计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(8):63−70 [doi: 10.15888/j.cnki.csa.007017]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Jupyter 交互式分析平台的微服务架构①

贺宗平[1], 张晓东[2], 刘　玉[3]

[1](南京审计大学 信息化办公室, 南京 211815)
[2](南京审计大学 信息工程学院, 南京 211815)
[3](南京审计大学 实验中心, 南京 211815)
通讯作者: 贺宗平, E-mail: hzp@nau.edu.cn

摘　要: 随着 Jupyter Notebook 在数据科学领域应用规模的不断扩大, 对于多用户管理和集群计算资源调度的功能需求越趋增加. 本文从 Jupyter 相关基本概念入手, 阐述了 Jupyter 对于科研成果交流传播的作用影响, 总结了目前国外科研机构、高等院校等组织在研究 Jupyter 分布式架构方面的现状; 详细分析了 Jupyter 体系架构特点, 运用微服务的方式重构 Jupyter, 并通过 Kubernetes 的资源调度分配算法, 实现了基于容器技术的高弹性分布式微服务架构. 测试结果数据表明, 本文提出的架构在访问负载性能上得到了一定程度的提升, 在用户运行数量方面达到了集群上负载均衡的目标.

关键词: Jupyter; Kubernetes; 容器; 微服务架构

引用格式: 贺宗平,张晓东,刘玉.基于 Jupyter 交互式分析平台的微服务架构.计算机系统应用,2019,28(8):63–70. http://www.c-s-a.org.cn/1003-3254/7017.html

Microservice Architecture for Jupyter-Based Interactive Analysis Platform

HE Zong-Ping[1], ZHANG Xiao-Dong[2], LIU Yu[3]

[1](Information Office, Nanjing Audit University, Nanjing 211815, China)
[2](School of Information Engineering, Nanjing Audit University, Nanjing 211815, China)
[3](Experiment Center, Nanjing Audit University, Nanjing 211815, China)

Abstract: With the increasing application of Jupyter Notebook in the field of data science, the more functional requirements for multi-user management and cluster computing resource scheduling are increasing. This study, starting with the basic concepts of Jupyter, expounds the influence of Jupyter on the broadcast of scientific research achievements, summarizes the present situation of the research organizations and institutions of higher learning and other organizations in the field of research on the distributed architecture of Jupyter, analyzes the characteristics of the architecture of the Jupyter system in detail, and reconstructs the Jupyter by means of microservice. Through the resource scheduling and allocation algorithm of Kubernetes, a highly elastic distributed architecture based on container technology is implemented. Finally, the test results show that the architecture proposed in this study has been improved to a certain extent on the performance of the access load, and the target of load balancing on the cluster is achieved in the number of users' running.

Key words: Jupyter; Kubernetes; container; microservice

① 基金项目: 江苏高校品牌建设工程一期项目立项专业审计学 (PPZY2015A077); 南京审计大学 2018 年度高教所课题 (2018JG061); 南京审计大学政府审计研究基金 (GAS161039)

Foundation item: Stage 1 Project "Auditing", Brand Construction Program of Jiangsu Higher Educations (PPZY2015A077); Year 2018, Higher Education Institute Project of Nanjing Audit University (2018JG061); Government Auditing Research Fund of Nanjing Audit University (GAS161039)

收稿时间: 2019-02-16; 修改时间: 2019-03-08; 采用时间: 2019-03-27; csa 在线出版时间: 2019-08-08

1 引言

Jupyter Notebook 是当前在数据科学领域非常受欢迎的交互式分析软件, 支持 Python、R、Julia、Scala、SQL 等多种编程语言内核, 具备即时编译、可视化和 markdown 语法编辑等功能[1], 为数据分析、数值计算、统计建模、机器学习等方面提供了便捷. 尽管基于 web 的设计实现方式有许多局限性, Jupyter Notebook 在数据科学领域得到了极为广泛应用, 主要原因就在于其能够进行交互式、可视化的数据探索性分析, 从最简单的点线图、地图以及复杂的 D3.js 可视化等, 甚至可以在某些场景中替代 BI 工具的功能, 这些特点相对于传统 IDE 编辑器具有颠覆性优势. 但是 Jupyter Notebook 是传统的单体服务架构模式, 缺少多用户管理和访问认证等方面的功能, 无法直接部署于计算集群上, 难以充分利用和调度计算中心的计算资源.

