学校代码 10702 密级 公 开

中图分类号 TP311.1 学号 1916310782

专业硕士学位论文

基于战场分析系统的 K8s

云平台设计与实现

学位申请人： 郭 玥 鑫

校 内 导 师 ： 高 武 奇 副 教 授

企 业 导 师 ： 赵 锋 高 工

专 业 领 域 ： 兵 器 工 程

学 位 类 别 ：

工 程

2022 年 5 月

**专业硕士学位论文**

**（学 位 研 究 生）**

**题目：**基于战场分析系统的 K8s

云平台设计与实现

**作者 郭玥鑫**

**校内导师 高武奇 专业技术职务 副教授企业导师 赵 锋 专业技术职务 高 工**

**专业领域**

**兵器工程**

**西安工业大学**

**2022 年 5 月 中国.西安**

**Design and Implementation of K8s(Kubernetes) Cloud Platform Based on**

**Battlefield Analysis System**

by

*Guo Yuexin*

Thesis Submitted to the faculty of the Xi’an Technological University in

Partial Fulfillment of

the Requirements for the Degree of

**MASTER**

in

Arms Engineering

Supervisor: Associate Professor Gao Wuqi Supervisor: Senior Engineer Zhao Feng

**Xi’an Technological University**

May 2022

Xi’an, Shaanxi, P.R.China

**摘** **要**

信息化作战的升级换代对军事云平台提出了更高的要求，当前云计算技术对战场分析系统（即云平台中的战场分析类应用）的运行主要存在两个问题：第一是资源分配方案不灵活，资源利用率低；第二是数据缺乏安全性，应用也面临中断的威胁。因此本课题基于K8s(Kubernetes)容器集群虚拟化技术和 Fabric 区块链技术，研发面向战场分析应用的 K8s云平台，可有效解决上述问题，实现资源的有效利用、应用的连续运行和数据的可信存储。

论文设计了适合战场分析类应用的 K8s 云平台。该平台通过改进的资源调度策略和面向 K8s 的容器应用迁移实现高可用；通过基于 K8s 的 Fabric 应用实现关键数据高可信存储；采用 Golang 语言进行平台功能的二次开发实现可视化管理。主要研究内容如下：

首先，在分析并运行 K8s 容器集群默认资源调度策略的基础上，针对战场分析应用的资源需求与特点，从突出 Pod 优先级、GPU 资源、显存资源三个方面对调度策略进行改进，并综合考虑应用需求与节点资源的匹配程度，实现云平台中节点资源的充分利用和集群的负载均衡。

其次，研究了虚拟机在线迁移方法和单个主机间容器迁移方法，对面向集群的容器应用迁移方法进行优化，利用 Docker 分层特点对文件系统进行对比迁移，减少了冗余数据的传输；优化内存停机迁移的条件，避免无限制迭代的问题。

然后，研究了 Fabric 区块链的底层技术，实现了基于 K8s 的区块链应用，使基础组件随网络直接启动，可扩展的链码通过控制组件按需启动，完成了关键数据的可信存储。

最后，采用 Golang 语言实现云平台的功能开发，包括平台总览、应用管理、镜像管理以及数据可信管理四大模块，为用户提供可视化操作界面以方便使用。

经过平台功能测试与性能分析，本课题设计的基于战场分析应用的 K8s 云平台功能完善，运行稳定。改进的调度器提升了单个节点和集群的负载均衡效率；容器迁移在传输的数据量、迁移时间和宕机时间三方面的性能都得到了提升。平台从功能和性能上都能够满足实际需求，在战场分析仿真场景下实现了一定的应用创新，具备良好的应用价值。

**关键词：**K8s 云平台；资源调度；容器迁移；Fabric 区块链

**Abstract**

The upgrading of information-based operations has put forward higher requirements for military cloud platforms. There are two main problems in the operation of the current cloud computing technology for the battlefield analysis and system(that is, the battlefield analysis application in cloud platform): The first is that resource allocation scheme is inflexible and the resource utilization rate is low; The second is the lack of data security, and the threat of interruption to applications. Therefore, this paper based on K8s(Kubernetes) container cluster virtualization technology and Fabric blockchain technology, develop K8s cloud platform based on battlefield analysis applications. The platform can effectively solve the above problems, realize the effective utilization of resources, continuous operation of applications and reliable storage of data.

This paper designs a K8s cloud platform suitable for battlefield analysis applications. The platform achieves high availability by improving resource scheduling strategies and K8s-oriented container application migration; Realize high-trust storage of key data through K8s-based Fabric applications; Uses Golang language for secondary development of platform functions to achieve visual management. The main research contents are as follows:

Firstly, on the basis of analyzing and running the default resource scheduling policy of K8s container cluster, In view of the resource requirements and characteristics of battlefield analysis applications, this paper improves the scheduling strategy from three aspects: Pod priority, GPU resources, and video memory resources; and comprehensively considers the degree of matching between application demand and node resource, So as to realize the full utilization of node resources in the cloud platform and the load balancing of the cluster.

