云环境下大规模分布式计算数据感知的调度系统

刘汪根[1]，郑淮城[1]，荣国平[2]
1. 星环信息科技（上海）有限公司，上海 200233；2. 南京大学软件学院，江苏 南京 210093

摘要
介绍了新的调度系统，包括资源调度、应用编排、配置标签中心、云网络和云存储服务等子系统。系统通过数据拓扑感知能力保证了计算和数据的局部性，节约网络I/O开销；通过优化点对点大数据量读取的资源调度，解决网络风暴造成的影响；通过网络和磁盘隔离技术以及可抢占的方式来保证服务等级协议。

关键词
云计算；调度系统；大数据；AI平台；数据局部性；分布式计算；抢占

中图分类号：TP315 文献标识码：A doi: 10.11959/j.issn.2096-0271.2020007

A scheduler system for large-scale distributed data computing in cloud

LIU Wanggen[1], ZHENG Huaicheng[1], RONG Guoping[2]
1. Transwarp Technology (Shanghai) Co., Ltd., Shanghai 200233, China
2. Software Institute of Nanjing University, Nanjing 210093, China

Abstract
A novel scheduler system including resource scheduling, application scheduling, configuration and label management center, cloud network and cloud storage services was introduced. The locality of computation and data was ensured by the ability of data topology awareness, and the I/O cost was saved. The impact of network storm was solved by optimizing the resource scheduling of point to point large data reading. The service level protocol was guaranteed by network and disk isolation technology and preemptive way.

Key words
cloud computing, scheduling system, big data, artificial intelligence platform, data locality, distributed computing, preemption

1 引言

随着云计算、大数据与人工智能（artificial intelligence，AI）技术的发展以及行业对这3种技术的综合应用需求，目前国内外出现了大数据、云计算、人工智能三者相互融合的技术发展趋势。中国计算机学会大数据专家委员会在2019年大数据发展趋势调查报告中指出：人工智能、大数据、云计算将高度融合为一体化系统[1]。

2006年Hadoop项目的诞生标志着大数据技术时代的开始，而亚马逊云计算服务（Amazon web services，AWS）商用则标志着云计算正式迈出了改变信息时代的步伐。自此之后，大数据和云计算成为最近十几年十分火热的技术，学术界和工业界都大力投入相关技术研发和落地应用中，并且创造了几个学术界和工业界协作的经典之作。如2012年加州大学伯克利分校推出的新型计算引擎Spark技术迅速被工业界接受，并成为新一代的标准计算引擎。在云计算领域，更多的是企业级应用推动了技术的发展，如2014年开始兴起的容器技术和编排系统，最终推进了新一代原生云平台的快速发展，Docker和Kubernetes[2]技术则成为新一代原生云的标准。

基础软件的研发不是简单的功能堆积，必须经过详细的架构设计和功能验证。大数据和AI计算都是典型的分布式计算模型，是基于有向无环图（directed acyclic graph，DAG）[3]或者大规模并行处理（massive parallel programming，MPP）迭代的计算模式，意味着计算任务都是运行时才能生成的，因而难以进行预先调度，而分布式的特点又要求调度系统有更高的灵活性和自适应性。目前工业界在努力尝试将大数据平台和AI技术部署在原生云上，以做到更大的弹性和可扩展性。但是，目前全球范围内在这个领域取得的突破性成就还比较少。本文将介绍核心创新：云平台中的核心调度系统如何在原生云平台上管理和调度大数据和AI应用。

2 云原生

云原生（cloud native）[4]是指在应用开发时，以云作为其生产部署方式，从而充分利用云的弹性、可扩展、自愈等核心优势。区别于传统的臃肿的单体应用开发模式，云原生应用因为其有效的协同开发、测试和运维，极大地提高了软件开发效率和质量，支持产品快速地上线和迭代，已经成为当前主流的应用开发模式。

通常称能够有效支撑云原生应用的平台为原生云平台，其主要特点是容器化的封装、自动化的管理以及面向微服务的体系。Docker直接使用Linux Cgroup等技术提供逻辑上的虚拟化，因其系统开销小、启动快、隔离性较好、方便应用封装等特点，Docker已经成为首选的应用容器技术。此外Docker技术支持跨平台部署，能够实现“一次编译，多次部署”。基于虚拟机的技术对高负载的应用可能有30%的性能损耗，而Docker技术没有硬件虚拟化，跟裸机相比几乎没有性能损耗，因此也可以用于部署类似大数据和AI应用的计算或者I/O资源密集型的应用。

一个大的云平台可能需要高效稳定地运行数万个容器，这就需要非常强大的服务编排系统。为了满足日益增多的服务和任务的资源需求，Borg[5]、Mesos[6]、Omega[7]等一系列的集群资源调度系统

相继出现。其中，Borg是集中式调度器的代表，其作为集群调度器的鼻祖，在Google公司有超过10年的生产经验，其上运行了GFS[8]、BigTable[9]、Gmail、MapReduce[10]等各种类型的任务。Mesos是双层调度器的代表，可以让多个框架公平地共享集群资源。Omega则是共享状态调度器的代表，它的特点是支持使用不同的调度器调度不同类型的任务，解决资源请求冲突的问题。

Kubernetes是Google开源用来管理Docker集群的项目，继承了Borg的优点，实现了编排、部署、运行以及管理容器应用的目的，Kubernetes的总体架构如**图1**所示。Kubernetes提供资源池化管理，可以将整个集群内的中央处理器（center processing unit，CPU）、图形处理器（graphic processing unit，GPU）、内存、网络和硬盘等资源抽象为一个资源池，可以根据应用的资源需求、资源池中的实时资源情况进行灵活调度；Kubernetes包含一个统一的调度框架，最多可以管理数千个服务器和数万个容器，同时提供插件化的接口，让第三方来定制和扩展新的调度系统；此外Kubernetes支持通过ConfigMap等方式动态地调整应用配置，从而具备动态调配的基础能力。

本文基于这些基础技术开发了支持复杂应用平台的调度系统。

3 大数据和AI计算模型

分布式计算在复杂的工业应用需求下快速迭代和发展，从离线处理的MapReduce[5]计算模型开始，逐渐发展出了以Spark为代表的在线计算（Spark、Tez[11]、Druid[12]等）以及实时计算模型（Flink[13]、Spark Streaming[14]等）。新的分布式计算模型开拓了新的应用领域，同时也对大规模系统的管理、调度和运维提出了较大的挑战。

