meitu美图

Titan:分布式Redis协议存储

本次分享的内容

- 讲到的
  - Titan、Pika、Redis性能对比
  - Redis在某些场景下的局限
  - Titan在可用性扩展性和分布式事务方面带来的好处
  - Titan、Pika、Redis选型
  - 性能测试
- 没有覆盖的内容
  - TiKV底层原理
  - Titan的具体实现

meitu美图

兼容Redis协议

meitu美图

想用Titan替换Redis?

Titan vs Pika vs Redis

meitu美图

压测机

CPU: Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, Mem: 96G, Disk: 5*480G

Redis and Pika (SSD stat)

CPU: Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, Mem: 96G, Disk: 5*480G

TiKV (SSD stat)

3 x (CPU: Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, Mem: 96G, Disk: 5*480G)

Titan(TiKV) configuration

```
[server]
grpc-concurrency = 8
[raftstore]
sync-log = false
[rocksdb.defaultcf]
block-cache-size = “9.6GB"
[rocksdb.writecf]
block-cache-size = “3.6GB”
```

Pika configuration

worker_thread: 60 (1.5 x number of cores)

Benchmark Commands

LPUSH

./fperf -tick 1s -connection <Concurrency> -N 1000000 -server redis://10.12.132.16:6379 redis lpush lst:__rand_int__ helloworld

LPOP

./fperf -tick 1s -connection <Concurrency> -N 1000000 -server redis://10.12.132.16:6379 redis lpop lst:__rand_int__

LRANGE

./fperf -tick 1s -connection <Concurrency> -N 1000000 -server redis://10.12.132.16:6379 redis lrange lst:__rand_int__ 0 -1

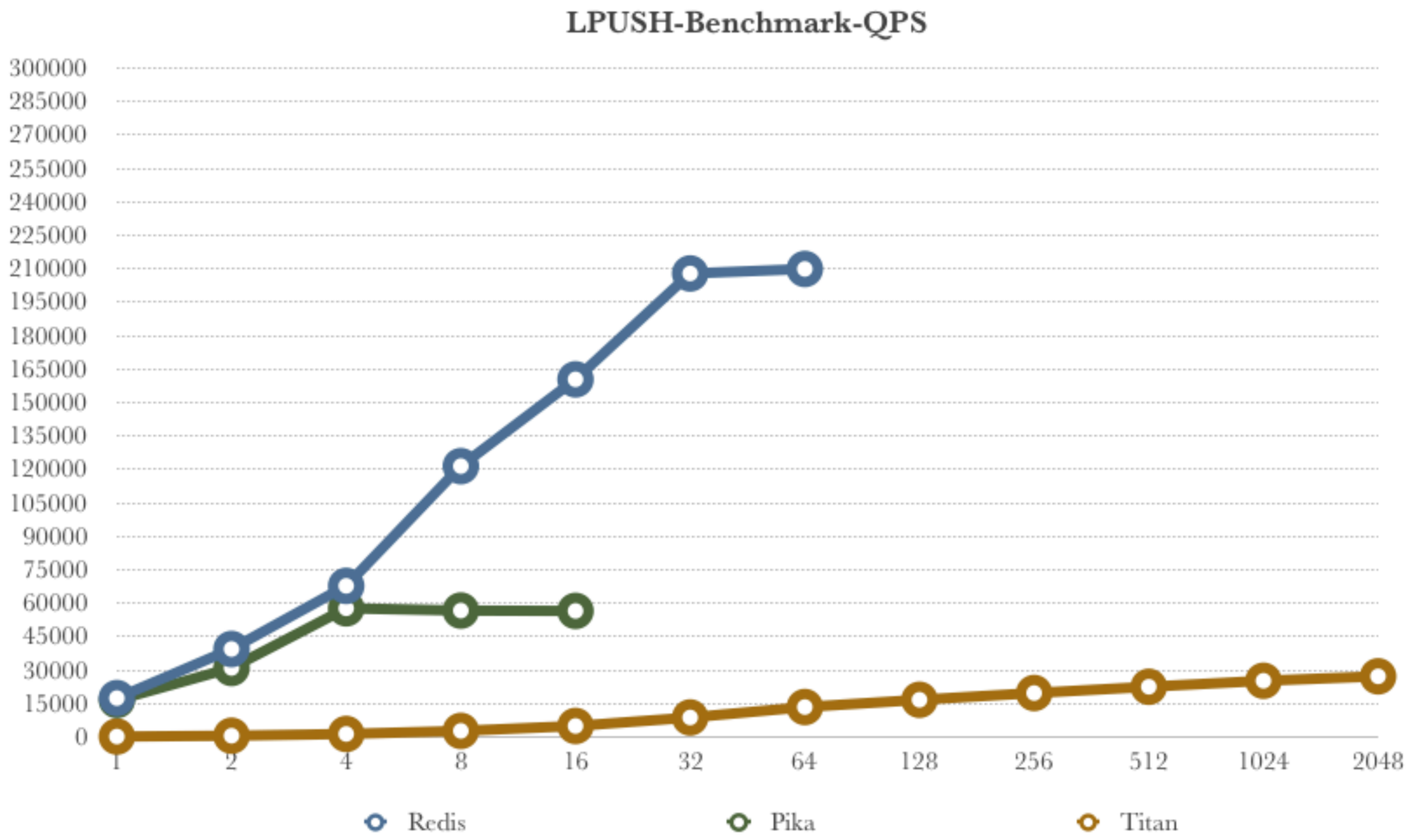
SET

./fperf -tick 1s -connection <Concurrency> -N 1000000 -server redis://10.12.132.16:6379 redis set key:__rand_int__ helloworld

