CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 43 卷第 7 期 2021 年 7 月
Vol. 43, No. 7, Jul. 2021

文章编号:1007-130X(2021)07-1173-12

面向容器环境的 Flink 的任务调度优化研究*

黄　山[1,2,3],房六一[1,2,3],徐浩桐[1,2,3],段晓东[1,2,3]

(1.大连民族大学计算机科学与工程学院,辽宁 大连 116600;2.大数据应用技术国家民委重点实验室,辽宁 大连 116600;
3.大连市民族文化数字技术重点实验室,辽宁 大连 116600)

摘　要:随着互联网技术的飞速发展,人类正在走向大数据时代与云计算时代。Flink 作为最新一代的大数据计算引擎,具有低延迟、高吞吐等优势,受到学术界与工业界的青睐。Flink 在云环境下部署时,其默认任务调度由于无法获取容器部署分布信息,会导致负载分配不均衡。针对这一问题,提出一种面向容器环境的 Flink 任务调度算法 FSACE,获取每个结点性能信息与容器在结点上的分布信息,优先选择空闲资源较多的结点的容器,同时可以避免容器被频繁选中造成负载不均。使用云主机与合成数据集对算法进行评测,评测结果表明,在容器环境下部署时,所提出的算法能更均衡地分配任务,可以提高资源使用率和计算速度。

关键词:容器;大数据;Flink;任务调度;负载均衡;容器环境

中图分类号:TP301　　**文献标志码**:A

doi:10.3969/j.issn.1007-130X.2021.07.005

Task scheduling optimization of Flink in container environment

HUANG Shan[1,2,3], FANG Liu-yi[1,2,3], XU Hao-tong[1,2,3], DUAN Xiao-dong[1,2,3]

(1. College of Computer Science and Technology, Dalian Minzu University, Dalian 116600;
2. State Ethnic Affairs Commission Key Laboratory of Big Data Applied Technology, Dalian 116600;
3. Dalian Key Laboratory of Digital Technology for National Culture, Dalian 116600, China)

Abstract: With the rapid development of Internet technology, human beings are moving towards the era of big data and cloud computing. As the latest generation of big data computing engine, Flink is favored by academia and industry for its advantages such as low latency and high throughput. When Flink is deployed in the cloud environment, its default task scheduling will lead to uneven load distribution due to the inability to obtain container deployment distribution information. To solve this problem, this paper proposes a Flink task scheduling load balancing algorithm in container environment to obtain the performance information of each node and the distribution information of the container on the node, give priority to the container of nodes with more free resources, and avoid the uneven load caused by the frequent selection of containers. The evaluation results show that the proposed algorithm can more evenly allocate tasks and improve resource utilization and computing speed when deployed in container environment

Key words: container; big data; Flink; task scheduling; load balancing; container environment

* **收稿日期**:2021-02-04;**修回日期**:2021-04-12
基金项目:国家重点研发计划(2018YFB1004402)
通信作者:段晓东(duanxd@dlnu.edu.cn)
通信地址:116600 辽宁省大连市大连民族大学计算机科学与工程学院
Address: College of Computer Science and Technology, Dalian Minzu University, Dalian 116600, Liaoning, P. R. China

1 引言

随着互联网技术的飞速发展,万物互联成为未来发展的必然趋势,人类正从计算机时代走向数据技术时代。据国际数据公司IDC(International Data Corporation)预测,2025年全球产生的数据量将达到163 ZB[1]。与此同时,具有高可伸缩性、部署方便、初期成本低、按使用付费、运行维护成本低特点的云计算技术也得到了迅猛发展。大数据管理系统正在向支持可伸缩性和支持批计算、流计算、机器学习等多计算模式相融合方向发展[2]。

Flink[3]作为最新一代的大数据计算引擎,具有低延迟、高吞吐等优势,使其备受学术界和工业界青睐。与此同时,随着虚拟化技术的发展,各种应用都在使用容器化部署。然而,Flink在容器中部署时,由于其不能感知到结点负载情况,会导致任务分配不均匀,造成性能下降。

本文提出一种面向容器环境的Flink任务调度算法FSACE(Flink Shedule Algorithm for Container Environment),该算法在容器环境下部署Flink任务时,结合集群容器负载信息调度任务,从而使任务分配更加均衡,提升了Flink在容器环境下的处理效率。

本文的主要贡献有:

(1)提出一种结合容器部署负载信息的Flink任务调度算法。该算法获取每个结点性能信息与容器在结点上的分布信息,优先选择空闲资源较多的结点的容器,同时可以避免容器被频繁选中造成负载不均。

(2)利用不同规模的数据集进行对比实验,实验表明,本文提出的FSACE算法相比默认的Flink任务调度算法在计算时间、吞吐量、时延和资源利用方面都有优化。

2 相关工作

容器环境下大数据计算平台的任务调度优化算法主要分为以下2类:

(1)优化大数据处理引擎的任务调度算法。文献[4]提出了处理空间数据的批流融合处理架构,可以支持多源空间连接查询。文献[5]研究了批流融合系统中多算子放置问题,使算子能更均衡地分布到集群各结点上,提升效率。文献[6]提出了适用于Nephel[7]数据流平台的响应式资源调度策略,通过建立数学模型计算每个算子的并行度,并通过任务迁移实现集群资源的动态伸缩,但是在其任务迁移过程中,网络传输开销较大。文献[8]通过监控集群性能指标,建立了针对Strom[9]平台无状态数据流的弹性资源调度策略。文献[10]提出分布式弹性资源管理协议,实现了集群规模对输入负载的快速响应。文献[11]通过实现上下游结点算子的灵活迁移和动态链接,应对内存不足造成的背压[12]问题。文献[13]提出了自定义代价模型,在邻近代价阈值[14]时启动分区映射算法[15],实现结点间计算负载的最优分配。文献[16]将流式计算拓扑定义为流网络模型[17]并从中寻找优化路径,从而提高集群吞吐量。此外,文献[18-21]也分别提出了不同的负载均衡策略,以提升集群性能。这些算法由于没有考虑容器部署情况对于Flink资源调度的影响,造成任务分配不均匀,进而拖慢任务执行速度。

(2)基于结点性能信息的容器优化调度算法。文献[22]提出了基于负载预测的扩容策略和综合考虑状态因素、资源因素的优化策略,可以减少应用请求响应时间,减少结点资源碎片化,提升集群服务质量。文献[23]提出一种动态缩容算法,在缩容过程中根据某一服务在不同结点上分布的Pod(Kubernetes创建或部署的最小基本单位)[24]实际资源使用情况,计算出该结点删除Pod后的CPU内存资源均衡度,最后选择删除资源均衡度最小的结点,可以使集群具有更好的资源均衡度。这些研究工作考虑了容器情况的影响,但调度的基本单位为容器,调度粒度较粗,无法根据Flink任务具体情况进行资源调度。

综上,现有研究工作一部分没有考虑容器情况对于Flink任务调度的影响,一部分无法适应Flink资源调度。针对这一问题,本文提出了面向容器的Flink任务调度算法FSACE,算法通过获取容器的性能信息,优化Flink的任务调度,有效地改善了容器环境下Flink负载不均的问题,提高了计算效率。

