计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(7):77−84 [doi: 10.15888/j.cnki.csa.008565]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

面向 Kubernetes 的多集群资源监控方案①

李　轲, 窦　亮, 杨　静

(华东师范大学 计算机科学与技术学院, 上海 200062)
通信作者: 窦　亮, E-mail: ldou@cs.ecnu.edu.cn

摘　要: 应用系统的复杂化与微服务化促进了容器的广泛使用, 企业往往会根据业务需要使用 Kubernetes 搭建多个集群进行容器的编排管理与资源分配. 为实时监控多个集群的工作状态与资源使用情况, 提出了面向 Kubernetes 的多集群资源监控方案, 对 Kubernetes 提供的 CPU、内存、网络以及存储指标进行采集, 根据采集数据的类型对部分数据进行计算以获取更直观的监控指标, 实现了多层级多类型的存储, 并提供监控数据的 REST 接口. 通过实验, 验证了本设计对集群资源的消耗低, 具有较好的性能.

关键词: 容器技术; Kubernetes; 容器监控; 资源监控; 多集群; Docker

引用格式: 李轲,窦亮,杨静.面向 Kubernetes 的多集群资源监控方案.计算机系统应用,2022,31(7):77–84. http://www.c-s-a.org.cn/1003-3254/8565.html

Multi-cluster Resource Monitoring Scheme for Kubernetes

LI Ke, DOU Liang, YANG Jing

(School of Computer Science and Technology, East China Normal University, Shanghai 200062, China)

Abstract: The increasing complexity and microservice of application systems promote the widespread use of containers. Enterprises often build multiple clusters for container arrangement and management and resource allocation with Kubernetes according to their business needs. To monitor the working status and resource usage by multiple clusters in real time, this study proposes a multi-cluster resource monitoring scheme for Kubernetes. The CPU, memory, network, and storage indicators provided by Kubernetes are collected, and part of the data collected are calculated according to their data type to obtain more intuitive monitoring indicators. As a result, multi-level and multi-type storage is achieved, and REST interfaces for monitoring data are provided. Experiments verify that this design consumes a small amount of cluster resources and achieves good performance.

Key words: container technique; Kubernetes; container monitoring; resource monitoring; multi-cluster; Docker

云计算日益成为信息社会的基础设施, 微服务和容器的应用越来越广泛. 为了让容器按照计划有组织地运行并进行合理的资源调度与分配, 容器编排工具由此产生. Kubernetes[1] 是 Google 开源的容器编排工具, 目前使用率在所有容器编排工具中可达 75%[2]. 随着业务的快速增长, 工业界使用 Kubernetes 部署大规模集群时, 其规模可达到节点数以万计, Pod (Kubernetes 创建或部署的最小基本单位, 代表集群上正在运行的一个进程) 数以十万计. 为了能够实时掌握 Kubernetes 集群的资源使用情况与工作状态, 监控工具必不可少, 对于大规模多集群部署情况, 更是迫切需要及时获取多集群的资源监控信息, 实现有效的运维和管理.

Kubernetes 自身提供一定程度的资源监控, 通过部署 metrics-server[3] 和 Dashboard[4] 能够可视化地展示集群中节点级别和 Pod 级别的 CPU 与内存使用情况, 然而与网络和存储相关的指标并未涉及, 并且无法获

① 基金项目: 上海市科学技术委员会 2020 年度“科技创新行动计划”高新技术领域项目 (20511102502)
收稿时间: 2021-10-09; 修改时间: 2021-11-08; 采用时间: 2021-11-12; csa 在线出版时间: 2022-05-30

取容器级别的资源使用情况. 因此, 业界出现了许多适配容器的监控工具, 典型的如 Heapster[5,6] 是原生集群监控方案, 但后被弃用; cAdvisor[7] 兼容 Docker, 现已内嵌到 Kubernetes 作为监控组件, 提供独立的 API 接口; Prometheus[8] 是开源的业务监控和时序数据库, 属于新一代云原生监控系统. 而传统的主机监控工具, 如 Zabbix[9]、Nagios[10] 等, 也提供了 Kubernetes 相关的监控插件[11,12], 能够识别集群的组件部署状态.