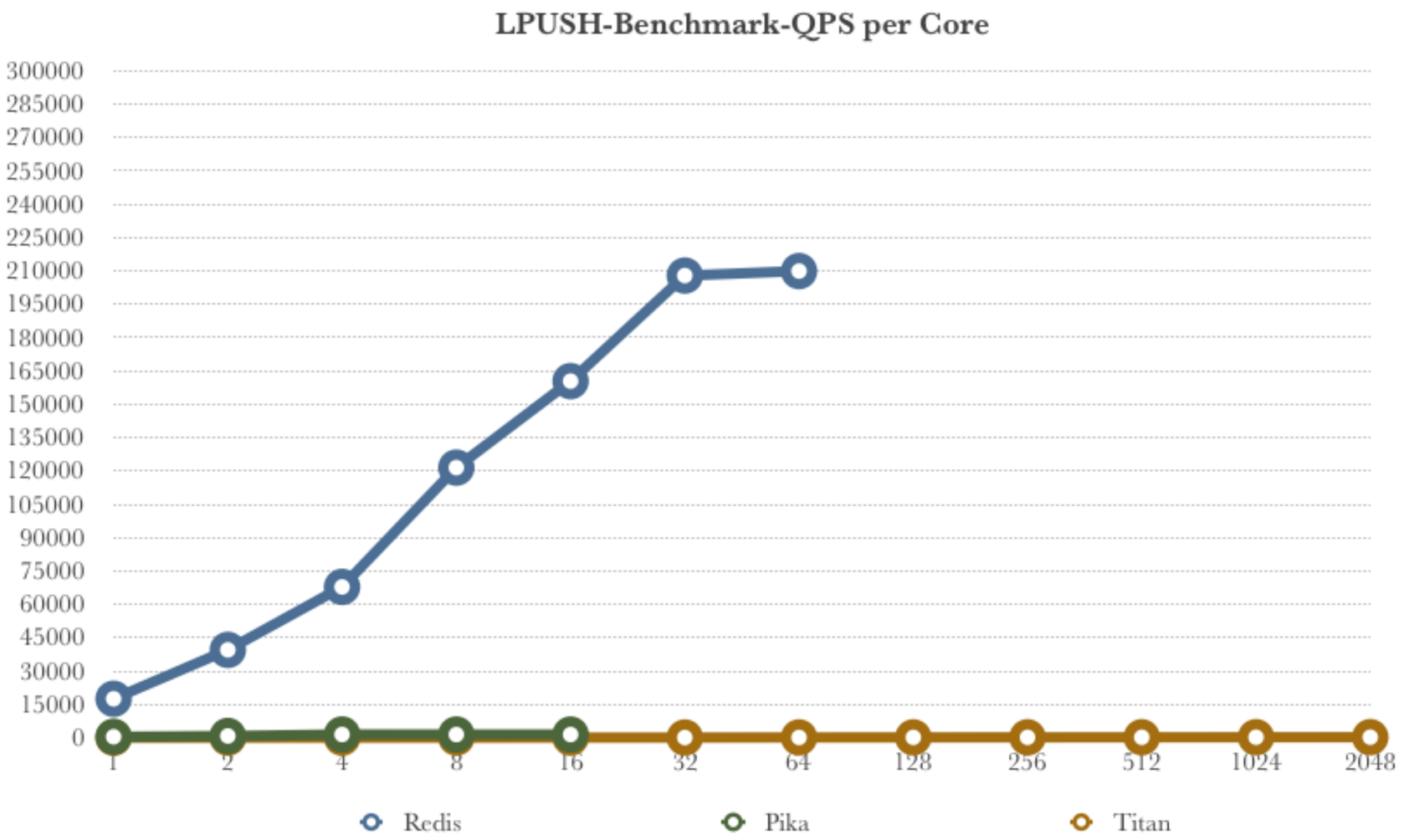
GET

./fperf -tick 1s -connection <Concurrency> -N 1000000 -server redis://10.12.132.16:6379 redis get key:__rand_int__

