8、Hadoop再探讨

- Hadoop的优化与发展
  - Hadoop的局限与不足
    - 抽象层次低
      需要手工编写代码来完成，有时只是为了实现一个简单的功能，也需要编写大量代码
    - 表达能力有限
      mapreduce把复杂分布式编程工作高度抽象到两个函数上，map和reduce，在降低开发人员开发复杂度的同时，也带来了表达能力有限的问题，实际生产环境中的一些应用是无法用简单的map和reduce来完成的。
    - 作业之间的依赖关系
      一个作业（job）只包含map和reduce阶段，通常的实际应用问题需要大量的job协作才能顺利解决，这些作业之间往往存在复杂的依赖关系，但是mapreduce框架本身并没有提供相关的机制对这些依赖关系进行有效管理，只能由开发者自己管理。
    - 难以看到程序的整体逻辑
      用户的处理逻辑都隐藏在代码细节中，没有更高层次的抽象机制对程序整体逻辑进行设计，这就给代码理解和后期维护带来了障碍
    - 迭代操作效率低
      对于需要多轮迭代才能得到结果的任务，采用mapreduce实现这些任务时，每次迭代都是一次执行map、reduce任务的过程，这个过程的数据来自分布式文件系统hdfs，本次迭代的处理结果也被放到hdfs中，继续用于下一次迭代过程。反复读写hdfs文件中的数据，大大降低了迭代操作的效率。
    - 资源浪费
      reduce任务需要等待所有map任务都完成后才可以开始
    - 实时性差
      只适用于离线批数据处理，无法支持交互式数据处理、实时数据处理
  - 针对Hadoop的改进和提升
    - Hadoop自身框架的改进
      - hdfs
        - hdfs ha
          提供命名节点热备份机制
        - HDFS Federation
          设计了HDFS联邦，管理多个命名空间
      - mapreduce
        设计了新的资源管理框架
    - Hadoop生态系统的完善
      - pig
        处理大规模数据的脚本语言，用户只需要编写几条简单的语句，系统会自动转换为mapreduce作业，用于解决Hadoop1.0中抽象层次低的问题
      - Oozie
        工作流和协作服务引擎，协调Hadoop上运行的不同任务
      - Tez
        支持DAG作业的计算框架，对作业的操作进行重新分解和组合，形成一个大的DAG作业，减少不必要的操作，这是用于解决Hadoop1.0中，不同mapreduce任务之间存在重复操作的问题
      - Kafka

> 分布式发布订阅消息系统，一般作为企业大数据分析平台的数据交换枢纽，不同类型的分布式系统可以统一接入到kafka，实现Hadoop各个组件之间的不同类型数据的实时高效交换

- HDFS 2.0中的新特性
  - HDFS HA

    虽然HDFS 1.0中存在一个第二名称节点，但并不是热备，它与名称节点有着不同的职责，其主要功能是周期性地从名称节点获取命名空间镜像文件和修改日志，进行合并后再发送给名称节点，替换掉原来的FSImage，以防止editlog文件过大，导致名称节点失败恢复时消耗过多时间。合并后的FSImage在第二名称节点中也保存一份，当名称节点失效的时候，可以使用第二名称节点中的FSImage进行恢复。在HDFS 2.0 中，在一个典型的HA集群中，一般设置两个名称节点，其中一个处于active状态，另一个处于standby状态，active节点负责处理所有客户端的请求，standby节点则是热备节点
    - 状态信息同步

      由于standby节点是热备，因此active节点的状态信息必须实时同步到standby节点，这可以借助一个共享存储系统来实现（比如NFS、QJM或者zookeeper）。active节点将更新数据写入到共享存储系统，standby一直监听共享存储系统，一旦发现由新的写入，就立即从公共存储系统中读取这些数据并加载到自己的内存中，从而保证与活跃名称节点状态完全同步。（这里同步的貌似是editlog）
    - 文件块到数据节点的映射

      名称节点中保存了数据块到实际存储位置的映射信息，即每个数据块是由哪个数据节点存储的。当一个数据节点加入HDFS集群时，它会把自己所包含的数据块列表报告给名称节点，此后会通过心跳方式定期执行这种告知操作。而且这种报告是同时向两个名称节点报告的
  - HDFS Federation

