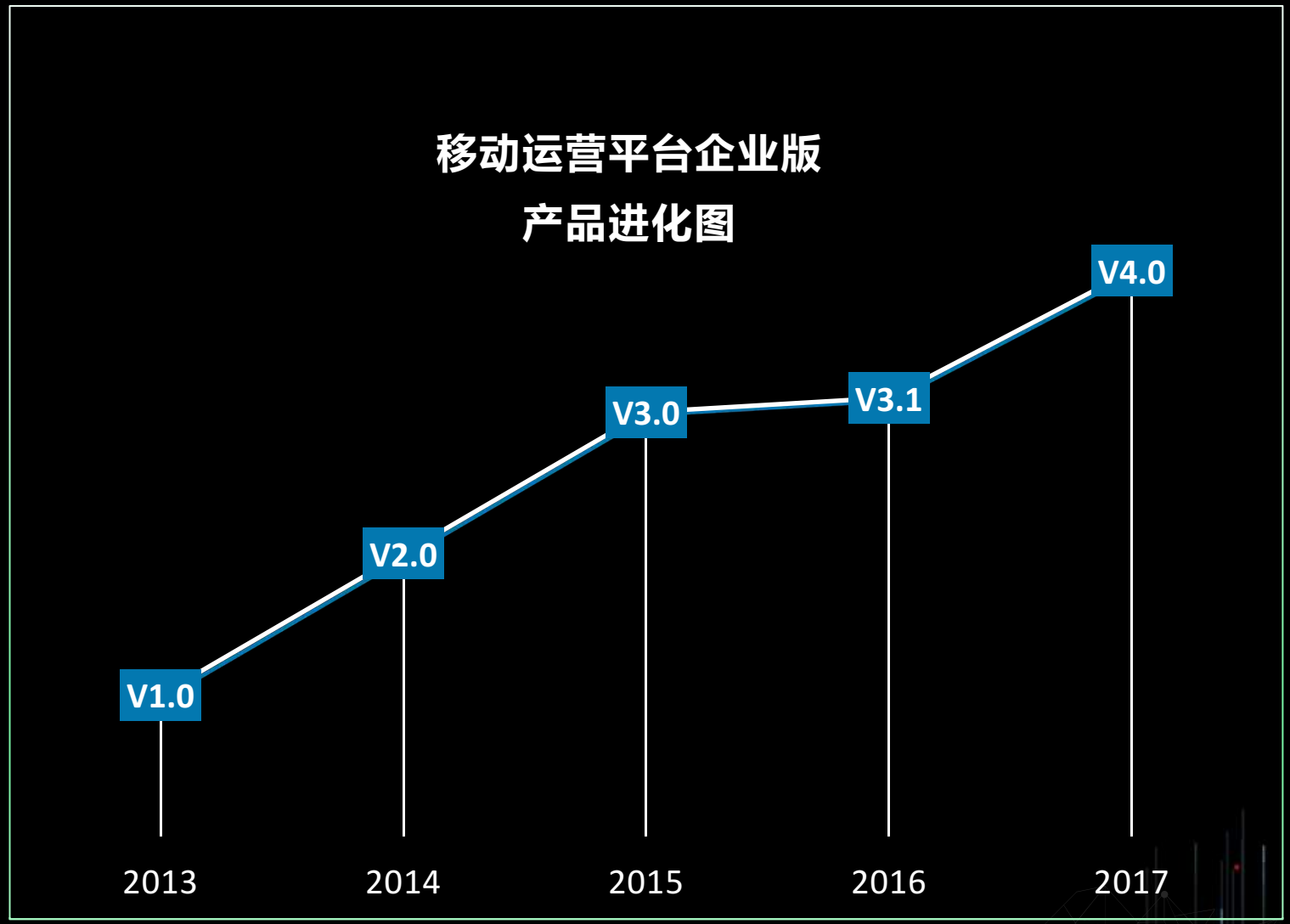
基于内存的分布式计算

主讲人：TalkingData 企业产品研发总监 周国平

背景简介

我们团队专注于**移动运营平台企业版**，客户的APP日活小的有不到10万，大的可以达到数千万。

我们致力于让**移动运营平台企业版**能够**弹性**地支持小、中、大型企业客户，系统**稳定**且**易于维护**。



问题

2015/05

我们团队负责移动运营平台企业版V3.0及后续版本迭代，当时的设计目标支撑500w日活 ，数据库使用MySQL。

…

事实证明移动运营平台企业版V3.0能够满足绝大多数企业客户的需求，能够稳定地支撑他们的业务。

2016/09

有一天，我们的一个客户，他们上线了几个日活量较大的APP，系统的整体日活达到了2000万，运维人员反映MySQL的binlog增长很快，快把剩余磁盘空间占满了。

