对每个项目需要记录的内容

1.项目简介：介绍项目内容、目的、效果（尽量通俗）

2.关键词：涉及领域、知识、编程语言

3.难度评价：维度包括需要的知识广度、复杂性

4.其他

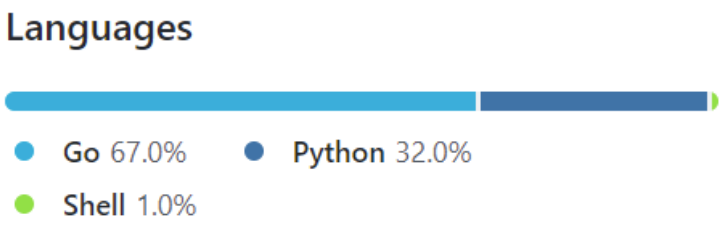
在调研的最后需要写下自己感兴趣的领域、想研究的课题与设计思路

DDL：3.6中午12:00

KATA-Unikernel

关键词：虚拟化、**Unikernel**

涉及语言：**go**、**python**



项目简介

利用`Nanos`、`KVM`和`libvirt`实现了虚拟机对`Unikernel`的封装，并对其资源占用和运行速度作了进一步优化。

项目的难点主要在于Unikernel的迁移，以及没有统一的方式来方便地定义虚拟机相关的各种可管理对象。前者通过使用可移植性良好的`Nanos`解决，后者利用`libvirt`提供的工具解决。

原文描述如下：

> 本次大作业致力于实现更为便捷轻量的*Unikernel*管理应用，并在这个过程中学习理解工业级代码，最终收获一个简易的虚拟机和*Unikernel*结合的实验性产品。

名词解释

`KVM`：基于内核的虚拟机（英语：Kernel-based Virtual Machine，缩写为`KVM`）是一种用于Linux内核中的虚拟化基础设施，可将Linux内核转化为一个虚拟机监视器。

`Nanos`：Unikernel的一种，`Nanos`的最大特点是，可以覆盖到主流的Python，PHP，C++，Golang以及Rust等多种语言环境，使其通用性得到进一步扩展。

`libvirt`：一个管理虚拟化平台的工具包。目前，`libvirt` 已经成为使用最为广泛的对各种虚拟机进行管理的工具和应用程序接口（API）。

`virt-viewer`：一个用于显示虚拟机的图形控制台的最小工具。

大致步骤

1. 通过利用`ops`（`Nanos unikernel` 的编译和编排工具）并分离`ops`中的`build`模块，构建`img`文件，以及对虚拟机进行硬件加速
2. 封装`libvirt` API（用于创建和管理虚拟机，涉及python和xml）
3. 实现虚拟机的可视化（利用`virt-viewer`）
4. 测试性能（测试该虚拟机运行某些算法消耗的时间与空间资源，并与`linux`树莓派作对比）

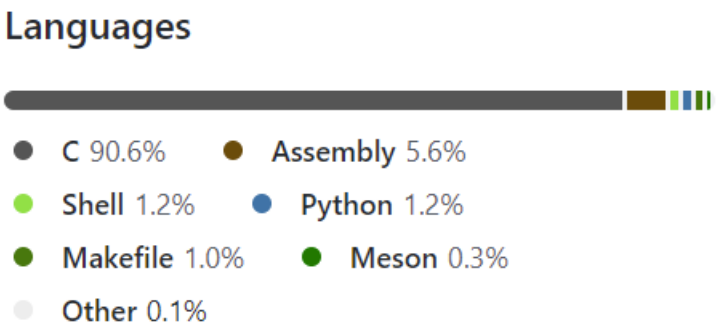
总结

该项目对虚拟化技术有相当深的探索，在调研时难度较大，需要查阅大量资料，以了解常见的轻量虚拟机、容器等的优缺点，寻找合适的`Unikernel`、易于适配的`KVM`和诸如`virt viewer`等工具，同时go语言的使用也加大了项目的上手难度，总体感觉比另外两个都要复杂。

NBFS (NVMe Better File System)

关键词：文件系统

语言：**C**



项目简介

该项目旨在设计和实现一个针对改进`NVMe SSD`读写性能的文件系统，并基于`SPDK`工具实现了该文件系统，同时提供了两套用于访问底层设备的接口，一套是用于`FUSE`挂载的通用接口，另一套是用于专用应用的专用接口。

原文对于两套接口的阐述如下：

> 既保证文件系统的通用性又要追求极致的性能，这在短时间内是难以完成的，所以我们选择实现两套接口：
>
> - 通用接口：以通用性为第一目标，可对接`SPDK`提供的`FUSE`插件，进而可被正常软件调用，在此基础上，利用`SPDK`的特性提高其性能。
> - 专用接口：以性能提升为第一目标，直接利用`SPDK`平台搭建一套异步、无锁、并发的文件系统，实现我们所设想的性能提升。

名词解释

`NVMe`：`NVM Express(NVMe)`，是一个逻辑设备接口规范。此规范主要是为基于闪存的存储设备提供一个低延时、内部并发化的原生界面规范，也为现代CPU、电脑平台及相关应用提供原生存储并发化的支持，令主机硬件和软件可以充分利用固态存储设备的并行化存储能力。

