饿了么数据库架构变迁

- 自我介绍
- 开始阶段
- 阶段一：DB升级和扩容
- 阶段二：垂直拆分
- 阶段三：Sharding
- 阶段四：异地容灾
- 总结

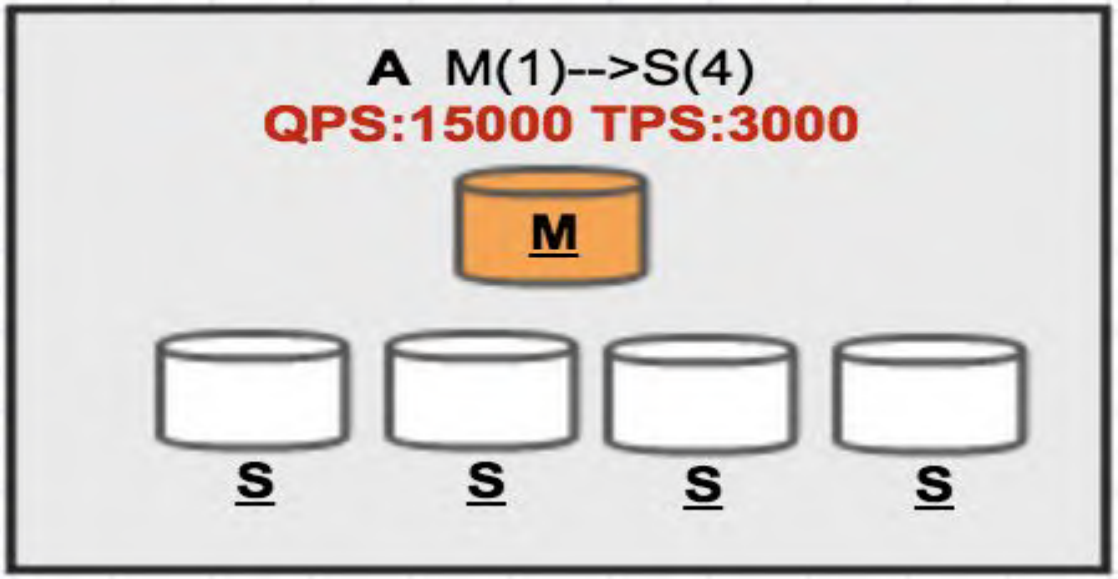
自我介绍

- 虢（guo）国飞，网名“飞扬过海”
- DB工作10+（年）
- 关注MySQL、MSSQL、PostgreSQL和部分NoSQL
- 热衷研究数据库监控和自动化
- 5173、新蛋网、沪江网、饿了么