问题分析 1

我们使用了bitmap索引技术保证移动运营各项指标（如日活、留存、转化漏斗等）的**实时**计算，因为bitmap索引高效且能节省存储空间，它能很方便地做指标的实时**排重**。

某APP某天0：0的初始活跃状态

bitmap

第1位 设备1	第2位 设备2	第3位 设备3	…	第N-1位 设备N-1	第N位 设备N
0	0	0	0	0	0

⬇ 当天8：10，设备3和设备N-1访问了APP

bitmap

1 设备1	2 设备2	3 设备3	…	N-1 设备N-1	N 设备N
0	0	1	0	1	0

⬇ 当天9：20，设备2和设备N-1访问了APP

bitmap

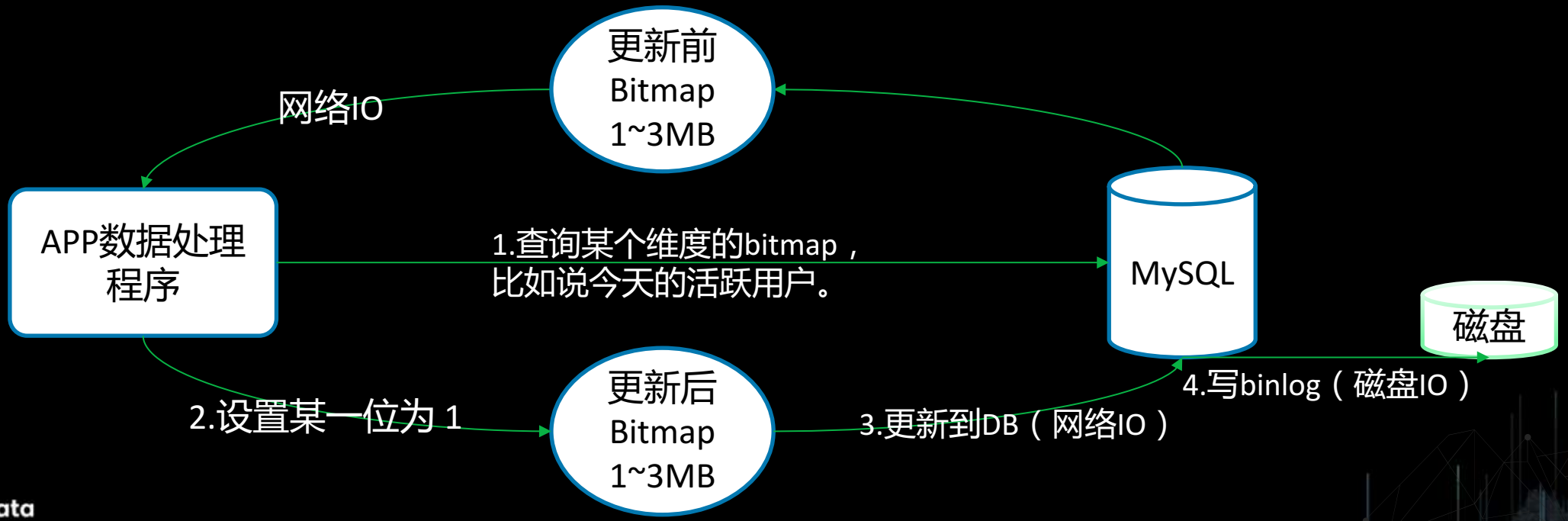
1 设备1	2 设备2	3 设备3	…	N-1 设备N-1	N 设备N
0	1	1	0	1	0

