

阿里巴巴在线技术峰会
Alibaba Online Technology Summit

云数据库十大经典案例

ApsaraDB专家服务组
玄慚



自我介绍

- 玄惭出自天龙八部
- 2010年加入淘宝DBA
- 2012年加入阿里云RDS
- 负责RDS线上的稳定
- 历年RDS双11的负责人
- RDS专家服务组负责人
- 博客：hidba.org



目录

案例一 索引

案例二 SQL优化

案例三 锁

案例四 延迟

案例五 参数优化

案例六 conn 100%

案例七 cpu 100%

案例八 iops 100%

案例九 disk 100%

案例十 mem 100%

一、案例一 索引

案例一 无索引案例

问题描述：用户系统打开缓慢，数据库 CPU 100%

问题排查：发现数据库中大量的慢sql，执行时间超过了2S

慢SQL：SELECT uid FROM `user` WHERE mo=13772556391 LIMIT 0,1;

执行计划

```
mysql> explain SELECT uid FROM `user` WHERE mo=13772556391 LIMIT 0,1;
```

```
*****  
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: user
```

```
type: ALL
```

```
possible_keys: NULL
```

```
key: NULL
```

```
key_len: NULL
```

```
ref: NULL
```

```
rows: 707250
```

```
Extra: Using where
```

执行时间

```
mysql> SELECT uid FROM `user` WHERE mo=13772556391 LIMIT 0,1;
```

```
Empty set (2.11 sec)
```

案例一 无索引案例

表结构

```
CREATE TABLE `user` (  
  `uid` int(11) unsigned NOT NULL AUTO_INCREMENT COMMENT 'id',  
  `pid` int(11) unsigned NOT NULL DEFAULT '0' ,  
  `email` char(60) NOT NULL ,  
  `name` char(32) NOT NULL DEFAULT "",  
  `mo` char(11) NOT NULL DEFAULT "",  
  PRIMARY KEY (`uid`),  
  UNIQUE KEY `email` (`email`),  
  KEY `pid` (`pid`)  
) ENGINE=InnoDB ENGINE=InnoDB AUTO_INCREMENT=972600 DEFAULT  
CHARSET=utf8;
```


案例一 无索引案例

验证mo字段的过滤性

```
mysql> select count(*) from user where mo=13772556391;
```

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

```
| count(*) |
```

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

```
| 0 |
```

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

```
1 row in set (0.05 sec)
```

添加索引

```
mysql> alter table user add index ind_mo(mo);
```

执行时间

```
mysql> SELECT uid FROM `user` WHERE mo=13772556391 LIMIT 0,1;
```

```
Empty set (0.05 sec)
```

案例一 无索引案例

执行计划

```
mysql> explain SELECT uid FROM `user` WHERE mo=13772556391 LIMIT 0,1\G;
```

```
***** 1. row *****
```

id: 1

select_type: SIMPLE

table: user

type: index

possible_keys: ind_mo

key: ind_mo

key_len: 33

ref: NULL

rows: 707250

Extra: Using where; Using index

案例一 隐式转换案例

为什么索引的过滤性这么差？

```
mysql> explain extended select uid from `user` where mo=13772556391 limit 0,1;
mysql> show warnings;
Warning1 : Cannot use index 'ind_mo' due to type or collation conversion on field 'mo'
Note : select `user`.`uid` AS `uid` from `user` where (`user`.`mo` = 13772556391) limit 0,1
```

表结构

```
CREATE TABLE `user` (
    .....
    `mo` char(11) NOT NULL DEFAULT "",
    .....
) ENGINE=InnoDB;
```

案例一 隐式转换案例

```
mysql> explain SELECT uid FROM `user` WHERE mo='13772556391' LIMIT 0,1\G;
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: user
```

```
type: ref
```

```
possible_keys: ind_mo
```

```
key: ind_mo
```

```
key_len: 33
```

```
ref: const
```

```
rows: 1
```

```
Extra: Using where; Using index
```

```
mysql> SELECT uid FROM `user` WHERE mo='13772556391' LIMIT 0,1;
```

```
Empty set (0.00 sec)
```

索引最佳实践

1

通过explain查看sql的执行计划

判断是否使用到了索引以及隐士转换

2

常见的隐式转换

包括字段数据类型类型以及字符集定义不当导致

3

设计开发阶段

避免数据库字段定义与应用程序参数定义出现不一致
不支持函数索引,避免在查询条件加入函数:date(a.gmt_create)

