摘要

谷歌的Borg系统群集管理器运行几十万个以上的jobs，来自几千个不同的应用，跨多个集群，每个集群有上万个机器。

它通过管理控制、高效的任务包装、超售、和进程级别性能隔离实现了高利用率。它支持高可用性应用程序与运行时功能，最大限度地减少故障恢复时间，减少相关故障概率的调度策略。Borg简化了用户生活，通过提供一个声明性的工作规范语言，名称服务集成，实时作业监控，和分析和模拟系统行为的工具。

我们将会展现Borg系统架构和特点，重要的设计决策，定量分析它的一些策略，和十年以来的运维经验和学到的东西。

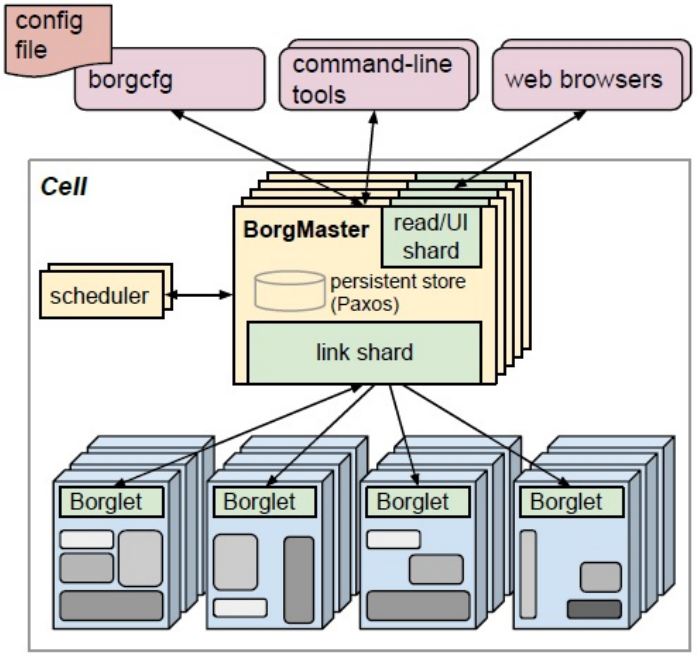

Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

1. 简介

集群管理系统我们内部叫Borg，它管理、调度、开始、重启和监控谷歌运行的应用程序的生命周期。本文介绍它是怎么做到这些的。

Borg提供了三个主要的好处：它（1）隐藏资源管理和故障处理细节，使其用户可以专注于应用开发；（2）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；（3）让我们在跨数以万计的机器上有效运行。Borg不是第一个来解决这些问题的系统，但它是在这个规模，这种程度的弹性和完整性下运行的为数不多的几个系统之一。

本文围绕这些主题来编写，包括了我们在生产环境运行十年的一些功力。

2.用户视角

Borg的用户是谷歌开发人员和系统管理员（网站可靠性工程师 SRE），他们运行谷歌应用与服务。用户以job的方式提交他们的工作给Borg，job由一个或多个task组成，每个task含有同样的二进制程序。一个job在一个Borg的Cell里面跑，一个Cell是包括了多台机器的单元。这一节主要讲用户视角下的Borg系统。

2.1 工作负载

Borg Cell主要运行两种异构的工作负载。第一种是长期的服务，应该“永远”运行下去，并处理短时间的敏感请求（几微秒到几百毫秒）。这种服务是面向终端用户的产品如Gmail、Google Docs、网页搜索，内部基础设施服务（例如，Bigtable）。第二种是批处理任务，需要几秒到几天来完成，对短期性能波动不敏感。在一个Cell上混合运行了这两种负载，取决于他们的主要租户（比如说，有些Cell就是专门用来跑密集的批处理任务的）。工作负载也随着时间会产生变化：批处理任务做完就好，终端用户服务的负载是以每天为周期的。Borg需要把这两种情况都处理好。

Borg有一个2011年5月的负载数据[80]，已经被广泛的分析了[68,26，27，57，1]。

最近几年，很多应用框架是搭建在Borg上的，包括我们内部的MapReduce[23]、flumejava[18]、Millwheel[3]、Pregel[59]。这中间的大部分都是有一个控制器，可以提交job。前2个框架类似于YARN的应用管理器[76]。我们的分布式存储系统，例如GFS[34]和他的后继者CFS、Bigtable[19]、Megastore[8]都是跑在Borg上的。

在这篇文章里面，我们把高优先级的Borg的jobs定义为生产（prod），剩下的是非生产的（non-prod）。大多长期服务是prod的，大部分批处理任务是non-prod的。在一个典型的Cell里面，prod job分配了70%的CPU资源然后实际用了60%；分配了55%的内存资源然后实际用了85%。在$5.5会展示分配和实际值的差是很重要的。

