doi:10.3772/j.issn.1002-0470.2014.08.002

面向混合负载的集群资源弹性调度①

李 勇②[*,***] 张 章[**] 孟 丹[**] 韩冀中[**] 李 青[*,***] 王 旻[*,***]

([*]中国科学院计算技术研究所计算机应用研究中心 北京 100190)
([**]中国科学院信息工程研究所第二研究室 北京 100093)
([***]中国科学院研究生院 北京 100049)

摘 要 针对目前集群资源调度方法难以适应互联网业务多样化、定制化特征的问题,提出了一种面向混合负载的集群资源弹性调度方法。该方法通过构建作业约束描述语言,允许作业基于自身负载特征提出多维度的资源申请和具有负载意识的资源调度算法,实现在同一集群内各类业务统一部署与管理,及时匹配资源需求的变化;通过建立作业的软约束与硬约束之间的转化机制,满足作业在不同执行阶段对资源的定制化需求。实验表明,该方法相比于 Hadoop,可允许作业利用较少资源获得更优性能,在实际生产系统中,基于该方法可将集群资源利用率由 62% 提升到 75%。

关键词 集群资源调度,作业约束调度,负载感知,混合负载,互联网数据中心(IDC)

0 引 言

近年来,新兴的互联网应用发展迅速,互联网企业纷纷建立大规模的集群系统,不仅支持直接面向大量用户的常驻型服务,也支持藏于后端处理海量数据的批处理作业。互联网应用对集群资源的需求纷繁多变,一方面常驻型服务的负载随用户喜好、节假日等因素动态波动,另一方面大数据的多样性也使得批处理作业类型多种多样,包含 MapReduce[1]、DAG、流式计算[2]、迭代计算和图计算[3]等。因此,互联网集群环境各类业务形成混合型负载,不同类型的业务有不同的资源需求,这使得集群资源的高效利用成为一个难题。

当在同一集群中混合部署不同业务时,静态划分资源的方式虽然可以解决资源竞争和冲突,但难以满足业务对资源的定制化需求,且难以实现业务之间的分时复用资源。并且,现有的策略[4,5]主要面向批处理作业这一类负载,未充分考虑对包含常驻服务型应用的混合负载的支持。不同于传统的批处理应用,互联网应用是一种服务型应用,具有业务类型多、性能瓶颈多样、负载波动大、应用间存在依赖关系和用户访问存在突发性等特点。这就给互联网数据中心(Internet Data Center,IDC)资源管理技术带来极大的冲击与挑战:既要支持传统型的批处理应用,又要满足服务型应用的特殊要求,同时还要保证良好的用户体验。现有的云资源共享平台或者不能支持服务型应用,或者不能充分满足服务型应用的特殊需求。针对上述问题,本研究提出了一种面向混合负载的集群资源弹性调度方法,并基于该方法实现了一个集群资源弹性调度系统(cluster resource elastic scheduling system,CRESS)。该系统在国内某知名互联网企业成功部署,可支持常驻型互联网服务应用和传统批处理作业共享同一应用集群,可以使得集群在高峰时段的资源利用率由原先的 62% 提升到 75%,显著地降低了应用成本。

1 相关研究

1.1 集群资源调度方法

伴随 Web 2.0 的兴起,编程框架因处理和挖掘网络上的海量数据而产生。研究发现,一种编程框架一般只专注于解决某一应用领域的问题,没有单一的编程框架可适用于所有的应用场景,因此产生

① 国家自然科学基金(61070028),863 计划(2012AA01A401)和中国科学院先导专项(XDA06030200)资助项目。
② 男,1980 年生,博士生;研究方向:分布式系统和云计算平台;联系人,E-mail:liyong@ncic.ac.cn
(收稿日期:2014-03-17)

了多种不同的编程框架。例如,Google MapReduce[1]和 Yahoo Hadoop[6]面向大规模数据并行处理,Google Pregel[3]面向图计算数据处理,微软 Drayd[7]面向工作流数据处理,Storm[2]面向实时数据处理等。面对新的应用编程模型不断出现的情况,互联网数据中心(IDC)通常的做法是为不同的应用编程模型分配物理上相互隔离的应用集群,这就造成了单个应用集群规模较小、负载不均衡、不利于大作业运行、管理复杂和维护成本高昂等问题。

