid  
mysqld\_safe Number of processes running now: 0  
mysqld\_safe mysqld restarted

# 案例六 两台核心数据库被同时关闭

可以让MySQL Normal Shutdown的命令  
本地执行

```
mysqladmin -uroot -pxxx shutdown  
mysqladmin -uroot -pxxx sh
```

超级用户远程shutdown

```
mysqladmin -utest -pxxx -hxxxx shutdown
```

单时候开着电脑的DBA不多，除了并发更新监控程序，再没有其他操作。调度程序里面也没有进行shutdown的操作

# 案例六 两台核心数据库被同时关闭

可以让MySQL Normal Shutdown的命令  
本地执行

```
mysqladmin -uroot -pxxx shutdown  
mysqladmin -uroot -pxxx sh
```

超级用户远程shutdown

```
mysqladmin -utest -pxxx -hxxxx shutdown
```

那时候开着电脑的DBA不多，除了并发更新监控程序，再没有其他操作。监控程序里面也没有带有shutdown的操作

# 案例六 两台核心数据库被同时关闭

 kill -9 mysql\_d\_pid 和 kill mysql\_d\_pid 的区别？

# 案例六 两台核心数据库被同时关闭

Kill -9 mysql\_d\_pid

mysql\_safe Number of processes running now: 0

mysql\_safe mysql restarted

 kill -9 mysql\_d\_pid 和 kill mysql\_d\_pid 的区别?

kill mysql\_d\_pid

InnoDB: Normal Shutdown...

# 案例六 两台核心数据库被同时关闭

调度程序(agent)的启动和重启

 启动agent的同时,把自己的pid记录到一个文件

重启agent的时候,获取文件里面的pid,调用  
os.kill(pid)杀掉agent进程



# 案例六 两台核心数据库被同时关闭

相关事件回放

9.13之前:agent和MySQL都存活,假设pid分别为a,m

9.13:机器硬件维护,MySQL正常关闭,机器重启,agent被异常关闭,但记录pid的文件没被删除

9.13: MySQL重新启动,OS分配了pid(a)给MySQL, 但agent没被启动

# 案例六 两台核心数据库被同时关闭

## 相关事件回放

9.21(昨天) 升级agent程序,先将agent关闭,agent里面的代码直接执行了os.kill(a)

由于a这个pid已经是属于MySQL的pid,  
当agent执行os.kill(a)的时候  
MySQL日志打印Normal shutdown

# 案例六 两台核心数据库被同时关闭

## 总结

os.kill(pid)得判断这个pid是否属于自己,避免误杀

OS的pid分配,可能会分配其他已被关闭程序的pid

内核版本 2.6.18-164.el5

/proc/sys/kernel/pid\_max 32768([signed] short int)

# 案例七 Slave 事件Loop

Seconds\_Behind\_Master的定义

每一个 Binlog事件,都存了当前事件的时间戳

IO\_Thread执行当前事件的时间戳 (A)

