



基于 Kubernetes 云平台的弹性伸缩方案设计与实现

单朋荣¹, 杨美红¹, 赵志刚¹, 李志鹏², 杨丽娜¹

(1. 齐鲁工业大学(山东省科学院) 山东省计算中心(国家超级计算济南中心) 山东省计算机网络重点实验室, 济南 250000;
2. 同济大学 电子信息工程学院, 上海 201804)

摘 要: 容器技术可为多个业务需求及其依赖组件提供独立的应用资源, 在现实生产环境中由于容器中的业务需求不断变化, 使得与其对应的应用资源在线负载处于动态变化中, 面临固定资源容量规划不能满足在线负载变化的困境。为解决该问题, 设计一种基于 Kubernetes 云平台的弹性伸缩方案。该方案通过集成 Prometheus 监控系统来自定义指标与采集业务指标, 并结合 HPA、VPA 等组件, 实现包括自定义指标和不同维度伸缩方法相结合的最佳弹性伸缩方法。通过集成 Grafana 页面显示和报警等组件, 实现实时查看弹性伸缩状态变化以及伸缩预警功能, 以实时观测集群健康状态, 使得集群操作更加友好、便于维护。实验结果表明, 在不同压力测量测试下, 该弹性伸缩方案具有随负载增加扩大集群规模的作用, 能够增强应用集群的高可用能力。

关键词: 云平台; Kubernetes 技术; 弹性伸缩; Prometheus 系统; HPA 组件

开放科学(资源服务)标志码(OSID):



中文引用格式: 单朋荣, 杨美红, 赵志刚, 等. 基于 Kubernetes 云平台的弹性伸缩方案设计与实现[J]. 计算机工程, 2021, 47(1): 312-320.

英文引用格式: SHAN Pengrong, YANG Meihong, ZHAO Zhigang, et al. Design and implementation of elastic scaling scheme based on Kubernetes cloud platform[J]. Computer Engineering, 2021, 47(1): 312-320.

Design and Implementation of Elastic Scaling Scheme Based on Kubernetes Cloud Platform

SHAN Pengrong¹, YANG Meihong¹, ZHAO Zhigang¹, LI Zhipeng², YANG Lina¹

(1. Shandong Provincial Key Laboratory of Computer Networks, Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Jinan 250000, China;
2. School of Electronic Information Engineering, Tongji University, Shanghai 201804, China)

[Abstract] Container technology can provide independent application resources for multiple service requirements and their dependent components. In the real production scenarios, the online loads of application resources are always in change along with the continuously changing service requirements in the container, so the fixed resource capacity planning fail to meet the changing requirements of online loads. In order to solve the problem, this paper designs an elastic scaling scheme based on the Kubernetes cloud platform. This scheme integrates the Prometheus monitoring system to customize the indicators and collect service indicators, and combines Horizontal Pod Autoscaler (HPA), Vertical Pod Autoscaler (VPA) and other components to realize the optimal elastic scaling method that is capable of both customizing indicators and scaling in different dimensions. Furthermore, by integrating Grafana page display, alarm and other components, this scheme realizes the real-time view of elastic scaling state changes and early warning function for scaling, so as to observe the health state of the cluster in real time, making the cluster operations more friendly and easy to maintain. Experimental results show that under different pressure measurement and tests, the elastic scaling scheme is capable of increasing the cluster size along with the increase of loads, and can enhance the high availability of the application cluster.

[Key words] cloud platform; Kubernetes technology; elastic scaling; Prometheus system; HPA components

DOI: 10.19678/j.issn.1000-3428.0056560

基金项目: 国家重点研发计划(2018YFB0203903)。

作者简介: 单朋荣(1995—), 男, 硕士研究生, 主研方向为云计算、软件工程; 杨美红, 研究员; 赵志刚, 副研究员; 李志鹏, 博士研究生; 杨丽娜, 硕士研究生。

收稿日期: 2019-11-11 **修回日期:** 2020-01-07 **E-mail:** 1542651257@qq.com

0 概述

近年来,容器(Docker)^[1]技术的飞速发展带来了云平台技术的新一轮变革,它使得打包应用可以无缝迁移到具备容器基础运行环境的平台上,其本质是建立在Linux的Cgroup、Namespace等技术上的虚拟化技术,而容器(镜像)打包的本质是打包本地的文件系统,文件系统代表本地的应用环境,从而实现将应用及其依赖环境一起打包。同时,容器技术解耦了Linux底层实现机制,由此赋予了容器轻量、灵活等特性,但容器技术归根到底只是一种虚拟化技术,虽然能将应用抽象成云端的一个可迁移单位,但云平台上所承载的应用数以亿万计,由此需要通过工具来编排容器,使容器技术上升到PaaS层,从而带来真正的商业价值。

Kubernetes^[2]作为业内领先的基于容器技术的分布式系统支撑平台,提供了完备的集群管理能力及工具,具备开放式的可扩展机制和先进的大规模集群编排理念。Kubernetes项目是随着Docker公司发布容器技术后逐渐兴起的,目前已成为容器编排领域的事实标准^[3]。容器编排面向的是PaaS层,为领先该领域,以Docker Swarm、Mesos和Kubernetes为代表的技术进行了改进^[4]。Kubernetes技术凭借“先进的设计理念”和开源生态所落地的“用户二次创新”^[5]能力,最终确立了其在容器编排领域的主导地位,也使得以容器为代表的形态技术成为了引领“下一代数据中心”^[8]的关键技术之一。OpenStack^[6]虽然也通过各种方式增加对容器的支持,但它仍然是以资源为中心,且管理的核心目标是机器,目前不被视为管理容器的主流平台。Mesos作为业内主流的通用计算资源管理平台,为编排容器推出了Mesosphere项目^[7],但Mesos社区与容器技术的关系更多地是“借势”,加上它所属Apache社区固有的封闭性,在容器编排领域缺乏创新性而逐渐失去竞争力。