Secondly, the online migration method of virtual machine and the container migration method between single hosts are studied. The migration method of container application for cluster is optimized, and the file system is compared and migrated by using the layered characteristics of docker to reduce the transmission of redundant data; Optimize the conditions of memory downtime migration to avoid the problem of unlimited iteration.

Then, the underlying technology of Fabric blockchain is studied, and the K8s-based blockchain application is implemented. So that the basic components can be started directly with the network, and the scalable Chaincode can be started on demand through the control components, which further completes the trusted storage of key data.

Finally, Golang language is used to realize the function development of cloud platform, including four modules: platform overview, applications management, images management and

data trust management, providing users with a visual operation interface for easy use.

After the functional test and performance analysis of the platform, the K8s cloud platform based on the battlefield analysis application designed in this paper has perfect functions and runs stably. The improved scheduler improves the load balancing efficiency of individual nodes and clusters; Container transfer in the amount of data transferred, transfer time and downtime has improved the performance of the three aspects. The platform can meet the actual demand in terms of function and performance, and has achieved certain application innovation in battlefield analysis and simulation scenarios, which has good application value.

**Key words:** K8s cloud platform; resource schedule; container migration; Fabric blockchain

目 录

1. [绪 论 1](#_TOC_300001)
   1. [研究背景及意义 1](#_TOC_300002)
   2. [国内外研究现状 2](#_TOC_300003)
      1. [K8s 资源调度国内外研究现状 2](#_TOC_300004)
      2. [Docker 容器迁移国内外研究现状 3](#_TOC_300005)
      3. [基于 K8s 的区块链国内外研究现状 4](#_TOC_300006)
   3. [论文主要研究内容 5](#_TOC_300007)
   4. [论文组织结构 6](#_TOC_300008)
2. [平台总体设计 8](#_TOC_300009)
   1. [战场分析云平台需求分析 8](#_TOC_300010)
   2. [云平台方案设计 9](#_TOC_300011)
      1. [云平台总体架构设计 9](#_TOC_300012)
      2. [云平台功能模块设计 10](#_TOC_300013)
   3. [相关技术基础 11](#_TOC_300014)
      1. [K8s 架构 11](#_TOC_300015)
      2. [Docker 迁移技术 13](#_TOC_300016)
      3. [Fabric 区块链基础 14](#_TOC_300017)
   4. [本章小结 17](#_TOC_300018)
3. [基于战场分析的 K8s 资源调度优化设计与实现 18](#_TOC_300019)
   1. [K8s 调度策略研究 18](#_TOC_300020)
      1. [K8s 默认调度概述 18](#_TOC_300021)
      2. [K8s 默认调度算法研究 19](#_TOC_300022)
   2. [基于战场分析的 K8s 资源调度优化设计 20](#_TOC_300023)
      1. [问题分析 20](#_TOC_300024)
      2. [优化预选阶段 20](#_TOC_300025)
      3. [优化优选阶段 21](#_TOC_300026)
   3. [优化调度流程与实现 22](#_TOC_300027)
      1. [优化调度整体流程 22](#_TOC_300028)
      2. [优化调度器实现原理 24](#_TOC_300029)
      3. [优化预选阶段实现 24](#_TOC_300030)
      4. [优化优选阶段实现 25](#_TOC_300031)
   4. [本章小结 26](#_TOC_300032)
4. [基于 K8s 的容器应用迁移优化设计与实现 27](#_TOC_300033)
   1. [问题分析 27](#_TOC_300034)

i

* 1. [基于 K8s 的容器应用迁移优化设计 27](#_TOC_300035)
     1. [迁移准备阶段设计 27](#_TOC_300036)
     2. [正式迁移阶段设计 28](#_TOC_300037)
  2. [基于 K8s 的容器应用迁移实现 30](#_TOC_300038)
     1. [迁移整体流程 30](#_TOC_300039)
     2. [迁移准备阶段实现 31](#_TOC_300040)
     3. [正式迁移阶段实现 32](#_TOC_300041)
  3. [本章小结 33](#_TOC_300042)

1. [基于 K8s 的 Fabric 区块链高可信设计与实现 34](#_TOC_300043)
   1. [问题分析 34](#_TOC_300044)
   2. [基于 K8s 的 Fabric 区块链高可信设计 35](#_TOC_300045)
      1. [基础组件部署设计 35](#_TOC_300046)
      2. [可扩展链码部署设计 36](#_TOC_300047)
   3. [基于 K8s 的 Fabric 区块链高可信实现 37](#_TOC_300048)
      1. [云化部署整体流程 37](#_TOC_300049)
      2. [基础组件部署实现 37](#_TOC_300050)
      3. [战场分析数据可扩展链码部署实现 39](#_TOC_300051)
   4. [本章小结 41](#_TOC_300052)
2. [平台实现与测试分析 42](#_TOC_300053)
   1. [平台环境配置 42](#_TOC_300054)
   2. [云平台功能测试 43](#_TOC_300055)
      1. [平台总览模块功能测试 43](#_TOC_300056)
      2. [应用部署模块功能测试 44](#_TOC_300057)
      3. [镜像管理模块功能测试 45](#_TOC_300058)
      4. [可信数据管理模块功能测试 46](#_TOC_300059)
   3. [性能测试与结果验证 48](#_TOC_300060)
      1. [云平台应用调度优化效果验证 48](#_TOC_300061)
      2. [云平台容器应用迁移效果验证 50](#_TOC_300062)
   4. [本章小结 52](#_TOC_300063)
3. [结论与展望 53](#_TOC_300064)
   1. [结论 53](#_TOC_300065)
   2. [展望 54](#_TOC_300066)