3 研究技术背景介绍

本节主要介绍FSACE算法相关的技术背景,主要包括3个方面,分别是Flink大数据计算平台、容器技术[25]和Flink容器环境下部署方式。

3.1 Flink大数据计算平台

Apache Flink是一个批流融合分布式处理框

架,其功能十分强大,不仅可以运行在包括 YARN[26]、Mesos[27]和 Kubernetes[28]在内的多种资源管理框架上,还支持在裸机集群上独立部署。Flink 可以扩展到数千核心的集群中,其数据可以达到 TB 级别且仍能保持高吞吐、低延迟特性。

3.1.1　Flink 编程模型

Flink 的组件分为 4 层,各个模块之间的层次关系如图 1 所示。

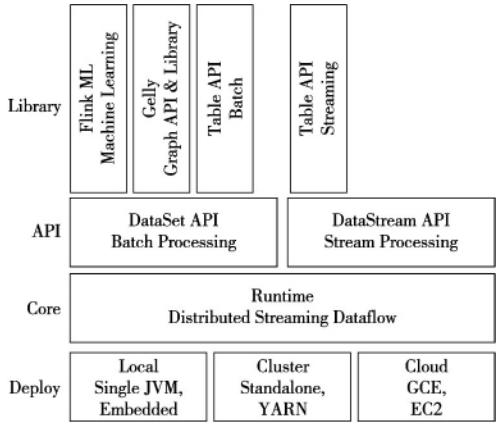

Figure 1　Flink programming model

图 1　Flink 编程模型

图 1 中底层是 Deploy 层,Flink 支持多种部署方式,例如本地(Local)单机版、Standalone 集群部署、YARN 集群部署和 Kubernetes 等云(Cloud)部署方式。

第 2 层是 Core 层,这一层是 Flink 分布式数据处理的核心实现层,包括计算图的所有底层实现。

第 3 层是 API 层,该层包括了流处理(DataStream)API 和批处理(DataSet)API,Flink 的批处理是建立在流处理架构之上的,因此 Flink 更适合流处理场景。

最上层是 Library 层,本层是 Flink 应用架构层,构建在 DataStream API 和 DataSet API 之上。

3.1.2　Flink 算子转换

Flink 中的执行图可以分成 4 层:StreamGraph、JobGraph、ExecutionGraph 和 PhysicalGraph。这 4 层执行图表示了从程序最初的拓扑结构到可执行 Task 的变化过程。

(1)StreamGraph:是根据用户通过 Stream API 编写的代码生成的最初的图,用来表示程序的拓扑结构。

(2)JobGraph:StreamGraph 经过优化后生成了 JobGraph,为提交给 JobManager 的数据结构。其中主要的优化为,将多个符合条件的结点链在一起作为一个结点,这样可以减少数据在结点之间流动所需要的序列化、反序列化和传输消耗。

(3)ExecutionGraph:JobManager 根据 JobGraph 生成 ExecutionGraph。ExecutionGraph 是 JobGraph 的并行化版本,是调度层最为核心的数据结构。

(4)PhysicalGraph:JobManager 根据 ExecutionGraph对 Job 进行调度后,在各个 TaskManager上部署 Task 后形成的"图",是各个 Task 分布在不同的结点上所形成的物理上的关系表示,并不是一个具体的数据结构。

以并行度为 2(其中 Source 并行度为 1)的 SocketTextStreamWordCount 为例,4 层执行图的演变过程如图 2 所示。

3.1.3　Flink 任务调度

Flink 集群启动后,首先会启动一个 JobManager 和多个 TaskManager。用户的代码会由 JobClient 提交给 JobManager,JobManager 再把来自不同用户的任务发给不同的 TaskManager 执行,每个 TaskManager 管理着多个 Task,Task 是执行计算的最小单元,TaskManager 将心跳和统计信息汇报给 JobManager。TaskManager 之间以流的形式进行数据传输。

上述除了 Task 外的三者均为独立的 JVM(Java Virtual Machine)进程。其中,TaskManager 和 Job 并非一一对应的关系,每个 TaskManager 最少持有 1 个 slot,slot 是 Flink 执行 Job 时的最小资源分配单位,在 slot 中运行着具体的 Task 任务。

Flink 提供了 2 种基本的任务调度逻辑,即 Eager 调度和 Lazy From Source。Eager 调度会在作业启动时申请资源将所有的 Task 调度运行,更加适用于流作业;而 Lazy From Source 则是按拓扑顺序来进行调度,更加适用于批作业。2 种任务调度均是从各 TaskManager 中的 slot 池中依次获取可用的 slot 进行资源分配。

3.2　容器技术

容器技术是一种可以有效地将单个操作系统的资源划分到独立的组中,以便更好地在独立的组之间平衡有冲突的资源使用需求的技术。

Docker[29]是基于 Go 语言的开源容器项目,是 C/S 架构,主要由客户端和服务器 2 大核心组件组成,通过镜像仓库来存储镜像。客户端和服务器可以运行在同一台机器中,也可以通过 Socket 或 RESTful API 来进行通信。