Kubernetes已成为业内主流的容器编排方案,Kubernetes集群中抽象出各类资源对象应用程序编程接口(Application Programming Interface, API),用于资源对象的统一管理和维护。Kubernetes集群编排的基本可运行的单位为Pod^[2],它是应用资源的抽象和对容器的进一步封装,也是Kubernetes平台中弹性伸缩和调度的基本单位,Pod与容器一样具备轻量 and 易移植性等特性。因此,Kubernetes编排系统可根据编排对象模板在短时间内生成极为庞大的可再生Pod副本集。正是因为Kubernetes技术的独特性,它赋予了云原生^[2]时代下弹性伸缩细粒度、多维度、可扩展、高效率、低成本和高可用等新特点。因为

Pod非常轻量 and 灵活,所以弹性伸缩在伸缩效率和成本上相较于虚拟机都有明显优势^[8]。

作为Kubernetes核心的发布功能之一,弹性伸缩技术及其方案的设计与实现已成为衡量“容器云”平台服务能力的重要参考标准。弹性伸缩技术中最具代表性的技术分别是垂直弹性伸缩技术和水平弹性伸缩技术。文献[9]提出一种使用cAdvisor和Heapster^[10]采集汇集数据的弹性伸缩技术,但Heapster的强耦合性使得它的扩展性较差,难以解决实际应用中度量指标的多样化问题。在社区Kubernetes1.11版本后,Kubernetes使用Metric Server(指标服务器)来替代Heapster,虽然Metric Server对CPU和内存等资源支持良好,但不支持用户自定义指标。Kubernetes1.11版本后提供的“指标聚合”^[11]功能,为自定义指标的引入提供了基础。文献[12]提出基于HPA的叠加E-HPA弹性伸缩容系统只给出了水平弹性伸缩方法,对异构环境下的多维度弹性伸缩需求支撑不足。

本文提出Kubernetes云平台自定义指标^[13]和不同维度相结合的弹性伸缩方案。该方案通过集成Prometheus^[14]监控系统来自定义和采集业务指标,并结合HPA组件实现自定义指标的弹性伸缩方案,以满足不同业务场景根据业务指标完成伸缩的需求,通过搭配使用HPA、CPA和VPA^[15]等组件,实现水平、垂直和根据集群规模伸缩资源的策略,以在异构环境下选择弹性伸缩方案。

1 弹性伸缩场景和方案分析

1.1 弹性伸缩场景

因为业务需求在不断变化,应用资源的在线负载也处于动态的变化中,所以在生产环境中常面临资源的容量规划不能满足在线负载变化的困境。为解决这个问题,需要云平台具备动态扩缩容应用和虚拟机集群节点的能力。随着应用类型的多样性发展及云平台技术的革新,应用对云资源的需求也出现更加丰富的态势。常见的应用类型可分在线负载型、离线任务型、定时任务型和特殊场景型等。不同的应用类型往往会对弹性伸缩提出不同的要求,如:在线负载型应用对弹出时间敏感,机器学习等离线任务型对价格敏感,定时任务对调度敏感,自定义伸缩和超算等异构场景对弹出稳定性敏感。单一的弹性伸缩策略已不能满足这种多元化的需求,云平台需要一种多维度、立体的弹性伸缩方案来解决这一难题。同时,随着业务类型的丰富,弹性伸缩方案不仅需要基于“资源类型”指标,也需要基于“业务指标”等自定义指标,来解决更加复杂的应用场景下弹性伸缩的难题,并达到弹性伸缩方案覆盖场景广、适

用性强和易扩展的目标。

1.2 不同维度的弹性伸缩方案

弹性伸缩方案首先通过“监控系统”采集指标,然后将采集的指标聚合到“指标服务器”,并由其提供指标,最后“弹性伸缩组件”依据提供的指标触发伸缩动作。用户通常会根据业务的需求和集群的特点集成不同的方案,从而形成不同维度和业务类型的弹性伸缩方案。

从用户角度分析,弹性伸缩分为集群非自定义弹性伸缩方案和自定义弹性伸缩方案。非自定义弹性伸缩方案是指自 Kubernetes1.11 版本废弃 heapster 后,迭代出了 Metric Server 组件,实现了内部组件间的松耦合和可扩展性。自定义弹性伸缩方案是指引入外部监控系统,并实现相应的适配器以适配到 Kubernetes 中。从指标维度分析可将指标大致分为 Resource Metrics、Custom Metrics 和 External Metrics 3 类指标。其中:Resource Metrics 是指系统(核心)资源指标,指标由系统组件 kubelet 提供;Custom Metrics 是包括 Pods 和 Object 的自定义指标类型,需配套搭建自定义指标服务器和“监控系统”,其中监控系统负责进行采集和处理数据并提供给自定义指标服务器;External Metrics 指标由公有云厂商提供,通常基于云端的消息服务和负载均衡器的 QPS 等来实现弹性扩缩容。另外,“指标”通常以自定义资源(Custom Resource Definition, CRD)^[2]方式定义和注册为 API 资源对象,从而被伸缩组件获取到并根据预先设定规则进行弹性伸缩。

以弹性伸缩的对象为维度,弹性伸缩方案可分为应用和集群节点的弹性伸缩。应用节点的弹性伸缩是指扩缩容应用集群的节点数量,即 Pod 的数量;集群节点的弹性伸缩是增加或减少虚拟机/物理节点的数量。

从资源的伸缩方向来分析,弹性伸缩大致分为两类组件:一类是修改节点的数量从水平方向来弹性伸缩,使应用和集群在水平方向上具备弹性能力;另一类是从垂直方向来改变资源的分配量,而不改变 Pod 的数量来实现资源容量的扩缩容。