问题分析 2

我们使用MySQL存储bitmap索引，因为MySQL稳定且易于运维。

但是，MySQL本身在业务上是不支持bitmap类型的数据，不能够发送指令给MySQL让它把bitmap的某一位设置为1或0。

我们将bitmap对象作为blob类型存入MySQL，对bitmap索引的某一位更新时需要先从DB查询出来，更新之后再update到DB中。



问题分析 3

每个bitmap对象的大小从数百KB到数MB不等。

数据分析：

- 100w存量用户，随机60w日活，每个bitmap原始大小130KB，压缩后126KB
- 2000w存量用户，随机200w日活，每个bitmap原始大小2.6MB，压缩后1.5MB
- 1亿存量用户，随机500w日活，每个bitmap原始大小11.5MB，压缩后5.2MB

频繁地更新blob二进制数据，导致binlog数据量极大，从而导致存储空间不够用。这就是前面某客户出现瓶颈的原因所在。

问题域

大块（1M~10M）的二进制对象（约30w个bitmap对象）频繁读写，导致网络IO、磁盘IO等资源耗费巨大，MySQL binlog增长过快导致存储空间不够与浪费。

➡

我们需要考虑如何在分布式内存中计算以解决此类问题，解决方案需要满足：

- 第一个，缓存备份
- 第二个，性能高
- 第三个，易于维护

候选解决方案

方案一：替换MySQL，使用druid/rockdb等大数据组件

方案二：在MySQL前面引入redis缓存层，定时同步到MySQL

方案三：调研使用Apache Ignite组件

为什么不选用Druid/RockDB方案

我们在互联网研发线使用了此种方案，但调研下来不适合企业侧产品研发：

- 原来我们的系统运维工作很少，整个2000万日活体量的系统，也只需要1个运维人员；而换了Druid/rockdb，需要有较多的运维工作，等于放弃了我们原有的优势，增加了对客户运维人员的要求。
- Druid/RockDB 需要大量的服务器资源。

为什么不选用纯Redis缓存方案

- Redis在高并发下不能支撑对较大bitmap索引的频繁更新，单个bitmap索引平均能够达到2、3MB，而Redis的value达到1MB时吞吐量不到1000每秒，远远达不到要求的吞吐量。

- 另外一个重要的原因是Redis的事件机制有问题，expired和eviction事件拿不到缓存对象的值，这样会导致一旦缓存对象过期或被驱逐，我们无法把缓存对象更新到数据库。

