上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目：分布式环境下调度算法的设计与优化

学生姓名：贾兴国

学生学号：516030910084

专　　业：软件工程

指导教师：戚正伟教授

学院(系)：电子信息与电气工程学院

上海交通大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：_____

日　期：_____年_____月_____日

上海交通大学
学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

保　密 □，在 _____ 年解密后适用本授权书。

不保密 □。

（请在以上方框内打 ✓）

学位论文作者签名：_____　　指导教师签名：_____

日　期：_____年 _____月 _____日　　日　期：_____年 _____月 _____日

分布式环境下调度算法的设计与优化

摘　要

关键词：巨型虚拟机，调度策略，分布式系统，资源使用率，任务迁移

IMPLEMENTATION AND OPTIMIZATION OF SCHEDULING ALGORITHM IN DISTRIBUTED ENVIRONMENT

ABSTRACT

With the rapid evolution of machine learning and data analysis, a single machine fails to provide sufficient resources to the current applications. Distributed systems have the ability to provide a vast amount of computing and memory resources to the upper applications, thus gaining more and more attention from both the industry and the academy. Giant Virtual Machine(GiantVM) solves the compatibility problem between conventional software and distributed environments. It virtualizes multiple physically separated machines to be a single virtual machine, providing a consistent OS interface to the upper software. As a result, traditional applications are able to run on a clustered environment consisted of multiple nodes without any modification. However, the low resource utilization problem still continues to exist. The demand of resources varies among different tasks, and resources required by the same task also changes dramatically over time. As a result, there is a serious waste of a fixed amount of resources assigned to the tasks. Data reported by the industry indicates the under-utilized CPU resources in the distributed environments. For example, average CPU utilization is only 7%-17% in an Amazon cluster. Giant Virtual Machine, however, facilitates the migration of tasks among the cluster, as it exposes a NUMA(Non-uniform memory access) machine to the guest OS, in which a dedicated scheduler can be designed to migrate tasks between busy and idle virtual NUMA nodes, which are physical nodes in a cluster. Thus, QoS (Quality of Service) is guaranteed, and CPU utilization is increased. This paper devises a task scheduler in GiantVM that can dynamically detects the workload of non-migratable tasks in the host and migrate those migratable tasks in GiantVM, thus optimizing the CPU utilization rate in the cluster and ensuring the QoS of LC(latency critical) tasks. Also, Google trace is used to simulate the migrating and scheduling of tasks in a cluster, to compare the performance of multiple scheduling policies and their effects on resource utilization and QoS in a distributed environment.

KEY WORDS: Giant Virtual Machine, scheduling policy, distributed system, resource utilization, task migration

目 录

插图索引 **iv**

表格索引 **v**

算法索引 **vi**

第一章 绪论 **1**
1.1 研究背景与意义 1
1.2 研究现状 2
1.3 本文工作 2
1.4 本文结构 2

第二章 技术背景 **4**
2.1 系统虚拟化简介 4
2.2 巨型虚拟机架构简述 5
2.3 分布式共享内存性能问题 8
2.4 DSM 原理介绍 9

第三章 5G Cloud RAN 中的负载均衡 **10**

第四章 基于深度增强学习的流量优化系统 **11**
4.1 网络拓扑 11
4.2 系统架构 11
4.3 性能分析 11

参考文献 **12**

致 谢 **14**

插图索引

2–1 扩展页表结构 5
2–2 巨型虚拟机架构 6
2–3 MSI 协议状态转化图 7

表格索引

算法索引

第一章 绪论

1.1 研究背景与意义

随着单机纵向扩展的难度越来越大、价格越来越昂贵，企业用户对横向扩展架构越来越青睐。由大量价格低廉的普通机器组成的分布式系统满足了海量数据处理、机器学习等任务的资源需求，逐渐成为工业界和学术界关注的重点。然而，分布式系统对系统软件的开发者提出了新的挑战。例如，如果一个应用程序想要运行在 MapReduce[1] 的分布式平台之上，则必须调用 MapReduce 框架的接口，甚至修改其内部逻辑。这将会带来很大的工作量，削弱分布式平台的优势。巨型虚拟机（Giant Virtual Machine）[2] 解决了现有程序和分布式系统的兼容问题，它向上层应用提供了单一操作系统镜像，将分布式系统抽象为单一的虚拟机平台，使得现有软件无需修改即可运行在分布式系统之上。虽然巨型虚拟机极大的提高了开发者使用分布式平台的便利性，分布式系统的平均资源使用率偏低的问题依然没有得到解决。从阿里巴巴等技术企业提供的数据来看，数据中心的平均 CPU 使用率维持在 30% 左右，不超过 40%[1]。