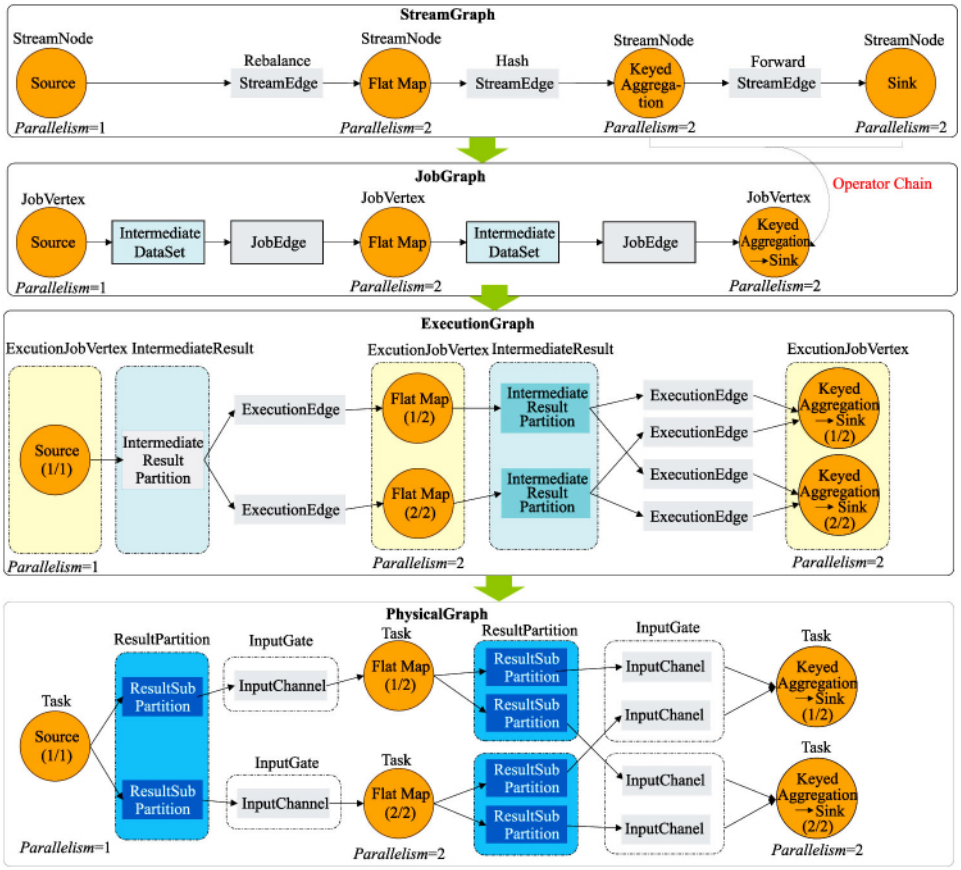

Figure 2 Flink operator conversion

图 2 Flink 算子转换

Kubernetes 用于容器编排，具有自动化程度高的优势。Kubernetes 可以将各种服务器资源整合到一起，应用的整个生命周期都可以实现自动化管理，用户不用关心具体容器如何部署，并且可以获取容器的相关信息。

3.3 Flink 容器环境下部署方式

将 Flink 在容器环境下部署的架构如图 3 所示。

Flink 的容器组件主要包括 JobManager 与 TaskManager，其中 JobManager 容器主要负责任务调度，而 TaskManager 容器主要负责执行计算任务。

为了监控容器资源性能部署了开源的 Metric-Server。在 Kubernetes 上部署 Metric-Server 之后，它在每个结点上都会有一个副本，负责收集容器性能信息，并实现 Flink 应用整个生命周期的自

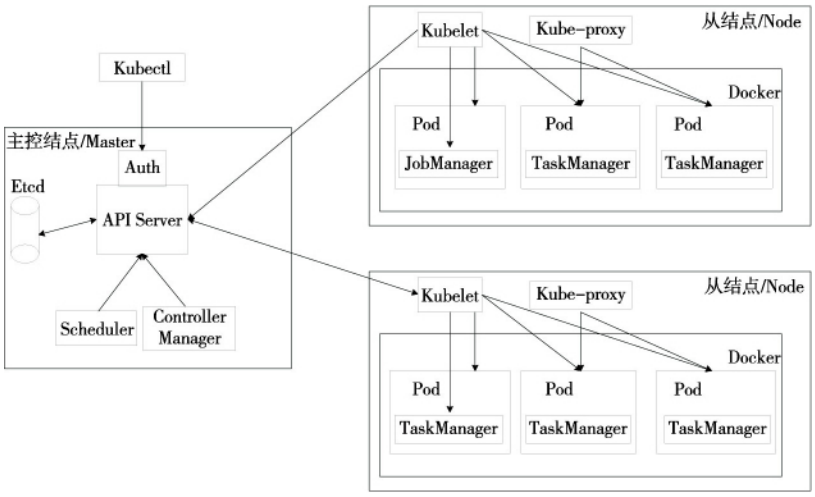

Figure 3 Deployment of Flink in a container environment

图 3 Flink 容器环境下部署

动化管理。

在 Kubernetes 主结点上的组件主要包括 Controller Manager、Scheduler 和 Etcd。其中 Controller Manager 可以用来控制 Flink 容器副本的数量,Scheduler 负责应用的调度,Etcd 键值数据库负责持久化存储集群配置。从结点上的组件主要 Kubelet 和 Kube-proxy,负责与 API 进行通信。

4　任务调度优化算法

本节首先分析 Flink 容器环境下部署时默认的任务调度算法存在的问题,之后介绍本文提出的 Flink 任务调度优化算法(FSACE)。

4.1　问题分析

在容器环境下部署 Flink 大数据处理平台时,由于每个结点上的容器数量不完全相同,容器也并不是都用于 Flink 部署,因此使用 Flink 默认的任务调度算法,会将每个容器作为一个分配单元分配,这将导致各个结点上任务负载不均。

Flink 默认任务调度算法、Kubernetes 容器调度算法和本文所提算法的任务调度实例如图 4 所示。

Flink 集群中有 3 个结点分别编号为 A、B、C,这 3 个结点在性能上存在差别,结点 C 的性能最好,结点 B 次之,结点 A 最差。每个结点上运行着不同的容器,占用着大小不同的资源,结点 A 容器资源占用最多,结点 B 次之,结点 C 容器资源占用最少。

集群中结点 C 的性能明显优于结点 A 和结点 B 的性能。假设此时集群中需要处理含有 7 个任务的任务集合。

按照 Flink 默认的任务调度算法,结点 A 处理任务 1,4,7,结点 B 需要处理任务 2,5,结点 C 处理任务 3,6。性能最差的结点 A 被分配了最多的任务,结点 B 和结点 C 分配了相等的任务。这样的任务分配结果就会造成负载不均衡。由此导致结点 A 处理时间拖慢了全部任务完成的时间,集群吞吐量下降,延时增加。

在 Kubernetes 容器集群情况下,按照其默认的容器调度算法,此时集群中,结点 A 处理任务 1,4,结点 B 需要处理任务 2,5,7,结点 C 处理任务 3,6。性能最差的结点 A 和性能最好的结点 C 分配了相同数量的任务,结点 B 分配了最多的任务,也造成了结点负载不均,从而增加了整个 Job 的运行时间。

按照本文提出的 FSACE 算法,结点 A 性能最差,处理任务 7,结点 B 性能好于结点 A 的性能,处理任务 5,6,结点 C 性能最好,处理任务 1,2,3,4。此时,性能好的结点处理更多的任务,性能相对

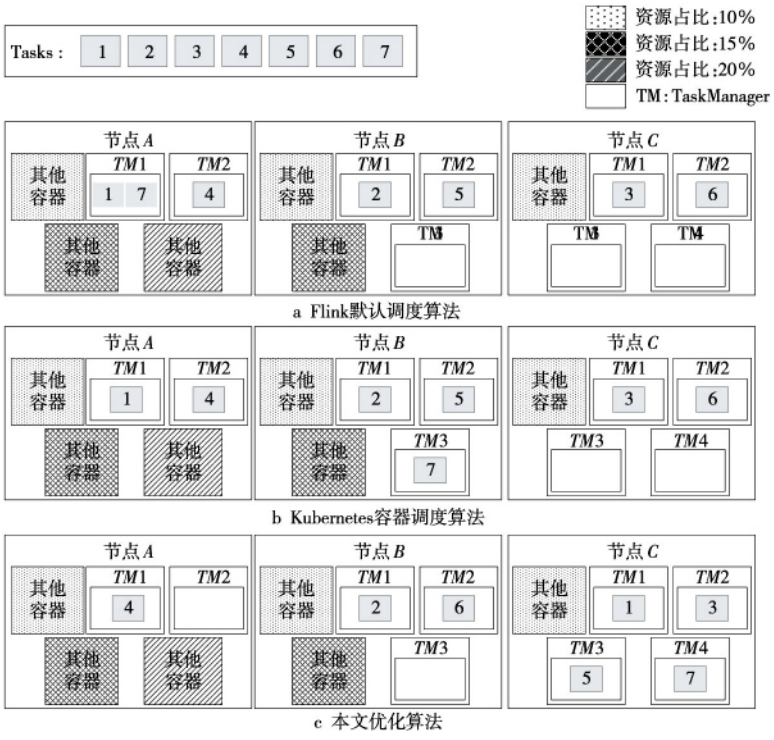

Figure 4　Flink containerized cluster processing task set

图 4　Flink 容器化集群处理任务集合

差的结点处理更少的任务,集群中任务分配相对均衡,从而减少了任务的运行时间。

在容器化的 Flink 集群中,容器部署不均会造成结点间存在很大的性能差别,如果采用 Flink 默认的轮询策略,会造成任务在结点上分布不均衡,进而影响集群计算效率。

