DTCC

第十一届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2020

架构革新 高效可控

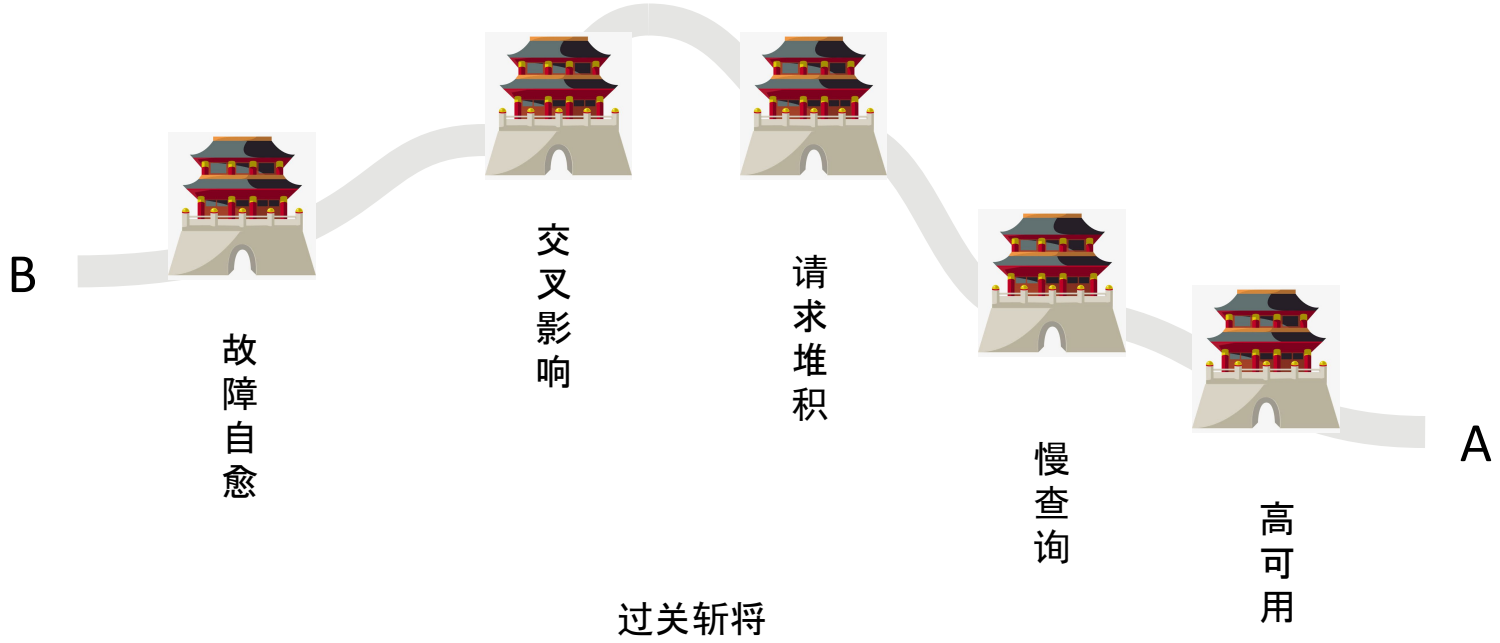
IT168.com ChinaUnix.net ITPUB

北京国际会议中心 | 2020/12/21-12/23

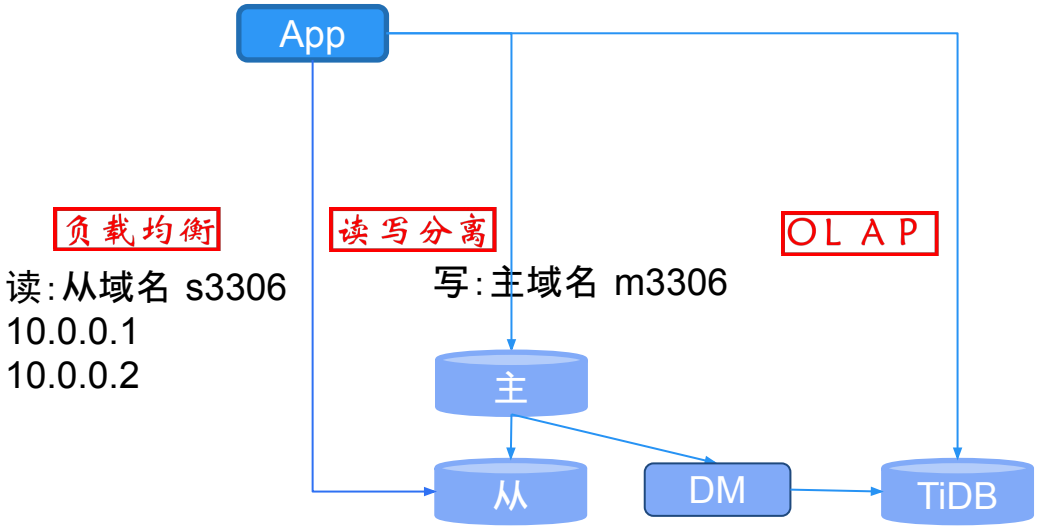
贝壳找房数据库服务综合治理之路

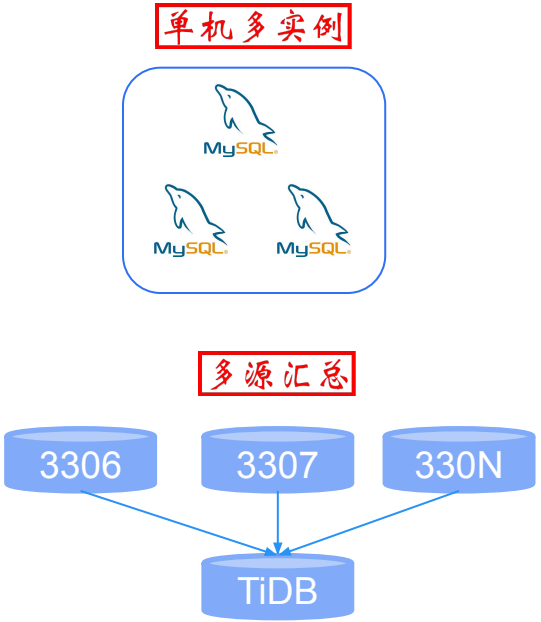
贝壳找房 DBA 负责人 张良

目录



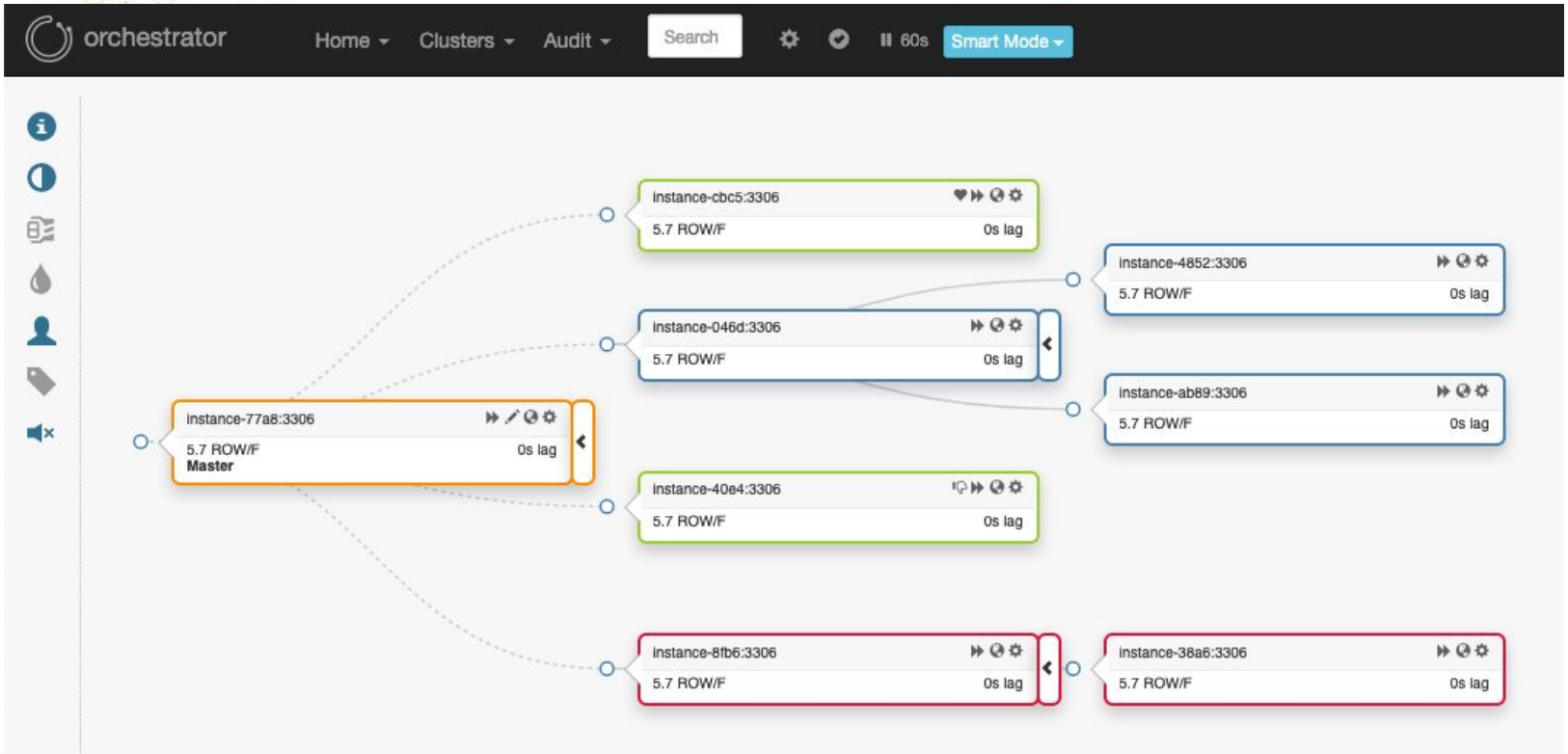
数据库访问方式--知己知彼





高可用

MySQL 高可用


orchestrator
Home
Clusters
Audit
Search
60s
Smart Mode
instance-cbc5:3306
5.7 ROW/F
0s lag
instance-4852:3306
5.7 ROW/F
0s lag
instance-046d:3306
5.7 ROW/F
0s lag
instance-ab89:3306
5.7 ROW/F
0s lag
instance-77a8:3306
5.7 ROW/F