3.1 MapReduce

MapReduce框架是Hadoop技术的核

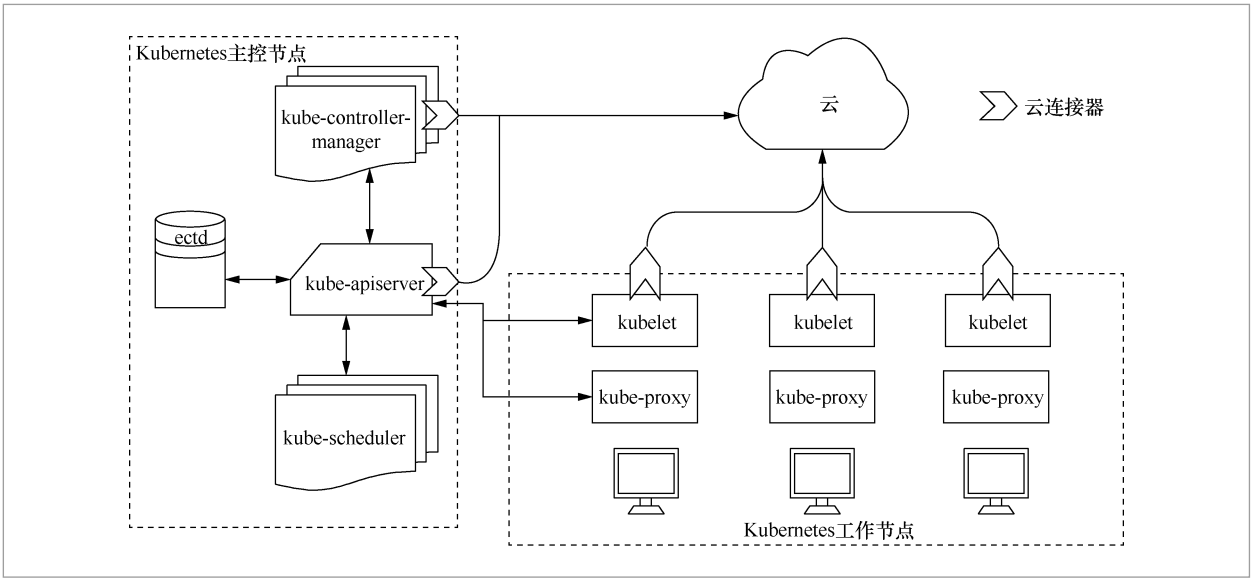

图1 Kubernetes的总体架构

心，它的出现在计算模式历史上是一个重大事件，在此之前行业内主要通过MPP的方式增强系统的计算能力，一般是利用复杂而昂贵的硬件加速计算，如高性能计算机和数据库一体机等。而MapReduce是通过分布式计算来提高计算速度的，并且只需要廉价的硬件就可以实现可扩展的、高并行的计算能力。

3.2 Spark

随着大量的企业开始使用Hadoop构建企业应用，MapReduce性能慢的问题逐渐成为研究瓶颈，MapReduce只能用于离线数据处理，而不能用于对性能要求高的计算场景，如在线交互式分析、实时分析等。在此背景下，Spark计算模型诞生了。虽然本质上Spark仍然是一个MapReduce计算模式，但是几个核心的创新使得Spark的性能在迭代计算应用中比MapReduce快一个数量级以上：第一，数据尽量通过内存进行交互，相比基于磁盘的交换，能够避免I/O带来的性能问题；第二，采用延时评估（lazy evaluation）的计算模型和基于DAG的执行模式，可以生成更好的执行计划。此外，通过有效的检查点（check pointing）机制可以实现良好的容错机制，避免内存失效带来的计算问题。

3.3 TensorFlow

TensorFlow[15]是Google公司开发的第二代人工智能学习系统，它可以将复杂的数据结构传输至人工智能神经网络中进行分析和处理，可用于语音识别、图像识别等多项机器学习和深度学习任务，并且完全开源，目前也是开源社区活跃的开发项目之一。TensorFlow支持异构设备的分布式计算，可以在各个平台上自动运行模型，支持在手机、单个CPU/GPU、大型数据中心服务器等各种设备上运行。

4 原生云与大数据、AI融合的技术难点和解决思路

4.1 基于容器云开发大数据或AI产品的技术难点

原生云平台的一个主要特点是面向微服务设计，使用容器的方式编排普通的Web应用或者微服务。但是大数据或者AI计算平台本身并没有采用微服务设计，各个系统都采用自身的分布式系统设计完成服务的管理和调度，并且执行时服务会一直变化。以MapReduce为例，每个MR程序启动Map或者Reduce工作任务的数量是由应用本身来指定的（通过参数set mapreduce.reduce.tasks=*N*），调度器通过解析相关的参数来生成给定数量的工作任务，并在相关的任务完成后立即销毁。

大数据系统内的服务有自己的服务重启或者迁移逻辑，并且其配置或参数可能会随着应用或用户的输入而变化，而很多组件之间有很长的依赖链，如何及时地将某个参数的变化推送到所有的下游服务中，是调度系统的一个重要挑战。

对于没有数据存储的无状态服务，容器有多种编排方式进行编排和管理。但是对于有数据状态的服务，如数据库（MySQL）或者分布式存储服务（Hadoop distributed file system，HDFS），由于容器内的数据会随着容器的销毁而销毁，因此采用传统的编排方式无法保证数据状态的一致性和数据持久化。

大数据和AI计算都涉及大量的数据和复杂的迭代运算，为了达到更好的性能，

MapReduce、Spark和TensorFlow在架构中都特别注重“计算贴近数据”设计，一般会根据数据的位置选择启动相应的计算服务，尽量让计算任务和数据节点在同一个节点上，这样就可以避免网络带宽带来的性能损失。而在容器模式下，不同的服务封装在不同的容器内，并且使用容器自身的网络；而容器网络一般采用重叠网络（overlay network）模式，因此容器内的应用并不能感知调度容器的机器的物理拓扑，因此也就无法确定从哪个节点调度计算任务所在的容器。为此，重新设计了云网络服务，以有效解决相应的调度信息缺失问题。

另外，大数据或者AI应用都是资源密集型应用，在运行时会对CPU、GPU、内存、磁盘、网络等资源进行动态申请和释放。如某个应用可能会根据用户的输入生成一个Spark的机器学习任务，在大数据平台上生成若干个执行任务（executor），并申请一定的CPU和内存资源。及时地响应这些短生命周期服务的资源需求，是当前各种原生云的调度系统无法有效支持的功能。

4.2 融合大数据和AI的云调度系统的设计考量

为了有效解决各种实际技术难题，重新设计云平台的调度系统成为必须考虑和完成的工作。相较于原生的Kubernetes调度系统，云平台的调度系统需要从以下方面重新设计。

（1）基于资源需求的调度能力

整个云平台的资源从逻辑上被抽象为一个大的资源池，按照CPU、GPU、内存、存储和网络进行分类管理。一个容器在被调度的时候，通过编排文件或者配置来描述相应的资源需求。调度器需要根据资源池的实际资源情况和各个主机的物理资源情况，在毫秒级时延内找到合适的物理节点来调度当前容器。在超大规模的云集群里，调度器的效率、性能和负载均衡是重要的技术指标，最终会影响云平台的资源使用率。

（2）支持本地存储并基于存储感知的容器调度

为了有效支持有状态的服务，如数据库服务（MySQL）、分布式存储服务，本文设计了基于本地存储的存储服务。云储存将所有的存储资源抽象为一个存储池，每个节点的本地存储抽象为若干持久化存储池（persistence volume group），而单个可以被容器申请的存储资源定义为存储卷（persistence volume，PV）。每个PV只能被一个容器挂载，可以是任意大小，支持硬盘驱动器（hard disk drive，HDD）或固态硬盘（solid state drive，SSD），也支持HDD与SSD组成的分层式存储。有状态的容器服务在启动时会申请需要的PV资源（如大小、IOPS要求、SSD或HDD），调度器需要在毫秒内从平台内找到合适的机器，从而将容器调度起来；如果容器因为各种原因不能正常工作，调度器能够检测到不健康的状态，同时根据数据的物理拓扑情况，选择合适的机器重启相应的容器。

（3）网络物理拓扑的感知与实时调度能力

为了支持灵活弹性的调度，一般情况下云原生平台为容器设计专门的网络空间，并且一般与主机网络保持透明。通常一个服务器节点上会有多个容器服务，因此主机IP的数量远小于容器IP的数量。在MapReduce、Spark和TensorFlow的设计中，在启动计算任务时，会根据数据存储的物理拓扑决定最终服务启动在哪个服务器上。但是在容器平台上，所有的任务都只能感知容器网络，而不能感知物理机器的