4.2 FSACE 算法思想

FSACE 算法基本思想是,Flink 在容器环境部署时,为使任务分配更为均衡,需要获取结点性能与容器在结点上的分布情况,然后根据结点性能与容器在结点上的分布情况调度任务,每次选择评分最高的容器分配任务,并调整各结点的评分。在调度任务时考虑以下 3 个问题:

(1)结点上容器资源分配不均问题。拥有较多空闲资源的结点不应被频繁分配任务,需要根据容器的分布计算结点的性能,给结点评分,并优先将计算任务分配给评分高的结点所在的 Flink 容器。

(2)结点性能上限问题。为了避免评分较大的结点被连续选中,导致评分较大的结点很快达到它的性能上限,最终该结点过负载。FSACE 算法会在选中结点时,适当降低结点评分,这样使性能较高的结点不会连续被选中,而是相对错开被选中,从而使任务分配更均衡。

(3)分布在同一结点的各容器资源会相互抢占资源问题。在一个容器中分配任务时,该结点占用的资源增加,为使任务调度更加均衡,应适当减少在该结点上的其他容器中分配任务,因此该结点其他容器分配任务的优先级应当下降。

例如,容器环境下 Flink 集群中 3 个结点 A,B,C,它们的权值比值为 3∶1∶2,因此当有 6 个任务需要被调度时,前 3 个任务分配给结点 A,接下 2 个任务分配给结点 C,最后一个任务分配给结点 B。这样的任务调度策略就能充分发挥结点 A 的性能优势,同时规避了结点 C 的性能劣势。因此 FSACE 算法能够充分利用集群各个结点的计算能力。

4.3 技术架构

FSACE 的技术架构如图 5 所示,其中架构的最底层为容器(Container),以 Docker 作为容器运行引擎,HDFS(Hadoop Distributed File System)作为分布式存储系统。

使用 Metric-Server 监控结点上的容器性能信息,Flannel 作为容器网络插件,为每一个容器分配独立的 IP。

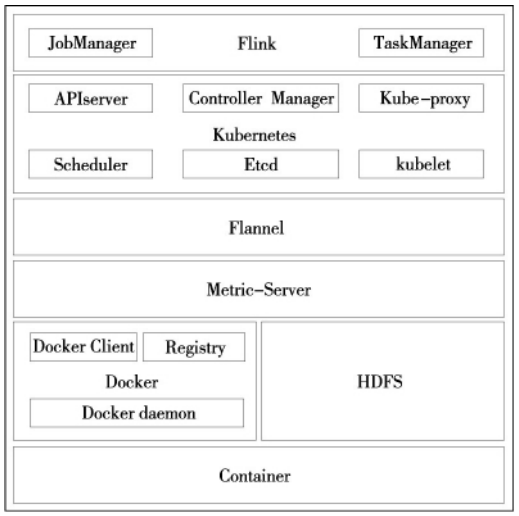

Figure 5 Technical architecture

图 5 技术架构

使用 Kubernetes 作为容器集群管理工具。最上层为 Flink 大数据处理平台,主要包括 JobManager 和 TaskManager 2 种容器。

FSACE 的调度流程如图 6 所示。FSACE 算法首先会通过 Metric-Server 获取各个结点上的容器信息,以及每个结点的性能信息,之后将这些需要的信息发送给 Flink。

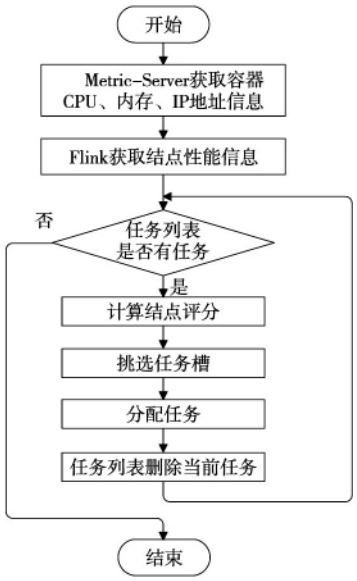

Figure 6 FSACE scheduling process

图 6 FSACE 调度流程

接下来判断任务调度列表中是否存在未调度的任务,如果没有还未调度的任务,则流程结束;如果有任务,那么根据结点性能进行结点评分,然后挑选任务槽,分配任务,将任务优先分配给评分高的结点所在的任务槽,并且避免评分过高的结点被频繁选中导致负载过高的问题,之后在任务调度列表中将已经分配的任务删除。

4.4　FSACE 算法

在容器化 Flink 集群中解决负载不均衡问题,就需要采用优化的结点性能评分任务调度算法,而这种算法需要得到集群中每个结点的性能评分。通常用计算资源来代表结点的评分,而影响 Flink 集群运行效率最主要的就是计算资源。计算资源中最具代表性的是 CPU 可用资源和内存可用资源,因此本文使用 CPU 可用资源和内存可用资源来给结点评分。

初始调度时根据 CPU 可用资源和内存可用资源指标得到一个结点初始评分 P。使用初始评分 P 来衡量该结点的性能高低,初始评分的计算如式(1)所示:

$$P = \delta C + (1-\delta)M \tag{1}$$

其中,C 是 CPU 可用资源,M 是内存可用资源,δ 代表 CPU 可用资源在整个初始评分中所占的比例,$1-\delta$ 代表内存可用资源在整个初始评分中所占的比例。通过实验发现,CPU 与内存的比例不同,会影响任务运行时间,经过多次实验发现,Flink 容器集群对 CPU 的资源敏感性大于对内存的资源敏感性,δ 为 0.9 时效果最好。

使用当前评分 Y 来衡量运行过程中每个结点的评分。当前评分等于上一次的评分加初始评分,第 1 次当前评分等于初始评分。每经过一次调度有效评分都会发生变化,使用当前评分来衡量结点的性能,每次挑选当前评分最高的结点作为目标结点。削减当前评分的计算如式(2)所示:

$$Y = \left(\sum_{i=1}^{n} Y_i + P\right) - T \tag{2}$$

其中,Y 是当前评分,n 是当前任务数,P 是结点 i 的初始评分,当 Y 是被选中的最高评分时,T 是每个结点的初始评分的和,若 Y 不是最高评分则 T 为 0。