当前, Prometheus 是主流的容器监控工具, 在所有的用户自定义指标中, 平均有 62% 由 Prometheus 提供[2]. 通过对集群部署 node-exporter 和 kube-state-metrics, 用户可采集集群中的监控指标. 文献 [13] 提出 Kubernetes 监控体系下的 3 种监控指标: 宿主机监控数据、Kubernetes 组件的 metrics API 以及 Kubernetes 核心监控数据, 其中宿主机监控数据可以通过部署 Prometheus 获取. 目前, 有不少工作将 Prometheus 作为监控模块的核心工具使用, 如搭建云平台监控告警系统[14]、设计实现基于 Kubernetes 云平台弹性伸缩方案[15] 等. 然而, 目前 Prometheus 提供的监控方式对于多集群的情况适配性不佳. 如果需要进行多集群监控, 有两种解决的办法, 一是对每个集群部署 Prometheus, 再进行指标聚合, 这种方式每个集群都要消耗资源开销, 因此整体资源开销相对较大; 二是全局只部署一套 Prometheus, 统一采集多个集群的指标, 这种方式需要修改配置文件中的代码, 较为复杂且容易出错.

由此, 本文提出一种面向 Kubernetes 的资源监控方案并基于 Java 语言实现, 可实时获取多集群多层级的资源监控指标, 设计简化了集群配置难度, 具有良好的可扩展性和灵活性.

1 Kubernetes 容器资源监控指标介绍

Kubernetes 自身提供集群相关的指标, 可以通过 API Server 提供的 REST 接口获取. 为了保证集群的安全性, Kubernetes 默认使用 HTTPS (6443 端口) API, 需要进行认证才能访问集群接口, 认证方式有账户密码认证、证书认证以及 token 认证等.

Kubernetes 的资源监控指标[16] 分为系统指标和服务指标. 系统指标是集群中每个组件都能够采集到的指标, 其中能够被 Kubernetes 自身理解并用于了解自身组件与核心的使用情况、作为做出相应指令的依据的指标, 称为核心指标, 包括 CPU 的累计使用量、内存的当前使用量以及 Pod 和容器的磁盘使用量; 其余指标统称为非核心指标. 服务指标则是 Kubernetes 基础设施组件以及用户应用提供的指标, 其中用于 Kubernetes 的 Pod 自动水平扩展的指标也可以被称为自定义指标.

Kubernetes 发展至今, 向用户提供了以下几类指标接口: (1) “stats”, 该接口相对较老, 可以查询具体某个特定的容器下的指标数据; (2) “metrics.k8s.io”, 该接口由 metrics-server 提供, 为 kube-scheduler、Horizontal Pod Autoscaler 等核心组件以及“kubectl top”命令和 Dashboard 等 UI 组件提供数据来源; (3) “metrics”和“metrics/cadvisor”, 分别提供了 Kubernetes 自身的监控指标以及内嵌的 cAdvisor 获取的监控指标, 数据格式适配 Prometheus 监控工具; (4) “stats/summary”[17], 该接口是 Kubernetes 社区目前主推的数据接口. “stats”已被 Kubernetes 现版本废弃, 其余 3 类接口能够提供的指标种类与对应层级如表 1 所示.

表 1 接口指标种类与对应层级

接口名称	指标种类	指标层级
metrics.k8s.io	CPU	节点、Pod
	内存	节点、Pod
metrics/cadvisor	CPU	节点、Pod、容器
	内存	节点、Pod、容器
	网络	节点、Pod
	存储	节点、Pod、容器
stats/summary	CPU	节点、Pod、容器
	内存	节点、Pod、容器
	网络	节点、Pod
	存储	节点、Pod、容器

Kubernetes 监控指标类型主要有以下 4 种: counter、gauge、histogram 和 summary. counter 是只增不减的计数器, 可以用于统计某种资源的累计消耗或者累计时间; gauge 用于那些具有增减变化的指标, 比如当前某种资源的利用率或者可用量等; histogram 表示条形直方统计图, 可以表示数据的分布情况, 比如某个时间段内的请求耗时分布; summary 类似与 histogram, 但是用于标识分位值, 根据分位值显示数据的分布情况.

2 容器资源监控的总体设计

本文设计实现的面向 Kubernetes 的多集群资源监控分为 4 个模块: 集群管理模块, 数据采集模块、数据处理模块以及外部接口模块. 集群管理模块用于配置

集群与检测集群连通性; 采集模块用于采集集群指定的接口数据与定时采集的设置; 数据处理模块用于接口数据的解析、指标的提取与计算、数据格式的规范化以及数据的存储; 接口设计模块用于提供给可视化界面数据接口.