图 1 所示分别以伸缩对象和伸缩方向为横纵坐标建立弹性伸缩二象限。每个坐标点代表它的伸缩组件和配套的实施方案,忽略指标获取和调度层面,单从弹性伸缩动作触发层面分析,弹性伸缩组件包括 CA (Cluster Autoscaler)、HPA (Horizontal Pod Autoscaler)、CPA(Cluster Proportional Autoscaler)、VPA (Vertical Pod Autoscaler) 和 AR (Addon Resizer)^[2]。其中,CA 的功能是从水平方向扩缩容资源池的大小,即增加或减少物理节点的数目,HPA 的功能是动态增加或减少集群中 Pod 的数目,CPA 可以依据集

群的规模来同比例增加或减少集群中“核心组件”的数目,主要是为了解决“核心组件”的弹性问题,VPA 的功能是增加或减少资源的请求值,不改变 Pod 数目,Addon Resizer 则具备根据集群中节点的数目来调整负载的资源请求值,目前尚不成熟。另外,互不冲突的组件之间的搭配使用可以实现更加“极致”的弹性伸缩方法。

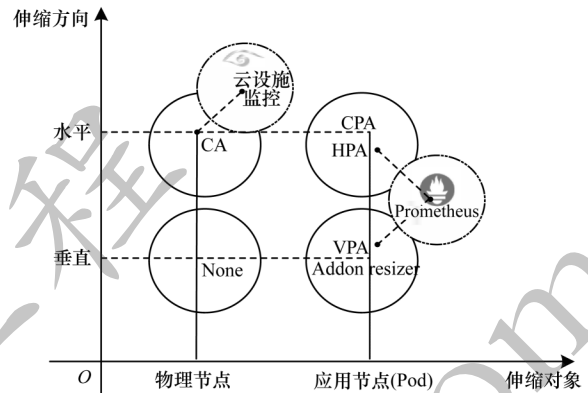


图 1 弹性伸缩维度示意图

Fig.1 Schematic diagram of elastic telescopic dimension

2 基于自定义指标的弹性伸缩方案

基于自定义指标的弹性伸缩方案是指引入三方监控提供“业务”指标类型,结合 HPA 组件实现基于业务指标的弹性伸缩方案。其中 HPA 组件是弹性伸缩方案中的核心组件,三方监控系统是指 Prometheus 监控体系。

2.1 HPA 组件

2.1.1 HPA 架构

HPA 架构如图 2 所示。

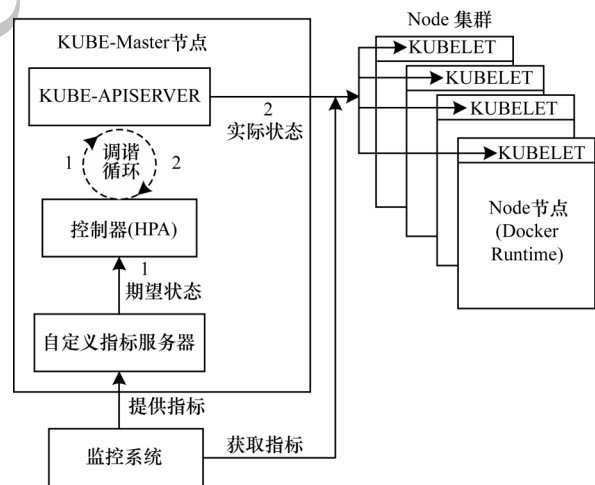


图 2 HPA 伸缩架构

Fig.2 HPA telescopic architecture

HPA 架构基本遵循 Kubernetes 中声明式 API 和控制模型^[5]的设计理念。声明式 API 是 Kubernetes

(API Server)的一项重要能力,即以YAML(Yet Another Markup Language)^[2]文件的方式声明所期望集群的状态,然后由控制器获取集群的实际状态并执行相应的逻辑,来完成期望和实际状态的调谐过程。其中,Kube apiserver组件负责声明式API对象的管理,Controller(HPA)组件是弹性伸缩的控制器,Tuning Process代表调谐的循环流程。它们统一由KUBERNETES MASTER管理,由三方监控提供指标服务,最终将期望结果作用到实际的物理集群中。

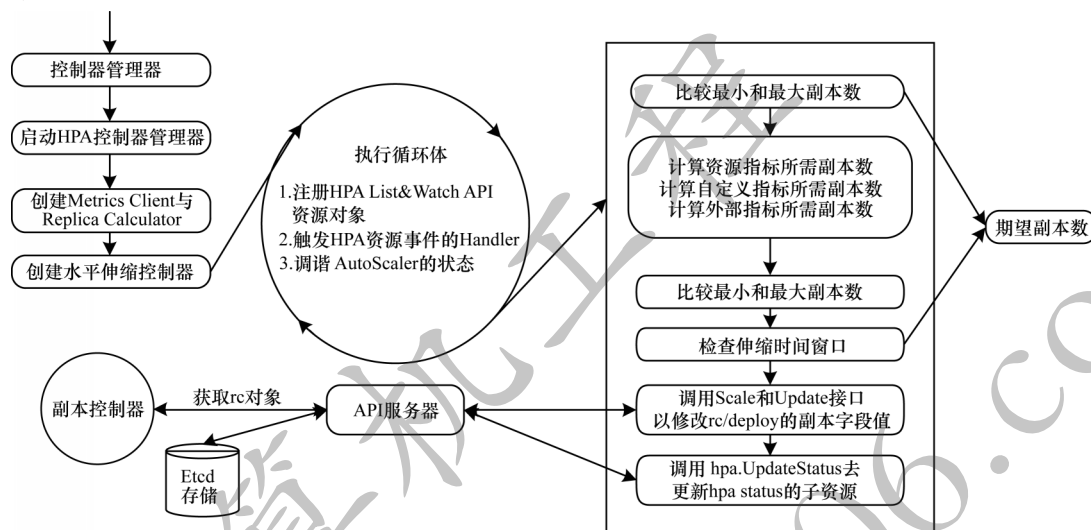


图3 HPA代码流程

Fig.3 HPA code procedure

代码的入口部分是控制器管理器^[2],大部分Kubernetes内置核心组件都由该组件负责管理和维护,然后进入HPA Controller部分,其中Metrics指标获取客户端由之前的New Heapster Metrics Client替换为New REST Metrics Client。前者耦合了Heapster进行资源监控和指标获取;后者以松耦合的能力集成了三方监控并提供了Resources Metrics、Custom Metrics和External Metrics等的指标类型,目前由autoscaling/v2beta2版本支持。接着进入创建HPA Controller流程,执行控制循环并计算得出新的Replica的期望值,并同步HPA的资源状态。最后RC会监测(Watch)Replica资源对象的状态,并执行后续的代码逻辑。