IP，无法了解数据分布的物理拓扑，因此也就无法保证计算和数据的局部性。为了解决这个问题，在调度系统内增加了网络物理拓扑模块，实时地维护物理IP和容器IP的映射关系表。当有状态服务在申请调度的时候，可以通过IP映射表找到可用于最终调度的物理机器。

（4）动态标签能力与基于标签的调度能力

复杂的应用系统对调度系统有更多细节的要求，如多租户业务系统往往要求一个租户的所有服务尽量调度在同一批物理机器中；支持高可用的服务往往要求调度系统有反亲和性（anti-affinity）要求，即将一个服务的多个实例调度到不同的物理机器上，这样就不会发生一个物理机器宕机后影响多个高可用实例，进而导致服务不可用的情况。因此在调度系统中增加了动态标签能力和基于标签的调度能力，调度模块可以根据运行时的数据给物理机器和容器应用动态增加或删除标签，而调度系统根据标签的匹配度选择更合适的调度策略。

（5）应用依赖运行参数的感知以及基于参数的调度能力

分布式应用在微服务拆分后，每个服务都有比较复杂的依赖服务链以及影响的服务链，而每个微服务在运行时都可能发生运行参数变化、服务启停、重新调度等操作，这些变化不仅影响当前的微服务，还可能影响所有的依赖链的服务。因此调度系统中增加了应用配置感知模块，通过与全局的配置中心的数据交互，调度系统能够实时地感知一个应用变化的事件的所有影响情况，并根据多个可能的状态迁移图选择最合适的调度策略。

（6）不同优先级下基于抢占的调度能力

在业务负载比较重并可能超过云系统内资源总量的时候，需要有机制保证高优先级的服务正常运行，牺牲低优先级应用的服务质量，也就是服务质量等级保证（service level agreement，SLA），这是云平台的一个重要特性。在调度系统内增加了基于抢占的调度模式，将服务和资源按照优先级进行区分，如果出现资源不足以调度高优先级应用的情况，则通过抢占的方式调度低优先级应用的资源。

5 融合大数据和AI的云调度系统的实现

类似于操作系统的调度模块，云平台的调度是整个云平台能够有效运行的关键技术。云平台的总体架构如**图2**所示，最底层是Kubernetes服务，其上层运行着自研的几个产品或服务。其中，配置中心用于实时地收集和管理云平台内运行的所有服务的配置参数，支持运维人员手动地对一些服务进行参数设置或管理；物理资源池是通过各个服务进行池化后的逻辑资源，应用申请的物理资源都会从这个资源池中逻辑地划分出去，在应用使用完结后再返还给资源池，平台会给不同的租户做资源配额限制；云存储服务是基于本地存储开发的分布式存储服务，会保留状态服务的数据，保证应用数据的最终持久化和灾备能力；云网络服务是自研的网络服务，提供应用和租户类似虚拟私有云（virtual private cloud，VPC）的网络能力，可以做到完整的资源隔离和服务等级协议（service level agreement，SLA）管控；标签中心用于实时地收集和管理各个应用、主机、资源的标签，支持动态的标签修改与实时调度触发。

在此之上是云调度系统，它接收应用的

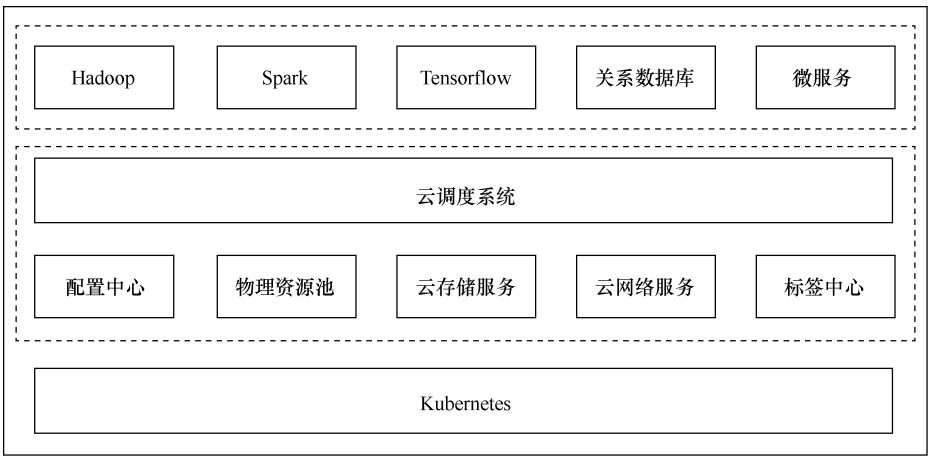

图 2 调度系统的总体架构

输入，从配置中心、标签中心、云存储服务和云网络服务中实时获取平台运行指标，并从物理资源池中获取资源的使用情况，从而根据运行时的信息进行精确的调度决策。云调度系统之上是各类应用服务，包括大数据、AI、数据库类以及各种微服务。

5.1 调度系统架构

本文设计的云平台的调度系统的内部架构如**图3**所示，包含元信息模块、对外服务模块以及调度决策模块。当一个应用通过Kubernetes API或者命令行（基于Kubectl）的方式启动时，它会传入一个指定服务编排方式的yaml文件以及启动应用需要的容器镜像（docker image）。在yaml文件中会指定当前应用需要的资源、与调度相关的标注信息（或标签信息）以及对其他应用的依赖关系。

对外服务模块负责解析编排文件，同时负责完成应用的依赖解析和管理，生成对服务的依赖图；参数计算模块负责编排文件中与配置变量相关的计算，并生成最终的资源需求；实例渲染模块最终生成一个完整的应用描述，并将这个描述文件提交给调度决策模块。

元信息模块负责与Kubernetes平台交互，实时地维护用于调度的各种元数据。资源池数据主要包括当前集群的各种资源总量以及当前的使用情况，并可以精细到各个节点和硬盘级别；网络拓扑模块负责与容器网络进行交互，实时地维护当前容器网络与主机网络的拓扑；存储拓扑模块负责与云存储服务交互，实时记录当前存储池的各个存储卷（PV）的容量使用情况和每秒进行读写操作的次数（input/output operations per second，IOPS）；服务运行指标主要同步当前集群各个物理节点上各个容器的运行指标以及各个物理机的各项资源的使用指标；配置元数据模块主要协助配置中心维护各个应用的实时配置参数。因此，通过元信息模块，调度器能够维护当前云平台的各种调度决策数据，从而实时地根据运行数据进行最优化调度。

