Introduction to ZooKeeper

A Wait-free coordination for Internet-scale systems

yunxi

2016 年 11 月 2 日

Computer Science of SCNU

Table of contents

What is distributed system?

What is distributed system?

A distributed system consists of multiple computers that communicate through a computer network and interact with each other to achieve a common goal.

–Wikipedia

初识 ZooKeeper

初识 ZooKeeper

ZooKeeper 是一个开放源代码的分布式协调服务，由知名互联网公司雅虎创建，是 Google Chubby 的开源实现。ZooKeeper 的设计目标是将那些复杂且容易出错的分布式一致性服务封装起来，构成一个高效可靠的原语集，并以一系列简单易用的接口提供给用户使用。

初识 ZooKeeper

ZooKeeper 是一个典型的分布式数据一致性的解决方案，分布式应用程序可以基于它实现诸如数据发布/订阅、负载均衡、命名服务、分布式协调/通知、集群管理、Master 选举、分布式锁和分布式队列等功能。

初识 ZooKeeper

ZooKeeper 可以保证如下分布式一致性特性

- 顺序一致性
- 原子性
- 单一视图 (Single System Image)
- 可靠性
- 实时性

ZooKeeper 的设计目标

ZooKeeper 致力于提供一个高性能、高可用，具有严格顺序访问控制能力 (主要针对写操作的严格顺序性) 的分布式协调服务。高性能使得 ZooKeeper 能够应用于那些系统吞吐量有明确要求的大型分布式系统中，高可用使得分布式的单点问题得到了很好的解决，而严格的顺序访问控制使得客户端能够基于 ZooKeeper 实现一些复杂的同步原语。

- 1. 简单的数据模型
- 2. 可以构建集群
- 3. 顺序访问
- 4. 高性能

1. 简单的数据模型

ZooKeeper 使得分布式程序能够通过一个共享的、树形结构的名字空间来进行相互协调。这里所说的树形结构的名字空间，是指 ZooKeeper 服务器内存中的一个数据模型，其由一系列的 Znode 数据节点组成。其数据模型类似于一个文件系统，而 Znode 之间的层次关系，就像文件系统的目录结构一样。不过 ZooKeeper 将全量数据存储在内存中，以此来提高服务器的吞吐、减少延迟的目的。