2.1.3 HPA执行流程

如图4所示,HPA控制器首先从聚合API持续获取指标,再基于内部的扩缩容规则进行计算,得到目标Pod的副本数量,即Replicas字段值,最后发出扩容的伸缩指令。当集群中“期望”和“实际”状态不匹配时,HPA控制器就会向RC、Deployment或者ReplicaSet发起Scale指令,修改Replicas字段的值,然后由相应控制器检测该字段值的变化,从而调整Pod的副本数量。

2.1.2 HPA代码流程

如图3所示,HPA Controller使用List&Watch^[2]方法获得所监控对象的实际状态,然后触发相应的事件处理逻辑,最后完成调谐的过程。需要注意的是,HPA Controller直接作用的资源对象并不是集群里的应用(Pod),而是一种“中间”资源对象的抽象概念,即Replicas。Replica资源对象由复制控制器(Replication Controller,RC)^[2]管理和维护,然后由它来控制Pod的数量变化。

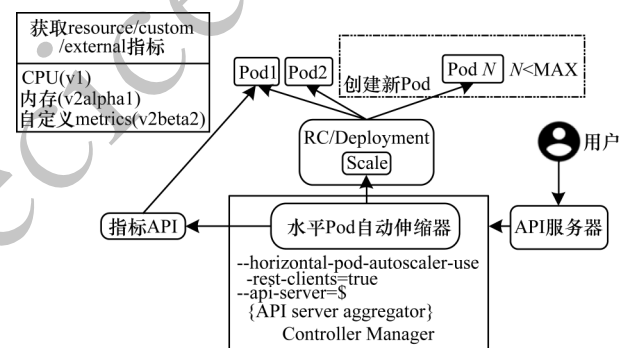


图4 HPA执行流程

Fig.4 HPA execution procedure

自Kubernetes1.9版本以后,社区对StatefulSet(有状态应用控制器)和Deployment^[2]进行改进,切换到了统一的Scaler Interface实现接口,所以HPA控制器同样可以向StatefulSet发起Scale指令,从而调整“有状态”应用的Pod的数量。同理,如果用户自己的CRD支持Scaler接口,就可以被HPA管理,从而实现自定义资源类型的动态扩缩容。

3 弹性伸缩组件搭配方案

弹性伸缩方案的搭配可分为两个分析方向:一是弹性伸缩组件和第三方监控配套方案的搭配;

二是弹性伸缩组件之间从不同维度上的搭配使用。其中,弹性伸缩组件搭配的配套方案是指集成 Prometheus 监控系统,实现自定义业务指标的弹性伸缩策略。组件之间的搭配是指面向不同的异构环境、业务场景和用户需求,各组件组合的多维度弹性伸缩方案。

3.1 自定义弹性伸缩配套方案

如图5所示,自定义弹性伸缩是指在指标层面上集成三方监控,以获取业务指标来实现自定义的弹性伸缩方法。目前,三方监控主要有 Prometheus、Microsoft Azure^[16]和 Datadog Cluster^[17]等,然后实现相应的指标适配器(Custom Metrics Server),将指标聚合到 Aggregator,由其向弹性伸缩控制器提供所需指标。本文论述的是基于 Prometheus 监控系统实现的自定义指标服务器方案, Prometheus 可以支持 Kubernetes 集群的监控,对时序数据具备优秀的处理能力。

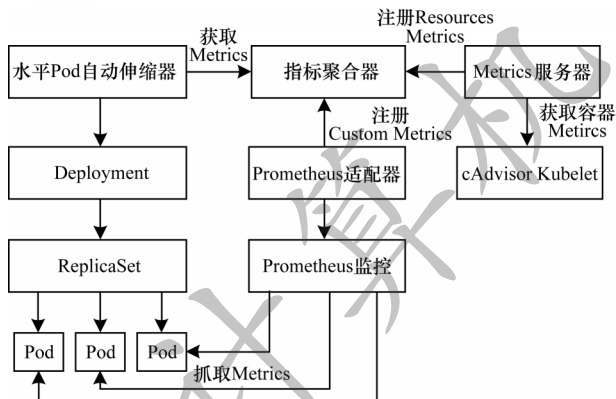


图5 HPA指标的配套方案

Fig.5 Supporting scheme of HPA index

3.2 不同维度上弹性伸缩方案的组合

HPA在资源池容量充足的情况下,可以方便地水平扩展应用节点的数量,但当资源池容量匮乏时,往往难以发挥作用。为解决该难题,需要搭配一个弹性伸缩组件,该组件具备在资源池容量不足时弹性扩展虚拟机/物理集群节点的数量,以增加资源池容量。资源需要缩容时同理。

3.3 CA组件

CA是物理集群节点级别的扩缩容,扩容的条件是集群存在未调度的Pod且不在“冷却周期”内,缩容的条件是节点利用率低于阈值。

如图6所示,CA会监听所有的Pod,当出现未调度Pod时,便尝试从配置好的弹性伸缩组(Auto Scaling Group, ASG)中选择虚拟化的节点进行“模拟调度”。需要注意的是,增删节点只是触发ASG增删节点的接口,具体的实现由云厂商来完成。而当某节点资源利用率低于阈值并到达指定时间时,CA会在该节点打上禁止调度标签,然后驱逐容器,