当今的科研学术论文向着越来越复杂的方式演变, 各个研究领域和研究内容对程序依赖度高, 需要通过编程来处理数据、绘制图表以及统计分析等. 研究中的程序代码繁琐复杂, 直接导致了研究结论难以被他人重复实现, 科研的真正价值无法得到交流传播[2]. Jupyter Notebook 提供了一种混合的编辑方式, 将程序代码运行、文字图表编辑等功能糅合在同一电子笔记中, 打破了两者之间的隔离界限, 将研究中的论述表达和实验分析等过程有机结合起来. 对于科研结论的交流、分享和传播的形式方法上, 具有里程碑式的重要意义.

目前, 国外一些大型研究机构、实验室以及超算中心开始提供面向多用户的 Jupyter Notebook 服务[3], 目的就是将高性能计算集群资源充分利用起来, 为研究人员提供一个性能优异、操作便捷的工具[4–6].

美国国家能源研究科学计算中心 (NERSC) 在其 Cori 超算集群上部署 Jupyter, 为其 6000 多用户提供了更便捷化的计算资源访问方式, NERSC 在“Science Gateway”节点设置多用户管理环境, 当用户登录后可以启动一个专属的项目环境, 并且能够访问计算中心其他服务、文件和作业, 直接访问大规模数据集、作业队列系统, 允许用户提交作业, 查询作业批处理运行情况. OpenMSI[7]是 NERSC 目前正在运行的基于云计算平台和 Jupyter 的质谱成像项目, 研究人员可以在 web 浏览器中对质谱成像数据进行分析、展示和操作.

科罗拉多大学波尔得分校 (CU-Boulder) 计算研究中心为研究人员提供了大规模计算资源、海量存储等服务, 此外还提供计算科学和数据管理方面的咨询服务. 为了更好地提供优质服务, CU-Boulder 计算研究中心在其 HPC 集群上部署了支持并行计算的 Jupyter 环境, 能够支持 ipyparallel 和 MPI for Python.

欧洲原子能研究机构 (CERN) 开发了 SWAN[8] (Service for Web based ANalysis) 交互式数据分析平台, 在集群中通过 Docker 镜像构建 Jupyter Notebook 应用, 并且支持其内部开发的 CERNBox 数据云存储服务, 使得代码、数据集、文档在其组织内部安全无缝地实现同步和共享, 如图 1.

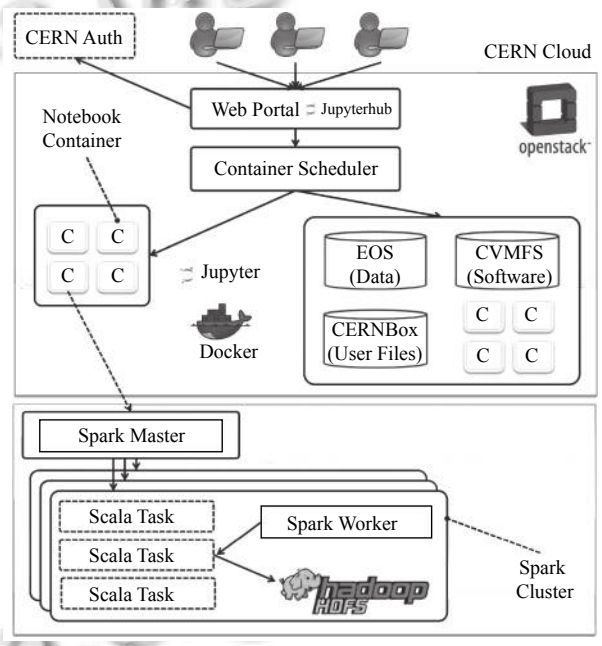


图 1 SWAN 体系架构示意

本文研究的重心关注 Jupyter 体系的架构优化与拓展, 通过容器和微服务化改造 Jupyter 体系架构, 研究基于 Kubernetes 集群构建面向多用户的交互式数据分析平台.

2 Jupyter 体系架构

2.1 Jupyter Notebook 架构特点

Jupyter Notebook 是一个典型 Web 架构的应用, 客户端部分负责笔记代码的运行、存储和输出等功能, 并通过 markdown 语法进行标记, 以 JSON 格式发送给服务器端存储, 服务器端负责存取笔记代码、调用编译内核等功能.