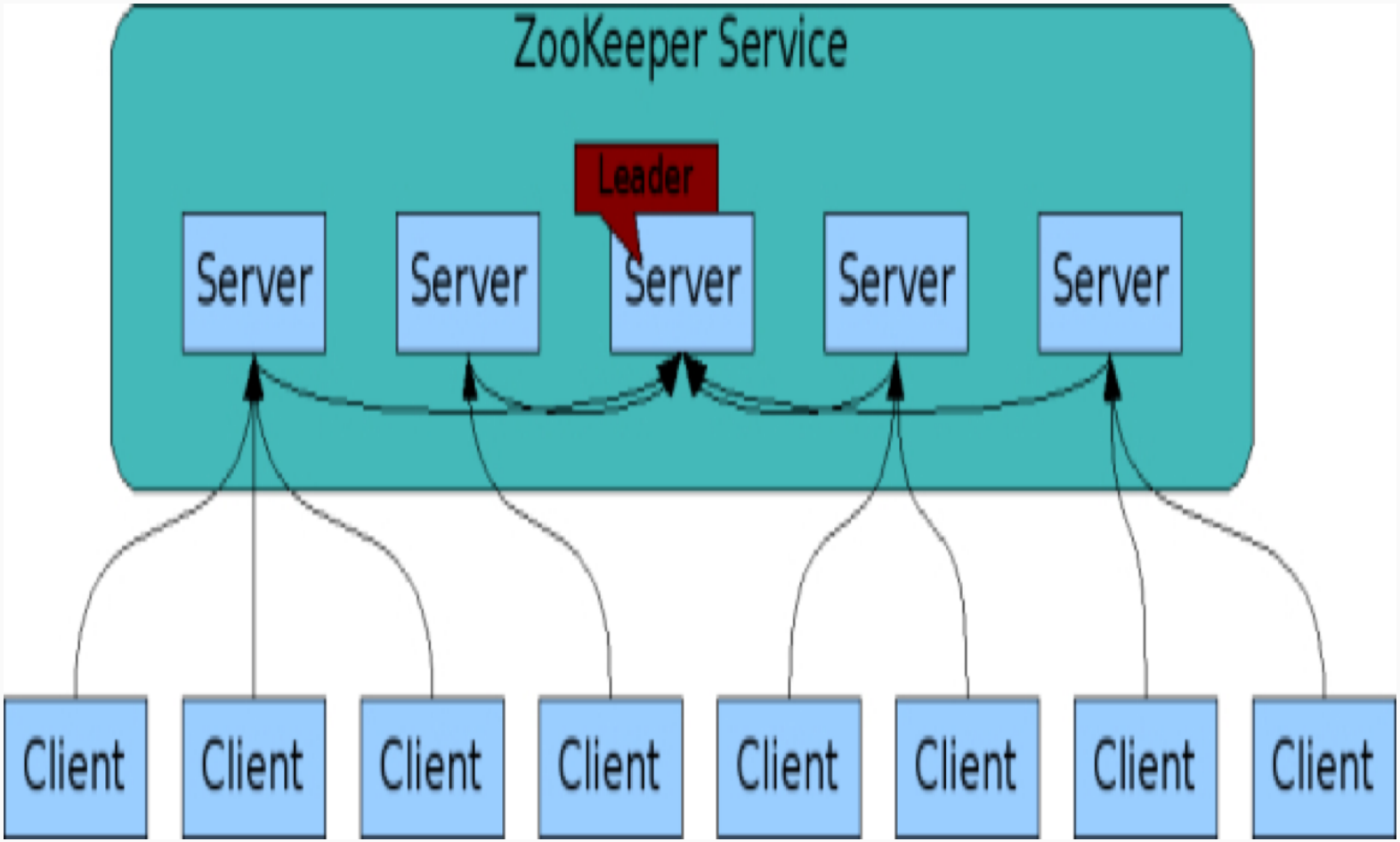
2. 可以构建集群

一个 ZooKeeper 集群通常由一组机器组成，一般 3-5 台机器可以组成一个可用 ZooKeeper 集群。如下图：

组成 ZooKeeper 集群的每台机器都会在内存中维护当前的服务器状态，并且每台机器之间都互相保持通信。只要集群中超过一半的机器能正常工作，整个集群就能对外服务。

ZooKeeper 的客户端程序会选择和集群中任意一台机器共同创建一个 TCP 连接，而一旦客户端和某台 ZooKeeper 服务器之间的连接断开后，客户端会自动连接到集群中的其他机器。

2. 可以构建集群


ZooKeeper Service
Leader
Server
Server
Server
Server
Server
Client
Client
Client
Client
Client
Client
Client
Client

3. 顺序访问

对于来自客户端的每个更新请求，ZooKeeper 都会分配一个全局唯一的递增编号，这个编号反映了所有事务操作的先后顺序，应用程序可以使用这个特性来实现更高层次的同步原语。

在 ZooKeeper 中事务是：能改变 ZooKeeper 服务器状态的操作，也称之为事务操作或更新操作，一般包括数据节点的创建与删除、数据节点内容更新和客户端会话创建与失效等操作。

4. 高性能

由于 ZooKeeper 将全量数据存储在内存中，并直接服务于客户端的所有非事务请求，因此它尤其适用于以读操作为主的应用场景。

ZooKeeper 系统模型

ZooKeeper 系统模型

- 集群角色
- 会话 (Session)
- 数据模型 (Znode)
- 事件监听器 (Watcher)
- ACL(Access Control Lists)

ZooKeeper 系统模型

集群角色

- Leader 为客户端提供读和写服务, 负责投票的发起和决议，更新系统状态
- Follower(Learner) 提供读服务，所有写服务都需要转交给 Leader 角色，参与 Leader 选举
- Observer(Learner) 提供读服务，不参与选举过程，一般是为了增强 zk 集群的读请求并发能力
- Client 请求发起