[参考文献](#_TOC_300067) 55

[致 谢](#_TOC_300068) 59

[学位论文独创性与知识产权声明](#_TOC_300069) 60

ii

1. 绪 论
   1. 研究背景及意义

军事应用云平台是一种专用云平台，利用云计算技术获取各参战单位的实时信息，以实现军队跨兵种、跨地域战斗时信息数据的联通与操作。例如在联合作战体系中，使用云平台对战场的实时情报进行综合处理，为指挥提供可信的参考依据[1]。底层的云计算由于资源虚拟化、业务弹性可扩展等优点[2]，被广泛应用于后勤系统、情报获取、指挥决策、数据安全等多个军事领域[3]，不仅是建设军队信息化以及国防现代化的关键技术，更是社会变革与军事变革的新动力。

但由于军队现代化战略的升级换代，各级部队各类信息系统数量呈现出的爆炸式增长对云平台的可用性和可靠性提出了更高的要求。一方面从装备发展与应用部署角度来说，武器装备越来越智能化，装备威胁度也不断增强；大量战场分析应用的部署与运行，需要更加灵活的资源分配方案；另一方面从作战环境角度来说，战场综合系统通常工作于各种复杂环境之下，存在一定的信息安全问题，战时还会面临敌方硬摧毁的威胁。这些都严重影响了军事应用云平台的使用效果。因此，战场分析云平台的构建，既是解决当下作战体系所面临问题的有效手段，也是未来作战的发展趋势和方向，目前各国军队和军工企业都加大了对作战云的研发投入和实际应用。比如美国的作战云，能够联结各武器节点与云端进行数据共享，实现跨域协同作战；俄罗斯研发的无人机作战云系统，可以实现对信息的协同处理。不仅如此，随着近几年移动互联网的蓬勃发展，国内外互联网巨头在各类云计算方面也有着长足的进步。例如百度通过利用 K8s(Kubernetes)实现支持深度学习容器云平台的基础架构；IBM、AWS 等相继推出云平台上的区块链服务，为用户提供快速构建和数据可信存储的服务。在这种背景与需求下，战场分析云平台便应运而生。

云计算平台的关键是如何实现虚拟化，目前的虚拟化技术主要包括两类，一是硬件级别虚拟化，如 KVM、ESXi；二是操作系统级别的容器虚拟化，如 Docker、Rkt 等[4]。硬件级别虚拟化通常以虚拟机的形式对外提供服务，存在资源利用率低、启用速度慢等问题，不能满足应用快速部署与灵活移植的需求。而容器虚拟化具备更加全面的资源控制与资源隔离能力，其中 Docker 容器以轻量易移植、使用简单等特点，从众多容器引擎中脱颖而出。与此同时，容器技术也推动了容器编排工具的发展，目前使用最多的有 Docker Swarm、Marathon 和 K8s[5]。所以，综合课题需求与各类编排工具的特点，本文选择 K8s 搭建战场分析云平台。

虚拟化技术实现对资源的抽象，而调度器则完成对资源的有效分配。但是，仅使用默认调度方法会造成云平台的资源负载不均衡、应用完成运行的时间久等问题，甚至影响整个系统的高效执行。战场分析应用多涉及深度学习方面的处理，而传统的 K8s 调度策略

1

并没有突出 GPU 和显存对应用的重要性，造成 GPU 和显存资源没有得到充分利用；在应用的多样性上，调度策略也缺乏对节点与应用匹配程度的衡量，造成集群内部负载不均衡。除了不合理的资源调度问题外，战场分析云平台还面临着高可用与高可信的问题。

高可用是指当本地系统或应用在某个软硬件出现故障时，还可以继续保证业务系统的功能，且用户感觉不到服务访问上的异常。本文的战场分析应用基于复杂作战环境下，容易受到被损毁的威胁，所以利用高可用技术确保业务系统持续性和非中断性运行是非常重要的。目前不论是学术领域还是工业领域对虚拟机的在线迁移都有成熟的理论和技术支持，但由于两者虚拟化技术之间的差异，使得虚拟机迁移方法不能直接应用在容器迁移中；且目前大多数的容器迁移都是针对单个容器的研究，并没有考虑到容器集群以及 K8s 中容器结构的问题。