提高分布式系统中资源使用率的一般方法是，将任务分类为时延敏感型任务（Latency Critical，LC）和尽力而为型任务（Best Effort，BE）。为了保证服务质量（Quality of Service，QoS），时延敏感型任务对资源的要求十分苛刻，不可与其他任务混部，调度器为其提供足够的资源。可以将大量 BE 型任务混部，提高资源利用率。然而，这会导致 BE 型任务的服务质量严重下降，而时延敏感型任务偶尔占用大量资源，而大多数时间占用资源少，使得资源的平均占用率无法提高。为了进一步提高资源平均使用率，分布式系统的任务调度器应该动态地感知系统中各个进程的资源使用量，尽可能地将资源紧缺节点上的任务迁移到资源富余的节点上去，从而保证将资源紧缺节点上任务的服务质量，同时充分利用资源富余节点上的资源。目前实现迁移的方式有进程级迁移（process migration），但由于进程和操作系统共享了内存等资源，遇到了残余依赖（residual dependencies）[3] 的问题；由于虚拟机迁移涉及整个操作系统的状态迁移，虚拟机在线迁移（VM Live Migration）[4] 则会造成巨大的网络开销，也会造成客户机对外提供服务的暂时性中止。

而巨型虚拟机将分布式系统抽象为简单的单一的客户机操作系统，而通过向客户机操作系统暴露非一致性共享内存的硬件架构，使得客户机操作系统得知底层分布式系统的拓扑结构。在隐藏分布式软件框架复杂性的同时，巨型虚拟机也隐藏了分布式系统之间迁移的复杂性，即可以仅仅通过调度客户机内部进程完成集群之间的任务调度。本文通过编写操作系统内的调度脚本，实现了集群节点间的负载均衡，在提高集群总体 CPU 使用率的同时，满足集群中 LC 型任务的 QoS 要求。为了进一步减小集群间任务调度的网络开销和提高集群 CPU 使用率的效果，本文利用 Google trace[2]对分布式集群进行仿真，模拟各类调度策略，研究了不同调度算法以及调度参数对集群性能、网络开销的影响。

[1]不进行任务混部，则仅有 20% 的 CPU 利用率，详见 https://102.alibaba.com/detail/?id=61

[2]谷歌 Borg 集群 29 天内获得的任务调度与资源用量数据：https://github.com/google/cluster-data

1.2 研究现状

在分布式集群的调度策略方面，Mesos[5] 通过细粒度的资源共享提高了集群的资源使用率，然而由于现今的分布式框架都自带有极其复杂的调度器，会彼此产生影响，故 Mesos 设计了一个双层调度器，在各个分布式框架之间进行调度，使得各个框架达到接近最优的数据局部性（data locality）。Omega[6] 是谷歌的集群管理系统，其核心是一个共享状态、无锁的并行程序调度器，使得调度器的延迟大大下降，相比于集中式的集群调度器更好地应对了集群中任务对资源需求的极速变化。随着数据处理任务的并行度越来越高，延迟要求越来越高，Sparrow[7] 提出了一个分布式的、细粒度的调度器，解决了集中式采样系统所造成的吞吐量下降、延迟提高的问题。Graphene[8] 关注的是分布式系统中任务之间的依赖关系，以及多元化的资源需求。在并行数据处理系统中，任务之间的 DAG（directed acyclic graph，依赖关系网）是调度器作出调度决策时所需要关注的主要信息。Graphene 则在任务运行时计算出任务之间的 DAG，对未来的调度策略进行改进。本文的实现依赖于巨型虚拟机，它通过系统虚拟化的方式将分布式系统的复杂性屏蔽，在减轻上层软件编写者负担的同时，也隐藏了诸多优化的可能性。通过修改巨型虚拟机监控器的代码来完成集群调度的优化则是一项相当复杂的工作，不是一个好的选项。

1.3 本文工作

Steal time 指 vCPU（虚拟机 CPU）运行过程中等待物理 CPU 上其他任务所占的时间。本文将巨型虚拟机部署在有四个节点的分布式集群上，将 BE 型任务（可迁移任务）运行在客户机中，同时将 LC 型任务与巨型虚拟机混部，通过读取客户机中的 steal time 计算宿主机上的工作负载，选取巨型虚拟机中 steal time 最低的 NUMA node（非一致性共享内存节点），将虚拟机中所有的 BE 型任务迁移到该 node 上，从而提高了集群总体的 CPU 使用率，也保证了集群中任务的 QoS。本文还利用 Google trace 中 clusterdata2011-2 的 task-uasge 和 task-event 数据，模拟了一个由 800 台相同机器组成的分布式集群，使用 Python 脚本读取数据并模拟了调度过程，设计并测试了各种调度策略的效果。模拟的调度策略有：

