

全日制专业学位研究生

专业实习环节总结报告

学 院 计算机学院

姓 名 李东泽

学 号 ZY1706212

导 师 樊文飞

北京航空航天大学研究生院

2019年 9 月 17 日

# 一、实习目的

通过研一阶段的理论学习，对图计算领域已有一定认识，包括业界主流的图数据库、图计算引擎、以分布式图计算中同步、异步、混合计算模型等。但上述学习还仅限于理论上的认知，往往理论与实践有很大的不同，需要通过实践去加深甚至是修正对理论的理解，因此通过在图计算领域上的专业实习，可加深对整个图计算领域的了解，拓宽知识面，并从工程的角度提高分析和解决问题的实际能力，培养自己的刻苦精神、踏实的工作态度和团队协调能力，有利于毕业就业的双向选择。

# 二、实习内容与主要成果

在实习期间，我主要从事“分布式图计算引擎”项目的研究与研发工作，其中涉及图计算引擎核心代码开发、引擎资源调度管理服务的部署与接入、引擎与k8s生态的对接以及图计算引擎的python接口开发等工作，其中具体工作内容如下。

1、引擎资源调度管理服务的部署与接入

随着全球大数据进入快速发展时期，数据量呈现指数级爆发式增长，而这些数据中不同个体间彼此交互产生的数据以图的形式表现，例如淘宝的交易网络是由节点（个人或商品）和边（关注或点赞）构成的图。因此大规模图数据计算已经越来越受到工业界及学术界的重视。为此，众多分布式的图计算系统应运而生，其中就包括GRAPE（**GRAP**h **E**ngine），一个使用MPI（Message Passing Interface）机制的支持大规模数据计算的的分布式图计算系统，在这类系统中，往往一次计算所涉及的数据高达数百GB的量级，因此针对此类计算引擎如何合理利用集群计算资源显得尤其重要。

在合理利用集群计算资源的情形下，负载均衡是一种比较常用的计算机技术，用在多个计算机、计算机集群、网络连接、CPU、磁盘驱动器或者其他资源中分配负载，以达到最优化资源的使用、最大化吞吐率、最小化响应时间、同时避免过载的目的。使用带有负载平衡的多个服务器组件，取代单一的组件，可以通过冗余提高可靠性。负载平衡服务通常是由专用软件和硬件来完成。主要作用是将大量作业合理地分摊到多个操作单元上进行执行，用于解决互联网架构中的高并发和高可用的问题。因此我们需要将现有的负载均衡、甚至是集群资源管理框架服务应用到我们的图计算引擎中。

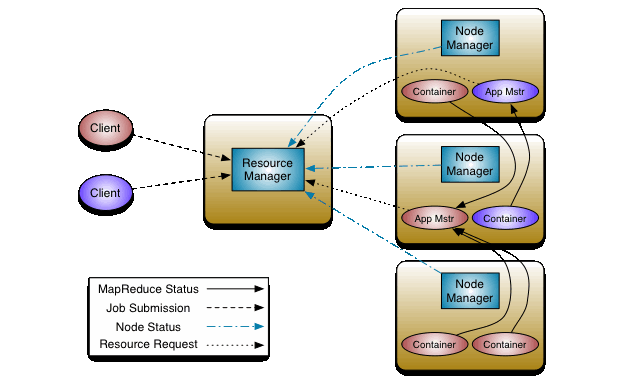


图2-1 Yarn架构图

目前工业界存在众多资源管理调度框架，其中YARN (Yet Another Resource Negotiator)已经成为大数据场景下存储与资源调度的统一方案，并被广泛应用于批量计算的各个领域，因此我们最终选用YARN作为我们资源管理框架。图2-1为Yarn的基本架构图，其中Yarn主要由Resource Manager、Node Manager、Application Master、Container四个组件组成。1) Resource Manager是一个全局的资源管理器嘛，负责整个系统的资源管理和分配，主要由调度器Scheduler和应用程序管理器Applications Manager组成，其中调度器根据容量、队列等限制条件（如某个队列分配一定的资源，最多执行一定数量的作业等），将系统中的资源分配给各个正在运行的应用程序。需要额外注意的是，该调度器是一个纯调度器，它不再从事任何与应用程序有关的工作，比如不负责重新启动（因应用程序失败或者硬件故障导致的失败）工作节点，这些均交由应用程序相关的 Application Master完成。调度器仅根据各个应用程序的资源需求进行资源分配，而资源分配单位用一个抽象概念资源容器(Resource Container，也即 Container)，Container 是一个动态资源分配单位，它将内存、CPU、磁盘、网络等资源封装在一起，从而限定每个任务使用的资源量。此外，该调度器是一个可插拔的组件，用户可根据自己的需求设计新的调度器，YARN提供了多种直接可用的调度器，比如Fair Scheduler和Capacity Schedule等；应用程序管理器主要负责整个系统中所有应用程序、包括应用程序提交、与调度协商资源启动第一个Application Master容器、监控Application Master容器的运行状态并在失败时重新启动它；2) Node Manager是每个节点行运行的资源和任务管理器，一方面它会定时向Resource Manager汇报本节点行的资源使用情况和各个Container的运行状态，另一方面，它也负责接收来自Application Master的Container启动或终止等各种请求；3) 提交的每个作业在Yarn中都包含一个Application Master，主要功能包括与Resource Manager协商以获取资源、将得到的任务进一步以Container的形式分配给内部的任务、与Node Manager通信以启动或终止任务、以及监控所有任务的运行状态，当有任务失败时，重新为任务申请资源并重启任务；4) Container是Yarn中的资源抽象，它封装了某个节点上的多维度资源，如内存、CPU、磁盘、网络等；当Application Master向Resource Manager申请资源时候，Resource Manager返回的资源是使用Container的形式表示的，Yarn会为每个任务分配一个Container且该任务只能使用该Container中所描述的资源。