- 对大块bitmap索引的频繁更新，导致存储空间耗用巨大，而且大块数据的复制延迟很严重，Redis变得不稳定。

为什么不选用Apache Ignite方案

- 使用了数据备份功能，因为频繁更新较大的bitmap索引，涉及到数据在不同节点的备份，导致系统不稳定，经常出现OOM问题。

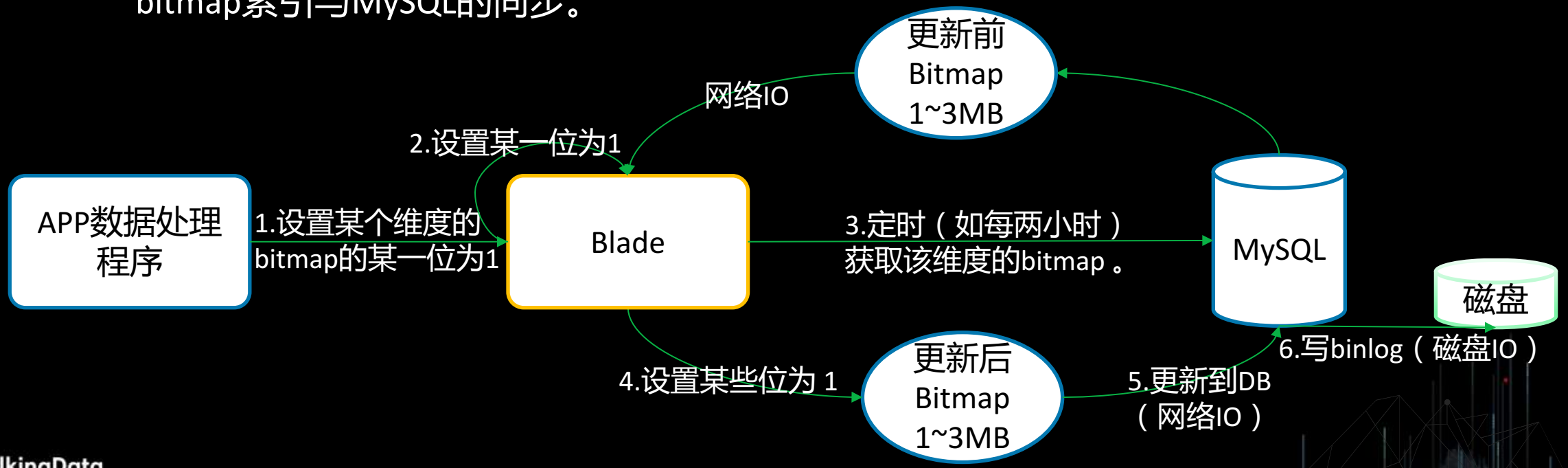
最终方案

权衡下来，我们决定基于Ehcache、 redis和zookeeper实现一套分布式缓存计算框架。

分布式缓存计算框架简介

代号 Blade，意在像锋利的刀锋一样有效解决企业产品的问题

- 它是一个bitmap索引计算的加速框架，专门针对海量bitmap索引计算时频繁更新DB操作进行优化。
- 它会把bitmap索引缓存在内存中，数据处理程序只需要告诉Blade修改哪个bit即可，无需把整条bitmap索引从DB拉取到本地更新后再写回DB，Blade会负责内存中的bitmap索引与MySQL的同步。



为什么Blade？

Blade能够减少更新MySQL中bitmap索引数据的频率，彻底解决大活跃客户出现的问题，对比之前提出的三种候选解决方案，它主要有如下优势：

- 基于成熟的组件（Ehcache、redis、zookeeper），易于维护；
- 对比单纯使用redis，不需要处理达2到3MB的bitmap索引数据，系统稳定，高并发有保证；
- 能够实现Apache Ignite的Expired、Eviction等缓存功能，频繁更新时不会出现OOM问题。