没有高大上的工具，
如何做好数据库架构？

我在饿了么的
数据库架构之旅

开始阶段

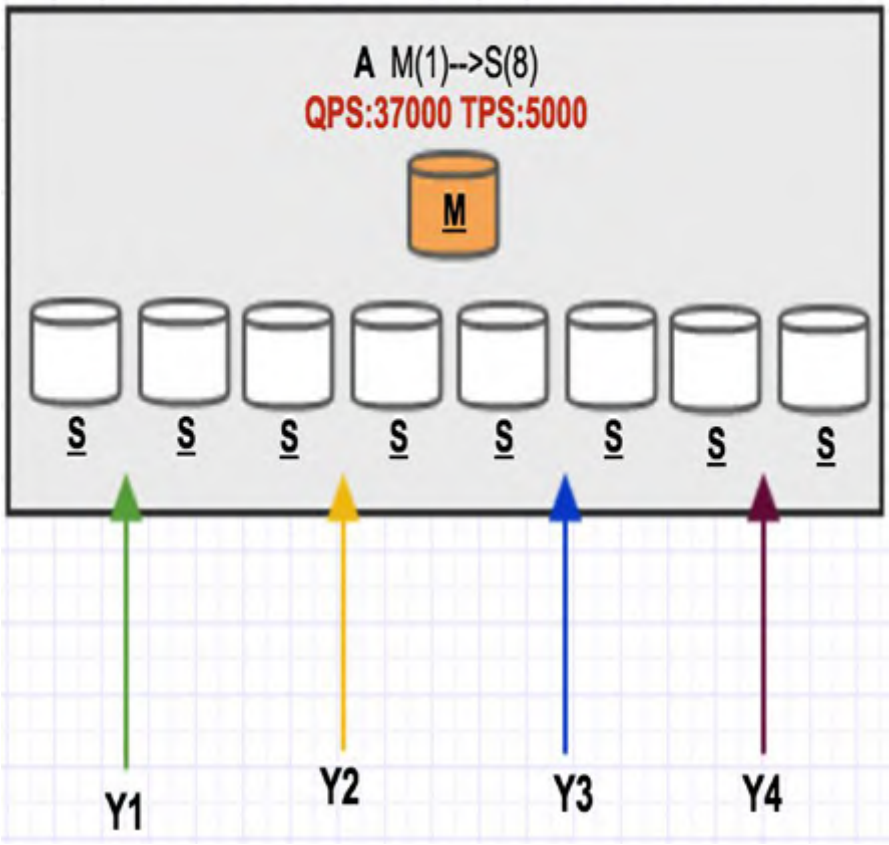


冒烟

救火

说明：
一开始的数据库架构，存在比较多的问题：
磁盘空间不足、主从延时、连接数不足、
SlowSQL打垮集群、无自动容灾机制等。

DB升级和扩容

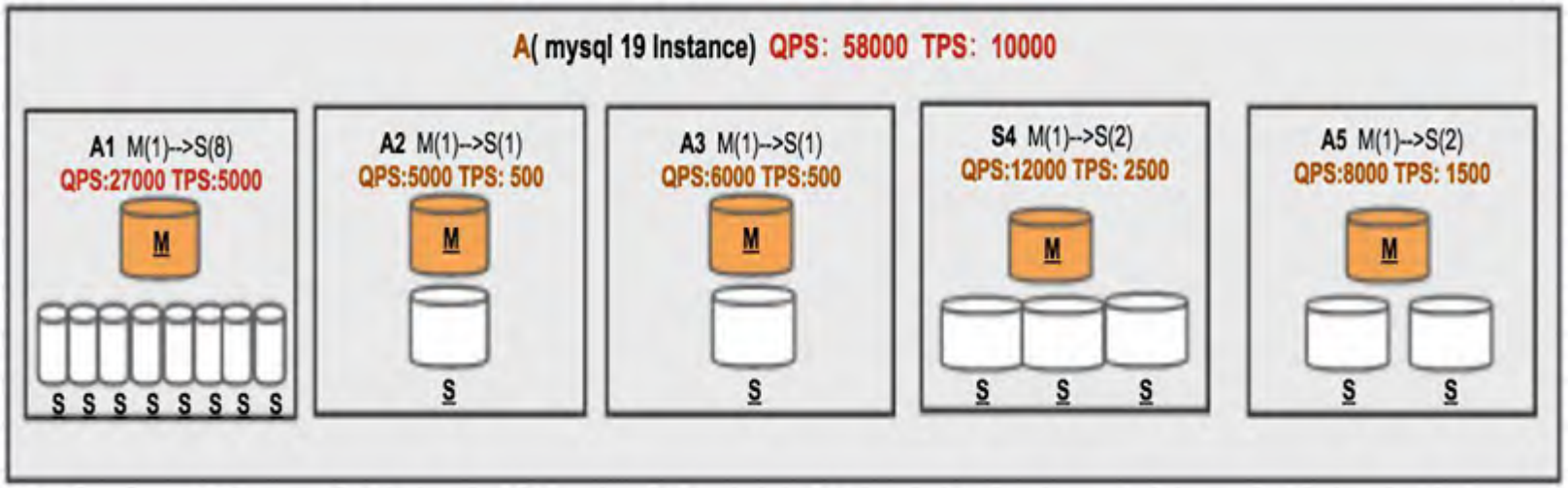


说明：
第一阶段的改造，主要在这几方面：
1. 升级硬件，磁盘升级到SSD，加大内存；
2. 增加slave，将业务按不同的slave组划分访问；
3. 引入了MHA。

优点：
1. 增加了系统吞吐量（提高2倍），缓解了延时；
2. 单业务的SlowSQL不会打垮整个集群，而且每套业务有两台slave可以访问，无单点；
3. 增加了master自动failover；
4. 增加了Slave，缓解了连接数不足。