针对以上问题,伯克利大学开发的 Mesos[8]、中科院计算所开发的 Transformer[9]和雅虎开发的 Hadoop Yarn[10]等数据中心资源共享平台,将多个应用独占集群集合起来组成一个大的应用共享集群。采用公平调度算法,根据应用编程模型实时的资源需求,动态地为其分配资源,从而有效地提高 IDC 资源利用率,降低应用生产成本。然而,这些调度方法主要关注资源分配的公平性,而较少考虑作业自身的特殊需求。不同的编程框架产生不同的业务负载,对资源提出不同的需求。资源需求的差异性体现在硬件配置、运行时环境、资源数量和性能等多个方面,这就给集群作业调度和资源管理带来困难。为此,Google 提出了作业约束调度[4],允许用户根据应用需求设置作业运行必须满足的条件,进而根据这些约束条件进行集群的调度。在作业约束调度的基础之上,美国卡内基梅隆大学(CMU)进一步提出了硬约束和软约束[5],硬约束是指作业运行必须满足的条件,软约束是指作业运行优先满足的条件,这种方法增强了作业描述的语义,可以更好地满足应用多样化的资源需求。

但是,以上调度方法仅仅针对批处理应用而设计,并没有考虑常驻型互联网应用的需求。据 Omega[11]的数据显示,Google 数据中心批处理作业和常驻型应用的资源使用量各约占 50%。常驻型应用的负载波动非常大,资源调度需要支撑集群具有应对突发事件的能力。例如,汶川地震和奥运会等突发或者重大新闻事件会导致应用的用户访问量突增到平时的 2 ~ 10 倍。为此,IDC 通常会预留相当数量的空闲资源,例如国内某知名互联网企业限制应用集群高峰时段 CPU 负载不超过 60%,使得发生突发事件时有足够多的时间去调配 IDC 备份资源或者其他应用的资源,避免应用服务质量大幅下降甚至应用崩溃,这在一定程度上造成了 IDC 资源的严重浪费。

通过分析服务型应用负载特征及实际需求,我们可以看到将不同类型的应用混合部署,并针对不同应用对资源定制化的需求进行动态调度,不仅可以大幅扩展共享应用集群的规模,提高服务型应用应对突发事件的能力,而且可以充分利用批处理应用和服务型应用性能瓶颈和高低峰时段互补的特征,有效地提高应用集群的资源利用率。

1.2　问题分析

如前所述,虽然 Google 和 CMU 已经提出了基于约束的调度机制,然而,无论硬约束还是软约束,都是一种静态的约束描述方式,无法满足混合应用场景的需求。服务型作业具有约束条件在整个作业调度周期内会发生变化的显著特征。为了解决作业动态约束的问题,需要通过识别作业约束的变化,将这些可能发生变化的作业约束定义为作业动态约束。另外,需要采用硬约束和软约束的转化机制,使得作业硬约束和软约束的划分并不是固定不变的,而是在一定的条件下可以相互转换,以进一步提高调度的灵活性。

总体而言,互联网集群平台面对混合型负载面临两个挑战:(1)如何有效满足混合型作业对作业约束调度的需求,比如作业动态约束描述以及作业硬/软约束动态转化机制等;(2)如何挖掘有效的资源共享方式和资源调度策略,适应不同类型的作业有不同负载模式及资源需求的特征,以提高集群资源利用率。针对上述问题,本文提出面向混合负载的集群资源调度方法,具体贡献如下:

(1)拓展现有的作业约束描述,识别并定义作业动态约束,给出一种作业动态约束描述语言,满足不同类型的作业对运行环境、资源配置和性能优化等多方面的需求。

(2)提出支持多种应用类型的具有负载意识的资源管理算法,实现不同资源需求和性能瓶颈的作业自动混合部署,避免相同性能瓶颈的应用部署在同一台节点上,从而提高应用集群整体的资源利用率。

(3)依据作业运行要求和应用集群的实时负载状态,动态地生成作业约束条件,并提供在一定条件下硬约束和软约束相互转化的机制,满足作业在不同执行阶段对资源的定制化需求。

(4)基于该方法,设计与实现集群资源弹性调度系统(CRESS),并成功部署于国内某知名互联网公司的业务系统,同时支持多种类型的批处理作业与互联网常驻型服务。

2 作业弹性约束调度方法

2.1 资源约束问题分析

互联网业务具有类型多、性能瓶颈多样、部署流程复杂、负载波动大和应用之间存在依赖关系等特点,因此对资源的需求提出不同的约束条件。

有一类约束条件是必须满足的。由于开发和编译环境的限制,一些应用会要求节点操作系统、GLIBC 或者 GCC 的版本必须高于某个特定的版本;为了保证性能,图形处理和生物 DNA 序列匹配等应用,会要求节点配置高性能的 GPU 代替通常的 CPU;矩阵运算应用要求节点具有高性能的 CPU 和大容量的内存,等等,我们将上述必须满足的约束条件称之为“硬约束”。

此外,应用编程模型为了优化作业性能也会提出一些作业约束条件。例如 Hadoop 要求将作业下发至存储数据的节点上运行以节省网络带宽;网络密集型应用,要求将作业下发至相同或者相邻交换机的节点上运行,以充分利用相同或者相邻交换机的节点具有相对高的网络带宽。这些约束条件通常需要优先满足,我们称之为“软约束”。