高可信是指存储的数据是未被篡改的且数据是可信的。虽然云计算为军事领域的数据采集和处理提供了新的解决思路，但云数据存储仍然存在一定的安全威胁[6]，尤其是处于复杂作战环境下的系统应用，容易发生数据篡改与数据丢失的情况，无法为后续指挥提供可靠的依据。而区块链作为云平台中的一部分，其共识机制可以实现数据的可信存储，但目前的研究并没有将区块链的可扩展组件纳入到 K8s 的管理中，同时也没有考虑分布式集群中可扩展组件（链码）的移动性问题。

综上，在基于战场分析应用的云平台中，实现资源的合理分配、系统的高可用和数据的高可靠还有许多工作要做。因此，为解决上述问题，本文基于 K8s 搭建针对战场分析应用的云平台，研究资源调度策略、容器应用迁移和数据可信存储，有效解决传统军事指挥系统在复杂战场环境下配置部署难、资源分配不均、发生故障服务易中断和数据可信存储的问题。对于战场分析云平台的研究与构建，不仅可以提高作战指挥系统处理战场目标信息的速度，提升作战效率，保障国家安全；还可以广泛应用于情报侦察、指挥控制、安全保密等公共安全领域。

* 1. 国内外研究现状
     1. **K8s 资源调度国内外研究现状**

K8s 是由谷歌内部的容器管理平台发展而来的，理论基础与实际应用基础完备，不论是谷歌内部的容器管理平台还是 K8s，其目的都是为了让部署和管理大量分布式容器应用变得简单易操作[7]。K8s 能够将用户的应用程序打包，然后在一组容器上运行，在这种情况下，编排系统要考虑各种应用程序的不同特性*，尽* 可能充分地利用各种资源为应用进行调度，从而提高集群的负载均衡[8]。

国内如张玉芳等学者针对服务器集群负载不均的问题，在节点本身的性能基础上，提出了以权值衡量负载程度的方法，提高了资源的利用和整体的的负载均衡程度[9]。谭莉等人针对原生调度算法没有考虑节点性能的问题，在综合考虑网络、磁盘的使用情况下对默认调度算法进行了改进[10]。杜军针对一次调度失败的 Pod 进行重调度优化，将调度优先

2

级仅划分为两种（高和低），然后按创建的时间顺序进行调度[11]。这种抢占式调度改进方案虽然能有效提高资源的使用，但是缺乏一定的自由度，不能很好的满足用户对 Pod设定自定义优先级的需求。杨鹏飞针对 K8s 中资源的动态调度进行了研究与改进，提出一种预测模型，能够预测集群中的应用在未来一段时间内的资源用量[12]。虽然可以实现资源的有效调度，但该模型仅仅使用集群中应用的 CPU 和内存使用情况作为调度依据，并没有综合考虑其他资源指标。徐敏等人针对 OpenStack 提出一种依据 CPU、内存和 I/O利用率进行资源调度的 Swift 负载均衡算法[13]。但该算法是基于 OpenStack 平台实现的，并不能很好的移植到 K8s 集群当中。卢胜林、倪明等人在 Swarm 默认的调度器中增加权重调度模块，根据节点权重分配资源[14]。该方法可以提升 Swarm 集群整体负载均衡能力，但并不能保证各种资源的利用率。

国外如 Mohamed K 等人在容器直接对接物理机的单级结构基础之上，增加虚拟机级别的资源分配构成两级结构，在考虑虚拟机资源利用率的同时，也考虑物理机的资源利用率情况[15]。Xu 等人提出一种以网络负载为主的，划分虚拟机优先级并进行资源调度的方法[16-17]。该方法只适用于传统 VM 架构的虚拟机，并不适用于 K8s 架构中的容器。Dong Zhang 等人从网络传输开销、镜像传输开销和节点资源消耗三个方面进行考虑，通过线性方程得到调度的最优解，以达到提升资源使用率的目的[18]。该方法为本课题设置 Pod 应用的权重提供了思路。Liu Bo 等人提出针对多目标的调度方法，充分考虑节点的 CPU 利

内存利用率、镜像传输开销以及容器与节点的关系，从而选择最优节点运行应用[19]。该方法虽然能够有效缩短响应时间，但是对于本文的战场分析应用来说，缺少 GPU 和显存指标的考虑。

* + 1. **Docker 容器迁移国内外研究现状**

国内外不论是学术领域还是工业领域对虚拟机的在线迁移都有成熟的理论和技术支持。比如由 Xen 支持的 Xen Live Migration[20]迁移机制、由 VMware 支持的 Vmotion[21]迁移机制等，都选择了预拷贝内存迭代迁移的方法，且在此期间业务系统并不中断。由于两者虚拟化技术之间的差异，所以虚拟机迁移方法不能直接应用在容器迁移中。