`FUSE`：用户空间文件系统（`Filesystem in Userspace`，简称`FUSE`）是一个面向类`Unix`计算机操作系统的软件接口，它使无特权的用户能够无需编辑内核代码而创建自己的文件系统。

`SPDK`：针对高性能`NVMe SSD`设备的存储性能开发工具包。`SPDK`（The Storage Performance Development Kit），提供了一套工具和库，用于编写高性能、可扩展的用户模式存储应用程序。其底层是一个用户空间、`polled`模式、异步、无锁的`NVMe`驱动。

`BlobStote`：`Blobstore`是`SPDK`中的一个模块，它实现对`Blob`的管理，包括`Blob`的分配、删除、读取、写入、元数据的管理等。在`Blobstore`下层，与`SPDK bdev`层对接。`SPDK bdev`层类似于内核中的通用块设备层，是对底层不同类型设备的统一抽象管理。

大致步骤

1. 利用`B+`树和`SPDK`提供的用户态框架设计通用接口，减少了传统文件系统多层切换，与此同时，由于整套系统处于用户态（利用轮询代替中断），减少了用户态与内核态的切换开销。
2. 通过`SPDK`的`BlobStore`包设计专用接口。
3. 进行通用接口挂载`FUSE`测试，以及专用接口性能测试。

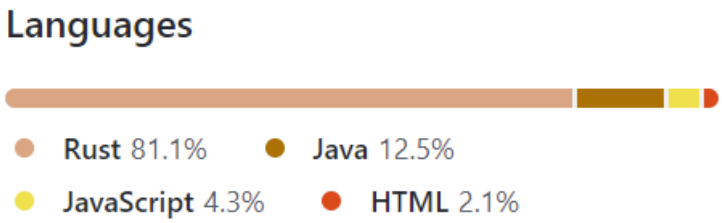
总结

该项目易于上手的点在于利用C语言编写，但是要求对文件系统的结构及其运作有相当深度的理解，在寻找合适的开发工具时也会遇到不小的困难。

基于 **Rust** 和 **WebAssembly** 的分布式文件系统

关键词：网络文件系统、分布式文件系统

语言：**Rust、Java、JS**



简介

该项目以2017年OSH“基于互联网的小型分布式文件系统”项目为基础，利用rust语言对后端中的服务端和客户端进行改写（原基于Java），并对纠删码部分应用`webAssembly`，从而降低了系统运行时的内存消耗；前端基本沿用了17年项目的Java代码。

原项目中描述如下：

> 在原基于互联网的小型分布式文件系统项目的基础上，小组成员齐心协力，学习*rust*语言并完成了*client*、*server*端的完全改写，实现了整个分布式文件系统的逻辑架构。此外，采用前后端分离的模式，采用*actix-web*框架将*web*应用的后端用*rust*语言改写，并与原项目的*web*前端衔接，最终呈现出完整的分布式文件系统，并在浏览器端提供了用户对文件系统的管理操作。

项目目标：

- 高效利用分散在不同设备上的存储空间，避免资源浪费
- 所有安装了客户端的机器共享、贡献存储空间和存储文件碎片，并对数据做好高效的备份；一个专门用于协调处理请求，维持系统状态的服务端；
- 有浏览器的电脑都可访问分布式文件系统
- 在以上要求外，尽可能让性能相对好，安全性相对高，操作简单

名词解释

`分布式文件系统(DFS)`：分布式文件系统（Distributed File System，DFS）是指文件系统管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点（可简单的理解为一台计算机）相连；或是若干不同的逻辑磁盘分区或卷标组合在一起而形成的完整的有层次的文件系统。

`WebAssembly`：一种可以使用非 JavaScript 编程语言编写代码并且能在浏览器上运行的技术方案。

`Actix-Web`：用于Rust的功能强大、使用且快速的Web框架。

`纠删码`：一种编码技术，相比于副本策略，纠删码具有更高的磁盘利用率。

`crate`：Rust 中，`crate` 是一个独立的可编译单元。具体说来，就是一个或一批文件（如果是一批文件，那么有一个文件是这个 crate 的入口）。它编译后，会对应着生成一个可执行文件或一个库。

大致步骤

1. 学习前后端相关知识（rust、相关`crate`文档、`MySQL`、`WebAssembly`、JavaScript、rust相关的Web框架-`Actix-Web`等）
2. 对17年项目后端的client、server和web模块进行改写
3. 分析比较系统性能

总结

该项目以往届大作业为基础，因此上手更加容易。然而，做网络文件分布系统需要系统学习Web开发的架构和前后端的具体知识（如JS、HTML、MySQL、Rust及其相关框架），存在一定的挑战。不过该项目在总结文件的末尾附上了他们开发过程中使用到的工具教程链接，为从事相关的开发提供了一定便捷途径。

ps: 该项目在OSH大作业的ddl前只完成了对后端的改写，用rust重新编写前端一直到大作业结束后才完成。

感兴趣的方向

或许可以像KATA-Unikernel一样往虚拟化、云计算的大方向上靠一靠，感觉比文件系统的可能性和趣味性要多一些，也可以学到更多的知识。