4

SQL审核

所有上线的sql都要经过严格的审核, 创建合适的索引

二、案例二 SQL优化

案例二 分页优化案例



```
| Query | 51 | Sending data |  
select *  
from buyer  
where sellerId = 765922982  
and gmt_modified >= '1970-01-01 08:00:00'  
and gmt_modified <= '2013-06-05 17:11:31'  
Limit 255000, 5000 ;
```

案例二 分页优化案例

普通写法：

```
select * from buyer where sellerid=100 limit 100000 , 5000
```

普通limit M , N的翻页写法，在越往后翻页的过程中速度越慢，原因mysql会读取表中的前M+N条数据，**M越大，性能就越差。**

优化写法：

```
select t1.* from buyer t1,  
       (select id from buyer sellerid=100 limit 100000 , 5000) t2  
where t1.id=t2.id;
```

注意：需要在t表的sellerid字段上创建索引，id为表的主键
create index ind_sellerid on buyer(sellerid);

案例二 分页优化案例

原始语句：

```
select id, ...  
      from buyer  
     where sellerId = 765922982  
        and gmt_modified >= '1970-01-01 08:00:00'  
        and gmt_modified <= '2013-06-05 17:11:31'  
 limit 255000, 5000 ;
```

优化后语句：

```
select t2.*  
      from (select id  
            from buyer where sellerId = 765922982  
           and gmt_modified >= '1970-01-01 08:00:00'  
           and gmt_modified <= '2013-06-05 17:11:31'  
           limit 255000, 5000 ) t1,  
 buyer t2  
     where t1.id=t2.id
```

查询时间：60S→0.2S

案例二 子查询优化案例

典型子查询

```
SELECT first_name  
FROM employees  
WHERE emp_no IN  
(SELECT emp_no FROM salaries_2000 WHERE salary = 5000);
```

MySQL的处理逻辑是遍历employees表中的每一条记录，代入到子查询中去

改写子查询

```
SELECT first_name  
FROM employees emp,  
(SELECT emp_no FROM salaries_2000 WHERE salary = 5000) sal  
WHERE emp.emp_no = sal.emp_no;
```

执行时间 : 1200S → 0.1S

SQL优化最佳实践

1

分页优化

采用高效的 Limit 写法，避免分页查询给数据库带来性能影响

2

子查询优化

子查询在5.1，5.5版本中都存在较大风险，将子查询改为关联使用Mysql 5.6的版本，可以避免麻烦的子查询改写

3

查询需要的字段

避免用 SELECT * 查询所有字段数据，只查询需要的字段数据

三、案例三 锁

案例三 表级锁

InnoDB与MyISAM

引擎	支持事务	并发	索引损坏	锁级别	在线备份
MyISAM	不支持	查询堵塞更新	索引损坏	表	不支持
InnoDB	支持	不堵塞	不损坏	行	支持

案例三 表级锁

```
CREATE TABLE `t_myisam` (  
  `id` int(11) DEFAULT NULL  
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ;
```

查询堵塞更新

```
Query |111 | User sleep | select id,sleep(100) from t_myisam |  
Query |108 | Waiting for table level lock | update t_myisam set id=2 where id=1|  
Query | 3 | Waiting for table level lock | update t_myisam set id=2 where id=1|
```

解决：

```
Alter table t_myisam engine=innodb;
```

案例三 Metadata lock

DDL操作：alter table t add column gmt_create datetime

数据库连接状态：

Query |6 | User sleep | select id ,sleep(50) from t

Query |4 | Waiting for table metadata lock | alter table t add column gmt_create datetime

Query |2 | Waiting for table metadata lock | select * from t where id=1

Query |1 | Waiting for table metadata lock | select * from t where id=2

Query |1 | Waiting for table metadata lock | update t set id =2 where id=1

Tips：DDL过程中注意数据库中大长事务，大查询

锁问题最佳实践

1

设计开发阶段

- 1.避免使用myisam存储引擎，改用innodb引擎
- 2.避免大事务，长事务导致事务在数据库中的运行时间加长
- 3.选择升级到MySQL5.6版本，支持online ddl