- 根据 CPU 占用率调整任务在不同节点上的分布进行调度。其中需要考虑的参数有：每个巨型虚拟机拥有节点的个数、触发进程迁移的 CPU 临界值（CPU threshold）。每个巨型虚拟机占用的节点越少、CPU 占用率的临界值越低，该调度策略就越激进，网络带宽的占用也越大。
- 在考虑 CPU 占用率的基础上同时考虑 Memory 占用率。设置一个内存占用量的临界值（Memory usage threshold），只有在此临界值之下的进程才算作可迁移进程。进程占用的内存越小，网络开销也越小。

1.4 本文结构

本文将按如下形式进行叙述：

- 第二章介绍和本文有关的背景知识，包括虚拟化技术、巨型虚拟机的架构，以及巨型虚拟机中 DSM 组件与 NUMA 的相似性，为后文减小巨型虚拟机任务迁移的开销所做的工作做铺垫。

- 第三章介绍巨型虚拟机中任务调度器的实现，通过动态感知宿主机工作负载进行虚拟机中的任务调度，从而达到提高分布式系统资源使用率的目的，优化集群的任务调度。
- 第四章

第二章 技术背景

2.1 系统虚拟化简介

在计算机科学中，许多问题都可以通过增加或减少一个抽象层解决。在不同的场景下添加不同的抽象层，得到的结果大不相同。虚拟化技术即为一个抽象层。如果将该抽象层置于 CPU，则会产生进程的概念，为系统中所有的任务抽象出可以独占的 CPU，从而使得物理 CPU 的性能得到充分的利用。如果将该抽象层置于物理内存之上，则产生虚拟内存的概念，进程获得了一个连续的广阔的虚拟地址空间，虽然物理内存不一定是连续的。如果抽象出一个 ISA（Instruction set architecture，指令集架构），即一个计算机运行的硬件环境，则产生了系统虚拟化（system virtualization）的概念。通过设计一个 VMM（Virtual machine monitor，虚拟机监控器），即可将客户机操作系统（Guest Operating System，Guest OS）运行在虚拟的硬件之上。系统虚拟化带来了诸多的好处，例如良好的封装性、硬件无关性[9]，这使得虚拟机可以在任何时候停止执行，通过快照在任何时间恢复先前的执行，这方便了虚拟机的热迁移，使得数据中心做到负载均衡。本文正是利用了这一优点。

历史上的虚拟化技术由纯软件虚拟化渐渐转向硬件辅助的虚拟化。早期的虚拟机监控器采用二进制翻译技术，是完全基于软件的虚拟化。由于软件模拟硬件行为的复杂性，纯软件的虚拟机监控器工程量巨大，代码复杂，同时性能相比于硬件环境有明显下降。之后学术界提出了半虚拟化（Para-Virtualization）技术来弥补纯软件虚拟化方式的不足。其想法是通过修改客户机操作系统，使得客户机明确知晓自己处在虚拟化环境中，与虚拟机管理器相互配合，避免了体系结构造成的虚拟化漏洞。最具有代表性的基于半虚拟化的虚拟机监控器就是 Xen[10]，客户机操作系统调用 Xen 提供的 hypercall（虚拟化调用）配合 VMM 完成虚拟化功能。但是，由于 Windows 等闭源操作系统的存在，半虚拟化的可应用范围依然受限。现如今，基于硬件辅助的虚拟化大行其道，Intel 推出的 VT-x（Virtualization Technology for x86 processors）技术[11] 和 AMD 推出的 SVM（Secure Virtual Machine Architecture）技术[12] 在现有的 CPU 指令集架构上增加了专用于虚拟化的指令，例如 Intel 的 VMX 指令[13]。x86 架构的处理器有两种运行模式，root mode（根模式）和 non-root mode（非根模式），客户机运行在非根模式，当客户机需要执行特权指令时，触发 VMexit，CPU 运行模式由非根模式转换为根模式，通过 trap-and-emulate（陷入并模拟）方式模拟客户机的特权指令。硬件辅助的虚拟化的优势在于，免去了对客户机的修改，由于客户机指令可以直接运行在宿主机的 CPU 上，又使得 CPU 虚拟化的代价大大减小。