虽然单主机间的容器在线迁移已有了不少可行的方案，但是针对多主机构成的集群内的容器应用迁移仍有许多问题需要解决，国内外学者提供了一些不同的解决思路。对于容器迁移技术，使用最多的是 CRIU，一种对进程进行冻结和恢复操作的工具。大多数容器迁移都是在 CRIU 的基础之上结合不同使用情况进行优化改进。

国内如彭丽苹、吕晓丹等人考虑到数据存储和集群负载的情况，提出了将 Ceph 存储集群与 Docker 容器相结合的方案[22]。虽然在 Swarm 集群中应用程序的容器部署和迁移具有良好的性能优势，但在 K8s 编排工具中却由于容器启动慢而造成迁移代价较大的问题。房锦章、武延军为解决 GUI 容器应用的迁移，设计了 Monitor 容器监控模块，并扩展了CRIU 迁移工具的功能[23]。虽然该模块能够实现单个主机间容器的迁移，但是并没有涉及到针对整个集群内的容器迁移。毛祺，卢胜林为了解决 Docker Swarm 集群中的容器迁移

3

问题，设计了通过概率计算选取待迁容器和迁移目标节点，并利用私有镜像仓库用于不同主机间迁移的情况[24]。严康针对目前容器迁移方案所存在的容器状态大文件保存、传输速度低和恢复时间长等问题，设计出完备的在线迁移系统以降低应用中断时间[25]。其中大文件通过动态压缩减少数据传输量，传输时间通过优化网络协议有效减少，恢复时间通过基于 CRIU 扩展出的容器预拷贝功能有效降低。通过实验测试与比较分析，该系统实现了更快速的应用恢复，容器迁移所耗费的时间平均减少了 26.5%。聂虎卿设计了迁移代价最小的容器选择算法，优化了预测模型，以减少容器迁移次数的形式提高了内存迁移的效率[26]。但该算法主要优化了容器迁移次数和迁移时间，没有考虑集群和节点在资源消耗上的问题，对资源使用情况有特殊要求的应用还需要进一步优化。

国外如 Nadgowda S 等人基于联合文件系统设计了迁移方法

[27]

，虽然能对容器运行时

进行在线迁移，但该方法在目的端丢失了 Docker 架构对容器的原生管理。Ma Lele 等人

基于 CRIU 技术，依据镜像分层特性构建中心镜像仓库，加快容器镜像的迁移

[28]

。但是由

于中心镜像仓库拉取镜像的方式受网络带宽影响较大，存在后期不易扩展的问题。Zhang Liang, Litton J 等人实现了集群中容器存储数据的快速交换，缩短了整体迁移时间[29]。Yu Chengying 和 Fei Huan 利用日志文件和响应机制，在目标节点上使用导入命令传入压缩文件，生成镜像并以容器运行，完成容器的迁移[30]。Barbalace A., Karaoui M.L 等人基于 CRIU和 Docker 的集成，设计 H-Container 迁移技术，实现异构 CPU 的主机之间容器应用的迁移[31]。上述研究[29-31]只针对了单个主机间和边缘计算场景下的容器迁移，并没有考虑集群中甚至是 K8s 集群中以 Pod 为基本单位的迁移问题。Rattihalli G.,Govindaraju M.等人在分析并确定容器资源需求的前提下，设计了基于资源利用的自动伸缩系统[32]。虽然该系统能够动态调整 K8s 集群中运行容器的资源分配，有效提高了集群 CPU 和内存的利用率并减少了迁移时间，但是该方法在迁移过程中没有考虑容器文件内基础数据的重复传输，这也会增加一定的迁移时间。Truyen E., Lagaisse B.等人设计了一种由多个 Pod 共同组成的 PolyPod，通过内存共享提升迁移过程的灵活性[33]。但是在迁移过程中可能会发生内存无限制迭代传输的情况，延长整体迁移时间。

* + 1. **基于 K8s 的区块链国内外研究现状**

课题需要将战场分析应用处理得到的目标关键信息进行存储，为后续提供可靠的数据依据。区块链可追溯、防篡改和分布式的特点正好可以达到这样的目的。由于区块链技术难度大、入门门槛高，逐渐成为近年来的研究热点[34]。本文基于区块链云上运行和管理的问题，对区块链服务云平台进行大量研究，目前许多专家和云计算巨头都有了一定的进展。

工业界如腾讯在 2017 年推出自研发的区块链平台；华为云在 2018 年推出区块链解决方案，为各行各业实现数字化转型提供了一种有效的方法；阿里发布的一种商业区块链，能够协助用户短时间内搭建可用的区块链服务，使得用户可以将重心放在核心业务的实现上；而京东推出的区块链平台，也同样可以实现应用快速上链[35]。IBM 在 2016 年推出了

4

包含区块链服务创建、部署、运行和监控的全套服务平台。微软在 2016 年推出 Baas 服务，支持多种区块链网络，能够实现业务的高效上链。亚马逊在 2018 年推出包含不同类型区块链技术的模板服务，提供 Ethereum、Fabric 或其他模板以便用户迅速搭建区块链网络进

