基于 SDN 的数据中心网络资源调度机制①

汪正康[1,2], 周 鹏[1], 肖俊超[1], 武延军[1]

[1](中国科学院 软件研究所, 北京 100190)
[2](中国科学院大学, 北京 100190)

摘 要: 随着大数据应用的不断丰富, 现在的数据中心通常部署着多种集群计算框架, 并由统一的集群资源管理器(如 Mesos)进行管理. 目前的集群资源管理主要关注计算资源和存储资源, 较少的涉及网络资源. 但研究表明高效的网络资源管理对于优化作业性能十分重要. 本文提出了一种基于 SDN(Software Defined Network)的数据中心网络资源调度机制, 该机制可以根据管理员预设的网络资源分配策略, 加权的进行网络资源调度, 为高优先级的作业分配更多网络资源以优化性能, 并且实现不同作业之间的网络性能隔离. 我们基于开源 SDN 控制器实现了原型系统, 并通过实验验证了该机制的有效性.

关键词: SDN; 数据中心; 网络资源调度

Network Resource Scheduling Mechanism Based on SDN in Data Center

WANG Zheng-Kang[1,2], ZHOU Peng[1], XIAO Jun-Chao[1], WU Yan-Jun[1]

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Science, Beijing 100190, China)

Abstract: With the emergence of Big Data applications in recent years, nowadays data centers are usually deployed with various kinds of Big Data applications and managed by resource manager such as Mesos. Mesos mainly focuses on computing resource and storage resource (CPU and memory), while ignoring network management. But research shows that efficient network resource management is crucial to optimize performance. In this work we present a network management mechanism based on SDN(Software Defined Network). This mechanism can allocate weighted network resource to different jobs based on preconfigured policy. It can allocate more bandwidth to high-priority job to optimize performance while keep the network performance isolation at the same time. We have implemented a prototype system based on SDN controller, floodlight and the experiments show that this prototype has strong usability and high expansibility.

Key words: SDN; data center; network resource management

1 引言

随着大数据时代的来临, 近年来涌现出了很多集群计算框架如 MapReduce[1]、 Spark[2]、Storm[3]等. 这些分布式计算框架定义了各自的计算模型, 通常数据集会经历一系列的处理阶段从而获得计算结果. 因为不同的计算场景在实时性、交互性、迭代性等方面各有差异, 所以没有一种分布式计算框架适用于所有计算场景. 每种框架擅长处理的计算模型不同: MapReduce 支持离线处理, Spark 擅长迭代式计算, 而 Storm 主要处理流式计算. 所以通常在一个数据中心中会运行多个计算框架的任务, 他们共享集群的计算、存储、网络资源. 为了满足大数据的计算需求, 集群通常会接入成百上千台机器, 因此为了降低成本, 高效的管理各种计算框架以提高集群资源利用率便成

① 基金项目:中国科学院知识创新工程重要方向项目(KGCX2-YW-174);中国科学院战略性科技先导专项课题(XDA06010600);国家自然科学基金(91318301,61432001)

收稿时间:2014-12-18;收到修改稿时间:2015-03-12

为急需解决的问题. 在这种背景下, 学者们提出了Mesos[4]、YARN[5]等集群资源管理框架, 用以高效的整合并管理集群资源. 但 Mesos , YARN 主要管理的是集群的计算和存储资源, 较少的涉及到网络资源.

但是高效的网络资源管理对于优化作业性能十分重要. 一份基于 Facebook 作业日志的研究表明, 平均来看, 网络资源传输占作业完成总时间的 33%, 对于很多作业来说甚至超过了 50%[6]. 所以针对于网络资源调度的研究也开始涌现了出来. 学者们主要从两个方向进行网络传输优化: 带宽分配优化和动态路由. 文献[6, 7]通过加权的分配带宽资源, 来实现网络资源调度的优化. 但是他们都是基于网络终端主机来进行调控, 当集群增加或者减少机器时都需要做相应的调整, 扩展性较差. 文献[8]通过预先计算数据流的带宽需求, 继而通过动态路由将该数据流调度到可以满足带宽需求的链路上从而避免网络拥塞. 这种方式可以有效的实现网络传输的负载均衡, 但引入的额外计算开销较高.

