大规模虚拟网络镜像分发优化策略研究

姓名：胡尧

专业：计算机科学与技术

导师：张伟哲

日期：2018.06.13

目录

1.1 背景介绍

近年来，基于传统的云计算平台，出现了很多大规模虚拟网络部署需求，传统的云计算服务在很多场景中并不能提供好的解决方案：

1. 虚拟机镜像越来越大，在虚拟网络部署过程中有大量的时间消耗在镜像分发

2. 大规模虚拟网络场景中，单一虚拟化方案成为性能瓶颈，现有方案不能很好融合

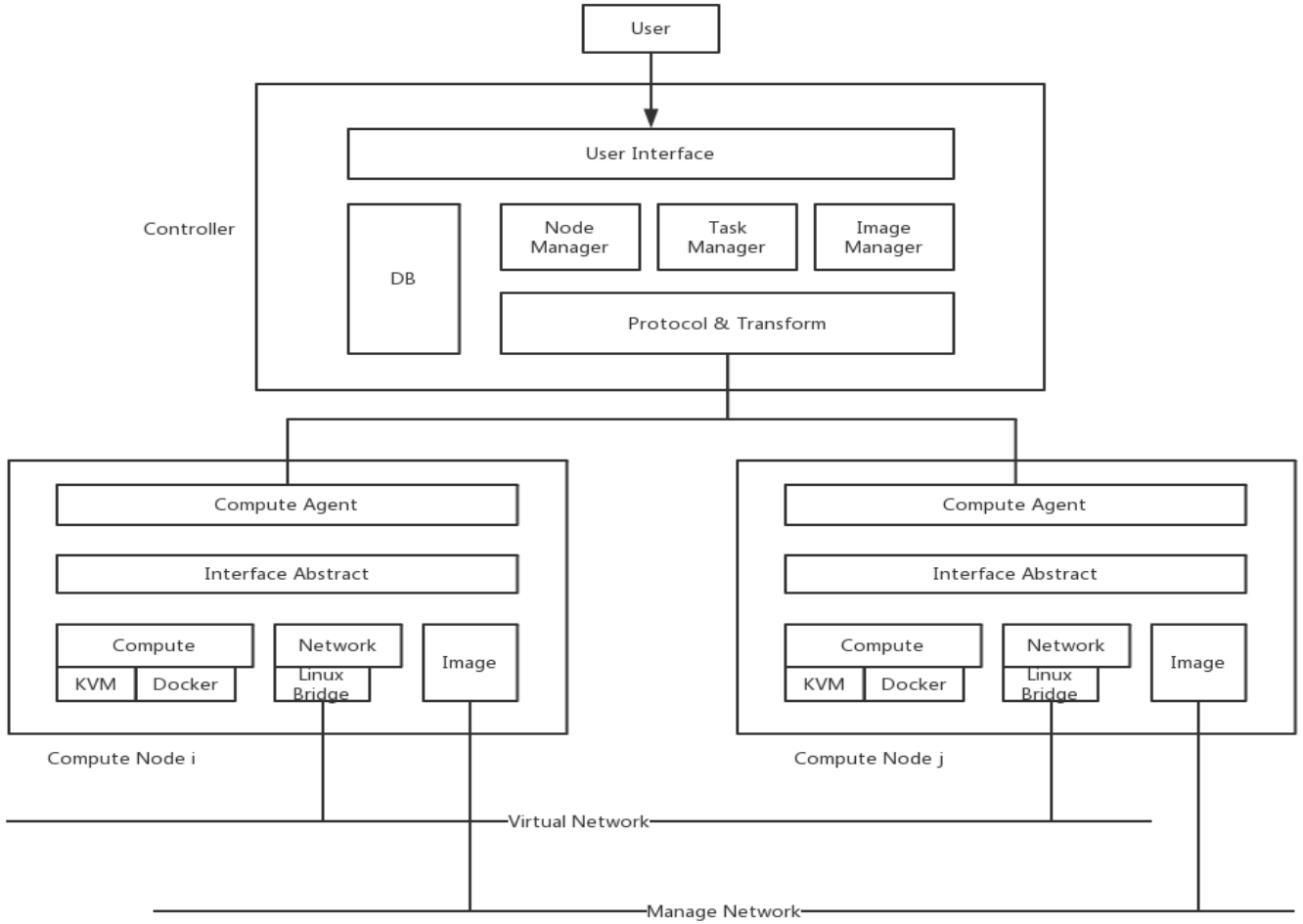
1.2 研究工作

本课题聚焦于虚拟机的镜像格式和分发方案，从以下两个方面进行研究：

1. 构建一个统一的部署框架，将容器技术等技术融入传统虚拟化方案中。

2. 分析虚拟机引导过程对系统镜像的操作，精简镜像的体积，减少镜像传输在部署过程中对网络资源的消耗，提高虚拟机部署效率。

2.1 统一部署框架


User
Controller
User Interface
DB
Node Manager
Task Manager
Image Manager
Protocol & Transform
Compute Agent
Interface Abstract
Compute
KVM
Docker
Network
Linux Bridge
Image
Compute Node i
Compute Agent
Interface Abstract
Compute
KVM
Docker
Network
Linux Bridge
Image
Compute Node j
Virtual Network
Manage Network

2.2 虚拟机容器混合部署系统

在大规模虚拟网络部署场景中，有很多类似于用户行为模拟的虚拟实例，这些实例往往存在几个特点：数量众多、功能简单和资源需求少，对于这些实例，一个完整的最小虚拟化实例所提供的资源已经超过了需求，这样就造成了浪费。

使用容器替代这些实例，可以降低镜像容量，提高资源的利用率。但部分实例是不能用容器替代的，课题实现一个混合的部署系统，融合优化的虚拟机镜像部署系统和容器虚拟化技术，隐藏虚拟化实现细节，为用户提供统一接口。