在对外服务模块提交了实时渲染的应用描述请求后，调度决策模块会根据请求中的资源大小，通过内部的6个调度器进

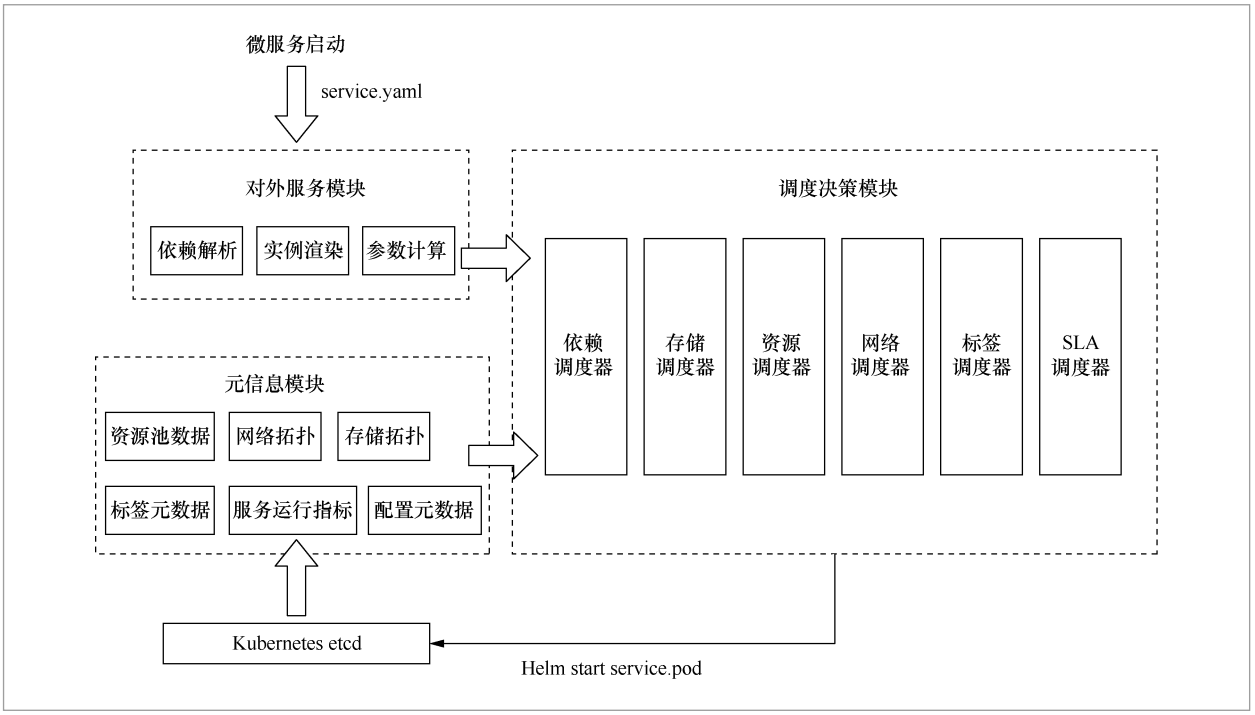

图 3 调度系统的内部架构

行调度目标的选择。依赖调度器会解析应用的需求，同时遍历当前的服务元数据，查找是否有已经运行的可以给当前应用依赖的微服务。如果有，则直接使用该服务，并更新相应的配置参数；如果没有，则解析被依赖的服务的参数，同时选择运行一遍完整的调度流程。如果有嵌套服务依赖，则递归调用相关的调度逻辑，直至所有依赖链的服务都处于运行状态。存储调度器提供基于存储感知的容器调度能力，它会解析应用是否有明确的本地数据的要求，如果有，则从存储拓扑中找到合适的物理节点组，并将其他节点标记为此次调度无效；资源调度器负责找到有足够资源启动应用或者容器的物理节点组；网络调度器负责根据应用的网络IP要求选择满足网络规则的物理节点组；标签调度器负责选择满足各种标签规则的物理节点组。通过这5个调度器选择的物理节点如果有多个，调度器则根据负载均衡的原则选择合适的物理节点，从而启动这个容器；如果没有任何节点有足够的资源，SLA调度器会遍历被选择指定节点上已经在运行的所有容器，找到当前被优先级更低的应用占用的容器，并选择kill等机制，循环此操作，直到资源满足本次调度为止，从而实现高优先级应用的抢占式调度策略。

5.2 服务的编排系统

应用服务的调度需求包含依赖的服务、服务自身的Docker镜像、资源需求、网络与存储需求、配置参数等各个方面的属性，因此选择Jsonnet语言进行应用的资源描述文件。Jsonnet是Google公司开源的新一代JSON协议，支持引用、算术运算、条件操作符、函数、变量、继承等强大的计算能力，可以描述多样化的需求。

一个简单的基于Jsonnet的资源编排文件示例如**图4**所示。其中app.yaml 描述

依赖关系和依赖变量，config.jsonnet描述配置参数和定义输入，而customize-main.jsonnet 定义模板入口和输出变量。在运行时，通过服务模块的实例渲染组件即可将这些编排文件和当前平台内的运行参数进行实例化，渲染出一个可以被Kubernetes调度的service/replication controller或者deployment的描述文件。

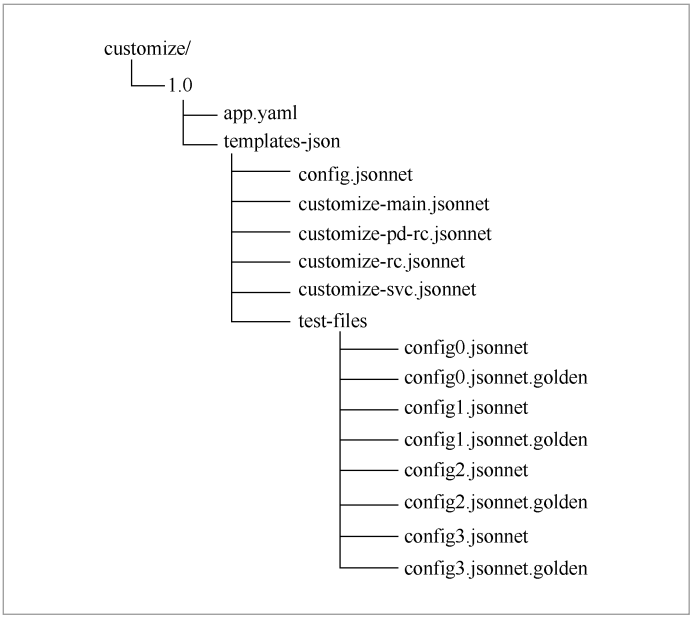

图 4 基于 Jsonnet 的资源编排文件示例

```
kind: ConfigMap
metadata:
  labels:
    transwarp.hash:ac45a765
    transwarp.name: zookeeper-confd-conf
  name: zookeeper-confd-conf-gm2cz
  namespace: systemtenant
apiVersion: v1
data:
  jaas.conf.tmpl: |
    {{- if eq (getv "/security/auth_type") "kerberos" }}
    Server {
      useKeyTab=true
      keyTab="/etc/keytabs/keytab"
    };
    Client {
      com.sun.security.auth.module.Krb5LoginModule required
      useKeyTab=false
    };
```

图 5 ConfigMap 的示例

5.3 配置标签中心

由于每个服务运行时都可能发生重启或者迁移，且API网关等可能会动态地设置运行参数，为了让服务能够及时地感知各种运行参数，创建了一个统一的配置标签中心作为中心化的服务，提供配置和标签元数据的管理和查询。应用程序使用的配置基于Kubernetes ConfigMap的方式来实现配置与容器镜像的解耦。ConfigMap的示例如图5所示，包含了标签和数据属性，都是以键-值数据对的方式来保存的。容器可以直接引用相应的ConfigMap中的数据，而这些配置数据则存储在配置标签中心服务中。用户可以通过界面或者命令行的方式修改配置的数值或者增加新的键-值对，并且当有任何配置数据修改后，Kubernetes会动态地修改每个节点的ConfigMap文件，并在短时间内让容器知晓相应的数据变化，从而实现中心化的动态配置和标签修改。

由于配置和标签中心服务中包含了运行时的各种配置和标签数据，因此调度器通过与配置标签中心交互就可以获取整个云平台的标签和配置元数据，从而可以支持基于标签的调度能力。

5.4 云存储服务

云平台中的存储服务Warpdrive提供一个中心化的RESTful服务，可以支持外部服务对存储卷的实际使用情况的实时查询，因此调度器通过监听相关的接口来实时感知每个主机的存储卷使用情况以及每个容器与物理节点的绑定关系。