Master
0s lag
instance-40e4:3306
5.7 ROW/F
0s lag
instance-8fb6:3306
5.7 ROW/F
0s lag
instance-38a6:3306
5.7 ROW/F
0s lag

贝壳定制版

alert (监控报警)
dns (DNS 切换)
failure-detection-check (故障场景检测, 主库死机等待)
find-errant-gtid-events (errant-gtid 定位)
kill-all-connections (杀长连接)
pre-failover-check (故障前场景检测，防止异常误切)
pre-graceful-takeover-check (自动切换前长查询检测，防止切换中途失败)
promotion-rules (自动定义切换规则，防止线下库误切为主)
slave-failover (从库故障高可用，同步延迟，同步中断自动切换)
Prometheus exporter
支持实例级别 Ignore Filter
MySQL 8.0 兼容性问题解决

贝壳定制版之 主库防误切换

100% 线上部署 Topolog Discover，在禁用自动切换下观察连续运行 >1 个月
凌晨批量更新及查询压力，主库 CPU 打满新建连接慢。且单条 SQL 大事务，从库无法获取新增更新。
增加 failure-detection-check，DeadMaster 故障等待 180s 后判断是否需要触发决策。
防止跨机房误切换

- "PreventCrossDataCenterMasterFailover": true

防止备份机或统计为误切换

- 动态自定义 promotion-rules
- 禁用备份实例、离线统计实例切换

贝壳定制版之 从库高可用

从库无法访问(last_check_invalid)
- 连接数打满
- 服务器宕机
- 网络中断

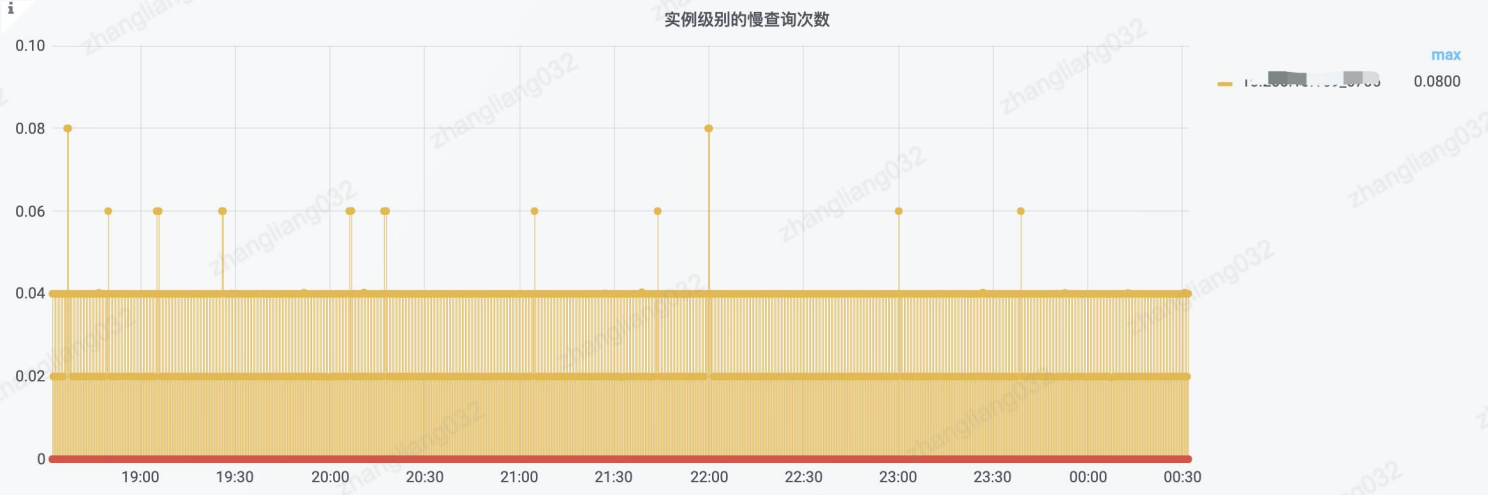
从库同步中断(not_replicating)
- SQL 线程执行出错
- IO 线程读包异常

从库同步延迟(replication_lag)
- 写流量突增，个别追不上主库
- 从库硬盘 IO 异常，写性能下降
- 从库查询压力过大或出现互斥锁

慢查询

慢查询统计

慢查询统计



慢查询分析

慢查询指纹前缀(点击查看详情)	时长(s)	查询时间	最早出现	db	次数	实例信息	客户端ip	用户名
select	5.00	2020-12-07T20:05:56Z	2019-08-24T07:17:42Z		1.00			
select	2.46	2020-12-07T22:59:44Z	2019-09-06T14:09:09Z		4.00	6		