本文提出了一种基于 SDN 的数据中心网络资源调度机制. 该机制通过读取应用层的需求信息, 根据指定的作业权重为作业设置网络传输权重, 从而实现加权控制的网络资源共享. 本文的主要贡献: 1) 提出了一种数据中心网络资源调度机制. 目前在数据中心中, 我们可以为高优先级的作业分配更多的计算和存储资源, 以加快作业完成时间. 作为这类机制的补充, 本文提出的方法可以为高优先级的作业分配更多的网络资源, 加快作业网络传输, 从而优化作业整体完成时间. 2) 基于 SDN 设计了框架, 并实现了原型系统. 基于该框架的灵活性, 我们只需要在控制器端实现策略和算法, 直接下发到 SDN 交换机即可使策略生效. 相比于传统的逐个配置交换机和路由器, 或者在终端机器上进行网络资源调控, 我们的框架具有更好的便捷性和易扩展性.

2 相关研究

2.1 数据中心网络架构

在新的计算模型下, 传统数据中心网络在扩展性、灵活性、成本等方面都有着诸多不足. 近年来学者们提出了很多新型的数据中心网络架构, 旨在提高数据中心网络的扩展性, 健壮性, 可配置性和能耗效率. 很多网络架构[9-11]设计的主旨是为了达到完全对分带宽(Full Bisection Bandwidth). 在这些网络架构中, 网络拓扑通常是由多根树组成, 任意两个主机之间有多条等价路由, 数据包通过 ECMP(Equal Cost Multi-path Routing)协议进行转发. 但是, 完全对分带宽不意味着无限带宽, 本文提出的网络资源调度机制在这种网络架构中依然能有重要价值.

2.2 数据中心资源调度

如引言中提到, 目前数据中心利用统一的资源调度器进行集群资源管理. Mesos 采用了两层的调度机制: 首先由 Mesos 整合集群所有资源, 并决定向各个框架提供多少资源; 而每个框架决定接受多少资源并在这些资源上调度任务. YARN 是由 Hadoop 演进而来, 主要是为了解决原始 Hadoop 扩展性较差, 不支持多计算框架而提出的. 较于原始 Hadoop 其最大特点是将 JobTracker 拆分成 Resource Manager 和 Application Master, 其中 Resource Manager 是全局的资源管理器, 仅负责资源分配, 而 Application Master 管理一个具体的应用(如 Hadoop job, Spark Job 等), 它主要负责应用的资源申请, 各个任务的调度和运行状态监控等. 但这两种资源管理器主要调度的是计算和存储资源, 而本文关注的是网络资源的调度.

2.3 数据中心网络调度

如前文提到, 数据中心已有的资源调度框架 Mesos、YARN 并没有涉及网络资源共享, 而是依赖于底层 TCP 的拥塞控制. 但 TCP 本身有一定的缺陷, 它无法为应用程序之间提供有效的网络隔离, 并且对于恶意的网络流量没有很好的鉴别和控制机制. Seawall[6] 实现了基于用户指定权重的网络带宽分配, 可以加权的、动态的为数据通道分配网络带宽. Seawall 在所有发送端的主机上实现了具有拥塞控制能力的逻辑通道(一个符合 NDIS 标准的数据包过滤器), 所有数据包都需要从该通道进入或者发出, 通过控制逻辑通道的带宽以可以实现网络的共享. Orchestra[7]也提出了一种网络共享的方式, 不过它是通过改变 TCP 连接数来改变数据流的权重, 拥有更多 TCP 连接的数据流将获得更大的网络带宽, 从而实现网络的加权共享.

