混合型虚拟化资源管理系统①

任 凯[1,2,3], 吴 恒[2], 吴悦文[1,2], 张文博[2]

[1](中国科学院大学, 北京 100049)
[2](中国科学院软件研究所 软件工程技术研究开发中心, 北京 100190)
[3](中国科学院软件研究所 计算机科学国家重点实验室, 北京 100190)

摘 要: 虚拟化发展呈现出多样化的发展趋势, 主要包括Hypervisor和Container两类. 前者隔离型好, 操作便捷; 后者轻量级, 供给快捷. 随着IT技术的快速发展和应用深入, 复杂应用需要协调两类虚拟化对外界提供服务. 提出了一种两级资源管理方法, 第一级调度用以解决物理资源监测、统计、分配决策和隔离, 使得单一物理资源具有多种虚拟化资源抽象能力; 第二级调度用于解决用户资源需求与底层物理资源的放置, 以达到提高物理资源利用率的目的. 同时第一级调度还考虑异构物理资源的差异性, 引入加权 DRF 算法评价异构物理资源对应用性能的影响. 基于CloudSuite测试基准显示, 在保障QoS前提下, 该系统整个资源利用率有效的提升了20%左右.

关键词: 虚拟化; Hypervisor; Container; 资源管理; 两级调度器; 加权 DRF

Hybrid Virtualization Resource Management System

REN Kai[1,2,3], WU Heng[2], WU Yue-Wen[1,2], ZHANG Wen-Bo[2]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[3](State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Virtualization presents a variety of trends in development, including both hypervisor-based and container-based classes. The former virtualization has the characteristic of well isolated and convenient operation, the latter one is lightweight and supply shortcut. With the rapid development of IT technology and the depth in application, complex applications require both types of virtualization to provide services. This paper proposes a two-stage approach to resource management, the first stage is to resolve physical resources monitoring, statistics, allocation of decision-making and isolation, allows a single physical resource has the ability of multiple virtualization resource abstraction; the second stage schedule for solving the resource needs of users and the underlying physical resources in place in order to achieve the object of increased physical resource utilization. While first stage scheduler also takes into account the difference in heterogeneous physical resources, and introduces a weighted DRF algorithm evaluation of heterogeneous physical effect on application performance. Based on the CloudSuite experiments, on the premise of guaranteed QOS parameters, the system resource utilization of the whole system is effectively promoted 20%.

Key words: virtualization; Hypervisor; container; resource management; two-stage scheduler; weighted DRF

1 背景

近年来, 虚拟化技术快速发展, 呈现多样性的发展趋势. Container[1]与 Hypervisor[2][3]技术是两种主要的可商用的虚拟化方案, 正逐渐成为未来IT信息化的主流计算模型. 根据相关研究报告指出, Hypervisor 虚拟化的核心思想是虚拟机指令到物理机指令的重映射技术, 开销相对较大, 适合作为 I/O 密集型(eBay[4]等)服务的运行环境, 典型实现包括 VMWare ESX Server、Xen、KVM 等[5,6]; Container 虚拟化的核心思想是 OS 副本技术, 开销小, 可作为 CPU 密集型(Hadoop[7]、

① 基金项目:国家科技支撑计划(2015BAH55F02);国家自然科学基金(61402450,61363003)
收稿时间:2016-01-31;收到修改稿时间:2016-03-14 [doi:10.15888/j.cnki.csa.005384]

MPI[8]等)服务的底层支撑, 典型实现包括 LXC, Docker, Spoon 等[9]. 可以看出, Hypervisor 具有隔离型好, 操作便捷的特点, 而 Container 具有轻量级, 供给快捷的特点. 根据 Gartner 报告[10]指出, 大型应用需要多种模块协调供给, 这些模块多分为 CPU 密集型应用与 I/O 密集型应用两种. CPU 密集型应用对性能要求较高需要依赖于Container虚拟化方案, I/O密集型应用对读写要求较高依赖于 Hypervisor 虚拟化方案. 多种模块之间需要混合 Hypervisor 与 Container 这两种方案进行配合. 目前来看, 产业界多采用分别管理的方式来单独管理这两种虚拟化架构, 如图 1 所示. 所谓分别管理, 是指虚拟化和物理服务器之间是一对一部署关系, Hypervisor 应用与 Container 应用分别运行在不同的集群之上. 然而, 这种模式意味着 Hypervisor 和 Container 之间不能互相复用资源, 两者都需要按照峰值资源需求提供物理资源, 存在全局资源浪费.