    在HDFS联邦中，设计了多个相互独立的名称节点，使得HDFS的命名服务能够水平扩展，这些名称节点分别进行各自命名空间和块的管理，相互之间是联邦关系，不需要彼此协调。
- YARN

  mapreduce 1.0 采用master/slave 架构，包括一个JobTracker和若干个TaskTracker，前者负责作业的调度和资源的管理，后者执行JobTracker指派的具体任务。这种架构设计具有一些很难克服的缺陷，比如：（1）JobTracker单点故障，（2）JobTracker任务过重，（3）容易出现内存溢出，（4）资源划分不合理
  - 设计思路

    把原JobTracker的三大功能（资源管理、任务调度和任务监控）进行拆分，分别交给不同的新组件去处理。重新设计后的yarn包括ResourceManager、ApplicationMaster和NodeManager，其中，ResourceManager负责资源管理，由ApplicationMaster负责任务调度和监控，由NodeManager负责执行原TaskTracker的任务。在Hadoop 1.0 中，其核心子项目MapReduce1.0即是一个计算框架，也是一个资源管理调度框架。到了Hadoop 2.0之后，MapReduce1.0 中的资源管理调度功能被单独分离出来形成了yarn，它是一个存粹的资源管理调度框架，而不是一个计算框架；被剥离了资源管理调度功能的MapReduce框架就变成了MapReduce2.0，它是运行在yarn之上的一个纯粹的计算框架，不再自己负责资源调度管理服务，而是由yarn为其提供资源管理调度服务。
  - 体系结构

    在集群部署方面，yarn的各个组件是和Hadoop集群中的其他组件一起统一部署的。yarn的ResourceManager组件和HDFS的namenode部署在一个节点上，yarn的ApplicationMaster及NodeManager是和HDFS的数据节点部署在一起的。yarn中的容器代表了CPU、内存、网络等计算资源，它也是和HDFS数据节点一起的。
    - ResourceManager

      RM是一个全局的资源管理器，负责整个系统的资源管理和分配，主要包括两个组件，即调度器Scheduler和应用程序管理器ApplicationManager。在Hadoop平台上，用户的应用程序是以作业（job）的形式提交的，然后一个作业会被分解成多个任务（包括map任务和reduce任务）进行分布式执行。ResourceManager接受用户提交的作业，按照作业的上下文信息和从NodeManager收集来的容器状态信息，启动调度进程，为用户作业启动一个ApplicationMaster

        - Scheduler
            负责资源管理和分配，不再负责跟踪和监控应用程序的执行状态，也不负责执行失败恢复，因为这些任务都已经交给ApplicationMaster组件来负责。调度器接收来自ApplicationMaster的应用程序资源请求，把集群中的资源以“容器”的形式分配给提出申请的应用程序。
        - ApplicationsManager
    - ApplicationMaster
        applicationMaster的主要功能是：（1）当用户作业提交时，ApplicationMaster与ResourceManager协商获取资源，ResourceManager会以容器的形式为ApplicationMaster分配资源；（2）把获得的资源进一步分配给内部的各个任务（Map任务或者Reduce任务），实现资源的二次分配；（3）与NodeManager保持交互通信进行应用程序的启动、运行、监控和停止，监控申请到的资源的使用情况，对所有任务的执行进度和状态进行监控，并在任务发生失败时执行失败恢复；（4）定时向ResourceManager发送心跳消息，报告资源的使用情况和应用的进度信息；（5）作业完成时，ApplicationMaster向ResourceManager注销容器，执行周期完成。
    - NodeManager
        是一个驻留在YARN集群中每个节点上的代理，主要负责容器生命周期管理，监控每个容器的资源（CPU、内存）使用情况，跟踪节点健康情况，并以“心跳”的方式与ResourceManager保持通信，向ResourceManager汇报作业的使用情况和每个容器的运行状态，同时，它还要接受来自ApplicationMaster的启动/停止容器的各种请求。需要说明的是，NodeManager主要负责管理抽象的容器，只处理与容器相关的事情，而不具体负责每个任务（map任务或reduce任务）自身状态的管理，因为这些管理工作是由ApplicationMaster完成的，ApplicationMaster会通过不断与NodeManager通信来掌握每个任务的执行状态。
- 工作流程
    - 1、客户端提交应用程序
        用户编写客户端应用程序，向yarn提交应用程序，提交的内容包括ApplicationMaster程序、启动ApplicationMaster的命令、用户程序等。
    - 2、创建ApplicaitonMaster
        yarn中的resourceManager负责接收和处理来自客户端的请求。接到客户端应用程序请求以后，resourceManager里面的调度器回味应用程序分配一个容器（一个DataNode中可以有多个container，现在的这个container专门用于运行这个ApplicationMaster，不进行其他任务，如map或reduce，且每个作业都有一个专属于自己的ApplicationMaster，这个ApplicationMaster就是监控各个task的运行状态的）。同时，ResourceManager的应用程序管理器会与该容器所在的NodeManager通信，为该应用程序在该容器中启动一个ApplicationMaster
    - 3、注册
        ApplicationMaster被创建以后会首先向ResourceManager（ApplicationsManager）注册，从而使得用户可以通过ResourceManager来直接查看应用程序的运行状态。接下来的4-7是具体的应用程序执行步骤
    - 4、申请资源
        ApplicationMaster采用轮询的方式通过RPC协议向ResourceManager申请资源
    - 5、分配资源
        ResourceManager以“容器”的形式向提出申请的ApplicationMaster分配资源（此时分配的容器是用于执行具体任务的，如map或reduce），一旦ApplicationMaster申请到资源以后，就会与该容器所在的NodeManager进行通信，要求它启动任务（map or reduce）
    - 6、启动任务
        当ApplicaitonMaster要求容器启动任务时，它会为任务设置好运行环境（包括环境变量、JAR包、二进制程序等），然后将任务启动命令写到一个脚本中，最后通过在容器中运行该脚本来启动任务。
    - 7、汇报状态