ZooKeeper 系统模型

图 1: 集群角色示意图

ZooKeeper 系统模型

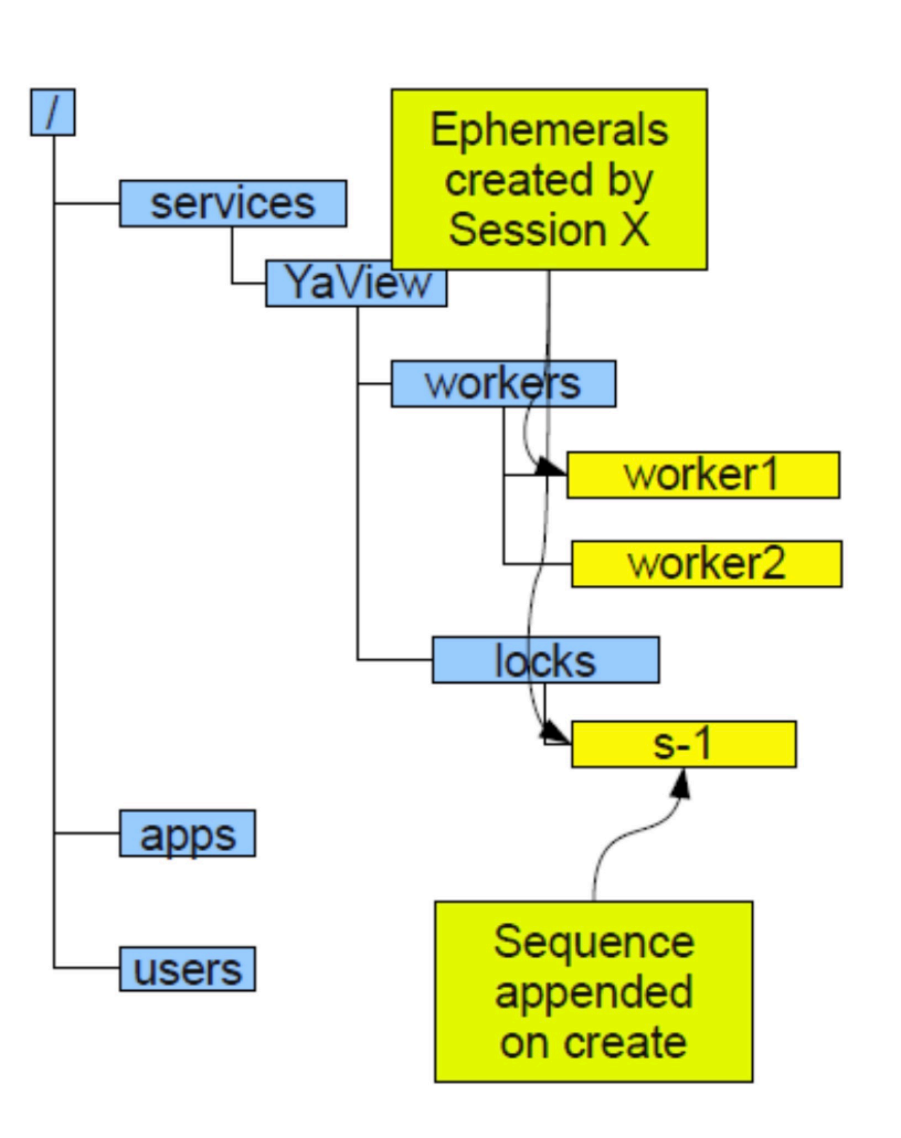
Session

Client 与 ZooKeeper 之间的通信，需要创建一个 Session，这个 Session 会有一个超时时间。因为 ZooKeeper 集群会把 Client 的 Session 信息持久化，所以在 Session 没超时之前，Client 与 ZooKeeper Server 的连接可以在各个 ZooKeeper Server 之间透明地移动。

在实际的应用中，如果 Client 与 Server 之间的通信足够频繁，Session 的维护就不需要其它额外的消息了。否则，ZooKeeper Client 会每 t/3 ms 发一次心跳给 Server，如果 Client 2t/3 ms 没收到来自 Server 的心跳回应，就会换到一个新的 ZooKeeper Server 上。这里 t 是用户配置的 Session 的超时时间。

ZooKeeper 系统模型

数据模型图



数据模型

- 分层结构
- 图中每个节点称为 Znode
- 每个 Znode 都有数据 byte[] 类型，也可以有子节点
- 节点路径
  - 斜线分隔: /Zoo/Duck
  - 没有相对路径
- 通过数据结构 stat 存储数据的变化、ACL 的变化和时间戳
- 数据发生变化时，版本号会递增 (即使更新操作并没有真正改变数据内容 a=0,a=0)
- 可以对 Znode 中的数据进行读写操作

ZooKeeper 系统模型

表 1: Stat 对象状态属性说明

状态属性	说明
czxid	数据节点被创建时的事务 ID
mzxid	节点最后一次被更新时的事务 ID
ctime	Create Time
mtime	Modified Time
version	数据节点的版本号
cversion	子节点的版本号
aversion	节点的 ACL 版本号
ephemeralOwner	创建该临时节点的会话的 sessionID
dataLength	数据内容的长度
numChildren	当前节点的子节点个数

ZooKeeper 系统模型

Znode Types

- 持久节点 (Persistent)
- 临时节点 (Ephemeral) (die when session expires, or node close connection)
- 临时顺序节点 (Ephemeral Sequential)
- 持久顺序节点 (Persistent Sequential)

Sequential

```
[zk: localhost:2181(CONNECTED) 58] ls /tmp-test-zk-path
[node0000000004, node0000000003, node0000000002, node0000000001, node0000000000]
```

非 Sequential

```
[zk: localhost:2181(CONNECTED) 69] ls /tmp-test-zk-path
[node, nodf, nodg, nodh]
```

ZooKeeper 系统模型

Znode Watches

A watch event is one-time trigger, sent to the client that set the watch, which occurs when the data for which the watch was set changes.