2.2 集群和Cell

一个Cell里面的所有机器都属于单个集群，集群是由高性能的数据中心级别的光纤网络连接起来的。一个集群安装在数据中心的一座楼里面，n座楼合在一起成为一个site。一个集群通常包括一个大的Cell还有一些小的或测试性质的Cell。我们尽量避免任何单点故障。

在测试的Cell之外，我们中等大小的Cell大概包括10000台机器；一些Cell还要大很多。一个Cell中的机器在很多方面都是异构的：大小（CPU、RAM、disk、network）、处理器类型、性能以及外部IP地址或flash存储。Borg隔离了这些差异，让用户单纯的选择用哪个Cell来跑任务，分配资源、安装程序和其它依赖、监控系统的健康并在故障时重启。

（译者：Cell其实就是逻辑上的集群）

2.3 job和task

一个Borg的job的属性有：名字、拥有者和有多少个task。job可以有一些约束，来指定这个job跑在什么架构的处理器、操作系统版本、是否有外部IP。约束可以是硬的或者软的。一个job可以指定在另一个job跑完后再开始。一个job只在一个Cell里面跑。

每个task包括了一组linux进程，跑在一台机器的一个容器内[62]。大部分Borg的工作负载没有跑在虚拟机（VM）里面，因为我们不想付出虚拟化的代价。而且，Borg在设计的时候还没硬件虚拟化什么事儿哪。

task也有一些属性，包括资源用量，在job中的排序。大多task的属性和job的通用task属性是一样的，也可以被覆盖 —— 例如，提供task专用的命令行参数，包括CPU核、内存、磁盘空间、磁盘访问速度、TCP端口等等，这些都是可以分别设置并按照一个好的粒度提供。我们不提供固定的资源的单元。Borg程序都是静态编译的，这样在跑的环境下就没有依赖，这些程序都被打成一个包，包括二进制和数据文件，能被Borg安装起来。

用户通过RPC来操作Borg的job，大多是从命令行工具，或者从我们的监控系统（$2.6）。大多job描述文件是用一种申明式配置文件BCL -- GCL[12]的一个变种，会产生一个protobuf文件[67]。BCL有一些自己的关键字。GCL提供了lambda表达式来允许计算，这样就能让应用在环境里面调整自己的配置。上万个BCL配置文件超过一千行长，系统中累计跑了了千万行BCL。Borg的job配置很类似于Aurora配置文件[6]。

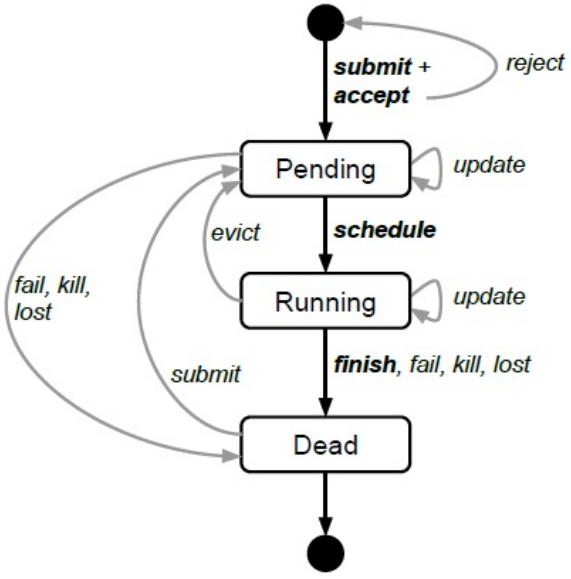

Figure 2: The state diagram for both jobs and tasks. *Users can trigger submit, kill, and update transitions.*

图2展现了job的和task的状态机和生命周期。

用户可以在运行时改变一个job中的task的属性，通过推送一个新的job配置给Borg。这个新的配置命令Borg更新task的规格。这就像是跑一个轻量级的，非原子性的事务，而且可以在提交后轻易再改回来。更新是滚动式的，在更新中可以限制task重启的数量，如果有太多task停掉，操作可以终止。

一些task更新，例如更新二进制程序，需要task重启；另外一些例如修改资源需求和限制会导致这个机器不适合跑现有的task，需要停止task再重新调度到别的机器上；还有一些例如修改优先级是可以不用重启或者移动task的。

task需要能够接受Unix的SIGTERM信号，在他们被强制发送SIGKILL之前，这样就有时间去做清理、保存状态、结束现有请求执行、拒绝新请求。实际的notice的delay bound。实践中，80%的task能正常处理终止信号。

2.4 Allocs

Borg的*alloc*（allocation的缩写）是在单台机器上的一组保留的资源配额，用来让一个或更多的task跑；这些资源一直分配在那边，无论有没有被用。allocs可以被分配出来给未来的task，用来保持资源在停止一个task和重启这个task之间，用来聚集不同jobs的tasks到同一台机器上——例如一个web server实例和附加的，用于把serverURL日志发送到一个分布式文件系统的日志搜集实例。一个alloc的资源管理方式和一台机器上的资源管理方式是类似的；多个tasks在一个alloc上跑并共享资源。如果一个alloc必须被重新定位到其他的机器上，那么它的task也要跟着重新调度。