SQL 自助优化

SQL优化 ×

3306

评估 美化 压缩 指纹 清空

```
SELECT * FROM sakila.actor;
```

SOAR

★★★☆☆ 75分

```
SELECT
  *
FROM
  sakila. actor
```

最外层 SELECT 未指定 WHERE 条件

- **Item:** CLA.001
- **Severity:** L4
- **Content:** SELECT 语句没有 WHERE 子句，可能检查比预期更多的行(全表扫描)。对于 SELECT COUNT(*) 类型的请求如果不要求精度，建议使用 SHOW TABLE STATUS 或 EXPLAIN 替代。

不建议使用 SELECT * 类型查询

- **Item:** COL.001
- **Severity:** L1
- **Content:** 当表结构变更时，使用 * 通配符选择所有列将导致查询的含义和行为会发生更改，可能导致查询返回更多的数据。

没有慢日志怎样优化慢查询？

借助 sys 库

```
mysql> select * from sys.statement_analysis order by avg_latency desc limit 10\G
```

借助 performance_schema 库

```
mysql> SELECT SCHEMA_NAME, DIGEST_TEXT, COUNT_STAR, AVG_TIMER_WAIT/1000000000000,
SUM_ROWS_SENT, SUM_ROWS_EXAMINED, SUM_NO_INDEX_USED, SUM_NO_GOOD_INDEX_USED,
FIRST_SEEN, LAST_SEEN
FROM performance_schema.events_statements_summary_by_digest
ORDER BY AVG_TIMER_WAIT desc;
```

请求堆积

主动防御之 kill

长事务，少量慢查询

```
mysql > show full processlist

mysql > kill [query | connection]
```

强化版

```
pt-kill --busy-time=5s --victims=all --interval=5 --run-time=1s --print --kill
```

主动防御之 max_execution_time

特点：

- 超过阈值的 SQL 将被自动 KILL，无需像 pt-kill 一样设置间隔。
- 8.0 前仅对 SELECT 生效，8.0 后可 KILL 更新请求。

注意：

- max_execution_time 单位是毫秒
- 各别查询想逃过限制 SELECT /*+ MAX_EXECUTION_TIME(0) */

如何监控、定位被 KILL 的语句？

```
mysql> show global status like "Max_execution_time_exceeded";
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| Max_execution_time_exceeded | 0     |
+-----------------------------+-------+
1 row in set (0.01 sec)
```

cat mysql-slow.log | grep -B 2 '^# Query_time: N.0'

pt-query-digest --order-by Query_time:max

主动防御之 Query Rewrite

SQL 

- MySQL 5.7+ 只支持 SELECT
- 8.0+ 支持 UPDATE/INSERT/DELETE 等
- “?” 只支持单值匹配
- pattern_database 长度为 24 < 64

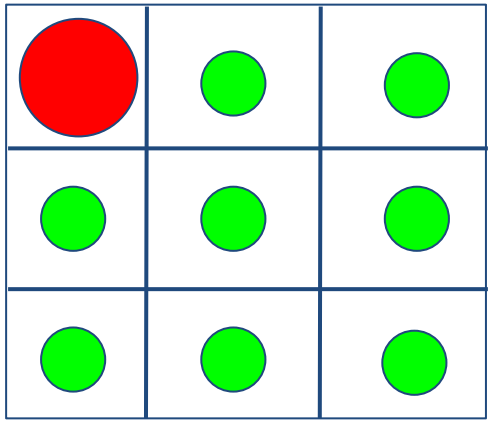
主动防御之 Query Rewrite

```
# 插件安装
mysql -u root -p < install_rewriter.sql

# 配置封禁策略
INSERT INTO query_rewrite.rewrite_rules (pattern, replacement) VALUES('SELECT SLEEP(?)', 'DO 1');
CALL query_rewrite.flush_rewrite_rules();

# 禁用封禁策略
UPDATE query_rewrite.rewrite_rules SET enabled = 'NO' WHERE id = 1;
CALL query_rewrite.flush_rewrite_rules();
SET GLOBAL rewriter_enabled = OFF;

# 卸载插件
mysql -u root -p < uninstall_rewriter.sql
```