watch 可以理解为一个分布式的回调 (callback), 当 client 关心的 znodes 发生变化 (Watched event) 时,zookeeper 将会把消息传回到 client, 并导致 client 的消息处理函数得到调用.zk 的任何一个读操作都能够设置 watch。
例如:getData(), getChildren(), and exists()

watcher 特性总结

- 一次性
- 客户端串行执行
- 轻量

表 2: ZooKeeper API

操作	描述
create	创建一个 znode(必须要有父节点，创建时可以设置数据)
delete	删除一个 znode(该 znode 不能有任何子节点)
exists	测试一个 znode 是否存在并且查询它的元数据
getACL,setACL	获取/设置一个 znode 的 ACL
getChildren	获取一个 znode 的子节点列表
getData,setData	获取/设置一个 znode 所保存的数据

ZooKeeper 典型的应用场景

ZooKeeper 典型的应用场景

- 数据发布/订阅
- 负载均衡
- 命名服务
- 分布式协调/通知
- 集群成员管理 (Group membership)
- Master 选择
- 分布式锁

ZooKeeper 典型的应用场景

数据发布/订阅

数据发布/订阅系统，即所谓配置中心，顾名思义就是发布者将数据发布到 ZooKeeper 的一个或一系列节点上，供订阅者进行数据订阅，进而达到动态获取数据的目的，实现配置信息的集中式管理和数据的动态更新。

ZooKeeper 实现方式：客户端向服务端注册自己需要关注的节点，一旦该节点的数据发生变更，服务端就会向相应的客户端发送 Watcher 事件通知，客户端收到消息通知后，主动去服务端获取最新的数据。

数据发布/订阅事例，数据库切换

在进行配置管理之前，首先我们需要将初始化配置存储到 ZooKeeper 上，图中使用/app1/database_config 作为配置节点

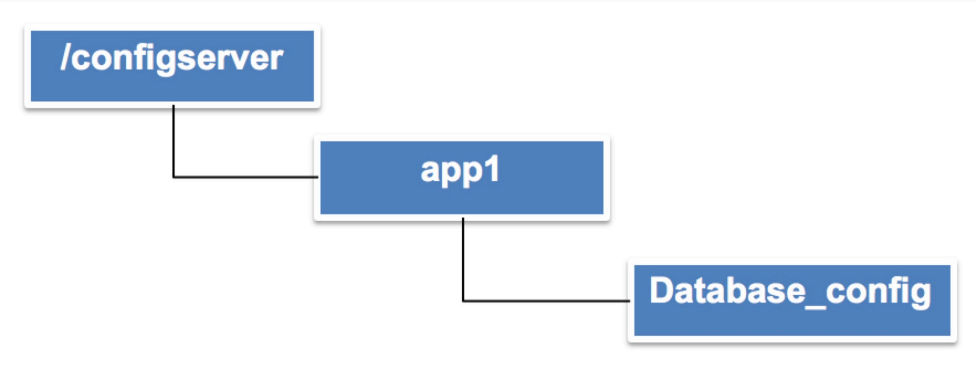


图 2: 配置管理 ZooKeeper 节点示意图

ZooKeeper 典型的应用场景

数据发布/订阅事例，数据库切换

数据库配置信息：

DBCP

dbcp.driverClassName=com.mysql.jdbc.Driver

dbcp.dbJDBCUrl=jdbc:mysql://1.1.1.1:3306/zzz

dbcp.characterEncoding=GBK

dbcp.username=genshuo

dbcp.password=123456

dbcp.maxActive=30

dbcp.maxIdle=10

dbcp.maxWait=10000

ZooKeeper 典型的应用场景

数据发布/订阅事例，数据库切换

- 配置获取：集群中的每台机器在初始化阶段，首先会从上面提到的配置节点桑读取数据库信息，同时客户端在配置节点上注册一个数据变更 Watcher 监听，一旦发生节点数据变更，所有订阅的客户端都能获取到数据变更通知。
- 配置变更：在系统运行过程中，可能会出现需要进行数据库切换的情况，这时候数据库信息变更，借助 ZooKeeper 我们只需要对 ZooKeeper 上的配置节点的数据内容进行更新，ZooKeeper 就能到将数据变更通知发送到各个客户端，每个客户端在获取到这个变更通知后，就可以重新进行最新的数据获取。

ZooKeeper 典型的应用场景

命名服务

命名服务 (Name Service) 也是分布式系统中比较常见的一类场景，在分布式系统中被命名的试题通常可以是集群中的机器、提供的服务地址等–我们统称为名字 (Name)。通过命名服务，客户端应用能够根据指定名字来获取资源的实体，服务地址和提供者的信息等。在分布式环境中，上层仅仅需要一个全局唯一的名字，类似于数据库中的唯一主键。下面主要讨论 ZooKeeper 来实现分布式全局唯一的 ID 的分配机制

