
04 Jan 2017

# 虚拟化学习小组分享之——Borg论文解读

## 背景简介

组里目前支持了一个非常重要的业务，为了支持业务快速的扩展，更好的进行产品维护，决定成立一个虚拟化学习小组。主要宗旨就是为了提升大家对虚拟化、资源管理这方面知识的理解，共同调研生产环境下关于资源虚拟化方面的主流解决方案

## Borg论文解读

先贴上原文的地址：http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.p

这里先做一下论文的简要梳理，然后对比一下开源平台kubernetes

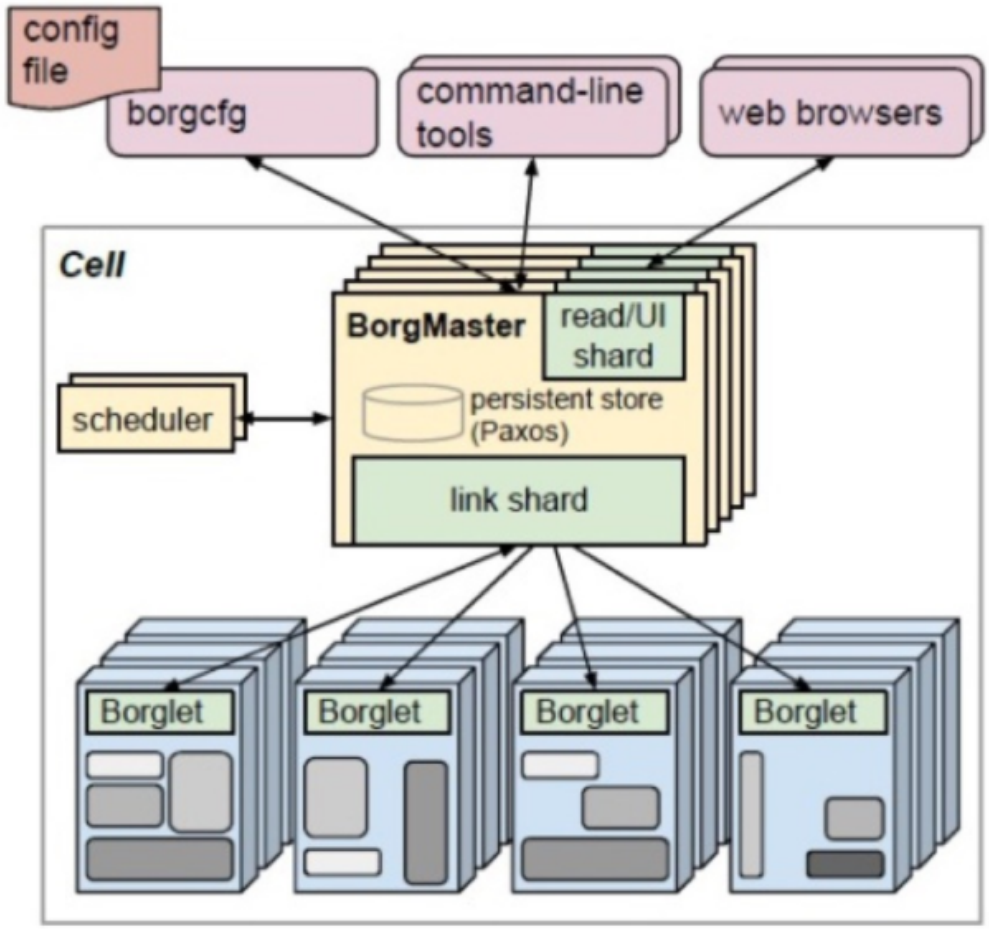
### Borg简介

Borg是什么呢？它是Google公司的大规模分布式的集群管理系统，负责管理、调度、运行和监控公司绝大部分的应用程序和框架包括Gmail、Google Docs、Web Search这样的应用程序，也包含一些底层的框架Map Reduce计算框架、GFS分布式的存储系统、Big Table。主要是解决了这么几个问题

- 隐藏了资源管理和故障处理细节，使其用户可以专注应用开发
- 支持应用程序做到高可用和高可靠
- 提升了资源利用率，有效的降低了成本

# 整体框架

整体来看，Borg是一个典型的分布式平台架构，每个节点部署一个代理（Borglet），统一由BorgMaster来管理这些节点的运行情况。同时对外部而言，Borg提供了命令行和Web UI的方式访问Borg系统，提交任务或者申请资源。这些请求会发送到BorgMaster那里，然后由BorgMaster来决定后续的操作，下面是Borg的基本架构图




一套由BorgMaster和Borglet部署的Borg环境称为cell，一个IDC可以部署多个cell。一般来说为了防止单点问题存在，每个cell都要由五个BorgMaster组成，它们通过paxos协议进行选主和同步，所有的写操作都要由leader来执行。BorgMaster要定期轮询所有Borglet节点的状态，如果机器规模很大的情况下就会造成负载过重，因此Borg系统将轮询工作平摊给每个BorgMaster的follower处理，只有Borglet的状态发生变化时，才会通知leader处理。

