

**企业实习初期报告**

**信息与软件工程学院**

学 号： 2014220301019

姓 名： 卢杰

专业方向： 嵌入式系统

企业名称： 英特尔亚太研发有限公司

实习岗位名称： 虚拟化开发工程师

企业指导教师： 王震宇

院内代管教师： 廖勇

目 录

[1. 企业实习目标任务 2](#_Toc479114665)

[1.1 实习单位情况和岗位职责 2](#_Toc479114666)

[1.1.1单位情况 2](#_Toc479114667)

[1.1.2 岗位职责 2](#_Toc479114668)

[1.2 实习目标、任务与复杂工程问题 2](#_Toc479114669)

[1.2.1 实习目标 2](#_Toc479114670)

[1.2.2 实习任务 2](#_Toc479114671)

[1.2.3 复杂问题分析 2](#_Toc479114672)

[2. 实施方案和可行性研究 4](#_Toc479114673)

[2.1 基于容器的虚拟化技术简介 4](#_Toc479114674)

[2.2 基于Docker Container的GPU虚拟化性能分析 5](#_Toc479114675)

[2.3 基于Docker Plugin的GPU资源分配设计 5](#_Toc479114676)

[2.3.1 Docker plugin 5](#_Toc479114677)

[2.3.2 Plugin发现 5](#_Toc479114678)

[2.3.3 Plugin生命周期 6](#_Toc479114679)

[2.3.4 Plugin激活 6](#_Toc479114680)

[2.4.5 Plugin API 设计 7](#_Toc479114681)

[3. 知识技能学习情况 11](#_Toc479114682)

[3.1 Git版本控制工具 11](#_Toc479114683)

[3.2 资源独立性 11](#_Toc479114684)

[3.3 安全独立性 11](#_Toc479114685)

[3.4 Docker 11](#_Toc479114686)

[4. 前期任务完成度与后续实施计划 13](#_Toc479114687)

[4.1 前期完成度 13](#_Toc479114688)

[4.2 后续实施计划 13](#_Toc479114689)

[参考文献 14](#_Toc479114690)

# 企业实习目标任务

## 1.1 实习单位情况和岗位职责

### 1.1.1单位情况

英特尔是计算创新领域的全球领先厂商，设计和构建关键技术，为全球的计算设备奠定基础。1968年，罗伯特•诺伊斯(Robert•Noyce)、戈登•摩尔(Gordon Moore)和安迪•格鲁夫(Andy•Grove)在硅谷共同创立了英特尔公司。超过40年的发展，英特尔公司在芯片创新、技术开发、产品与平台等领域奠定了全球领先的地位，并始终引领着相关行业的技术产品创新及产业与市场的发展。英特尔的愿景是：未来十年创新和扩展计算技术，连接世界上每一个人，让生活变得丰富多彩。英特尔为计算机工业提供关键元件，包括性能卓越的微处理器、芯片组、系统及软件等，这些产品是标准计算机架构的重要组成部分。

英特尔1985年在北京设立代表处。今天英特尔公司已经成为在华最大的外国投资企业之一，员工人数超过7,500人。中国也是英特尔在美国本土以外部署最全面的国家，包括了芯片制造、封装测试，新技术研究、产品开发及营销——英特尔大连芯片厂是英特尔亚洲首家晶圆制造厂，英特尔成都工厂是世界一流的芯片封装测试厂，位于北京的英特尔中国研究院专注于前瞻性嵌入式系统研究，位于上海的英特尔亚太研发中心则是英特尔在中国的研发基地和创新中心。

其中英特尔亚太区研发有限公司于 2005 年 9 月在上海市紫竹科学园区正式成立——作为一个职能完备的研发机构，它兼具先进产品的开发能力和市场推广能力，将为中国及全球提供创新产品，为客户提供全面支持。英特尔亚太区研发有限公司主要从事高科技信息和通讯领域内的研究开发，同时参与中试和研发成果的转让及为英特尔产品提供相应技术服务。致力于推进围绕所有英特尔平台事业部的主要产品与技术的开发与创新。这些部门包括软件服务部、新设备部。

### 1.1.2 岗位职责

利用英特尔硬件平台，搭建虚拟化软件环境，与英特尔软件工程师一起从事Linux内核，设备驱动开发以及调试工作。

## 1.2 实习目标、任务与复杂工程问题

## 1.2.1 实习目标

整个实习阶段最终目标为GPU Container POC，完成Plugin，能够将Host的GPU resource 根据一定策略分配给多个Container使用。