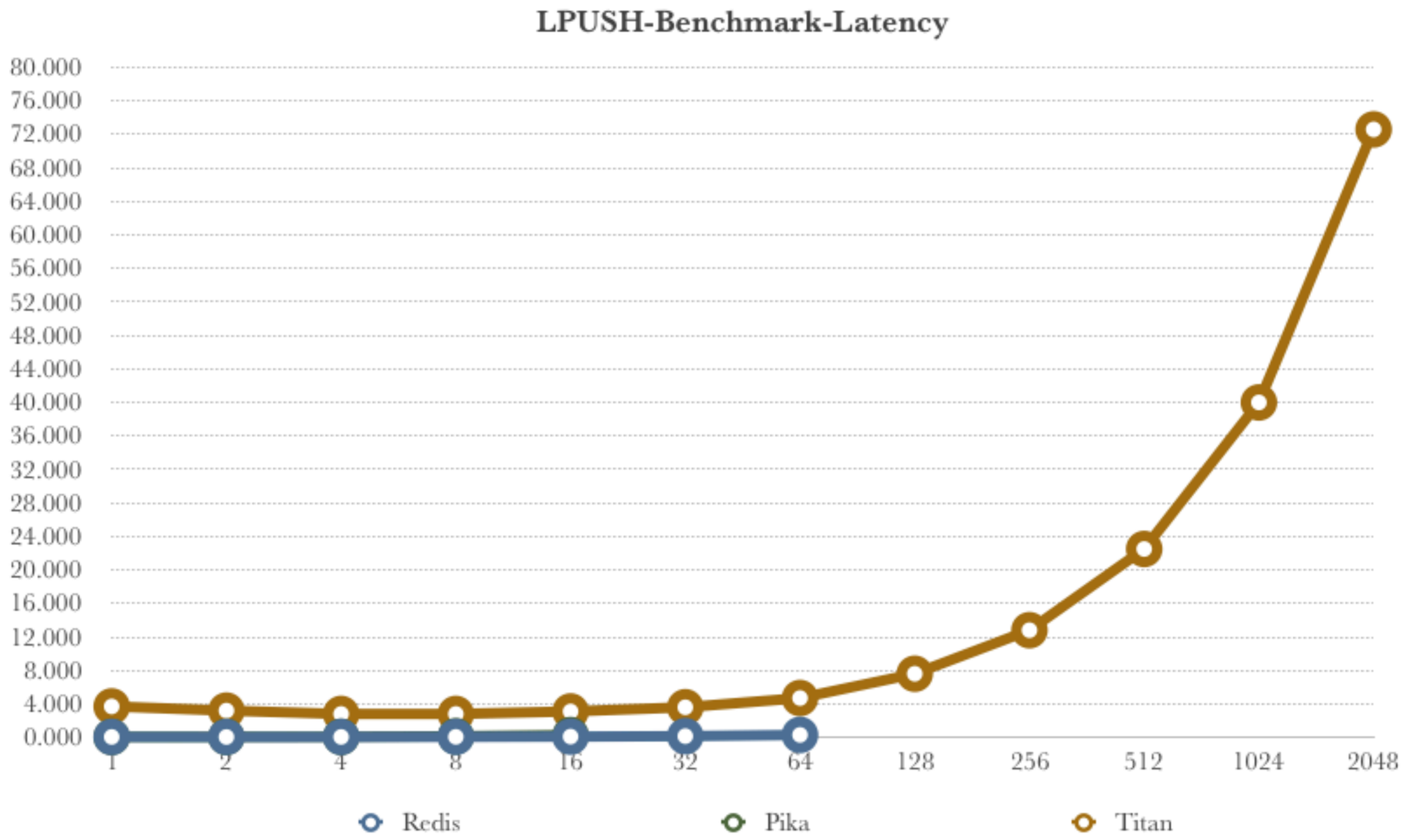
Titan vs Pika vs Redis



Titan vs Pika vs Redis



Titan vs Pika vs Redis



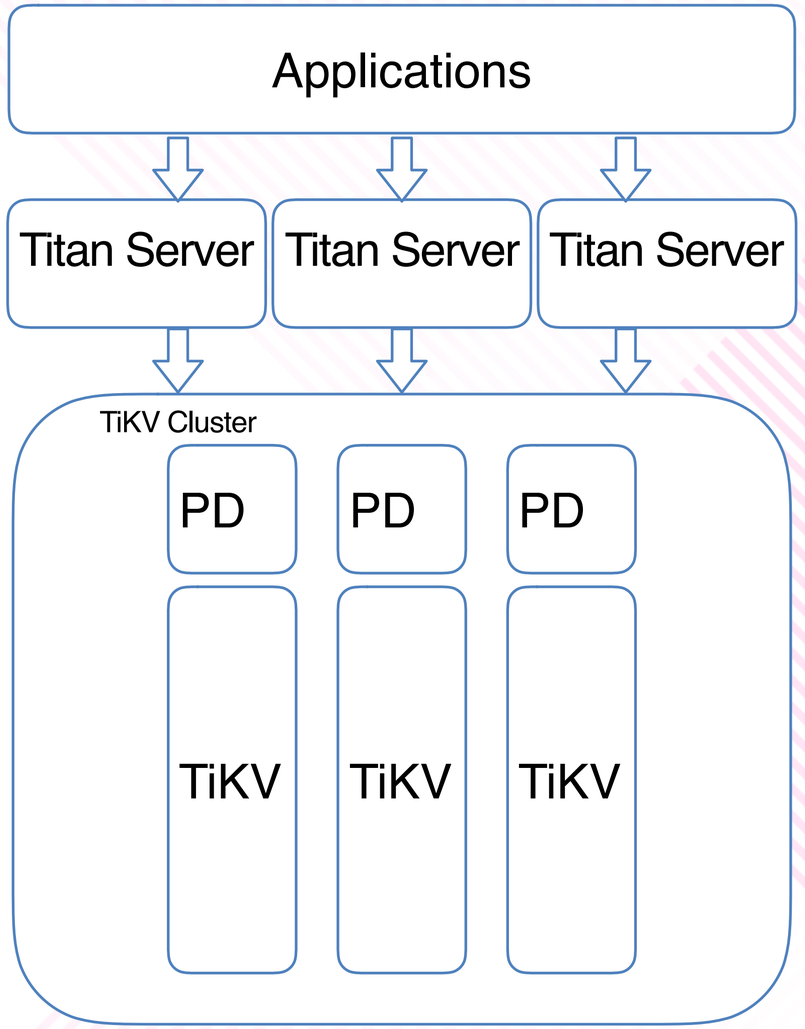
Titan vs Pika vs Redis

- Titan不是用来替换Redis的！！！
- QPS per core: Redis = 100xPika = 10xTitan

重新认识一下Titan

Titan是什么

- 兼容Redis协议
- 线性横向扩展
- 数据多副本高可用、强一致
- 分布式事务
- 海量数据存储



architecture of Titan

为什么要做Titan

- 可不敢再有故障了
- Redis扩容太痛苦
- 海量数据在内存，成本太高
- 分布式事务，你真的不需要吗?
- 多机房部署的死结：存储

高可用-SLA

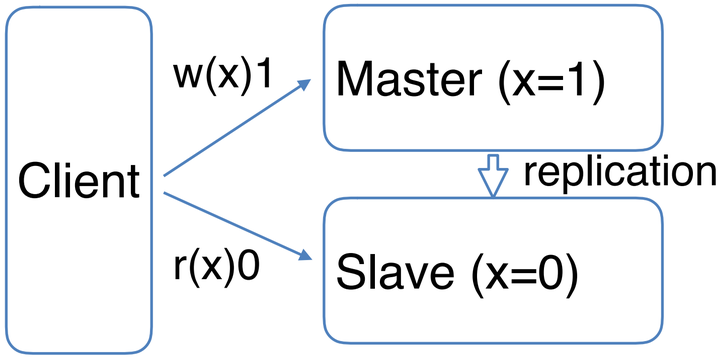
- downtime/unavailability

SLA	Daily	Weekly	Monthly	Yearly
99	14m 24s	1h 40m 48s	7h 18m 17.5s	3d 15h 39m 29.5s
99.9	1m 26.4s	10m 4.8s	43m 49.7s	8h 45m 57s
99.99	8.6s	1m 0.5s	4m 23s	52m 35.7s
99.999	0.9s	6.0s	26.3s	5m 15.6s

- 人工运维，99.99是不可能的

高可用-异步复制

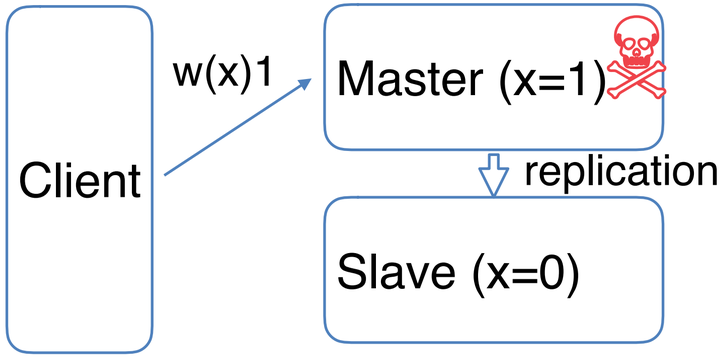
- FLP原理：异步系统，在任意故障下，无法达成共识
- 异步复制，读写分离时，可能读到旧数据