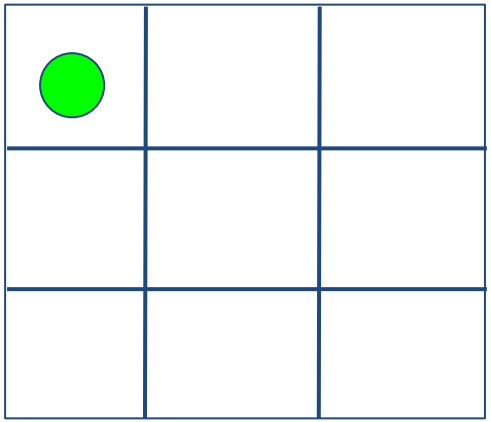
交叉影响

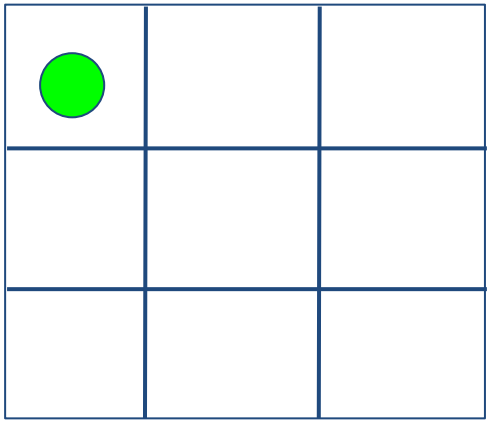
资源限制/隔离

预定义资源限制



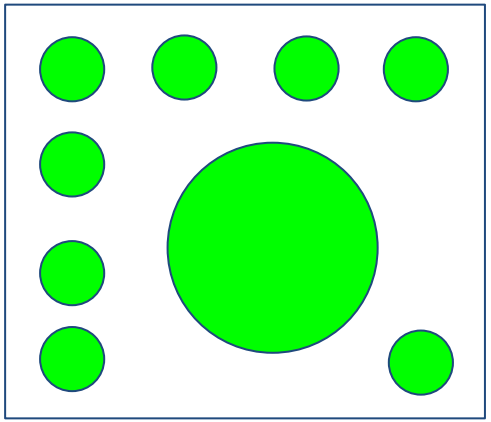
扩容：1 + 1 + 1 + 1...



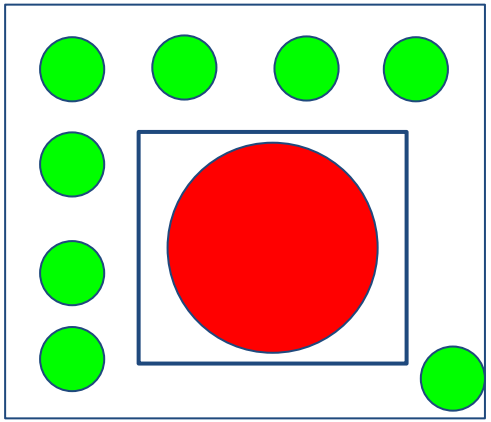


资源限制/隔离

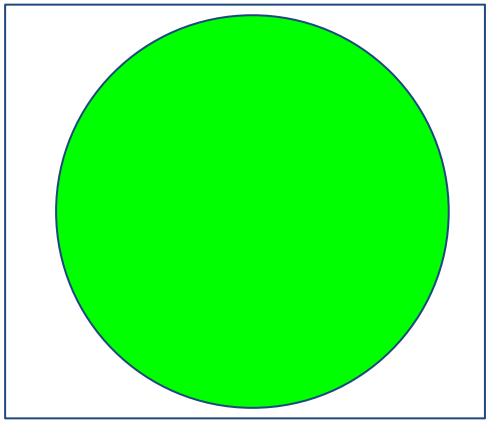
无资源限制



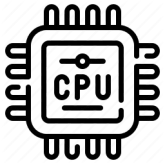
基于故障的资源限制



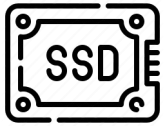
服务迁移



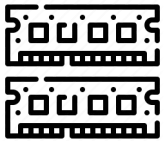
资源限制/隔离



MySQL：百分比
Redis：核绑定



防止离线任务影响在线请求



不做限制，避免进程僵死或 OOM

资源限制/隔离

```
# CPU 使用率
cgcreate -g cpu:mysql/3306
cgset -r cpu.cfs_period_us=${cfs_period_us} mysql/3306
cgset -r cpu.cfs_quota_us=${cfs_quota_us} mysql/3306

# CPU 核绑定
cgcreate -g cpuset:redis/6379
cgset -r cpuset.mems=0|1 redis/6379
cgset -r cpuset.cpus=N redis/6379

# IO 限制
cgcreate -g blkio:mysql/3306
cgset -r blkio.throttle.read_bps_device="${DEVICE} ${VALUE}" mysql/3306
cgset -r blkio.throttle.read_iops_device="${DEVICE} ${VALUE}" mysql/3306
cgset -r blkio.throttle.write_bps_device="${DEVICE} ${VALUE}" mysql/3306
cgset -r blkio.throttle.write_iops_device="${DEVICE} ${VALUE}" mysql/3306

# 策略绑定
ps -L ${PID} | grep -v LWP | awk '{print $2}' | tr "\n" " " | xargs cgclassify -g cpu:mysql/3306

# 删除策略
cgdelete -g cpu:mysql/3306
```