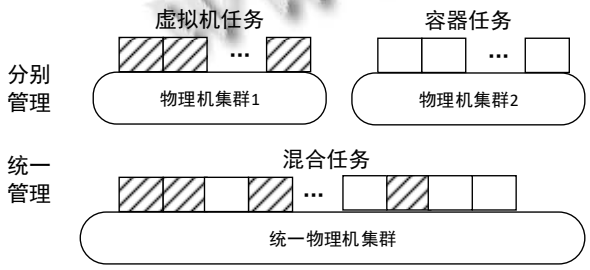

图 1　分别管理与统一管理示意图

因此, 学术界开始尝试混合型虚拟化架构, 允许 Hypervisor 和 Container 同时部署在同一物理服务器上, 通过物理资源的分时复用, 达到提高全局资源利用率的目的. 然而这种架构模式会给云服务的 QoS 保障提出了新的挑战.

本文通过分析 Hypervisor 应用与 Container 应用的不同特点, 设计并实现了一套基于 Mesos[15]系统的混合型虚拟化管理系统该系统通过两层调度结构做到了 Hypervisor 任务资源与 Container 任务资源的闲时复用, 通过实验验证, 系统可以混合管理两种虚拟化资源, 达到了资源的高效利用.

2　相关工作

2.1 以提高资源利用率为目的的云计算资源调度

当前云计算中的资源管理主要包含 3 个研究方向: 以降低云计算数据中心能耗为目标的资源管理、以提高系统资源利用率为目标的资源调度、基于经济学的云资源管理模型研究. 在提高系统利用率研究方面, 当前的主要方法是动态优化物理资源的分配, 以减少云计算所需的物理资源和提高系统资源利用率[11].

根据文献[11]所提出的资源调度模型, 用 $T=\{T_1, T_2, \ldots, T_m\}$ 表示需要放置的虚拟资源, 用 $R=\{R_1, R_2, \ldots, R_n\}$ 表示已存在的物理资源, 云计算中的资源调度可以概括为公式(1)所表示的将 T 放置到 R 上, 寻找少 R 的个数 N 的映射 C:

$$C: T \times R \to F^Z \tag{1}$$

大多数的调度算法研究, 都是通过优化放置映射, 进而实现云计算资源的优化调度. 在具体算法方面, Bardsiri[12], Kaur[13], Chawla[14]等人均提出以线性规划或者整数线性规划为基础的放置算法, 这些算法虽然可以找到最优解, 但在计算过程中需要冗余的迭代计算, 造成了时间与资源的浪费.

为了减少时间消耗, Speitkamp[15]提出了基于线性规划规约的启发式算法. 此外, Xu[16]使用了一种启发式回溯算法来解决云计算中资源调度问题, 他通过使用类似于 DFS 算法来产生最优放置解的子集, 然后通过基于线性规划的近似规约算法以解决用户需求与数据中心最小网络传输的问题.

但是大多数启发式算法往往是单点的, 尽管单点式启发性算法可以快速的发现并找到一个局部最优解, 但是在全局解决方案中单点启发式算法往往效率低下. 因此, 越来越多的云计算资源调度问题往往采用多种算法结合的方法来寻找最优解[14].

2.2 Mesos 统一资源管理框架

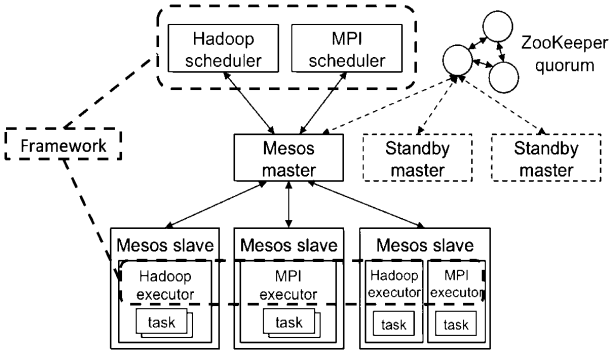

图 2　Mesos 整体架构图

总体架构上, Mesos[18]依然是一个 Master/Slave 结

构的系统, 如图 1 所示, 它主要由 4 个部分构成: Master, Slave, Framework 和 Executor. Master 是非常轻量级的, 负责资源的管理和分配, Slave 负责管理自身资源并向 Master 汇报资源, 同时接受 Master 的调度启动本地的 Executor 作业执行器. Framework 是接入 Master, 负责各类资源统一调度的计算框架.