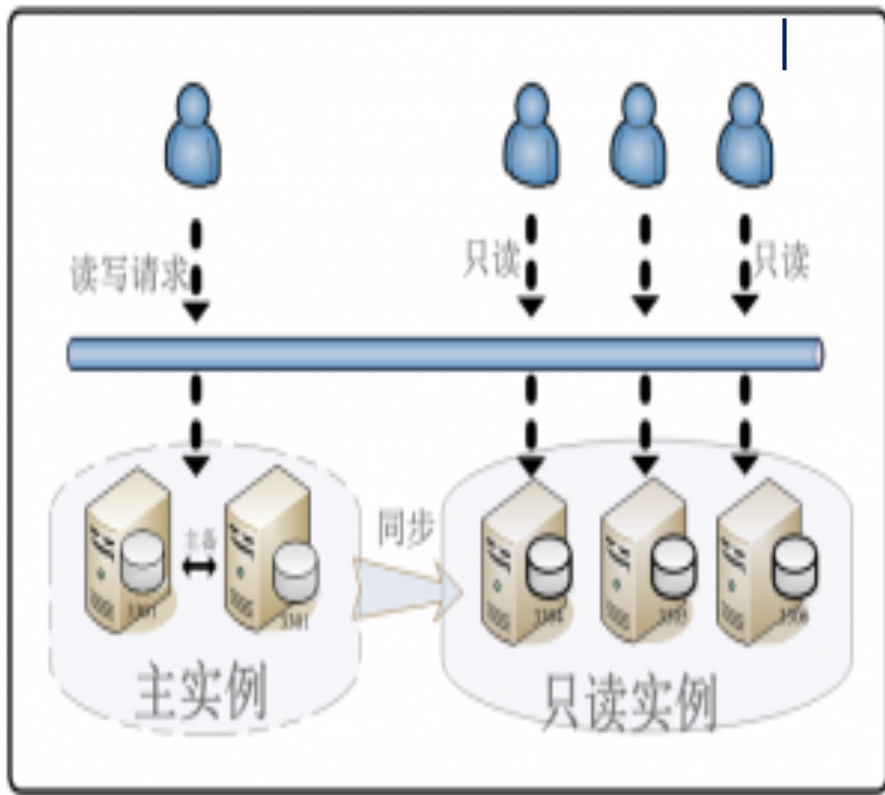
2

管理运维阶段

- 1.在业务低峰期执行上述操作，比如创建索引，添加字段；
- 2.在结构变更前，观察数据库中是否存在长SQL，大事务；
- 3.结构变更期间，监控数据库的线程状态是否存在lock wait；
- 4.ApsaraDB支持在DDL变更中加入 wait timeout；