2.4 基于 SDN 的网络调度

SDN 技术的出现使得我们在网络配置、管理、监控方面有了更大的灵活性. 负载均衡和动态路由是 SDN 的一大优势. Hedera[8]动态的从交换机中获取网络状态, 为数据包计算出最优的、无碰撞的路径, 从而

优化数据中心网络传输，并实现了网络流量的负载均衡. FlowComb[12] 基于领域知识来优化大数据应用(如Hadoop)的网络传输，它通过检测 map\reduce 任务的完成情况来预测将要产生的网络流量，并动态进行流控以避免拥塞. 文献[13]通过实验表明，动态的为 hadoop shuffle 阶段提供更高带宽，可以有效减少 hadoop 作业的执行时间. 综合来看，基于 SDN 的网络调度主要有两种: 其一，利用 SDN 全局管控的能力来监控网络状态，并动态的调整路由以避免网络拥塞，实现传输流量的负载均衡; 其二，利用 SDN 动态的提供或改变带宽，以加速网络传输. 因为本文的目的是实现加权的网络资源调度，所以采取了第二种方式，基于文献[13]所提出的带宽分配方法，本文扩展并提出了新的网络资源调度模型.

3 理论基础

3.1 SDN

SDN 是一种新型的网络设计思路，较之于传统的网络交换方式，根本不同点在于其控制平面和转发平面不再紧耦合在一起. 在 SDN 网络中，交换设备的数据转发层和控制层是分离的: SDN 控制器决定数据包的转发策略，并下发到 SDN 交换机中以指导数据包的转发. 相比于传统网络，SDN 网络主要有以下几点优势: 1) SDN 控制器可以收集整个网络的状态信息，以全局最优的方式进行网络流量的调控和动态路由，从而实现网络流量的负载均衡; 2) SDN 以软件编程的方式进行网络管理，极大的简化了网络配置，具有更好的便捷性和灵活性. 现在很多云环境下的数据中心都是通过虚拟机对外提供服务，在这种应用场景下 SDN 能发挥更大的作用.

OpenFlow 是应用范围最广的 SDN 解决方案[14]. OpenFlow 由两部分组成(如图 1): 控制器和 OpenFlow 交换机. 在 OpenFlow 网络中，网络设备维护一个流表(FlowTable)并且按照流表进行转发. 流表本身的生成、维护、下发完全由控制器来实现.

OpenFlow 通过用户定义的或者预设的规则(流表项)来匹配和处理网络包. 一条 OpenFlow 的流表项由匹配域(Match Fields)、统计数据(Counters)和处理指令(Instructions)组成. 首先数据包进行流表项匹配，匹配域的字段包括 L2、L3 或者 L4 等网络报文头的任意字段，如以太网帧的源 MAC 地址, IP 包的协议类型和 IP 地址，或者 TCP/UDP 的端口号等. 当数据包成功匹配一条规则后，将首先更新该规则对应的统计数据，然后根据规则中的指令进行相应操作，比如跳转至后续某一流表继续处理. 当数据包已经处于最后一个流表时，其对应的 Action Set 中的所有 Action 将被执行，包括转发至某一端口，修改数据包某一字段，丢弃数据包等，从而完成对一个数据包的所有处理.

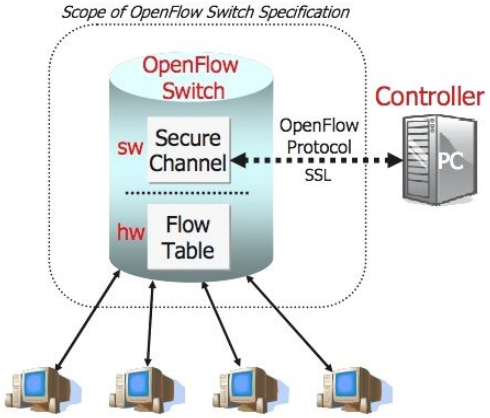

图 1 OpenFlow 架构图

OpenFlow 实现了 Qos 功能，它通过在每个端口建立 HTB 队列的方式提供带宽隔离. 当我们把一个数据流映射到特定队列时，该数据流就会遵从该队列的 Qos 设置，从而实现了数据流粒度的带宽分配. OpenFlow 的 Qos 功能为我们实现网络资源调度模型提供了便利.