为了支持混合型作业,作业调度策略必须同时满足上述多种不同类型的作业约束条件。现有的作业约束调度都面向批处理应用场景,无法满足批处理和服务型混合应用场景的需求,也不能满足不同类型作业多样化的性能优化需求。

针对上述问题,我们提出了一种弹性资源调度方法,将作业约束分为静态约束和动态约束两种类型。静态约束是指可以直接用于资源匹配的作业约束条件,例如上文提到的操作系统版本要求,GPU、CPU 和内存等硬件配置要求。动态约束是指不能直接用于资源匹配,必须根据作业运行状态和集群负载状态等信息动态地生成资源匹配规则的作业约束条件。

2.2 作业约束描述语言扩展

通过拓展资源描述语言,可以支持多维度的资源申请。传统的面向批处理作业、基于 Slot 的资源描述已经不能满足不同类型作业的资源需求,例如 MPI、Hadoop 和 Mesos 等。为此,需要增加作业资源需求的 Resource 属性,允许应用根据自身性能瓶颈和负载特征,以四元组 < CPU, Memory, Network, Disk > 的形式申请资源,分别表示作业需要的CPU 核数、内存大小、网络出口带宽和磁盘读写带宽。同时,还支持读/写、顺序/随机四种不同类型的磁盘 IO 访问。例如,对于顺序读/写型作业可以申请每秒读/写的字节数,对于随机读写型作业则可以申请每秒读/写次数。

2.3 针对软约束的优化调度策略

互联网应用类型多、性能瓶颈多样的特性也使得不同的应用类型有不同的作业性能优化调度需求。例如搜索索引、日志分析和用户行为分析等批处理应用要求优先将作业下发至保存应用数据的节点上运行,与 MapReduce 和 Hadoop 类似,称为数据本地性软约束(Data Locality);由于相同或者相邻交换机节点之间具有较高的网络带宽,MPI 应用要求优先将作业下发至相同或者相邻机架的节点上运行,称为机架相邻性软约束(NetworkAdjacent)。

首先,将机架相邻性软约束扩展应用到多层架构的服务型应用。一方面,优先将同一服务型作业的任务下发至相同或者相邻交换机的节点上运行;另一方面,优先将同一个服务型应用的不同架构层次的作业,下发至相同或者相邻交换机的节点上运行,以此优化网络密集型服务作业的网络带宽,从而提高它们的性能。

其次,针对作业之间存在依赖关系的特征,提出作业依赖软约束(JobDependency),即优先将依赖作业下发至被依赖作业运行的节点上执行,提高共享内存型应用的性能,例如,图计算和机器学习等内存迭代处理应用。

再次,针对目前数据中心节点一般配置多块高性能硬盘的情况(腾讯、百度和阿里巴巴等互联网公司的数据中心节点一般配置 6-12 块高性能硬盘),提出磁盘独占软约束(ExclusiveDisk),优先为节点上的每个任务分配一块单独的硬盘,避免多个作业同时访问同一块硬盘而引起磁盘 IO 访问竞争,提高了应用磁盘 IO 性能,同时也能实现多个磁盘之间的负载均衡,充分利用所有磁盘的 IO 访问带宽。

表 1 列举了 CRESS 新增的作业属性,并举例说明用户如何使用它们提交作业。

假设作业 A 要求操作系统内核版本不小于 2.6.32,Java 版本不小于 1.5,资源需求为 4 个 CPU 核、8G 内存、500M 网络带宽,50M 磁盘 IO 读写带宽,性能优化上要求部署在相同或者相邻交换机的节点上,满足作业运行要求的节点按照 CPU 和空闲网络带宽排序,则作业 A 的属性值为:

表 1　CRESS 新增属性表

名称	含义	属性值
Resource	资源需求	<1,2G,200M,30M >
DataLocality	数据本地性	DataLocalityin (192.168.1.2-8)
NetworkAdjacent	机架相邻	True 或者 False
JobDependency	依赖作业	DependedJob = JobA
ExclusiveDisk	独占硬盘	True 或者 False
Port	静态端口	Port = 80
DynamicPort	动态端口	True 或者 False
JobFailure	失效作业	True 或者 False
IssueAsWhole	整体下发	True 或者 False
Requirement	硬约束	必须满足的条件
Rank	软约束	节点排序规则

(i) Resource = <4,8G,500M,50M >; NetworkAdjacent = True; Requirement = < OSKernel = 2.6.32&&Java >=1.5 >; Rank = <0.5 × CPU + 0.5 × Idle Network >