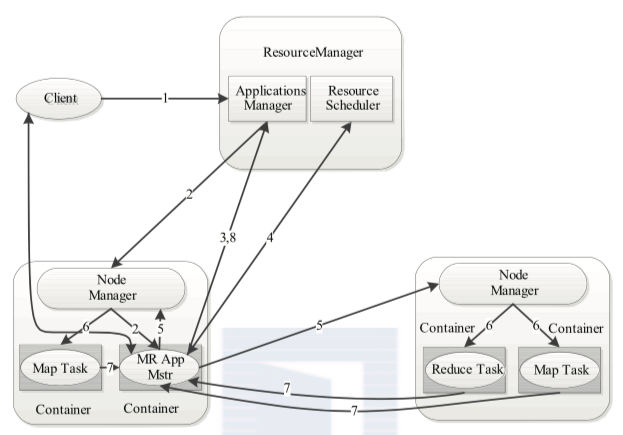


图2-2 Yarn 应用提交流程图

图2-2为Yarn作业提交流程图，当用户向Yarn中提交了一个应用程序后，Yarn将分两个阶段运行该应用程序：第一个阶段是启动 Application Master；第二个阶段是由 Application Master 创建应用程序，为它申请资源，并监控它的整个运行过程，直到运行完成，上图所示的 YARN 工作流程分为以下几个步骤：

1) 用户向 YARN 提交应用程序，其中包括 Application Master 程序，启动 Application Master 命令、用户程序等；

2) RM 为该应用程序分配第一个 Container，并与对应的 NM 通信，要求它在这个 Container 中启动应用程序的 Application Master；

3) Application Master 首先向 RM 注册，这样用户可以直接通过 NM 查看应用程序的运行状态，然后它将为各个任务申请资源，并监控它的运行状态，直到运行结束，一直重复下面的 4-7 步；

4) ApplicationMaster 采用轮询的方式通过 RPC 协议向 RM 申请和领取资源；

5) 一旦 ApplicationMaster 申请到资源后，便与对应的 NM 通信，要求它启动任务；

6) NM 为任务设置好运行环境（包括环境变量、jar 包等）后，将任务启动命令写到一个脚本中，并通过运行该脚本启动任务；

7) 各个任务通过某个 RPC 协议向 ApplicationMaster 汇报自己的状态和进度，以让 ApplicationMaster 随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务；

8) 应用程序运行完成后，ApplicationMaster 向 RM 注销并关闭自己。

因此，任何一个应用，包括我们的分布式图计算引擎，想要用Yarn作为我们的资源管理框架，必须严格按照上述流程将我们的图计算任务作为Yarn的应用提交给Yarn的Resource Manager，整个图计算引擎的资源管理过程主要分为以下两个部分：Yarn框架的部署和MPI分布式任务与Yarn平台的接入。