4 网络资源调度模型

4.1 场景定义

为了更好的描述该模型的原理，我们首先通过例子来说明为什么该模型能优化数据中心作业性能. 我们假设现在数据中心中共有四种类型的网络数据流: Hadoop Job、Spark Job、System log、System backup. 四种数据流都需要通过一个交换机端口进行转发. 因为 Spark Job 进行的是实时计算，所以时效性要求较高. 而 System log\ System backup 作为例行维护行任务所以时效性没有要求. 在没有网络资源调度系统的情况下，网卡的带宽分配是以数据流的 TCP 连接数为单位的，如果此时 System backup 和 Hadoop Job 流量较大，则他们会获得较多的网络带宽资源，而高优先级的 Spark Job 却无法获得较多的带宽资源(图 2). 但在网络

资源调度模型中, 我们会为高优先级的 Spark Job 分配更多的带宽资源, 而限制 System log\System backup 任务的上限带宽(图 3), 从而加速高优先级任务的网络传输, 优化任务的整体完成时间.

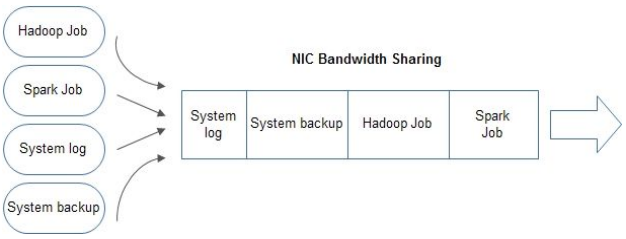

图 2 默认带宽分配

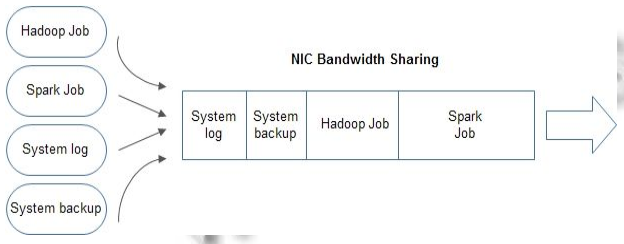

图 3 加权带宽分配

4.2 调度模型

为了实现网络资源调度模型, 我们需要加权的为数据流分配带宽. 但是为每个数据流分配固定的带宽却不是一个好的策略. 因为当该数据流较小, 无法占满所分配带宽时会造成资源的浪费; 反之如果数据流较大, 固定分配的带宽资源又可能无法满足其需求. 所以我们提出, 对于重要的任务, 我们为其设定带宽资源的下限(min-rate), 即它可占用的最少带宽, 并且在网卡带宽未占满的情况下它可用更多的带宽; 对于不重要的任务, 我们为其设定占用带宽资源的上限(max-rate), 以确保它不会影响其他任务传输, 从而实现网络传输层面的性能隔离. 而任务是“重要”还“非重要”, 将由管理员通过策略文件决定.

我们用 t_i 表示第 i 个作业(task), w_i 表示第 i 个 task 的权重, M 表示交换机端口的最大可用带宽, q_i 表示第 i 个 task 所在队列. 则该模型的思路是: 根据 w_i 为 q_i 分配相应权重的带宽资源, 若 t_i 为重要流量, 则设置 q_i 的下限带宽; 反之, 若 t_i 为非重要流量, 则设置 q_i 的上限带宽. 在权重计算完成以后, 我们创建对应于该数据流的 Qos 队列并将该数据流加入相应队列中. 该模型的核心算法的伪码表示:

算法 1

```
// 加载配置文件
load_config()
// 定义网卡的最大带宽为 100Mbps
M = 100
// 分别为每个 Task 设置 Qos
FOR t_i IN list(t_i)
    // 计算该 Task 应该分配的带宽资源
    m_i = (w_i * M) / ∑ w_i
    // 对于重要 Task, 设置其带宽资源的下限
    IF t_i is Important ; THEN
        q_i_min-rate = m_i
    ELSE
    // 对于非重要 Task, 设置其带宽资源的上限
        q_i_max-rate = m_i
    EndIF
    // 创建对应于该 Task 的 Qos 队列
    Create(q_i)
    // 将该 Task 数据流加入新创建的 Qos 队列中
    Push(t_i,q_i)
END
```