缺点：
1. 架构太重，failover机制风险比较高；
2. slave 压力不均，资源利用率不高；
3. 系统容量有限，不能继续扩展。

垂直拆分



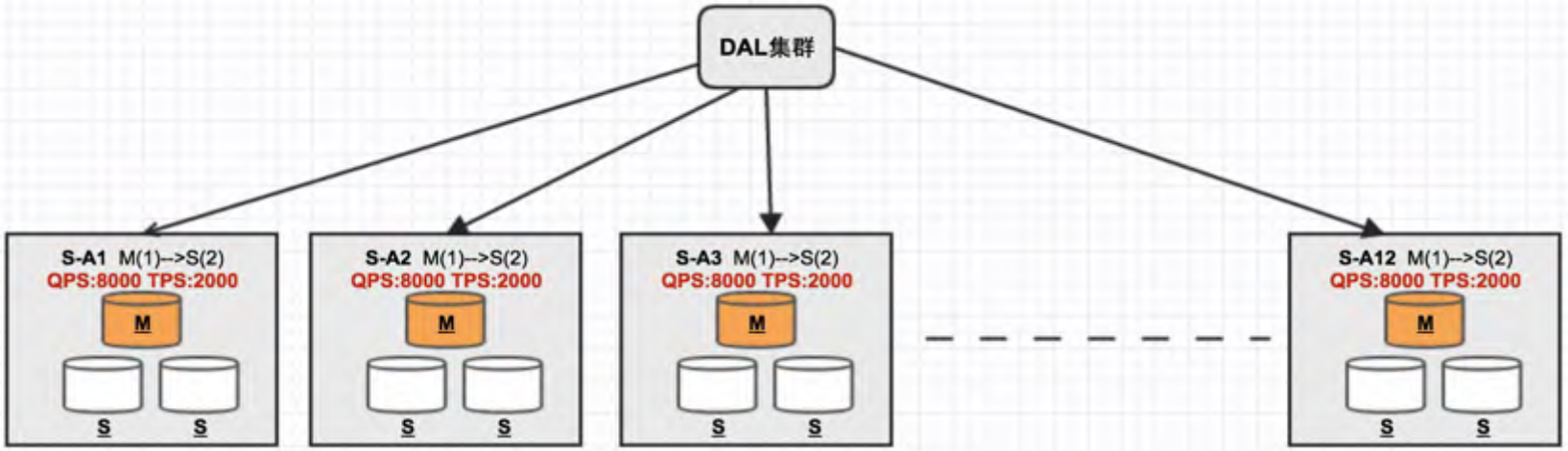
说明：
这个阶段是为满足300万单（实际支持了330多万单）的容量规划而做的架构调整，主要做了以下改进：
1. 将核心业务集群从一套拆分为5套；
2. 根据收集的数据规划每个集群的Instance数量，一个共分了19个Instance；
3. 最大的难点：如何有效的推动业务改造（数据、方案、节奏、沟通、协调）。

垂直拆分

优点：
1. 进一步提升了系统的容量（提高1倍）；
2. 业务垂直拆分后，磁盘空间问题得以解决；
3. 连接数和SlowSQL的影响得到控制；
4. 业务模块解耦，逻辑更清晰，方便对单块业务做控制。

缺点：
1. 主业务架构还是比较重，有8台slave，failover风险还是比较高；
2. 5套集群有一套出现问题，可能影响到主业务，风险点增加（不过后面业务做了降级处理）；
3. 主DB系统随着业务的增长，压力还是比较大，尤其是写的压力。