3　CRESS 的实现

3.1　系统组织架构

为了实现弹性的集群资源调度,首先要对应用集群进行组织与管理。CRESS 将多个小的应用集群集中起来组成一个大的应用共享集群,应用以分组(Group)的形式共享应用集群的资源。一个应用分组代表一个用户组或者一个应用,拥有一定的资源配额(Group Quota)。CRESS 作为整个 IDC 资源管理和作业调度的核心,将作业调度、作业监控、作业失效处理和资源管理等一般应用编程模型都具有的通用功能抽象成“操作系统”,为互联网常驻应用和并行编程框架提供统一访问的接口,如图 1 所示。

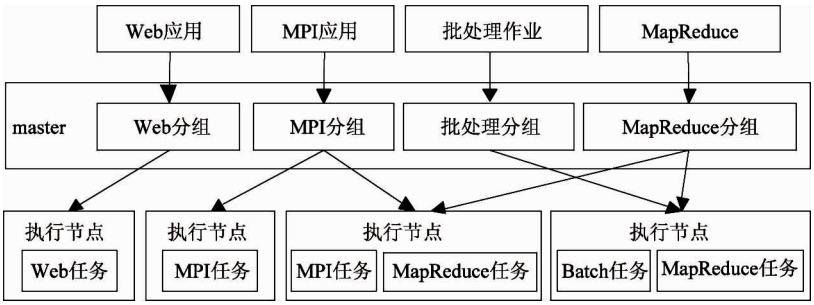

图 1　集群的应用分组

服务型应用通常采用多层架构,比如典型的三层架构模型(表示层、业务逻辑层和数据持久层)。我们将服务型应用的一个层次,称为该应用的一个服务型作业,包括一个或者多个运行在不同节点上的任务。系统可以支持 Batch、Cron、DAG、Web Service、Database 等多种作业类型,但不负责跨 IDC 的资源管理和作业调度。

系统的核心包括 Center Manager、Executor、ZooKeeper 和 Client API 四个主要模块,其中 Center Manager 分为 Resource Manager 和 Job Scheduler 两个子模块,如图 2 所示。各模块之间采用类似于 Condor 的 ClassAd 资源描述协议进行通讯。

用户通过命令行、Web Portal 和 Client API 向 Center Manager 提交、查询和控制作业,以及查询应用集群或者单个节点的资源状态。Job Scheduler 负责应用分组管理和作业调度。首先,它根据应用分组的资源配额以及已用资源数量选择优先级高、有等待作业的应用分组进行作业调度;然后,它根据应用分组选择的作业调度算法进行作业调度,并将被选取的作业信息(Job ClassAd)发送给 Resource Manager 进行资源匹配。Resource Manager 负责应用集群所有节点的状态监控、管理和作业资源匹配。Executor 负责作业的启动、执行和状态监控,并定期地向 ResourceManager 汇报节点资源状态信息。ZooKeeper 是配置中心,它保存所有模块和应用集群节点的配置信息;还起到名字服务的作用。同时,ZooKeeper 还是高可用机制的核心。当主 Center Manager 宕机或者服务不可用时,备份 Center Manager 通过 ZooKeeper 选择新的主 Center Manager。Job Scheduler 实时地向 ZooKeeper 写入所有作业运行状态信息。用户和应用可以通过作业名称获取该作业的运行状态、节点分布和网络端口等作业信息。