算法 1 是结点性能评分算法,其中输入的 *HttpDatas* 结点性能信息是通过 Metric-Server 容器组件获取到的每个容器信息,经过叠加计算获得的。然后,遍历这个结点性能集合,根据结点的 CPU、内存信息计算每个结点的初始评分 P,Y 为结点的当前评分,初始时当前评分等于初始评分。然后,将计算好的每个结点性能评分加入到结点性能评分的集合 *DataSets* 中。

算法 1　结点性能评分算法

输入:Flink 的结点数据集合 *HttpDatas*。

输出:带评分的结点性能集合 *DataSets*。

① **for** *data* in *HttpDatas*
② $P=(data.cpudata+(1-\delta)data.memorydata$;
③ $Y=P$;
④ $d=$new *DataSet*(*ip*,*cpudata*,*memorydata*,*Y*,*P*);
⑤ *DataSets*.*add*(*d*);
⑥ **end for**
⑦ ***return*** *DataSets*

算法 2 是结点的控制评分算法,首先进行数据的初始化,把目标结点的 *instance* 变量初始化为空,把评分初始化为 0(第 1 行)。然后将每个结点的 *DataSet* 都与 *instance* 比较,如果 *DataSet*.*Y* 比 *instance*.*Y* 大,则用 *instance* 记录 *DataSet*,同时用 T 不断累加初始评分 P,每个 *DataSet* 的当前评分加上自身的初始评分 P(第 2～8 行)。结束循环后,把 *instance* 记录的当前评分最大的 *DataSet* 的有效评分减去总的评分 T,之后 *getAvailableSlot*()函数会顺序遍历这个结点上的 TaskManager 容器,并返回一个 TaskManager 容器中可用的 *Slot*(第 9 行和 10 行)。

算法 2　控制评分算法。

输入:结点的性能评分数据集合 *DataSets*。

输出:当前评分最大结点的任务槽 *Slot*。

① Initialize *instance*=null;$T=0$;
② **for** *DataSet* in *DataSets* **do**
③ **if** *DataSet*.*Y* > *intance*.*Y* **then**
④ *instance*=*DataSet*;
⑤ **end if**
⑥ $T=T+$*DataSet*.*P*;
⑦ *DataSet*.*Y*=*DataSet*.*Y*+*DataSet*.*P*;
⑧ **end for**
⑨ *instance*.*Y*=*instance*.*Y*−*T*
⑩ *slot*=*instance*.*getAvailableSlot*()
⑪ **return** *slot*

本文提出的 FSACE 算法考虑到每个结点的性能差异,根据结点实时性能赋予每个结点合适的评分,根据评分大小进行轮询调度,这样就能够使任务调度负载均衡。

为了防止轮询调度时评分较大的结点被连续多次选中造成评分较大的结点迅速达到负载上限,结点的计算效率降低,因此采用一种更加合理的处理方式,即让结点被错开选中。

4.5　算法示例

为了能更好地解释优化的 Flink 任务调度算法,本节给出一个任务调度算法的示例。

容器化 Flink 集群中有 3 个 Slave 结点,分别为 *Slave*1、*Slave*2、*Slave*3。假设初始的评分比例是 *Slave*1 : *Slave*2 : *Slave*3 = 3 : 1 : 2,因为初始

的时候结点的有效评分等于它自身的评分,因此有效评分比 $Slave1:Slave2:Slave3=3:1:2$。

分别采用 Flink 默认任务调度算法和 FSACE 算法进行阐述。假设有一个任务集合共有 6 个子任务。

(1)Flink 默认任务调度算法。该算法不需要考虑评分,因此其任务调度过程如表 1 所示。其中 $Slave1$ 被选中 2 次,$Slave2$ 被选中 2 次,$Slave3$ 被选中 2 次,这 3 个结点被选中的次数一样多,即 3 个结点轮流被选中。显然这种任务调度策略没有考虑到每个结点的性能,$Slave2$ 的性能最差,却被分配到和 $Slave1$、$Slave3$ 同等工作量的任务,因此 $Slave2$ 成为提升集群效率的瓶颈。

Table 1 Task scheduling process by Flink's default algorithm

表 1 Flink 默认的算法任务调度过程

任务	选中下标 *Index*	选择的容器所在结点
1	0	*Slave*1
2	1	*Slave*2
3	2	*Slave*3
4	3	*Slave*1
5	4	*Slave*2
6	5	*Slave*3

(2)FSACE 算法。FSACE 算法就是在任务调度的过程中,考虑每个结点 CPU 和内存的大小,对每个结点进行评分,优先挑选评分高的结点分配任务,并且避免某个评分高的结点连续被选中导致该结点达到负载上限,影响整个集群计算效率的问题。FSACE 算法调度过程如图 7 和表 2 所示。结点被选中的顺序分别是 $Slave1$、$Slave3$、$Slave1$、$Slave2$、$Slave1$ 和 $Slave3$。最终的顺序考虑了每个结点的性能,又避免了性能较好评分高的结点连续被选中导致负载不均的问题。

Table 2 Task scheduling process of FSACE

表 2 FSACE 任务调度过程

Step	*instance*.Y	Y_1	Y_2	Y_3	选中容器	选中的容器所在结点
1	6	6	2	4	*TM*1	*Slave*1
2	6	3	3	6	*TM*1	*Slave*3
3	6	6	4	2	*TM*2	*Slave*1
4	5	3	5	4	*TM*1	*Slave*2
5	6	6	−1	6	*TM*3	*Slave*1
6	8	3	0	8	*TM*2	*Slave*3
7	6	6	1	4	*TM*4	*Slave*1

综上所述,通过对 2 种任务调度算法过程的详细描述可以看出,FSACE 算法在考虑了结点性能的基础之上,进一步避免了性能较好的结点连续被选中,使得整个 Flink 容器集群变得负载均衡,提高了整个集群的计算效率。

4.6 复杂性分析

在时间上,FSACE 算法的时间复杂度为 $T(n)=O(m*n)$,其中 m 是容器数,n 是任务数。

在空间上,FSACE 算法中的性能评分算法需要存储容器部署信息,包含容器所在结点位置、容器 IP 地址、结点 CPU 使用率和结点内存使用率等信息,在算法中还需要用到中间变量及评分信息。每一条信息约几十字节,故算法空间复杂度为 $O(m)$,其中 m 为集群中的容器数。

5 实验

本节主要描述 Flink 默认任务调度算法与本文提出的 FSACE 算法的对比实验。通过 Docker 容器平台将 Flink 各个组件容器化,之后通过配置文件将 Flink 容器配置到 Kubernetes 集群中。在 Kubernetes 容器集群中,配置了 Metric-Server 容