Jupyter Notebook (如图 2) 作为最基本的交互式分

析程序, 缺少平台级的资源调度与管理的能力, 不具备面向多用户的服务管理功能, 后台运行环境无法做到隔离. 这种状况限制了 Jupyter Notebook 在计算集群上的部署应用.

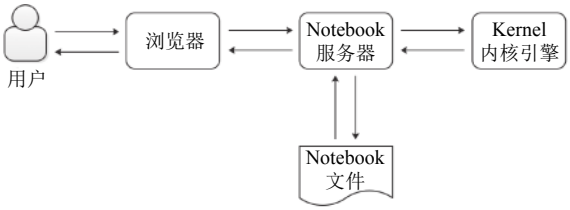


图 2 Jupyter Notebook 架构示意

2.2 Jupyter Hub 的改进与局限

Jupyter Hub 在 Jupyter Notebook 基础上实现了多用户的管理, 用于创建、管理、代理多个 Jupyter Notebook 实例. Jupyter Hub 包含 3 个组件: 多用户的 Hub 管理器、可配置的 http 代理、多个单用户 Jupyter Notebook 服务器. Jupyter Hub 架构如图 3 所示.

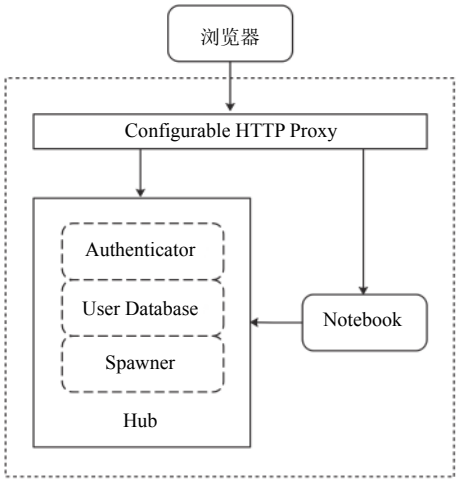


图 3 Jupyter Hub 架构示意

Jupyter Hub 的功能流程[9]包括: 首先, Hub 服务启动一个代理; 然后, 代理默认将所有访问请求转发给 Hub; Hub 处理用户登录认证并根据需求启动一个单用户的 Jupyter Notebook 服务器; 最后, Hub 配置代理将 URL 前缀转发给单用户的 Jupyter Notebook 服务器. Jupyter Hub 实现多用户的服务管理、具备访问认证机制, 但仍然无法实现集群调度、分布式计算、弹性扩容等功能, 特别是无法直接部署在大型计算集群. 对应用系统进行微服务化改造、运用分布式架构是解决问题的根本途径.

2.3 Jupyter 微服务化重构

微服务将单体应用 (Monolithic) 先拆分成多个单元应用, 服务间通过 HTTP API 或消息中间件进行通信, 再聚合组成实际应用, 这个过程显著改善了系统架构的弹性, 同时通过容器技术可以降低系统持续集成与持续部署 (CI/CD) 的复杂度. 容器作为微服务的一种基础运行方式, 具备成熟和强大的工具、平台以及开发生态圈. Jupyter 微服务化重构示意图如图 4.

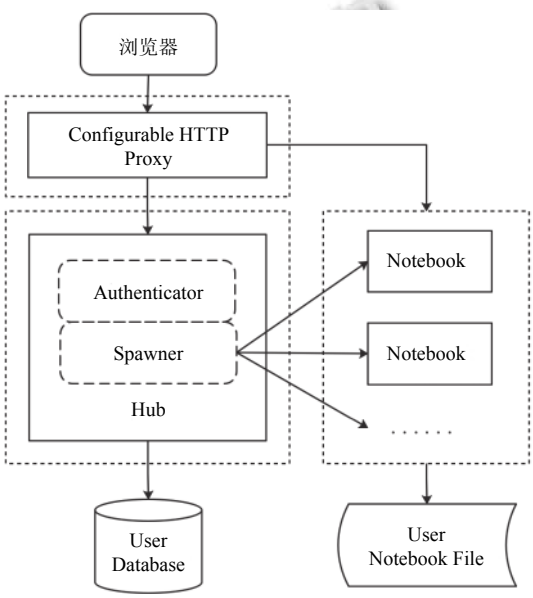


图 4 Jupyter 微服务化重构示意

2.3.1 微服务架构特点

微服务之父 Martin Fowler 总结微服务具有 3 个典型特征[10]: 微小化 (small), 独立化 (independently deployable) 和自动化 (automated deployment). “微小化”是“独立化”的基础, “微小化”代表微服务的设计应当遵循“高内聚、低耦合”的原则, 功能边界定义清晰无重叠, “自动化”代表系统组装、部署和运维可以通过工具实现高度的自动化.