四、案例四 延迟

案例四 只读实例架构



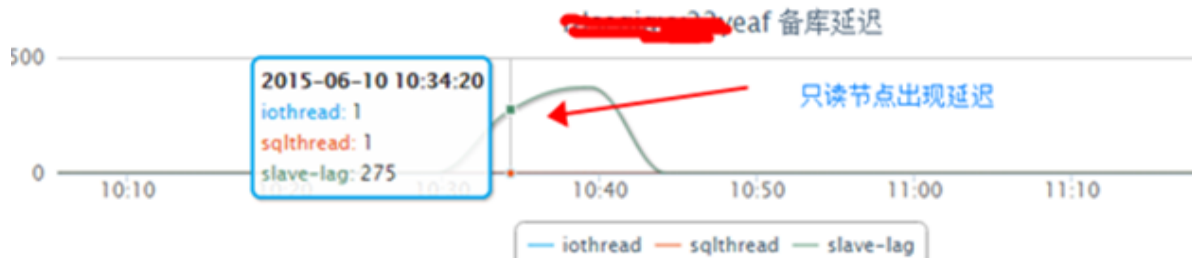
数据库需要升级到5.6版本

最多支持5个节点

采用MySQL复制原生实现数据同步

案例四 DDL导致延迟

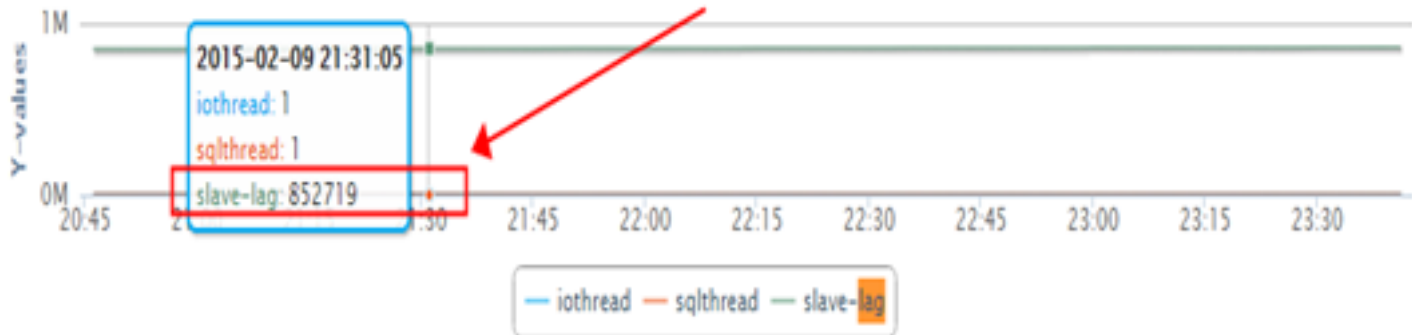
常见DDL : create index , repair , optimze table , alter table add column



SQL语句	执行时间	数据库	用户名	客户端IP	消耗时间(微秒)
ALTER TABLE [redacted] ADD INDEX IDX_MESSAGEID (MESSAGEID)	2015-06-10 10:27:18	[redacted]	mb	[redacted]	212693752
ALTER TABLE [redacted] ADD KEY 'messageid'('messageID')	2015-06-10 10:29:32	[redacted]	openfire	[redacted]	217252303

大事务 : create ..as select , insert...select , load...data , delete...from , update ..from

案例四 MDL锁导致延迟



[REDACTED]	[REDACTED]	Query	881025	Sending data	SELECT T_LON.CUSTOMER_NAME, T_LON.CERT_ID, ...
[REDACTED]	[REDACTED]	Query	458920	Waiting for table metadata lock	SELECT COUNT(1) FROM T_LON_APPLICATION AS ...

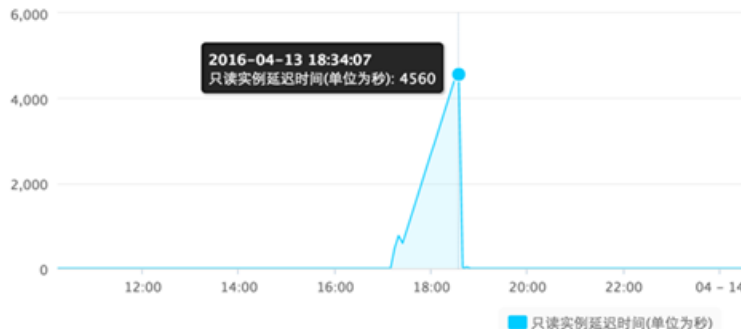
Tips

通过执行show processlist 查看连接的状态；
锁会阻塞复制线程导致复制延迟；

案例四 资源问题导致延迟



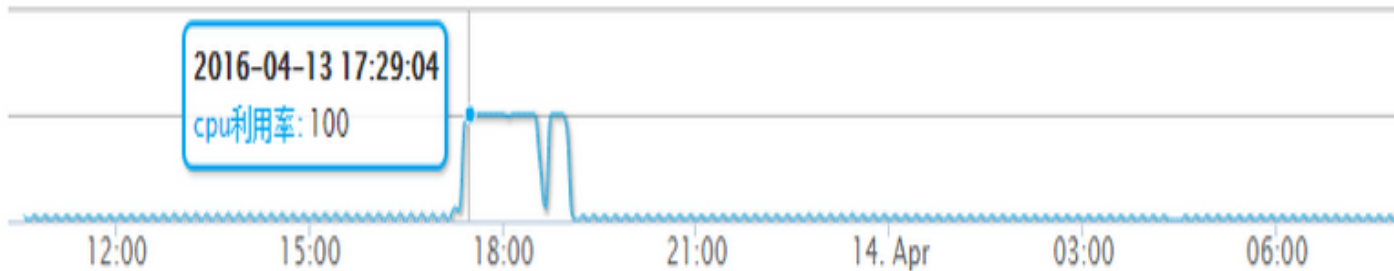
只读实例延迟时间(单位为秒)