[36-37]

行使用 。

学术界如 Zheng Weilin, Zheng Zibin 等人开发了一种 Baas 平台，实现在云计算环境下提供区块链的服务[38]。Asheralieva Alia 等人研究了基于物联网的 Baas 平台，该平台主要通过云服务器来执行区块链任务，通过对等节点收集设备信息，从而实现设备与网络的互联[39]。Singh J.等人详细阐述了 Baas 的架构，针对平台中基础设施的信任问题进行了讨论，提出了更加优化的 Baas 架构[40]。Ma Zhaofeng, Zhao Weizhe 等人基于联盟区块链设计了一

种具有高度灵活性和可扩展性的通用 Baas 平台，实现了基于 Fabric 的数据可信存储

[41]

。

但是没有考虑到分布式集群中链码可能跨节点调度或跨节点运行失败的问题。

综上所述，国内外学者对云平台中资源分配、容器迁移、数据可信存储的研究均取得了一定的成果，为基于战场分析应用的 K8s 云平台奠定了基础，未来战场分析云平台也将朝着更加智能化的方向发展。但目前存在的资源分配方案仍较为单一，容器应用迁移也没有考虑 K8s 集群的生态情况，这些问题都亟待解决。因此，本文通过对这三个方面的重点研究，以达到战场分析应用 K8s 云平台上资源有效利用、系统高可用和数据高可信的目的。

* 1. 论文主要研究内容

本文的重点是构建一个基于战场分析应用的 K8s 云平台，通过多台主机构建底层 K8s云架构，以容器的方式运行战场分析应用；并对集群中的主机与应用进行统一监控管理、优化资源调度、容器应用迁移和关键数据可信存储；除此之外还提供 Web 可视化操作界面，降低集群管理难度。本文研究的关键技术和研究内容如图 1.1 所示。

研究内容如下：

**1）K8s 集群环境搭建。**研究基于容器虚拟化的容器云平台的特点，确定本文的集群搭建拓扑，采用二进制文件方式完成 K8s 集群搭建，为后续研究提供环境基础。

**2）一种针对战场分析应用的 K8s 资源调度优化改进与实现。**针对战场分析应用的资源需求多样性、战场分析服务复杂性、默认调度策略性能较差等问题，本文研究并实现了一种优化调度策略，该策略综合节点各类资源性能，在突出 GPU、显存对战场分析应用的重要程度的同时，还充分考虑应用与节点的匹配程度。

**3）一种面向 K8s 的容器应用迁移方法改进与实现。**针对 K8s 集群中在迁移容器应用时产生冗余文件的问题，本文研究了面向集群的容器迁移方法，对容器镜像进行分层迁移，同时改进并简化内存迁移，优化针对战场分析应用 K8s 云平台的容器应用迁移方法。

**4）一种基于 K8s 的 Fabric 区块链可信存储设计与实现。**针对 K8s 分布式集群中 Fabric

动态链码可能存在跨节点编译与运行的问题，本文研究了 Fabric 基础组件与可扩展组件

5

的区别，改造可扩展组件（链码）使其按需启动运行，实现了对 K8s 集群处理得到的关键数据的可信存储。

**5）可视化 Web 管理的功能设计与实现。**使用 Golang 语言进行平台功能模块开发，并完成平台的功能测试与性能分析。

**基于战场分析的K8s云平台**

**研究内容五**

可视化Web界面

（平台总览、应用部署、镜像管理、可信数据管理）

**研究内容**

**研究内容一**K8s集群环

境搭建

**研究内容二**

基于战场分析的K8s资源调度

优化设计与实现

**研究内容三**

基于K8s的容器应用迁移

优化设计与实现

**研究内容四**

基于K8s的Fabric区块链

高可信设计与实现

**研究的关**

**键技术**

图 1.1 本文研究的关键技术和研究内容

* 1. 论文组织结构

本文的内容结构及章节安排如下：

第 1 章，绪论。本章节首先详细介绍了在基于军队信息化不断发展的背景下对云计算技术进行分析研究，阐明了针对战场分析应用的云平台的研究意义。然后分别介绍了目前K8s 集群中资源调度、Docker 容器迁移和基于云平台的区块链服务的研究现状。最后整合出本文的研究内容。

第 2 章，平台总体设计。本章节介绍了 K8s 云平台的总体设计和相关研究内容的关键技术。首先在充分了解上层业务和课题要求的基础上，对战场分析云平台进行需求分析，包括统一资源监控、系统高可用、数据高可信和管理可视化等需求。然后根据具体需求对平台进行详细的方案设计，包括云平台架构设计和云平台功能模块设计，并详细介绍云平台架构的各个分层，阐述各个功能模块之间的关系。最后对本文的研究内容和关键技术进行了介绍，包括 K8s 架构及资源对象、Docker 容器迁移技术和 Fabric 区块链技术。