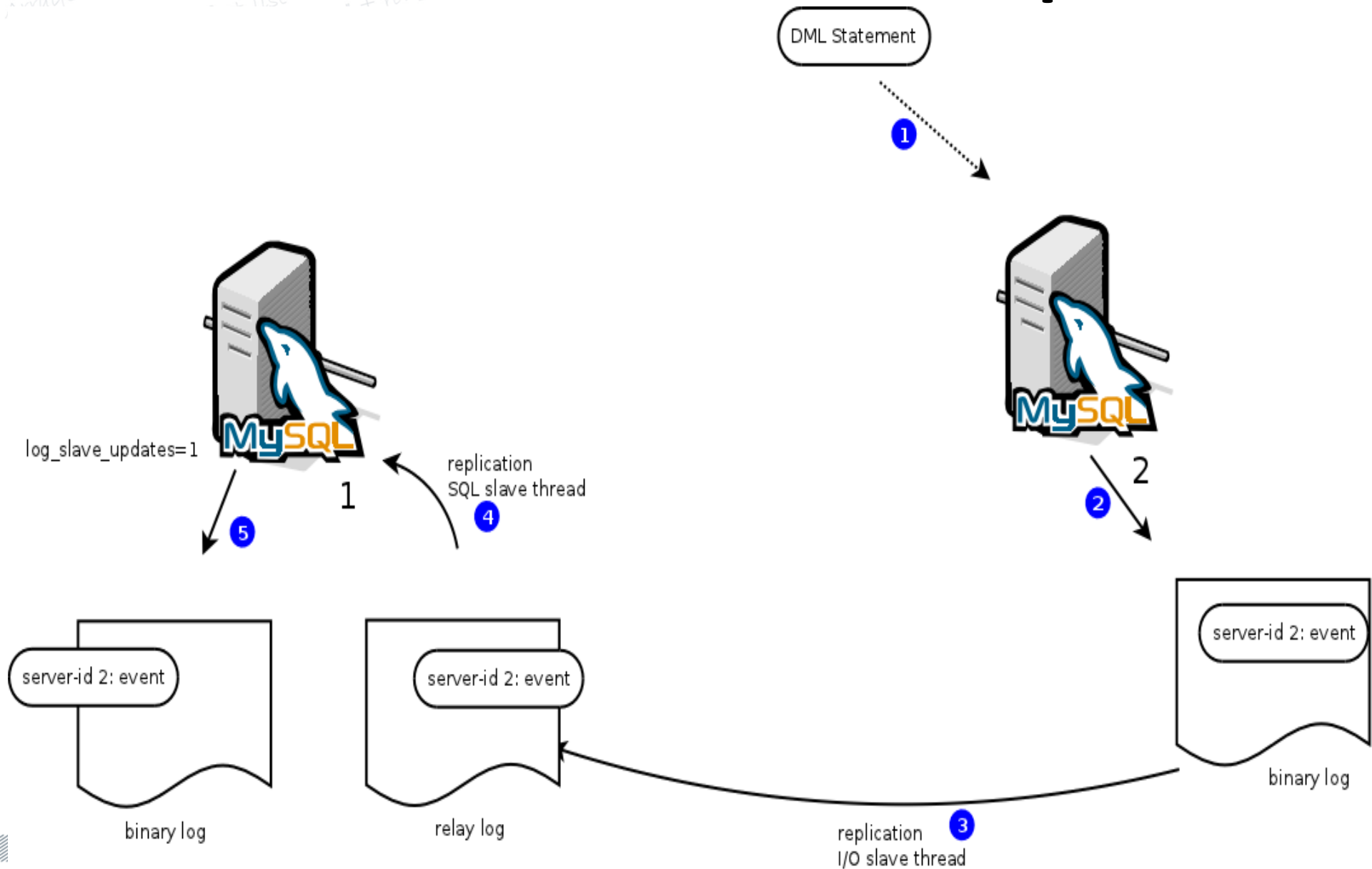


SQL\_Thread执行当前事件的时间戳 (B)