Tips

压力：同步压力+只读业务压力

效率：CPU+IOPS资源



延迟问题最佳实践

1

排查思路

1. 一看资源是否达到瓶颈；
2. 二看线程状态是否有锁；
3. 三判断是否存在大事务；

2

最佳实践

1. 使用innodb存储引擎；
2. 只读实例的规格不低于主实例；
3. 大事务拆分为小事务；
4. DDL变更期间观察是否有大查询；

五、案例五 参数优化

案例五 一个参数引发的血案

背景介绍：

某客户正在将本地的业务系统迁移上云

在rds上运行时间明显要比线下自建数据库运行时间要慢1倍

导致客户系统割接延期的风险

关键词：

上云，RDS，自建，慢1倍

案例五 一个参数引发的血案

经验分析：

- 1、 数据库跨平台迁移 (PG->MySQL、ORALCE->MySQL)
- 2、 跨版本升级(MySQL:5.1->5.5、5.5->5.6)
- 3、 执行计划，优化器，参数配置，硬件配置

案例五 一个参数引发的血案

确定优化器版本：用户5.6，RDS的版本5.6

OPTIMIZER_SWITCH:

```
index_merge=on,index_merge_union=on,index_merge_sort_union=on,  
index_merge_intersection=on,engine_condition_pushdown=on,  
index_condition_pushdown=on,mrr=on,mrr_cost_based=on,  
block_nested_loop=on,batched_key_access=off,materialization=on,  
semijoin=on,loosescan=on,firstmatch=on,  
subquery_materialization_cost_based=on,use_index_extensions=on
```

案例五 一个参数引发的血案

确定SQL执行计划：rows=39900*1*1*140*285*1*1*1*1*1*1*1

key_len	ref	rows	Extra
NULL	NULL	39900	Using where
5	v.ID	1	Using where
5	v.ID	1	Using where
768	NULL	140	Using index
5	wdw.s.CID	285	Using where
4	wdw.ub.UID_CUS	1	NULL
5	wdw.ub.UID	1	Using index condition
5	wdw.up.ID	1	Using where
4	wdw.bg.BID	1	Using where
4	wdw.ub.UID	1	NULL
4	wdw.ub.UID	1	NULL
4	wdw.B.UID	1	Using where

案例五 一个参数引发的血案

确定参数配置

用户配置：

```
join_buffer_size = 128M  
read_rnd_buffer_size = 128M  
tmp_table_size = 128M
```

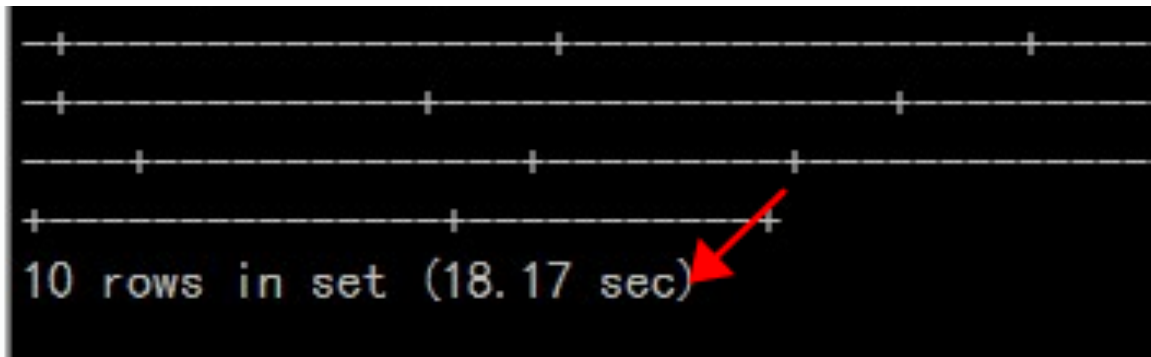
RDS配置

```
join_buffer_size = 1M  
read_buffer_size = 1M  
tmp_table_size = 256K
```