一个alloc set就像一个job：它是一组allocs保留了多台机器上的资源。一旦alloc set被创建，一个或多个jobs就可以被提交进去跑。简而言之，我们会用task来表示一个alloc或者一个top-level task（一个alloc之外的），用job来表示一个job或者alloc set。

2.5 优先级、配额和管理控制

当有超量的工作负载在运行的时候会发生什么事情？我们的解决方案是优先级和配额。

所有job都有优先级，一个小的正整数。高优先级的task可以优先获取资源，即使后面被杀掉。Borg定义了不重叠的优先级段给不同任务用，包括（优先级降序）：监控、生产、批任务、高性能（测试或免费）。在这篇文章里面，prod的jobs是在监控和生产段。

虽然一个降级的task总会在cell的其他地方找到一席之地。降级瀑布也有可能会发生，就是一个task降下来之后，把下面运行的task再挤到别的机器上，如此往复。为了避免这种情况，我们禁止了prod级task互相排挤。合理粒度的优先级在其他场景下也很有用——MapReduce的master跑的优先级比worker高一点，来保证他们的可用性。

优先级是jobs的相对重要性，决定了jobs在一个cell里面是跑还是等（pending）。配额则是用来决定jobs是否运行被调度。配额就是一组资源（CPU，RAM，disk）的数量在一个指定的优先级、一个指定的时间段（月这个量级）。数量决定了这个用户的job可以用的最多资源（例子：20TB内存和prod优先级从现在到7月在xx cell内）。配额检查是管理控制的一部分，不是调度层的：配额不足的任务在提交的时候就会被拒绝。

高优先级的配额总是花费的比低优先级要多。prod级的配额是被限制为一个cell里面实际的资源量，所以用户提交了prod级的job的配额时，可以期待这个job一定会跑，去掉一些碎片外。即使这样，我们鼓励用户多买一点比自己需要多一点的配额，很多用户超买是因为他们的应用程序的用户数量增长后需要的配额就大了。对于超买，我们的应对方案是低优先级资源的超售：所有用户在0优先级都可以用无限的配额，虽然在实际运行中这种情况很难跑起来。一个低优先级的job在资源不足时会保持等（pending）状态。

配额分配在Borg系统之外，和我们的物理资源计划有关。这些资源计划在不同的数据中心产生不同的价格和配额。用户jobs只在有足够配额和足够优先级之后才能启动。配额的使用让Dominant Resource Fairness（DRF）[29, 35, 36, 66]不是那么必要了。

Borg有一个容量系统给一些特殊权限给某些用户，例如，允许管理员删除或修改cell里面的job，或者允许用户区访问特定的内核特性或者让Borg对自己的job不做资源估算（$5.5）。

2.6 命名和监控

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，即使它换了个地方。为了搞定这一点，Borg创造了一个稳定的“Borg name Service”（BNS）名字给每个task，这个名字包括了cell名字，job名字，和task编号。Borg把task的主机名和端口写入到一个持久化高可用文件里，以BNS名为文件名，放在Chubby[14]上。这个文件被我们的RPC系统使用，用来发现task的终端地址。BNS名称也是task的DNS名的基础构成部分，所以，cc cell的ubar用户的jfoo job的第50个task的DNS名称会是50.jfoo.ubar.cc.borg.google.com。Borg同时还会把job的大小和task的健康信息写入到Chubby在任何情况改变时，这样负载均衡就能知道怎么去路由请求了。

几乎所有的Borg的task都会包含一个内置的HTTP服务，用来发布健康信息和几千个性能指标（例如RPC延时）。Borg监控这些健康检查URL，把其中响应超时的和error的task重启。其他数据也被监控工具追踪并在Dashboard上展示，当服务级别对象（SLO）出问题时就会报警。

用户可以使用一个名叫Sigma的web UI，用来检查他们所有的job状态，一个特殊的cell，或者深入到某个job的某个task的资源用率，详细日志，操作历史，和最终命运。我们的应用产生大量的日志，都会被自动的滚动来避免塞满硬盘，会在一个task结束后保留一小段时间用来debug。如果一个job没有被跑起来，Borg会提供一个为什么挂起的解释，指导用户怎么修改这个job的资源需求来符合目前这个cell的情况。我们发布资源的使用方针，按照这个方针来做就容易被调度起来。

Borg记录所有的job提交和task时间，以及每task的资源使用细节在基础存储服务里面。这个存储服务有一个分布式的只读的SQL-like的交互式接口，通过Dremel[61]提供出来。这些数据在实时使用、debug、系统查错和长期容量规划上都很有用。这些数据也是Google集群负载追踪的数据来源之一[80]。

所有这些特性帮助用户理解和debug Borg的行为和管理他们的job，并且帮助我们的SRE每个人管理超过上万台机器。