为了减少BorgMaster的压力、灵活的迭代调度器的模型，Borg系统融入了omega的设计，将调度器scheduler作为一个独立的服务拆分出来了~ 下面梳理一下调度器方面的细节

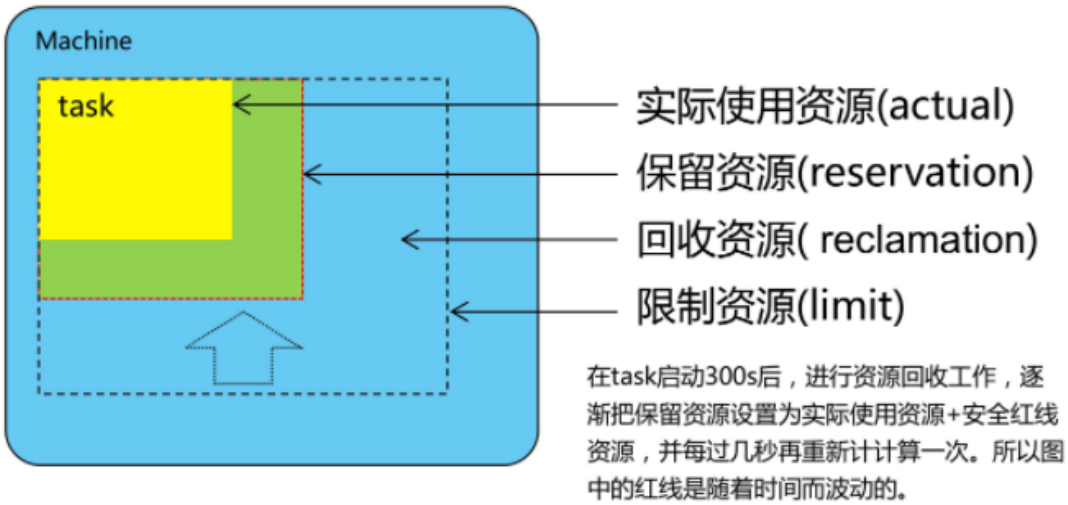
# Borg的调度器

决策调度的过程主要涉及到以下两个方面：**适用性**(feasibility checking)和**最适合**(scoring)

**适用性：**是指找到哪些机器能够负载（运行）这个任务
**最适合：**是指在这些适合的机器中，按照一定的打分策略选择分数最高的那个

## 适用性估量

Borg会对每个任务Task做了一个资源上的估量，下图是一张资源限制图的说明：




最外层的那个虚线是这个Task的配额，这是一个硬性的限制，如果超过了这个限制，这个Task是无法运行起来的。实际生产环境中，对一个Task实际需要多少资源是很难估量的，所以Borg提出了一种资源回收的解决方法，它在Task启动300秒之后，就会进行一个资源回收的工作，上图中黄色区域是程序实际使用的资源，然后Borg会从外围的硬限制资源区域慢慢向里推进，直到推到一个安全的区域，那么剩下的资源Borg可以重新利用起来。

绿色区域的那个范围是一个动态的过程，Borg会隔几秒就会重新计算一下应用程序耗费了多少资源。为了更合理的进行回收资源的过程，它对应用类型做了区分，分成生产型的应用（prod task）和非生产型的应用(non-prod task)，这些概念会在一下章节中提及。

## 最适合

根据Task的资源估量，来选出哪些机器可以负载这个Task，然后通过一些打分策略，选择分数最高的机器来运行这个Task。目前有2种打分模型一种是考虑负载均衡，将负载平均分配到所有机器上，这样做的代价是增加了碎片，对于那些需要大部分的机器的Task来说，是很难获取到足够的资源的，论文中也称为**最差匹配**策略。

还有一种相对应的分配策略称为**最佳匹配**：就是把机器塞的越满越好，这样能够一下一批空闲的机器给用户。这样的策略对那些资源预估不足的用户来说，是很不友好的，比如用户的Task负载突然升高将会大大降低Task的服务质量。

Borg论文里使用的是两种方式混合的，试图减少碎片资源。还有一个比较明显的特征是如果一些高优先级的任务没有找到合适的资源来运行服务时，就会试图驱除一些低优先级的任务来达到释放资源的目的。

启动延迟（Task从提交到真正运行起来）是一个比较关注的问题。Borg为了减少Task的启动时间，scheduler希望机器上已经有足够的包(程序和数据)：大部分包是只读的所以可以被分享和缓存。

## 任务类型

运行于Borg系统之上的应用主要分成两类，一类称为**prod task**：启动后作为一个服务存在，长时间不断接受并处理收到的请求。这类任务对请求处理延迟和可用性比较敏感，多数服务于终端用户（比如Gmail、Google Docs等）这些应用也称为**在线服务**。另一类应用称为non-prod task，这类任务执行结束后自行退出，对执行时间和执行失败不那么敏感，也可称这些服务为**离线作业**。