故障自愈

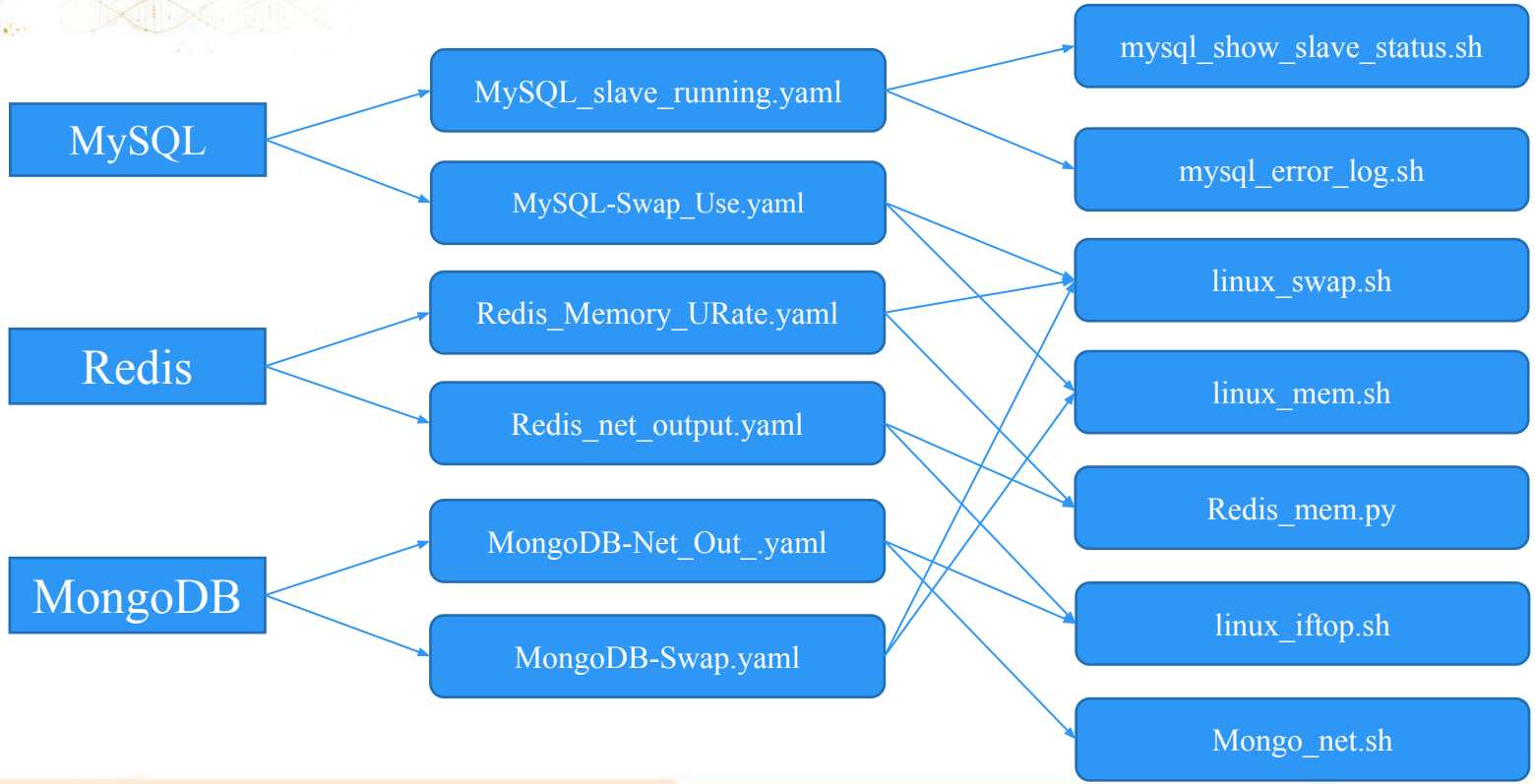
故障诊断之根因分类

一级分类	二级分类	解释说明
业务类	流量异常	流量突增，并发过高，批量任务等原因引起的故障，例如网卡打 满，主从延迟
业务类	索引设计缺陷	索引缺失，冗余，索引列 顺序不当
数据库类	参数配置不当	连接参数，性能参数，其它参数
数据库类	容量不足	例如磁盘打满，内存打满，QPS裕度不足
服务器类	硬件故障	内存/CPU/硬盘/网卡 等方面的故障
服务器类	资源抢占	服务器上其它进程占用了大量资源，导致数据库性能受到影响
非技术类	误报	监控系统的问题或监控配置不当等原因 导致的误报

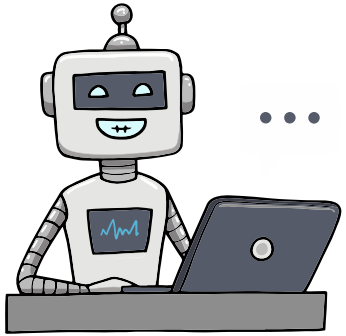
故障诊断之分析套餐

故障类型	故障处理套餐	检查点
Redis内存报警	Redis内存检查套餐	Redis实例相关各项内存指标(用户数据/客户端缓冲/aof等)，服务器内存余量等
MySQL主从延迟	主从延迟检查套餐	线上流量情况，从库慢查询情况，从库的CPU负载/IO负载/网络情况等
MongoDB cpu报警	MongoDB cpu检查套餐	线上流量情况，慢查询情况，服务器上CPU占用进程的TOP N等
服务器磁盘报警	服务器磁盘检查套餐	服务器各关键路径下(数据/日志)的磁盘占用情况，线上流量情况等
主动检查	主动检查套餐	根据存储类型的不同(MySQL/Redis)，检查不同的常规检查项