康威定律[9](Conway’s Law) 认为组织结构决定系统结构. 随着组织规模的不断扩大, 必然会演化出更多的小型组织. 从这个角度来看, 传统单体应用拆分转换为微服务的过程, 是系统结构适应平台功能扩展的演变. 微服务降低了系统架构的复杂性, 通过将单体应用分解为一组服务, 在保持功能不变的条件下, 强化了系统模块化的组织水平, 使得开发更易于维护.

2.3.2 Jupyter 的微服务拆分

单体应用的主要架构特征是系统模块间的“紧耦

合”, 模块运行在同一进程中. 在系统部分模块更新升级、故障掉线等情况下, 需要重新启动整个应用进程, 导致系统可靠性和可维护性水平低下. 微服务架构通过对单体应用的不同功能模块进行拆分重构, 转换为多个独立的微服务, 运行为多个独立的进程, 微服务间通过 REST、RPC 等远程接口通信, 实现分布式架构部署[11].

微服务架构包括了无状态化和容器化两个前提准备环节, 从架构角度来看, 单体架构对状态信息的读写在本地的内存或存储中, 从而导致难以进行横向扩展. 无状态化的过程是将业务逻辑无状态部分与数据读写存储的有状态部分做分离, 业务逻辑经过无状态化处理能够实现横向扩展, 状态部分则通过中间件、分布式数据库等进行存储, 从而实现单体架构向分布式架构的转化改造. Jupyter Hub 的 3 个主要组成部分: Configurable HTTP Proxy、Hub、Notebook, 3 个模块间功能界限清晰, 完全可以做到微服务化拆分, 符合能够独立运行的标准, 其中的 Hub 和 Notebook 模块需要进行无状态化 (stateless) 处理. Configurable HTTP Proxy 是可配置的访问代理模块, 负责代理转发用户的访问请求; Hub 中主要包括了认证功能模块 (Authenticator) 和 Notebook 生成模块 (Spawner), Authenticator 提供 web 安全访问认证的接口, Spawner 负责生成和启动各个隔离的 Notebook 容器.

3 基于 Kubernetes 微服务架构设计

3.1 相关基本概念

Kubernetes(简称 K8S) 作为一个分布式集群管理平台, 同时也是一个容器编排系统, 用于在主机集群之间自动部署、扩展和运行应用程序容器, 提供以容器为中心的基础架构. K8S 具有如服务命名与发现、负载均衡、运行状况检查、横向弹性伸缩和滚动更新等功能[12], 适合在生产环境中部署应用程序. 基于 Kubernetes 的 Jupyter 微服务架构如图 5 所示.

Pod: Pod 是 Kubernetes 的基本单元, 由一个或多个容器组成, 并在同一个物理主机内, 共享相同的资源. 在 pod 内部署的所有容器都可以通过 localhost 看到其他容器. 每个 Pod 在集群中具有唯一的 IP 地址.

Service: Service 是一组 Pod 的逻辑区分, 是 Pod 在集群内部被公开访问的策略, 同时允许在集群外部 IP 地址上访问, 主要通过 ClusterIP、NodePort、Load Balancer 和 ExternalName 等四种方式进行地址暴露.

Replication Controller: Kubernetes 的一种控制器, 能够在集群中运行指定数目的 Pod 副本, 实现 Pod 的数量伸缩 (scale) 操作.