**Yarn框架的部署：**关于Hadoop Yarn框架的部署官网已有很详细的文档，因此我们只给出我们采用的调度策略。Yarn中的调度器是一个可插拔的组件，目前社区已经提供了FIFO Scheduler (先进先出调度器)、Capacity Scheduler (能力调度器)、Fair Scheduler (公平调度器)。**先进先出调度器**是最简单的资源调度策略，提交的作业按照提交时间先后顺序或者根据优先级次序将其放入线性队列相应的位置，在资源调度时，按照队列的先后顺序、先进先出地进行调度和资源分配；**公平调度器**先将用户的任务分配到多个资源池中，每个资源池设定资源分配的最低保障和最高上限，管理员也可以指定资源池的优先级，优先级高的资源池也将会分配更多的资源，当一个资源池有剩余时，可以临时将甚于资源共享给其他资源池，公平调度器的调度过程如下：1) 根据每个资源池的最小资源保障，将系统中的部分资源分配各个资源池；2) 根据资源池的指定优先级将剩余资源按照比例分配给各个资源池；3) 在各个资源池中，按照作业的优先级或者根据公平策略将资源分配给各个作业。公平调度器几个特点：1) 支持抢占模式，即如果某个资源长时间未能被分配到公平共享量的资源，则调度器可以杀死过多分配资源的资源池的任务，以空出资源供这个资源池使用；2) 强调作业之间的公平性：在每个资源池中，公平调度器默认使用公平策略来实现资源分配，这种公平策略是最大最小公平算法的一种具体实现，可以尽可能保证作业间的资源分配公平性；3) 负载均衡：公平调度器提供了一个基于任务数目的负载均衡机制，该机制尽可能将系统中的任务均匀分配到给各个节点上；4) 调度策略配置灵活：允许管理员为每个队列单独设置调度策略；5) 提高小应用程序响应时间：由于采用了最大最小公平算法，小作业可以快速获得资源并运行完成。**能力调度器**是为 Hadoop 开发的多用户调度器，应用于用户量众多的应用场景，与公平调度器相比，其更强调资源在用户之间而非作业之间的公平性。它将用户和任务组织成多个队列，每个队列可以设定资源最低保障和使用上限，当一个队列的资源有剩余时，可以将剩余资源暂时分享给其他队列。调度器在调度时，优先将资源分配给资源使用率最低的队列（即队列已使用资源量占分配给队列的资源量比例最小的队列）；在队列内部，则按照作业优先级的先后顺序遵循 FIFO 策略进行调度。能力调度器有以下几点特点：1) **容量保证**：管理员可为每个队列设置资源最低保证和资源使用上限，而所有提交到该队列的应用程序共享这些资源；2) **灵活性**：如果一个队列资源有剩余，可以暂时共享给那些需要资源的队列，而一旦该队列有新的应用程序提交，则其他队列释放的资源会归还给该队列；3) **多重租赁**：支持多用户共享集群和多应用程序同时运行，为防止单个应用程序、用户或者队列独占集群中的资源，管理员可为之增多多重约束；4) **安全保证**：每个队列有严格的 ACL 列表规定它访问用户，每个用户可指定哪些用户允许查看自己应用程序的运行状态或者控制应用程序；5) **动态更新配置文件**：管理可以根据需要动态修改各种配置参数。

作为支持大规模图数据的计算引擎，我们的产品已经为阿里巴巴集团不同的业务团队提供众多应用场景下的图计算支持，因此在我们的生产集群中包含业务方的图计算任务、团队开发人员的测试任务以及团队生产线上的Benchmark任务，这三种任务的优先级顺序为业务方图计算任务 > 团队生产线上任务 > 团队开发人员测试任务，且如果集群没有任何正在执行的任务时，上述任何一种任务都可占用集群全部计算资源，而当有高优先级任务提交后，集群要首先满足高优先级任务，甚至杀死低优先级任务，因此在上述三种调度器中，我们图计算引擎最终选用了能力调度器，其Hadoop配置如表2-1所示：

表2-1 capacity调度策略配置表

|  |  |
| --- | --- |
| Key | Value |
| yarn.scheduler.capacity.root.queues | Product dev |
| yarn.scheduler.capacity.root.product.capacity | 90 |
| yarn.scheduler.capacity.root.dev.capacity | 10 |
| yarn.scheduler.capacity.root.product.maximum-capacity | 100 |
| yarn.scheduler.capacity.root.dev.maximum-capacity | 100 |
| yarn.scheduler.capacity.root.product.user-limit-factor | 10 |
| yarn.scheduler.capacity.root.dev.user-limit-factor | 10 |
| yarn.scheduler.capacity.root.product.maximum-applications | 10 |
| yarn.scheduler.capacity.root.dev.maximum-applications | 20 |
| yarn.scheduler.capacity.maximum-am-resource-percent | 0.1 |
| yarn.scheduler.capacity.root.product.state | RUNNING |
| yarn.scheduler.capacity.root.dev.state | RUNNING |
| yarn.scheduler.capacity.root.product.acl\_submit\_applications | \* |
| yarn.scheduler.capacity.root.product.acl\_administer\_queue | \* |
| yarn.scheduler.capacity.root.dev.acl\_submit\_applications | \* |
| yarn.scheduler.capacity.root.dev.acl\_administer\_queue | \* |
| yarn.scheduler.capacity.root.product.default-application-priority | 90 |
| yarn.scheduler.capacity.root.dev.default-application-priority | 10 |
| yarn.scheduler.capacity.root.product.disable\_preemption | flase |
| yarn.scheduler.capacity.root.dev.disable\_preemption | flase |
| yarn.scheduler.capacity.root.product.intra-queue-preemption.disable\_preemption | flase |
| yarn.scheduler.capacity.root.dev.intra-queue-preemption.disable\_preemption | flase |
| yarn.scheduler.capacity.node-locality-delay< | 40 |

可以看到，我们将Yarn任务队列分为product队列和dev队列，product队列占有集群90%的计算资源、dev队列占有集群10%的计算资源，当集群没有任何计算任务时，product队列与dev队列均可使用集群全部计算资源；而当product队列和dev队列均有任务时，彼此只能分别使用集群90%和10%的计算资源。且整个调度处于强抢占模式，即集群没有任何计算任务时，dev队列的任务可使用集群全部计算资源，而product队列提交任务后，若集群剩余计算资源不满足该任务的计算需求时，便会抢占dev队列已使用的计算资源，该抢占过程逐渐进行，直到满足product队列任务的计算资源。