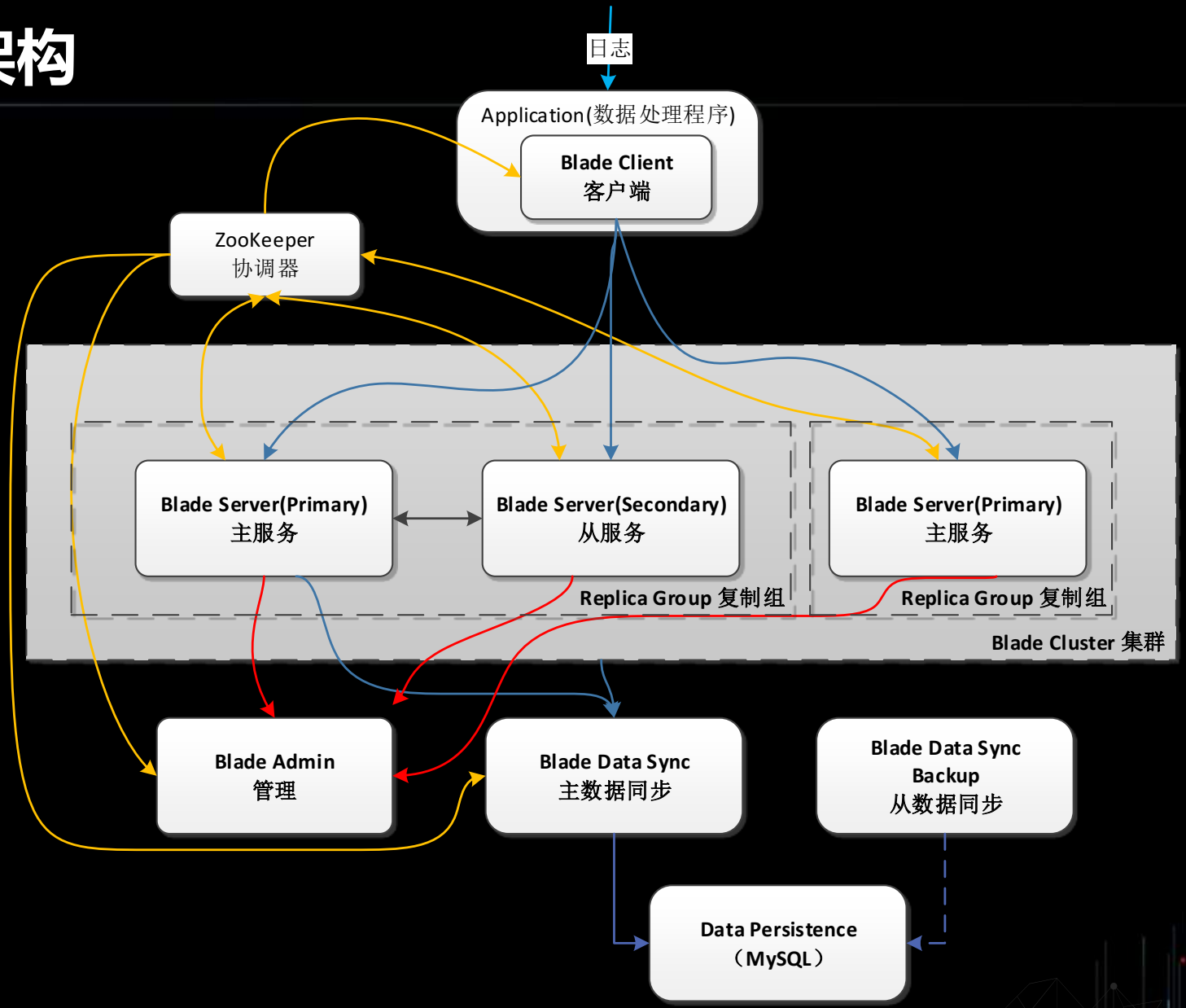
那么，Blade具体是怎么做到的呢？

Blade主要功能

- 提供分布式内存缓存，缓存对象支持Replication、 Expired、 Eviction等
- 提供UI监控、管理

Blade主要模块及架构

- Blade Client
  客户端
- Blade Cluster 集群
  Replica Group 复制组
  Blade Server 服务
- Blade Data Sync
  数据同步
- Blade Admin
  管理