读到旧数据

高可用–节点挂掉

- 主节点挂掉，无法避免数据的丢失



数据丢失

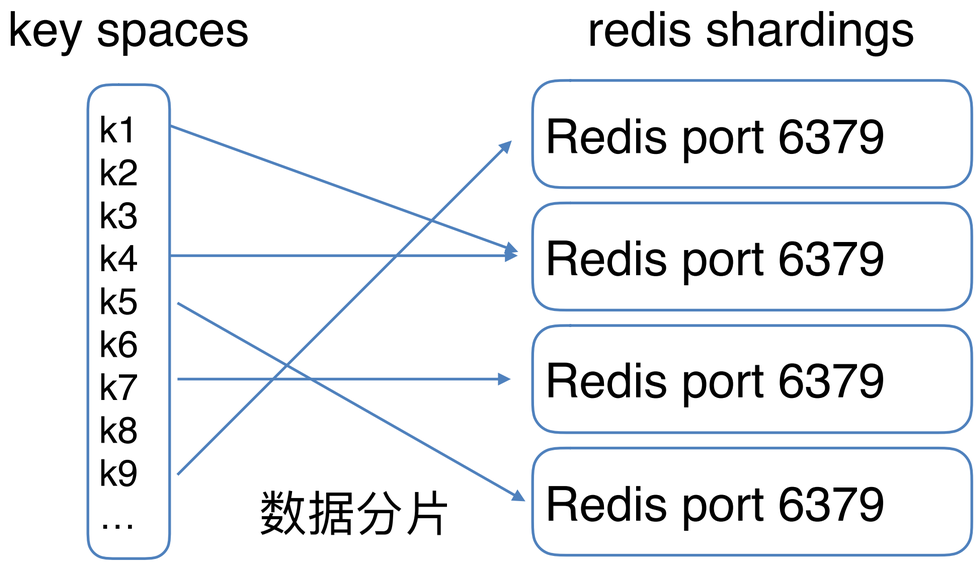
扩展性–需求

- 读扩展
- 写扩展
- 存储空间扩展

读可以通过添加Slave来扩展，但写请求和存储需求，则需要重新做数据分片

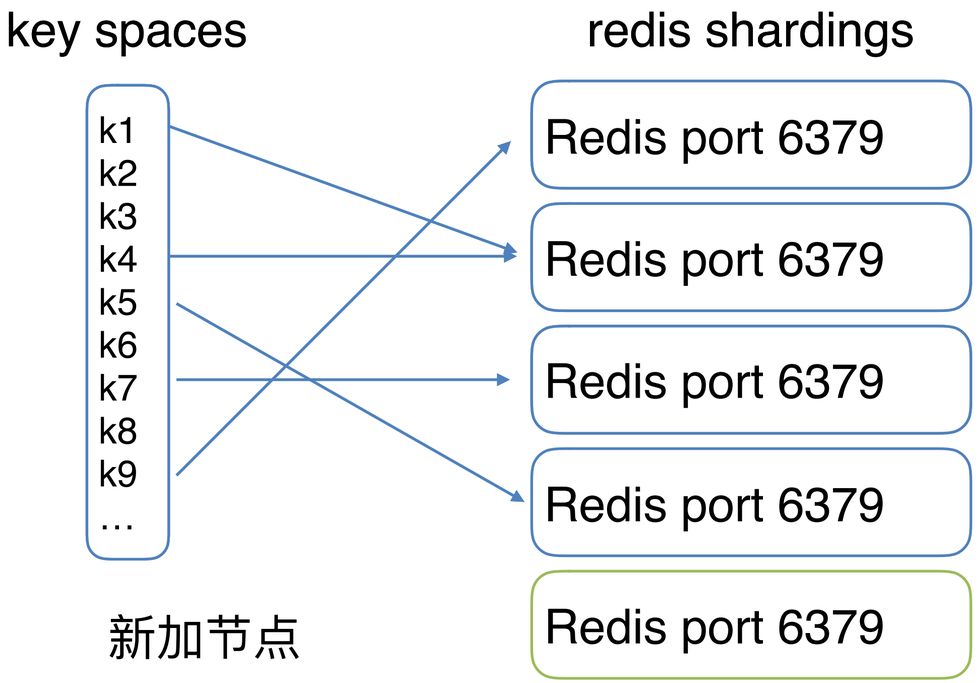
扩展性–数据分片

- 提前评估数据量及请求压力
- 选择适合分片的Key
- 通过一定策略，将Key分片到多个端口



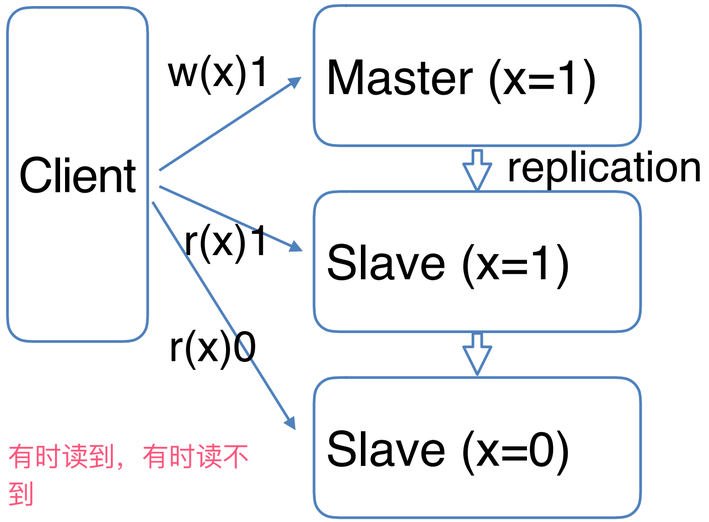
扩展性–增删节点

- 增删节点时，数据需要重新均衡



一致性-最终一致

- 刚写入的读不到
- 读到旧数据
- 有时读到，有时读不到



事务–原子性提交

- 添加好友

在A的好友列表中添加B

在B的好友列表中添加A

```
multi
hset A:friends B ts
hset B:friends A ts
exec
```

- 数据分片后，无法实现原子性