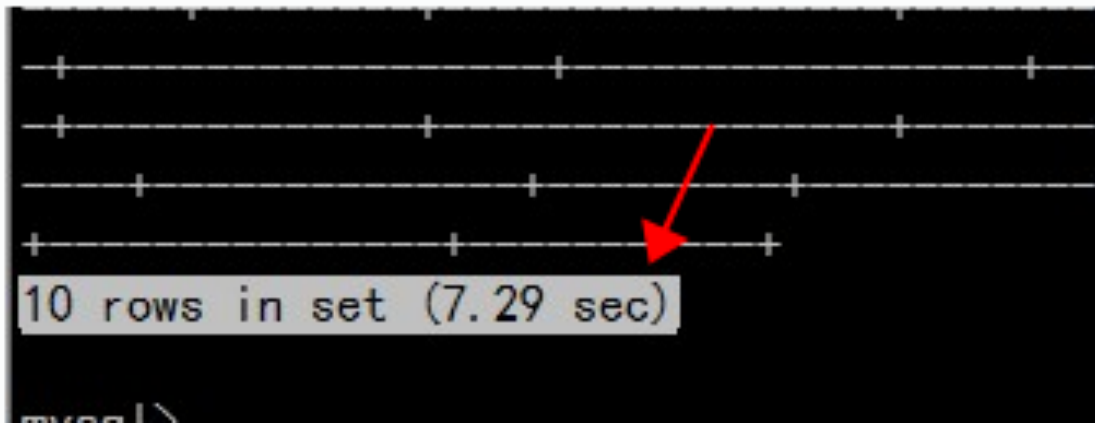
案例五 一个参数引发的血案

验证阶段：tmp_table_size由256K调整至128MB

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
10 rows in set (18.17 sec)
```