整体模块功能示意图如图 1 所示: ① 外部访问与接口的交互, 包括配置集群与获取监控指标; ② 通过外部接口将集群配置数据传递给集群管理模块; ③ 集群管理模块与数据库集群配置表交互, 针对集群配置表进行增删改查; ④ 集群管理模块将集群配置信息传递给数据采集模块; ⑤ 数据采集模块通过访问 API Server 接口获取集群的资源监控指标; ⑥ 将采集到的资源监控指标送入数据处理模块进行数据处理; ⑦ 将处理完毕的数据存储至数据库的资源监控指标表中; ⑧ 提供获取资源监控指标的外部接口; ⑨ 提供获取集群配置的外部接口; ⑩ 提供获取 Kubernetes 集群组件状态信息的外部接口.

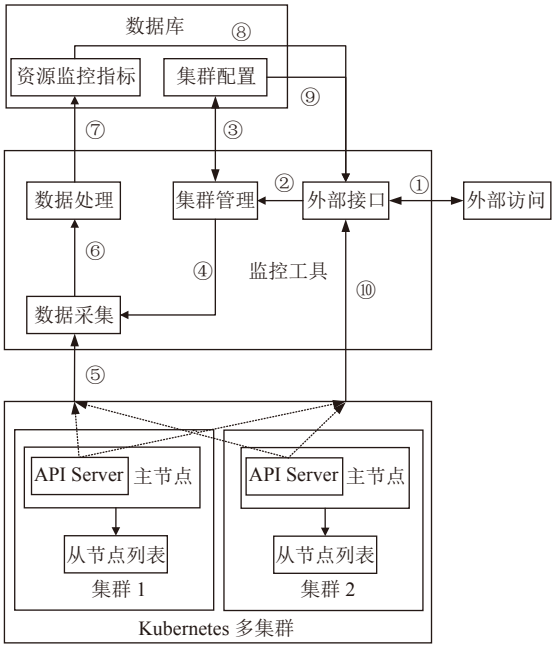

图 1 面向 Kubernetes 的多集群资源监控模块功能示意图

3 容器资源监控的实现

3.1 集群管理

集群管理分为集群配置管理和集群连通性测试. 集群配置管理主要的功能是管理集群的配置信息, 包括集群名称 (用户根据需求自定义)、集群 API Server 的端口地址 (默认为集群的主节点的 6443 端口) 和相应的 token (用于认证, 须预先在集群中配置好), 为防止数据重复采集, 集群名称、端口地址须保证唯一. 为了验证数据配置的准确性, 提供集群连通性测试的功能, 根据端口地址和 token 创建一个 API 用户 (ApiClient), 使用这个用户去访问该集群的某个 API 接口, 根据是否能够正常返回接口数据来判断是否成功连通集群.

3.2 资源数据采集

资源数据的采集分为采集接口的确定, 采集算法的设计与定时采集的设计.

本设计对以下 3 类接口[18] 进行数据采集: “api/v1”接口获取核心的集群指标, 包括集群的节点、Pod、命名空间、服务等架构资源的数据; “apis/”接口获取集群相关部署情况的指标, 如 Daemon Sets、Deployments、Replica Sets 等; “stats/summary”接口获取集群资源使用情况的监控指标, 具体指标详情见表 2. 需要注意的是, 表 1 中网络指标提供的是端口 (interface) 级别的数据指标, 因此需要针对节点和 Pod 的每个端口进行数据处理与存储; 存储指标根据层级使用了不同的名称 (节点为“fs”, Pod 为“volume”和“ephemeral-storage”, 容器为“rootfs”). 另外, 每个 Pod 可以挂载多个 volume, 每个 volume 都有 Pod 内唯一的名称, 因此对每个 volume 都要单独生成一条数据. 每个指标都会有自己的生成时间, 这是因为每个指标是独立生成的, 从接口中获取的数据也并非是实时数据, 而是数据接口最近一次刷新后的数据, 为保证二次计算的准确性, 把该字段单独进行存放.

3 类接口中, “api/v1”和“apis/”提供集群级别的数据, 直接采集接口数据即可, 而“stats/summary”提供节点级别的数据, 因此需要先获取集群的节点列表后对每一个节点进行数据的提取. 为保证采集效率, 设计采用多线程并行采集.

对于定时采集, “api/v1”和“apis/”提供的数据主要是 Kubernetes 集群的部署或者配置资源, 因此不进行定时采集. 而“stats/summary”获取的是实时的集群各资源使用量的情况, 需要进行定时采集. 由于 Kubernetes 接口的刷新间隔最短是 10 s, 因此采集间隔最好长于刷新时间, 防止重复采集数据. 本设计定为 1 分钟采集 1 次.