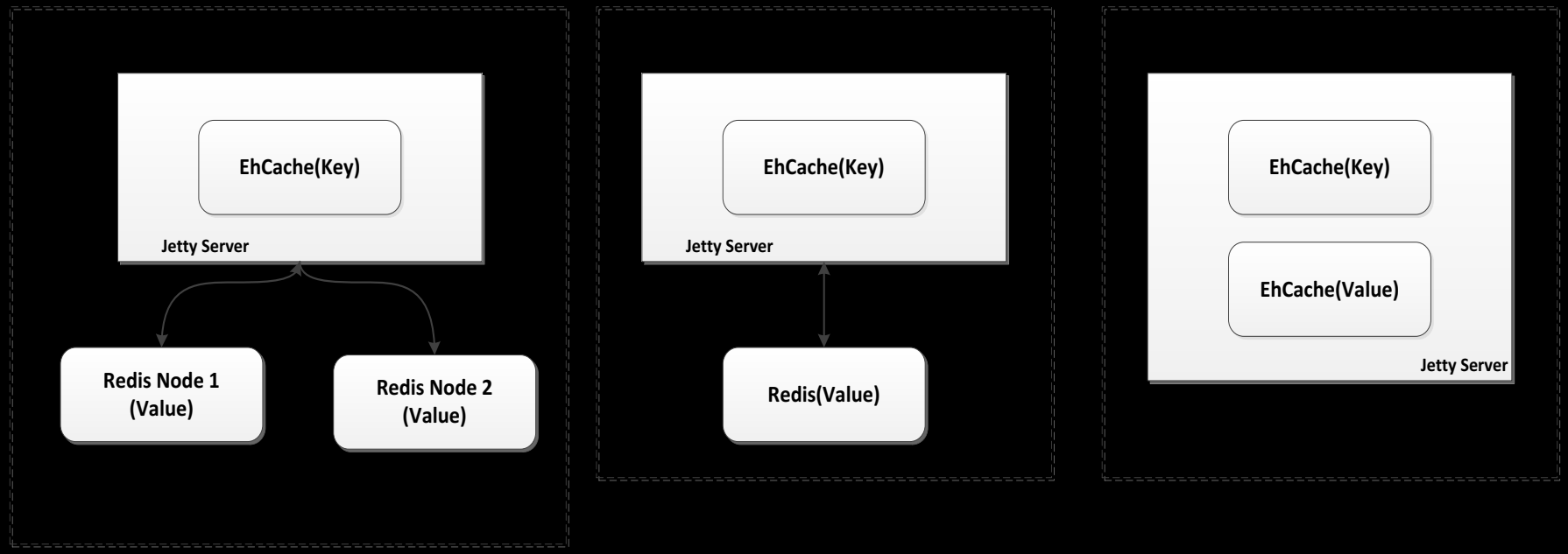


模块：Blade Client

- Blade Client是一个Jar，应用程序用这个Jar的API发起对Blade Cluster的调用。
- Blade Client在启动时，会从ZK上获取到整个Blade Cluster的运行情况。它会监听ZK事件，当Primary Node宕机或新的Secondary Node启动了，它可以立刻得到通知。
- Blade Client使用一致性Hash算法，把Key尽量均匀分布在整个Blade Cluster的所有ReplicaGroup上。