5.5 云网络服务

网络服务也同样提供RESTful API查询服务，使得调度器可以知晓各个容器IP和物理IP的使用情况以及各个容器的网络防火墙规则等实时数据，因此调度器能够实时地感知云平台内的网络和存储实际运行情况。

5.6 计算调度过程中的数据拓扑感知

在容器云计算环境下，分布式网络中的各个主机可以是一个或多个独立的网络，每个网络都可以包括一个或多个子网络，网络地址从该主机上的子网络中获得。当容器销毁时，容器的网络地址被回收，等待下次使用。

应用感知数据拓扑的方式如图6所示，当容器中的数据应用（如Spark应用）需要启动计算任务时，调度器首先根据该服务的依赖关系找到其需要的数据应用的容器（如HDFS数据节点），然后通过元信息模块查询其所在的物理节点的网络和存储信息。元信息模块通过对存储服务和网络服务的实时查询来获取和维护最新的元数据。一般情况下，分布式存储有多个存储副本，会通过调度器的反亲和性保证调度在不同的物理节点上，因此调度器会得到一个物理节点的列表。然后根据这几个节点的实际资源使用负载情况，选择当前资源负载最低的节点来启动对应的计算服务容器。由于两个容器都在一个主机上，计算服务可以与存储服务通过本地网络进行数据传输，或者创建域套接字（domain socket）等机制来提高数据读取速度，增强性能。

在云平台集群规模比较大的情况下，网络IP数量和网段数量都比较大，而应用查询的请求也有很高的并发度。为了保证高速的网络检索能力，使用基数树（radix tree）对各个应用的网络配置信息进行缓存，基于基数树的快速匹配方法如图7所示。查询网络地址172.16.1.23，对应的二进制是10101100000100000000000100010111，在基数树中可以很快匹配到对应的子网为172.16.1.0/24，并从该子网获取所有必需的网络信息，包括容器网段、主机信息、交换机号以及机架位等。

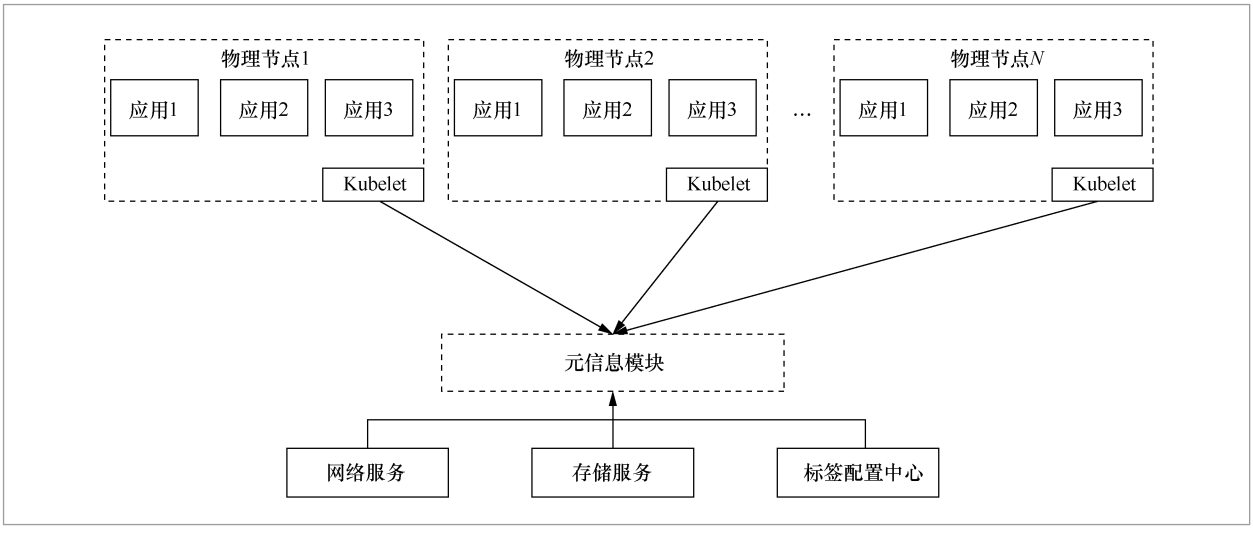

图 6 应用感知数据拓扑的方式

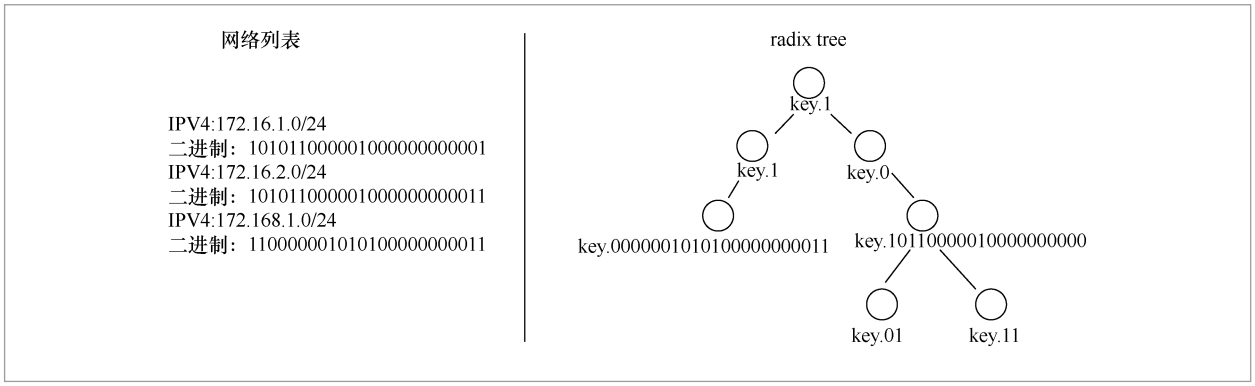

图 7　基于 Radix Tree 的快速匹配方法

5.7 调度的总体流程

对于调度策略模块，它的输入是应用的资源需求、标签属性、依赖关系和输入输出参数等信息，输出是选定的物理节点或者节点组以及对应的容器的优先级信息，同时，依赖元数据模块提供的平台运行数据进行调度决策支撑。调度器的决策流程如图8所示。

调度器从任务调度队列获取待调度的任务，形成节点-任务映射表。其中，调度器持续对API服务器中创建的任务进行监听，并读取新的任务形成任务调度队列。

（1）筛选阶段

调度器根据映射表以及预设筛选条件确定一组符合条件的目标物理节点。其中，预设筛选条件可以是当前任务所需的资源与物理节点上的剩余资源之间的匹配条件、当前任务所需的端口与物理节点上的端口之间的匹配条件以及是否带有特殊标签等。如果当前任务配置了对持久化存储的需求或者对数据拓扑的感知，调度器也会考虑满足这些条件的节点。最终调度器会将满足上述所有筛选条件的节点整理出来进行下一步处理。

（2）优选阶段

调度器根据筛选阶段拿到的节点以及预设优选条件对各个物理节点进行评分，最终选择分数最高的作为目标物理节点。其中优选条件包括：节点空闲资源越多，评分越高；节点保存的任务镜像越多，评分越高；任务亲和性/反亲和性越匹配，评分越高；同类任务分布越均匀，评分越高等。

通过上述两次筛选，选择出一个分数最高的物理节点。调度器将当前任务与目标物理节点绑定，并将绑定的信息发送到API服务器。

5.8 基于优先级的抢占式调度方法

在平台的资源不足而有更高优先级的服务需要被调度的情况下，调度器就需要进行抢占式调度。在目前的实现中，抢占式调度提供了2种方式：调度阶段的资源抢占和运行阶段的资源抢占。