各个任务通过某个RPC协议向ApplicationMaster汇报自己的状态和进度，让ApplicationMaster可以随时掌握各个任务的运行状态，从而可以在任务失败时重启任务。

- 8、注销Application

应用程序运行完成后，ApplicationMaster向ResourceManager的应用程序管理器注销并关闭自己。若ApplicationMaster因故失败，ResourceManager中的应用程序管理器会检测到失败的情形，然后将其重启，知道所有的任务执行完毕。

- 发展目标

YARN希望发展成集群中统一的资源管理调度框架，在一个集群中为上层的各种计算框架提供统一的资源管理调度服务。在yarn上可以部署其他各种计算框架，比如mapreduce、tez、hbase、storm、spark等，由yarn为这些计算框架提供统一的资管调度管理服务，并且能够根据各种计算框架的负载需求，调整各自占用的计算资源，实现集群资源共享和资源弹性收缩。

- 生态圈中其他组件
  - pig

pig可以加载数据、表达转换数据以及存储最终结果，因此在企业实际应用中，pig常用于etl（extraction、transformation、loading）过程，即来自各个不同数据源的数据被收集过来以后，采用pig进行统一加工处理，然后加载到数据仓库hive中，由hive实现对海量数据的分析。对于pig Latin脚本，pig会自动转换成mapreduce任务。

  - tez

tez是支持dag作业的计算框架，直接源于mapreduce框架，核心思想是将map和reduce两个任务进一步进行拆分，即map被拆分为input、process、sort、merge和output，reduce被拆分为input、shuffle、sort、merge、process和output，经过分解后的这些元操作可以进行自由任意组合产生新的操作，经过一些控制程序组装后就可以形成一个大的DAG作业。通过DAG作业的方式运行mapreduce作业，提供了程序运行的整体处理逻辑，就可以去除工作流中多余的map阶段，减少不必要的操作，提升数据处理的性能。在Hadoop 2.0 生态系统中，mapreduce、hive、pig等计算框架都需要最终以mapreduce任务的形式执行数据分析，因此tez框架可以发挥重要的作用。可以让tez框架运行在yarn框架之上，然后让mapreduce、pig和hive等计算框架运行于tez框架之上，从而借助于tez框架实现对mapreduce、pig和hive等的性能优化，更好地解决现有mapreduce框架在迭代计算（如pagerank）和交互式计算方面存在的问题。

  - kafka

Kafka是一种高吞吐量的分布式发布订阅消息系统，用户通过Kafka系统可以发布大量的消息，同时也能订阅消费消息。在大数据时代涌现出的新的日志收集处理系统（flume、scribe等）往往更擅长于批量离线处理，而不能较好地支持实时在线处理。相对而言，Kafka可以同时满足在线实时处理和批量离线处理。在公司的大数据生态系统中，可以把Kafka作为数据交换枢纽，不同类型的分布式系统（如关系数据库、NoSQL数据块、流处理系统、批处理系统等）可以统一接入Kafka，实现和Hadoop各个组件之间的不同类型数据的实时高效交换。