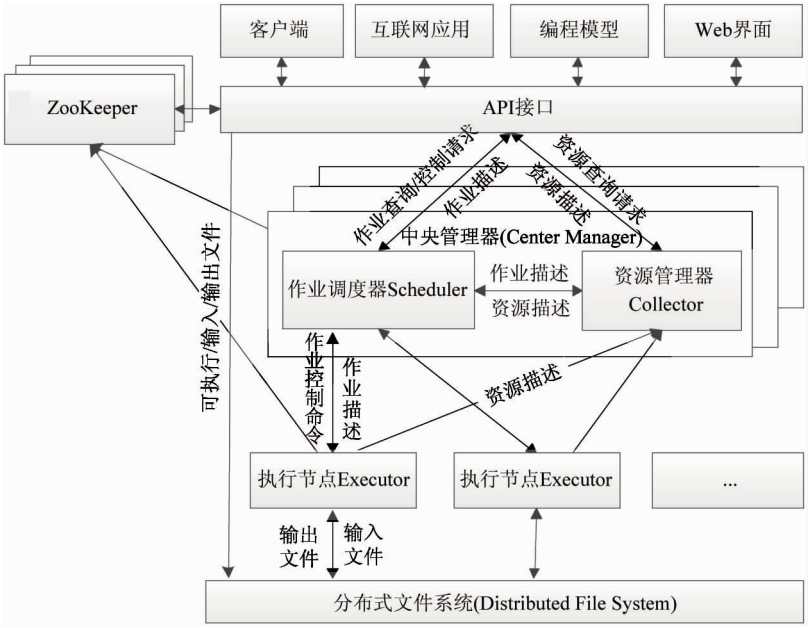

图 2 CRESS 系统结构图

系统采用分布式文件，如 Google 文件系统(GFS)、Hadoop 分布式文件系统(HDFS)和网络文件系统(NFS)等，解决作业的提交和部署、作业数据和日志信息的保存以及应用之间的数据共享问题。为了提高应用性能，系统采用资源容器(Resource Container)，例如 Linux Container 等，而不是传统的虚拟机，实现应用的性能和安全隔离。在拥有高性能的同时，由于资源容器本身不具有独立运行的操作系统，而是与宿主机共享操作系统内核及运行时环境，这就要求系统必须提供额外的支持机制，例如统一操作系统版本和 Glibc、Gcc、Java 等运行时环境，避免将相同端口的互联网应用部署在同一节点等。

3.2 调度动态约束策略实现

作业动态约束的扩展体现在如下三个方面：

(1)动态端口

对于不要求固定端口的作业，例如业务逻辑层和 MySQL 数据库等，用户设置作业属性 Dynamic-Port 属性为 True，表示需要为作业分配一个动态端口。资源匹配时 Resource Manager 根据节点端口占用情况动态地为其分配一个可用端口号。Executor 启动作业时会将该端口号写入到 ZooKeeper 名字服务，方便其它作业查询。

(2)依赖型作业

服务型作业负载波动大的特征使得作业在数据中心节点的部署情况也是动态变化的。作业提交时只能指定它们依赖作业的名称。作业资源匹配时，Resource Manager 根据作业名称从名字服务查询被依赖作业运行的节点列表，生成作业依赖约束条件。

(3)节点失效处理

与 MapReduce、Hadoop 等编程框架类似，调度策略需要制定一套应对节点自动失效的处理机制。然而，依赖型作业的自动失效处理要求保持作业之间的依赖关系，即被依赖作业被重新调度下发执行以后，依赖作业自动下发至相同的节点上运行。这就使得重新调度被依赖作业时，Resource Manager 必须结合被依赖作业和依赖作业的约束条件，动态地生成一个同时满足两者运行要求的组合约束条件。

综上所述，ZooKeeper 名字服务与作业动态约束密切相关：

(1)Executor 启动作业时会将作业的节点机架、IP 地址、端口和磁盘独占等作业动态信息实时地写入 ZooKeeper 名字服务中；

(2)作业资源匹配时，Resource Manager 根据作业名称从 ZooKeeper 名字服务查询上述作业动态信息，并用其替换作业动态约束属性生成作业动态约束条件；

(3)Resource Manager 根据作业动态约束条件进行作业资源匹配，并将节点列表信息返回给 Job Scheduler；

(4)Job Scheduler 将作业下发至 Executor 执行。

3.3 硬/软约束转化机制

不同类型的应用对作业硬约束和软约束的需求也不尽相同，部分应用的硬约束条件一直保持不变，

部分应用的硬约束条件在一定条件下可以转换成软约束条件。例如,标准消息传递接口(message passing interface, MPI)作业的所有任务必须同时下发的作业硬约束条件在整个生命周期内保持不变;Hadoop 的延迟作业调度策略设置了一个作业等待时间阈值 Ts,当作业等待时间小于等于 Ts 时,数据本地性要求是作业的硬约束条件;当作业等待时间大于 Ts 时,数据本地性要求变为作业的软约束条件。为了满足所有应用的需求,CRESS 基于作业硬约束和软约束转换机制,在作业硬约束和软约束设置的基础上,增加一个硬约束转换函数($f(x)$)。目前,硬约束转换函数($f(x)$)主要支持作业等待时间(Waittime)和集群负载(Cluster Load)两个变量,未来可以按照应用需求扩展变量的个数。例如,MPI 作业任务整体下发的条件是不可转换的硬约束条件,因此它的 IssueAsWhole 不带硬约束转换函数 $f(x)$;而 Hadoop 采用延迟作业调度,因此它的作业数据本地性需求是可转换的硬约束条件。假设硬约束转换函数 $f(x)$ = (Waittime > 15s),表示当作业等待时间小于等于 15s 时,数据本地性要求是作业硬约束条件,大于 15s 时,数据本地性要求转为作业软约束条件,则作业 DataLocality 属性值表示为

(ii) DataLocality in (192.168.1.120-121) && $f(x)$ = (Waittime > 15s)

3.4　作业动态约束生成流程

服务型作业负载动态变化、作业之间存在依赖性以及作业硬/软约束的转换机制,使得 CRESS 必须根据作业运行情况动态地生成作业约束条件。假设作业 B 要求 GCC 版本大于 4.5,资源需求为 <1, 2G, 100M, 50M >,采用固定 80 端口,性能优化策略是独占磁盘,转换条件为等待时间大于 5min,优先选择空闲 CPU 较多的节点,作业 Rank 属性为 < CPU >,并且作业 B 依赖于作业 A。假设作业 B 部署的节点有 8 个 CPU 核,磁盘每秒平均顺序读写速度为 100M/s。动态生成作业约束的主要流程如下:

(1)根据作业的节点硬件要求、运行时环境要求、资源需求和固定端口等作业属性,生成作业初始 Requirement 属性和初始 Rank 属性,即作业静态硬约束和静态软约束。例如,作业 B 的初始 Requirement 属性值为

(iii) Requirement = (GCC > = 4.5) && (Resource = <1,2G,100M,50M >) && (Port = 80)

作业 B 的初始 Rank 属性值为(IdleCPU 表示空闲 CPU 核数):

(iv) Rank = 1 × (IdleCPU/8)

为了方便使用,CRESS 支持简单必要的 Rank 属性表达式,与简单的四则运算表达式类似,用户可以为单个表达式赋予权值,不支持多层嵌套的复杂表达式。并且,CRESS 为 CPU、内存、网络和磁盘等性能优化属性提供归一化权重处理。例如作业 B 的 Rank 属性值等于空闲 CPU 的比率而不是空闲 CPU 的个数,以方便 Rank 值的计算。

(2)判断作业性能优化属性是否为可转换硬约束条件。如果是,则判断是否满足转换条件。例如,当作业 B 的等待时间小于 300s 时,不满足 $f(x)$ 函数的转换条件,Requirement 属性值变为

(v) Requirement = (GCC > = 4.5) && (Resource = <1, 2G, 100M, 50M >) && (Port = 80) && (ExclusiveDisk = True)

当作业 B 的等待时间大于 300s 时,满足 $f(x)$ 函数的转换条件,Rank 属性值变为

(vi) Rank = (IdleCPU/8) + (ExclusiveDisk = True)

(3)判断作业是否是依赖型作业,如果是,则 Resource Manager 根据作业名称从 ZooKeeper 名字服务查询上述作业 A 的节点列表,并加入到作业 Requirement 属性中。例如,作业 B 的 Requirement 属性值变为

(vii) Requirement = (GCC > = 4.5) && (Resource = <1, 2G, 100M, 50MD >) && (Port = 80) && (ExclusiveDisk = True) &&Nodes in (192.168.1.120-121)

(4)判断作业是否是自动恢复型作业并且被别的作业所依赖。如果是,则需要将它们的约束条件合并,组成一个统一的作业约束条件。假设运行的节点宕机后,作业 A 和作业 B 需要重新调度运行。此时作业 A 属于自动恢复型作业,并且被作业 B 所依赖。则此时需要合并作业 A 和作业 B 的作业约束条件,作业 A Rank 属性保持不变,Requirement 属性值变为

(viii) Requirement = (OS Kernel > = 2.6.32) && (Java > = 1.5) && (GCC > = 4.5) && (Resource = <5, 10G, 600M, 100M >) && (NetworkAdjacent = True) && (Port = 80) && (ExclusiveDisk = True)

3.5　负载意识的资源调度算法

针对混合型应用性能瓶颈多样、资源需求多样的特征,本文提出了负载意识的弹性资源匹配算法,

实现应用自动混合部署,提高资源匹配的灵活性和可扩展性,满足互联网应用不断发展的需求。主要流程如下:

(1)Collector 收集集群节点实时的作业运行和资源状态信息;

(2)Scheduler 将作业 Requirement 属性和 Rank 属性发送给 Collector 进行资源匹配;

(3)Collector 排除负载过高的节点,例如 CPU 利用率超过 90%,剩余内存空间不足 1G 等,组成待选节点列表;

(4)Collector 根据作业 Requirement 属性从待选节点列表中筛选出满足作业运行要求的可选节点列表;

(5)Collector 根据作业 Rank 属性计算可选节点的优先级,并按优先级由高到低排序;

(6)如果可选节点少于作业任务数量,并且作业要求任务整体下发,则 Collector 向 Scheduler 返回资源匹配失败信息;

(7)作业要求动态端口,则 Collector 为每一个可选节点选择一个可用端口,并将其加入到节点已用端口列表中;

(8)可选节点大于作业任务数量,则 Collector 选择优先级最高的 N 个节点返回给 Scheduler,否则,Collector 将所有可选节点返回给 Scheduler;

(9)Scheduler 根据节点信息下发作业。

为了计算节点的优先级,首先需要计算节点的可用资源数量。CRESS 采用关键资源计算方法计算作业资源使用量。假设节点 i 的资源总量为 < CPU, Memory, Network, Disk >,作业 j 的资源需求量为 $< CPU_j, Memory_j, Network_j, Disk_j >$,则作业资源使用量为:

$$(ix)\ R_i = \mathrm{Max}\left(\frac{CPUj}{CPU}, \frac{Memoryj}{Memory}, \frac{Networkj}{Network}, \frac{Diskj}{Disk}\right)$$