对任务类型进行区分有很多好处，因为这两种负载的差异性比较大，不能以相同的调度策略来对待。对任务类型进行区分一方面可以优化调度策略（prod任务可以通过抢占non-prod任务来提高服务的可用性），一方面可以针对任务类型来进行相应的优化（对prod任务来说需要关注的是可用性，对non-prod来说系统的整体吞吐才比较重要）

## 服务发现和命名机制

光是创建和部署task是不够的：一个服务的客户端和其他系统需要能找到它们。为了实现这一点，Borg内建了一个Borg Name Service的系统，这个系统将Task的主机名、端口、属性等信息写入到一个持久化高可用的文件里，以BNS为文件名放在Chubby中，通过访问这个文件的形式来发现Task的地址。

每个Task都会有一个内置的HTTP服务，用来健康检查和收集各项性能指标信息，并通过Dashboard来展示出来。为了防止某些Task运行中产生大量的日志塞满磁盘，Borg会自动的滚动这些日志信息，会在Task结束后保留一小段时间用来调试。

## 为了提升资源利用率，Borg的做法

Borg集群支持混布（离线作业和在线作业在一起调度）：为了实现这一点，Borg首次使用cgroup技术进行物理资源隔离。这个工作是后续众多容器技术的基础，早期的lxc以及后续的docker都得益于Google的贡献

Borg支持优先级调度，支持抢占、支持超发：Borg上运行的任务都需要指定具体的优先级，主要分成四种任务：基础业务（monitoring）、在线业务（production）、离线计算（batch）、best-effort。best-effort是在机器利用率低峰期跑一些任务，不会保证任务的可靠性，这样可以大大提升集群整体的资源利用率。对于抢占策略，一般是高优先级抢占低优先级，不过对于production和monitoring来说是不可被抢占的。

在系统运行的过程中，会出现这种情况：用户往往会多申请一部分资源，以确保服务压力突增时有一定的冗余空间。如果在分配资源时，以用户的请求值来实际的分配资源，那么就会导致集群累积大量的闲置资源。为此，borg提出 reclaimed-resource的概念解决这个问题（over-commit思想）。

简单通过来说，如果节点有 100 的物理资源，这时候先以monitor/ production优先级申请 60，但是实际只使 40，那么这个节点就出现 (60-40)=20 的闲置资源(reclaimed-resource)。borg的处理思想是，当来有个batch/best-effort优先级的资源需求的时候，那么就认为这台机总共有（100+20=120）、还剩（120-60=60）的可分配资源， 对于monitor/production优先级的请求来说，则认为节点还有（100-60=40）的可分配。

这种分配方式，确保在单机上分配给monitor/production级别任务的资源之和不会超过真实物理资源。当monitor/production级别的资源，增加到其申请的配额数的时候(比如服务器压力突升)，可以通过抢占低优先级任务来满足其需求。通过这种资源超发机制，大大提高了资源利用率。

## Borg的经验和教训

Job是唯一的Task分组机制，无法很好的标识服务间的关系，因此Borg无法很好的管理多个Job组成的单个服务。为了避免这些困难，Kubernetes不用job这个概念，而是用标签(label)来管理它的调度单位(pods)，标签是任意的键值对，用户可以把标签打在系统的所有对象上。Kubernetes用标签选择这种方式来选取对象，完成操作。这样就比固定的job分组更加灵活好用。

每台机器只有1个IP把事情弄复杂了，这种策略将端口和IP作为资源参与到Task调度中去了，对于Task的发起者也要事先声明一下需要的端口。Kubernetes可以用一种更用户友好的方式来消解这些复杂性：所有pod和service都可以有一个自己的IP地址，允许开发者选择端口而不是委托基础设施来帮他们选择，这些就消除了基础设置管理端口的复杂性。

# Borg与Kubernetes的对比

Kubernetes是一个开源的容器集群系统，为容器化的应用提供资源调度、部署运行、服务发现、扩容缩容等整一套功能。kubernetes与Borg非常类似，是Borg的一个开源实现。Borg底层是lxc容器，而kubernetes支持不止一种容器类型，默认是docker容器，还支持着rocket容器。Borg是C++编写的，而kubernetes则是Go语言编写的，Borg在集群调度性能上做了很多的优化，但是kubernetes对于调度这块没有做过多的优化措施，仅仅是一些机器适用性的选择。Borg可以支持上万的集群规模，而kubernetes支持的机器数有限，千台左右。

Kubernetes可以说是Borg的简化版的实现，它虽然不能像Borg那样支持超大规模的集群、无法使用优秀的调度策略，但是它弥补了很多Borg设计之初的不足。比如细分了任务的类型，通过Pod形式作为一个服务的管理单位；其次kubernetes将资源隔离交给基础容器去做，为每一个服务单独赋予一个IP，而不是像Borg那样，一个机器上的所有应用共享本地的IP和端口，这导致端口成为资源调度的一部分，复杂了资源管理的过程。

*liuchang*

*https://github.com/willstudy*

*纸上得来终觉浅，绝知此事要躬行~*