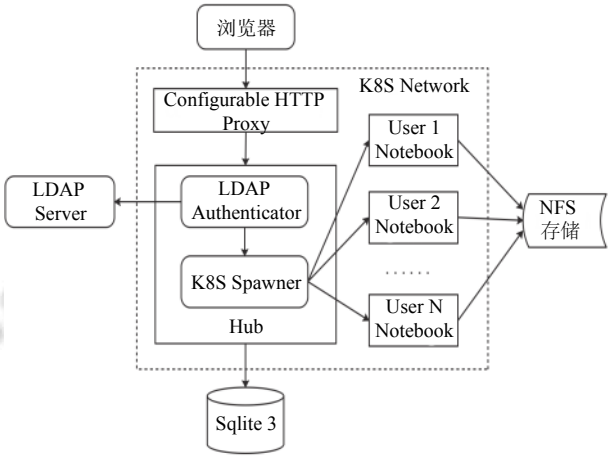


图 5 基于 Kubernetes 的 Jupyter 微服务架构

3.2 Jupyter 微服务架构要素

3.2.1 访问认证

交互式数据分析平台通常部署在计算中心, 向组织内部提供计算服务, 其访问认证的形式应当能够与组织内部现有账户信息做到无缝对接和集成. LDAP 是轻量目录访问协议 (Lightweight Directory Access Protocol) 的缩写, 是从 X.500 目录访问协议的基础上发展而来, 是一种集中账号管理架构的实现协议. LDAP 通常应用于组织内部的账户密码管理[13], 构建集中的身份验证系统, 可以降低管理成本, 增强安全性.

Jupyter Hub 中访问认证功能可通过插件的方式配置 LDAP 访问认证功能, 与组织内部现有用户访问认证体系完整集成. 这种方式对于大型科研机构、高校等组织, 可以显著降低用户管理和安全运维成本.

3.2.2 持久化存储

Kubernetes 持久化存储通过 API 资源管理, 包括 PersistentVolume 和 PersistentVolumeClaim. Kubernetes 中的存储组件支持 NFS、EBS 等多种后端存储, 存储具有独立于 Pod 的生命周期. NFS (Network File System) 作为 FreeBSD 支持的网络文件系统, NFS 节点之间通过 TCP/IP 协议访问, NFS 的客户端应用可以透明地读写位于远端 NFS 服务器上的文件, 跟本地文件访问相同.

Jupyter Hub 配置 Kubernetes 中使用 NFS, 文件系

统将被挂载在 Pod 中. NFS 允许多个 Pod 同时进行写操作, 这些 Pod 使用相同的 PersistentVolumeClaim. 通过使用 NFS 卷, 相同的数据可以在多个 Pod 之间共享.

3.2.3 用户资源分配

Jupyter Hub 用户资源分配内容一般包括了 CPU、内存和存储, 对于高性能计算还需要重点关注 GPU 资源的分配. CPU、内存和存储等资源可以直接在 Kubernetes 中弹性配置, 对于 GPU 资源需要通过驱动挂载的方式, 将宿主机上的 GPU 资源提供给容器使用. Jupyter Hub 的资源定制化配置能力, 关系到满足不同用户群体的不同层次需求, 以及用户体验的改进. 用户资源分配主要有两种方式: 资源保证 (resource guarantees) 和资源限制 (resource limits).

资源保证的方式是确保所有用户在任何情况下至少可以使用的资源量, 如果有多余的资源仍然可以在加配. 例如, 如果一个用户得到了 1 GB 内存的资源保证, 如果系统存在闲置多余的内存资源, 此用户可以分配到高于 1 GB 的内存量.

资源限制的方式设置了用户可用资源的上限, 如果一个用户只给了 1 GB 内存的资源限制, 那么此用户在任何时间、任何系统资源富余的情况下, 都不能使用超过 1 GB 内存的资源量.

3.3 集群资源调度

资源调度是 Jupyter Hub 充分利用计算集群资源和具备面向多用户开放的关键. Kubernetes 内置了默认的调度规则为 Pod 分配运行主机, 并且开放自定义编写调度算法. 调度规则区分为预选 (predicates) 和优选 (priorities) 两个阶段规则. 系统首先通过预选初步排除不符合 Pod 运行要求的主机, 然后再对预选后的主机进行量化评分, 分值高低代表主机状态优劣, 最后将分值最高的主机调度为 Pod 的运行主机.

3.3.1 预选规则与节点筛选

预选规则是在集群节点中排除不满足基本运行条件主机的规则集, Kubernetes 集群排除不符合预定条件的节点. Kubernetes 默认预选规则集见表 1.