模块：Blade Server

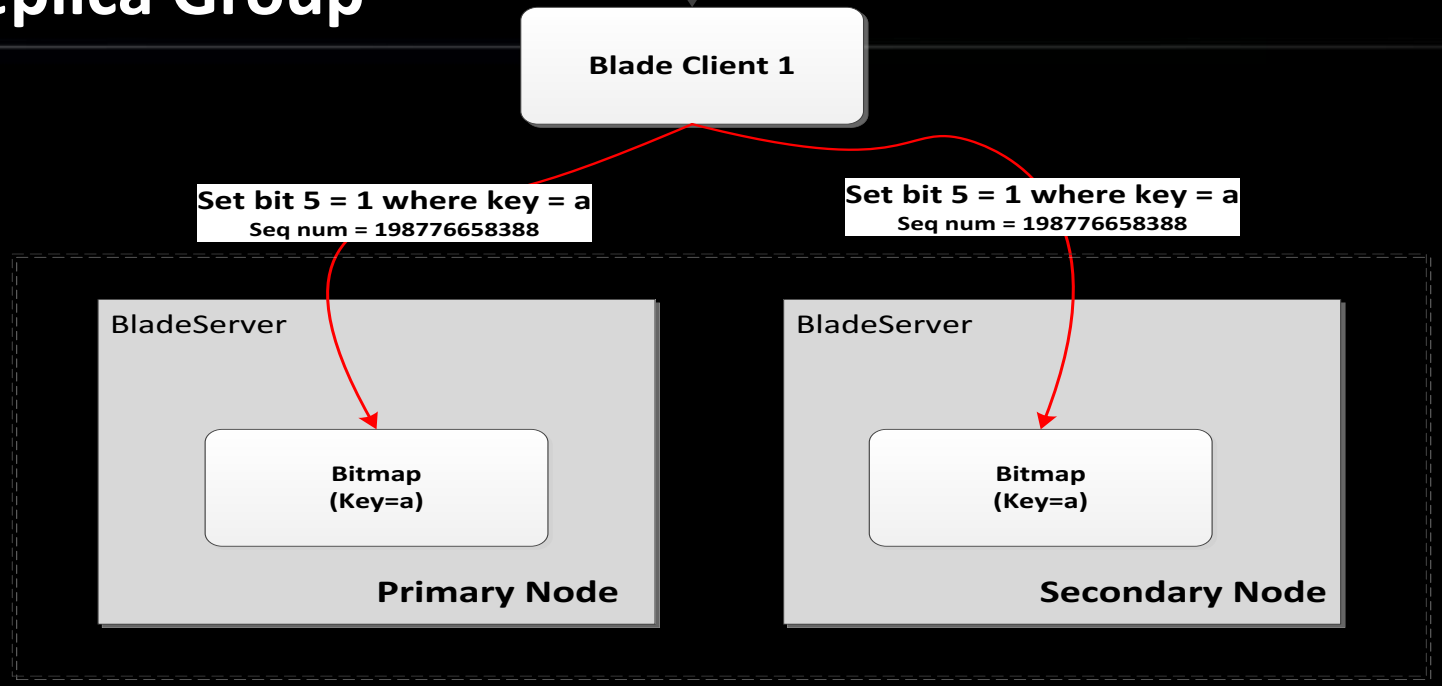
Blade Server可以有三种架构：



目前，我们选择第二种，即一个Blade Server包含一个Jetty和一个Redis。

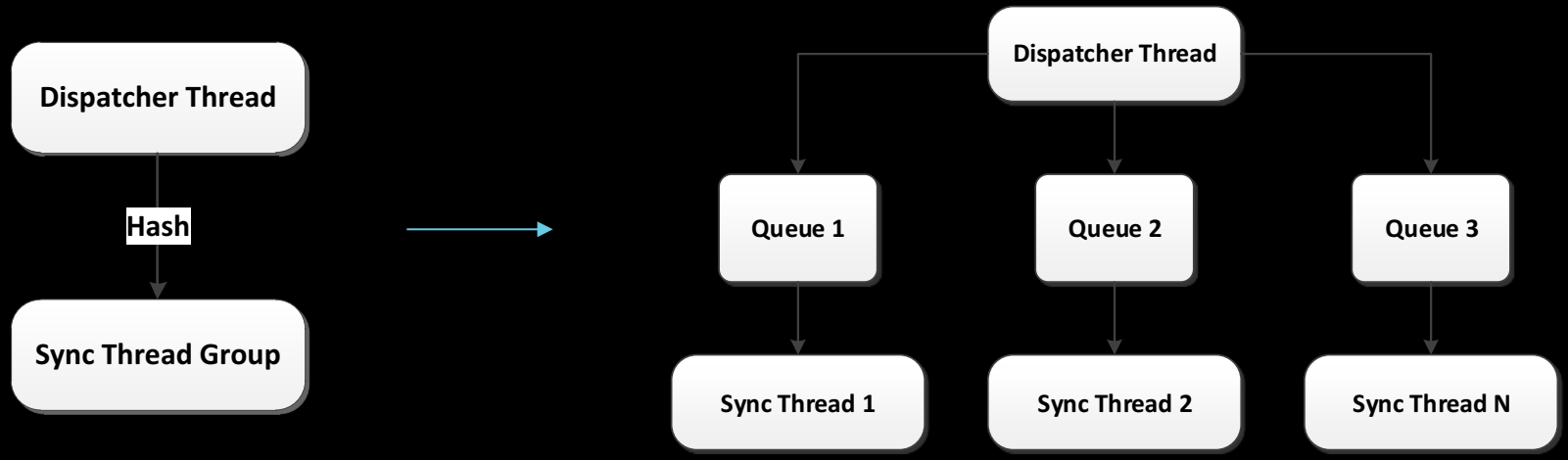
Jetty和Redis部署在一个节点上，能够保证在不占用系统带宽的情况下实现对bitmap索引的高速存取。Redis只作为一个大缓存使用，Redis本身的分片、集群、主备等功能全部都不使用，因为Blade本身架构就已经实现了这些功能。

模块：Replica Group



- Replica Group复制组主要用于防止单节点Blade Server宕机后，丢失缓存对象信息，同一个复制组中的Blade Server缓存对象会保持一致。
- 每个复制组中可以有1到N个Blade Server。一般来说，每个复制组中有2个Blade Server足以。需要注意的是，相同Replica Group中的Blade Server的硬件配置需要一致，最好分别运行在不同的主机上。

模块：Data Sync



- Data Sync负责把缓存中的bitmap索引定时同步到Data Persistence中。
- Data Sync采用主备架构，平时由主节点负责同步数据，当主节点挂掉时由从节点负责同步数据。
- Data Sync使用Dispatcher Thread遍历Primary Node的所有Key，Dispatch Thread会根据Key做Hash，把Key的同步操作分配给后面的Sync Thread Group中的一个线程来执行。
- 为了提高性能，每个Sync Thread都有一个Queue，异步处理请求。