```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
10 rows in set (7.29 sec)
```



参数最佳实践

1

排查思路

- 1.查看SQL执行计划；
- 2.查看数据库版本和优化器规则；
- 3.对比参数设置；
- 4.对比硬件配置；

2

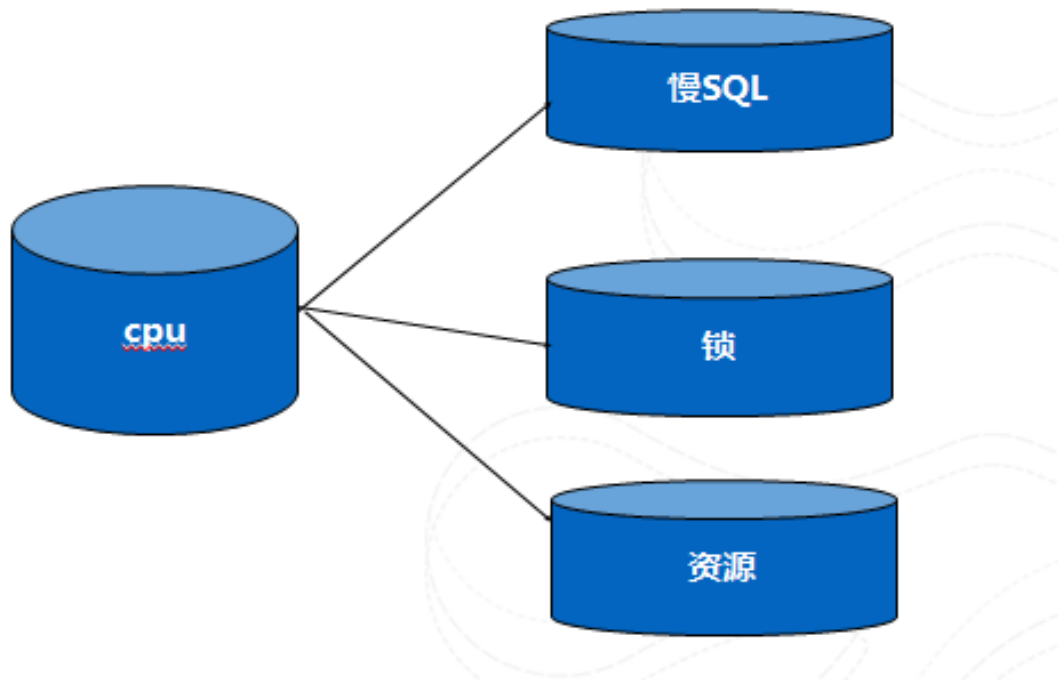
最佳实践

- 1.Query_cache_size；
- 2.Temp_table_size
- 3.Tokudb_buffer_pool_ratio
- 4.Back_log

六、案例六 cpu 100%

案例六 cpu 100%

三大因素：慢SQL，锁，资源



Cpu 100%最佳实践

1

慢SQL问题

1.通过优化索引，子查询，隐士转换，分页改写等优化；

2

锁等待问题

1.通过设计开发和管理运维优化锁等待；

3

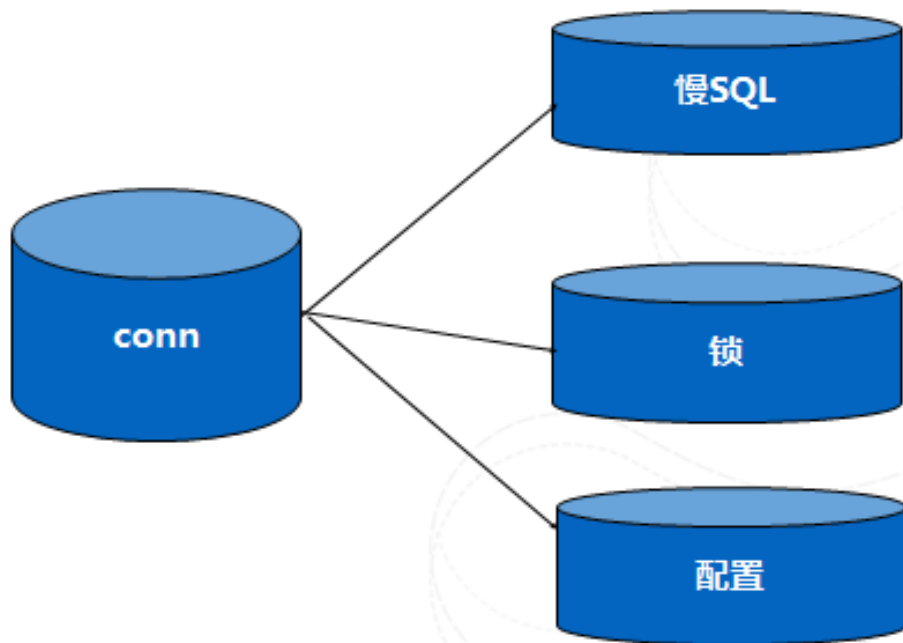
资源问题

1.通过参数优化，弹性升级，读写分离，数据库拆分等方式优化；

七、案例七 conn 100%

案例七 conn 100%

三大因素：慢SQL，锁，配置



Conn 100%最佳实践

1

慢SQL问题

1.通过优化索引，子查询，隐士转换，分页改写等优化；

2

锁等待问题

1.通过设计开发和管理运维优化锁等待；

3

配置问题

1.客户端连接池参数配置超出实例最大连接数；
2.弹性升级RDS的规格配置；

八、案例八 iops 100%

Iops 100%最佳实践

1