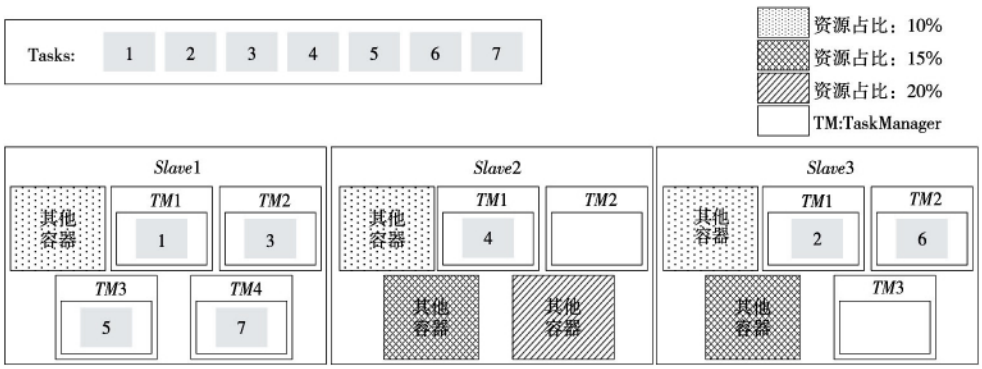

Figure 7 Task scheduling process of optimized algorithm

图 7 优化的算法任务调度过程

器监控组件来获取容器的性能信息。

5.1　实验环境说明

FSACE 算法使用 Java 作为编程语言实现,操作系统使用 CentOS,容器运行平台使用 Docker,容器管理系统使用 Kubernetes,容器性能监控组件使用 Metric-Server。各软件版本如表 3 所示。

实验硬件环境配置:本文共使用 4 台阿里云服务器组成集群,其中性能最好的结点作为 Master 结点。各结点硬件配置如下:

结点 1:通用型 ecs. g6,CPU:4 核,内存:8 GB,系统盘(SSD 云盘):200 GB;

结点 2:通用型 ecs. g6,CPU:2 核,内存:4 GB,系统盘(SSD 云盘):200 GB;

结点 3、4:通用型 ecs. g6,CPU:2 核,内存:2 GB,系统盘(SSD 云盘):200 GB。

Table 3　Software version

表 3　软件版本

软件	版本
CentOS	7.5
JDK	1.8
Docker	18.09.2
Kubernetes	1.9
Metric-Server	0.3.3
Flink	1.4.1
Hadoop	2.7.5

生产情况下任务负载,每个结点上由于运行的容器的数量和规模不同,使得每个结点的性能变得不同,实验中为了尽可能模拟生产情况下任务负载,除了 Flink 的容器之外,在结点上部署了一些用于其他服务的 Pod。每个结点的 Pod 以及资源配额数量如表 4 所示(集群单一结点 CPU 容量为 1 000m,CPU 分配单位为 m,代表千分之一 CPU)。

Table 4　Node resource quota

表 4　结点资源配额

结点	其他容器占用 CPU	其他容器占用内存/MB	Flink 容器
结点 1	100m	100	JobManager
结点 2	300m	200	TaskManager
结点 3	400m	500	TaskManager
结点 4	500m	800	TaskManager

本节主要进行了 CPU 评分影响运行时间的实验、Flink 默认任务调度算法、Kubernetes 容器调度算法与本文提出的 FSACE 算法的对比实验,实验数据集为 3 种不同规模数据集。为了减小误差,每次实验进行 3 次取平均值。

5.2　实验参数说明

本文使用了 WordCount 计算任务进行测试,使用的实验数据集共有 3 种规模,测试程序分别在这 3 种数据集、不同的容器数量、并行度情况下进行了多组实验,之后选取实验效果较好的参数设置进行了 TeraSort 计算任务的实验。

实验中使用 Flink 默认任务调度算法、Kubernetes 容器调度算法、FSACE 算法从时延、吞吐量和运行时间方面做了对比和分析。

实验中使用一个计算任务例子 WordCount,该程序主要是统计每个单词出现的次数,执行过程是 Source-FlatMap-GroupBy-Sum-Sink,其中 Source 是读取数据源,FlatMap 是将每一行语句按照空格拆开变成多个单词,GroupBy 则是将单词分组,相同的单词被数据重组(shuffle)到同一个结点,后续可以统计单词个数。Sum 统计每个单词出现的次数,Sink 把统计结果写入到 HDFS 中。

实验中另一个计算任务是 TeraSort,它是分布式计算平台中用于对数据进行排序 Benchmark,在不同平台上对数据排序的效率是衡量分布式系统处理能力的公认标准。整个实验中所用到的实验参数如表 5 所示。

Table 5　Experimental parameters

表 5　实验参数

参数	取值范围	默认值
数据文件大小	1 GB,2 GB,4 GB	1 GB
计算任务类型	WordCount、TeraSort	WordCount
容器数	4,6	4
并行度	6,8	6
值	0～1	0.9

5.3　实验结果与分析

图 8 展示了 δ 对于运行时间的影响。由实验结果可以看出,δ 会影响任务的运行时间,容器环境下 Flink 任务的执行时间对 CPU 的资源敏感性大于对内存的资源敏感性,并且 δ 为 0.9 时,运行时间最短,后面的实验均采用 $\delta=0.9$。

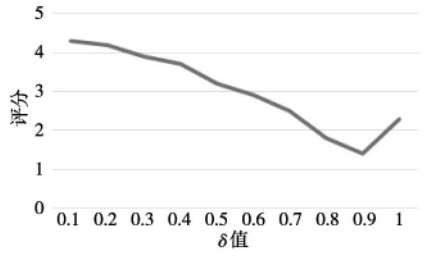

Figure 8　Running time v. s. δ

图 8　运行时间随 δ 值变化

在不同规模数据集上,算法任务运行时间评测结果如图 9 所示。FSACE 算法相对其他 2 种算法,Flink 任务的运行时间明显减少,并且随着数据集规模的增加,优化效果更加明显。实验结果显示,本文提出的算法由于结合了容器部署信息调度任务,使任务分布得更加均衡,从而提高了集群效率,减少了任务运行时间。本文算法也能够将新分配的计算任务尽可能多地分配到计算资源丰富的结点所在的 TaskManager 容器中,并且能够避免连续选择某一结点,这样使得计算资源丰富的结点能够负担更多的任务,而计算资源少的结点负担相应少的任务,因此使 Flink 容器集群负载均衡,从而缩短任务的运行时间。

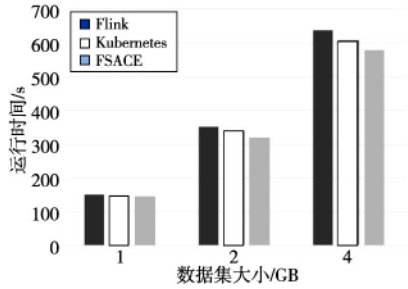

Figure 9 Comparison of task running time under different data set size

图 9 不同规模数据集上任务运行时间对比

在不同并行度下,算法任务运行时间评测结果如图 10 所示。并行度更高时,本文提出的 FSACE 算法相对其他 2 种算法的 Flink 任务的运行时间提升更加明显。实验结果显示本文提出的算法有较好的可扩展性。

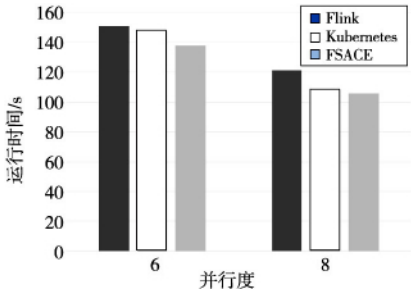

Figure 10 Comparison of task running time under different parallelism

图 10 不同并行度下任务运行时间对比

在不同容器数量下,算法任务运行时间评测结果如图 11 所示。实验结果显示,FSACE 算法相对其他 2 种算法 Flink 的运行时间提升相对明显。实验结果表明,本文算法更适于多容器环境下部署。FSACE 算法的运行时间比 Flink 默认的任务调度算法的运行时间少,平均运行时间少 8%,显示了本文算法在容器环境下部署 Flink 任务调度能有效提升任务运行效率。