对于内存虚拟化，传统的方式是为每一个客户机进程维护一个影子页表（SPT，shadow page table），将客户机进程的页表（GPT，guest page table）添加上由 GPA（guest physical address，客户机物理地址）到 HPA（host physical address，宿主机物理地址）的映射，于是 SPT 可以将 GVA（guest virtual address，客户机虚拟地址）映射到 HPA。然而当客户机进程数量较大时，影子页表维护和保存的开销将会十分可观。同时，由于客户机修改自己的页表（GPT）时 SPT 也要进行相应的修改，VMM 将 GPT 所占的内存标记为写保护（write protected），当客户机修改 GPT 时会引起 VMexit，退出到 VMM 中由 VMM 完成对 SPT 的维护。这对于 memory intensive（内存密集）型任务是一种灾难，会引起

大量的 VMexit，严重影响其性能。Intel 提出了 EPT（Extended Page Table），使用硬件维护 GPA 到 HPA 的映射，用硬件替代了软件。如图2–1所示，客户机依然使用自己的 CR3 指针进行地址翻译，将 GVA 翻译为 GPA，而虚拟机监控器为每一个客户机维护一张扩展页表，EPT base pointer（扩展页表基指针）指向扩展页表（EPT），将 GPA 通过硬件翻译为 HPA。于是在客户机修改自身的 GPT（guest page table）时无需引起 VMexit，从而提高了性能。硬件上也有类似于传统页表 TLB（translation look aside buffer）的应用于 EPT 的页表，缓存了 EPTE（Extended Page Table Entry，扩展页表表项），加快了由 GPA 到 HPA 的翻译。使用本文所使用的巨型虚拟机即利用了这一硬件扩展，实现了内存的高效虚拟化。至于 I/O 虚拟化和中断虚拟化，则不在本文的讨论范围内。

图 2–1 扩展页表（EPT）结构[14]

Figure 2–1 Structure of Extended Page Table

根据虚拟机监控器的软件架构，VMM 又可分为 Type-I 型和 Type-II 型。Type-I 型又称为 Hypervisor 型，该类型虚拟机监控器直接运行在裸金属上，直接负责虚拟机的创建、调度、运行、电源管理等，是一个完备的操作系统，所以需要编写大量驱动，工程量较大，但也具有更好的性能。而 Type-II 型虚拟机监控器运行在宿主机操作系统上，仅仅具有虚拟化的功能，而其他的功能由宿主机操作系统实现。还有混合型虚拟机监控器，它同样运行在裸金属上，但是将设备驱动、设备模型的控制权交由一个特权虚拟机。Xen 属于混合性虚拟机，它将操作系统应当实现的功能交由 domain 0 这一特权操作系统完成。然而，Type-I 型和混合型虚拟机需要修改客户机操作系统，依赖于客户机操作系统中的特定驱动，这对闭源操作系统而言是不现实的，同时减小了虚拟化的灵活性，故没有得到大范围应用。而开源虚拟机监控器 QEMU-KVM 属于 Type-I 型，KVM[15] 是 Linux 内核中的内核模块，而 QEMU[16] 是运行于用户态的宿主机应用程序，主要处理客户机的 I/O 请求，将 KVM 作为其虚拟化加速器，KVM 利用 x86 处理器的 VT-x、EPT 等硬件辅助虚拟化功能获得了更优的虚拟机执行效率，而无需修改客户机操作系统，故得到了广泛的应用。下一节中的巨型虚拟机即通过修改 QEMU-KVM 这一开源虚拟机监控器得以实现。

2.2 巨型虚拟机架构简述

巨型虚拟机属于分布式虚拟机，即把一个分布式系统作为运行的物理基础，对上层提供统一的操作系统接口。其主要目的是为了资源聚合，即将分布式系统中的多个普通单机聚合起来，形成一个纵向扩展的虚拟机。举例而言，普通计算机的 CPU 核心数目一般在 50 以内，而更多 CPU 核心数目的机器则会非常昂贵[1]。有了巨型虚拟机，我们可以将多个普通且廉价的物理机聚合起来，启动一

[1]36 个 CPU 核心是服务器较为典型的配置。详见 https://www.quora.com/What-are-the-specs-of-a-typical-modern-server

个拥有海量资源的虚拟机，例如将 5 个配备 36 个 CPU 的服务器进行资源聚集，我们可以得到一个拥有 180 个虚拟 CPU 的客户机，这是单个物理机很难达到的配置。巨型虚拟机基于 QEMU-KVM 这