定义 Kubernetes 集群的工作节点集合为:

$$K=\{k_0,k_1,\cdots,k_m\} \tag{1}$$

预选算法的规则集合为:

$$C=\{c_0,c_1,\cdots,c_n\} \tag{2}$$

定义规则的运算结果:

$$c_i(k_j)=\begin{cases}\{k_j\}, & \text{当节点}k_j\text{满足}c_i\text{条件}\\ \emptyset, & \text{当节点}k_j\text{不满足}c_i\text{条件}\end{cases},\forall i,\forall j \tag{3}$$

当节点不满足预选算法规则集合中的任一规则, 则节点被筛选掉, 最终通过预选规则过滤得到的集合为:

$$\tilde{K}=\bigcup_{j=0}^{m}\bigcap_{i=0}^{n}c_i(k_j) \tag{4}$$

表 1 Kubernetes 默认预选规则集

编号	规则名	功能
C0	NoDiskConflict	Pod volume 和节点已存在的 volume 是否冲突
C1	NoVolumeZoneConflict	检查在此主机上部署 Pod 是否存在卷冲突
C2	PodFitsResources	节点是否有足够资源满足 Pod 的运行需求
C3	PodFitsHostPorts	Pod 所需的 HostPort 是否被占用
C4	HostName	是否满足指定字段中指定节点主机名
C5	MatchNodeSelector	节点标签是否匹配 Pod 的 nodeSelector 属性要求
C6	MaxEBSVolumeCount	确保已挂载的 EBS 存储卷不超过设置的最大值
C7	MaxGCEPDVolumeCount	确保已挂载的 GCE 存储卷不超过预设的最大值
C8	CheckNodeMemoryPressure	判断节点是否已经进入到内存压力状态
C9	CheckNodeDiskPressure	判断节点是否已经进入到磁盘压力状态

3.3.2 优选算法与节点评分

在筛选出符合基本要求的候选节点后, 通过优选算法的节点评分决定 Pod 容器具体调度运行的集群宿主机. 优选算法通过不同的评分函数以及其权重, 得出候选节点的综合分值为评分函数加权求和, 最后根据得分情况, 将容器运行在最佳节点上.

$$s(k)=\sum w_i p_j(k), k\in\tilde{K} \tag{5}$$

$$\sum_{i=1}^{n}w_i=1 \tag{6}$$

其中, $s(k)$ 为优选算法的节点综合评分; w_i 为权重系数, 权重系数一般为均匀等比例分配, 也可以由用户根据主观情况进行自定义分配调整, $p_j(k)$ 为评分函数.

Kubernetes 中默认内置了几种典型评分函数:

(1) LeastRequestedPriority, 当新的 Pod 被调度到节点上, 节点的优先级取决于节点空闲资源与节点总容量的比值, 比值越高的节点得分越高, CPU 和内存的权重相同, 具有参照资源消耗跨节点调度 Pod 的作用.

(2) BalancedResourceAllocation, 尝试分析部署 Pod 后集群节点的 CPU 和内存利用率, 以达到集群均衡状态为目标, 可以避免出现 CPU、内存负载不均衡的情况.

(3) SelectorSpreadPriority, 主要针对多实例的场景下使用, 将优先级交给运行 Pod 实例数量少的节点.

4 性能测试

为了对比 Jupyter“微服务化”分布式架构改进性能, 将 Jupyter 单体服务架构应用与基于 Kubernetes 集群架构的两种不同部署方式进行测试, 测试内容包括了两个部分: 并发访问负载测试、Pod 运行数量负载均衡测试. 其中, 访问负载测试的是为了对比两种架构方式下后台服务的响应能力; Pod 运行数量负载均衡测试是通过观察集群中 Pod 运行数量变化与节点计算资源使用率间的变化关系, 分析验证集群计算资源调度分配的有效性.

4.1 测试条件

本文研究重点关注资源调度和横向扩容测试, 为了方便实验, 可以控制测试节点规模, 本实验中搭建了 3 个节点 (节点配置如表 2 所示) 的 Kubernetes 集群, 其中 1 个 master 集群控制节点、2 个 Node 工作节点.

表 2 节点配置

类型	配置
内存	8 G
CPU	2×8 核 2.2 GHz
网口	1000 Mbps
OS	Ubuntu 16.04 Xenial
Docker	Version 1.12.3

4.2 访问负载测试

Apache Bench (AB) 是一种简单有效的压力负载测试工具包, 通常用于对服务器的 HTTP 请求性能测试. 本研究通过 AB 工具分别对基于 Kubernetes 微服务架构和单体架构的 Jupyter 负载能力进行测试, 测试模拟 100 个用户共 20 000 次服务请求的响应情况, 测试结果主要关注“平均响应时间 (time per request)”[13,14] 指标, 其计算公式如下:

$$Time\_per\_request = T \times C/N \tag{7}$$

其中, T 代表请求测试花费的总时间, N 代表总请求数量, C 代表并发用户数.