调度阶段的资源抢占即在调度器筛选节点阶段，如果发现集群中所有节点都无法满足当前任务的资源需求，调度器会在集群中选择节点，并试图抢占其中正在运行的低优先级的任务的节点。抢占完成后，调度器会再次尝试调度当前任务到目

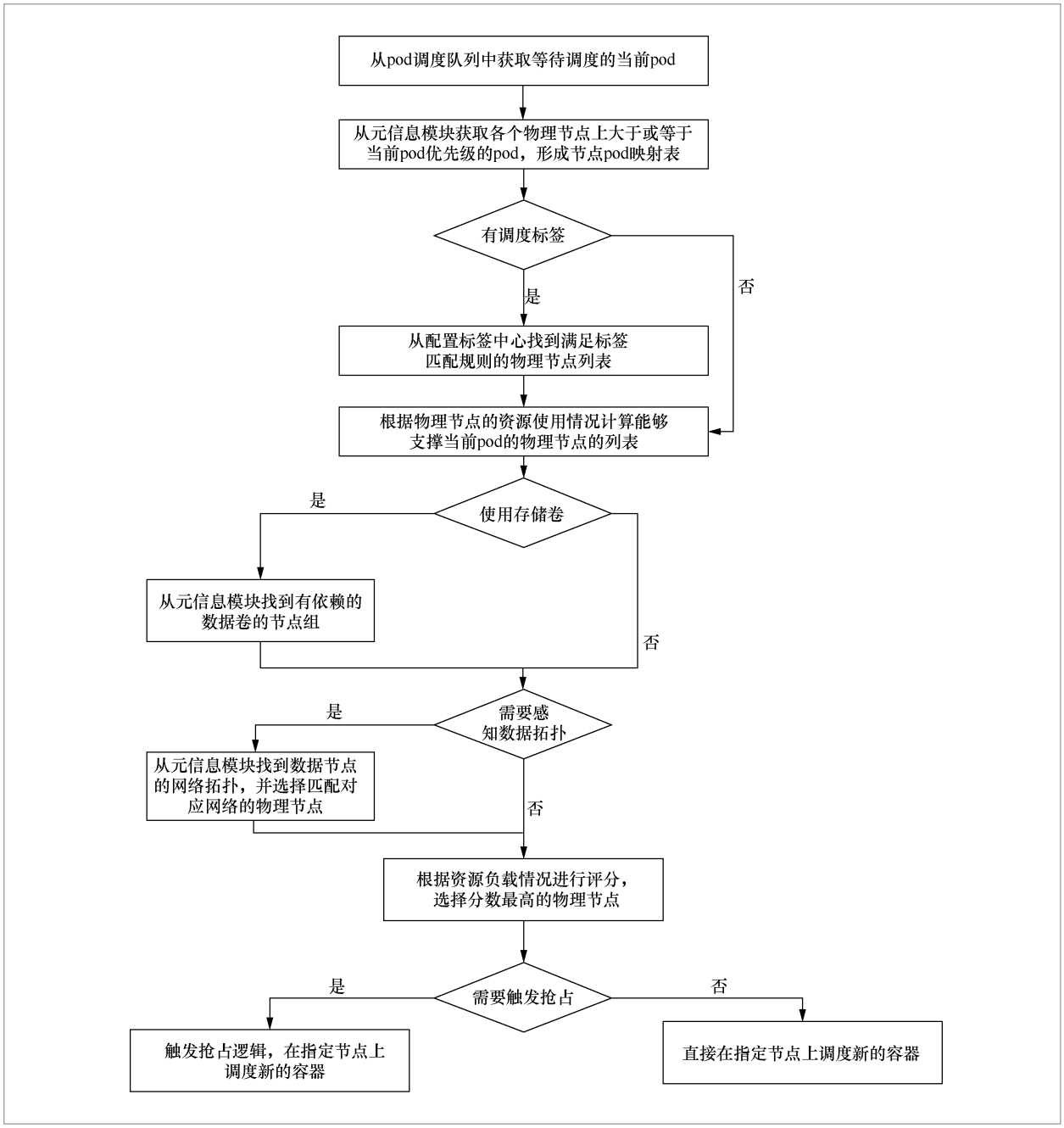

图 8 调度器的决策流程

标节点上。调度阶段的资源抢占策略如**图9**所示。

调度阶段的资源抢占实际上是一种逻辑层面的抢占，依据是任务在启动时声明的资源请求，它可以帮助系统避免任何资源超配的现象，但同时也会造成一定程度的资源浪费。比如一个声明了10 core CPU的任务可能当前只消耗了2 core CPU，此时完全不需要抢占掉，它也可以让别的任务运行起来。基于此种考虑，又引入了运行阶段的资源抢占算法，如**图10**所示。运行阶段的资源抢占的主线程逻辑如**图11**所示。

运行阶段的资源抢占会忽略优先级低于当前任务的任务，即如果有必要，可以完全抢占这些低优先级的任务，因此它们不在节点资源消耗的统计范围内。抢占阶段发生在物理节点上，当物理节点发现某种资源剩余小于预期值时，会对其上低优先级的任务进行清除，直到物理节点状态恢复正常。此种抢占方式基于物理节点的资源真实使用情况，可以保证集群资源利用最大化。

6 实验评估

为了验证数据应用尤其是大数据计算在云平台上有良好的性能表现，选择了3个基准测试程序来验证所设计实现的系统的性能。

- TestDFSIO是Hadoop自带的性能测试工具，主要为了测试Apache HDFS的I/O性能，采用MapReduce程序进行并发读写并做结果统计，主要涉及读、随机读、追加写、写等业务行为。
- HBase Performance Evaluation[16]是Apache HBase自带的性能测试工具，提供了顺序读写、随机读写、扫描等性能测试。
- TPC-DS[17]是事物处理性能委员会（Transaction Processing Performance Council，TPC）推出的用于数据决策支持系统的基准测试，包含对大数据集的统计、报表生成、联机查询、数据挖掘等复杂应用，包含了7张事实表、17张维度表和99个复杂SQL，是一个典型的数据仓库的测试场景，可以用于衡量大数据分析数据库的性能和鲁棒性。

选定好基准测试后，在50台x86服务器上部署了云平台，并且创建了3个不同的租户用于测试，每个租户都将部署Hadoop

```
for each pod p in Queue.pop():
  get noodeInfoMap from cache.
  for node in noodeInfoMap:
    qualified = prioritiezFuncs(node)
    if qualified: nodeList.append(node)
  end for
  if nodeList is not empty:
     for node in nodeList:
       score = prioritizeFuncs(node)
       if score > maxScore:  destinationNode = node
     end for
     bind(destinationNode, p)
  else:
     for node in nodeInfoMap:
       evictedPods = tryPremmption(node, p)
       if length(evictedPods) ==0 : continue
       if length(evictedPods) < minEvictedPods: NominatedNode = node
     end for
     if NominatedNode is empty:
       schedulerFailed()
     else:
       pod.NominatedNode = NominatedNode
       doPremmption(NominatedNode)
     end if
  end if
end for
```

图 9 调度阶段的资源抢占策略

```
for each pod p in Queue.pop():
  get noodeInfoMap from cache.
  for node in noodeInfoMap:
    for rpod in node.RunningPods:
      if rpod.priority < p.priority: node.delete(rpod)
    end for
    qualified = prioritiezFuncs(node)
    if qualified:  nodeList.append(node)
  end for
  if nodeList is not empty:
     for node in nodeList:
       score = prioritizeFuncs(node)
       if score > maxScore: destinationNode = node
     end for
     bind(destinationNode, p)
  else:
     schedulerFailed()
  end if
end for
```