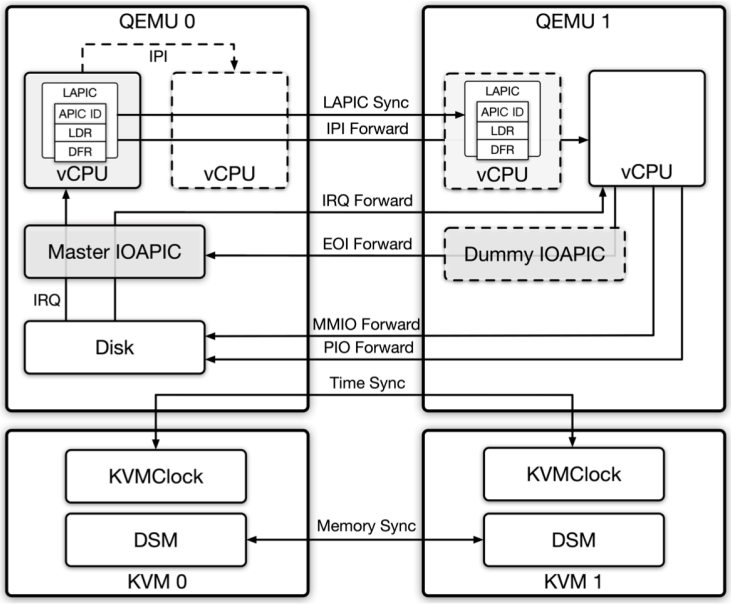

图 2–2 巨型虚拟机架构[2]

Figure 2–2 Architecture of GiantVM

一 Type-II 型虚拟机监控器，将 QEMU-KVM 的功能扩展为分布式虚拟机。其实现主要分为三个部分，这与系统虚拟化常见的三个需要完成的部分相同：CPU 虚拟化，内存虚拟化，I/O 虚拟化。巨型虚拟机的架构如图 2–2所示。支持巨型虚拟机运行的基础设施是多个物理机组成的集群，在集群的每个节点上，都运行着一个巨型虚拟机的 QEMU 实例。每个实例都拥有整个集群的资源，但是属于本机的资源标记为 local（本地资源），而集群中其他节点的资源标记为 remote（远程资源）。当本地虚拟机实例对远程资源发起请求的时候，路由模块会找到所访问资源的真实位置，将资源请求转发到真正拥有该资源的节点上。而内存资源是一个例外：所有的虚拟机的 QEMU 实例拥有全部的内存资源，由 DSM（distributed shared memory）进行提供。完成分布式虚拟化的三个主要功能模块有：

分布式 vCPU 巨型虚拟机添加了新的 QEMU 参数，将一部分本地运行的 vCPU 标记为 local vCPU，而其他未标记的 vCPU 则是 remote vCPU。QEMU 为所有的 vCPU 创建 APIC（advanced programmable interrupt controller，高级可编程中断控制器）。local vCPU 线程在完成初始化之后可以继续执行，而 remote vCPU 线程初始化完成后阻塞（被调度器放置于睡眠队列），直到虚拟机被销毁。当有 IPI（inter processor interrupt，处理器间中断）发送给 vCPU 时，会向目标 APIC 的 APIC ID 寄存器、LDR（Logical Destination Register）寄存器、DFR（Destination Format Register）寄存器写入 IPI 请求。远端 vCPU 的 APIC 称为 dummy APIC（傀儡 APIC）。对于一个不会阻塞的 vCPU 而言，其在不会阻塞的 QEMU 实例上维护一个真实 APIC，而在其他的所有 QEMU 实例上维护一个 dummy APIC。在写入请求时，检查该 IPI 是否发送给远程 vCPU，即检查上述

三个寄存器的写入是否是对 dummy APIC 进行写入。如果是发送给本地 vCPU，即向真实 APIC 写入，则按照原有流程处理；否则，该 QEMU 实例将对 IPI 请求进行转发，将 IPI 注入到远程 APIC，同时将集群中所有该 vCPU 的 APIC 拷贝强制同步，由远程的 QEMU 实例接收 IPI 请求并进行处理。举例而言：集群中有四个节点，每个节点分别运行 QEMU 0-3，而每个 QEMU 实例分别有 vCPU 0-3。当 vCPU 0 向 vCPU 3 发送中断时，先向 QEMU 0 中 vCPU 3 的 dummy APIC 写入上述三个寄存器，同时 QEMU 1-3 从 QEMU 0 得到最新的 APIC 状态。QEMU 3 通过 APIC 识别出 vCPU 3 是 IPI 的接收者，故 QEMU 3 将 IPI 注入到 vCPU 3。

