Akka persistence 实战分享

洋关森@水滴技术团队

Why use Akka persistence?

Akka 与 Akka persistence

1.Akka是一个基于JVM 和Actor模型实现的用于构建高性能，分布式，消息驱动应用的类库。

2.Actor VS PersistenceActor

业务背景

- 用户向公众号发送一条消息，系统根据请求先后顺序给它分配一个编号，并根据暗号活动设置分配对应的奖品。

- 核心逻辑：
- 1.分配编号
- 2.分配奖品

方案1——业务代码加锁

优点：

1.容易理解

2.实现简单

缺点：

1.普通的锁只能在单机下使用，分布式锁效率太低；

2.需要在业务代码层面保证线程安全，要小心各种死锁陷阱；

方案2——基于数据库乐观锁

优点：
1.利用数据库解决多机问题；
2.业务代码层面不需要加锁；

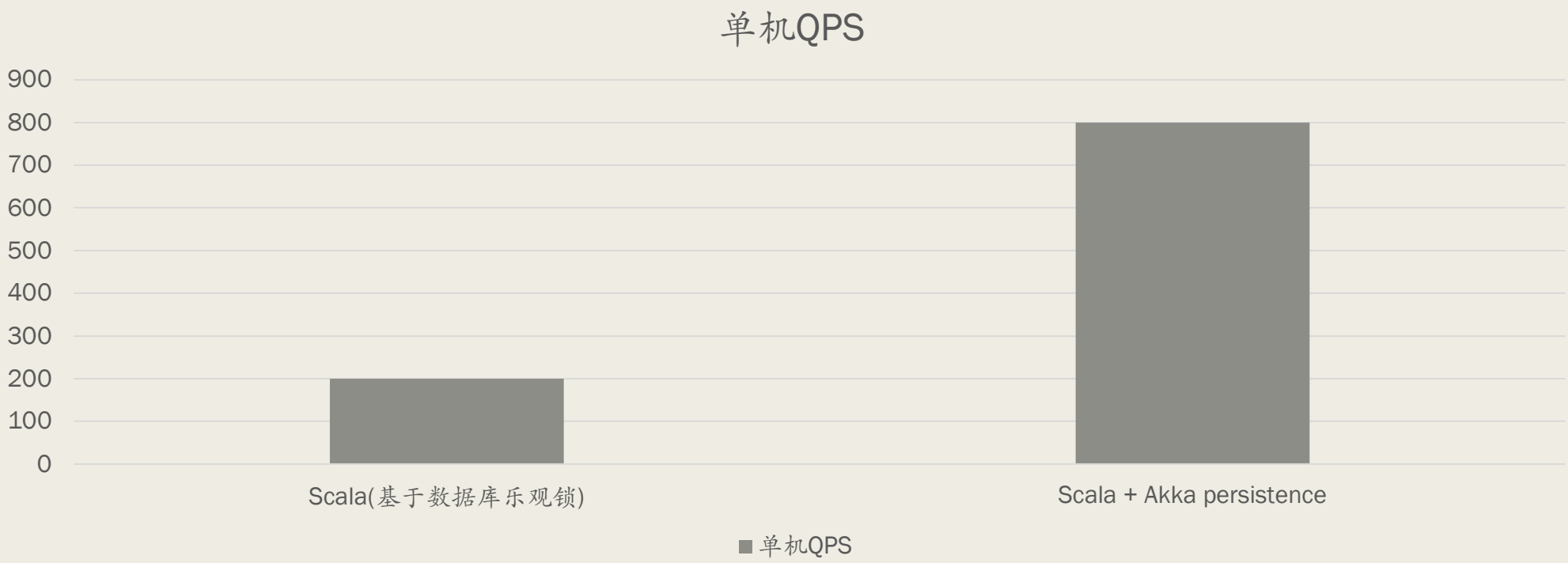
缺点：
1.并发竞争激烈时，数据库执行大量失败，程序错误增多，甚至拖慢数据库

方案3——CQRS架构之PersistenceActor

优点：
1.更新的是内存中的数据，速度快，同时基于Actor，不需要在业务代码层面考虑线程安全问题；
2.持久化所有数据变化的事件，可回溯，恢复任一时刻状态；

缺点：
1.读写有延迟，保障的是最终一致性；
2.业务架构会变的复杂，适用场景有限；
3.需要存储大量事件，存储成本上升；

数据对比


单机QPS
900
800
700
600
500
400
300
200
100
0
Scala(基于数据库乐观锁)
Scala + Akka persistence
单机QPS

事件批量持久化

虽然我们在数据库层面已经做了分离，不会有竞争操作，甚至只需要简单的插入，但压测下来效果总是达不到预期，后来排查日志发现，一个简单的插入竟然要耗时50ms，再加之Actor内部是串行处理的，那么即使不发生任何意外情况，每秒做多只能处理200多个请求，这远远达不到我们的需求。

事件批量持久化是一个好的选择，比如积累了一定事件或者一段时间执行一次持久化操作，具体参数需要根据真实请求来调整，通过这种优化，理想情况下达到了800多QPS，当然使用批量持久化也会让系统变得稍稍复杂，但由此带来性能却大大提升。

Actor版本管理

在业务发展的过程需求也会不断发生变化，因为PersistenceActor与普通Actor不同，它是有持久化状态，因为有大量事件，所有在持久化事件以及快照的时候我们往往会选择一些高效的存储方式，比如将存储的内容序列化成二进制数据，如果后续我们修改事件或者快照的结构则必须能兼容旧的数据。

这个问题也是我们在线上运行了一段时间后才重视的，因为修改了快照里面的一个类结构，导致Actor无法正常重启，当然所幸还是可以通过事件来恢复状态，这是一个教训，所以后来我们团队制定了策略，若是修改了Actor的结构，发布时一定要两个人以上的code review。

总结与期望

- 尝试着用新的思维去思考问题，或许有不一样的思路；
- 当前持久化选用的DB是MySQL，希望未来能尝试使用官方建议的cassandra ；

谢谢大家!