29 | 如何判断一个数据库是不是出问题了

25和27介绍了主备切换流程，

讨论出主题：**怎么判断一个主库出了问题？**

**Select 1判断**

**实际上，select 1成功返回，只能说明这个库的进程还存在，并不能说明主库没问题，**

set global innodb\_thread\_concurrency=3;

CREATE TABLE `t` (

`id` int(11) NOT NULL,

`c` int(11) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB;

insert into t values(1,1)

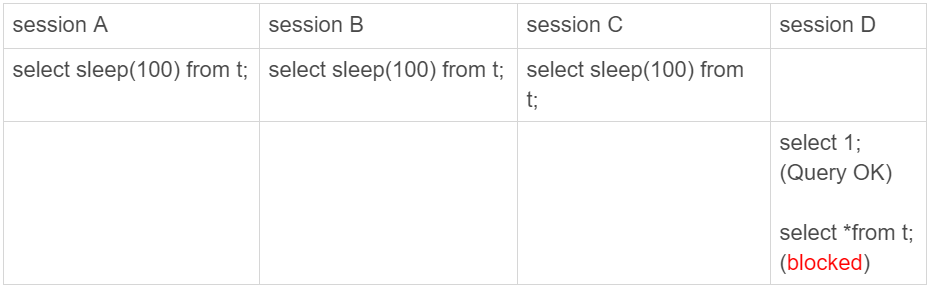


图 1 查询blocked

设置innodb\_thread\_concurrency参数目的是，控制InnoDB的并发线程上线，一旦并发线程达到这个值，InnoDB在接收心情求时，就会进去等在状态，直到有线程退出。

Innodb\_thread\_concurrency设置为3，表示innoDB只允许3个线程并行，前三个session中sleep(100)，使得这三个语句都处于“执行”状态，模拟大查询

SessionD里面，select 1是能执行成功的，但是查询表t的语句会被堵住，证明，用select 1 来测实例是否能正常的，是不能测出问题的

InnoDB中，innodb\_thread\_concurrency这个参数的默认值是0，表示不限制并发线程数量。但是，不限制并发线程数是不行的，因为一个CPU核数有限，线程全冲进来，上下文切换的成本太高。

通常情况下，建议把innodb\_thread\_concurrency设置为64~128之间的值，此时，并发线程上限为什么设置为128，线上的并发连接数动不动就是上千。

产生这个疑问的原因，是搞混了**并发连接和并发查询**

并发连接和并发查询，不是一个概念，在show processlist的结果里，看到几千个连接，指的是并发连接，当“当前正在执行”的语句，才是我们所说的并发查询。

并发连接数达到几千影响并不大，就是多占一些内存而已，应该关注的的并发查询，因为并发查询太高才是CPU杀手，这也是为什么设置innodb\_thread\_concurrency参数的原因。

在第7篇讲到热点更新和死锁检测时，如果把innodb\_thread\_concurrency设置为128的话，那么出现同一行热点更新的问题时，是不是很快就把128消耗完了呢？整个系统是不是就挂了？