**Yarn平台接入**：在生产集群上部署了Hadoop Yarn平台后，还不足以为我们的图计算引擎提供资源调度与管理服务，因为任何一个应用如MapReduce、Spark等若想运行于Yarn上，均需按照Yarn的API实现对应接口，因此我们在MPICH2-YARN库的基础上，针对图计算引擎自身的特点，进行定制开发，保证图计算任务顺利高效的运行于Yarn平台上。

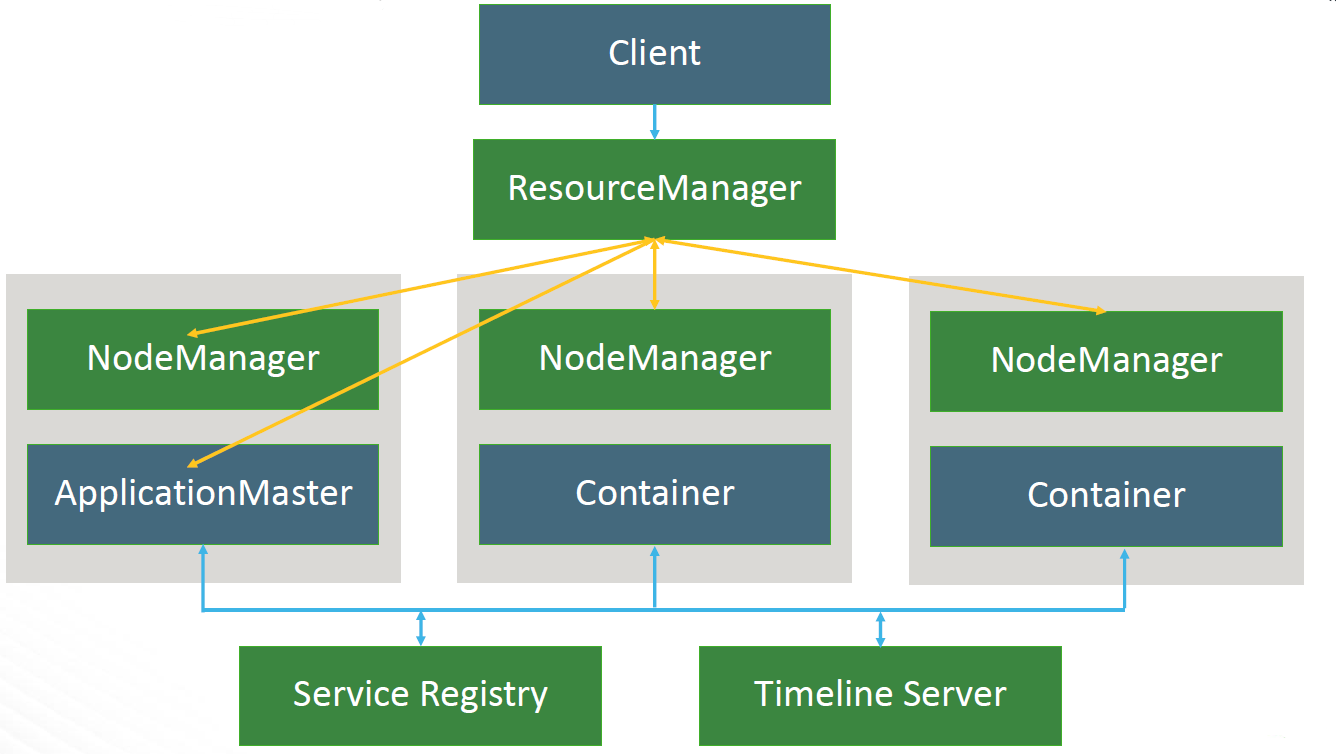


图2-3 Yarn 应用开发组件图

如图2-3所示，一个完整的Yarn应用，需要包含以下几个部分：1) Client用户提交程序，主要负责与用户交互接收用户传入的参数、负责上传或更新应用所依赖的资源(如jar包)、创建Application Master启动环境和启动参数、将Yarn 应用提交到集群上并监控程序执行完成；2) Application Master：作为Yarn应用的Master服务，主要负责资源的申请与释放、设置Container的启动环境与命令、负责启动Container、并同时监控Container的运行状态，对失败进行重试；3) Container：作为Yarn应用的计算服务。一个应用可以有一个或多个Container节点，主要负责执行Application Master分来的任务。因此按照图计算引擎的需求特点所开发的Yarn应用需要实现如下三个模块：

1. Application Client模块：采用ApplicationClientProtocol协议，用于将应用提交到Yarn上，使应用运行到Yarn上，同时，监控应用的运行状态，控制应用的运行

2. Application Master模块：分别采用ContainerManagementProtocol协议控制Node Manager启动Container和ApplicationMasterProtocol协议向Resource Manager注册并申请资源，应用实际的工作时在Container中进行的。

3. Application Worker模块：也叫Container模块，是应用的实际工作，Node Manager启动Application Master发送过来的容器，容器内封装了该应用worker上运行所需的资源和启动命令。并不是所有的应用都需要开发该模块，但我们的图计算引擎采用MPI消息传递机制来满足分布式图计算的信息传递，因此我们需要在图计算任务开始前每个计算节点做一些关于MPI SSH免密处理等相关操作，因此需要该模块。