4.3 模型实现

本文提出了基于 SDN 的数据中心网络资源调度机制, 并实现了原型系统. 原型系统是基于开源 OpenFlow 控制器 Floodlight[15]扩展实现的. Floodlight 是由 Big Switch Networks 提出的, 具有良好兼容性及可扩展性的 OpenFlow 控制器. Floodlight 使用模块化的设计思想, 在启动时可以选择性的加载不同模块, 用户也可以选择扩展和加载自己的模块, 因此具有很好的扩展性. Floodlight 封装了 OpenFlow 协议的一些基本功能, 如拓扑发现、事件处理、与交换机通信等, 但更高级的功能需要用户自己扩展实现.

为了实现本文提出的模型, 我们需要扩展实现三个模块: configLoader、weightManager、qosPusher. 系统原理图如图 4 所示. 首先, 系统加载时 configLoader 模块会加载用户的配置文件. 配置文件由多个 task 组成, 每个 task 表示一种任务的数据流. 例如计算集群中有 spark, hadoop, backup 任务等(用户可以根据集群的具体情况进行设置). 每个 task 会有四个属性: Name 表示该数据流的名称; Port 表示该数据流在网络传输中的端口(为了让 OpenFlow 识别出该数据流); Weight 表示该数据流的权重, 系统会根据该权重为该数据流

分配相应的带宽资源，权重数值为 1~10，数值越大权重越高；Important 字段表示该数据流是否是重要的数据流，该字段会影响网络流量分配策略．最后，需要注意的是，因为集群中可能会有其他未被列出来的网络流，所以配置文件中有默认的一项 SystemReserve Task，用来表示系统中未被列出的数据流．

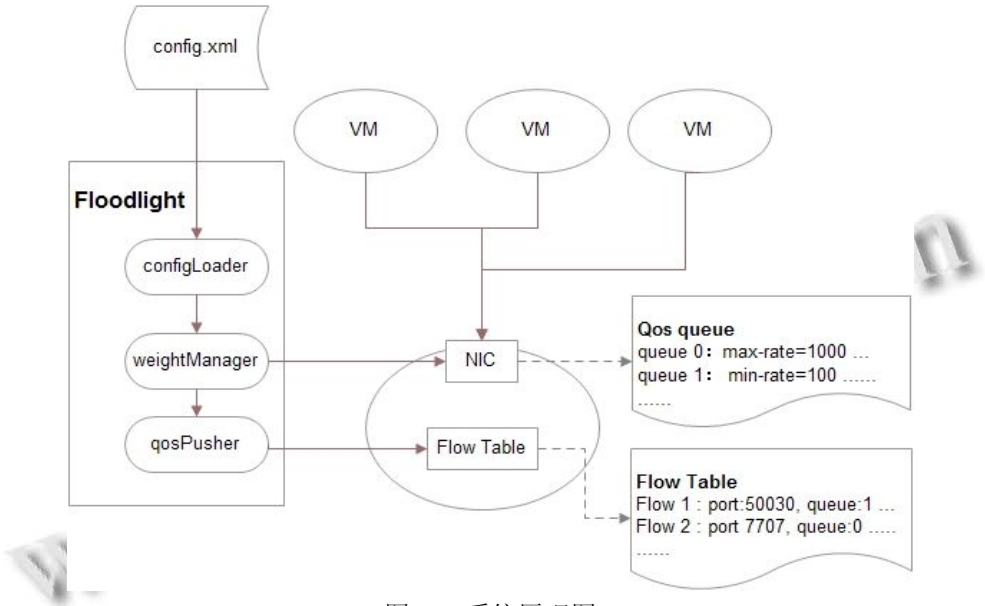

图 4 系统原理图

configLoader 模块完成配置加载以后，会通知 weightManager 模块．该模块主要负责根据刚刚加载的配置信息进行权值计算，计算完成之后在交换机上创建对应的 Qos 队列，并通知下一个模块．qosPusher 模块负责流表下发，将相应 Task 的数据流添加到相应队列中．这样后续当数据流到达 OpenFlow 交换机时，便会遵从相应队列的 Qos 配置，从而实现网络资源的加权调度．

