从Borg到Kubernetes

PaaS产品设计探讨







https://github.com/ HardySimpson









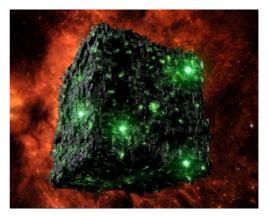


Borg 解决了什么问题?



What is Borg?





—We are the Borg. Lower your shields and surrender your ships. We will add your biological and technological distinctiveness to our own. Your culture will adapt to service us. Resistance is futile.





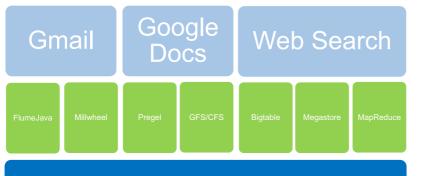


The cluster management system we internally call Borg

admits, schedules, starts, restarts, and monitors

the full range

of applications that Google runs.



Borg





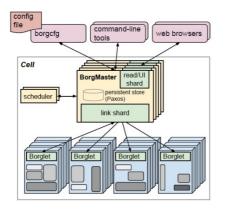


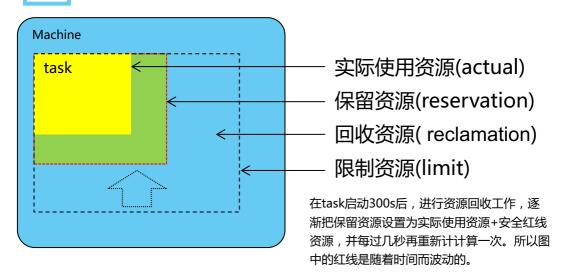
Figure 1: The high-level architecture of Borg. Only a tiny fraction of the thousands of worker nodes are shown.

- 用户使用Borgcfg或者Web UI提交需要 跑的应用(Task):例如一个跑100个 副本的web服务,或一个批处理任务
- Borgmaster接受这个请求,放入队列内
- Scheduler扫描队列,查看这个应用的资源需求,在集群中寻找匹配的机器
- Borgmaster通知Borglet,在相应机器 上启动应用

提交应用 → 应用启动 25秒



Schedule Policy





Schedule Policy

prod task

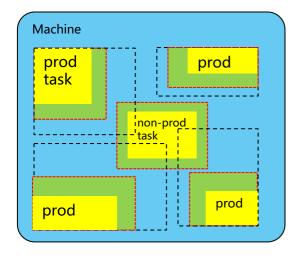
- 永不停止,面向用户 (Gmail , Google Search , Google Docs)
- 几微秒到几百毫秒
- 短期性能波动敏感

non-prod task

- 批处理任务,不面向用户(Map Reduce)
- 几秒到几天
- 短期性能波动不敏感



Schedule Policy



- 使用限制资源计算prod task的可用性
- prod task可以抢占non-prod task的资源,从而导致nonprod task被杀死而重调度
- prod task不能互相抢占资源而 驱逐对方

20%的工作负载跑在回收资源上

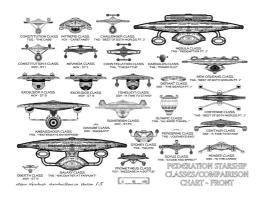




- 即使Borgmaster或Borglet挂了, task继续运行
- 99.99%可用性
- 10k 机器/Cell
- 10k task/分钟
- 99% UI < 1s
- 95% borglet poll < 10s



Utilization == Money



- 如何去定义一个异构集群的效率 ?
- 把多个用户、prod和non-prod 的task混合会提升还是降低效率?
- 资源回收和调度策略怎么样才是最佳的?
- 如何划分资源粒度?
- Cell是越大越好吗?



Utilization



- 压缩率,给定一个负载,部署到可以运行这个负载的最小Cell里面去
- prod和non-prod task混合运行,会降低3%-20%的CPU速度,但会节省20%-50%的机器
- task请求的资源粒度小(0.001核,byte计数内存)能 提升压缩率
- 详见论文《Google使用Borg进行大规模集群的管理》





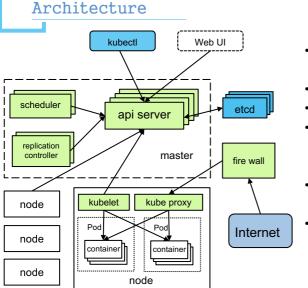
- 隐藏资源管理和故障处理细节,使用户可以 专注于应用开发
- 本身提供高可靠性和高可用性的操作,并支持应用程序做到高可靠高可用
- 在数以万计的机器上高资源利用率运行





Kubernetes的 不同之处和方向





- 用户通过kubectl提交需要运行的docker container(pod)
- api server把请求存储在etcd里面
- scheduler扫描,分配机器
- kubelet找到自己需要跑的container, 在本机上运行
- 用户提交RC描述, replication controller监视集群中的容器并保持数量
- 用户提交service描述文件,由kube proxy负责具体的工作流量转发



Difference

- 在Google的数以百万计的集群 上运行超过十年
- 使用lxc容器
- C++编写
- 对集群调度性能要求非常苛刻
- 单集群能调度超过上万台机器

- · 2014.7开始有提交记录,发展 较快
- 使用Docker容器
- Go语言编写
- 目前还没有做很多性能优化
- 目前单集群只支持几百台机器

Borg





- 静态编译,包括可执行程序和数据文件
- 接受SIGTERM信号,用于清理保存状态
- 被kill之后能够在其他机器上重启,无状态
- · 一般内置http服务,用于获取健康信息
- 数据和日志一般都存储在分布式存储上