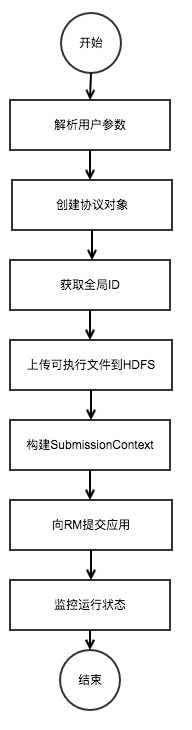


图2-4 客户端模块流程图

**客户端模块实现**：客户端实现流程图如图2-4所示，客户端工作流程如下：1) 客户端解析用户传入的命令行参数，该参数主要包含本次任务的可执行文件路径、期望内存、任务标签、优先级、MPI任务的进程数、提交队列等众多图计算相关的参数。2) 通过Hadoop RPC接口创建所用ApplicationClientProtocol协议的对象。3) 通过getNewApplication方法从Resource Manager获取此次任务的ID，该ID在整Yarn启动后全局唯一，同时还可获取当前Yarn集群的最大内存、CPU核数等信息，如果此时用户期望的内存大于当前集群最大节点内存，那么直接终止此次任务。4) 因为分布式图计算引擎需要每个计算节点均可访问可执行文件，因此需要将用户参数传入的可执行文件拷贝进HDFS以方便在Yarn集群中共享。5) 在通过上述步骤获取全局唯一ID后，便可准备本次任务的ApplicationSubmissionContext，该Context主要包含以下几个部分：applicationId：代表本次任务的全局ID；applicationName：应用名称，将显示在Yarn的Web页面上；priority：应用的优先级，数值越小优先级越高；queue：应用所属队列，对应之前部署的product队列或dev队列，如果设置为product队列，则表示此次任务作为线上生产任务，可能会抢占当前运行于dev队列任务的计算资源；maxAppAttempts：Application Master启动失败后的最大尝试重启次数；resource：启动Application Master所需的所有资源，包括虚拟CPU核数以及内存，其中虚拟CPU核数是一个归一化的值；environment：Application Master所使用的环境变量，主要包括本次图计算任务的可执行文件在HDFS上的路径等信息；commands：启动Application Master的命令列表；localResource：Application Master启动需要的本地资源列表，主要包含一些外部文件、压缩包等。6) 通过submitApplication方法将本次任务提交到Resource Manager上。7) 通过ApplicationClinetProtocol协议中的GetApplicationsResponse等方法监控本次任务的运行状态，并根据需要可随时通过客户端杀死本次任务。

**Application Master模块实现**：在图计算应用中，Application Master主要工作是按照客户端要求向Resource Manager申请本次任务的全部计算资源，并根据申请到的计算资源确定本次MPI任务的计算节点，等各节点完成MPI任务的所必须的初始化任务后，由Application Master负责启动本次图计算任务，并监控图计算任务执行状态，因此既需要与Resource Manager通信与需要与Node Manager通信，其流程图如图2-5所示，具体工作流程如下：1) 解析从客户端传递过来的运行参数，该参数主要包含本次任务申请的container、CPU核数、内存、可执行文件路径等参数。2) 初始化与Container和Clinet通信的RPC服务，其中与Container通信的RPC服务主要负责监听各个Container的心跳，与Clinet通信的RPC服务主要负责向Client传送本次任务的执行日志；3) 调用registerApplicationMaster方法向Resource Manager注册自己，注册信息包括Application Master所在节点和开发的RPC服务端口，以及一个应用状态跟踪Web接口，随后Resource Manager返回给Application Master一个对象，里面包含了应用最大可申请的单个容器容量、应用访问控制列表和一个用于与客户端通信的安全令牌。4) 在拿到上述容器容量后，Application Master便根据用户提交的container与容量请求通过allocate方法向Resource Manager申请资源，Application Master向Resource Manager发送的信息被封装在AllocateRequest里，包括reponseId：用于区分重复的响应；askList：AM向RM申请的资源列表，是一个List对象，其中ResourceRequest中一个资源请求的详细参数，包括优先级、容器个数、单个容器容量和分配策略(是否放宽本地化约束)；releaseList: AM主动释放的资源容器列表；progress：应用的运行进度。5) Resource Manager接受到Application Master的请求后，扫描器上的资源镜像，按照调度算法分配全部或部分申请的资源给Application Master，并返回一个AllocateResponse对象，该对象里面主要包含numClusterNodes：集群规模大小；updatedNodes：状态被更新过的所有节点列表；httpAddress：节点Web页面地址；numContainers：节点上当前运行的Container个数；nodeState：节点运行状态；used：节点上已经使用的资源量；capability：节点总的资源量；AMCommand：Resource Manager发送给Application Master的控制命令，包括重连和关闭；resource：该容器所持有的资源，包括内存和CPU核数；preemptionMessage：资源抢占信息，包括强制回收部分和自主调配部分，其中可自主调配部分主要包含了两个内容，分别是抢占资源需求和可抢占资源列表，Application Master需要从可抢占的资源中选择出部分资源进行释放，以满足抢占资源的需求。6) 根据上一步Resource Manager向Application Master返回的资源申请状况，Application Master构造出两个重要的数据结构，分别是Map<String, Integer> hostToProcNum和List<containerId> distinctContainers，其中hostToProcNum代表本次所申请的所有节点和每个节点对应的MPI任务的进程数，而distinctContainers代表本次申请的所有计算节点。7) 针对上述distinctContainers结构中的每个节点，Application通过ContainerManagementProtocol协议的startContainers方法启动对应容器，并通过该接口向Node Manager提供启动容器的必要配置，包括环境变量、启动命令、已经本次资源列表等信息，这些信息都被封装到了StartContainersRequest中，Node Manager收到请求后，会启动响应的容器，并返回启动成功的容器列表和失败的容器列表。8) Application Master在等待所有节点上的Container初始化完成后，便根据hostToProcNum中的进程数目通过Java Runtime类启动MPI图计算任务，并根据命令的返回接口监控并确认本次任务的执行状态。在此期间，Application Master通过监听所有Container的心跳同时跟踪各个任务的运行状态。9) MPI任务执行完成后，Application Master通过ContainerManagementProtocol的stopContainers方法停止Node Manager上运行的容器，释放相关资源，并将释放的资源上报给Resource Manager，以方便Resource Manager完成最后的资源回收工作。随后，Application Master调用unRegister方法向Resource Manager通知本次任务完成。