5 系统验证

5.1 实验环境

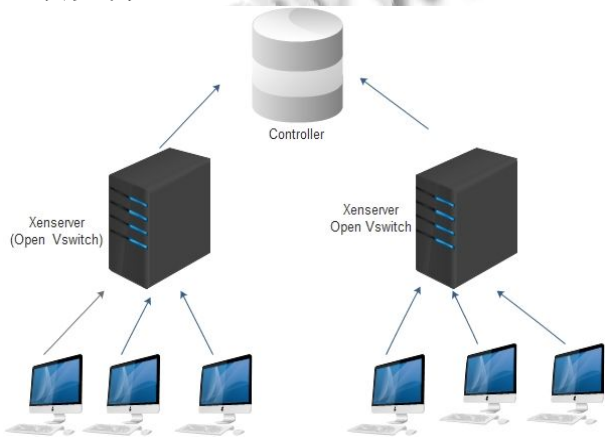

图 5 实验环境

为了验证系统的有效性，我们搭建了一个小的计算集群．我们在两台服务器上安装 Xenserver[16] 虚拟化平台，并在每台机器上创建三个虚拟机．Open vSwitch(OVS)[17]是支持 OpenFlow 协议的虚拟交换机，也是 Xenserver 默认的网络管理方式．因此 Xenserver 是最合适的实验平台．我们将虚拟机接入到 OVS 中，再将 OVS 接入扩展后的控制器 floodlight．接着，我们在这 6 个虚拟机中安装 hadoop2.2.0 以及 spark 1.0，使其成为共享的计算集群．实验环境如图 5 所示．

5.2 实验设计

在该计算集群中，除了拥有 hadoop 数据流和 spark 数据流，我们假设还有系统备份的数据流．我们用 Iperf 软件来模拟产生系统备份数据流．因此我们的配置文件中共有四个 Task: hadoop、spark、backup、SystemReserve，我们为相应的 Task 配置相应的端口和权重，并且，我们认为 hadoop 和 spark 是重要的任务，而 backup 是非重要的数据流．我们使用了 hadoop 自带的排序程序作为测试基准．Hadoop 排序程序分为三个阶段；产生随机数，执行排序和验证结果．我们每次生成 1G 的随机数并执行排序任务．对于 Spark 程序，我们使用其自带的 SparkKMeans 程序，输入数据是随机生成的 1G 的三维数据，通过 SparkKMeans 进行聚类分析．我们设计了四组

实验，第一组实验不打开网络资源调度系统，后三组实验打开调度系统，并设置不同的权重:

1) 不打开网络资源调度系统，默认情况下测试作业的完成时间;

2) BackupWeight=2, HadoopWeight=6, SparkWeight= 6;

3) BackupWeight=4, HadoopWeight=6, SparkWeight= 6;

4) BackupWeight=2, HadoopWeight=4, SparkWeight= 8;

我们在这四种情况下分别测试了 hadoop 和 spark 作业的完成时间.

5.3 实验结果

实验结果如图 6 所示. 从实验结果中我们可以看出，在实验一中，因为没有任何的网络资源调度，所以 backup 数据流抢占了大部分的带宽，此时 hadoop 任务和 spark 任务都是用时最长的. 而在实验二中，我们限制了 backup 数据流占用带宽，此时，hadoop 任务和 spark 任务都有了明显的加速. 在实验三中，我们提高了 backup 数据流权重，但其与任务的权重不变，可以看出，spark 任务和 hadoop 任务耗时又有所增加. 在实验四中，为了和实验二形成对比，我们固定 backup 数据流权重，但降低 hadoop 权重，同时增加 spark 权重，此时从结果可以观察到: hadoop 任务完成完成时间有了较明显的增加，而 spark 任务完成时间略有减少.