3.3 资源数据处理与存储

由于采集的数据包含 4 个种类和 3 个层级的数据,

因此需要对每条数据进行种类和层级的明确区分. 并且, 这些数据需要进行结构的解析、字段的提取、数值的计算, 将数据结构统一化、扁平化后再进行数据存储.

表 2 监控指标详情

指标种类	指标名称	指标解释
CPU	usageNanoCores	当前使用的核数 (单位: 1E–9个)
	usageCoreNanoSeconds	累计使用时间 (单位: 1E–9 s)
内存	availableBytes	当前内存可用量
	usageBytes	当前内存使用量
	workingSetBytes	当前内存工作集使用量
	rssBytes	当前常驻内存集使用量
	pageFaults	累计页错误数
	majorPageFaults	累计主要页错误数
网络	rxBytes	网络接收累计字节数
	rxErrors	网络接收累计错误数
	txBytes	网络发送累计字节数
	txErrors	网络发送累计错误数
存储	availableBytes	当前存储可用量
	capacityBytes	存储总量
	usedBytes	当前存储使用量
	inodesFree	可用Inode数
	inodes	总Inode数
	inodesUsed	使用的Inode数

3.3.1 资源数据的处理

数据处理的主要工作包括 3 部分: 一是数据封装, 提取关键字段; 二是根据层级解析数据; 三是对部分数据进行二次计算, 得到更直观的数据.

3 类接口中, Java 相关类库已将“api/v1”和“apis/”的所需的接口数据封装成对象, 通过现有方法提取关键字段即可; 针对不同层级的数据, Kubernetes 提供了不同的接口, 无需额外区分层级; 对于数据本身, 接口提供的是集群具体组件 (Pod、容器、部署任务等) 的状态信息, 无需进行数值上的计算. 故这两类接口的数据处理工作相对简单. 而“stats/summary”接口仅提供了获取数据接口的方法, 并没有对数据结构进行封装; 提供的数据包含节点、Pod、容器级别的数据, 需要对数据根据层级进行区分; 接口数据包含 gauge 类数据和 counter 类数据, 需要对部分数据进行二次计算.

“stats/summary”接口数据结构如图 2 所示, 每类指标详情参考表 2, 数据格式为 JSON, 可以使用 JSON 解析工具 (如“json2pojo”) 将其封装成 Java 对象.

为了使数据扁平化便于存储, 所有层级的数据均添加集群、节点、命名空间、Pod、容器级别的字段, 对于高层级数据中的低层级字段默认设置为空. 除此以外, 数据中还添加一个“层级”字段, 用于表明数据的层级, 保证不同层级的数据不会在进行数据查询时相互污染查询结果.

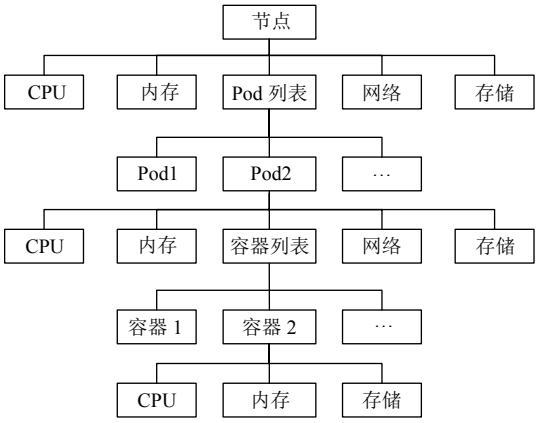

图 2 “stats/summary”接口数据结构图

另外, 针对网络的多接口添加“接口名”字段用于存放同一组件多个端口的数据, 针对存储的不同类型添加“存储类型”字段, 其中为 volume 类型的额外添加“volume 名称”字段用于保存同一组件中多个挂载 volume 的数据.

表 2 中显示的资源指标中, counter 类型的指标和部分 gauge 类型的指标需要进行二次计算, 根据计算产生的新指标与计算方式如表 3 所示. 由于采集可能因为集群通讯故障等原因导致采集的数据可能跨越多个时间间隔, 因此, 为了能够表现监控数据的可靠性, 表 3 中的部分指标可以进行再计算 (如单位时间内的平均网络流量或者多个时间段中的内存使用量的峰值谷值等).