Mesos 实现了两级调度架构, 它可以管理多种类型的应用程序. 第一级调度是 Master 守护进程, 管理 Mesos 集群中所有节点上运行的 Slave 守护进程. 集群由物理服务器和虚拟服务器组成, 用于运行应用程序的任务, 比如 Hadoop 和 MPI 作业. 第二级调度由 Framework 组成, Framework 包括调度器(Scheduler)和执行器(Executor)进程, 其中每个节点都会运行执行器. Mesos 能和不同类型的 Framework 通信, 每种 Framework 由相应的应用集群管理. 图 2 中只展示了 Hadoop 和 MPI 两种应用, 其他类型的应用也有 Framework.

Mesos 解决了异构应用的资源调度问题, 但是它底层采用的是基于 Container 虚拟化的隔离机制, 也即 Mesos 的异构应用各自运行在 Container 之上, 并没有支持 Hypervisor 的隔离机制, 这对某些需要 Hypervisor 虚拟化环境的应用是不适用的.

3 系统的体系结构

系统采用了一种两级资源调度框架, 具体的体系结构如图 3 所示. 第一级资源调度决定了底层物理资源的分配, 第二级资源调度决定了具体资源任务的调度与运行(Hypervisor 或者 Container)的分配调度.

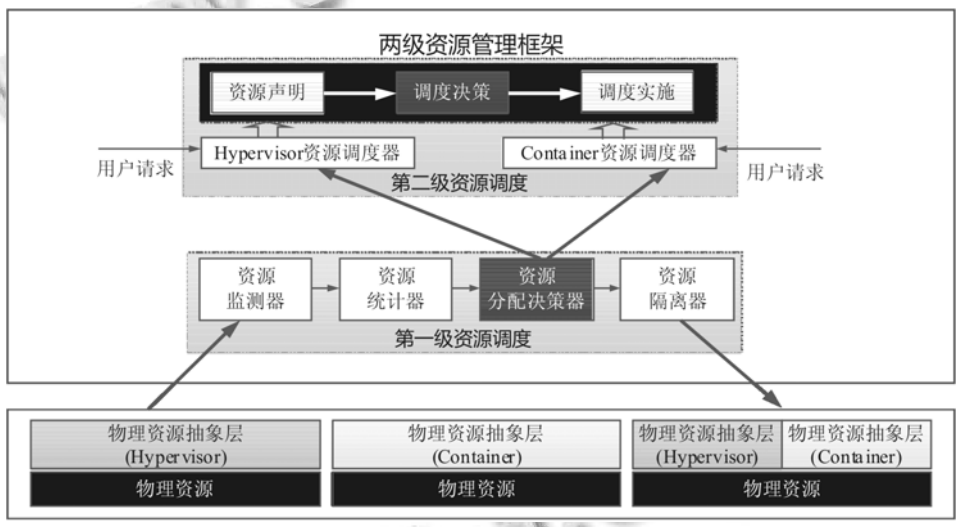

图 3 两级资源管理框架结构图

第一级资源调度器主要由四部分组成, 分别是资源监测器, 资源统计器, 资源分配决策器, 以及资源隔离器. 资源监测器在 zabbix 基础上进行实现, 运行在集群的每台物理机之上, 通过对底层物理资源进行实时监测, 统计可用物理资源(如 CPU, 内存, I/O)为之后资源任务的调度与分配提供数据基础. 资源统计器负责收集统计来自每个资源监测器上的具体资源信息数据, 进行资源的统计与汇总, 之后交由资源分配决策器完成第一级资源调度的分配. 资源分配决策器通过对资源统计器的得到的 Hypervisor 与 Container 具体使用资源情况进行分析, 通过加权 DRF[19]算法为这两种调度器分配底层物理资源. 之后将分配结果通知二级调度器, 二级调度器完成具体类型任务资源调度后反馈给一级调度器, 一级调度器以资源隔离器的形式完成具体资源任务的部署与运行.