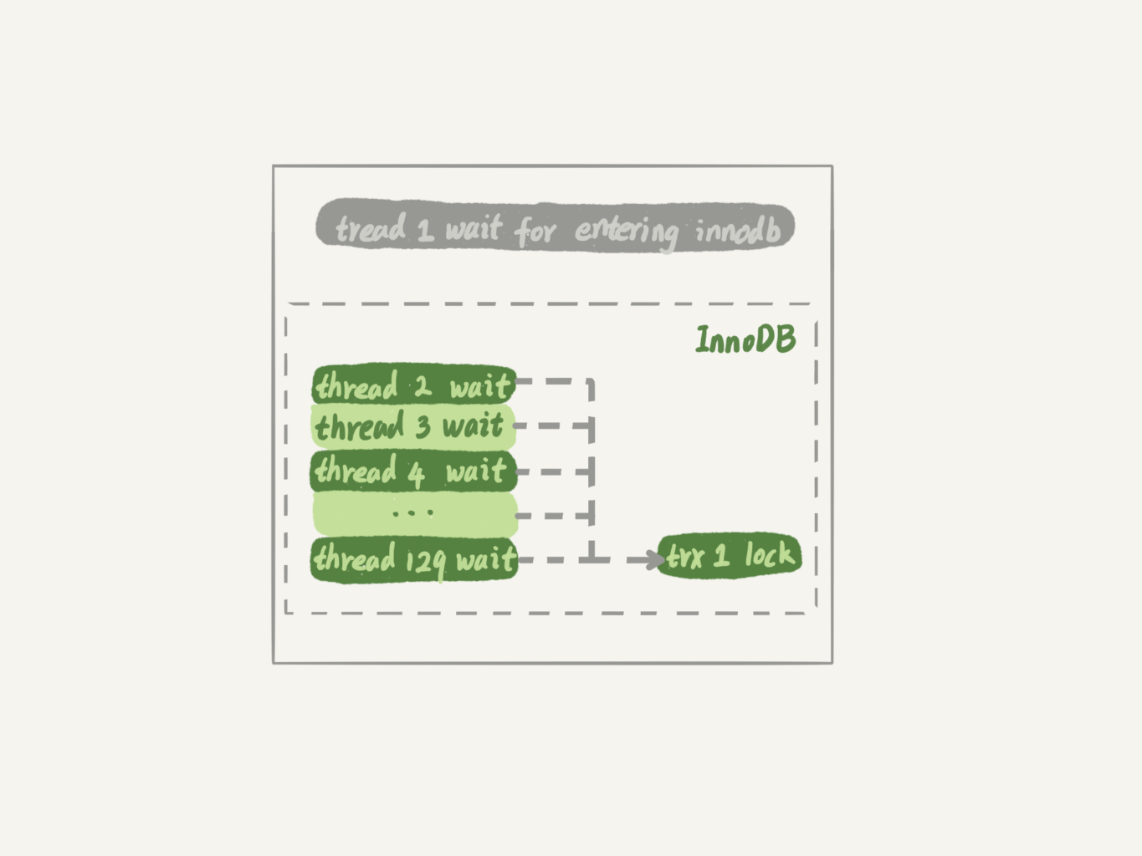
**在线程进入锁等待后，并发线程数的计数会减一，**也就是说等行锁（也包括间隙锁）的线程是不算在128里面的

因为进入所等待的线程已经不吃CPU了，更重要的是，必须这么设计，才能避免整个系统锁死

假设处于所等待的线程也占并发线程的计数，可以设想一下整个场景

1. 线程 1 执行begin;update t set c=c+1 where id = 1，启动了事务trx1,然后保持整个状态，此时，线程处于空闲状态，不算在并发线程里面
2. 线程2到线程129都执行update t set c=c+1 where id = 1;由于等行锁，进入等待状态，这样就有128个线程处于等待状态；
3. 如果处于锁等待状态的线程计数不减一，InnoDB会认为线程数用满了，会阻止其他语句进入引擎执行，这样线程1不能提交事务，而另外128个线程有处于锁等待状态，整个系统就堵住了。

图2 显示这个状态

 图 2 系统锁死状态（假设等行锁语句占用并发计数）

此时InnoDB不能响应任何请求，整个系统锁死，而且由于所有的线程都处于等待状态，此时占用的CPU是0，而这个明显不合理，所以，InnoDB设计时，遇到线程进入所等待的情况，将并发线程的计数减1的设计，是合理有必要的

虽说等所的线程不算在线程并发计数里，但如果它在真正的执行查询，就比如上面的例子前前三个事务中的select sleep(100) from t,还是要算进并发线程的技术的

在这个例子中，同时执行的语句超过了设置innodb\_thread\_concurrency的值，此时系统已经不行了，但是通过select 1来检测系统，会认为系统时正常的

因此，用select 1的判断逻辑要修改一下

**查表判断**

为了检测InnoDB并发线程数过多导致系统的不可用，需要一个访问InnoDB的场景，一般的做法是：在系统里创建一个表，比如命名为health\_check,里面只放一行数据，然后 定期执行

mysql> select \* from mysql.health\_check;