3.3.2 资源数据的存储

对于“api/v1”和“apis/”的数据, 只需要按需求获取接口数据即可, 无需进行数据存储. 本节中只关注“stats/summary”接口的数据存储工作.

整个资源数据根据资源类型分为 4 类表: CPU 表、内存表、网络表、以及存储表, 每类表分为 3 张表: 原始数据表 (metadata), 数据表 (data) 和最近一次数据表 (last_data).

metadata 表用于存储采集数据未处理过的指标, data 表用于存储二次处理的字段, 其中包含了 metadata

表中需要计算得到的指标以及需要转换单位格式的指标, 这张表将主要用于资源数据的展示与时间序列数据列的提取和分析; last_data 表中存放的是各组件最近一次采集到的指标数据, 这张表的作用一是参与 data 表字段数据的生成, 主要针对 metadata 表中 counter 类型的数据, 生成相邻时间戳间隔的指标数据; 二是用于资源数据的展示. 以容器为例, 由于容器的生命周期十分短暂, 当一个容器因为故障或者达到使用寿命后, Kubernetes 会将这个容器杀死, 然后根据相同的镜像生成一个新的容器继续维持服务的运作, 因此, 在 Kubernetes 集群中无法查找曾经生成过的容器. 尽管可以通过 data 表来获取历史组件信息, 但是表数据量的逐渐增大会逐渐降低查询效率. 通过 last_data 表, 便可以获取到历史中某个集群所有能够查询到监控数据的组件的列表, 根据这个列表就能够获取具体某个时间段中集群内存活的组件信息 (包括节点, Pod, 容器), 并且数据增长速率相较于其余两种表非常低, 可以保证查询的效率.

除此以外, 每条数据再插入以下 2 个字段: “id”字段作为主键, 用于记录数据的序号, 便于对数据进行排序; “create_time”字段用于记录插入当前数据的时间, 这个字段主要用于删除超过存放时限的数据. 对于 4 张最近一次数据表中, 额外添加了“update_time”字段, 用于记录最近一次数据表的时间, 那么就可以通过“create_time”和“update_time” 2 个字段表示某个组件资源数据的始末时间.

表 3 监控新指标详情

指标种类	指标名称	指标解释	计算方式
CPU	usageCores	当前使用的核数 (个)	usageNanoCores/1E9
	usageCorePercent	当前CPU使用率	usageCores/总CPU核数
	usageCoresSecondsPeriod	时间间隔内使用CPU的时间 (s)	(usageCoresNanoSeconds (本次) –usageCoresNanoSecond (上次))/1E9
内存	availablePercent	当前内存可用率	availableBytes/内存总量
	usagePercent	当前内存使用率	usageBytes/内存总量
	workingSetPercent	当前内存工作集使用率	workingSetBytes/内存总量
	rssPercent	当前常驻内存集使用率	rssBytes/内存总量
	pageFaultsPeriod	时间间隔内页错误数	pageFaults (本次)–pageFaults (上次)
	majorPageFaultsPeriod	时间间隔内主要页错误数	majorPageFaults (本次)–majorPageFaults (上次)
网络	rxAmount	时间间隔内网络接收字节数	rxBytes (本次)–rxBytes (上次)
	rxErrorsPeriod	时间间隔内网络接收错误数	rxErrors (本次)–rxErrors (上次)
	txAmount	时间间隔内网络发送字节数	txBytes (本次)–txBytes (上次)
	txErrorsPeriod	时间间隔内网络发送错误数	txErrors (本次)–txErrors (上次)
存储	availablePercent	当前存储可用率	availableBytes/capacityBytes
	usedPercent	当前存储使用率	usedBytes/capacityBytes
	inodesFreePercent	可用Inode数	inodesFree/inodes
	inodesUsedPercent	使用的Inode数	inodesUsed/inodes

3.3.3 资源数据定时任务算法

第 3.2 节中提到“stats/summary”接口提供的监控数据需要进行定时采集、处理和存储, 本文将这 3 个部分统合成一个定时任务来实现.

算法 1 给出任务的整体功能实现伪代码. 根据集群列表获取各集群的节点信息, 接着按节点依次获取监控数据, 根据数据结构进行分层, 再按照数据种类进行处理与存储. 算法中, 将同一类数据的处理与存储封装成一个服务 (service) 用于优化代码格式; 实际实现中, 在所有的 for 循环中均使用了多线程用于提高算法运行的效率.