二级资源调度器负责完成具体任务调度模块, 按照类型主要分为 Hypervisor 资源调度器以及 Container 资源调度器这两个部分. 具体到每一种资源调度器又可以分为资源声明, 调度决策, 以及调度实施这三个部分. 资源声明部分负责从前台接口获取具体类型资源需求的声明进行形式化转换以及具体资源类型任务生命周期的管理. 调度决策部分基于交叉装填算法根据一级调度器已经分配的资源配额将 Hypervisor 和 Container 任务分配到具体的物理节点上运行部署, 而

调度实施部分负责与一级调度器的资源隔离进行交互完成任务的生命监管.

系统的两级调度完成了从资源分配到任务运行的过程, 一级调度解决了两种调度器资源的公平分配问题, 耳机调度完成了每种调度器公平的处理相应任务队列的问题, 接下来两章将具体介绍这两级调度框架.

4 第一级资源调度的设计与实现

4.1 资源监测器与资源统计器

如前所述, 资源监测器负责收集每台物理主机的已用与可用计算资源. 资源监测器以 agent 的形式安插在每个物理主机上, 每个周期内向资源统计器报告所有监测情况以及任务运行情况. 资源监测器在每个物理主机上存有相应的数据副本, 这样可以保证其崩溃后可以从数据副本快速恢复前一个状态. 而资源统计器的高可用性由资源监测器保证, 这是因为资源统计器只负责具体数据的统计计算转发, 在一个周期内就可以从资源监测器完成这些流程, 当资源统计器崩溃或者宕机时, 一个周期即可完成从资源监测器的快速恢复.

4.2 资源分配决策器

算法 1　加权 DRF 算法

$R = r_1, r_2, \cdots, r_m$　　总的资源
$C = c_1, c_2, \cdots, c_m$　　已使用的资源
$W = w_1, w_2, \cdots, w_m$　　每种资源权重
$s_i (i = 1..n)$　　用户的主占资源, 初始为 0
$U_i = u_{i,1}, \cdots, u_{i,m}$ $(i = 1 \ldots n)$　　分配给用户 i 的资源, 初始为 0
从用户 i 中选取主占资源s_i占比最小的
D_i是用户的下一个任务的资源需求
if $C + D_i \leq WR$ then
　　$C = C + D_i / w_i$　　更新已使用资源
　　$U_i = U_i + D_i / w_i$　　为用户 i 分配资源
　　$s_i = max_{j=1}^{m} \{u_{i,j} / r_j\}$　　重新选取主占资源
else
　　return　　集群用满
end if

资源分配决策器是一级调度器的核心部分, 该部分负责将可用的计算资源按照加权的 DRF 算法分配给每个调度器, 为每个调度器进行资源在分配做准备. Apache Mesos 采用了一种 DRF 算法做资源分配, 使得每一种运行框架所分配资源具有绝对公平性. 但是这种分配方法在分配资源时认为所有的物理资源具有相同的配置. 考虑到我们实际使用过程中不可能保证我们所有的提供计算资源的物理机器具有相同的硬件配置, 我们在 DRF 算法的基础之上进行了相关改进, 对具体硬件资源(如 CPU, 内存)进行相应的硬件评级打分, 赋予一定的权值, 使配置差的机器得到较低的权值, 这样在最后分配时, 保证每种资源调度器的资源加权和是相等的. 具体的加权 DRF 算法如算 1 所示. 算法可以简单的概括为三个阶段: 第一个阶段是准备阶段, 这里从资源统计器的到数据输入, 同时从数据库中读入具体型号的计算资源评分; 第二阶段是计算阶段, 这一阶段是从两种任务调度器中选取分配主占资源加权和较小的资源调度器, 补平其与加权和较多者的主占资源差额; 第三阶段是分配阶段, 将该阶段完成分配的资源调度器的物理资源提供给二层调度框架. 在下个周期内, 再次重复这个过程, 为加权和较小的资源调度器进行任务分配.

4.3 资源隔离器

资源隔离器是资源任务运行的载体, 用以保证每个任务的计算资源不被其他任务侵犯. 在这里我们使用 control groups 对 Hypervisor 以及 Container 实例进行约束. Control groups 是 Linux 内核提供的一种可以限制、记录、隔离进程组所使用的物理资源的机制. 资源的隔离限制可以具体到 CPU, 内存, 磁盘, 网络这四个方面进行.