## 1.2.2 实习任务

1. 了解XenGT/KVMGT project technology
2. GVT-g Upstream的编译安装测试
3. MediaSDK performance in Docker Container (Docker)
4. MediaSDK performance in Clear Container (intel)
5. Docker Plugin to assign GPU resource to Container

## 1.2.3 复杂问题分析

操作系统领域一直以来面临的一个主要挑战来自于应用程序间存在的相互独立性和资源互操作性之间的矛盾，即每一个应用程序都希望运行在一个相对独立的系统环境下，不受到其他程序的干扰，同时又能以方便快捷的方式与其他程序交换和共享系统资源。当前面向个人计算机的通用操作系统更强调程序间的互操作性，而缺乏对程序间相互独立性的有效支持。

Vmware和Xen等虚拟化主流产品均采用Hypervisor模型，该模型通过将应用程序运行在多个不同虚拟机内，实现对上层应用程序的隔离。但是由于Hypervisor模型更倾向于每个虚拟机都拥有一份相对独立的系统资源，提供相对完全的独立性支持，这种策略造成处于不同虚拟机内的应用程序间实现互操作非常困难。

Hypervisor模型这种强独立性保障策略在一定程度上牺牲系统的执行效率。对于高性能计算，Web服务，数据库，游戏平台和分布式系统等许多应用领域，提供高效的资源互操作性同保持程序间的独立性具有同等重要的意义。因此，针对这样的需求，提出了一种基于资源和安全容器概念的虚拟化技术。

对于GPU这样一类特定资源的分配，就需要加入对容器虚拟化的支持。在intel公司有对应的Clear Container技术，外部有当前流行的Docker Container技术，结合Docker的Plugin技术来思考如何添加对于GPU resource的管理。在实现之前需要分析多个Container对于GPU resource的争夺对于各自的性能的影响。在分析基础上可以先从实现最基本assign一些资源给Container，再考虑添加一些调度策略进行GPU resource调度。此外，未来还需要添加一些接口方便供使用者调用。

# 2. 实施方案和可行性研究

## 2.1 基于容器的虚拟化技术简介

容器技术主要涉及两个方面：资源容器和安全容器。

1. 资源容器

基于容器的虚拟化技术通过借鉴资源容器的核心思想，将系统中的资源主体从运行主体（即虚拟机）中剥离出来，从而达到在虚拟化技术上对系统资源进行精确和高效控制的目的。虚拟机监控器负责对系统中所有的资源容器进行管理和控制，根据用户配置以及系统资源使用的实际情况进行合理的分配和回收，实现跨虚拟机的进程间资源共享。

1. 安全容器

基于容器的虚拟化技术借鉴了安全容器的思想，在使用资源容器实现资源共享的基础上通过安全容器技术实现对共享资源的有效访问控制。按照不同的安全对象，诸如域，应用程序和虚拟机等分配不同的可访问资源形成虚拟的安全容器，防止其他对象对其资源进行恶意未授权的访问。

**系统架构如下：**

**/proc**

**/home**

**/usr**

**/dev**

**/proc**

**/home**

**/usr**

**/dev**

**VM1**

**VM宿主机**

**VM管理**

**远程服务**

**核心服务**

**Apache**

**MySQL**

**PHP**

**Quake Svr**

**Postgres**

**宿**

**主**

**平**

**台**

**宿**

**主**

**平**

**台**

**VM2**

**/proc**

**/home**

**/usr**

**/dev**

**共享OS镜像**

图2.1 基于容器的系统架构

## 2.2 基于Docker Container的GPU虚拟化性能分析

Docker是Docker公司开源的一个基于轻量级虚拟化技术的容器引擎项目,整个项目基于Go语言开发，并遵从Apache 2.0协议。目前，Docker可以在容器内部快速自动化部署应用，并可以通过内核虚拟化技术（namespaces及cgroups等）来提供容器的资源隔离与安全保障等。由于Docker通过操作系统层的虚拟化实现隔离，所以Docker容器在运行时，不需要类似虚拟机（VM）额外的操作系统开销，提高资源利用率，并且提升诸如IO等方面的性能。