使用这个方法，可以检测出处于并发线程过多导致的数据库不可用的情况。

但是，会碰到下一个问题，即：空间满了，这种方法又变得不好使了。

更新事务要写binlog,而一旦binlog所在磁盘空间占用率到达100%.那么所有的更新语句和事务提交的commit都会被堵住，但是，系统此时还是可以正常读数据的。

可以把这条监控语句在修改一下，

**更新判断**

常见的做法是放一个timestamp字段，用来表示最后一次执行检测的时间

mysql> update mysql.health\_check set t\_modified=now();

节点可用性的检测都应该包含主库和备库，如果用更新来检测主库的话，那么备库也要进行更新检测。

但，备库的检测也是要写binlog的，由于一般把数据库A和B的主备关系设计为双M结构，所以在备库B上执行的检测命令，也要发回个主库A.

但是，如果主库A和备库B都用相同的命令，就可能出现行冲突，也就是可能会导致主备同步停止，所以，现在看来，mysql.health\_check这个表不能只有一行数据了。

为了让主备之间的更新不产生冲突，可以在mysql.health\_check表存入多行数据，并用A、B的server\_id做主键

mysql> CREATE TABLE `health\_check` (

`id` int(11) NOT NULL,

`t\_modified` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (`id`)

) ENGINE=InnoDB;

/\* 检测命令 \*/

insert into mysql.health\_check(id, t\_modified) values (@@server\_id, now()) on duplicate key update t\_modified=now();

由于MySQL 规定了主库和备库的server\_id必须不同（否则创建主备关系的时候就会报错），这样可以保证主、备库各自的检测命令不发生冲突。

更新判断是一个相对比较常用的方案，不过依然存在一些问题，其中“判断慢”一直是让DBA头疼的问题。

疑问：**更新语句，如果失败或超时，就可以发起主备切换了，为什么还会有判定慢的问题呢？**

其实，这设计的服务器的IO分配资源问题

首先，所有的检测逻辑都需要一个超时时间N,执行一条update语句，超过N秒后还不返回，就认为系统不可用。

设想一个日志盘的IO利用率已经是100%的场景，此时，整个系统响应非常慢，已经需要做主备切换了。

IO 利用率100%表示系统的IO是在工作的，每个请求都有机会获得IO资源，执行自己的任务，而检测使用update命令，需要的资源很少，所以可能在拿到IO资源的时候就可以提交成功，并且在超时时间N秒未到达之前就返回给了检测系统。