最后逐一删除节点。同时,CA在伸缩规格、伸缩策略、多可用区和自定义伸缩等方面拥有丰富配置项,需要用户按需进行配置。

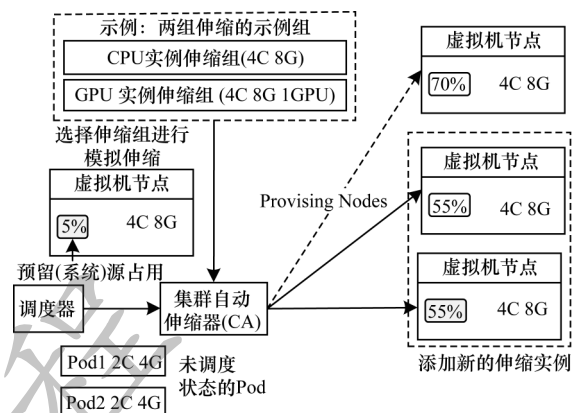


图6 CA伸缩流程

Fig.6 CA telescopic procedure

因调度器重新计算集群规模时具有时间间隔/冷却周期,当新节点加入集群时,并不能立即感知它的存在,所以CA对弹出时延敏感的应用场景支撑不足。为了使新弹出节点更好地满足弹出时延敏感的应用场景,社区内提出了两个主流的方案:一是在YAML文件中“声明字段”添加节点的声明信息来进行定向的调度,而不用一直等待冷却周期结束;二是“占位”思想,设置优先级非常低的Pod来进行抢占调度。其中,“占位”思想是指设置优先级较低的Pod占用而不使用资源。当集群中存在未调度Pod时,可直接对优先级低的Pod进行资源抢占;此时集群中会出现优先级低的Pod处于未调度的状态,进而触发CA组件来进行节点的伸缩。“占位”思想在不影响正常Pod调度的情况下,利用CA的触发条件使得集群可以“快速”地弹出节点,以满足弹出时延敏感的应用场景。

同时,“占位”思想也体现了当资源池资源充足时,使用HPA组件从调度的维度来使得集群充满弹性。当资源池资源匮乏时,使用CA组件对资源池资源进行扩充,实现了组件间搭配使用的“极致”弹性伸缩方案。

HPA常用于动态Pod的伸缩场景,无法支持静态Pod的伸缩需求。Kubeadm^[2]部署的系统“核心”组件都是以静态Pod方式存在于系统中,当集群规模变化时,为减轻系统组件的访问压力和增强可用性,需要“核心”组件具备弹性的能力。

3.4 CPA组件

CPA(Cluster Proportional Autoscaler)是根据集群节点数目进行Pod副本水平伸缩的组件,主要是解决Kubernetes集群中“核心组件”的负载弹性问题。它的弹性伸缩策略包括“线性模型”和“梯度模

型”两种,分别通过线性公式和匹配区间方式进行副本数的计算。CPA组件的集成分流了核心组件的负载压力,使集群服务更加稳定和高效。

HPA、CA和CPA组件的搭配使用赋予了水平方向上动态扩缩容集群节点的能力。当面向体量大的应用和“有状态应用”^[18]时,水平扩缩容节点便会变得极为困难。此时,需要一种解决方案来从不同维度和方向上扩缩容节点来解决这一难题,从而实现不同维度上的弹性伸缩方案。

3.5 VPA组件

VPA(Vertical Pod Autoscaler)是以CRD方式定义的垂直伸缩的组件,它能很好地支持有状态应用的弹性伸缩,有效弥补了HPA等组件对有状态应用弹性伸缩支持不完善的问题。VPA最具特色的是它的“资源推荐”功能,能根据实时和历史负载数据计算出合适的资源值,以初始化或者“更新”资源的请求值。VPA面向的是“离线”和“巨石”应用等场景,这些场景下的应用占用资源比例大或者因某些状态无法“解耦”,从而很难从数量方面来扩缩容资源。VPA具备从垂直方向来增加或减少资源量的能力,可以从垂直维度解决上述场景扩缩容需求。由于Kubernetes目前仍不支持资源请求值的热更新,VPA更新策略采取的是停止旧Pod并启动新Pod的方式。

如图7所示,VPA集成方案包括指标获取模块和VPA控制器两部分。其中,指标获取由Metric Server和Prometheus组件完成。前者负责“实时”数据的采集,后者提供“历史”监控数据。VPA控制器主要包含Recommender(资源建议)、Updater(资源更新)和Admission Controller(准入控制)等模块。其中:Recommender监控当前和历史负载数据,并提供资源请求的推荐值;Updater监控API Server中相关资源的变化,然后执行相应的更新策略;Admission Controller是Kubernetes API Server的一项重要功能,具备“拦截”和“热更新”相关API资源对象的能力。

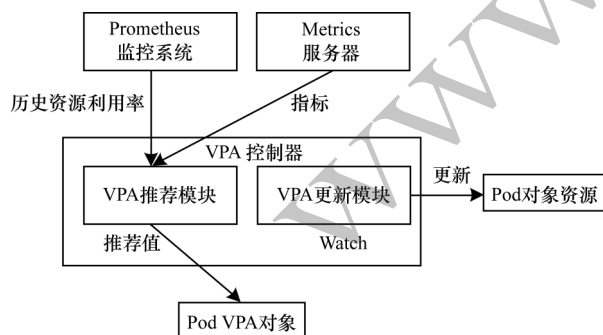


图7 VPA整体控制流程

Fig.7 Procedure of VPA overall control

执行流程首先是从监控组件中获取“实时”和“历史”的资源负载数据,然后由Recommender模块决策,并在VPA API对象中设置新的资源推荐值,此

时Updater模块会监测到VPA API资源对象的变化,并触发相应的“更新”和“异常处理”策略等。其中,“更新”简略流程主要包括:由VPA的“准入控制”模块,“拦截”并“更新”VPA API和它所关联的Pod资源对象的相关声明字段值,执行API Server的资源有效化流程,再由控制器写入重定义资源大小的注解,然后调度器负责调度,最后由Kubelet执行具体的Pod资源的变化需求。