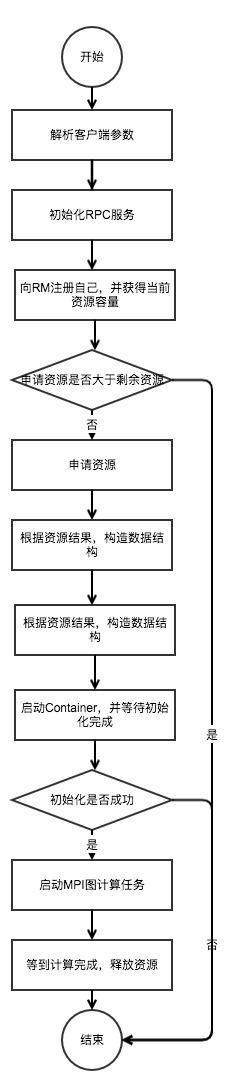


图2-5 AM流程

**Container模块实现**：在Application Master向Resource Manager成功申请资源后，还需在各个申请节点上启动Container容器，并等待其初始化完成。因此Container部分的实现主要是为了解决MPI图计算任务的前期准备工作，包括从HDFS上拷贝本次任务的可执行文件、将统一密钥写入节点对应的密钥文件中，以方便MPI消息通信等必备工作。待本次任务结束后，还需从密钥文件中擦去之前写入的密钥，以防止污染集群免密信息。

**成果**：目前，已在拥有53台服务的生产集群上部署Hadoop Yarn框架，并将我们的图计算引擎通过上述应用程序接入Yarn资源管理框架，目前服务稳定，高效，日均提供近240次的资源调度服务，极大的解决了集群资源的利用不合理和生产任务与测试任务的冲突问题，目前仍一直参与后续的迭代与维护工作。

2、引擎与k8s生态的对接

一方面，为了使图计算引擎面向阿里巴巴集团全体用户团队，提供图计算服务支持；另一方面，为了降低自己生产集群上的运维成本，将我们的图计算引擎作为k8s的一个应用运行与阿里巴巴云平台上的Cupid集群中。

整个接入过程需要与众多团队协调配合，从无到有实现图计算服务上云，其中所做工作如下：

1. 联系前端团队，协商制作页面节点，最终该节点需要用户设置当前引擎运行版本、worker数以及本次任务的参数信息。

2. 开发Wrapper层代码，该代码主要用于解析前端传入的参数，并根据此次任务启动Cupid集群。

3. 制作Application Master Docker镜像，该镜像用于向Cupid申请计算资源 (docker)，并根据申请的结构拉起Cupid Worker节点。