故障诊断之分析套餐

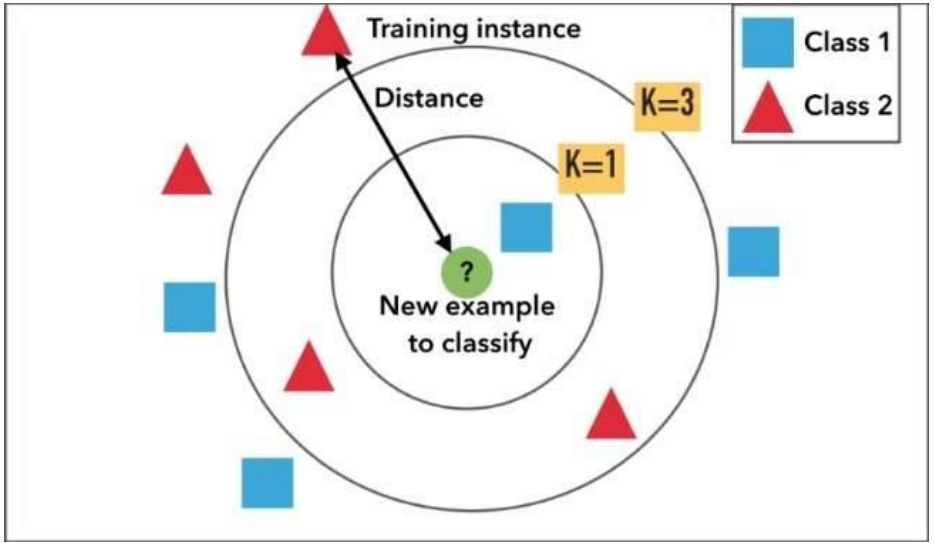

MySQL
Redis
MongoDB
MySQL_slave_running.yaml
MySQL-Swap_Use.yaml
Redis_Memory_URate.yaml
Redis_net_output.yaml
MongoDB-Net_Out_.yaml
MongoDB-Swap.yaml
mysql_show_slave_status.sh
mysql_error_log.sh
linux_swap.sh
linux_mem.sh
Redis_mem.py
linux_iftop.sh
Mongo_net.sh

故障诊断之机器学习



故障诊断之分类算法

如果一个样本在特征空间中的k个最邻近的样本中的大多数属于某一个类别，则该样本也划分为这个类别。KNN算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。



特征项(x1…xn)					类别
cpu_idle	Io_util	net_output	slow_pt	connected	reason
20	80%	200	500	500	原因1
25	95%	50	170	1000	原因1
90	30%	30	5	5000	原因2
35	75%	50	300	100	原因1
89	23%	40	15	3500	原因2

欧几里得距离

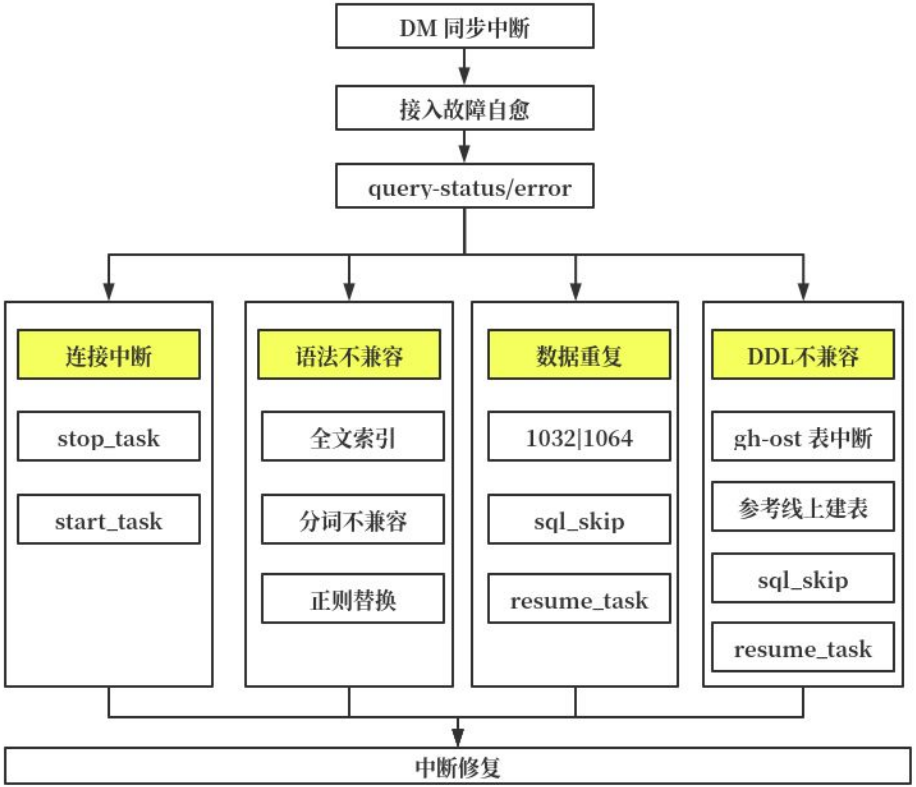
$$E(x,y) = \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}$$