算法 1. 资源数据定时任务

```
1.  function statsSummaryJob()
2.    从配置表中获取 Kubernetes 集群列表 clusterList
3.    for cluster in clusterList
4.      ip ← cluster.ip
5.      port ← cluster.port
6.      token ← cluster.token
7.      host ← ip + ":"+port
8.      //创建访问 API Server 的客户端
9.      apiClient ← createApiClient(host, token)
10.      //获取当前集群的节点列表
11.      nodeList ← getNodeList(apiClient)
12.      for node in nodeList
```

```
13.          //获取节点的监控数据
14.          i ← getStatsSummary(apiClient, node)
15.          //将数据解析成 json
16.          toJson(nodeStatsSummaryData)
17.          //对节点级别的 4 类数据进行处理与存储
18.          cpuService(i.getCpu())
19.          memoryService(i.getMemory())
20.          networkService(i.getNetwork())
21.          fsService(i.getFs())
22.          //获取 Pod 数据的列表, 存放在 i 中
23.          podStatsSummaryList ← i.getPodList()
24.          for j in podStatsSummaryList
25.            //对 Pod 级别的 4 类数据进行处理与存储
26.            cpuService(j.getCpu())
27.            memoryService(j.getMemory())
28.            networkService(j.getNetwork())
29.            fsService(j.getEphemeralStorage())
30.            fsService(j.getVolume())
31.            //获取容器数据的列表, 存放在 j 中
32.            containerStatsSummaryList ← j.getContainerList()
33.            for k in containerStatsSummaryList
34.              //对容器级别的 3 类数据进行处理和存储
35.              cpuService(k.getCpu())
36.              memoryService(k.getMemory())
37.              fsService(k.getRootfs())
38.            end for
39.          end for
40.        end for
41.      end for
42.    end function
```

对于处理 4 类资源的服务 (service), 主要实现的功能是第 3.3.2 节中 3 张表字段的提取与计算、表数据的添加与更新的功能, 伪代码见算法 2.

算法 2. 资源服务 (包含 CPU、内存、网络、存储)

```
输入: 指标原始数据
1.    function service(metadata)
2.      //将原始数据插入 metadata 表中
3.      addMetadata(metadata)
4.      //从 lastData 表中取出该组件的最近一次数据
5.      lastData ← findLastData(metadata)
6.      //处理 gauge 类数据
7.      gaugeData ← calGauge(metadata)
8.      if lastData != null then
9.        //处理 counter 类数据
10.         counterData ← calCounter(metadata, lastData)
11.         totalData ← gaugeData+counterData
12.         //存储处理后的数据到 data 表中
13.         addData(totalData)
14.         //更新最近一次数据
15.         updateLastData(metadata)
16.       else
17.         totalData ← gaugeData
18.         addData(totalData)
19.         //将此数据添加作为最近一次数据
20.         addLastData(metadata)
21.       end if
22.   end function
```

4 类数据均适用算法 2, 不过网络和存储类数据因为第 3.3.1 节中提到的注意事项, 需要添加一些循环来满足功能需求, 具体算法的实现过程基本一致.

3.4 外部访问接口设计

外部访问接口分为 3 类, 第 1 类用于集群配置的增删改查; 第 2 类用于集群组件信息的查询, 如查询集群的节点或者 Pod 信息等, 向用户提供对应接口字段精简后的数据; 第 3 类用于查询定时采集得到的监控指标, 比如某个容器最近 1 小时的 CPU 使用情况等. 具体实现的接口见表 4.

表 4 接口设计列表

接口类	接口名称	接口用途
集群配置	get-k8s-list	获取集群列表
	add-k8s-server	添加一个集群
	del-k8s-server	删除某个集群
	update-k8s-server	修改某个集群的配置
	find-k8s-server-by-id	获取某个集群的配置
集群组件信息	get-node-api	获取集群节点信息
	get-pod-api	获取集群Pod信息
	get-namespace-api	获取集群命名空间信息
	get-service-api	获取集群服务信息
	get-endpoint-api	获取集群Endpoint信息
	get-daemonset-api	获取集群Daemon Set信息
	get-deployment-api	获取集群Deployment信息
	get-replicaset-api	获取集群Replica Set信息
	get-job-api	获取集群Job信息
监控指标查询	get-cpu-metadata	获取CPU原始表数据
	get-cpu-data	获取CPU表数据
	get-memory-metadata	获取内存原始表数据
	get-memory-data	获取内存表数据
	get-network-metadata	获取网络原始表数据
	get-network-data	获取网络表数据
	get-fs-metadata	获取存储原始表数据
	get-fs-data	获取存储表数据

除此以外, 本文设计了 2 个参数模板用于传递接口参数, 用户可以根据查询的需求添加相应的参数, 其中“K8sServer”用于用户交互与 API Server 类接口, “K8sConfig”用于数据库类接口. 模板具体字段见表 5.