命名服务

通过调用 ZooKeeper 创建节点的 API 可以创建一个顺序节点，并且在 API 返回值中会返回这个节点的完整名字。

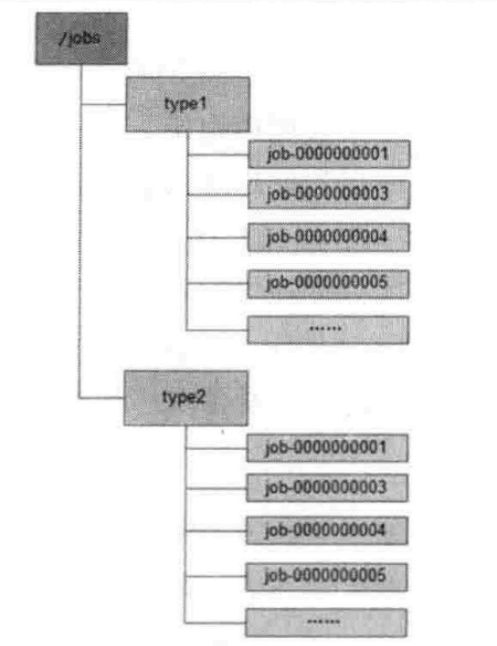


图 3: 全局唯一 IDE 生成 ZooKeeper 节点示意图

ZooKeeper 典型的应用场景

命名服务

- 1. 所以客户端都会根据自己的任务类型，在指定类型任务下 create() 创建一个顺序节点，例如“job-“节点。
- 2. 节点创建完毕后，create() 接口返回一个完整的节点名，如“job-0000000003”
- 3. 客户端拿到这个返回值后，拼接上 type 类型，如“type2-job-0000000003”，这便可以作为一个全局唯一 ID
- 4. 相比于 JAVA 中的 UUID，不紧解决了长度过长，且含义不明的问题

ZooKeeper 典型的应用场景

Master 选择

- 根据 create()API 的强一致性，能够保证在分布式高并发情况下节点的创建一定能全局唯一，即 ZooKeeper 保证无法创建一个已经存在的数据节点。
- 1. 首先会在 ZooKeeper 上创建一个日期节点，如“2016-11-01”
- 2. 客户端每天会定时往 ZooKeeper 上创建一个临时节点，如/master_election/2016-11-01/binding，在这个过程中只有一个客户端可以创建成功，那么这个客户端所在的机器就成为了 Master。
- 3. 同时，其他没有在 ZooKeeper 上成功创建节点的客户端，都会在节点/master_election/2016-11-01 上注册一个子节点变更的 Watcher，用于监控当前的 Master 机器是否存活，一旦挂了，其余的客户端会重新进行 Master 选举。

ZooKeeper 典型的应用场景

分布式排他锁

分布式锁是控制分布式系统之间同步访问共享资源的一种方式。
排他锁又称为写锁、独占锁，是一种基本的锁类型。如果事务 T1 对数据对象 O1 加上了排它锁，那么在整个加锁期间，只允许事务 T1 对 O1 进行读取和更新操作，其他任何事务都不能再对这个数据对象进行任何类型的操作–直到 T1 释放了排他锁。