通过集成VPA和HPA组件,赋予了集群从不同维度上弹性伸缩资源的能力,提供了在异构资源和不同业务场景下弹性伸缩方案选型的依据,同时满足了各类应用在不同维度上弹性伸缩的差异化需求。

弹性伸缩组件的伸缩动作都是以获取指标为前提,然后根据相应的计算规则得出所需的变化值。传统的指标获取手段是通过Metric Server指标服务器提供的“资源”指标,如CPU和内存的利用率等,但复杂场景下的弹性伸缩需要根据业务类型和需求来定制弹性伸缩方案,增加弹性伸缩方案的灵活性和适用性。

4 监控系统

无论是自定义指标的弹性伸缩方案还是VPA垂直伸缩获取历史负载信息,都需要集成三方的监控方案来获取丰富(业务)的指标类型。另外,可根据需要配套集成“可视化”和“报警”等组件。其中,“可视化”组件用来满足弹性伸缩Pod等各类资源的可视化分析需求,为监控系统和方案决策提供更好的支持。“报警”组件通过异常事件报警,整合故障恢复脚本,使集群更加稳定。

4.1 Prometheus监控系统

Prometheus作为云原生基金会(Cloud Native Computing Foundation, CNCF)的第二大开源项目,原生支持Kubernetes并已成为事实上的容器监控标准。它具备良好的时序数据处理能力和可扩展性,可以借助第三方存储离线监控数据或者选用Thanos^[19]方案,从而实现对历史数据的保存和利用。同时,它也具备自定义业务层指标的能力,可以很好地支撑自定义的弹性伸缩方案,以满足复杂场景下的弹性伸缩需求。

4.2 Prometheus架构和部署

如图8所示,Prometheus的监控方案主要包括指标采集、Prometheus服务器、报警和图形化显示等部分。Prometheus的指标采集方式分“推送”和“拉取”两种,其中“拉取”方式为官方推荐,但如果因网络或内网的防火墙等原因无法拉取指标时,可以使用“推送”的方式。另外,使用“推送”方式推送指标时,常借助第三方缓存中间件缓存中间数据,以免大量的被动推送的数据直接冲击服务器,从而引起服务器的瘫痪。而“拉取”指标需要提供Metrics数据接口,

如果采集目标没有直接提供该接口,则使用相应 Exporter 来暴露 Kubernetes 集群中相关指标数据。

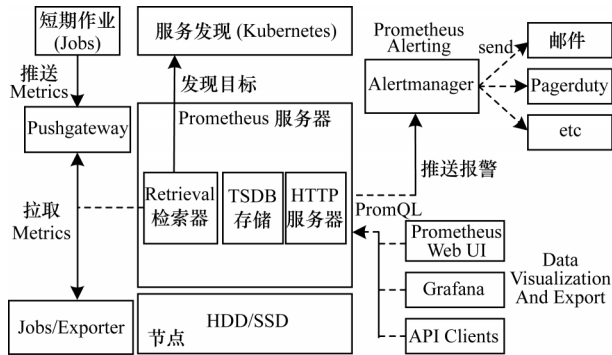


图 8 Prometheus 整体架构

Fig.8 Prometheus overall architecture

Prometheus 控制器使用 Prometheus Operator 方式部署,如图 9 所示,它使用了 CRD 的方式定义了“Prometheus 服务器”“资源监控”“告警组件”和“规则配置”的资源抽象,使得配置和管理监控系统变得简单而通用;同时它也具备“自动感知”“动态加载”Kubernetes 资源对象以及自我修复和确保服务的高可用等能力。

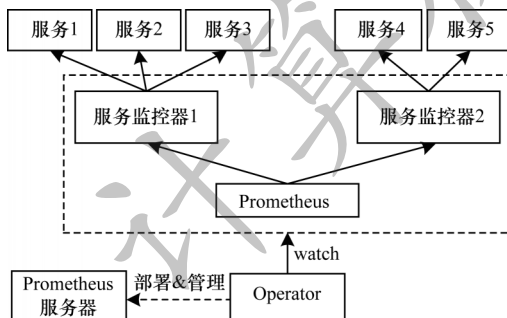


图 9 Prometheus Operator 部署架构

Fig.9 Prometheus Operator deployment architecture

由于 Prometheus 具备对 Kubernetes 资源的指标采集和配置的自动化处理能力,以及对时序数据检索、存储和聚类等高效的处理方法,使得 Prometheus 作为三方监控集成到 Kubernetes 集群中,并为弹性伸缩组件提供“指标”成为事实上的标准。其中, Prometheus 提供和支持用户自定义业务指标,其结合 HPA 组件是实现基于自定义质保弹性伸缩方案的最佳方案。

监控系统的报警模块主要依赖 Prometheus 控制器的“规则配置”和 AlertManager 报警接收器两部分,其主要通过 Prometheus 预设定规则来触发告警动作,然后发送告警到外部报警组件的方法来显现告警功能。图形化显示器除了 Prometheus 自带的 Prometheus Web UI 外,因 Grafana 出色的显示和易配置功能,业内常选用 Grafana 作为替代方案。Grafana 集群资源示意图如图 10 所示。

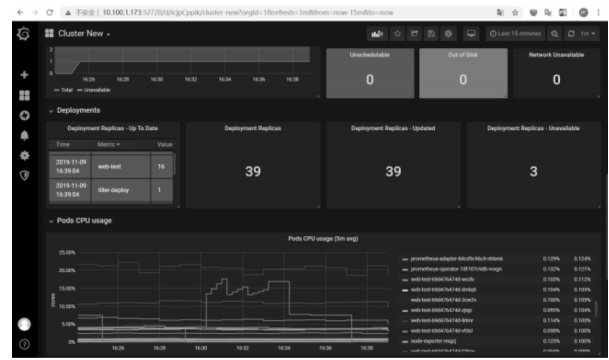


图 10 Grafana 集群资源示意图

Fig.10 Schematic diagram of Grafana cluster resources

5 实验结果与分析

本文基于 Kubernetes 集群环境进行实验,搭建方式为 Kubeadm,底层管理容器为 Docker。集群由 2 个 Master 节点(主节点、备节点)和 3 个 Node 节点构成。应用类型为在线负载型的 Web 应用(Web-test),可接受多用户端的并发访问。压测工具为 Hey,以 10 000 请求总量、并发度 10 和 10QPS 的访问速率对应用进行压力测试,在 2 min、6 min 和 10 min 3 个时间节点将上述的请求进行 1 倍压测量、2 倍压测量和 3 倍压测量的持续压力测试。