假设节点 i 运行作业个数为 n,则节点 i 已用资源数量为

$$(x) < \sum_{i=1}^{n} CPUi, \sum_{i=1}^{n} Memoryi, \sum_{i=1}^{n} Networki, \sum_{i=1}^{n} Diski >$$

节点 i 的空闲资源数量为:

$$(xi) < CPU - \sum_{i=1}^{n} CPUi, Mem - \sum_{i=1}^{n} Memoryi, Network - \sum_{i=1}^{n} Networki, Disk - \sum_{i=1}^{n} Diski >$$

如 2.3 节所述,作业 B 的 Rank 属性为 Rank = 1 × (Idle CPU/8) + 2 × (ExclusiveDisk = True)。假设目前有节点 C 和节点 D 的可用资源分别为 < 4, 8G, 600M, 100M > 和 < 2, 6G, 500M , 120M >,并且节点 C 有一个空闲的磁盘。则节点 C 的 Rank 值为 4/8 + 2 = 2.5,节点 D 的 Rank 值为 2/8 = 0.25,节点 C 的 Rank 值较大,因此 Collector 会优先将节点 C 分配给作业 B。

综上所述,负载意识的弹性资源管理算法的优点在于只需修改作业的 Rank 属性而无需修改整个资源管理算法,具有灵活、可扩展性强等特点。

4 性能评测

4.1 实际部署和运行情况

CRESS 在 2012 年初在国内某知名互联网企业数据中心成功上线运行,应用于多个互联网应用。例如,与 Google BigTable 类似的分布式数据库服务、搜索引擎倒排索引、面向搜索广告(AFS)和面向内容广告(AFC)系统的日志分析等十几个混合型生产性应用。随着系统的不断完善、成熟和接入应用数量的不断增加,应用集群规模也迅速增长,截至 2012 年底,应用集群总的节点数量达到 8000 台。并且,应用集群的高峰时段资源平均利用率从原先的 62% 提高到 75%,节点数量减少 17%,显著地降低了应用成本,取得良好的经济效果。

4.2 大数据处理性能测试

SortBenchmark[12] 是被知名公司和科研机构广泛认可的大规模数据排序评价的基准测试程序。本文采用 SortBenchmark 中 MinuteSort 和 TeraByteSort 作为测试 CRESS 性能的标准。TeraByteSort 测试 1TB 数据排序所需花费的时间,主要用于小数据集的排序性能测试。MinuteSort 测试平均每秒排序的数据量,主要用于大数据集的排序性能测试。测试应用集群包括 950 个节点,每个节点采用双路 Intel (R) Xeon(R) E5640(2.67GHz) CPU 芯片,64G 内存,12 块 1T 硬盘(SATA),1G 网卡,操作系统 2.6.32。文件系统同样采用公司内部开发的分布式文件系统,相当于 Google 的 GFS。在 2012 年 5 月进行的 TeraByteSort 性能测试中,总共启动 900 个任务,完成 1T 数据排序平均耗时 53s。与 2008 年至今的 TeraSort 世界纪录 Google08 和 Hadoop2009 相比,CRESS 首次实现一分钟内完成的 1TB 的数据排序,如表 2 所示。

表 2　TeraSortBenchmark 测试结果

名称	规模(台)	数据量(TB)	时间(s)
Google08	1000	1	68
Hadoop2009	910	1	62
CRESS	945	1	53

与 Hadoop2009(H 2009)和最新的 HadoopYarn2013(HYarn)的 MinuteSort 性能测试相比,CRESS 使用节点数量明显减少,性能得到很大的提升,具体性能对比数据如表 3 所示。由于硬件技术的发展,CRESS 和 Hadoop Yarn 2013 的硬件配置要优于 Hadoop 2009:三者都采用双路四核 CPU,CPU 频率分别为 2.33GHz,2.30GHz,2.67GHz,相差不大。CRESS 和 Hadoop Yarn 2013 采用 64G 内存、12 个磁盘而 Hadoop 2009 只有 8G 内存、4 个磁盘。由于 Hadoop 2009 和 Hadoop Yarn 2013 并没有采用作业独占磁盘的性能优化策略,因此,磁盘数量的增多对性能的影响并不明显。HadoopYarn 2013 最明显的优势是采用了 10Gb/s 的高速网络,显著提高了 Map 和 Reduce 作业输入输出数据的传输性能,因而性能有了相当大的提升。然而,使用 1Gb/s 带宽的普通网络,凭借良好的资源管理算法和磁盘独占的批处理应用性能优化策略,CRESS 用更少的节点数量取得了更好的数据处理效果:与 Hadoop 2009 相比,只用 2/3 的节点数量,性能提升 130%;与 Hadoop Yarn 2013 相比,使用 43% 的节点数量和普通带宽的网络,实现了 Hadoop Yarn 2013 80% 的整体性能,每个节点的处理性能提升了 87%。