事务–并发安全

- 转账

```
watch A:balance
remain = Get(A:balance)
if remain > 100{
  multi
  set A:balance remain-100
  set B:balance remain+100
  exec
}
```

- 数据分片后，无法使用跨节点使用事务

多机房部署–还是单机房写

- Redis的主，还是在单机房
- 物理机故障，机房故障，网络故障如何选主?
  - 数据最新的Slave
  - 同一个IDC的Slave
  - 不同IDC的Slave

meitu美图

所有的难题，都交给了应用层

业务的代码，需要解决这些问题

meitu美图

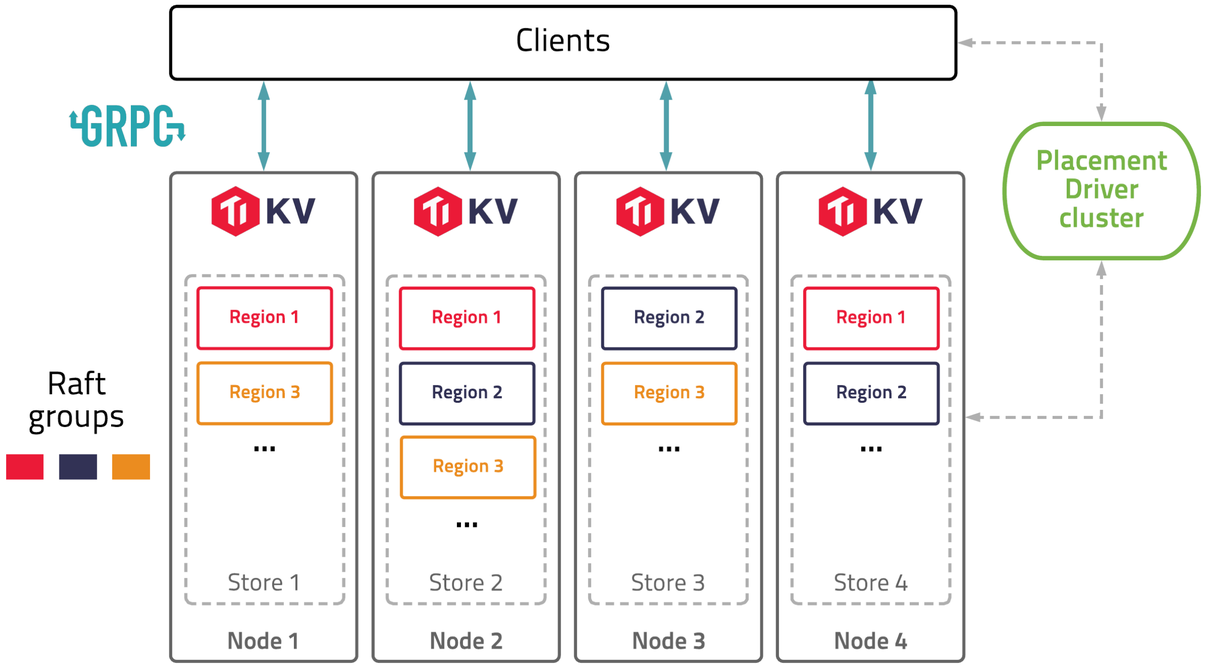
- 后端资源随时可能挂掉，如何做好fail-over
- 刚写入的数据可能读不出来，在某些场景下是否适用
- 资源挂掉后，已经写入的数据可能会丢
- 如何保证两个操作的原子性，如果数据不一致如何处理
- 数据分片，处理不同的分片维度
- 存储集群扩容如何过渡