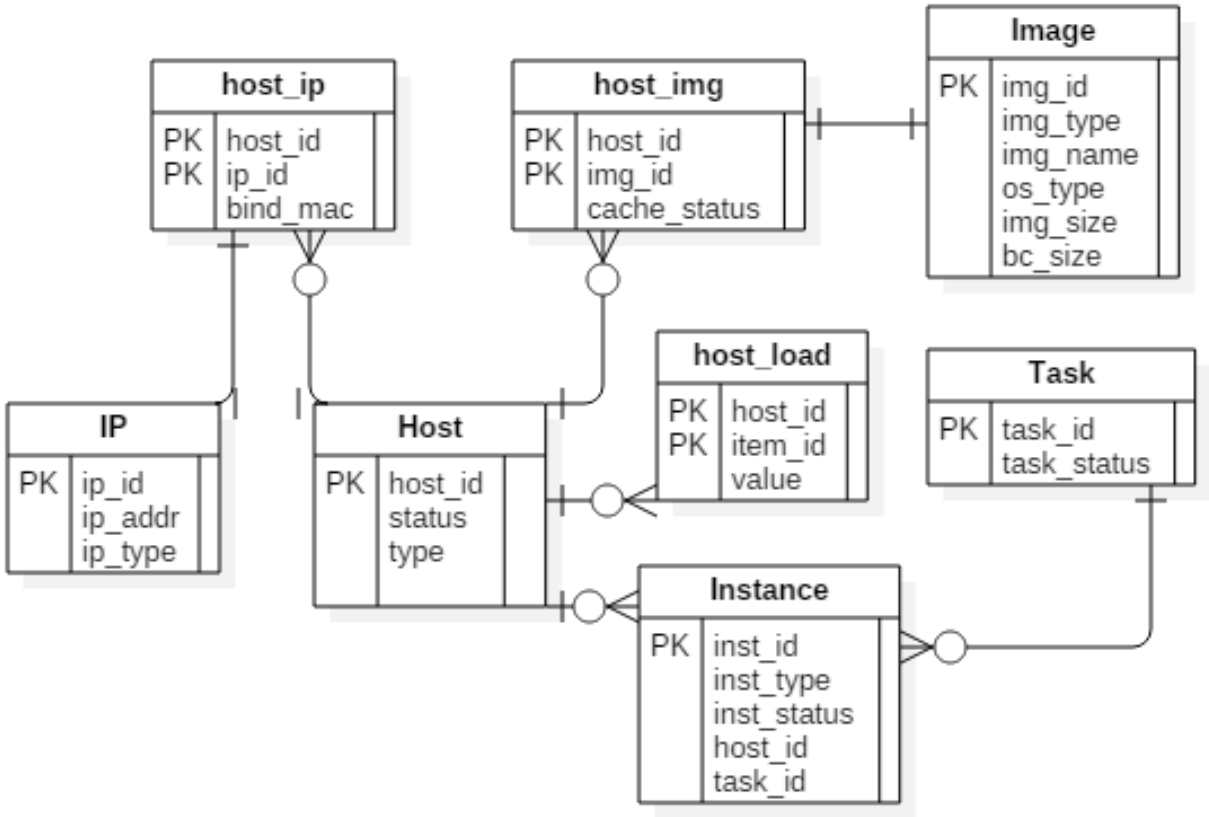
2.3 控制节点

节点管理：维护系统的节点信息，提供了节点注册、节点信息查询等服务。

任务管理：系统的任务管理模块用于根据用户的指令，管理虚拟机和Docker实例，任务管理模块实现了对实例启动和终止的操作。

镜像管理：镜像管理模块用于维护系统的镜像，提供镜像注册、镜像查询和镜像缓存节点优化选择服务。

2.4 数据表


host_ip
PK host_id
PK ip_id
bind_mac
host_img
PK host_id
PK img_id
cache_status
Image
PK img_id
img_type
img_name
os_type
img_size
bc_size
IP
PK ip_id
ip_addr
ip_type
Host
PK host_id
status
type
host_load
PK host_id
PK item_id
value
Task
PK task_id
task_status
Instance
PK inst_id
inst_type
inst_status
host_id
task_id