对于监控指标查询, 由于集群之间逻辑上互相隔离, 集群间的监控数据基本没有逻辑关系, 并且多集群

的监控数据量很大, 会大大影响查询的效率, 因此接口均基于某个特定的集群进行数据查询.

表 5 参数模型设计

名称	参数	参数含义
K8sServer	id	集群编号
	name	集群名称
	host	集群地址
	token	集群token
	status	集群连通状态
K8sConfig	level	数据层级
	clusterId	集群id
	clusterName	集群名称
	nodeName	节点名称
	namespace	命名空间
	podName	Pod名称
	containerName	容器名称
	networkName	网络名称 (仅网络表)
	interfaceName	网络端口名称 (仅网络表)
	fsType	存储类型 (仅存储表)
	volumeName	volume名称 (仅存储表)
	startTime	起始时间
	endTime	结束时间
	limit	数据条数限制
	order	数据排序方式

4 实验设计

本次实验使用到 4 个 Kubernetes 集群, 具体配置如表 6 所示, 其中集群 1、2 操作系统为 CentOS 7 , 集群 3、4 操作系统为 CentOS 8.

其中 172.16 与 10.168 两个子网能够相互通信, 监控工具的 IP 为 172.16.2.183. 本次性能测试实验主要测试集群资源开销和访问接口延时.

4.1 集群资源开销

由于本文设计的面向 Kubernetes 多集群资源监控仅涉及到 API 接口的访问, 因此只需关注部署在集群内部的 API Server 的资源消耗情况. 实验首先采集 4 个小时平时资源使用情况的数据, 然后使用压测工具 Tsung, 以每秒 10 次的频率 (高于定时任务的采集频率) 访问 4 个小时, 获取这段时间的资源使用情况, 具体数据分别见表 7 和表 8.

表 6 集群信息

集群编号	IP地址	节点数量	配置信息
1	172.16.100.1	3	主: 4 CPU + 8 GB内存 从: 4 CPU + 4 GB内存
2	172.16.100.10	4	主: 4 CPU + 8 GB内存 从: 4 CPU + 4 GB内存
3	10.168.1.139	2	主从: 3 CPU + 4 GB内存
4	10.168.1.177	2	主从: 3 CPU + 4 GB内存

表 7 平时集群 API Server 资源使用情况 (%)

集群编号	最大CPU	最小CPU	平均CPU	最大内存	最小内存	平均内存
1	2.90	1.89	2.23	4.55	4.50	4.53
2	10.44	2.32	4.42	4.82	4.81	4.81
3	3.08	1.98	2.39	9.87	9.87	9.87
4	4.24	2.90	3.38	12.91	12.90	12.90

表 8 模拟访问时集群 API Server 资源使用情况 (%)

集群编号	最大CPU	最小CPU	平均CPU	最大内存	最小内存	平均内存
1	13.17	2.13	3.83	4.85	4.47	4.67
2	10.60	2.20	4.02	5.11	5.00	5.05
3	3.64	1.96	2.63	10.13	9.99	10.03
4	5.30	3.77	4.30	14.27	14.18	14.22

可以看出, 模拟访问期间相较于空闲时集群的 CPU、内存的 使用量均有一定程度的提升, 但是平均使用量幅度不超过 2%, 可以看出这种访问方式对于集群的影响是较小的.

4.2 访问接口延时

对于接口延时, 设计在定时任务触发后 30 s 对接口进行数据访问, 分别采集集群级别、节点级别、Pod 级别、容器级别最近 1 个小时的接口数据, 定时任务和数据访问频率均为 1 次/min. 接口数据的格式如图 3 所示.

图 3 接口数据展示

接口累计采集时间为 2 235 min, 单表数据条数从 0 至 35 万余条, 展示的数据为近 10 min 的平均延时, 延时单位为 ms, 具体的延时情况见表 9.