在设计GPU Plugin之前需要测试多个Docker Container同时使用GPU resource对于GPU resource的影响，具体工作如下：

1. 准备宿主机kernel；
2. 安装Docker；
3. 创建Container；
4. 安装MediaSDK在Container里；
5. Performance数据；

通过测试数据分析，可以发现当多个Container开始使用GPU resource时，转解码时间开销变化不是非常大。

## 2.3 基于Docker Plugin的GPU资源分配设计

### 2.3.1 Docker plugin

Docker Plugin是 docker 提供出来的扩展机制，Plugin是一个独立的进程和 docker daemon 运行在同一台 host 上，通过 Plugin Discovery 的机制进行Plugin发现，Plugin有几个要求：

* Plugin名要求是小写；
* Plugin可运行在容器内也可以运行在容器外，现阶段建议运行在容器外。

### 2.3.2 Plugin发现

Plugin发现机制需要Plugin将自己的地址文件放在固定目录，方便 docker 发现Plugin进程，有三种文件可以设置：

* .sock 文件是 UNIX domain sockets
* .spec 文本文件内包含了一个 URL，比如： unix:///other.sock
* .json 文本文件包含了Plugin的完整 JSON 描述

UNIX domain socket 文件必须放在 /run/docker/plugins 目录，但是spec，json 文件则可以放在 /etc/docker/plugins 或者 /usr/lib/docker/plugins 中。

无后缀的文件名决定了Plugin的名字，比如 /run/d。ocker/plugins/myplugin.sock 的Plugin名就是 myplugin 。你可以在子目录中放置地址文件，比如 /run/docker/plugins/myplugin/myplugin.sock 。

docker 优先搜索 /run/docker/plugins 目录，如果没有 unix socket 的话才会去搜索 /etc/docker/plugins 和 /usr/lib/docker/plugins ，如果根据指定Plugin名搜到了Plugin就会立马停止搜索。

JSON 格式文件示例：

|  |
| --- |
| {  "Name": "plugin-example",  "Addr": "https://example.com/docker/plugin",  "TLSConfig": {  "InsecureSkipVerify": false,  "CAFile": "/usr/shared/docker/certs/example-ca.pem",  "CertFile": "/usr/shared/docker/certs/example-cert.pem",  "KeyFile": "/usr/shared/docker/certs/example-key.pem",  } } |

### 2.3.3 Plugin生命周期

* 启动Plugin
* 启动 docker
* 停止 docker
* 停止Plugin

### 2.3.4 Plugin激活

运行命令 docker run --gpu-driver=foo 即可以激活名为 foo 的Plugin，需要注意的是，Plugin是按需加载机制，只有被使用到了才会被激活。

### 2.4.5 Plugin API 设计

Plugin是 API 是基于 HTTP 的 JSON POST 请求，所以Plugin需要实现一个 HTTP 服务器并且将其 bind 到一个UNIX socket上。API 的版本设置在了 HTTP头里面，现在这个头的固定值为：application/vnd.docker.plugins.v1+json

**/Plugin.Activate :**

|  |
| --- |
| 请求：空  响应：  {  "Implements:" ["GPUDriver"] } |

返回Plugin实现，表示是 GPU Plugin

**/GPUDriver.Create ：**

|  |
| --- |
| 请求：  {  "Name": "GPU\_name" }  响应：  {  "Err": null } |

告诉Plugin用户想要创建一个GPU device，并将用户输入的GPU device名传给Plugin。Plugin在这个时候可以不用理会这个请求，会有真正挂载的请求。如果出错返回错误字符串。

**/GPUDriver.Remove ：**(与 Create 相对应)

|  |
| --- |
| 请求：  {  "Name": "GPU\_name" }  响应：  {  "Err": null } |

**/GPUDriver.Mount ：**

|  |
| --- |
| 请求：  {  "Name": "GPU\_name" }  响应：  {  "Mountpoint": "/path/to/directory/on/host",  "Err": null } |

用户请求挂载某个文件，这个请求仅会在容器启动时发送一次。

将 GPU\_name 挂载的真正挂载点返回给 docker，如果出错则返回错误字符串。

**/GPUDriver.Path ：**

|  |
| --- |
| 请求：  {  "Name": "GPU\_name" }  响应：  {  "Mountpoint": "/path/to/directory/on/host",  "Err": null } |