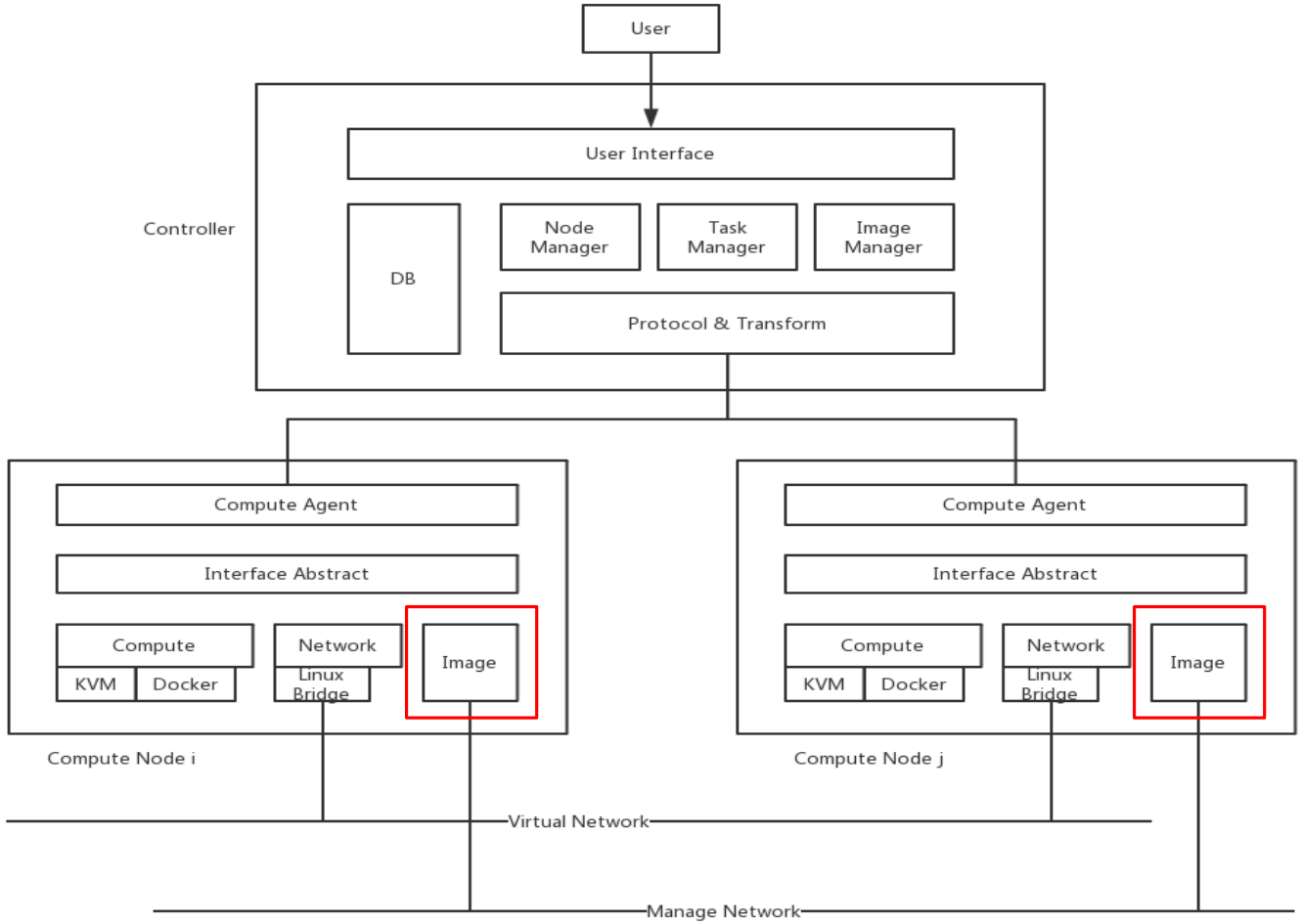
2.5 计算节点

接口抽象：计算节点实现了一个抽象层，用于隐藏各种不同Hypervisor和虚拟网络设备的实现，向控制节点提供统一的接口。

虚拟化：分别使用Libvirt和Docker作为传统虚拟机实现和容器实现，将其接口封装为统一的格式。

网络互通：使用Linux Bridge将虚拟机的网络与容器网络连通。

3.1 虚拟机镜像优化部署策略



3.1 虚拟机镜像优化部署策略

在很多虚拟化场景中，我们对虚拟实例部署的一个重要指标是虚拟实例从分发到进入工作状态的耗时。一般来说，部署的过程分为两个阶段：

1. 镜像数据从存储节点传输到计算节点

2. 计算节点使用镜像作为虚拟实例的存储磁盘并启动虚拟实例

3.2 优化原理

虚拟机操作系统在启动过程中实际需要的数据较少，课题将这部分数据（Boot Cache）从镜像中分离，优先传输到计算节点，让操作系统尽快引导起来。

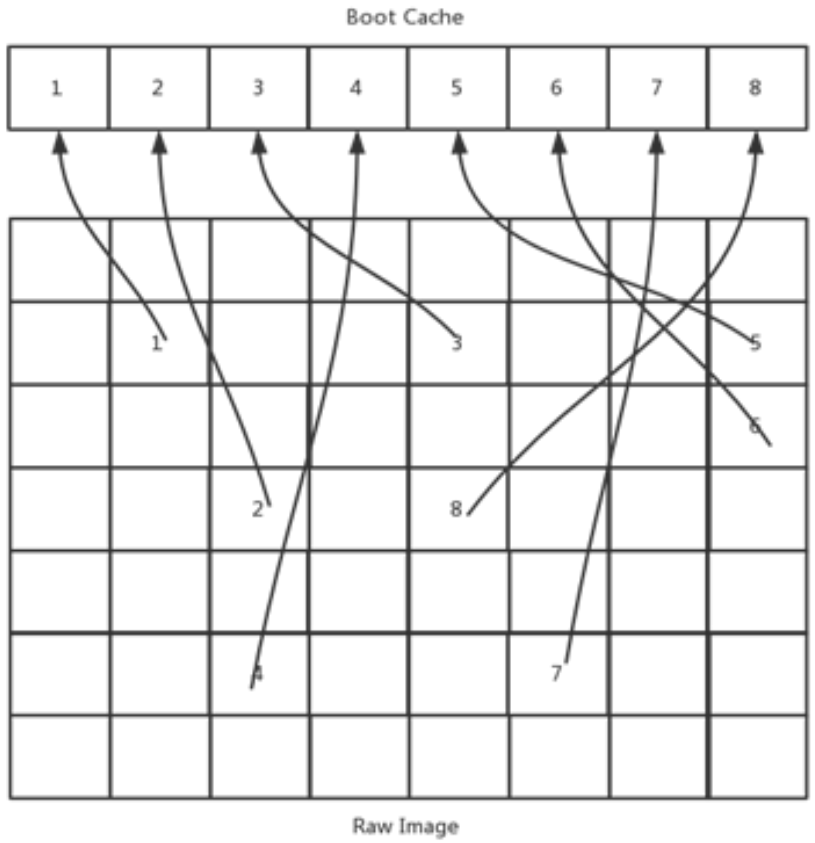
对这部分数据进行排序，排序的依据是操作系统启动时读取数据的顺序，排序靠前的数据将被优先传输到计算节点。

在镜像数据传输开始的同时，操作系统也可以启动

3.3 引导数据(Boot Cache)

操作系统引导时候需要的数据零碎地分布在磁盘镜像中

将这些数据从镜像中提取出来，按照数据读取的顺序排序之后存放在连续的数据块中。

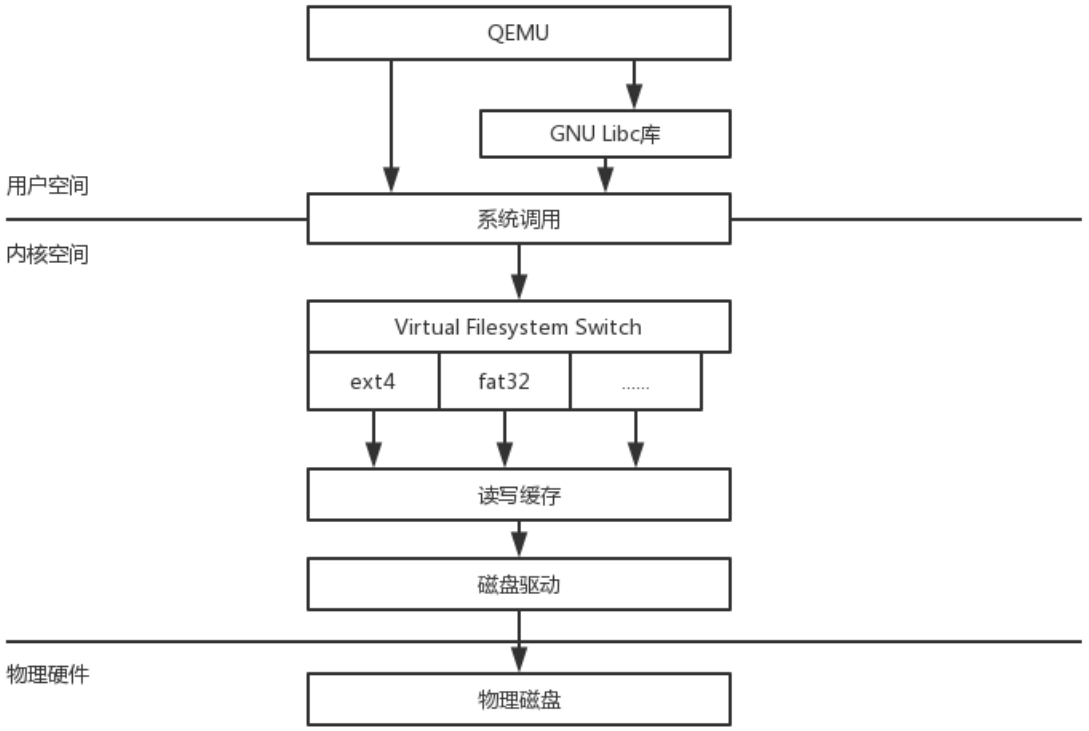


3.4 数据获取

操作系统引导过程对数据的读取都是通过宿主机的I/O请求完成

通过Hook来监视宿主机的I/O请求，记录数据读取的偏移、长度和时间戳

选择系统调用层进行Hook



3.5 Boot Cache构造

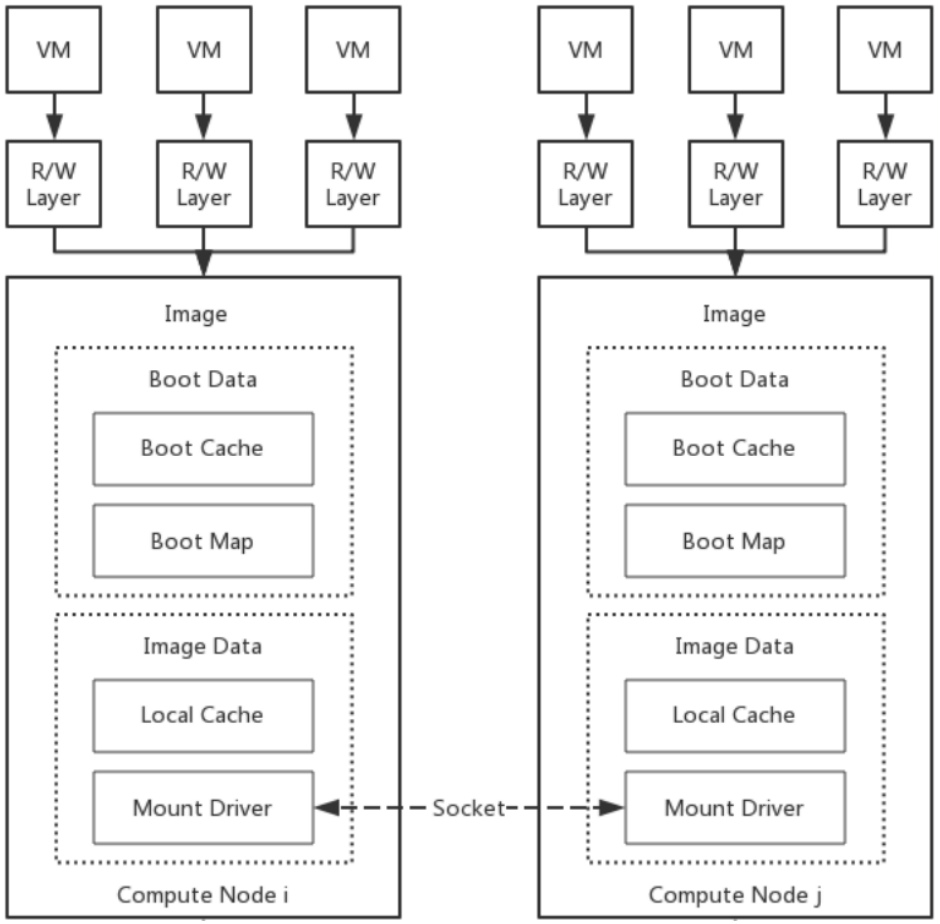
对任意一个系统镜像通过Hook方式拿到I/O请求之后，可以构造对应的Boot Cache：