第 3 章，基于战场分析的 K8s 资源调度优化设计与实现。本章节主要完成 K8s 云平

6

台对战场分析应用的调度策略进行优化改进的设计与实现过程。首先介绍 K8s 整体调度框架和默认调度器的基本调度策略。然后阐述了默认调度器在战场分析应用对 GPU 资源、显存资源调度需求上存在的问题，综合各类资源进行改进，重点突出 GPU 资源、显存资源、应用与节点的匹配度在调度过程中的重要性。最后确定优化调度器的整体调度流程，并阐述实现自定义调度策略的三种方法，选择扩展调度程序的方式实现调度的优化策略。

第 4 章，基于 K8s 的容器应用迁移优化设计与实现。本章节主要介绍了 K8s 云平台中 Docker 容器应用迁移的优化设计与实现过程。首先对现有集群中的容器迁移方法存在的问题进行分析。然后针对问题研究 K8s 集群中 Docker 迁移的方式，包括迁移准备、正式迁移和应用恢复，优化了镜像和文件系统的对比迁移以及内存迁移。最后确定迁移的整体流程，并实现优化改进的针对战场分析 K8s 集群的容器应用迁移。

第 5 章，基于 K8s 的 Fabric 区块链高可信设计与实现。本章节主要介绍了在 K8s 集群中运行区块链应用的设计与实现过程。首先对 Fabric 网络启动流程进行介绍，并指出在分布式 K8s 集群上进行区块链应用时所面临的跨节点编译与启动的问题。然后分析基础组件与可扩展组件的特点，确定基础组件直接随网络启动，可扩展组件跨节点按需启动的部署方式。最后给出区块链应用云上部署的整体流程，通过定义数据结构编码完成Fabric 在 K8s 上的运行，从而实现数据在区块链上的可信存储。

第 6 章，平台实现与测试分析。本章节是对上述研究内容的实现以及对结果的分析。首先介绍了平台的实现环境，包括平台拓扑结构、节点配置和引擎所用版本。然后按功能模块对平台功能进行测试。最后通过实验验证了本文提出的调度优化策略使集群的负载程度和节点的负载均衡都得以优化；同时本文提出的面向 K8s 的容器应用迁移方法能够减少文件传输量和迁移时间。

第 7 章，结论与展望。本章回顾了课题的研究内容。首先总结了本文主要完成的内容。然后分析并展望资源调度优化、容器应用迁移和 Fabric 云化的研究趋势和进一步优化的方向。

7

1. 平台总体设计

本章对本文需要研究并实现的云平台进行需求分析与方案设计，并介绍相关理论基础。首先对基于战场分析应用的 K8s 云平台进行需求分析。其次对应各个需求分析进行平台整体架构和平台功能模块的设计。最后对 K8s 云平台的相关理论基础与现有技术进行对比分析，引出本文在技术上的优化改进。

* 1. 战场分析云平台需求分析

军事云平台是基于底层云计算技术和网络来提供服务的专用平台，旨在建设并开发军事基础设施硬件体系和军事应用服务软件体系。随着信息化武器在现代化作战中所占的比例越来越高，实现对战场信息的集成和有效判别是至关重要的。当前云计算已逐渐应用于各个军事领域，例如利用强大的计算能力为舰队联合作战系统[1]和作战辅助决策系统[42]

提供可靠的数据参考依据。

但是由于军队现代化战略的升级换代，各级部队各类信息系统数量呈现出的爆炸式增长对云平台的可用性和可靠性提出了更高的要求，例如更加灵活的资源分配方案，以及大量信息的安全问题。在此背景下，针对战场分析应用的云平台应运而生，既解决了当下作战体系所面临的问题，也是未来作战的发展趋势和方向。国外方面，美国、欧洲以及俄罗斯都拥有功能和技术相对成熟的作战仿真平台。例如美陆军的“设施即服务/混合模式”的军事网，在阿富汗战场已经得到了检验；欧洲的层云“STRATUS”系统，在全平台为系统的指挥控制提供一种创新方法，能够管理作战资源并提供决策支持；俄罗斯研发的无人机作战云系统，可以实现信息协同处理；除此之外，IBM、AWS 等也相继推出云平台上的区块链服务，保证数据可信存储。国内方面，在夺取未来信息优势建立信息化作战体系方面取得了较大发展，但与发达国家仍存在一定差距，对体系的建立还处于论证阶段，尚且没有“作战云”平台的建设与实际落地应用；且从应用软件的角度来看，大部分都是大型企业解决特定问题而设计研发的，在军事战场分析场景下的使用具有较大的局限性。

因此，在此背景下，本文所研究的基于战场分析应用的云平台将运用于模拟战场背景下的应用部署与管理，实现以 Docker、K8s 集群为底层架构的，以资源优化调度、容器迁移、数据可信存储为核心的，包含平台总览、应用部署管理、镜像管理、可信数据管理等主要功能的云平台。其中资源优化调度与容器迁移运行在 K8s 集群的调度器中，数据可信存储对应可信数据管理模块。具体需求划分为以下几点：