Plugin需要管理 GPU\_name 的真正挂载地址，这个请求需要将 GPU\_name 挂载的真正挂载点返回给 docker，如果出错则返回错误字符串。

**/GPUDriver.Unmount ：**

|  |
| --- |
| 请求：  {  "Name": "GPU\_name" }  响应：  {  "Err": null } |

表示 docker 已经不需要这个 GPU device 了，Plugin需要安全的将这个挂载从挂载点卸载。

**Docker 系统架构图**

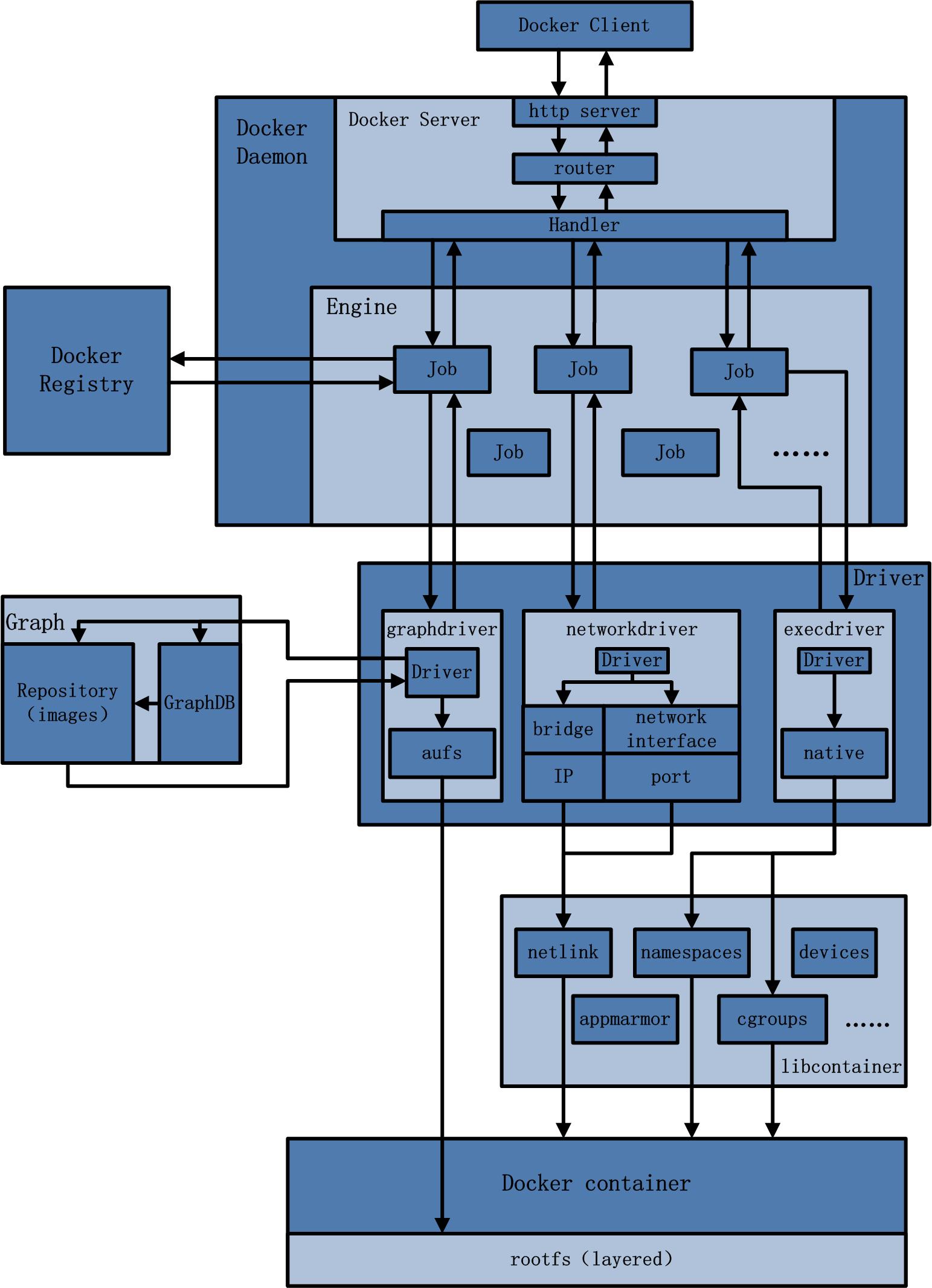


图2.3 Docker系统框架图

用户是使用Docker Client与Docker Daemon建立通信，并发送请求给后者。Docker Daemon作为Docker架构中的主体部分，首先提供Server的功能使其可以接受Docker Client的请求；而后Engine执行Docker内部的一系列工作，每一项工作都是以一个Job的形式的存在。Job的运行过程中，当需要容器镜像时，则从Docker Registry中下载镜像，并通过镜像管理驱动graphdriver将下载镜像以Graph的形式存储；当需要为Docker创建网络环境时，通过网络管理驱动networkdriver创建并配置Docker容器网络环境；当需要限制Docker容器运行资源或执行用户指令等操作时，则通过execdriver来完成。