从以上实验我们可以看出，带宽资源的调控会有效的影响大数据系统的性能. 而本文实现的网络资源调度系统，可以根据应用层需求，动态的分配带宽权重，从而实现网络资源的加权调度以及不同任务之间的网络性能隔离.

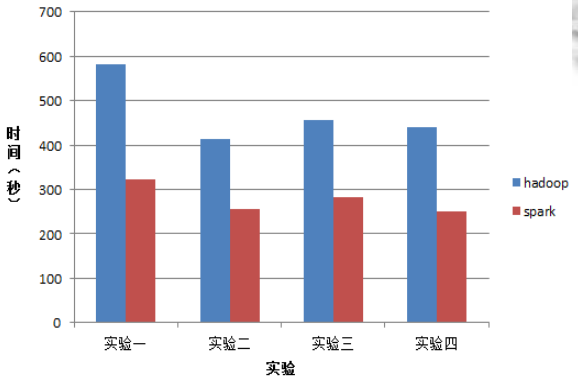

图 6 实验结果

6 结语

本文提出并实现了一种基于 SDN 的数据中心网络资源调度机制. 我们首先论述了高效的网络资源调度对于优化数据中心作业性能的重要性. 接着，我们提出了基于权重的网络资源调度模型，设计了系统框架，并实现了原型系统. 最后实验结果验证了该系统可以有效地优化作业性能，并能保证不同任务之间的网络性能隔离.

参考文献

1 Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2008, 51(1): 107–113.

2 Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets. Proc. of the 2nd USENIX Conference on Hot Topics in Cloud Computing. 2010. 10–10.

3 http://storm.incubator.apache.org/.

4 Hindman B, Konwinski A, Zaharia M, et al. Mesos: a platform for fine-grained resource sharing in the data center. Proc. of the 8th USENIX Conference on Networked Systems Design and Implementation. 2011. 22–22.

5 Vavilapalli VK, Murthy AC, Douglas C, et al. Apache hadoop yarn: Yet another resource negotiator. Proc. of the 4th annual Symposium on Cloud Computing. ACM. 2013, 5.

6 Chowdhury M, Zaharia M, Ma J, et al. Managing data transfers in computer clusters with orchestra. ACM SIGCOMM Computer Communication Review. ACM, 2011, 41(4): 98–109.

7 Shieh A, Kandula S, Greenberg A, et al. Sharing the data center network. Proc. of the 8th USENIX Conference on Networked Systems Design and Implementation. 2011. 23–23.

8 Al-Fares M, Radhakrishnan S, Raghavan B, et al. Hedera: dynamic flow scheduling for data center networks. NSDI, 2010, 10: 19.

9 Greenberg A, Hamilton JR, Jain N, Kandula S, Kim C, Lahiri P, Maltz DA, Patel P, Sengupta S. VL2: a scalable and flexible data center network. SIGCOMM. 2009.

10 Guo C, Lu G, Li D, Wu H, Zhang X, Shi Y, Tian C, Zhang Y, Lu S. BCube: a high performance, server-centric network architecture for modular data centers. SIGCOMM, 2009: 63–74.

11 Guo C, Wu H, Tan K, Shi L, Zhang Y, Lu S. DCell: a scalable and fault-tolerant network structure for data centers.

In SIGCOMM, 2008: 75–86.

12 Das A, Lumezanu C, Zhang Y, et al. Transparent and flexible network management for big data processing in the cloud. Presented as Part of the 5th USENIX Workshop on Hot Topics in Cloud Computing. USENIX.

13 Narayan S, Bailey S, Daga A. Hadoop acceleration in an OpenFlow-based cluster. High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion. IEEE. 2012. 535–538.

14 McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69–74.

15 http://www.projectfloodlight.org/floodlight/.

16 http://www.xenserver.org/.

17 http://openvswitch.org/.