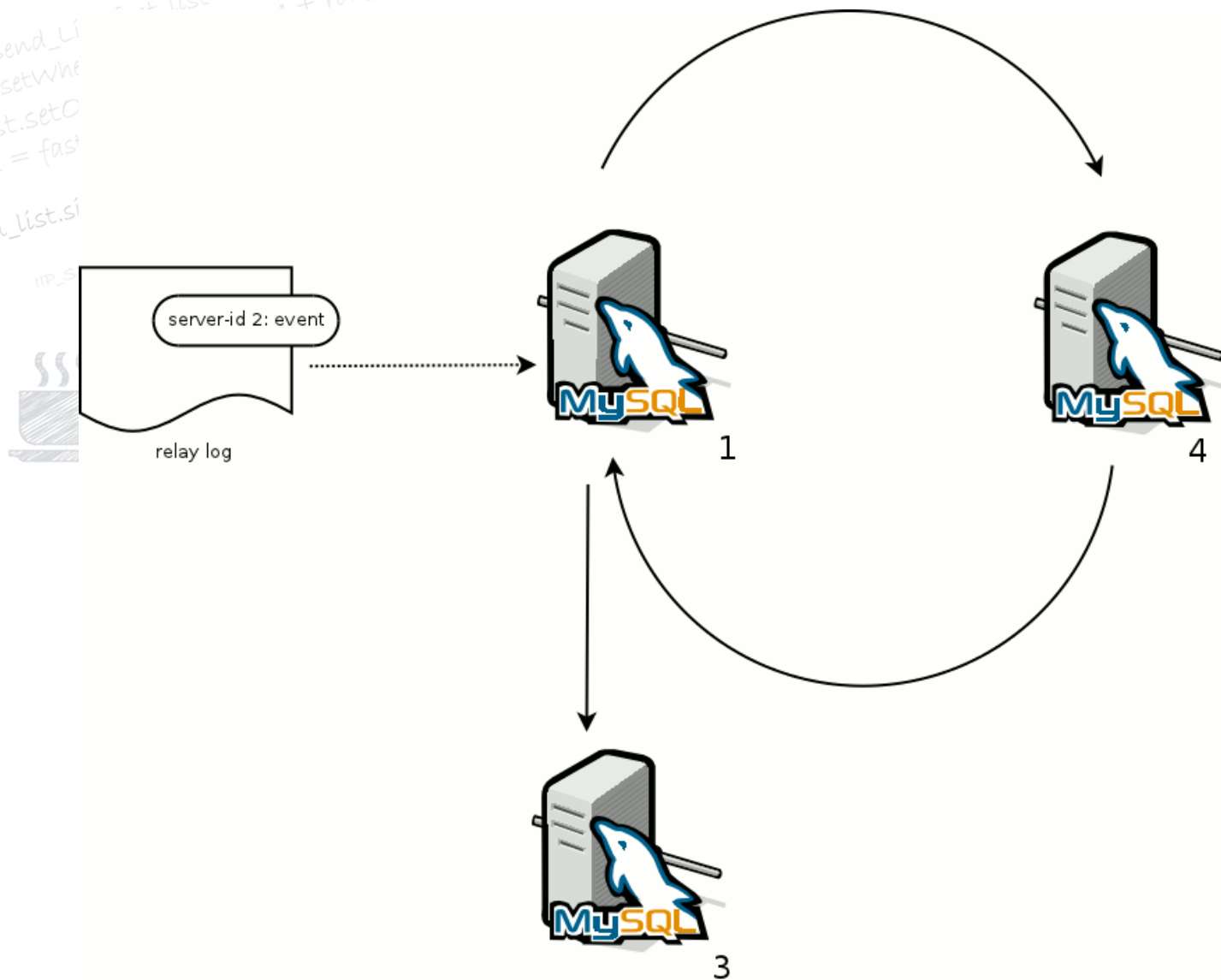
$\text{Seconds\_Behind\_Master} = A - B$

# 案例七 Slave 事件Loop

```
can_is_empty = false;  
wayList<IP_SM_Send  
... List();  
... + row_count);
```



# 案例七 Slave 事件Loop





# 案例七 Slave 事件Loop

- 简单重现的办法

1.配置server1和server2为双主结构:

server1(server\_id=1) <-(MM)-> server2(server\_id=1)

2、在server1执行以下SQL命令:

```
mysql> create table test.t1(id int);
```

```
mysql> stop slave; insert into t1 values(1);
```

```
mysql> set global server_id=3;
```

```
mysql> start slave;
```

3、在server1, server2上执行:

```
mysql> select count(*) from t1;
```

# 案例七 Slave 事件Loop

- 解决办法
- Change Master to ....IGNORE\_SERVER\_IDS
- 停掉主库的写,在备库寻找最后一个正确的 position ,change master to
- 如果修改过server\_id,把server\_id重新改回来
- 更详细信息 Taobaodba.com [MySQL复制事件在主备之间来回传输检测](#)

# 案例八 Slave 找不到对应记录

[ERROR] Slave SQL: Could not execute

Update\_rows event on table tbtry.try\_audit;

Can't find record in 'xxx\_table', Error\_code: 1032;

handler error HA\_ERR\_KEY\_NOT\_FOUND; the

event's master log mysql-bin.002403,

end\_log\_pos 67019815, Error\_code: 1032

# 案例八 Slave 找不到对应记录

[ERROR] Slave SQL: Could not execute  
Update\_rows event on table tbtry.try\_audit;  
Can't find record in 'xxx\_table', Error\_code: 1032;  
handler error HA\_ERR\_KEY\_NOT\_FOUND; the  
event's master log mysql-bin.002403,  
end\_log\_pos 67019815, Error\_code: 1032

binlog\_format='ROW' ,binlog里面记录每个字段  
的具体值,  
通过slave\_error\_handler进行检测回补

# 案例八 Slave 找不到对应记录



362 关注   477 粉丝   441 微博

小希\_TB\_印风 (印风) LV 7 <http://weibo.com/zhaiwx1987>

男 · 湖北 武汉 · 毕业于 武汉大学

标签: Linux · PostgreSQL · MySQL · 计算机 · Database · c语言

简介: 每一个早晨都充满了阳光的味道~~~~猪, 你又睡到太阳晒屁股了!! ... [更多资料](#)