ZooKeeper 典型的应用场景

排他锁

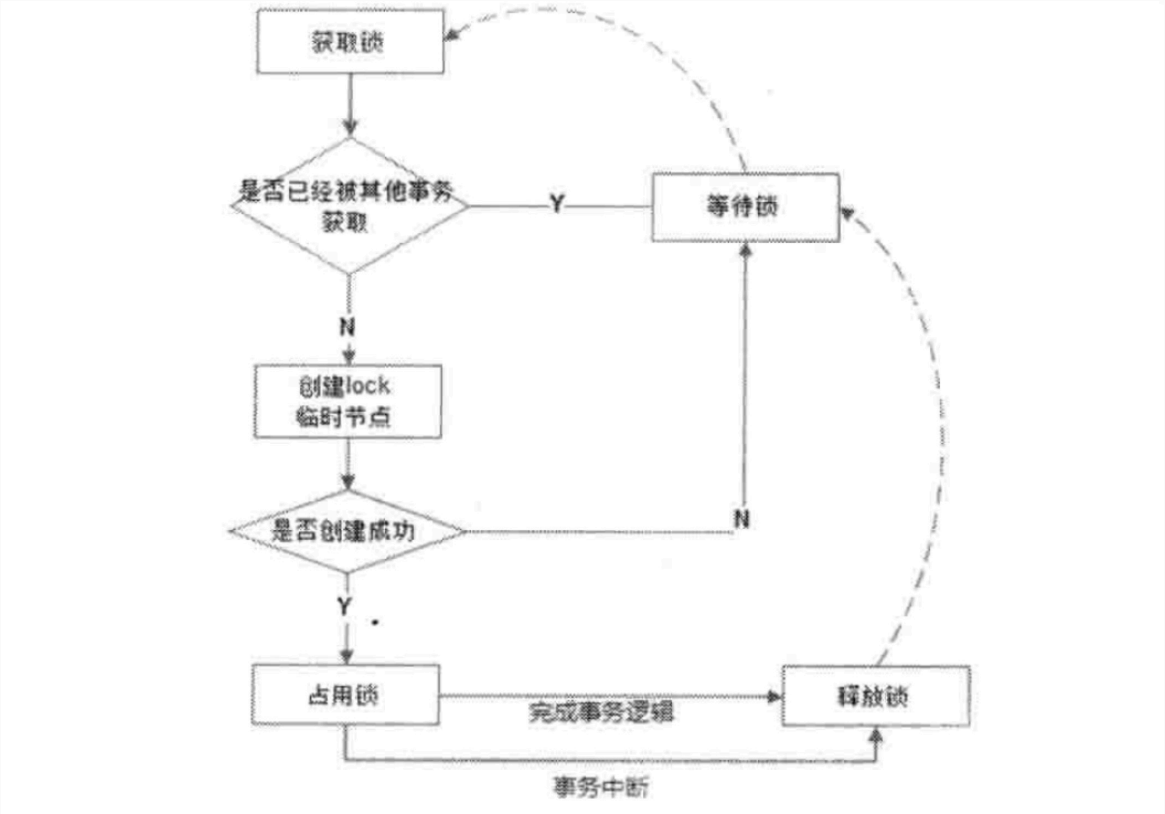


图 4: 排他锁的流程图

ZooKeeper 典型的应用场景

定义锁

通过在 ZooKeeper 上的数据节点表示一个锁，例如/exclusive_lock/lock 节点就可以被定义为一个锁。

- 需要获取排他锁时，所以客户端会尝试通过 create() 接口在/exclusive_lock 节点下创建临时子节点/exclusive_lock/lock。
- ZooKeeper 会保证在所有客户端中，最终只有一个客户端创建成功，该客户端接获取了锁。
- 同时所以没有获取到锁的客户端就需要在/exclusive_lock 节点上注册一个子节点变更的 Watcher 监听，以便于实时监听到 lock 节点的变更情况。

释放锁

在“定义锁”部分，我们提到/exclusive_lock 是一个临时节点，因此在一下两种情况下，都可能释放锁。

- 当前获取锁的客户端机器发生宕机，那么 ZooKeeper 上的这个临时节点会被移除
- 正常执行完成业务逻辑后，客户端会自动将创建的临时节点删除

无论在那种情形下移除了 lock 节点，ZooKeeper 都会通知所有在/exclusive_lock 节点上注册了节点变更 Watcher 监听的客户端。接受到通知后，再次重新发起分布式获取。

ZooKeeper 在分布式系统中的应用

ZooKeeper 在大型分布式系统中的应用

- Hadoop
- HBase
- ...

ZooKeeper 在大型分布式系统中的应用

Hadoop

- Hadoop 简介
- ZooKeeper 在其中的应用
- ...

Hadoop 简介

Hadoop生态圈	
Common	分布式文件系统和通用I/O的组件与接口（序列化，Java RPC和持久化数据结构）
Avro	支持高效的跨语言RPC和持久数据存储的序列化系统
MapReduce	分布式数据处理模型和执行环境，运行在大型商用机集群
HDFS	分布式文件系统，用于大型商用机集群
PIG	数据流语言和运行环境，检索大型数据集，Pig运行在MapReduce和HDFS的集群上
Hive	分布式、按列存储的数据仓库。Hive管理HDFS中存储的数据，并提供基于SQL的查询语言（由运行时引擎翻译成MapReduce作业）
HBase	分布式、按列存储的数据库。HBase使用HDFS作为底层存储，同时支持MapReduce的批量式计算和点查询（随机读取）
ZooKeeper	分布式、可用性高的协调服务。提供类似分布式锁的基础服务
Sqoop	在数据库和HDFS之间高效传输数据的工具

图 5: Hadoop 生态圈

ZooKeeper 在大型分布式系统中的应用

Hadoop

Hadoop 中，ZooKeeper 主要用于实现 HA(High Availability), 在 HDFS 的 NameNode 与 YARN 的 ResourceManager 都是基于此 HA 模块来实现自己的 HA 功能的。
接下来主要以 Cloudera 5.X 发布版本为例，围绕 YARN 中 ZooKeeper 的使用场景介绍。