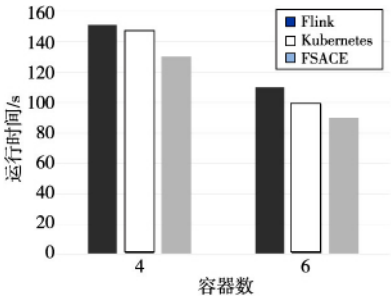

Figure 11 Comparison of task running time under different container numbers

图 11 不同容器数量下任务运行时间对比

在不同计算方面,算法任务运行时间评测结果如图 12 所示。实验结果显示,TeraSort 任务执行时间比 WordCount 的任务执行时间明显要少,这是由于 WordCount 计算过程相对更加复杂,因此需要的任务执行时间更多。实验结果还显示,FSACE 算法在 WordCount 计算任务上的优化效果比 TeraSort 计算任务更加明显,其原因是 TeraSort 计算任务结点间的负载较均衡。

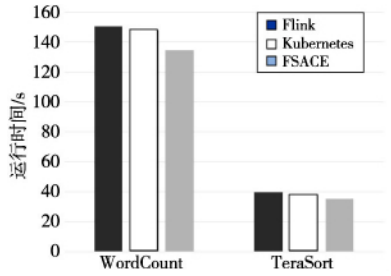

Figure 12 Running time of different computing tasks

图 12 不同计算的任务运行时间

算法调度后任务均衡性评测结果如图 13 所示。实验结果表明,使用 FSACE 算法,容器集群中不同结点的 CPU 和内存资源使用更加均衡。这是由于 FSACE 算法在任务调度时结合了容器部署信息,从而使任务更均衡地分布在结点上。

另外,本文对 FSACE 算法的任务调度时间与任务运行时间进行了评测,评测结果如图 14 所示。实验结果显示,在任务调度算法方面,由于 FSACE 算法需要获取容器部署信息,任务调度也比 Flink 默认算法复杂,因此任务调度稍慢于 Flink。但是由于任务执行时间占更主要部分,而 FSACE 算法能在容器环境下使任务分布得更加均衡,因此运行时间快于 Flink。

综上所述,在 Flink 容器集群中,在某些情况下,由于集群中运行着较多的提供不同服务的容器,导致每个结点的性能不一样,FSACE 使计算资源多的结点上的 TaskManager 容器被分配相对多

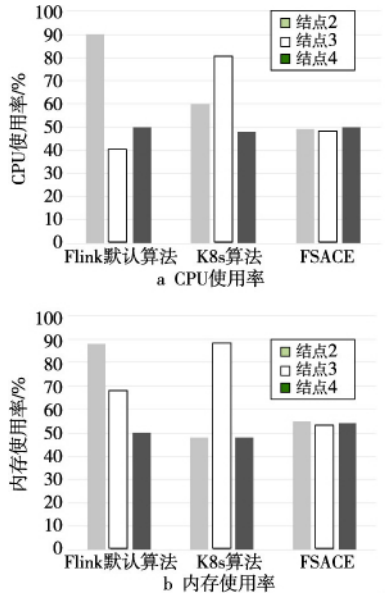

Figure 13　Load comparison of cluster nodes

图 13　集群结点负载对比

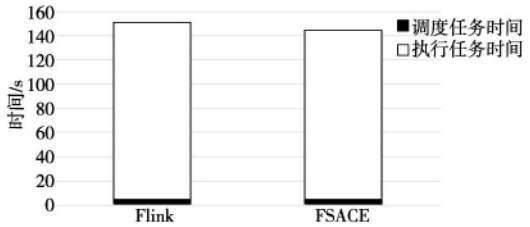

Figure 14　Task running time comparison

图 14　任务运行时间对比

的任务,而计算资源相对较少的结点上的 TaskManager 容器被分配相对较少的任务。这种任务分配策略能够实现负载均衡,缩短任务的运行时间,避免了默认的 Flink 任务调度算法中由于计算资源少的结点中的 TaskManager 容器完成和其他结点同等的任务而影响整体作业的完成进度,因此能够节省计算的时间,提高集群的计算效率。

6　结束语

本文首先介绍了容器环境下大数据处理引擎任务调度优化的研究背景与意义,以及大数据处理引擎任务调度优化方面的国内外研究现状。之后介绍了本文使用的相关技术,并仔细研究了 Flink 默认调度算法,分析了其在容器环境下任务调度容易发生负载不均衡的问题,提出了适用于容器环境下的 Flink 任务调度算法。实验结果表明,本文提出的 FSACE 算法,在容器集群中容器分布不均衡的情况下,能够减少作业运行时间,使得负载均衡,提高整个集群的计算效率。

在未来计划继续进行以下几方面的工作:

(1)使用更大规模的数据集验证实验效果;

(2)在进行任务调度优化的过程中考虑不同算子的特点进一步进行优化;

(3)根据不同的任务类型,调整 CPU 占用率和内存在评分中的权重,进一步优化评分算法。

参考文献:

[1] Data age 2025 [EB/OL]. [2017-12-11]. https://www.seagate.com/files/www-content/our-story/trends/files/seagate-WP-DataAge-2025-March-2017.pdf. (in Chinese)

[2] Du Xiao-yong, Lu Wei, Zhang Feng. History, present, and future of big data management systems[J]. Journal of Software, 2019, 30(1): 127-141. (in Chinese)

[3] Song Jin-quan. Research on optimization mechanism of containerized Spark resource scheduling in cloud environment [D]. Nanjing: Nanjing University of Posts and Telecommunications, 2019. (in Chinese)

[4] Li Zi-yang, Yu Jiong, Bian Chen, et al. Flow-network based auto rescale strategy for Flink [J]. Journal on Communications, 2019, 40(8): 85-101. (in Chinese)

[5] Luo Sheng-hao. Design and implementation of deep learning container cloud platform based on Docker and Kubernetes [D]. Beijing: Beijing Jiaotong University, 2019. (in Chinese)

[6] Ping Fan. Research on resource scheduler optimization strategy based on Kubernetes [D]. Xi'an: Xi'an University of Posts and Telecommunications, 2019. (in Chinese)

[7] Song Lin. Design and implementation of resource scheduling and monitoring system based on Kubernetes [D]. Beijing: Beijing University of Posts and Telecommunications, 2019. (in Chinese)

[8] Yu Chang-fa, Cheng Xue-lin, Yang Xiao-hu. Design and implementation of distributed TensorFlow platform based on Kubernetes [J]. Computer Science, 2018, 45(11A): 527-531. (in Chinese)

[9] Feng Xuan. Research on Hadoop performance optimization based on Docker technology [D]. Nanjing: Nanjing University of Posts and Telecommunications, 2018. (in Chinese)

[10] Dai Ming-zhu, Gao Song-feng. Framework performance evaluation based on Hadoop, Spark and Flink large-scale data analysis [J]. Journal of Chinese Academy of Electronics and Information Technology, 2018, 13(2): 149-155. (in Chinese)

[11] Li Cheng. Resource management system for big data cloud platform [D]. Beijing: China Academic of Electronics and Information Technology, 2018. (in Chinese)

[12] Chen Jin-guang. Design and implementation of Kubernetes container cloud platform based on Alicloud [D]. Hangzhou: Zhejiang University, 2018. (in Chinese)

[13] Kinnary J. Accelerating development velocity using docker [M]. Berkeley: Apress, 2018.