分布式共享内存 内存虚拟化模块是巨型虚拟机最关键的部件，为所有的 QEMU 实例提供了和普通内存完全相同的接口，使得所有虚拟机共享完全相同的虚拟地址空间。普通 QEMU 为虚拟机分配内存的原理是，在宿主机用户空间执行 mmap() 函数，为客户机分配内存。在分布式共享内存开发的初期，为了快速实现并测试分布式共享内存的代码，巨型虚拟机在单机上进行测试，将所有 QEMU 实例的虚拟机内存映射到同一块内存，即对相同的文件调用 mmap() 函数，从而使得所有 QEMU 实例拥有共享的内存空间。在开发的后期，在 KVM 中实现了 DSM 模块，并与 QEMU 对接，才转向真实的分布式共享内存的实现，即通过 RDMA 或 TCP 协议进行所有 QEMU 实例之间内存的同步。

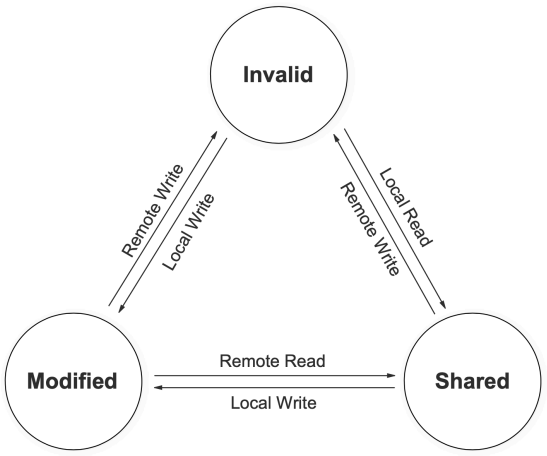

图 2–3 MSI 协议状态转化图[17]

Figure 2–3 Transition of States in MSI protocol

分布式共享内存的主要实现原理是，维护 EPT 中页表项的状态，即维护虚拟机所拥有物理内存页的状态。如图2–3，DSM 的设计基于 Ivy[18]，Ivy 实现了 MSI protocol，满足了 SC（sequential consistency，顺序一致性）。每个客户机物理页的状态分为 Modified：本地 vCPU 对页作出修改，远程 vCPU 尚未获得该页的修改，Shared：本地 vCPU 和远程多个 vCPU 共享该页，Invalid：需要向其他节点获取该页的最新拷贝，否则本地 vCPU 无权对该页进行读写。所以，当本地的 vCPU 发生 Page fault 时，需要使用 RDMA 协议或 TCP 协议向远程节点请求最新的数据页。由于巨型虚拟机中的分布式共享内存需要至少实现 x86-TSO（Total Store Order）[19] 弱的内存模型，故节点间内存同步的次数和较弱的内存一致性模型相比更多。如果巨型虚拟机的两个节点上的进程相

互共享过多的内存，势必会增加内存同步的开销。分析并减小这类开销是本文关注的重点之一。

I / O 虚拟化 I/O 虚拟化需要解决的问题有，虚拟机 vCPU 如何对远程节点上的的模拟设备进行 I/O 访问，以及如何使得远程的虚拟设备产生的中断准确的路由到对应的 vCPU。对于第二个问题，分布式 vCPU 的虚拟化中对于 IPI 的处理已经给出了答案，即在 QEMU 0 上放置 master IOAPIC，而在 QEMU 1-3 上放置 dummy IOAPIC。在设备中断到达之后写入 dummy IOAPIC 时，将信息同步至 master IOAPIC，由 master IOAPIC 将中断通过 dummy APIC 转发给目的 vCPU。对于第一个问题，CPU 对设备进行 I/O 访问有两种方式：PIO（Programmed I/O，针对于 x86 架构）和 MMIO（Memory Mapped I/O）。客户机执行 I/O 指令后退出到 VMM，VMM 检查 I/O 指令的目的设备。如果 I/O 访问的目的设备是远程机器上的设备，则需要将该请求转发给远程设备，在处理完成后将处理结果发送回来。目前，巨型虚拟机的模拟设备全部放置在 master node 上。