[互相关注 | 取消](#)   [私信](#)

他的主页   个人资料   关注/粉丝   相册

全部   微博

全部   原创   图片   视频   音乐   标签   心情

搜索他说的话

[动态隐私设置](#)

Facebook MySQL Rev3444: 可以配置主备数据传输的压缩级别net\_compression\_level, 默认zlib的压缩级别为6, 从1-9分别为更快的压缩--->更好的压缩效果. 如果CPU足够牛X, 可以将其调大, 来减少网络传输

30分钟前 来自新浪微博

[转发](#) | [收藏](#) | [评论](#)

微关系 我们之间

共同关注 (78) —

我关注的人也关注



# 案例九 备库设置read\_only被堵塞

\*\*\*\*\* 13. row \*\*\*\*\*

Id: 328071

User: root

Host: localhost

db: NULL

Command: Query

Time: 81

State: Waiting for table flush

Info: set global read\_only=1

Rows\_sent: 0

Rows\_examined: 0

Rows\_read: 1



# 案例九 备库设置read\_only被堵塞

\*\*\*\*\* 13. row \*\*\*\*\*

Id: 328071

User: root

Host: localhost

db: NULL

Command: Query

Time: 81

State: Waiting for table flush

Info: set global read\_only=1

Rows\_sent: 0

Rows\_examined: 0

Rows\_read: 1

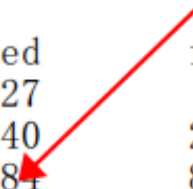
备库正在执行mysqldump

# 十 线上数据库出现swap的案例

- 前端请求响应变慢
- DB 负载上升
- free -m 后发现 开始使用Swap

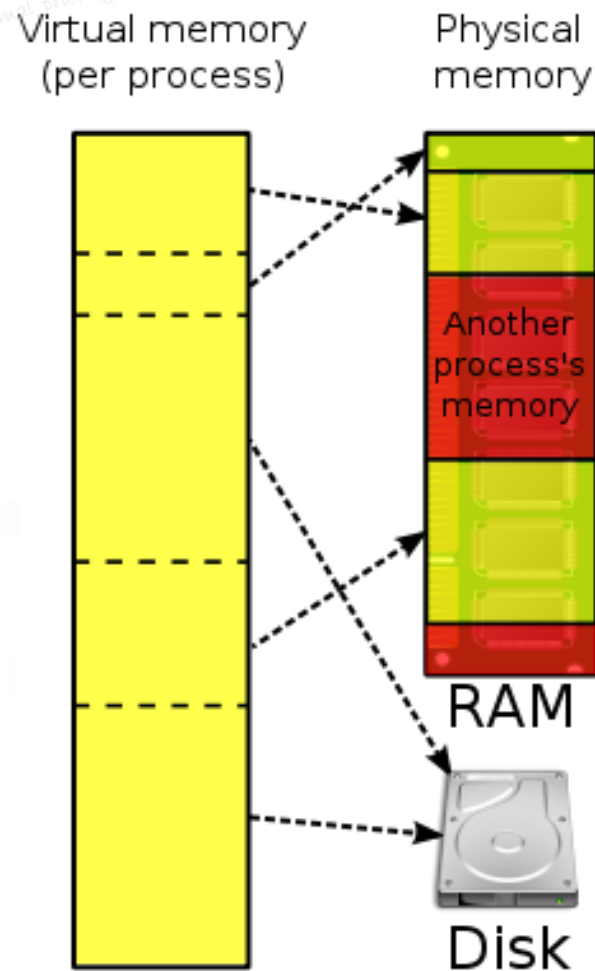
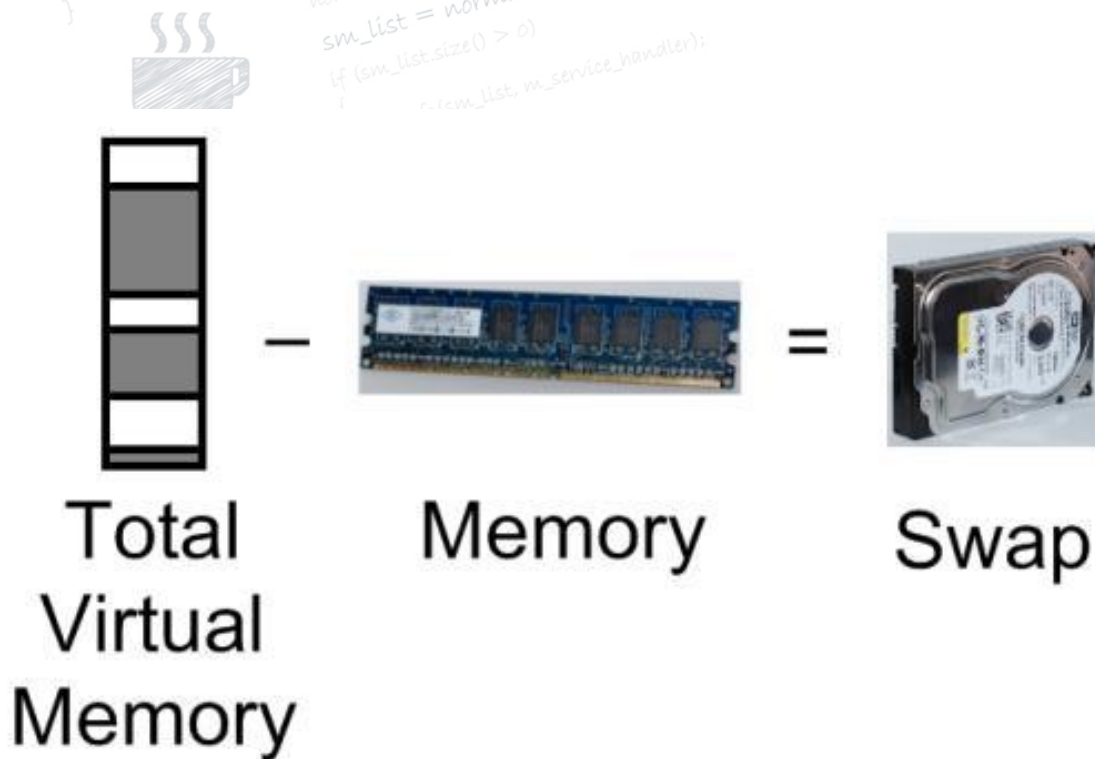
```
$free -m
```

	total	used	free	shared	buffers	cached
Mem:	7800	7727	72	0	277	2209
-/+ buffers/cache:		5240	2559			
Swap:	8189	81	8104			