Titan是什么–AGAIN

- 兼容Redis协议
- 线性横向扩展
- 数据多副本高可用、强一致
- 分布式事务
- 海量数据存储

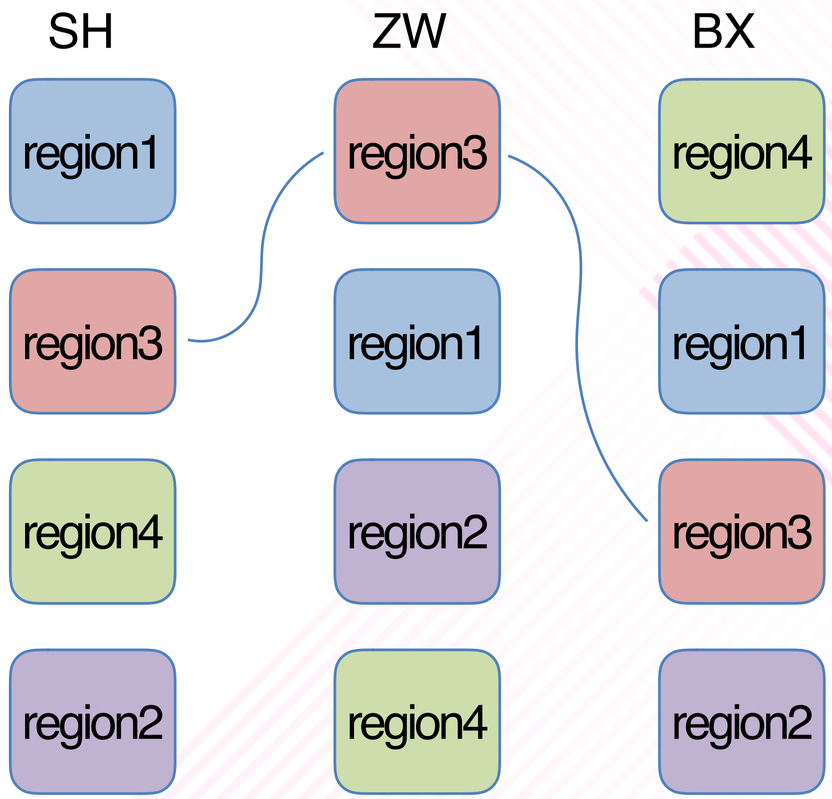
TiKV带来的可用性和扩展性

- Raft协议实现故障容忍的多副本共识
- Percolator事务模型实现线性一致性
- Region自动扩展



还有多机房呢?

- raft共识协议，容忍(n-1)/2个节点故障
- IDC分布：1-1-1，2-2-1



Titan与Redis

meitu美图

- Redis
  - 单机单核服务
  - 超高性能、超低延迟（1ms内）
  - 最终一致性
  - 主节点挂掉有丢数据的风险
- Titan
  - 分布式NoSQL存储
  - QPS可扩展，低延迟（20ms内）
  - 完整的ACID事务支持，强一致
  - Raft协议多副本共识，容忍小于半数节点挂掉

Titan与Pika

meitu美图

- Pika
  - 单机多线程服务
  - 数据存储在硬盘，高性能，低延迟
  - 最终一致性
  - 主节点挂掉有丢数据的风险
- Titan
  - 分布式NoSQL存储
  - 数据存储在硬盘，QPS可扩展，低延迟
  - 完整的ACID事务支持，强一致
  - Raft协议多副本共识，容忍小于半数节点挂掉