4. 制作Worker Docker镜像，该镜像含有图计算引擎的可执行环境，最终作为Cupid图计算的工作镜像执行图计算任务。

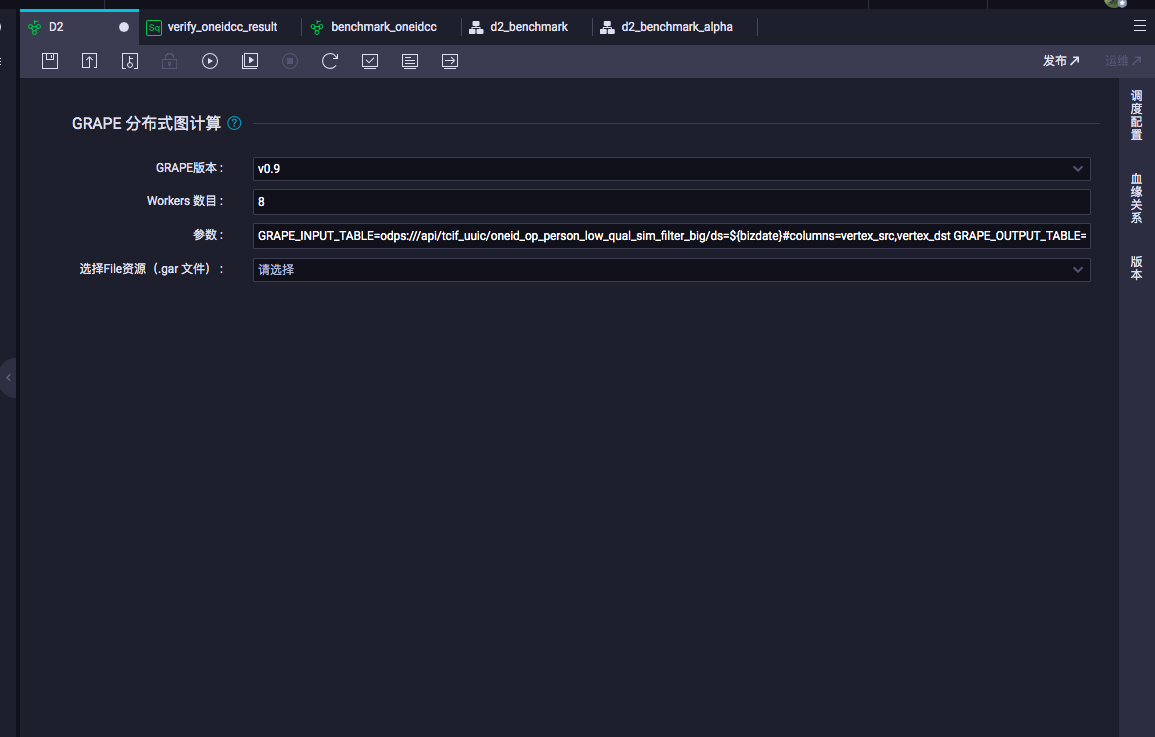


图2-6 引擎节点页面

**引擎Cupid节点**：图2-6所示是我们引擎的节点页面，秉着用户友好以及易于使用的原则，用户只需要选择引擎当前版本、本次任务的worker数目，以及输入对应任务的参数便可通过Cupid拉起集群完成定向图计算任务。

**Wrapper插件层**：Wrapper层作为连接Cupid前端节点与后端图计算任务的插件层，我们需要解析前端传递的参数，并通过调用Cupid API完成对Cupid集群的启动与终止操作，整个过程还需监听Cupid任务执行状态，并通过K8s ConfigMap将内部图引擎运行的细节实时反馈给用户前端。

**镜像制作**：除了上述节点页面于Wrapper代码的开发，我们还需制作中间过程所涉及的镜像，包括Application Master镜像和Worker镜像。其中Application Master主要负责向Cupid申请计算资源，并根据分区的资源拉起K8s集群；K8s集群的每个Pod上运行的是Worker镜像，主要包含图计算引擎所需的一切运行环境。

**成果**：目前成功完成图计算引擎与k8s Cupid的对接工作，已面向集团所有团队，均可提供图计算服务支持，运行稳定。目前有6个来自不同团队的应用，包括cc、oneidcc、cn、cn\_ui、sampling path、louvain，已经开始使用Cupid上的图计算服务，且oneidcc业务团队已将我们的服务作为生产环境的流程节点发布生产。