设镜像文件的总大小为$S\ (S \in N^*)$，操作系统引导过程对镜像文件的读请求数总为$C\ (C \in N^*)$，第$i\ (i \in N^*, 1 \leq i \leq\ C)$次请求的偏移为$O_i\ (O_i \in N^*, 0 \leq O_i < S)$，长度为$L_i\ (L_i \in N^*, 0 <\ L_i \leq S)$，则引导所需数据在镜像文件中所占区间$D$为：

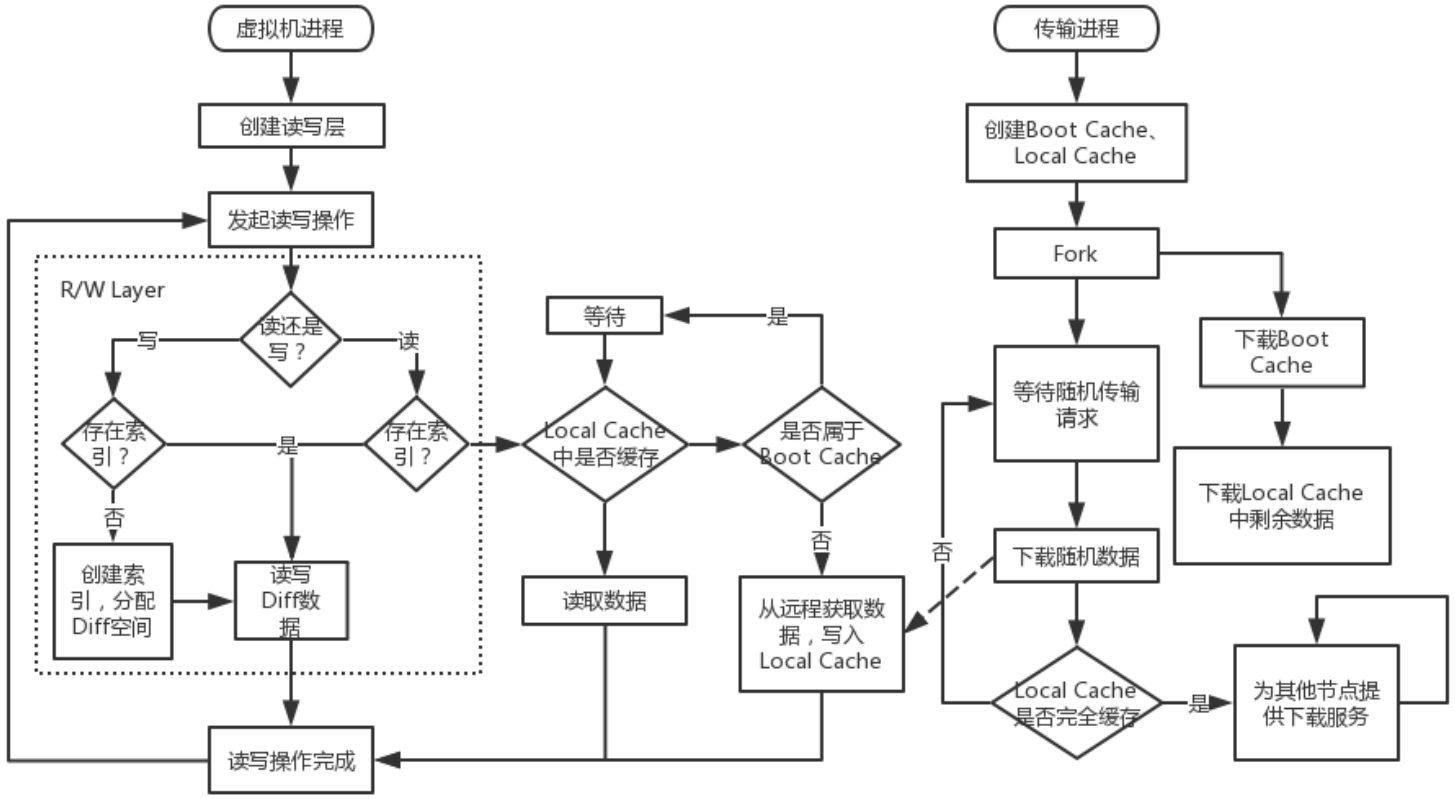
$$D = \cup_{i \in C}\ Q_i$$

$$\text{其中} Q_{\mathrm{i}} = \{x \in N^* \mid O_i \leq x <\ O_i + L_i\}$$

3.6 镜像结构


VM
VM
VM
VM
VM
VM
R/W Layer
R/W Layer
R/W Layer
R/W Layer
R/W Layer
R/W Layer
Image
Boot Data
Boot Cache
Boot Map
Image Data
Local Cache
Mount Driver
Socket
Image
Boot Data
Boot Cache
Boot Map
Image Data
Local Cache
Mount Driver
Compute Node i
Compute Node j