Titan、Redis和Pika

- 特性比较

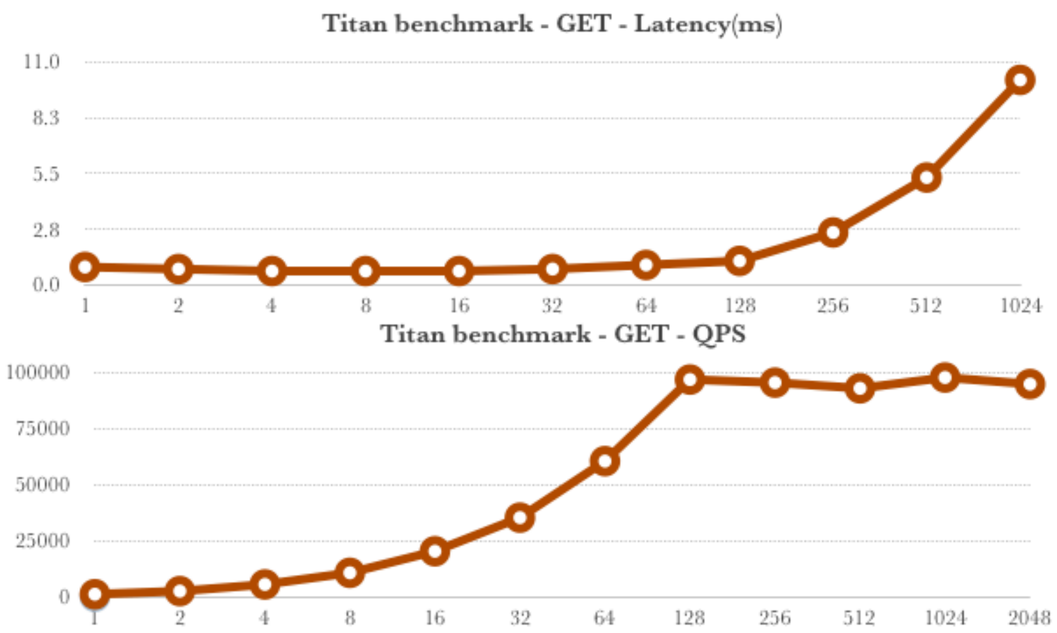
需求	延迟	QPS	可用性	一致性	扩展性	海量存储
Redis	★★★★★	★★★★★	★★	★★ 最终一致性	★★	★★
Titan	★★	★★ 可扩展	★★★★★	★★★★★	★★★★★	★★★★★
Pika	★★★	★★★	★★	★★ 最终一致性	★★	★★★★★

Titan、Redis和Pika

- 如何选择（只是相对比较）

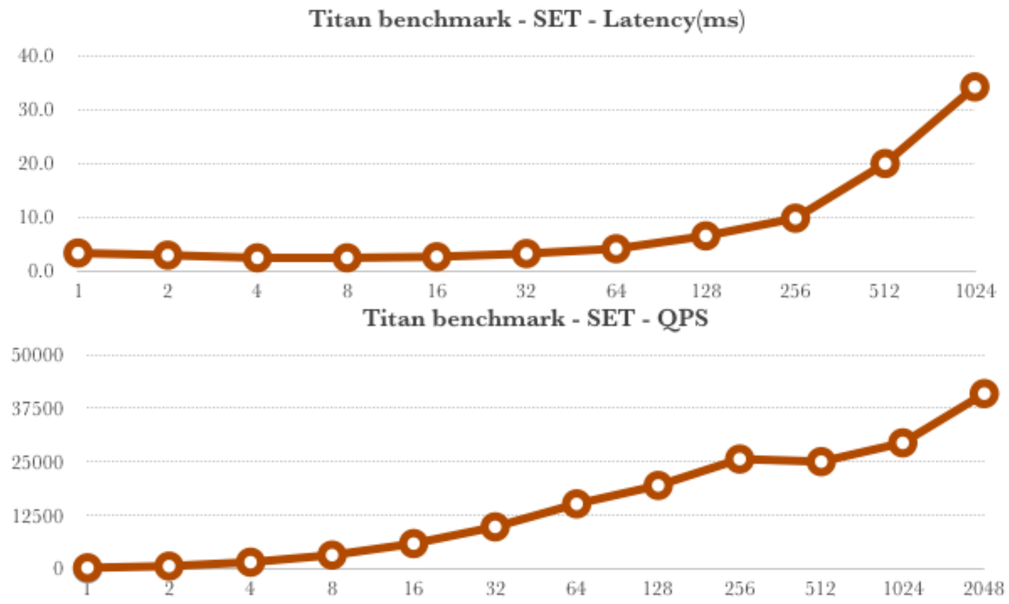
需求	延迟小于5ms	延迟小于20ms	数据量小于100G	数据量超过1T	在线伸缩	数据强一致	分布式事务	服务高可用
Redis	👍	👍	👍					
Titan		👍		👍	👍	👍	👍	👍
Pika	👍	👍	👍	👍				

Titan性能测试–STRING



说明：随机生成key，采用与set相同的随机序列，因此所有get都会命中存在的数据

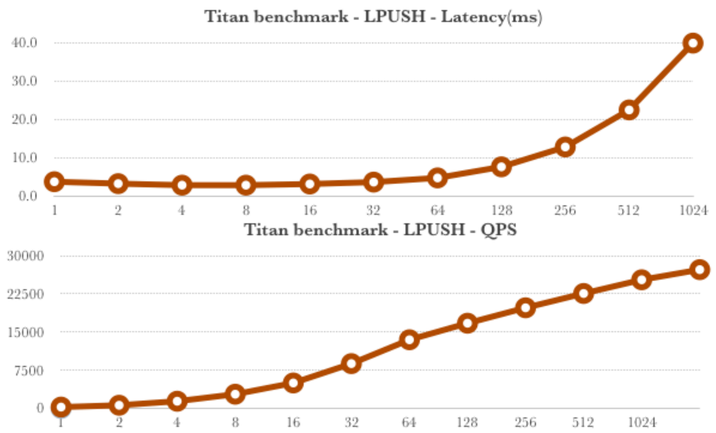
分析：在大约9.7w（97059）的QPS下，延迟可以控制在1.2ms左右，此时是服务表现最好的状态，随着压力增大，QPS增长不再明显，延迟升高，此时说明集群负载过高，考虑扩容