检测系统看到update命令没有超时，于是就得到了“系统正常”的结论

上面的饿方法都是基于外部检测的。外部检测天然有个问题，就是随机性

因为外部检测需要轮询的，系统可能已经出问题了，但是需要等到下一个检测发起执行语句的时候，才坑你发现问题，

**内部统计**

针对磁盘利用率这个问题，如果MySQL可以告诉我们，内部每一次IO请求的时间，判断数据库是否出问题的方法就可靠多了。

MySQL5.6 提供了performance\_schema库，就在file\_summaer\_by\_event\_name表里统计了每次IO请求的时间

File\_summer\_by\_event\_name表里很多行数据，

Enent\_name=’wait/io/file/innodb/innodb\_log\_file’这一行

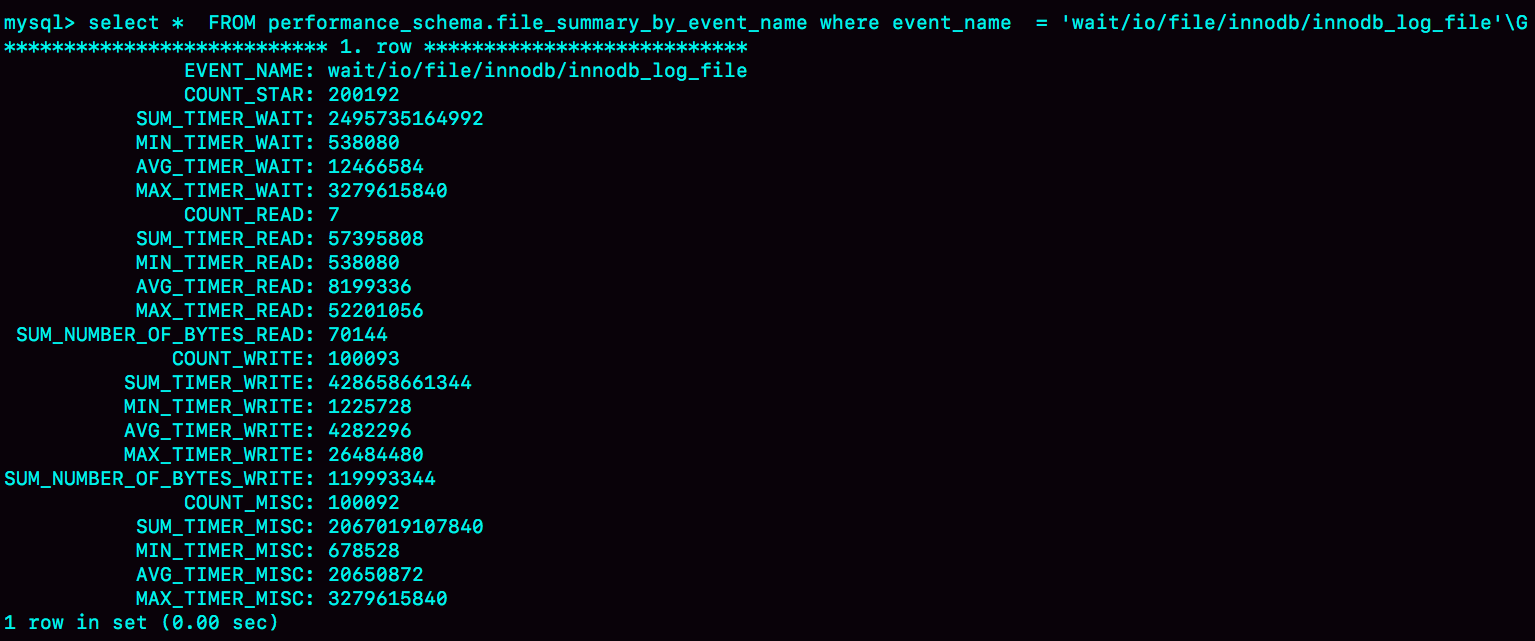


图3 performance\_schema.file\_summary\_by\_event\_name的一行

图中这一行表示统计的是redo log的写入时间，第一列EVENT\_NAME表是统计的类型

接下来的三组数据，显示的是redo log操作的时间统计

第一组五列，是所有IO类型的统计，其中，COUNT\_START是所有IO的总次数，接下来四列是具体的统计项，单位是皮秒:前缀SUM,MIN,AVG,MAX顾名思义就是总和，最小，平局值和最大值

第二组六列，统计的是写操纵

最后的第四组数据，是对其他类型的数据的统计，在redo log里，可以认为他们是对fsync的统计

在performance\_schema库的file\_summary\_by\_event\_name表里，binlog对应的是event\_name=’wait/io/file/sql/binlog’这一行，各个字段的统计逻辑，与redo log的各个字段完全相同

每一次操作用数据库，performance\_schema都需要额外的统计这些数据，打开这个统计功能是有性能能损耗的。如果打开performance\_schema项，性能大概会下降10%，可以通过下面的方法打开户关闭某个具体想的统计

如果想打开redo log的时间监控，

mysql> update setup\_instruments set ENABLED='YES', Timed='YES' where name like '%wait/io/file/innodb/innodb\_log\_file%';

假设，开启了redo log和binlog这两个统计信息，怎么把这个信息用在实例状态诊断上？

通过MAX\_TIMER的值来判断数据库是否出了问题，比如，可以设定阈值，单次IO请求时间超过200毫秒属于异常，然后使用类似下面这条语句作为检测逻辑

mysql> select event\_name,MAX\_TIMER\_WAIT FROM performance\_schema.file\_summary\_by\_event\_name where event\_name in ('wait/io/file/innodb/innodb\_log\_file','wait/io/file/sql/binlog') and MAX\_TIMER\_WAIT>200\*1000000000;

发现异常后，取到需要的信息，再通过这条语句

mysql> truncate table performance\_schema.file\_summary\_by\_event\_name;

把之前的统计信息清空，这样如果后面的监控中，再次出现异常，可以加入监控积累值