图 10 运行阶段的资源抢占算法

```
while node.underResourcePressure():
  pods = node.runningPods
  sortByPriority(pods)
  doPreemption(pods[0])
end while
```

图 11 运行阶段的资源抢占的主线程逻辑

服务以及分析数据库Inceptor，HBase分片服务器、HDFS数据节点和Inceptor任务执行节点都有8个实例。本次测试的设计目标和期望结果见**表1**。

- 第一个租户：通过修改标签中心的配置参数让其所有的服务只能在节点1~10上调度。
- 第二个租户：通过修改标签中心的配置参数让其可以在平台的50台机器上调度，同时设置其优先级比其他微服务的优先级高。
- 第三个租户和第二个租户配置相同，也可以在50台机器上调度，但是通过修改云平台的调度测试，让其使用Kubernetes原生的调度系统，为了解决开源Kubernetes平台没有本地存储的问题，在所有的节点上都部署了同样的数据，保证任何调度到其他节点机器上的服务都能读到数据，从而确保测试可以正常运行。
- 在8个节点的裸机上部署了Hadoop服务和Inceptor，采用主机的方式部署，每个节点上部署数据节点、HBase分片服务器和Inceptor任务执行节点，所有的数据和计算服务都预先按照物理拓扑划定好，因此没有其他的调度开销。

本文没有单独针对调度系统的每个设计指标进行测试，这些设计能力都会在这个综合的性能测试里被验证，如基于资源需求的调度能力、应用依赖和运行参数感知能力，因为大数据平台本身是分布式系统，多个服务之间存在互相依赖（如HDFS依赖ZooKeeper）的关系，每个服务通过自己的编排文件指定自己需要的硬件资源以及依赖的服务，如果不具备这方面的能力，大数据服务就无法在平台上运行起来。基于网络拓扑和物理存储的调度能力以及不同优先级的抢占能力会比较多地在性能测试结果中反映出来。

按照预期，主机部署的模式应该是性能最好的方式，其次是租户1，因为它只能在较少的节点内调度，发生计算数据分离的概率比较小；租户2和租户3在50个节点内有8个数据节点，如果不用新的调度

表1 测试项的设计原则与期望结果

测试比对项	期望的结果与原因分析
租户1与主机部署	主机部署模式不存在随意的调度，所有的角色都是预先分配好的。租户1在云平台上调度，但是通过标签配置来干预其调度方式，限制其在10个节点上有限调度。如果租户1能够正常运行性能测试程序，并且性能接近主机模式，那么就可以证明基于标签的调度能力是有效的；如果性能相近，则证明调度系统对于计算和数据局部性、网络感知能力也是有效的
租户1与租户2	租户2和租户1相比，能够在更多的机器上调度，因此相对来说，计算和数据更为离散，并且可能会存在网络瓶颈。可以通过对比测试来验证随着集群规模扩大，调度系统是否可以保证稳定的调度能力，其存储和网络感知能力是否在调度系统中有效发挥作用
租户3与租户2	租户3和租户2同样都是在大规模的节点上调度的分布式系统，租户3使用的Kubernetes调度系统不会感知应用和底层数据、网络、计算等因素，因此预期租户3的测试性能和稳定性都会差一些
租户2与主机部署	期望租户2在真实云环境下运行大数据或者AI应用，即在一个大的云平台上运行海量的应用，用户无需感知细节；而主机部署方式也就是现在主流使用的采用裸金属机器部署大数据和AI的方式，没有弹性，并且不能保证资源使用率；如果租户2和主机部署方式的性能比较相近，那就表明能够在不牺牲性能的前提下，实现弹性能力以及更高的资源使用率

算法，发生计算和数据偏离的可能性比较大，但是期望租户2的性能比较好，可以接近租户1，因为调度系统可以支持计算来感知数据，而租户3的性能会比较差，因为没有数据本地性的保证。

最终的测试结果如**图12**所示，基本符合预期。在裸机上部署的集群在3个测试里的性能都是最佳的；租户1和租户2的3个测试的性能数据不相上下（差异可以认为是数据波动），说明本文的调度算法有很好的可扩展性，在集群规模扩大5倍后可以保证性能没有损失；租户3的性能要差很多，因为计算任务和数据分布是无序的，所以有很多远程读写带来的性能损失。实验证明，有效的调度系统可以在保证云平台的灵活性和弹性的同时，提供和物理平台部署一样的性能。

7 结束语

目前工业界在如何解决大数据和AI应用在云上的开发和部署问题上，基本分为3种。

第一种是采用应用或者服务内多租户的方式提供云服务，即一个数据库或者AI服务的实例给用户提供服务，用户之间的隔离由数据库或者AI平台来提供，而不是由云平台提供。比较典型的有AWS Aurora或者Google Cloud AutoML等产品。这类解决方案灵活性不高，而且由于本身计算复杂度不高或者通过SaaS服务的方式规避了复杂业务的提交，因此也可以适应业务的需求。但是这两类产品更偏向于一类应用，而不是通用的大数据或者数据库平台，因此这种方式虽然解决了部署的问题，但是不通用，只能由云服务的运营商来提供相应的解决方案。

第二种是在云平台中使用单独的裸金属机器提供大数据或者AI服务，这样可以保证大数据或者AI平台的性能，但是同时放弃了对弹性或者灵活性的要求，因此这部分裸金属机器并不能实现良好的弹性调度和高效的运维能力。

第三种是容器平台只负责调度微服务等无状态类应用，而大数据、AI平台运行在传统的基于虚拟化的云上。这样可以保证良好的调度能力，但是牺牲了大数据或者AI平台的稳定性。这个方法最简单，但是在现实案例中也发现这个方法因为稳定性等问题，几乎没有成功的落地案例。

本文讨论了与以上3种云调度方案完全不同的全新的方式，即如何在容器云平台上实现一个支持复杂的大数据和AI平台服务的调度系统，既能保证复杂的平台应用的性能，又能够有很好的调度灵活性和运维便利性。从实验结果来看，本文的设计最终在生产平台上取得了符合预期的业务

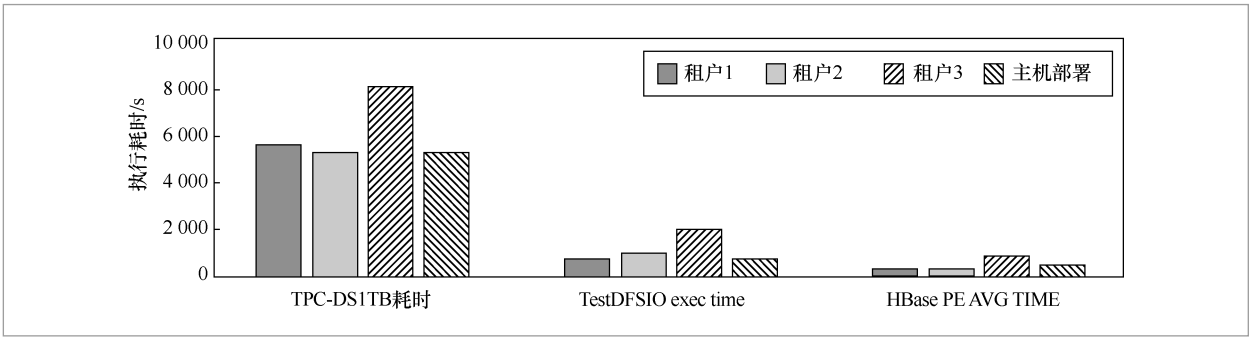

图 12 性能测试结果数据

性能数据，也证明了这个调度方法的有效性。不过受限于本系统的工程背景，本文是在实践中逐步优化了本调度系统，而不是从理论上根据调度系统的各个指标来设计系统。另外由于业务需求的驱动，本文的设计强调数据的局部性和决策的实时性，要求所有的调度决策在毫秒级完成，弱化了调度的公平性设计，而另外一些调度算法更侧重公平性[18]。