模块：Blade Admin

Blade Admin主要用于监控整个Blade Cluster的情况：

- 每个Blade Server的情况
- Replica Group的情况
- Blade Data Sync的同步情况

Blade Admin还提供以下管理功能：

- 启动某个Replica Group的Primary Node、Secondary Node之间的同步
- 启动Blade Data Sync的同步

压力测试：测试场景

模拟2000w日活用户，每个用户产生20条日志，每条日志大小约为12KB，总计约4.8TB数据。

压力测试：资源配置

序号	角色	机型	配置（CPU/内存/硬盘）
1	Collector/report/queryengine/um/makedata	物理机	40c/128g/4.4T
2	Kafka1/Zookeeper1	物理机	40c/128g/4.4T
3	Kafka2/Zookeeper2	物理机	40c/128g/4.4T
4	Kafka3/Zookeeper3	物理机	40c/128g/4.4T
5	Kafka4	物理机	40c/128g/4.4T
6	Storm1/nimbus	物理机	40c/128g/4.4T
7	Storm2	物理机	40c/128g/4.4T
8	Storm3	物理机	40c/128g/4.4T
9	Storm4	物理机	40c/128g/4.4T
10	Storm5	物理机	40c/128g/4.4T
11	Storm6	物理机	40c/128g/4.4T
12	Storm7	物理机	40c/128g/4.4T
13	Storm8	物理机	40c/128g/4.4T
14	Storm9	物理机	40c/128g/4.4T
15	Storm10	物理机	40c/128g/4.4T
16	BladeServer1/2/3/4/5/6	物理机	40c/128g/4.4T
17	mysql-bitmap	物理机	40c/128g/4.4T
18	mysql-counter	物理机	40c/128g/4.4T
19	Redis/bladeAdmin/bladeDataSync/collector	物理机	40c/128g/4.4T

压力测试：测试结果（不使用Blade）

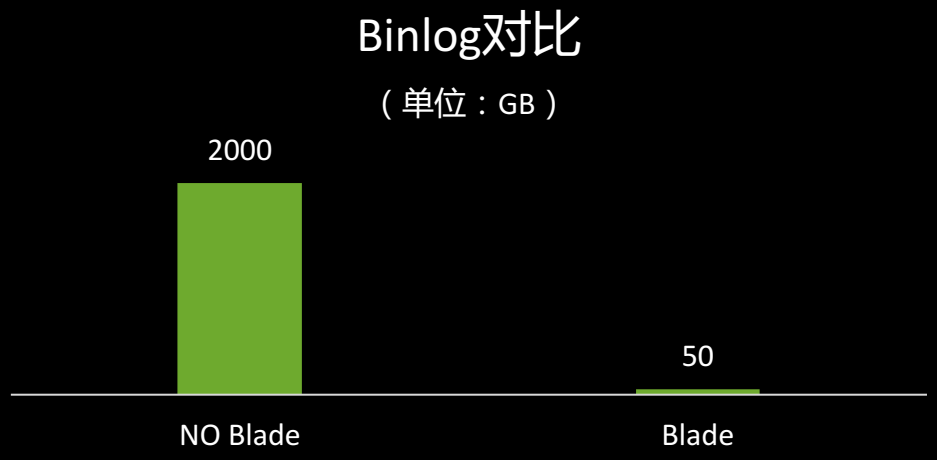
- MySQL binlog 2TB 左右
- 处理4.8T数据约需要70小时
- 不能支撑2000万日活

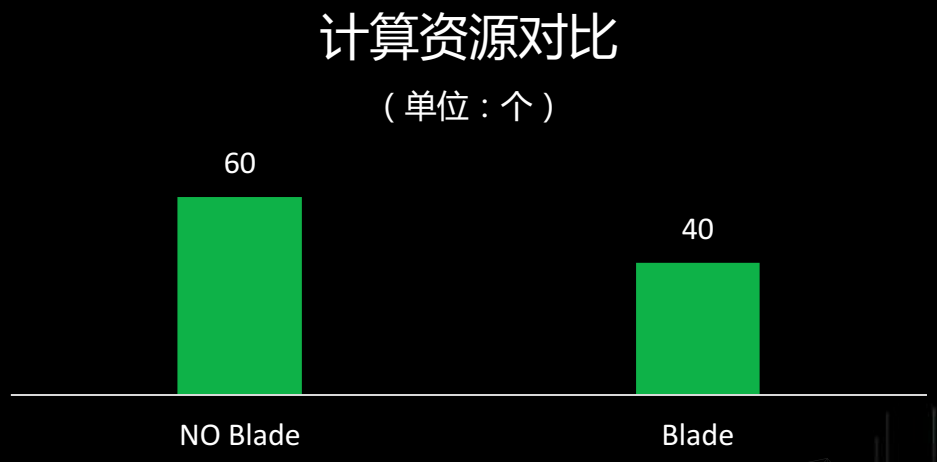
压力测试：测试结果（Blade架构）

- MySQL binlog 50GB 左右
- 处理4.8T数据约需要20小时
- 能够支撑2000万日活

压力测试：测试结果（对比）

- 同样机器资源下，Blade完全能够支撑2000万日活，而之前的架构不能。
- 两种数据处理方式，实时写MySql的binlog为分布式缓存同步DB（2小时同步一次）的近40倍。
- 对比之前某客户，在支撑同样2000w日活的情形下，为客户节省了近1/3的计算资源。






T11
2017
THANKS
TalkingData