① 对 CPU 的隔离限制通过 control groups 限制到 Hypervisor 或者 Container 所运行的具体 CPU 核数的序号, 以及该 CPU 下 Hypervisor 或者 Container 示例的使用时间片;

② 对内存的限制通过限制 Hypervisor 或者 Container 可使用的内存上限总量, 保证示例使用内存不越线;

③ 对磁盘的限制通过限制磁盘的实例的读取与写入速度, 保证速度总量不能超过磁盘的额定速度;

④ 对网络的限制通过限制 Hypervisor 或者 Container 实例所对应的虚拟网卡的流入与流出速度.

通过以上这四个方面约束 Hypervisor 或者 Container 实例运行, 保证具体资源任务都可以达到 QoS 需求.

5 第二级资源调度的设计与实现

第二级资源调度器分为 Hypervisor 资源调度器与 Container 资源调度器, 每种资源调度器任务运行又分

为三个阶段: 资源声明, 调度决策, 以及调度实施.

5.1 资源声明阶段

该阶段从用户输入读取资源任务的物理资源需求, 转换为形式化的计算数据. 同时, 该阶段还涉及到 Hypervisor 或者 Container 任务实例的可用形式. 相比与 Container 任务只有启动与停止这两种状态, Hypervisor 实例任务拥有更多的实例状态, 分别是: 运行状态(running), 停止状态(closed), 挂起状态(suspended), 以及暂停状态(stoped). Hypervisor 的挂起与暂停状态涉及到了实例的在线迁移以及动态调整资源的操作, 这四种状态可以相互转换, 转换图如图 4 所示.

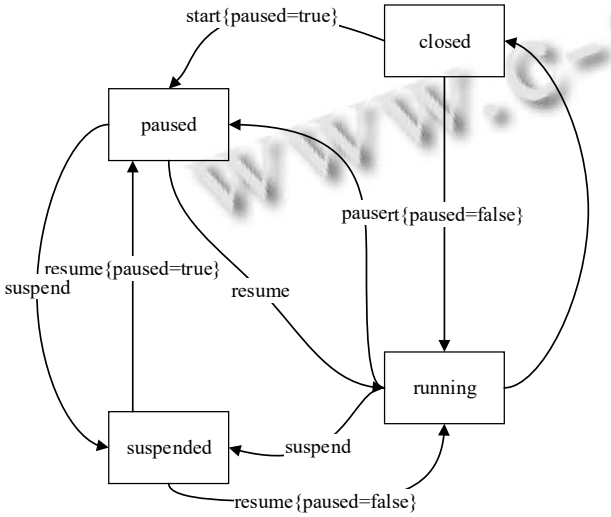

图 4 Hypervisor 实例状态转移图

资源声明通过表 1 所示的结构进行形式化, 以任务队列的形式作为调度决策的输入, 进行资源任务的调度.

表 1 资源声明形式化格式

CPU 需求(CPU 个数, 时间片百分比)
内存需求(内存上限)
磁盘 I/O 需求(磁盘读取与写入速度)
网络需求(网络上传与下载速度)
Hypervisor/Container 实例状态

5.2 调度决策阶段

该阶段是第二级调度的核心阶段, 完成资源任务队列的调度. 由于第一级调度时已经找到了每种调度器所对应的主占资源, 这里我们可以将 Hypervisor 或者 Container 任务放置问题类比为一种装箱问题. 装箱问题是一种典型的 NP—Hard 问题, 求解装箱问题的方法有很多, 如 FFD(First Fit Decreasing)、BFD(Best Fit Decreasing)遗传算法[20]、蚁群算法[21]等. 文献[22]提出了一种解决一维装箱问题的新的近似算法——交叉装填算法(CF 算法), 并证明了该算法可以达到装箱问题最优的近似值 3/2, 同时算法的复杂度也能达到非线性最优 O(nlogn).

相比于 FFD、BFD 等遗传算法, CF 算法具有更优的近似值; 相比于蚁群算法, CF 算法简化了计算步骤, 更加适用于一维装箱问题[22]. 综合考虑, CF 算法可以在 O(nlogn)的时间内获得最优 3/2 的近似值, 通过相关数据实验分析, 我们系统的二级调度算法最终选取了 CF 算法. 具体 CF 算法如算法 2 所示.