如图 6 测试结果数据统计分析表明, 在相同的访问量和用户数条件下, 在访问请求总数的各个百分比分位点上, 基于微服务架构的 Jupyter 在“平均响应时间”指标上优于单体架构.

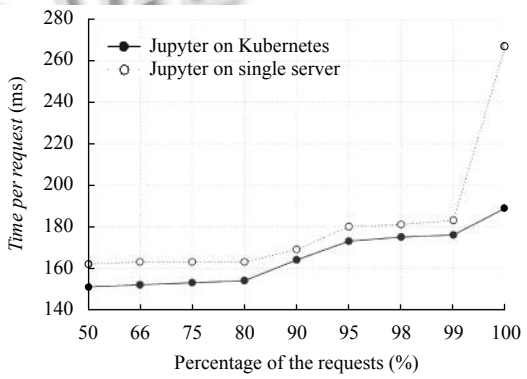


图 6 访问负载测试结果数据统计分析

4.3 Pod 运行数量负载均衡测试

基于 Kubernetes 分布式微服务架构的 Jupyter, 通过为不同的用户启动独立的 Pod 提供 Notebook 服务, 其 Pod 运行数量即代表用户运行数量. Pod 运行数量负载均衡测试模拟随着用户运行数量增加, 集群节点的资源使用率变化情况, 主要关注“内存使用率 (Memory Usage)”和“CPU 使用率 (CPU Usage)”两个指标, 计算公式见式 (8) 和式 (9):

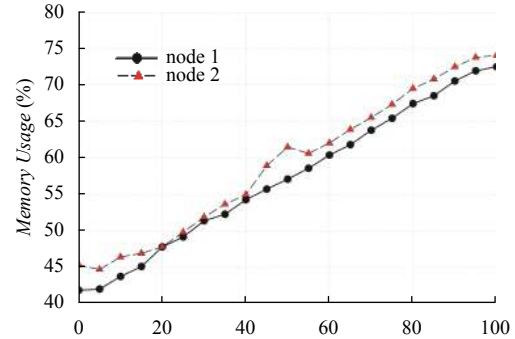
$$Memory\_Usage = \frac{\sum_{i=1}^{m} memory\_container\_usage\left(c^{i}_{node(j)}\right)}{node\_memory(node(j))} \tag{8}$$

$$CPU\_Usage = \frac{\sum_{i=1}^{m} cpu\_container\_usage\left(c^{i}_{node(j)}, \Delta t\right)}{node\_cores(node(j))} \tag{9}$$

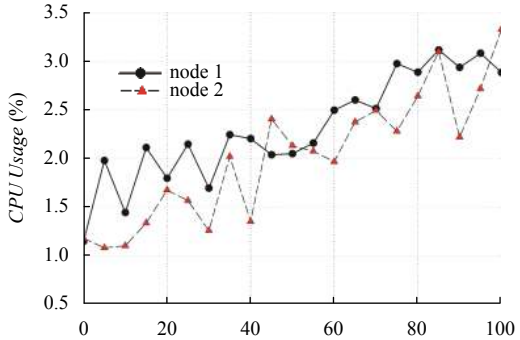
其中, 假设集群节点 $node(j)$ 共计运行 m 个容器实例, $c^{i}_{node(j)}$ 代表节点 $node(j)$ 序号为 i 的容器实例, $memory\_container\_usage$[15] 为节点某个容器内存使用率, $cpu\_$

container_usage[15]为节点某个容器在Δt时间间隔内的 CPU 使用率, *node_memory*为某节点总计内存资源量, *node_cores*为某节点 CPU 计算资源总量.

如图 7 所示测试结果数据分析表明, 随着用户数量的增加, 集群不同节点的资源使用率呈现总体同步提高的变化趋势, 证明了计算资源的负载被均衡分配到集群各个宿主节点上. 通过资源分配算法, Kubernetes 在集群上启动 Pod 单元时, 优先考虑宿主机的剩余计算资源, 将负载优先分配到计算资源、环境条件相对优越的节点上, 从而实现了在整个集群上的负载均衡.