YARN

YARN 是 Hadoop 为了提高计算节点 Master(JT) 的扩展性，同时为了支持计算模型和提供资源的细粒度调度而引入的全新一代分布式调度框架。YARN 上可以支持 MapReduce 计算引擎及其他如 Spark、Storm 等。

其架构体系如图：

ZooKeeper 典型的应用场景

YARN

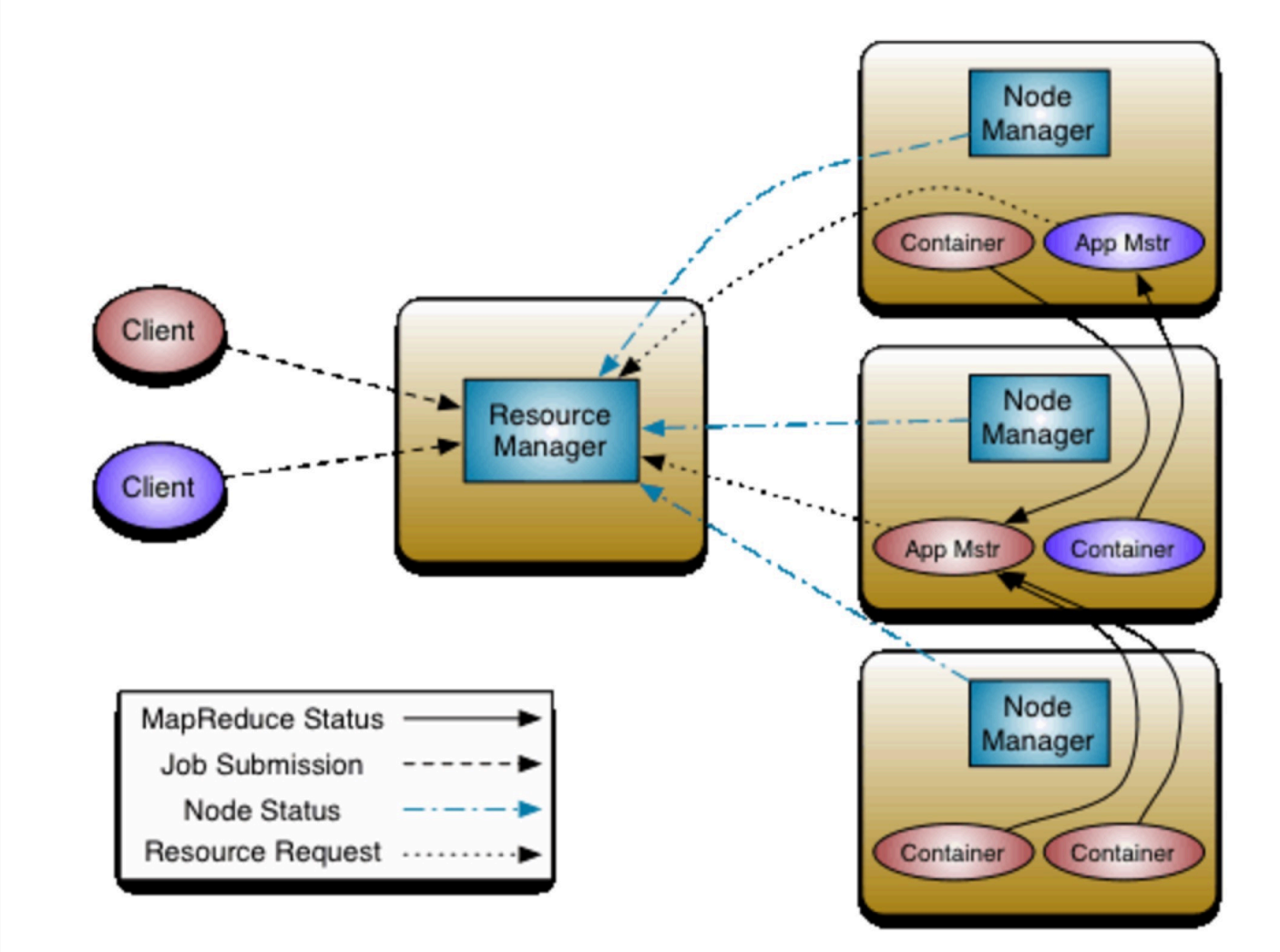


图 6: YARN 架构体系

ZooKeeper 典型的应用场景

YARN

从上图可以看出，YARN 主要由 ResourceManager(RM)、NodeManager(NM)、ApplicationMaster(AM) 和 Container 四部分组成。其中最核心的就是 ResourceManager，它作为全局的资源管理器，负责整个系统的资源管理和分配。