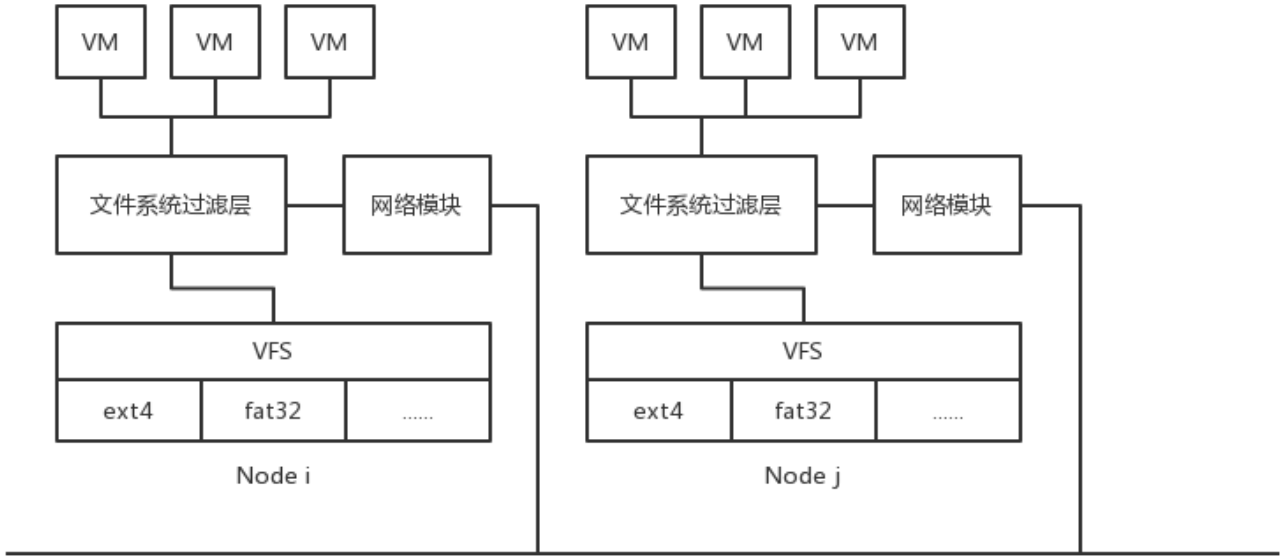
3.7 启动流程


虚拟机进程
创建读写层
发起读写操作
R/W Layer
读还是写？
写
读
存在索引？
是
存在索引？
否
创建索引，分配Diff空间
读写Diff数据
读写操作完成
等待
是
Local Cache中是否缓存
是否属于Boot Cache
读取数据
否
从远程获取数据，写入Local Cache
否
传输进程
创建Boot Cache、Local Cache
Fork
下载Boot Cache
等待随机传输请求
下载Local Cache中剩余数据
下载随机数据
Local Cache是否完全缓存
是
为其他节点提供下载服务

3.8 实现

通过在文件系统和虚拟机之前构建一个过滤层来处理虚拟机对镜像文件的操作

从网络传输过来的镜像数据会缓存在本地，通过Linux通用文件系统接口实现



4.1 实验及分析

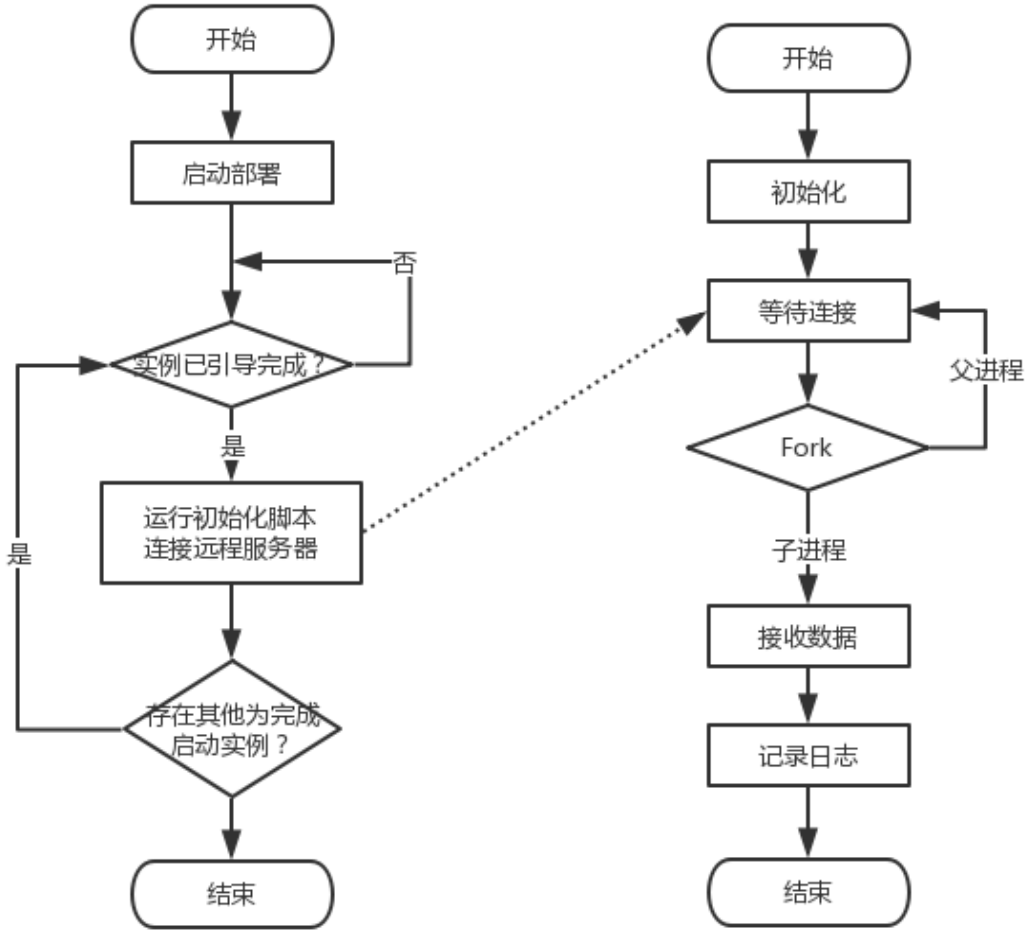
实验环境

节点	CPU数量 个	内存大小 GB	网络 Mbps	操作系统
控制节点*1	4*1	16*1	1000/100	Ubuntu 14.04 Server 64bit
计算节点*4	16*2	16*8	1000/100	Ubuntu 14.04 Server 64bit

测试用操作系统

#	操作系统	镜像实际大小 MB	镜像逻辑大小 MB
1	Ubuntu 14.04	1658	5120
2	Ubuntu 16.04	1759	5120
3	Cent OS 6.4	2133	5120
4	Windows 7	7456	20480
5	Ubuntu 14.04 (Docker)	188	5120

4.2 测试流程


开始
启动部署
否
实例已引导完成？
是
运行初始化脚本
连接远程服务器
是
存在其他为完成
启动实例？
结束
开始
初始化
等待连接
父进程
Fork
子进程
接收数据
记录日志
结束