[14] Du Wei-ke. Design and implementation of Spark platform for big data streaming computing based on Kubernetes [D]. Nanjing: Nanjing University of Posts and Telecommunications, 2017. (in Chinese)

[15] Li Zi-yang, Yu Jiong, Bian Chen, et al. Dynamic data stream

load balancing strategy based on load awareness[J]. Journal of Computer Applications, 2017, 37(10): 2760-2766. (in Chinese)
[16] Zhang Li-zong, Cui Yuan, Luo Guang-chun, et al. Dynamic load balance algorithm for big-data distributed storage [J]. Computer Science, 2017, 44(5): 178-183. (in Chinese)
[17] Tang Rui. Research on resources scheduling strategy of container cloud platform based on Kubernetes [D]. Chengdu: University of Electronic Science and Technology of China, 2017. (in Chinese)
[18] Yang Peng-fei. Research and implementation of resource dynamic scheduling based on Kubernetes [D]. Hangzhou: Zhejiang University, 2017. (in Chinese)
[19] Xie B, Sun G Y, Ma G. Docker based overlay network performance evaluation in large scale streaming system[C]// Proc of 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference, 2016: 55-67.
[20] Zhang Yi. Design and implementation of virtualization application platform based on Docker [D]. Guangzhou: South China University of Technology, 2016. (in Chinese)
[21] Deepak V. Kubernetes Microservices with Docker using Apache Hadoop ecosystem[M]. Beijing: Apress, 2016.
[22] He Yu-jie. Load-balanced placement strategy for big data storage [D]. Shanghai: Shanghai Jiao Tong University, 2015. (in Chinese)
[23] Marathe N, Gandhi A, Shah J M. Docker swarm and kubernetes in cloud computing environment[C]// Proc of 2019 3rd International Conference on Trends in Electronics and Informatics(ICOEI), 2019: 23-65.
[24] Ferreira A P, Sinnott R. A performance evaluation of containers running on managed Kubernetes services[C]// Proc of 2019 IEEE International Conference on Cloud Computing Technology and Science(CloudCom), 2019: 125-139.
[25] Varma P C V, Venkata K C K, Kumari V V, et al. Analysis of a network IO bottleneck in big data environments based on Docker containers[J]. Big Data Research, 2016, 3: 24-28.
[26] Kang D K, Choi G B, Kim S H, et al. Workload-aware resource management for energy efficient heterogeneous Docker containers[C]// Proc of 2016 IEEE Region 10 Conference, 2016: 2428-2431.
[27] Dua A, Randive S, Agarwal A, et al. Efficient load balancing to serve heterogeneous requests in clustered systems using Kubernetes[C]// Proc of 2020 IEEE 17th Annual Consumer Communications & Networking Conference(CCNC), 2020: 1-2.
[28] Rensin D K. Kubernetes-scheduling the future at cloud scale[M]. CA: O'Reilly, 2015.
[29] He Z. Novel container cloud elastic scaling strategy based on Kubernetes[C]// Proc of 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference(ITOEC), 2020: 1400-1404.

附中文参考文献:

[1] 数据时代 2025 [EB/OL]. [2017-12-11]. https://www.seagate.com/files/www-content/our-story/trends/files/seagate-WP-DataAge-2025-March-2017.pdf.
[2] 杜小勇,卢卫,张峰. 大数据管理系统的历史、现状与未来[J]. 软件学报, 2019, 30(1): 127-141.
[3] 宋金全. 云环境下容器化 Spark 资源调度优化机制研究[D]. 南京:南京邮电大学, 2019.
[4] 李梓杨,于炯,卞琛,等. 基于流网络的 Flink 平台弹性资源调度策略[J]. 通信学报, 2019, 40(8): 85-101.
[5] 罗晟皓. 基于 Docker 和 Kubernetes 的深度学习容器云平台的设计与实现[D]. 北京:北京交通大学, 2019.
[6] 平凡. 基于 Kubernetes 的资源调度器优化策略研究[D]. 西安:西安邮电大学, 2019.
[7] 宋霖. 基于 Kubernetes 的资源调度与监控系统的设计与实现[D]. 北京:北京邮电大学, 2019.
[8] 余昌发,程学林,杨小虎. 基于 Kubernetes 的分布式 TensorFlow 平台的设计与实现[J]. 计算机科学, 2018, 45(11A): 527-531.
[9] 冯轩. 基于 Docker 技术的 Hadoop 性能优化研究[D]. 南京:南京邮电大学, 2018.
[10] 代明竹,高嵩峰. 基于 Hadoop、Spark 及 Flink 大规模数据分析的性能评价[J]. 中国电子科学研究院学报, 2018, 13(2): 149-155.
[11] 李程. 面向大数据云平台的资源管理系统[D]. 北京:中国电子科技集团公司电子科学研究院, 2018.
[12] 陈金光. 基于阿里云的 Kubernetes 容器云平台的设计与实现[D]. 杭州:浙江大学, 2018.
[14] 杜威科. 基于 Kubernetes 的大数据流式计算 Spark 平台设计与实现[D]. 南京:南京邮电大学, 2017.
[15] 李梓杨,于炯,卞琛,等. 基于负载感知的数据流动态负载均衡策略[J]. 计算机应用, 2017, 37(10): 2760-2766.
[16] 张栗粽,崔园,罗光春,等. 面向大数据分布式存储的动态负载均衡算法[J]. 计算机科学, 2017, 44(5): 178-183.
[17] 唐瑞. 基于 Kubernetes 的容器云平台资源调度策略研究[D]. 成都:电子科技大学, 2017.
[18] 杨鹏飞. 基于 Kubernetes 的资源动态调度的研究与实现[D]. 杭州:浙江大学, 2017.
[20] 张怡. 基于 Docker 的虚拟化应用平台设计与实现[D]. 广州:华南理工大学, 2016.
[22] 贺昱洁. 负载均衡的大数据分布存储方法研究与实现[D]. 上海:上海交通大学, 2015.

作者简介:

黄山(1986-),男,辽宁锦州人,博士,讲师,研究方向为大数据和云计算。E-mail: huangshan@dlnu.edu.cn

HAUNG Shan, born in 1986, PhD, lecturer, his research interests include big data, and cloud computing.

房六一(1996-),男,山东济南人,硕士生,研究方向为大数据和虚拟化。E-mail: 864006062@qq.com

FANG Liu-yi, born in 1996, MS candidate, his research interests include big data, and virtualization.