2.3 分布式共享内存性能问题

综上所述，巨型虚拟机相比于普通虚拟机的实现增加的部分有：（1）分布式共享内存，这一部分需要通过 RDMA、TCP 等网络协议在集群之间同步物理页的状态；（2）转发机制，将对于远程资源的访问请求转发到对应的节点上去，由真实的资源拥有者处理访问请求。这两部分均会产生网络开销，而网络开销最大的则是分布式共享内存这一组件。由于顺序一致性是是较强的一致性模型，分布式共享内存会产生 false sharing（伪共享）现象和 page thrashing（页面颠簸）现象[20]。伪共享现象是指两个运行在不同 CPU 核心上的线程不断的写入同一个 cache line（缓存行）中的变量，当其中一个线程对变量写入后，另一个核上的缓存行即会失效，于是另一个线程需要从内存中读取最新的数据，而写入变量的线程也需要将缓存行写回主存。如此循环往复即出现了伪共享现象。而页面颠簸现象是由于内存页的换入换出。虚拟内存作为磁盘空间的缓存，可以支持大于物理地址空间的虚拟地址空间。当一个进程频繁访问的页面数量多于总的物理页面数量，或者操作系统为进程选取的驻留集（又称工作集，Resident Set Size，RSS，可以通过 ps aux 等命令行工具观察进程的驻留集）小于其频繁访问的虚拟空间大小，则会有虚拟页面不断换出到磁盘上，即产生了页面抖动现象。

伪共享现象和页面抖动现象都是因为不同层次的存储器访问延迟大不相同造成的。现代计算机有如下几个存储器层次[21]：

- L0 寄存器（Register）：在 CPU 内部用来存储指令和数据。通常在一个时钟周期内即可读写数据，用于缓存来自高速缓存（L1-L3）的数据
- L1-L3 高速缓存（SRAM）：静态 RAM（Static Random Access Memory），L1 访问需要几个时钟周期（约为 4 个时钟周期），L2 访问需要几十个时钟周期（约为 10 个时钟周期），L3 访问需要近百个时钟周期（约为 40 个）。L1 分为数据缓存（dcache）和指令缓存（icache），L1、L2 由单个 CPU 核独享，L3（LLC，Last Level Cache）由处理器中所有的核共享，保存来自于 L4 的数据[1]。
- L4 主存（DRAM）：动态 RAM（Dynamic Random Access Memory），访问需要几百个时钟周期（100ns），用来缓存来自磁盘的数据。
- L5 本地的二级存储，即磁盘（Disk）：磁盘分为 SSD（固态硬盘）和 HDD（机械硬盘）。SSD

[1]数据来源：https://v2ex.com/t/523069

一次读取时间为 16，000 纳秒，HDD 一次读取时间为 2，000，000 纳秒。

- L6 远程二级存储（Web 服务器等）：经过网络协议与网络进行数据交换，访问时间约为 150，000，000 纳秒。[1]

为了获得更大的容量，则会产生更高的延迟。由于分布式共享内存增加了物理内存的容量，需要通过网络访问远程节点的内存，虽然减小了所以引起了更加严重的性能问题。如果在不同节点上的应用程序频繁访问共享的地址空间，则需要分布式共享内存模块进行大量的内存同步工作，不仅造成内存访问的延迟明显提高，使得巨型虚拟机可扩展性降低，还会占用集群中宝贵的网络带宽。虽然分布式共享内存模块已经利用 RDMA、压缩优化等技术大大减小了其占用的网络带宽，但仍需要解决根本问题，即对巨型虚拟机的客户机调度器进行重新设计，使得节点间访问共享内存的机会更少，增强客户机内存访问的局部性。

2.4 DSM 原理介绍

[1]数据来源：https://stackoverflow.com/questions/4087280/approximate-cost-to-access-various-caches-and-main-memory

第三章 5G Cloud RAN 中的负载均衡

第四章 基于深度增强学习的流量优化系统

4.1 网络拓扑

4.2 系统架构

4.3 性能分析

参考文献

[1] DEAN J, GHEMAWAT S. MapReduce: Simplified Data Processing on Large Clusters[C/OL]//BREWER E A, CHEN P. 6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004. [S.l.]: USENIX Association, 2004: 137-150. http://www.usenix.org/events/osdi04/tech/dean.html.

[2] ZHANG J, DING Z, CHEN Y, et al. GiantVM: a type-II hypervisor implementing many-to-one virtualization[C/OL]//NAGARAKATTE S, BAUMANN A, KASIKCI B. VEE '20: 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, virtual event [Lausanne, Switzerland], March 17, 2020. [S.l.]: ACM, 2020: 30-44. https://doi.org/10.1145/3381052.3381324. DOI: 10.1145/3381052.3381324.