4.3 I/O分析

为了统计分析操作系统引导过程所有I/O请求的大小分布情况，课题记录了几种不同操作系统引导过程的I/O请求，并将这些请求数据的大小进行了统计。步骤如下：

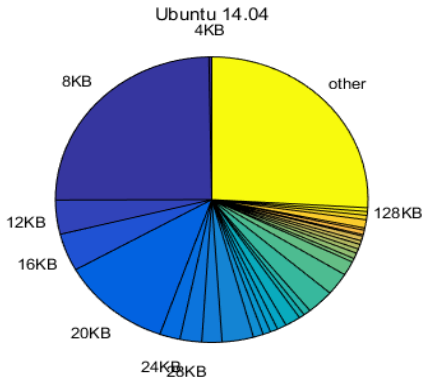
1. 启动虚拟机

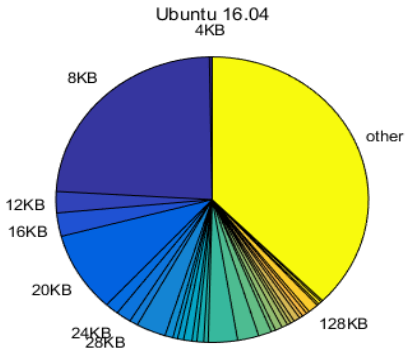
2. 记录虚拟机所有I/O请求

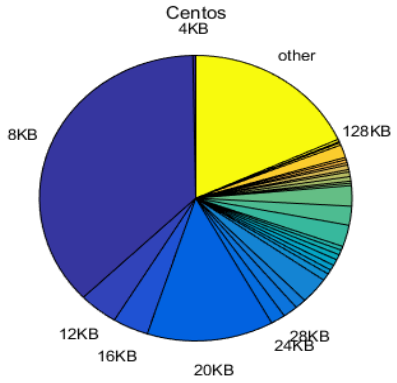
3. 虚拟机启动完成，停止记录

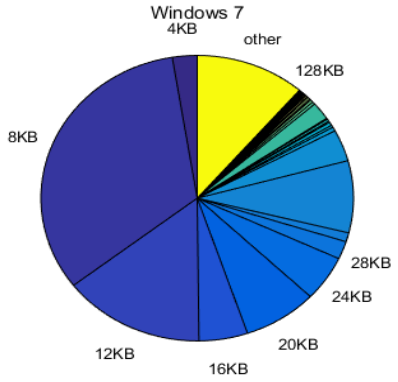
4. 统计分析

I/O分析—单次请求数据大小

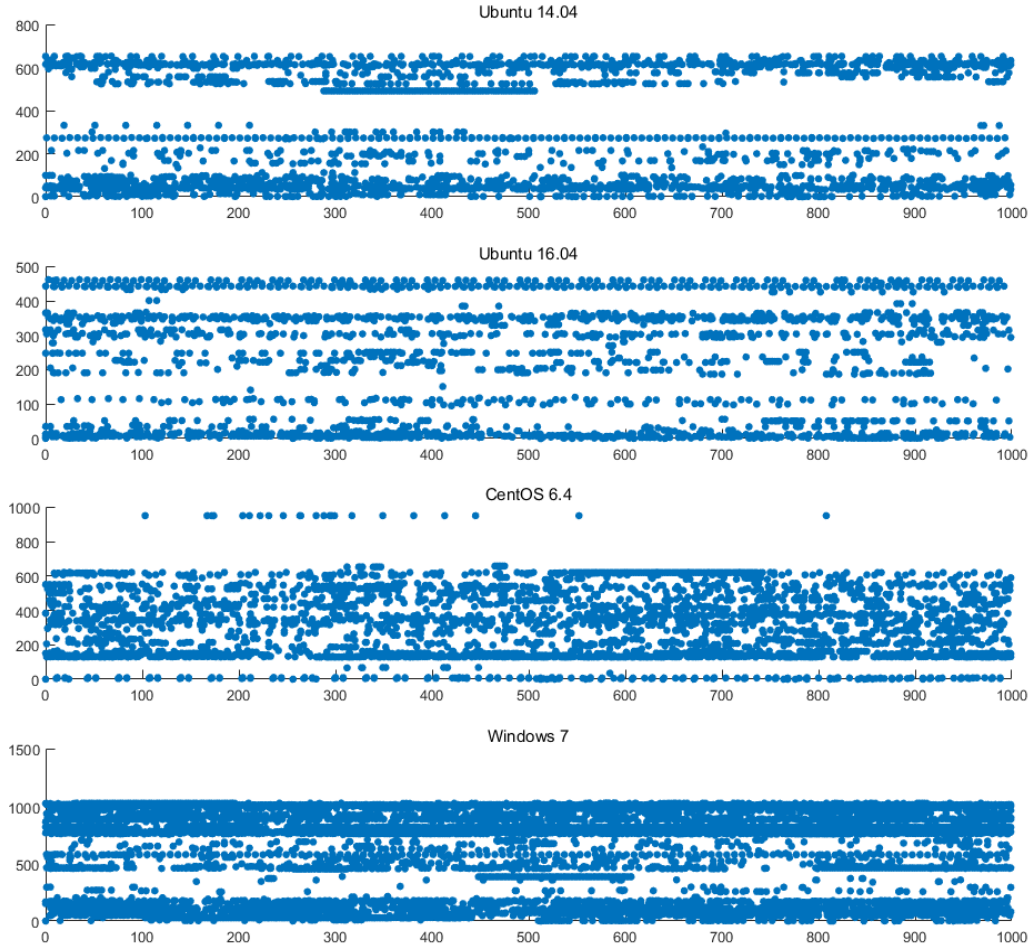




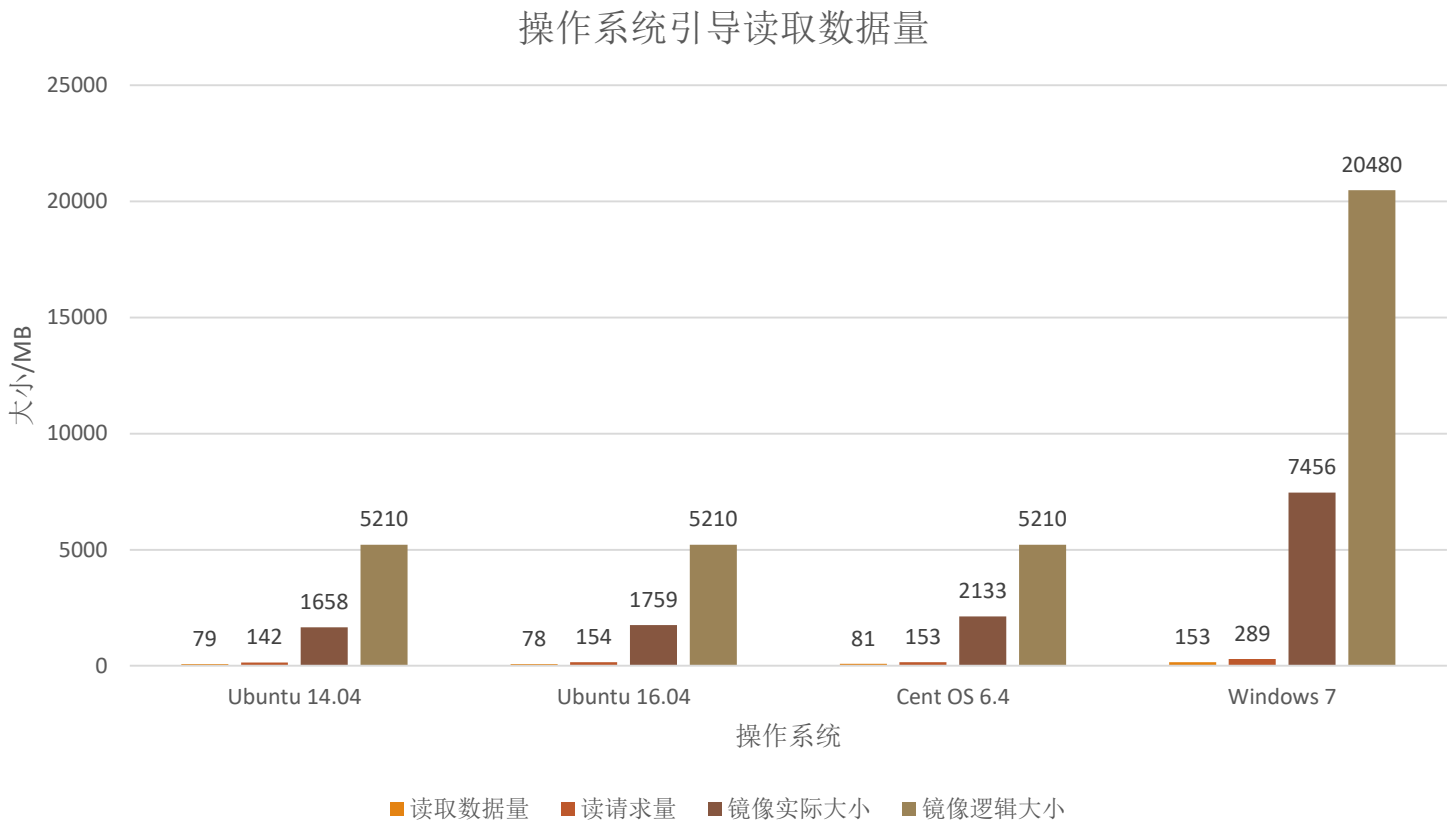




I/O分析—单次请求数据偏移



I/O分析—引导数据大小