libcontainer是一项独立的容器管理包，networkdriver以及execdriver都是通过libcontainer来实现具体对容器进行的操作。

当执行完运行容器的命令后，一个实际的Docker容器就处于运行状态，该容器拥有独立的文件系统，独立并且安全的运行环境等。

# 知识技能学习情况

## 3.1 Git版本控制工具

Git是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。在实习中，在公司服务器上有专门代码仓库用于存放项目代码和进行对项目的版本控制。在平常测试开发中经常需要从代码仓库clone或者pull最新的branch来进行开发。

## 3.2 资源独立性

资源的独立性主要是由CPU分配，I/O分配和存储分配三部分组成。无论哪种资源分配，公平性是首先要考虑到的，这是资源独立性的核心。此外，如何高效利用资源以免浪费也是一个需要考虑的因素。

## 3.3 安全独立性

基于容器的系统安全独立性主要由以下几个部分组成：进程空间的独立性，网络的独立性，文件系统的独立性和限制容载能力。不同部分虽各有差别，实现取舍也各不相同，但是都是通过对资源访问的严格控制达到了针对不同方面实现安全独立性的目的。

## 3.4 Docker

用户是使用Docker Client与Docker Daemon建立通信，并发送请求给后者。

而Docker Daemon作为Docker架构中的主体部分，首先提供Server的功能使其可以接受Docker Client的请求；而后Engine执行Docker内部的一系列工作，每一项工作都是以一个Job的形式的存在。

Job的运行过程中，当需要容器镜像时，则从Docker Registry中下载镜像，并通过镜像管理驱动graphdriver将下载镜像以Graph的形式存储；当需要为Docker创建网络环境时，通过网络管理驱动networkdriver创建并配置Docker容器网络环境；当需要限制Docker容器运行资源或执行用户指令等操作时，则通过execdriver来完成。

而libcontainer是一项独立的容器管理包，networkdriver以及execdriver都是通过libcontainer来实现具体对容器进行的操作。

当执行完运行容器的命令后，一个实际的Docker容器就处于运行状态，该容器拥有独立的文件系统，独立并且安全的运行环境等。

# 4. 前期任务完成度与后续实施计划

## 4.1 前期完成度

在实习的前两个月里，了解了关于XenGT/KVMGT的项目情况，完成了其环境的搭建过程；之后开始接触Container方面的内容，即创建了intel自己开发的Clear Container容器，又研究了MediaSDK在多个Docker Container中的性能测试，并开始着手将linux 4.4的补丁移植到最新的4.10或4.11 kernel上；了解并记录了Docker Plugin的实现机制，以及通信过程。

我觉得前两个月很积极和充实的参与到了实习过程中，学到了很多未曾接触到的知识，得到了许多linux 内核方面宝贵的经验。此外，近距离接触到了一批参与linux内核开发的工程师，了解了整个linux内核代码开发贡献的流程。

## 4.2 后续实施计划

后续需要在公司第二季度完成Docker GPU Plugin并能够完成基本调度分配任务；还有创建公司的Clear Container的guest，并测试MediaSDK在其上的性能；有可能在6月份需要参与提供一个demo供公司参加展示活动。

接下来的时间打算加强在linux 内核方面的，特别是driver方面的深入了解；加强对Docker Plugin的理解和实现；能够对Clear Container和Docker Container这两类container有个清晰的理解和认识。

# 参考文献

1. Kun Tian,Yaozu Dong,David Cowperthwaite,A Full Virtualization Solution with Mediated Pass-Through[J],Usenix Conference on Usenix Technical Conference,2014:121-132
2. 系统虚拟化：原理与实现/英特尔开源软件技术中心，复旦大学并行处理研究所著.—北京：清华大学出版社，2009.3