Sharding



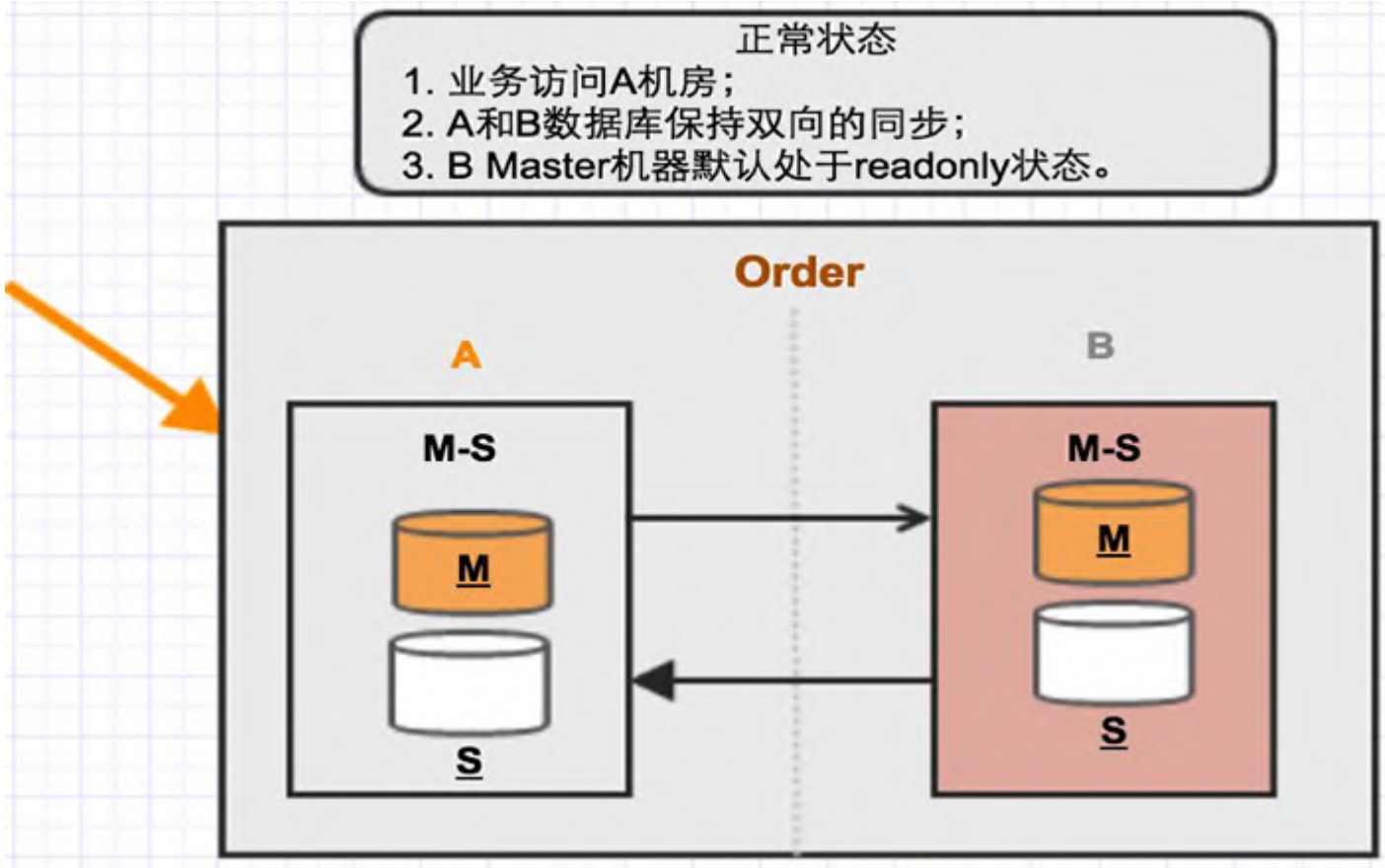
说明：
阶段三的架构就是我们现在的架构，改进如下：
1. 将数据按两个维度进行了拆分，支持用户和商家两个方向的查询；
2. 将核心DB的表 sharding 成120个分片，分配到12套集群里面；
2. 现在这套架构下可以支持 TPS：80000、QPS：200000。

Sharding

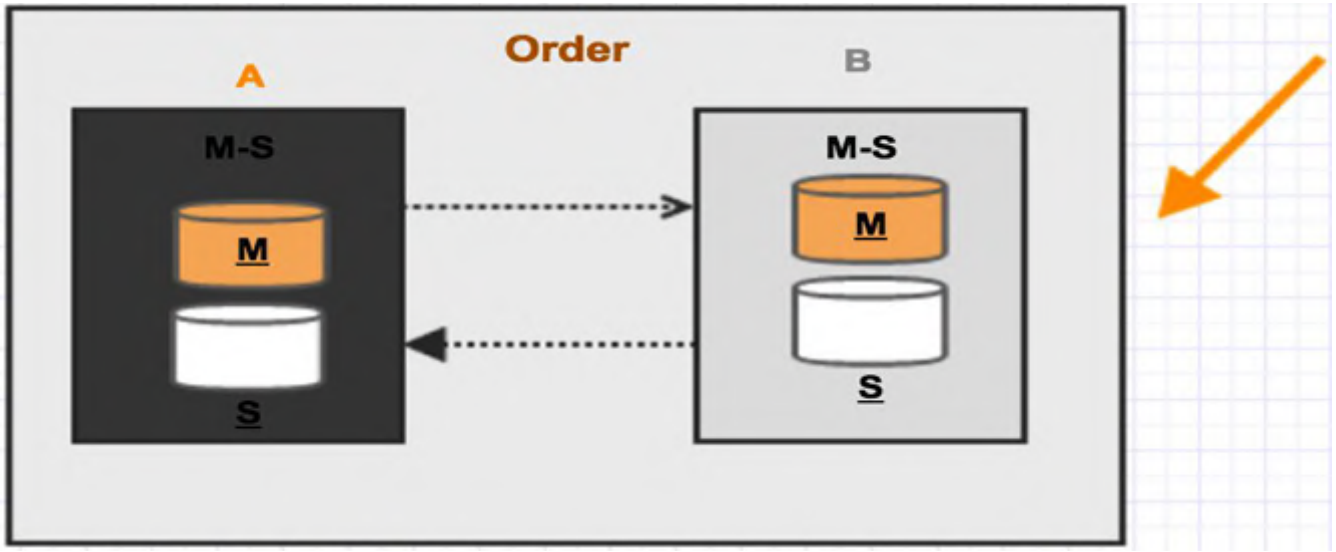
优点：
1. 极大的增加了系统的容量（差不多是原来的10倍）；
2. 缓解了核心主表（主库）TPS的压力；
3. 分散了风险，一套集群出现问题，只会影响部分用户，容灾机制轻巧；
4. DAL 为DB提供了有力的支持（连接池、限流、熔断、负载均衡）。