ResourceManager 单点问题，RM 负责集群中所有资源的统一管理和分配，同时接收来自各个节点 (NodeManager) 的资源汇报信息，并把这些信息按照一定的策略分配给应用程序 (Application Manager). 因此 ResourceManager 的工作状况直接决定了 YARN 框架是否可以正常运转。

ZooKeeper 典型的应用场景

ResourceManager HA

为了解决 ResourceManager 单点问题，YARN 设计了一套 Active/Standby 模式的架构

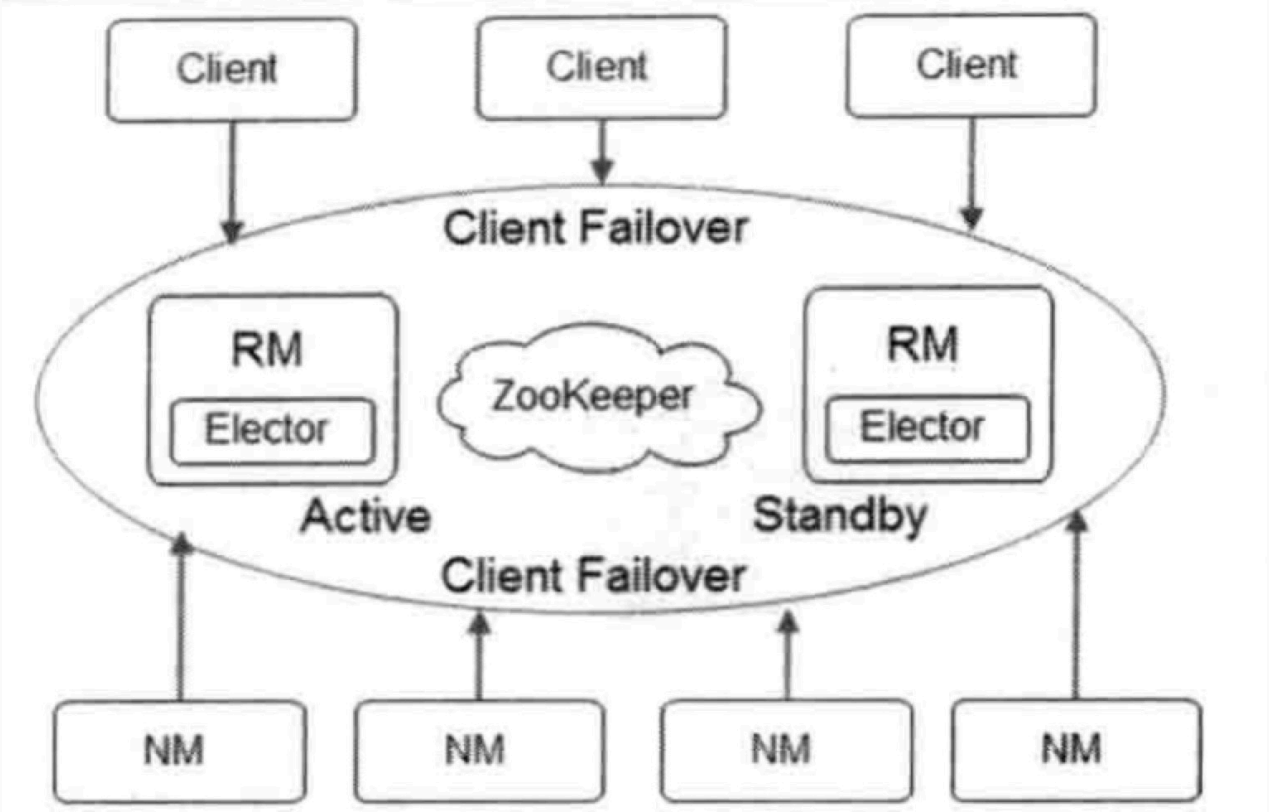


图 7: YARN HA 架构

ZooKeeper 典型的应用场景

YARN

在运行期间，会有多个 ResourceManager 并存，并且其中有一个 ResourceManager 处于 Active 状态，另外一些 (一个或多个) 处于 Standby 状态，当 Active 节点无法正常工作时 (宕机/重启)，其余处于 Standby 状态的节点则会通过竞争选举产生新的 Active 节点。

ZooKeeper 典型的应用场景

YARN ResourceManager 主备切换

- 1. 创建锁节点（临时节点）
- 2. 注册 Watcher 监听
- 3. 主备切换（锁失效，重复步骤 1）

Reference

Summary

theme metropolis

`github.com/matze/mtheme`

Paper：A Wait-free coordination for Internet-scale systems

Book：从 Paxos 到 ZooKeeper 分布式一致性原理与实战

Thanks!

Questions?