表 3　MinuteSortBenchmark 测试结果

名称	规模(台)	内存(GB)	磁盘(个)	网络 Gb/s	性能(TB/s)
H 2009	1406	8	4	1G	0.5
CRESS	950	64	12	1G	1.15
H Yarn	2200	64	12	10G	1.42

5　结　论

在互联网业务集群中,不仅需要合理调度系统资源,保证系统较高的利用率,而且需要充分满足作业不同的定制化需求。为此,本文提出了一种面向混合负载的集群资源弹性匹配方法,通过构建作业约束描述语言,设计具有负载意识的资源管理算法,建立作业软硬约束的转化机制,有效地为不同类型的批处理作业和互联网常驻型服务应用统一管理起来,不仅大幅增加了应用共享集群的规模,而且充分利用不同类型的应用性能瓶颈和高低峰时段互补的特征,显著地提高了应用集群的资源利用率,降低数据中心的总拥有成本(CTO)。基于在大型互联网成功企业部署与应用的成功实践,验证了该方法的有效性。

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] Stormhttp://storm-project.net: Apache, 2011

[3] Malewicz G, Austern M H, Aart J, et al. Pregel: A system for large-scale graph processing. In: Proceedings of the 2010 International Conference on Management of Data, Indianapolis, Indiana, 2010. 135-146

[4] Sharma B, Chudnovsky V, Joseph L, et al. Modeling and synthesizing task placement constraints in google compute clusters, ACM Symposium on Cloud Computing, Cascais, Portugal, 2011. 34-48

[5] Tumanov A, Cipar J, Gregory R, et al. alsched: Algebraic scheduling of mixed workloads In heterogeneous clouds. In: Proceedings of the Third ACM Symposium on Cloud Computing, San Jose, Canada, 2012. 346-360

[6] Bialecki A, Cafarella M. Hadoop: A Framework for RunningApplications on Large Clusters Built ofCommodity Hardware, http://lucene.apache.org/hadoop: Apache, 2008

[7] Isard M, Budiu M. Dryad: Distributed dataparallel programs from sequential buildingblocks. In: Proceedings of the Europe Conference on Computer Systems, Lisboa, Portugal, 2007. 59-72

[8] Hindman B, Konwinski A, Matei Z, et al. Mesos: A platform for fine-grained resource sharing in the datacenter. In: Proceedings of the 8th USENIX conference on Networked systems design and implementation, Berkeley, USA, 2011. 22-34

[9] Wang P, Mend D, Jizhong Han, et al. Transformer: A New Paradigm for Building Data-Parallel Programming Models. *IEEE Micro*, 2010, 30(4):55-64

[10] VinodK, Arun C. Murthy, et al. YARN: yet another resource negotiator. In: Proceedings of the 4th annual Symposium on Cloud Computing, Santa Clara,Canada, 2013. 58-76

[11] Schwarzkopf M, Konwinski A, Michael A, et al. Omega: flexible, scalable schedulers for large compute clusters. In: Proceedings of the European Conference on Computer Systems, Prague, Czech, 2013. 351-364

[12] Sortbenchmark home pagehttp://sortbenchmark.org: Chris Nyberg, 2012

Cluster elastic resource scheduling for mixed workloads

Li Yong[*,***], Zhang Zhang[**], Meng Dan[**], Han Jizhong[**], Li Qing[*,***], Wang Min[*,***]
([*] Computer Applications Research Center, Institute of Computing Technology, Chinese Academy Of Sciences, Beijing 100190)
([**] The second lab, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)
([***] University of Chinese Academy of Sciences, Beijing 100049)

Abstract

Aiming at the current situation that existing cluster resource scheduling methods cannot meet the diversified and customized requirements of the Internet services, an approach for elastic cluster resource scheduling for mixed workloads was studied. Firstly, a dynamic description language of job constraints was constructed to describe the multi-dimensional resource requirements from various jobs with different workloads. Secondly, a workload-aware resource scheduling algorithm was used to deploy and make overall plan for various business applications in a single cluster, which could respond to the changes in resource requirements in time. Thirdly, a convertible mechanism for soft and hard constraints was designed for meeting customized resource requirements from different running stages of applications. The experimental results show that the above-mentioned method can outperform the Hadoop in terms of consuming less resources and achieving better performance when running jobs. In real production systems, it can increase the resource utilization from 62% up to 75%.

Key words: cluster resource scheduling, job constraint scheduling, workload-aware, mixed workload, internet data center (IDC)