接口延时的变化情况有如下几个原因导致: 一是表数据量的增长增加了检索数据的时间, 二是采集的数据量的变化, 由于获取集群级别的数据量大于节点级别的数据量, 其得到的接口延时也相对更高, 三是网络本身, 由于网络流量的波动, 会在一定程度上影响接

口的访问延时, 在计算 2 160 min 处节点级别的 10 条延时数据中, 最短延时为 1 981 ms, 而最长的则为 5 450 ms, 可见其影响程度.

除了数据本身与网络的影响, 数据库也是其中的影响之一, 本次设计采用的是 PostgreSQL 进行数据的存取, 针对 Kubernetes 监控数据量大、结构单一、时间属性强的数据在性能上显得有些不足. 如果使用针对性的时序数据库, 如 InfluxDB, 可以提高整个数据的存取性能.

表 9 监控数据接口延时 (ms)

累计时间 (min)	集群	节点	Pod	容器
10	527.5	125.4	72.7	94.9
30	1 266.8	126.5	203.2	186.0
60	2 287.8	274.6	352.7	503.3
180	2 344.6	605.4	500.7	699.3
360	2 196.8	717.8	715.5	905.8
540	2 961.7	1 250.8	1 616.3	1 597.7
720	3 262.4	1 196.6	1 366.3	2 250.2
1 440	5 145.2	3 054.1	3 335.5	3 863.7
2 160	6 251.5	4 132.7	4 243.7	4 917.1

5 结束语

本文基于 Java 语言提出了一种面向 Kubernetes 的多集群资源监方案, 实现了针对多个 Kubernetes 集群的资源监控, 此方案具有良好的可扩展性和灵活性, 且实验表明对集群资源的消耗低. 下一步的研究方向是采用时序数据库来优化接口性能, 还可以设计日志监控与告警功能来实现容器级别的立体化监控.

参考文献

1 The Kubernetes Authors. Kubernetes. https://kubernetes.io/. [2021-10-09].

2 Sysdig. Sysdig 2021 container security and usage report. https://dig.sysdig.com/c/pf-2021-container-security-and-usage-report?x=u_WFRi&utm_source=gated-organic&utm_medium=website. [2021-10-09].

3 Kubernetes SIGs. Metrics-server: Scalable and efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines. https://github.com/kubernetes-sigs/metrics-server. [2021-10-09].

4 Kubernetes. Dashboard: General-purpose web UI for Kubernetes clusters. https://github.com/kubernetes/dashboard. [2021-10-09].

5 Kubernetes Retired. Heapster: Compute resource usage analysis and monitoring of container clusters. https://github.com/kubernetes-retired/heapster. [2021-10-09].

6 Jimmy Song. Heapster. https://jimmysong.io/kubernetes-handbook/practice/heapster.html. [2021-10-09].

7 Google. cadvisor: Analyzes resource usage and performance characteristics of running containers. https://github.com/google/cadvisor. [2021-10-09].

8 Prometheus Authors. Prometheus–monitoring system & time series database. https://prometheus.io/docs/prometheus/latest/. [2021-10-09].

9 Zabbix LLC. ZABBIX: The enterprise-class open source network monitoring solution. https://www.zabbix.com/. [2021-10-09].

10 Nagios Enterprises, LLC. Nagios: The industry standard in IT infrastructure monitoring. https://www.nagios.org/. [2021-10-09].

11 Vitaly Agapov. Check_kubernetes: Nagios/Icinga/Zabbix style plugin for checking Kubernetes. https://github.com/agapoff/check_kubernetes. [2021-10-09].

12 Miller J. Kubernetes-nagios: Basic health checks for a Kubernetes cluster. https://github.com/colebrooke/kubernetes-nagios. [2021-10-09].

13 魏饴. 基于 Kubernetes 的监控和调度技术研究 [硕士学位论文]. 广州: 华南理工大学, 2020.

14 郝鹏海, 徐成龙, 刘一田. 基于 Kafka 和 Kubernetes 的云平台监控告警系统. 计算机系统应用, 2020, 29(8): 121–126. [doi: 10.15888/j.cnki.csa.007611]

15 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. 计算机工程, 2021, 47(1): 312–320. [doi: 10.19678/j.issn.1000-3428.0056560]

16 Kubernetes. Community: Kubernetes community content. https://github.com/kubernetes/community. [2021-10-09].

17 Splunk. Kubernetes network stats. https://docs.splunk.com/Observability/gdi/kubelet-stats/kubelet-stats.html#metrics. [2021-10-09].

18 The Kubernetes Authors. Kubernetes API reference docs. https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.21/. [2021-10-09].

(校对责编: 孙君艳)