算法 2 交叉装填(CF)算法

$V = v_1, v_2, \cdots, v_m$ 资源任务

$P = p_1, p_2, \cdots, p_n$ 已分配的物理资源

1 将虚拟机任务 v_1、v_2、…v_m按主占资源需求 r 大小进行非增排序, 不妨设排序完后的虚拟机序列为 $v_1 \geq v_2 \ldots \geq v_m$;

2 将可用计算资源p_1、p_2、…p_n按照主占资源需求 r 大小进行非增排序, 不妨设排序完后的计算资源序列为$p_1 \geq p_2 \ldots \geq p_N$;

3 将v_1放入物理机p_1中然后从最右端开始依次放入v_m、v_{m-1}直到开启新的虚拟机p_2;

4 重复此步骤直到所有虚拟机放置完毕.

5.3 调度实施阶段

该阶段涉及到资源任务的部署运行与运行时的保障措施. 部署运行是经过两层任务调度后, 任务队列中的资源任务找到了具体运行的物理机, 通过相应的调度 API, 完成资源隔离器的创建以及该任务的部署运行. 任务运行时, 可能遇到任务所在物理主机资源殆尽的无法保证任务声明时期所声明的资源, 这时该阶段的保障措施通过实例的迁移操作以保障任务实例的声明资源得到满足.

图 5 是调度实施过程中通过迁移保证任务运行时声明资源持续保障图. 首先资源监测其模块每个周期向资源统计器汇报实时情况, 当资源统计器得到 Slave1 资源已经用尽时会通知迁移模块对 Slave1 进行资源实例的迁移. 这时, 迁移系统会协调调度系统进行工作: 首先从 Slave1 中选取消耗资源最大的资源实

力 v1, 从其他 Slave 中选取可以提供比 v1 所需资源充足的 Slave 作为目标 Slave. 当源 Slave 与目标 Slave 都选取完毕后, 将该任务插入到调度系统队列进行调度迁移. 如果迁移执行失败则忽略, 等待下次进行迁移.

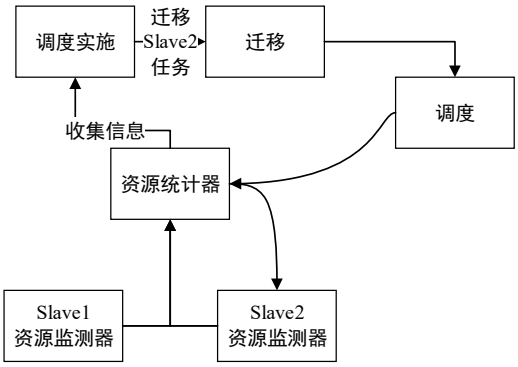

图 5　调度实施迁移图

在迁移过程中, 针对 Hypervisor 与 Container 这两种实例又有两种不同的行为. 对于 Container 这种实例, 我们通过关闭 Slave1 上的实例, 同时在 Slave2 上再次启动这个实例即可完成. 针对 Hypervisor 实例, 我们首先将该实例切换到挂起状态, 完成任务从 Slave1 向 Slave2 的热迁移即可.

6　系统评估测试

6.1 实验环境

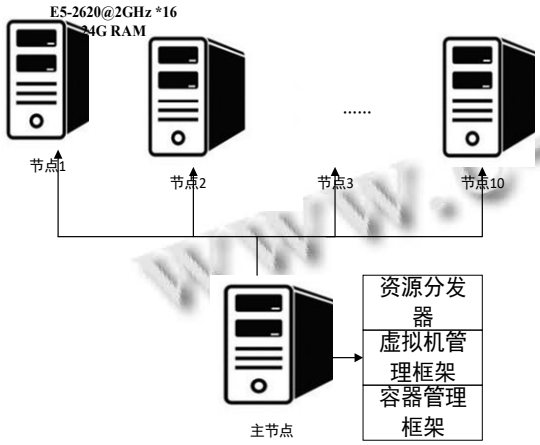

图 6　实验机器部署图

本文使用十台物理机作为整个实验评估测试的运行环境. 十台物理机配置都是 24 核 CPU, 16G 内存. 十台物理机互相通过千兆路由器连接. 每台物理机上部署 CentOS7.1 操作系统, 系统使用 KVM 1.5 与 Docker 1.8.2 版本. Mesos 基于官方 0.24.1 进行修改并使用. 十台物理机, 其中九台分别安装 Slave, 另一台物理机同时安装 Master 与两个资源管理 Framework. 系统的运行部署图如图 6 所示.