整个测试得出的数据结果和统计结果如下:

1) 压力测试前后应用集群的变化,即 Pod 数量的变化。

2) 在压力测试中,应用集群中各 Pod 的 CPU 和内存负载随着时间的变化结果。

3) 应用集群中所有 Pod 的 CPU 和内存的变化平均值随着时间的统计结果。

如图 11 所示,在压力测试过程中,随着应用集群负载的增加,Pod 的副本数出现明显变化,在 2 min、6 min 和 10 min 施加压力测试的时间节点上 Pod 副本数变化明显,说明弹性伸缩方案起到了随着负载增加,扩大集群规模的作用。

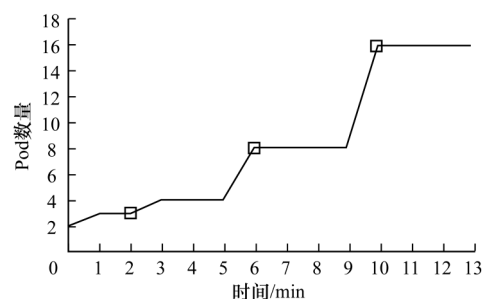


图 11 应用集群副本数变化趋势图

Fig.11 Change trend diagram of application cluster copies number

如表 1、表 2 所示,在 2 min、6 min 和 10 min 3 个压力测试时间节点上,CPU 和内存的负载在增加,同

时应用集群中出现了新的 Pod 副本。可以看出集群随着访问量的增加,各副本 Pod 资源消耗的变化情况,以及当负载持续增加时,新副本的出现对集群中各副本 Pod 负载逐渐趋于稳定的影响情况。

表 1 Web-test 集群中各 Pod CPU(cores,m)分配值分析比较
Table 1 Analysis and comparison of each Pod CPU(cores,m) allocation values in Web-test cluster

时间/min	Pod1	Pod2	Pod3	Pod4	Pod5	Pod6	Pod7	Pod8	Pod9	Pod10	Pod11	Pod12	Pod13	Pod14	Pod15	Pod16
0	1	1														
1	1	1	1													
2	1	1	1													
3	3	4	3	1												
4	3	4	3	1												
5	3	5	3	1												
6	3	4	3	1												
7	4	5	3	4	1	1	1	1								
8	3	5	3	3	1	1	1	1								
9	4	5	3	4	1	1	1	1								
10	4	6	3	3	1	1	1	1								
11	4	7	4	3	1	4	1	1	1	0	1	0	1			
12	3	6	4	4	1	5	1	1	1	1	1	1	1	1	1	1
13	4	6	4	4	1	4	1	1	1	1	1	1	1	1	1	1

表 2 Web-test 集群中各 Pod 内存分配值分析比较
Table 2 Analysis and comparison of each Pod memory allocation values in Web-test cluster

时间/min	Pod1	Pod2	Pod3	Pod4	Pod5	Pod6	Pod7	Pod8	Pod9	Pod10	Pod11	Pod12	Pod13	Pod14	Pod15	Pod16
0	2	2														
1	5	5	3													
2	6	7	6													
3	7	7	6	3												
4	7	8	6	5												
5	7	8	6	6												
6	9	8	6	7												
7	9	8	7	7	2	4	2	2								
8	9	8	7	8	5	5	4	5								
9	9	8	9	8	6	7	6	6								
10	9	9	9	8	6	7	6	6								
11	9	9	9	8	6	8	6	6	3	0	3	0	3			
12	9	9	10	8	6	8	6	6	4	5	3	6	3	5	4	4
13	9	9	10	8	6	8	7	6	6	6	7	6	6	7	6	6

如图 12 所示,在 2 min、6 min 和 10 min 3 个时间节点,当负载增加时(压力测试)的 1 min 内,应用集群总体的 CPU 和内存使用总量在开始时迅速递增,但随之趋于平缓,甚至有所回落。

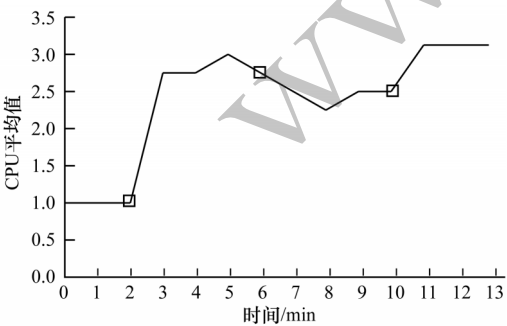


图 12 Web-test 应用集群 CPU 平均负载时间走势
Fig.12 CPU average load time trend in Web-test application cluster

结合表 2 可以看出,在同样的时间节点范围内,当负载增加时,Pod 的副本数量也在增加,均衡了应用集群整体的负载流量,使整个集群的平均负载值没有无节制性向上攀升直至 Pod 崩溃,而是使应用逐渐趋于“平缓”状态,甚至在 6 min 中后的一段时间内整体负载平均值有所回落,此时的 Pod 副本数增量较大。可以看出,弹性伸缩可以增强应用集群的高可用能力,使集群趋于稳定。

从表 3 可以看出,弹性伸缩弹出的 Pod 副本数在均衡策略的作用下均衡分布在各节点上,有利于集群资源的均衡及充分利用。表 3 的验证结果表明,节点的资源使用率是弹性伸缩弹出节点的重要考量因素之一,所以各节点的资源使用率基本保持在较均衡的状态。