# 十 线上数据库出现swap的案例

## 操作系统设置swap的目的



ean  
An  
117  
f

mainDty = false;  
...->sm\_list;  
cat();

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

- <http://www.linux-mag.com/id/7589/>
- <http://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf>

# 十 线上数据库出现swap的案例

- Flashcache压测，Flashcache元数据本身需要内存
- 多实例的主备库混搭
  - mysqldump -q (Don't buffer query, dump directly to stdout)
- /proc/sys/vm/swappiness和/etc/sysctl.conf不一致
- table\_open\_cache 和table\_definition\_cache过大

# 谢谢大家

```
ean is_empty = false;  
ArrayList<IP_SM_Send_List> sm_list;
```

```
IP_SM_Send_List fast_list = new IP_SM_Send_List();  
fast_list.setWhere("rownum <= " + row_count);  
fast_list.setOrderBy("IP_ID");  
sm_list = fast_list.select(tlConn);
```

```
if (sm_list.size() > 0)  
{  
    IP_SM_Send_List normal_list = new IP_SM_Send_List();  
    normal_list.setWhere("IP_ServiceID in (" + m_service_handler.getServiceList(m_int_normal_priority) + ")");  
    normal_list.setWhere("rownum <= " + row_count);  
    normal_list.setOrderBy("IP_ID");  
    sm_list = normal_list.select(tlConn);  
    if (sm_list.size() > 0)  
    {  
        certInfo(sm_list, m_service_handler);  
    }  
}
```



- 新浪微博@杨德华Devin