从另外一个维度，本文提出的调度系统的设计目标是给应用提供更好的性能，因此会牺牲一些其他的指标，如低优先级任务的可靠性。如为了在有数据节点的服务器上调度计算任务，会增加抢占对应节点上其他低优先级服务的概率，因而需要云平台上的微服务有更好的容错能力。考虑到计算任务也会有一些数据的操作，根据HDFS的特性，一般情况下在有更多计算任务的节点上就可能有更多的数据写入，这就要求平台对数据的生命周期有良好的管理能力，否则可能会有部分节点数据量越积越多，从而使得整个云平台出现数据分布不均衡的情况，需要定期进行人为触发的重新平衡行为。

本文主要描述了一个适用于容器云平台的调度系统，它通过实时获取Kubernetes集群的各种资源数据，结合数据和计算的拓扑特性，可以在容器平台上有效地支持大数据与人工智能的应用和服务，在带来弹性和灵活性的同时，提供了与裸机部署相同的性能，能够同时提供云计算的弹性和大数据的计算能力。通过有效的调度设计，基于容器的云平台不仅可以用于微服务类应用的DevOps支持，还可以用于大数据平台以及应用的开发和部署，支持大规模数据服务和AI服务的自动化部署和灵活调度。云计算提供基础，大数据提供生产资料，AI提供价值输出，因此，能够同时支持大数据+AI的云平台是未来的趋势，也是下一代数据中心的技术基础平台。

大规模的云平台调度是一个非常复杂的技术问题，需要反复的基于经验的调优和迭代才能够达到比较好的效果。本文的方法和实验是对多个实际运行项目中积累的观测数据和性能指标进行迭代而最终成型的，目前本文的调度系统主要调优的场景是大数据平台服务（如Spark、Hadoop、TensorFlow等）以及数据分析类微服务（如各种分析类报表、实时事件分析、在线模型等）在云平台上的混合部署场景，而没有考虑更多的复杂的服务混合场景（如复杂的OLTP数据库业务），因此在其他的云业务场景下可能还有迭代的过程。希望在下一阶段引入更多的负载场景使得调度算法更加通用化。

如何将人工智能应用在调度系统中是另外一个有价值的研究方向，调度系统能够监控当前云平台的各种资源指标以及服务的性能指标，可以通过对历史调度数据引入AI分析能力，对各个服务进行画像（如其生命周期可以分为活跃时间段和非活跃时间段等，不同的时间段对应不同的调度属性），从而让调度器可以更好地均衡调度所有的服务；预测节点以及平台在某个时间段的负载分布，并基于这些数据进行决策，如提供服务的时分复用调度策略，从而进一步提高云平台的资源使用率，降低企业的IT成本。

参考文献:

[1] 周涛，潘柱廷，程学旗. CCF大专委2019年大数据发展趋势预测[J]. 大数据，2019,5(1):109-115.
ZHOU T，PAN Z T，CHENG X Q Developing tendency prediction of big

data in 2019 from CCF TFBD[J]. Big Data Research, 2019, 5(1): 109-115.

[2] BURNS B, GRANT B, OPPENHEIMER D, et al. Borg, omega, and Kubernetes[J]. Communications of the ACM, 2016, 59(5): 50-57.

[3] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing[C]// The 9th Usenix Conference on Networked Systems Design and Implementation, April 25-27, 2012, San Jose, USA. Berkeley: USENIX Association, 2012.

[4] BREWER E A. Kubernetes and the path to cloud native[C]// The 6th ACM Symposium on Cloud Computing, August 27-29, 2015, Kohala Coast, USA. New York: ACM Press, 2015: 167.

[5] VERMA A, PEDROSA L, KORUPOLU M, et al. Large-scale cluster management at Google with Borg[C]// The 10th European Conference on Computer Systems, April 21-24, 2015, Bordeaux, France. New York: ACM Press, 2015.

[6] HINDMAN B, KONWINSKI A, ZAHARIA M, et al. Mesos: a platform for fine-grained resource sharing in the data center[C]// The 8th USENIX Conference on Networked Systems Design and Implementation, March 30-April 1, 2011, Boston, USA. Berkeley: USENIX Association, 2013: 295-308.

[7] SCHWARZKOPF M, KONWINSKI A, ABD-EL-MALEK M, et al. Omega: flexible, scalable schedulers for large compute clusters[C]// The 8th ACM European Conference on Computer Systems, April 15 - 17, 2013, Prague, Czech Republic. New York: ACM Press, 2013: 351-364.

[8] GHEMAWAT S, GOBIOFF H, LEUNG S T. The Google file system[J]. ACM SIGOPS Operating Systems Principles, 2003, 37(5): 29-43.

[9] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: a distributed storage system for structured data[J]. ACM Transactions on Computer System, 2008, 26(2): 1-26.

[10] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[C]// The 6th Conference on Symposium on Operating Systems Design & Implementation, December 6-8, 2004, San Francisco, USA. Berkeley: USENIX Association, 2004: 10.

[11] LESLIE L. The part-time parliament[J]. ACM Transactions on Computer Systems, 1998: 133-169.

[12] YANG F, TSCHETTER E, MERLINO G, et al. Druid: a real-time analytical data store[C]// The 2014 ACM SIGMOD International Conference on Management of Data, June 22-27, 2014, Snowbird, USA. New York: ACM Press, 2014: 157-168.

[13] PARIS C, STEPHAN E, SEIF H. Apache Flink: stream and batch processing in a single engine[J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015, 36(4): 28-38.

[14] ZAHARIA M, DAS T, LI H Y, et al. Discretized streams: fault-tolerant streaming computation at scale[C]// The 24th ACM SOSP Symposium on Operating Systems Principles, November 3-6, 2013, Farmington, USA. New York: ACM Press, 2013: 423-428.

[15] MARTIN A, PAUL B, CHEN J M, et al. TensorFlow: a system for large-scale machine learning[C]// The 12th USENIX Symposium on Operating Systems Design and Implementation, November 2-4, 2016, Savannah, USA. Berkeley: USENIX Association, 2016: 265-283.

[16] GANDINI A, GRIBAUDO M, KNOTTENBELT W J, et al. Performance evaluation of NoSQL databases[M]. Heidelberg: Springer, 2014.

[17] NAMBIAR R O, POESS M. The making of TPC-DS[C]// The 32nd International Conference on Very Large Data Bases, September 12-15, 2006, Seoul, Korea.

[S.l.: s.n.], 2006: 1049-1058.
[18] ZAHARIA M, BORTHAKUR D, SARMA J S, et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling[C]// The 5th European Conference on Computer Systems, April 13-16, 2010, Paris, France. New York: ACM Press, 2010: 265-278.

作者简介

刘汪根(1985-),男,星环信息科技(上海)有限公司研发总监、总架构师,主要研究方向为新一代的大数据架构、分布式数据库技术和容器云等。

郑淮城(1987-),男,星环信息科技(上海)有限公司软件工程师,星环原生云操作系统研发负责人,主要研究方向为复杂业务场景的底层容器云技术工程化。

荣国平(1977-),男,博士,南京大学软件学院副研究员,主要研究方向为DevOps、微服务架构、虚拟化技术等。

收稿日期:2019-08-10