慢SQL问题

1.通过优化索引，子查询，隐士转换，分页改写等优化；

2

DDL

1. create index , optimze table , alter table add column ;

3

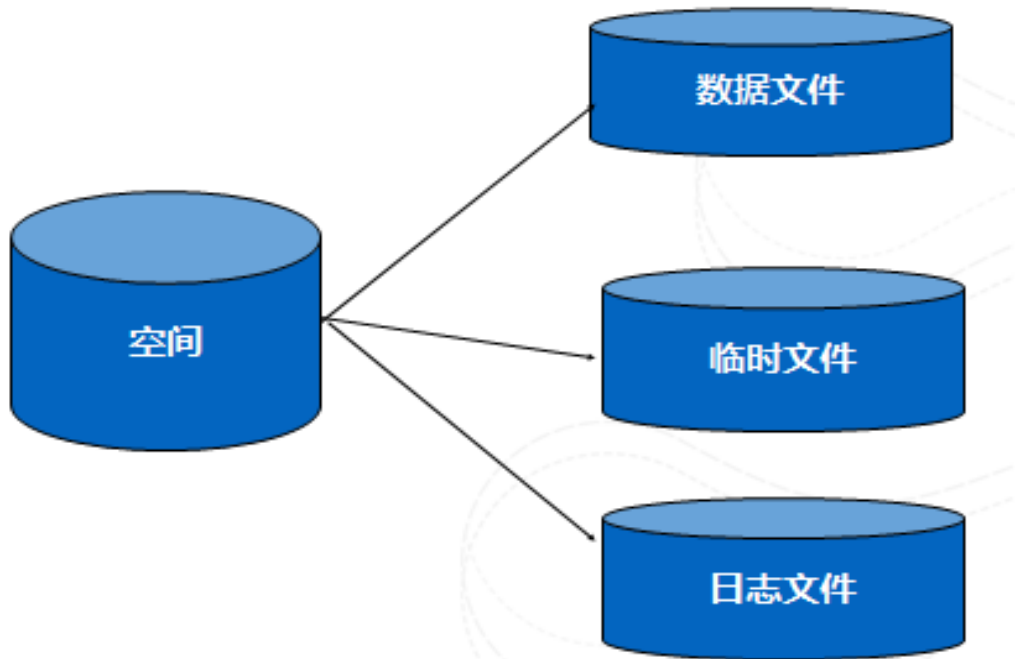
配置问题

1.内存规格不足，弹性升级RDS的规格配置；

九、案例九 disk 100%

案例九 disk 100%

磁盘空间组成：数据文件，日志文件，临时文件



空间最佳实践

1

数据空间问题

- 1.采用optimize table收缩表空间；删除不必要的索引；
- 2.使用tokudb压缩引擎；

2

日志空间问题

- 1.减少大字段的使用；
- 2.使用truncate 替代delete from；

3

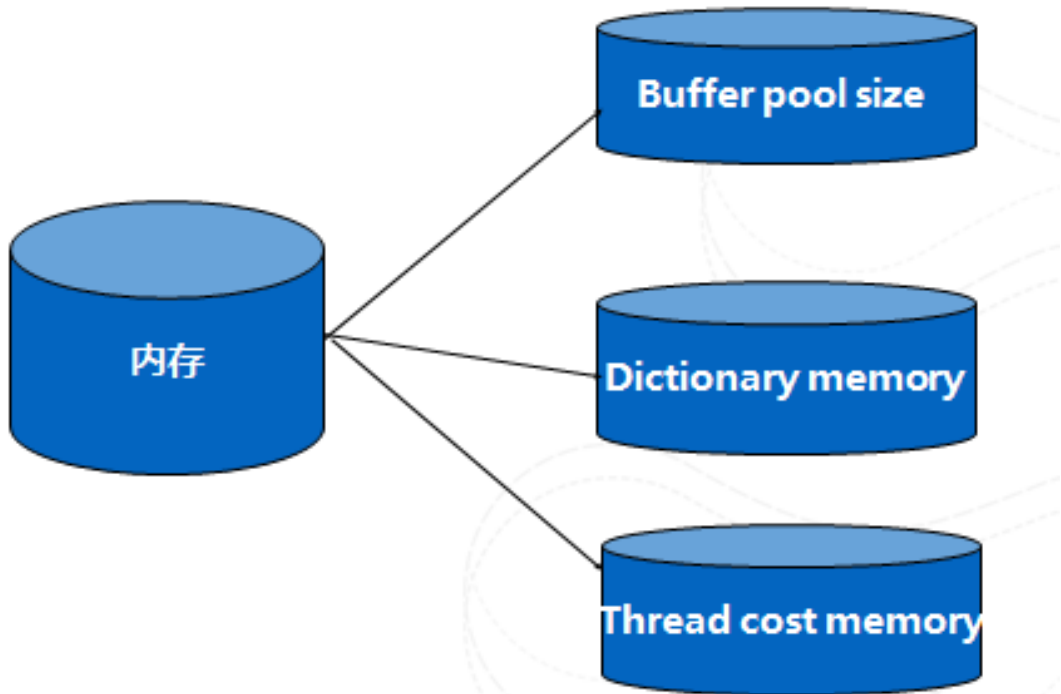
临时空间问题

- 1.适当调大sort_buffer_size；
- 2.创建合适索引避免排序；

十、案例十 mem 100%

案例十 mem 100%

内存组成：Buffer pool size，Dictionary memory，Thread cost memory



内存最佳实践

1

Buffer pool size

- 1.创建合适的索引，避免大量的数据扫描；
- 2.去除不必要的索引，降低内存的消耗；

2

Thread cost memory

- 1.创建合适的索引避免排序；
- 2.只查询应用所需要的数据；

3

Dictionary memory

- 1.不要过度分表；



Thank you !