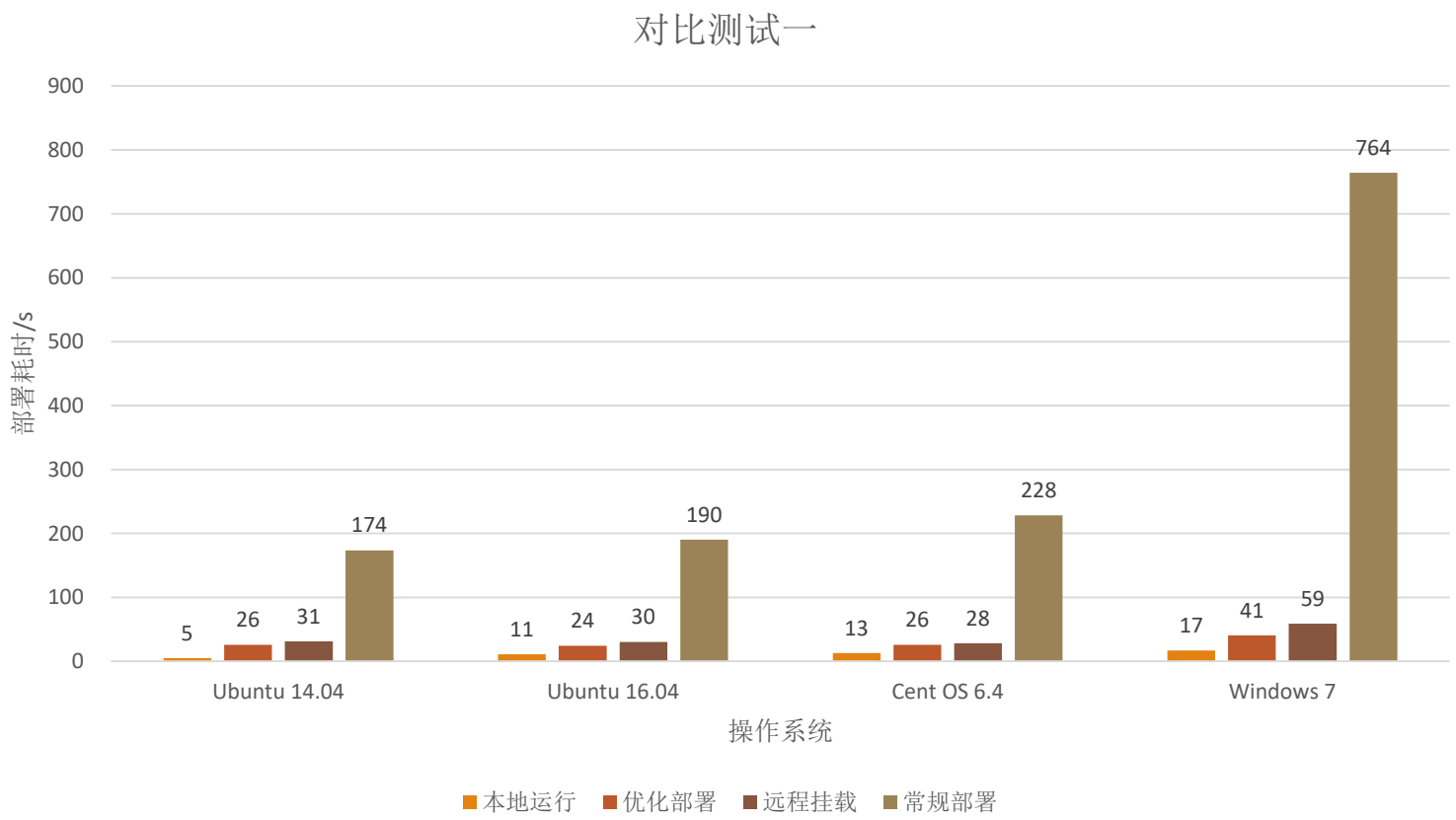

操作系统引导读取数据量
25000
20000
15000
10000
5000
0
大小/MB
79
142
1658
5210
78
154
1759
5210
81
153
2133
5210
153
289
7456
20480
Ubuntu 14.04
Ubuntu 16.04
Cent OS 6.4
Windows 7
操作系统
读取数据量
读请求量
镜像实际大小
镜像逻辑大小

4.4 虚拟机镜像优化部署性能

为了测试虚拟机镜像优化部署的性能，我们将与不同运行方案进行对比测试：

运行方案		说明
对比方案	本地运行	镜像就在本地，直接使用Libvirt启动
	常规部署	将完整的镜像文件从远程下载到本地，再启动
	远程挂载	将远程镜像使用网络文件系统挂载到本地并启动（高I/O）
优化部署	单镜像节点	使用课题的优化系统进行启动
	双镜像节点	同优化部署，区别在于部分计算节点上已经缓存有部署镜像