6.2 实验评估方法与实验结果

首先进行调度方面的实验, 按照表 2 的任务数目进行模拟, 分析任务用时与机器使用数量:

表 2　实验测试集任务分配表

	虚拟机数目	Docker 数目
测试 1	10	30
测试 2	40	120
测试 3	70	210
测试 4	100	300
测试 5	130	390
测试 6	160	380

虚拟机任务运行时间在[60min,80min]之间随机产生, Docker 任务运行时间在[20min,25min]之间随机产生; 虚拟机的配置在[1cpu, 4cpu]和[2g, 4g]内存之间随机产生, Docker 配置在[1cpu,2cpu]和[1g,2g]内存之间随机产生. 将实验分别运行在传统环境下使用先到先服务的方法与在本系统下运行的时间进行对比, 结果如图 7. 从图中可以看出, 随着任务量的增加, 系统比传统控制的 FCFS 算法任务完成时间有了本质的提高, 这是由于系统做到了物理资源的闲时复用, 在同一个物理机上可以同时运行 Hypervisor 与 Container 这两种类型的虚拟化任务.

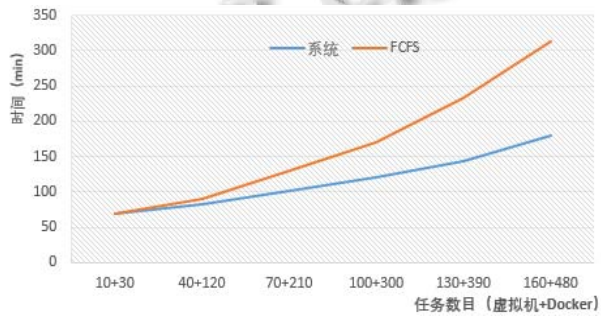

图 7　随机任务调度时间对比图

性能方面实验采用分别单独使用虚拟机运行 CloudSuite[23]的 Web Serving 实验与使用容器运行 CloudSuite 的 Data Analysis 实验, 通过统计两者的资源利用率和性能与通过本系统运行这两个实验的资源利用率和性能进行对比, 做具体分析.

实验环境采用一台刀片服务器做系统的 Slave 结点同时运行这两个测试, 同时使用另一台配置相同的

刀片服务器单独运行这两项实验，实验结果如图 8-10 所示.

图 8 Web Serving 平均相应时间对比图

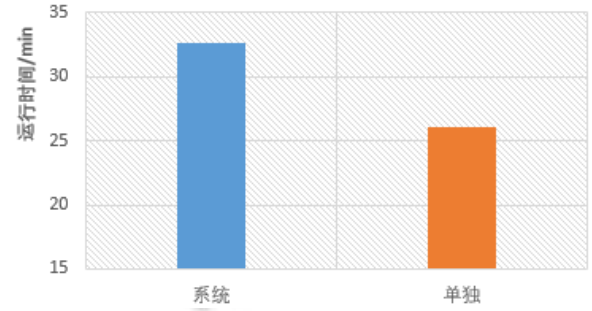

图 9 Data Analysis 运行时间对比图

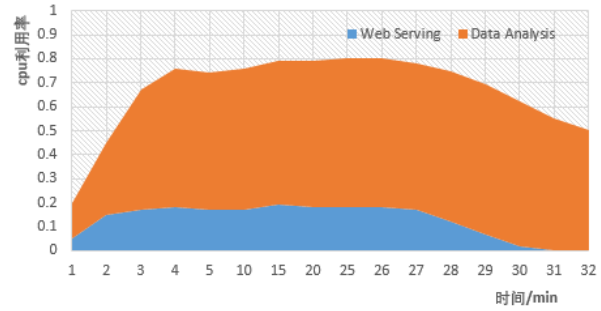

图 10 系统 CloudSuite 实验 CPU 利用率图

可以看出，Web Serving 的平均相应时间在两种运行环境中几乎没有什么差别，均在[0.7s, 1s]这个区间内，可以看出在本系统中运行与原有系统几乎没有差距. 而在 Data Analysis 实验中，由于本系统所在环境中同时进行了两项实验，性能较单独运行环境有些许劣势，但从整个系统的 CPU 利用率图中可以看出，本系统的 CPU 利用率持续在 80%左右，远远高于单独系统的最高 60%的 CPU 利用率. 可以看出，在保证一定性能的情况下，本系统大大提升了集群环境的资源利用率.