应用在设计期就是分布式的

- Docker容器, 自带干粮
- 支持挂载外部的各种持久层 (GCEPersistentDisk, AWSElasticBlockStore, NFS, iSCSI.....)
- · 从容器中读取监控信息,从多个层面检 查应用性能
- 支持在Pod中包含日志处理容器

假定容器能在其他机器上重启,但 实际上还需应用做一定改造

Borg





- 静态编译,包括可执行程序和数据文件
- 接受SIGTERM信号,用于清理保存状态
- 被kill之后能够在其他机器上重启,无状态
- · 一般内置http服务,用于获取健康信息
- 数据和日志一般都存储在分布式存储上

应用在设计期就是分布式的

- Docker容器, 自带干粮
- 支持挂载外部的各种持久层 (GCEPersistentDisk, AWSElasticBlockStore, NFS, iSCSI.....)
- · 从容器中读取监控信息,从多个层面检 查应用性能
- 支持在Pod中包含日志处理容器

假定容器能在其他机器上重启,但 实际上还需应用做一定改造

Borg





- •机器/IP vs Pod/IP
 - 允许应用自由选择端口,不必考虑冲突
 - 保证外部的其他的服务发现/命名也不用操心端口
- Jobs分组 vs Selector/Label分组
 - 提供更灵活的组合搭配
- •对资深用户优化 vs 对初级用户友好
 - Borg有230个多参数



Kubernetes Special

- 插件化
 - rc, scheduler, persist volume...
 - 用户可以按照plugin接口自定义实现,扩展功能
- 容器化
 - api server, scheduler, controller, etcd, cadvisor, flannel...
 - kubelet不会被容器化
- 支持多种I层部署和适配
 - GCE, Vagrant, Microsoft Azure, CoreOS, vSphere, Amazon Web Service...
 - SaltStack部署支持









Borg

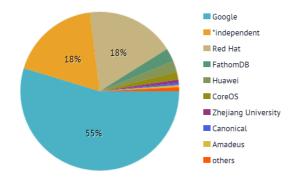
Future



- 多租户支持(namespace)
- 容器持久化
- 提升集群规模 , 100->1k , 模拟器
- 并行共享资源调度(Omega),提升利用率
- 容器网络层优化, proxy->内核
- 多集群管理
- 单集群跨I层调度



Relate Work @Huawei



- · CentOS baremetal的部署脚本
- Heapster standalone 文档和 bug提交
- Ubuntu k8s升级脚本和文档
- Heapster kafka sink
- Heapster elasticsearch sink
- Bugfix, cmd.....



PATR 3

未来的云需要什么?

Technology Lifecycle





- IBM PC
- Linux/Unix
- C/Java
- JavaScript
- PHP
- Windows
- Git

- Cobol
- Dephi
- EJB
- OS/360
- Multics
- DOS
- Minix
- Perl
- Microkernel

•





Kubernetes/Docker





Fail of Tower of Babel









三十幅共一毂, 当其无, 有车之用。 埏埴以为器, 当其无, 有器之用。 凿户牖以为室, 当其无, 有室之用。 故有之以为利,无之以为用。



The Long March



| Define | Design | Develop | Test | Deploy |
|--------|--------|---------|------|--------|
| | | | | |

- Engineering
- · Establish the business requirements
- Perform requirements study
- and analysis · Flicit user
- requirements
- Analyze requirements
- · Specify SRS · Verify and validate requirements
- Requirements Engineering Plan
- Business Requirements Specifications /Project Charter
- · Use Cases and Test Cases

- Application Architecture and Design
- · Define software
- architecture Demonstrate
- architecture
- · Design solutions
- Data Modeling
- Design logical data model
- · Design physical data model
- · Application Application Code Architecture Design · Unit Test Cases
- · Application Design Code Specifications · Physical Data Model Updated
- Logical Data Model Requirements Traceability

Coding and Unit coding and unit

testing

test

· Develop code

· Conduct unit

- Test Definition Testing
- · Define test · Prepare for approach

· Set-up test

environment

· Execute test

next level

Test Approach

Cases

. Test Plan and Test

Test Environment

Defect Tracking log

· Test Status Report

. Transit test to

- Establish impl. approach · Create test plan · Create impl. plan
- and support · Create test cases procedures Test Execution
 - · Manage impl.

Go Live

- · Provide Impl. support
- Transition · Create transition
- approach and supporting material
- Enable transition
- Implementation Plan · Command Center Governance Plan
- Support Manual
- · Transition Approach

Traditional IT

manage

Jon J

Virtualization

You manage

laaS

Virtualization Server HW

Storage

Networking

PaaS

Runtimes Security & Integration

Databases Servers

Virtualization

Server HW

Storage

Networking

è

SaaS Applications

Runtimes Security & Integration

> Databases Servers

Virtualization

ō

Server HW

Storage

Networking





- 产品是否能减少语言、程序、框架不同带来的 复杂性?
- 产品是否能减少设计、编码、测试、部署流程中带来的复杂性?
- 产品是否能减少大规模集群、网络、服务依赖 、错误追踪带来的复杂性?

Make Life Simpler!









下次再见