[3] DOUGLIS F, OUSTERHOUT J K. Transparent Process Migration: Design Alternatives and the Sprite Implementation[J/OL]. Softw. Pract. Exp., 1991, 21(8): 757-785. https://doi.org/10.1002/spe.4380210802. DOI: 10.1002/spe.4380210802.

[4] CLARK C, FRASER K, HAND S, et al. Live Migration of Virtual Machines[C/OL]//VAHDAT A, WETHERALL D. 2nd Symposium on Networked Systems Design and Implementation (NSDI 2005), May 2-4, 2005, Boston, Massachusetts, USA, Proceedings. [S.l.]: USENIX, 2005. http://www.usenix.org/events/nsdi05/tech/clark.html.

[5] HINDMAN B, KONWINSKI A, ZAHARIA M, et al. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center[C/OL]//ANDERSEN D G, RATNASAMY S. Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011. [S.l.]: USENIX Association, 2011. https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center.

[6] SCHWARZKOPF M, KONWINSKI A, ABD-EL-MALEK M, et al. Omega: flexible, scalable schedulers for large compute clusters[C/OL]//HANZÁLEK Z, HÄRTIG H, CASTRO M, et al. Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013. [S.l.]: ACM, 2013: 351-364. https://doi.org/10.1145/2465351.2465386. DOI: 10.1145/2465351.2465386.

[7] OUSTERHOUT K, WENDELL P, ZAHARIA M, et al. Sparrow: distributed, low latency scheduling[C/OL]//KAMINSKY M, DAHLIN M. ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013. [S.l.]: ACM, 2013: 69-84. https://doi.org/10.1145/2517349.2522716. DOI: 10.1145/2517349.2522716.

[8] GRANDL R, KANDULA S, RAO S, et al. GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters[C/OL]//KEETON K, ROSCOE T. 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016. [S.l.]:

USENIX Association, 2016: 81-97. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/grandl%5C_graphene.

[9] 英特尔开源软件技术中心, 复旦大学并行处理研究所. 系统虚拟化：原理与实现[M]. 北京: 清华大学出版社, 2009.

[10] BARHAM P, DRAGOVIC B, FRASER K, et al. Xen and the art of virtualization[C/OL]//. Bolton Landing, NY, USA: [s.n.], 2003. http://doi.acm.org/10.1145/945445.945462.

[11] UHLIG R, NEIGER G, RODGERS D, et al. Intel virtualization technology[J/OL]. Computer, 2005, 38(5): 48-56. https://ieeexplore.ieee.org/document/1430631.

[12] Advanced-Micro-Devices. AMD64 Virtualization Codenamed "Pacifica" Technology: Secure Virtual Machine Architecture Reference Manual[J/OL]., 2005. http://www.0x04.net/doc/amd/33047.pdf.

[13] Intel. Intel 64 and IA-32 Architectures Software Developer Manual[J/OL]., 2020. https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html.

[14] 董耀祖. 高性能虚拟化及其适用性研究[D]. 上海: 上海交通大学, 2015.

[15] Kernel Virtual Machine[Z]. Website. https://www.linux-kvm.org/page/Main_Page. 2020.

[16] The FAST! processor emulator[Z]. Website. https://www.qemu.org/. 2020.

[17] 丁卓成. 虚拟化系统中基于 RDMA 的分布式共享内存研究[D]. 上海: 上海交通大学, 2019.

[18] LI K, HUDAK P. Memory Coherence in Shared Virtual Memory Systems[J/OL]. ACM Trans. Comput. Syst., 1989, 7(4): 321-359. https://doi.org/10.1145/75104.75105. DOI: 10.1145/75104.75105.

[19] SEWELL P, SARKAR S, OWENS S, et al. X86-TSO: a rigorous and usable programmer's model for x86 multiprocessors[J/OL]. Commun. ACM, 2010, 53(7): 89-97. https://doi.org/10.1145/1785414.1785443. DOI: 10.1145/1785414.1785443.

[20] AMZA C, COX A L, DWARKADAS S, et al. ThreadMarks: Shared Memory Computing on Networks of Workstations[J/OL]. IEEE Computer, 1996, 29(2): 18-28. https://doi.org/10.1109/2.485843. DOI: 10.1109/2.485843.

[21] Randal E. Bryant et al. 深入理解计算机系统[M]. 北京: 机械工业出版社, 2016.

致　谢

IMPLEMENTATION AND OPTIMIZATION OF SCHEDULING ALGORITHM IN DISTRIBUTED ENVIRONMENT