单镜像服务器



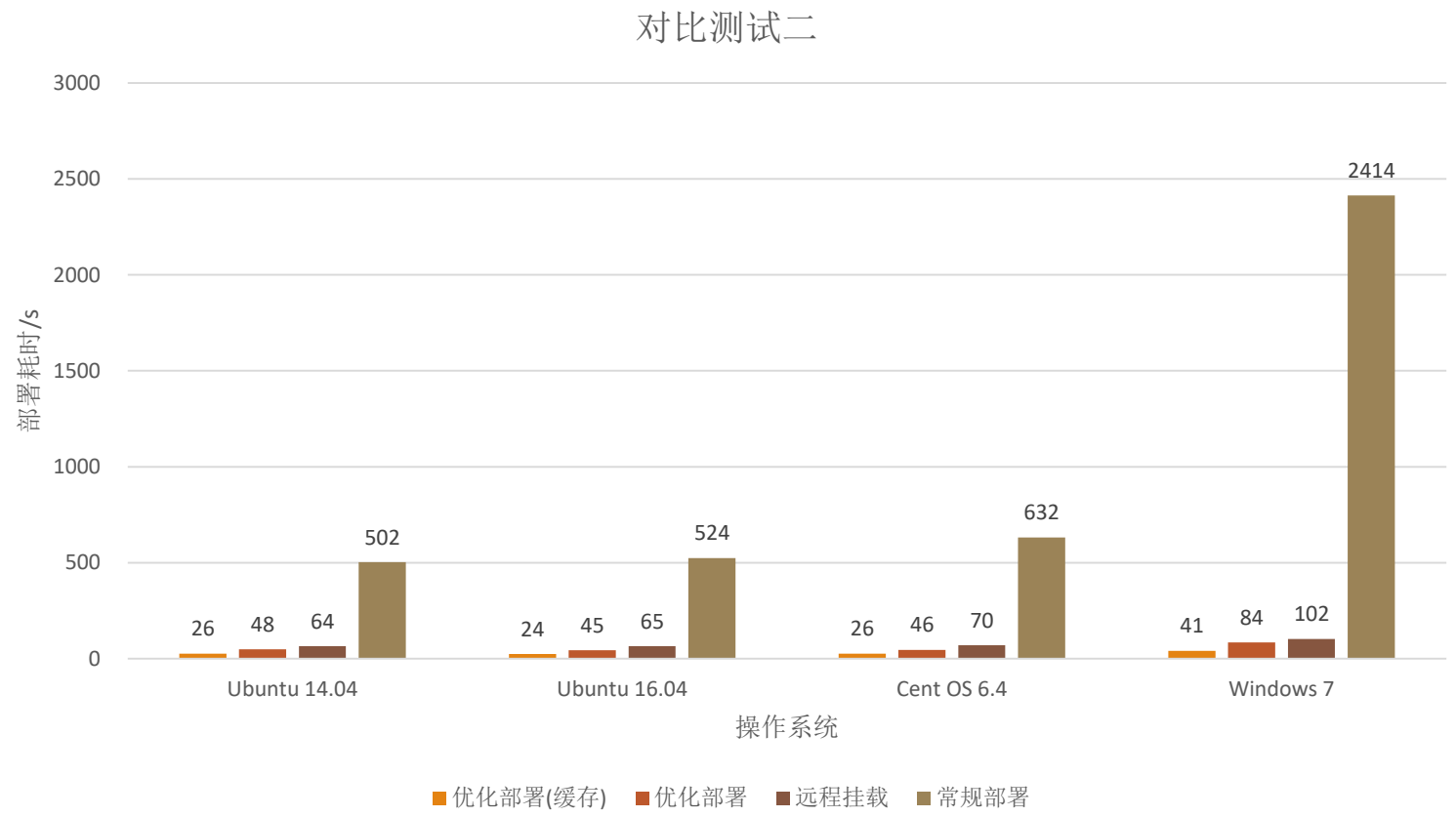
1个控制节点

1个计算节点

1个实例

控制节点缓存有镜像

双镜像服务器



1个控制节点

3个计算节点

1个实例

控制节点和1个控制节点缓存有镜像

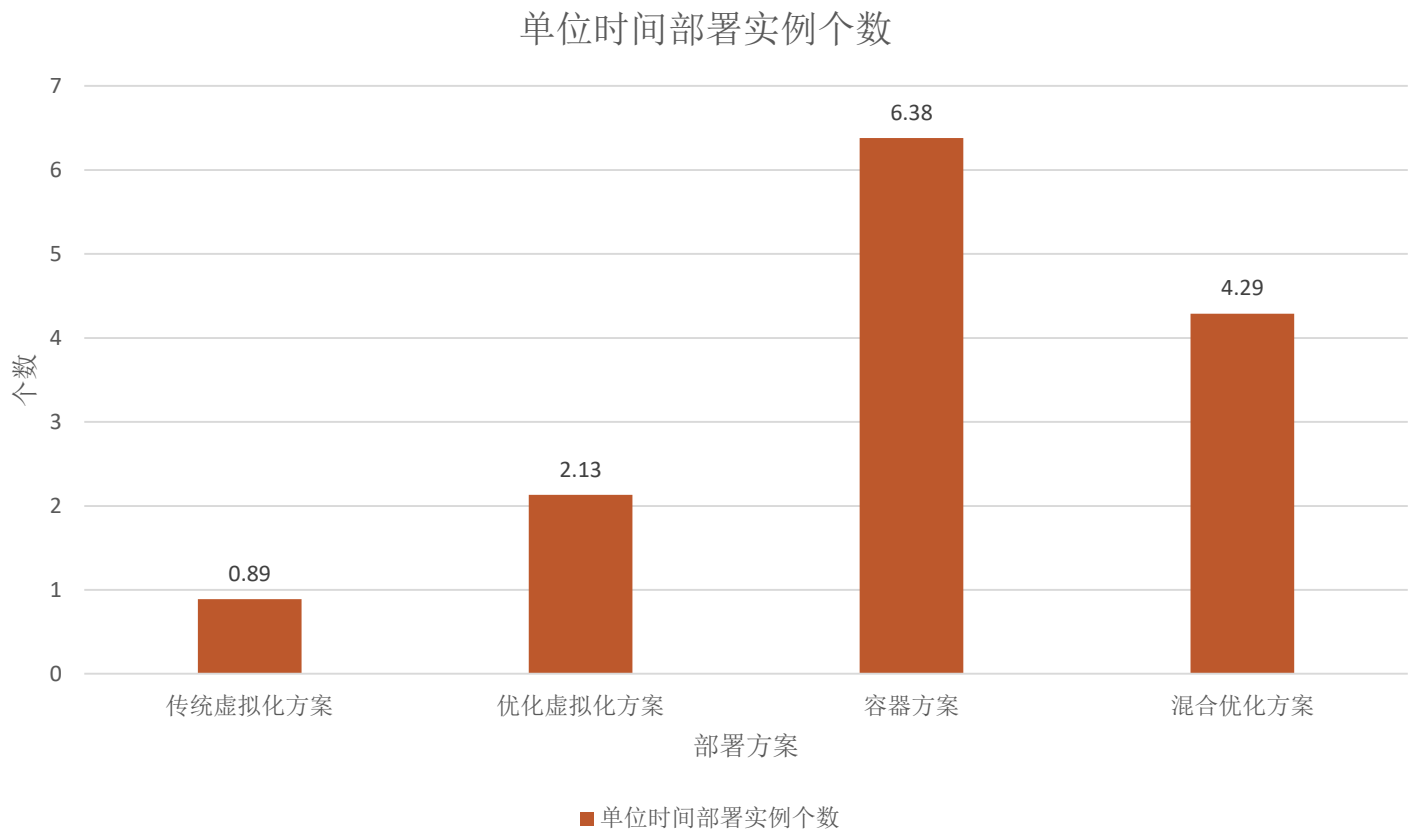
虚拟机容器混合部署系统性能

为了测试虚拟机容器混合部署系统的性能，我们将与不同用例进行对比测试：

#	实例个数	计算节点个数	部署配置
1	100	4	传统虚拟化方案
2	100	4	优化虚拟化方案
3	100	4	容器方案
4	20 + 80	4	混合优化方案（虚拟机:Docker = 2:8）

在每个计算节点上运行25个（一共100个）虚拟机实例，记录其启动耗时，计算出启动一台虚拟实例的单位耗时。

4.5 虚拟机容器混合部署系统性能



5 总结

1. 分析了操作系统引导过程对I/O请求的过程，并提取引导必需数据

2. 设计了基于网络流的虚拟机优化部署策略

3. 实现了一个虚拟机容器混合部署框架

论文审查意见

1. 图2-2中虚线建议修改

2. 国外参考文献较少，近几年的参考文献很少，1.2.4没有一篇参考文献，建议增加。

3. 第4章对实验的分析不够充分。

4. 某些流程图不规范，如图3-5没有结束框。

5. 存在一些格式和文字错误，如：(1)中文关键词应用分号分隔；(2)1.1.1小节的第三段，“起规模可以做的很大”不通顺；(3) 4.4小节的第一段结尾处，不应出现“在4.2.5中”。

Q&A