表3 Web-test集群Pod分布和节点内存利用率统计结果

Table 3 Statistical results of Pod distribution and node memory utilization in Web-test cluster

节点	Pod数量	内存/Mi	利用率/%
Node1	6	32 672	74
Node2	5	74 341	77
Node3	5	65 683	68

6 结束语

本文设计一种基于 Kubernetes 云平台的弹性伸缩方案。该方案在 Kubernetes 中集成弹性伸缩服务,当负载流量突发变化时,可使应用集群更加稳定和健壮,并在不同业务场景下根据需求选择弹性伸缩组件之间的搭配方案,以满足不同场景下业务的个性化弹性伸缩需求。实验结果表明,通过集成监控系统 Prometheus、可视化和报警工具,该方案能够增强应用集群的高可用能力,使集群趋于稳定。在目前的研究中,弹性伸缩仍然基于实时的“容器工作流”负载工作,下一步将运用模型对负载进行短期或较长期预测,从而提出一种具有精准预测功能的“智能化”的弹性伸缩方案。

参考文献

- [1] ANDERSON C. Docker[J]. IEEE Software, 2015, 32(3): 102-103.
- [2] Kubernetes Community. Kubernetes official website[EB/OL]. [2019-10-05]. <https://kubernetes.io/docs/concepts>.
- [3] BURNS B, GRANT B, OPPENHEIMER D, et al. Borg, omega, and Kubernetes[J]. Queue, 2016, 14(1): 70-93.
- [4] DIKALEH S, SHEIKH O, FELIX C. Introduction to Kubernetes [C]//Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering. New York, USA: ACM Press, 2017: 310-310.
- [5] GONG Zheng. Kubernetes authoritative guide[M]. 4th ed. Beijing: Electronic Industry Press, 2019. (in Chinese)
龚正. Kubernetes权威指南[M]. 4版. 北京: 电子工业出版社, 2019.
- [6] ROSADO T, BERNARDINO J. An overview of Openstack architecture [C]//Proceedings of the 18th International Database Engineering & Applications Symposium. New York, USA: ACM Press, 2014: 366-367.
- [7] SAHA P, GOVINDARAJU M, MARRU S, et al. Integrating Apache Airavata with Docker, Marathon, and Mesos[J]. Concurrency and Computation: Practice and Experience, 2016, 28(7): 1952-1959.
- [8] YU Jingang, ZHAI Yarong, YU Bo, et al. Research and application of auto-scaling unified communication server based on Docker[C]//Proceedings of the 10th International Conference on Intelligent Computation Technology and Automation. Washington D. C., USA: IEEE Press, 2017: 152-156.
- [9] YANG Mao. Research on container auto-scaling based on Kubernetes [D]. Xian: Xidian University, 2018. (in Chinese)
杨茂. 基于 Kubernetes 的容器自动伸缩技术的研究[D]. 西安: 西安电子科技大学, 2018.
- [10] Compute resource usage analysis and monitoring of container clusters [EB/OL]. [2019-10-05]. <https://github.com/kubernetes-retired/heapster>.
- [11] BRATTSTROM M, MORREALE P. Scalable agentless cloud network monitoring[C]//Proceedings of the 4th IEEE International Conference on Cyber Security and Cloud Computing. Washington D. C., USA: IEEE Press, 2017: 171-176.
- [12] TU Xuezheng, YANG Haichao. A horizontal elastic expansion and contraction system based on Kubernetes[J]. Computer and Modernization, 2019(7): 25-31. (in Chinese)
屠雪真, 杨海潮. 基于 Kubernetes 的水平弹性伸缩容系统[J]. 计算机与现代化, 2019(7): 25-31.
- [13] ALTAU U, JAYAPUTERA G, LI J, et al. Auto-scaling a defence application across the cloud using Docker and Kubernetes[C]//Proceedings of IEEE/ACM International Conference on Utility and Cloud Computing Companion. Washington D. C., USA: IEEE Press, 2018: 327-334.
- [14] CHEN Xiaoyu. Detailed principles, applications, source code and expansion of Prometheus[M]. Beijing: Electronic Industry Press, 2019. (in Chinese)
陈晓宇. Prometheus原理、应用、源码与拓展详情[M]. 北京: 电子工业出版社, 2019.
- [15] CARDENAS Y M R. Scaling policies derivation for predictive autoscaling of cloud applications [EB/OL]. [2019-10-10]. <https://mediatum.ub.tum.de/1462656>.
- [16] ALHAMAZANI K, RANJAN R, MITRA K, et al. CLAMS: cross-layer multi-cloud application monitoring-as-a-service framework [C]//Proceedings of 2014 IEEE International Conference on Services Computing. Washington D. C., USA: IEEE Press, 2014: 283-290.
- [17] ABUSINNAIN A K, MOKHTAR T A, YOUSEF M A, et al. Web-based application performance monitoring tool [EB/OL]. [2019-10-10]. <http://repository.sustech.edu/handle/123456789/21332>.
- [18] VAYGHAN L A, SAIED M A, TOEROE M, et al. Microservice based architecture: towards high-availability for stateful applications with Kubernetes [C]//Proceedings of the 19th IEEE International Conference on Software Quality, Reliability and Security. Washington D. C., USA: IEEE Press, 2019: 176-185.
- [19] JASSAT U F. Point 5 compliant Kubernetes cluster[M]. Geneva, Swiss Confederation: [s. n.], 2019.
- [20] LIU Biao, WANG Baosheng, DENG Wenping. Containerized workflow framework supporting elastic scaling in cloud environment[J]. Computer Engineering, 2019, 45(3): 7-13. (in Chinese)
刘彪, 王宝生, 邓文平. 云环境下支持弹性伸缩的容器化工作流框架[J]. 计算机工程, 2019, 45(3): 7-13.