(a) 内存使用率



(b) CPU 使用率

图 7 Pod 运行负载测试结果数据分析, node1、node2 为测试集群中的两个计算节点, 为 Pod 的运行宿主机.

5 结语

Jupyter 作为集合了代码运行、数据分析、论文编写等功能为一体的优秀平台工具, 逐渐走向大型计算中心的后端, 向着基于容器技术的分布式微服务架构方向演进. 本文研究通过对 Jupyter 的微服务化重构, 并结合 Kubernetes 资源调度分配算法, 实现了在计算集群上多用户资源需求的负载均衡, 能够充分优化利用计算中心的集群资源[16], 具备了横向动态扩容的能力, 为用户提供更为便捷、高效的计算资源访问方式. 在下一步的研究中, 将重点关注面向 HPC 高性能计算的 Jupyter 分布式并行架构, 以优化计算资源利用率为主要目标[17], 并融合主流大数据存储计算框架, 为计算中心和用户构建易于管理操作的交互式分析计算平台.

参考文献

1 Project Jupyter team. The Jupyter notebook. https://jupyter-notebook.readthedocs.io/en/stable, [2018-06-20].

2 Somers J. The scientific paper is obsolete. https://us-issues.com/2019/01/11/the-scientific-paper-is-obsolete/, [2018-04-05].

3 Braun N, Hauth T, Pulvermacher C, *et al*. An interactive and comprehensive working environment for high-energy physics software with python and jupyter notebooks. Journal of Physics: Conference Series, 2017, 898(7): 072020.

4 Ustyuzhanin A, Head T, Babuschkin I, *et al*. Everware toolkit. Supporting reproducible science and challenge-driven education. Journal of Physics: Conference Series, 2017, 898(7): 072051.

5 Yu W, Carrasco Kind M, Brunner RJ. Vizic: A Jupyter-based interactive visualization tool for astronomical catalogs. Astronomy and Computing, 2017, 20: 128–139. [doi: 10.1016/j.ascom.2017.06.004]

6 Richardson ML, Amini B. Scientific notebook software: Applications for academic radiology. Current Problems in Diagnostic Radiology, 2018, 47(6): 368–377. [doi: 10.1067/j.cpradiol.2017.09.005]

7 Fischer CR, Ruebel O, Bowen BP. An accessible, scalable ecosystem for enabling and sharing diverse mass spectrometry imaging analyses. Archives of Biochemistry and Biophysics, 2016, 589: 18–26. [doi: 10.1016/j.abb.2015.08.021]

8 Piparo D, Tejedor E, Mato P, *et al*. SWAN: A service for interactive analysis in the cloud. Future Generation Computer Systems, 2018, 78: 1071–1078. [doi: 10.1016/j.future.2016.11.035]

9 Project Jupyter Team. Technical reference. https://jupyterhub.readthedocs.io/en/stable/reference/index.html, [2017-11-07].

10 Lewis J, Fowler M. Microservices. https://martinfowler.com/articles/microservices.html, [2014-04-25].

11 Newman S. 微服务设计. 崔力强, 张骏, 译. 北京: 人民邮电出版社, 2016. 161–166.

12 The Linux Foundation. Documentation for kubernetes v1.7. https://v1–7.docs.kubernetes.io/docs/concepts, [2018-06-23].
13 贺宗平, 李光瑞, 张晓东. 面向大数据安全访问认证的 LDAP 集成架构设计. 智能计算机与应用, 2018, 8(1): 95–98. [doi: 10.3969/j.issn.2095-2163.2018.01.024]
14 The Apache Software Foundation. ab - Apache HTTP server benchmarking tool. http://httpd.apache.org/docs/current/programs/ab.html, [2018-09-05].
15 Prometheus Authors. Querying Prometheus. https://prometheus.io/docs/prometheus/latest/querying/basics, [2018-10-12].
16 林伟伟, 齐德昱. 云计算资源调度研究综述. 计算机科学, 2012, 39(10): 1–6. [doi: 10.3969/j.issn.1002-137X.2012.10.001]
17 陈新宇, 刘倩, 张宝花, 等. 基于高性能计算环境的科学计算应用平台设计与实现. 科研信息化技术与应用, 2016, 7(5): 43–51.