缺点：
1. 引入了新的风险点DAL，DAL也需要完善的容灾机制；
2. DBA维护量增加；
3. 成本增加。

异地容灾--Standby



异地容灾—切换



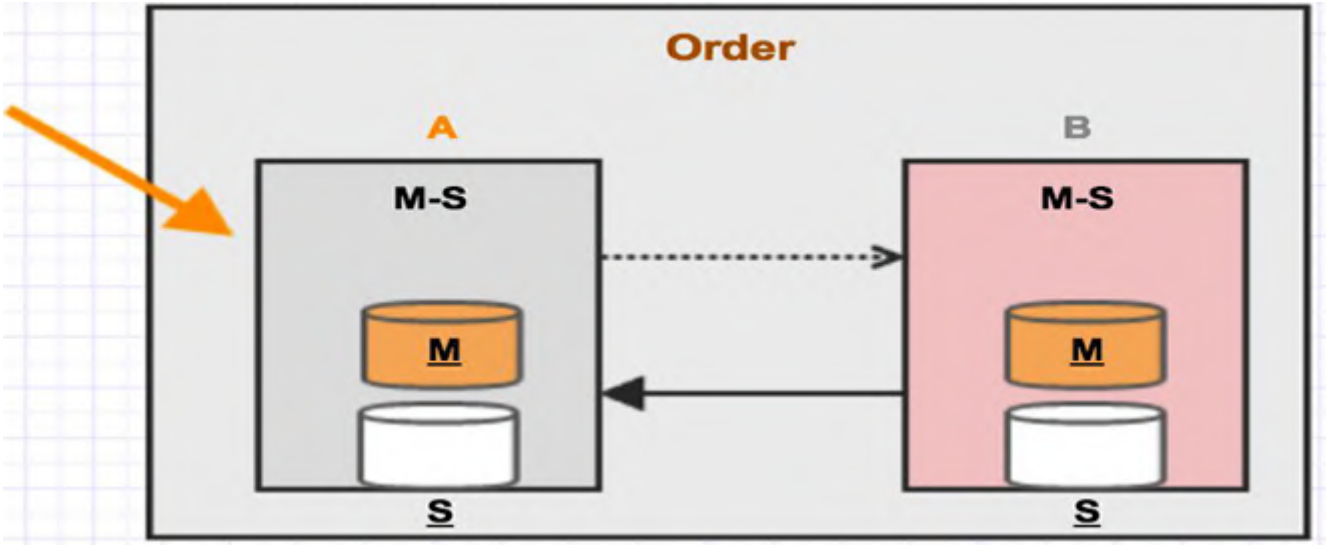
切换B操作

1. DBA 更改自增因子（auto增加10万）；
2. DBA 暂停 同步(两边同步都需要暂停)；
3. Dal 调整订单生成配置；
4. 访问源指向B；
5. 业务调整未同步的订单状态。

数据冲突

数据状态

异地容灾—切回



切回A操作

1. 业务先用脚本处理掉A残留的数据；
2. DBA 打开B到A的数据同步（等数据处理完成再打开）；
3. 切断B对业务的访问，DBA将B 数据库设置为Readonly状态，并确保两边数据一致，然后通知业务切换访问到A。

架构感悟

- 数据
- 沟通
- 节奏
- 信誉
- 平衡（ROI）
- 简单

Q&A

饿了么和你一起拼，我们招人：

DBA、Java、Python、架构师、OPS …

mail: guofei.guo@ele.me

Thanks

数据库核心技术与应用实战峰会

CSDN