说明：系统中存在500G数据，但都是list前缀，因此操作kv时，使用不同的前缀，由于没有预先生成数据，在写入初期会主要集中在一个region，随着不断写入，数据会分布到不同的region，从而性能也会提升。

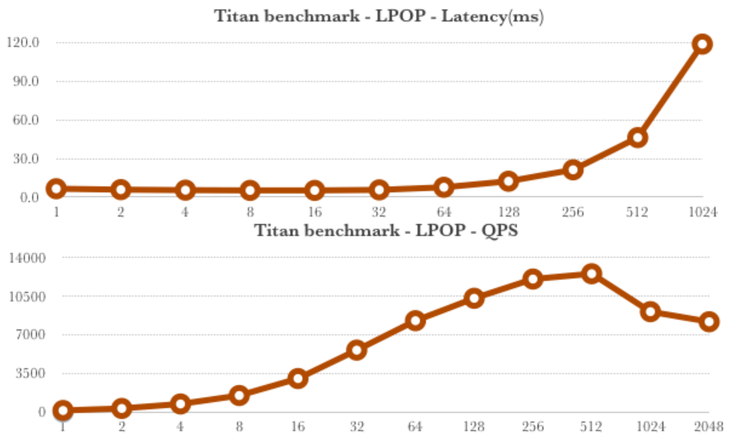
分析：在大约2.5w（25666）的QPS下，延迟可以控制在9.8ms左右，此时达到单个region的写入瓶颈，随着数据写入，性能达到4w QPS，延迟在48.8ms左后，随着参与的region越来越多，并发QPS还会有部分提升。

Titan性能测试–LIST



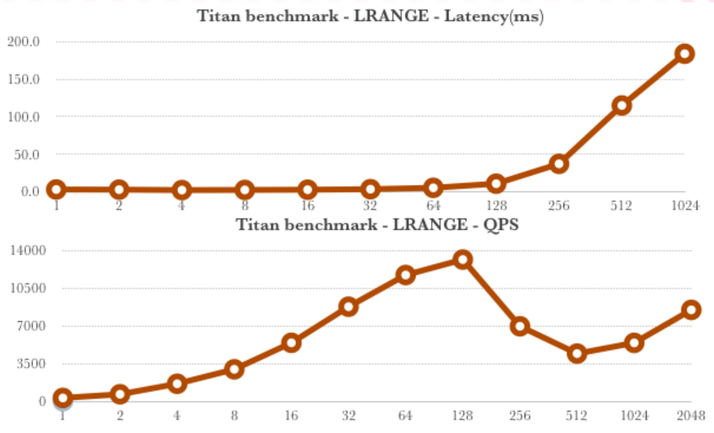
说明：前面已经写入500G数据，并且都是list相关的key，因此在测试list时，key的分布是相对比较均衡的

分析：在大约2w（19773）的QPS下，延迟可以控制在12.8ms左右，此时是服务表现最好的状态，随着压力增大，QPS增长不再明显，延迟升高，此时说明集群负载过高，考虑扩容



说明：随机生成**key**，并保证生成序列与**lpush**相同，这样可以确保**lpop**操作总能命中一个存在的**list**

分析：大约在1w（10319）QPS时，延迟在12.3左右，此时是服务表现最好的状态，之后随着压力增大，QPS提升不再明显，延迟升高，当并发过高时，性能波动较大，初步推断是Seek影响，这是因为当有大量并发Seek请求时，由于Seek一次返回256条数据，最终返回“256*实际并发”条数据，对网络延迟影响较大。



说明：随机生成**key**，并保证生成序列与**lpush**相同，这样可以确保**lrange**操作总能命中一个存在的**list**

分析：大约在1w（13189）QPS时，延迟在10.9左右，此时是服务表现最好的状态，之后随着压力增大，QPS提升不再明显，延迟升高，当并发过高时，性能严重下降，且波动较大，初步推断是Seek影响，这是因为当有大量并发Seek请求时，由于Seek一次返回256条数据，最终返回“256*实际并发”条数据，对网络延迟影响较大。

Titan性能测试

meitu美图

- QPS可扩展
  - 随核数增加近似线性扩展
  - 加节点则有更高的QPS
  - 考虑性能与成本的折中
- Latency 小于10~20ms
  - 选型时，优先考虑延迟需求
- 保证正确性后，我们会不遗余力持续优化性能

本次分享的内容–回顾

- 讲到的
  - QPS per Core： Redis = 100xPika=10Titan
  - Redis扩展困难，不支持分布式事务
  - Titan高可用可扩展强一致，支持分布式事务
  - Titan、Pika、Redis选型
  - 性能测试
- 没有覆盖的内容
  - TiKV底层原理
  - Titan的具体实现

本PPT来自Redis技术交流群2018年线下交流会 YouTube: https://youtu.be/lsX5qbosfE8
欢迎交流redis的开发和运维，群主每日精选一篇redis有关的文章在群里分享。 所有每日分享的文章都记录在以下星球（免费）。 希望入群的请扫描二维码加我的微信

meitu美图

Thanks, Any Questions?

https://github.com/meitu/titan