3、图计算引擎的Python接口开发

为了进一步增强我们图计算引擎在集团内部以及业界的影响力，使其用户友好，更加易于使用，因此开发Python接口，并集成进Jupyter NoteBook。

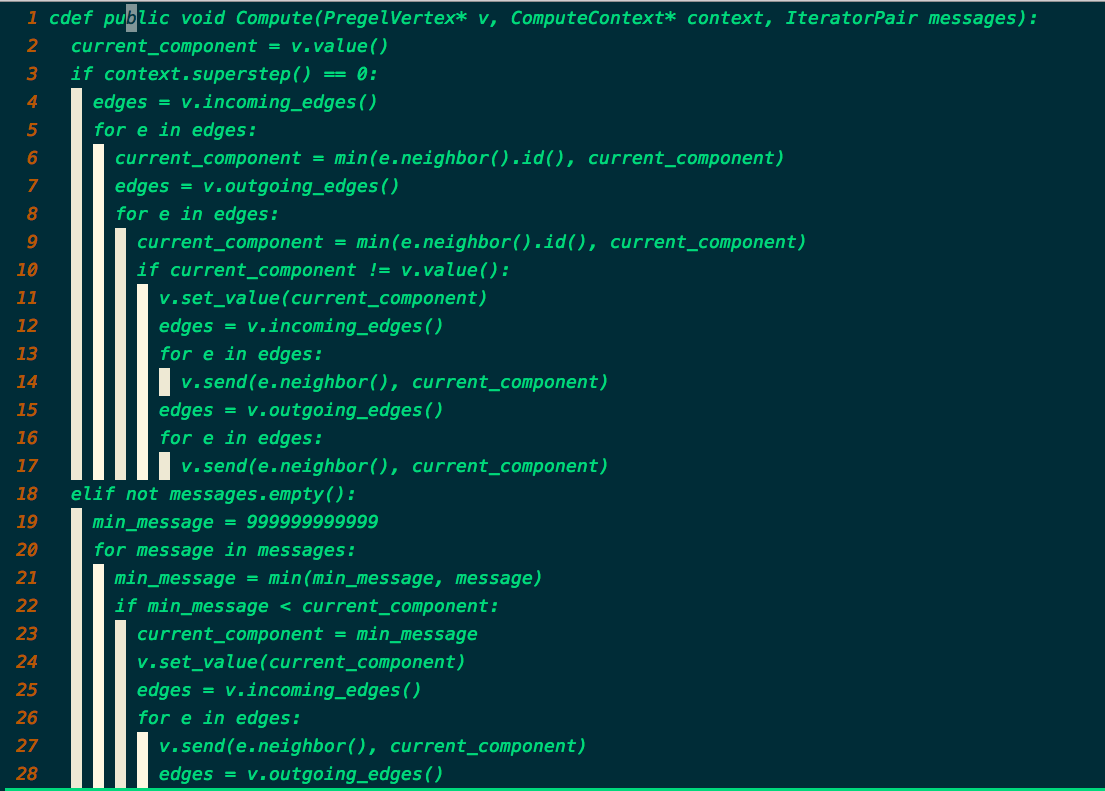


图2-7 图联通性算法Python Pregel编程接口

**Python Pregel编程接口：**为了用户友好，我们首先实现了一版Pregel编程模型下的Python接口，如图2-7所示。

**Jupyter NoteBook集成**：为了与Jupyter NoteBook集成，采用GRPC框架开发了图计算引擎的远程调用服务版本，之后进一步开发用于接口的Ipython Magic Function以方便用户在NoteBook中快速开发应用。

# 三、实习表现

2019年1月至9月8个月期间，我在阿里巴巴集团达摩院图计算团队进行实习。在实习过程中，我时刻注意加强自己的主人公意识，完善个人表达能力，在以后的研究规划上能够深入思考，主动学习新技术，不然很容易掉队；且在实习期间，对于分给我的每一份工作，我都认真负责，积极专研，学习新的知识，顺利完成了部门的多个子任务，个人能力得到了部门领导的认可，我们每周都会有组会和项目会，以及定期的项目总结，通过这些方式，我的表达能力得到了极大的锻炼，这是第一次正式与社会接轨，踏上工作岗位，才慢慢的感觉到，每天的生活节奏和校园生活完全不一样，首先每天有较严格的上下班时间，且上班期间要认真准时地完成自己的工作任务，不能草率敷衍了事；其次，每周一次的组会使我能够回顾这一周以来的工作内容，可以很清楚的看到每周任务的完成情况与进度信息；最后我深刻的体会到了校园学习的意义，那就是整个学生时代不仅仅是学习书本上的知识，更多的是要培养自己自学的能力和针对某一方向深入探索的精神。这两点在参加工作后才能深刻体会。这次实习基本达到了我的预期目标，个人能力得到了展现，也发现了自己的不足之处，希望以后正式工作后能逐渐锻炼自己各方面的能力，成为一名技术骨干，为团队、为企业创造商业价值，实现自己的人生目标。

# 四、收获体会

整个实习期间，从技术、价值观、人生规划等方面均有颇多体会。首先是技术方面，通过部署与接入Yarn平台，是我对Hadoop的架构和Yarn应用的执行流程有了比较深入的理解，尤其针对MPI分布式任务与Yarn平台的对接流程有了非常深入的了解与体会；通过与Cupid团队协调配合，使我对K8s的基本操作有了一定的了解，同时也对Docker的概念与操作有一定的锻炼；通过编写Python SDK接口使我更加深入了解C++与Python这两门语言，以及如何完成跨语言间的通信。其次从价值观方面来说，在整个阿里巴巴实习期间，时时刻刻都能感受到客户第一的原则，真的不是说说而已，比如我们的图计算引擎目前虽然只是面向公司内部的团队用户，但我们每次上线均会将所有用户的用例全部执行一遍，无论耗时多长时间要做零故障；并且任何用户提出的任何需求，均需第一时间与用户沟通，确定下最终的解决方案，立即进入开发迭代流程。最后是我自己的人生规划方面，到了公司才发现，周围都是所谓的“大神”级人物，他们也是从实习阶段一点点走到今天，因为他们身上有很多东西值得我去思考、去学习，比如既然自己喜欢技术，那毕业后的3-5年要争取所有时间努力提升自己的技术竞争力，要在某一个领域，甚至是某一个主流技术框架有自己的深入理解甚至是创造创新能力，5年后要在继续加强自己技术之余，要乐于奉献，去分享自己的所学所想。总之8个月实习看似漫长，但过的飞快，每一天的酸甜苦辣构成了一幅幅画面，希望自己不忘初心，在计算机的道路上越走越长。