7 总结

在现在的云计算环境下中，基于两种虚拟化的数据中心往往是单独优化与管理的. 这导致了需要按照峰值资源进行资源评估与供给，致使集群的整体资源利用率不高. 本文利用开源系统 Apache Mesos 设计并实现了一种在单一集群下混合管理 Hypervisor 虚拟化与 Container 虚拟化的管理系统，在保证原有性能的前提下提高了混合集群的资源利用率. 通过任务分配测试以及 CloudSuite 性能测试，本文设计的混合型虚拟化管理系统达到了混合管理两种虚拟化同时提高资源利用率的目的.

参考文献

1 Soltesz S, Pötzl H, Fiuczynski ME. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. ACM SIGOPS. 2007.

2 Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization. ACM SIGOPS Operating Systems Review, 2003, 37(5): 164–177.

3 Kivity A, Kamay Y, Laor D, et al. KVM: the Linux virtual machine monitor. Proc. of the Linux Symposium. 2007, 1: 225–230.

4 Menasce D. TPC-W: A benchmark for e-commerce. Internet Computing, IEEE, 2002, 6(3): 83–87.

5 Sotomayor B, Montero R S, Llorente I M, et al. Virtual infrastructure management in private and hybrid clouds. Internet computing, IEEE, 2009, 13(5): 14–22.

6 Binu A, Kumar G S. Virtualization techniques: A methodical review of XEN and KVM. Advances in Computing and Communications. Springer Berlin Heidelberg, 2011: 399–410.

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

8 Gropp W, Lusk E, Skjellum A. Using MPI: Portable parallel programming with the message-passing interface. MIT Press, 1999.

9 Xavier MG, Neves MV, Rossi FD, et al. Performance evaluation of container-based virtualization for high performance computing environments. 2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). IEEE. 2013. 233–240.

10 http://www.gartner.com/doc/2862717/-planning-guide-cloud

-computing.

11 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.

12 Bardsiri K, Hashemi SM. 2012. A review of workflow scheduling in cloud computing environment. International Journal of Computer Science and Management Research 1, 2012, (3): 348–351.

13 Kaur H, Singh M. 2012. Review of various scheduling techniques in cloud computing. International Journal of Networking & Parallel Computing, 2012, 1, 2.

14 Chawla Y, Bhonsle M. A study on scheduling methods in cloud computing. International Journal of Emerging Trends & Technology in Computer Science, 2012, 3: 12–17.

15 Speitkamp B, Bichler M. A mathematical programming approach server consolidation problems in virtualized data centers. IEEE Trans. on Services Computing 3, 4 (2010), 266–278.

16 Xu F, Liu FM, Jin H, Vasilakos AV. Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. Proc. of the IEEE 2014, 102(1): 11–31.

17 Zhan ZH, Liu XF, Gong YJ, et al. Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Computing Surveys (CSUR), 2015, 47(4): 63.

18 Hindman B, Konwinski A, Zaharia M, et al. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. NSDI. 2011, 11: 22–22.

19 Ghodsi A, Zaharia M, Hindman B, et al. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. NSDI. 2011, 11: 24–24.

20 Li JC, Chen JY, Wu J, et al. Virtual machine placement research based on improved grouping genetic algorithm. Computer Engineering and Design, 2012, 33(5): 2053–2056.

21 Gao Y, Guan H, Qi Z, et al. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. Journal of Computer and System Sciences, 2013, 79(8): 1230–1242

22 孙春玲,陈智斌,李建平.装箱问题的一种新的近似算法.云南大学学报(自然科学版),2004,26(5):392–396.

23 Ferdman M, Adileh A, Kocberber O, Volos S, Alisafaee M, Jevdjic D, Kaynak C, Popescu AD, Ailamaki A, Falsafi B. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), March 2012.