故障自愈举例之 DM 中断

MySQL 多集群通过 TiDB 的 DM 工具进行多源汇聚，为业务提供 OLAP 的融合查询能力。

但 TiDB 并不完全与 MySQL 兼容，上游业务 MySQL 频繁变更频繁，下游 TiDB 会频繁中断同步。

人工处理故障恢复速度慢，简单重复劳动成就感低。



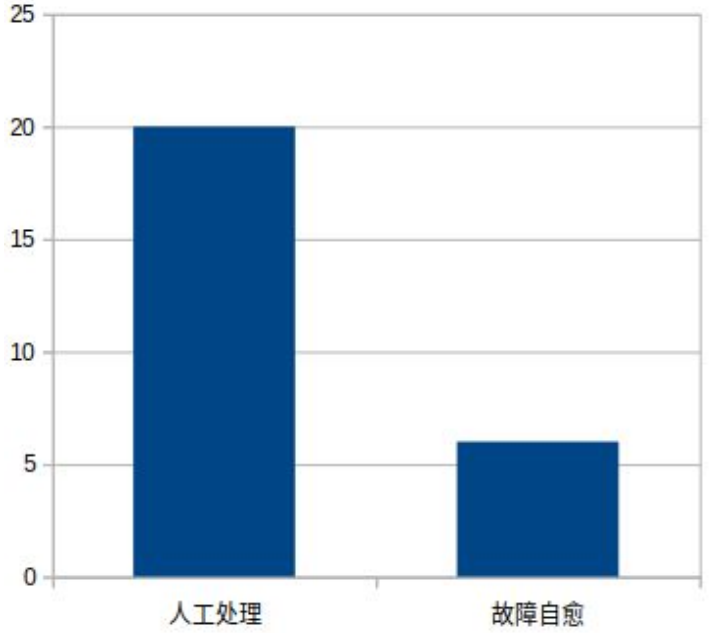
故障自愈举例之 DM 中断

效率

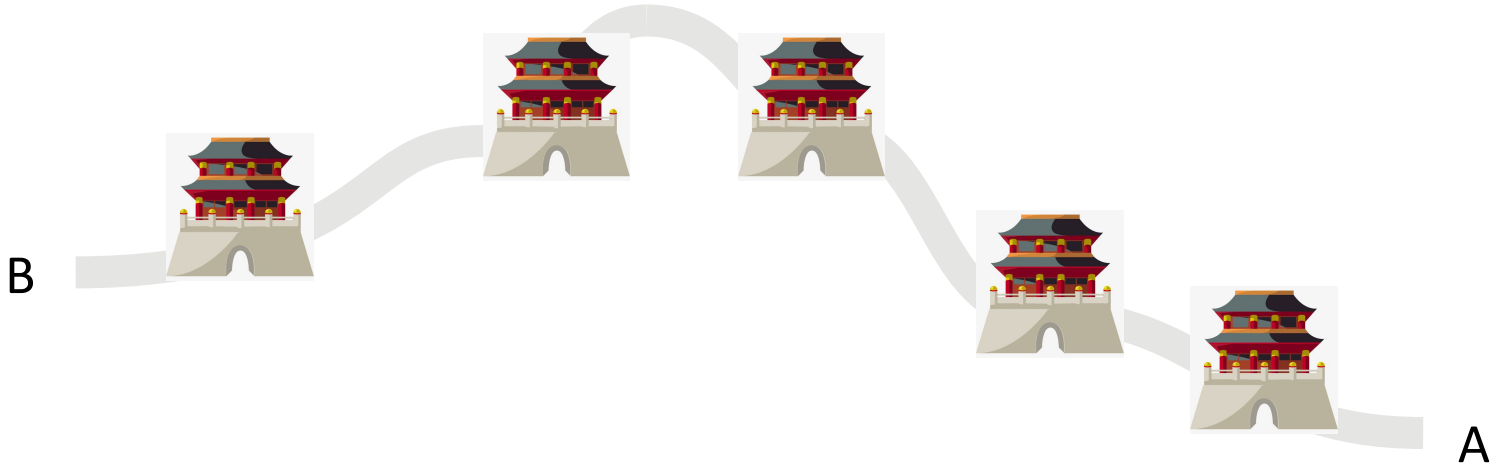
效率提升

可用性

SLA提升



【注】2020年上半年 DM 自动修复 1275 次，不可用时间缩短：300h（即：1275*(20min-6min)）



雄关漫道真如铁，而今迈步从头越

THANKS