**1）统一资源监控**

统一资源监控需要对集群内部的运行情况进行信息监控与收集，是其他功能与性能要求的基础，例如资源调度负载均衡和应用高可用性都是以平台中的主机和应用的状态为前提来实现。平台需要对物理主机的 CPU、内存、GPU、磁盘存储、网络资源使用情况进

8

行监控。

**2）战场环境下平台应用的高可用**

平台应用的高可用需求包括两个方面，一是资源的有效分配，本文构建的云平台内大多运行处理战场信息的应用，例如目标识别、目标跟踪等基于深度学习的业务，需要针对性的改进资源调度策略保证应用的完成，提升集群的负载均衡；二是主机故障时应用的有效迁移，军事系统通常工作于各种复杂的环境之下，战时还会面临敌方硬摧毁的威胁，造成业务中断，严重影响使用效果，需要通过优化容器应用迁移技术对发生故障或者需要硬件升级维护的容器业务进行迁移。

**3）平台关键数据高可信**

在作战中，单个终端容易遭受数据篡改和破坏，由终端应用程序处理得到的数据在为后续提供依据时无法保证可靠性，所以需要在云平台上利用区块链技术，对经过处理的数据进行存储，保证数据不可篡改与可信性。

**4）可视化 Web 管理**

本文选用 K8s 作为云平台的底层架构，主要通过 Linux 命令进行操作，对于使用者来说需要记住大量的操作命令，导致学习和操作难度大。而可视化 Web 管理是用户与平台进行交互的统一入口，通过管理中心屏蔽底层的实现细节，实现对平台中的物理主机和资源、容器应用、镜像以及存储数据的管理，不需要记繁琐的命令，降低对平台的使用门槛。

* 1. 云平台方案设计
     1. **云平台总体架构设计**

综合前文对基于战场分析应用的 K8s 云平台的需求分析和当前可视化 Web 管理的现状，设计平台的总体架构如图 2.1 所示。平台从下到上一共分为五个部分，包括资源层、支撑层、服务层、私有镜像仓库和 Web 可视化管理。

**1）资源层：**为上层提供应用运行的各类资源，一般由平台中多个物理主机的基础设施提供，包括计算资源（CPU、GPU）、内存资源、磁盘资源以及网络资源。同时需要在底层实现 Fabric 区块链网络，为服务层的可信数据存储提供支持。资源层通过 Docker 容器虚拟化技术实现对基础资源的统一管理与分配。

**2）支撑层：**由主节点和工作节点构成具有支持云平台容器编排管理工具的 K8s 集群，并将扩展优化的调度策略和容器迁移方法运行在 Scheduler 调度器当中，优化资源的调度分配，并实现容器应用的迁移。

**3）服务层：**通过开发实现四大功能模块，平台总览功能监控集群中主机资源的使用情况，包括 CPU、内存、磁盘和网络等资源，实现对集群信息以及节点信息的统计与展示，方便用户操作；应用部署实现应用的创建、编辑、停止和迁移功能，将具体战场分析业务容器化；镜像管理实现镜像的创建、下载和删除功能，将应用程序镜像化；可信数据存储主要实现关键数据在区块链上的存储与查询。

9

**4）私有镜像仓库：**将支撑层所有的镜像存储在统一镜像仓库进行管理，方便对镜像的存储与操作。

**5）Web 可视化管理：**Web 可视化管理是对集群操作的统一入口，用户通过该可视化界面屏蔽底层细节，实现与平台的交互，降低集群的使用门槛。

**Web可视化管理**

**云**

**平**

**台**

图 2.1 基于战场分析应用的 K8s 云平台总体架构

* + 1. **云平台功能模块设计**

由于对容器云平台的使用多通过命令行的方式进行操作，自带的 Dashboard 面板只能进行简单的查看，并不能对集群进行操作，所以课题通过二次开发完成对云平台进行管理的功能模块，实现对容器应用和镜像的管理。根据平台需求分析，基于战场分析应用的K8s 云平台的功能模块如图 2.2 所示。共包括平台总览、应用部署管理、镜像管理和可信数据管理四大模块。本文的资源优化调度策略运行在 K8s 底层架构的调度器当中，在集群中发挥节点分配的作用，结果显示在应用部署模块中。下面是对 Web 可视化管理模块各功能的具体设计。

**1）平台总览模块**

平台总览模块主要是对通过统一资源监控收集到的平台信息进行展示。包含集群信息和节点信息两个部分，展示的内容包括集群和主机的 CPU、内存等资源的使用情况。

10

**2）应用部署模块**

应用部署模块实现对平台中战场分析应用的管理。应用部署模块主要包含四个部分：创建容器应用、容器应用编辑、容器应用停止以及容器应用迁移，还可以在底层控制台查看运行应用容器的日志、当前所在节点以及是否更换节点等信息，这是调度与迁移的结果展现。

**3）镜像管理模块**

镜像是构成 Docker 容器的基础，而仓库是管理大量镜像的基础。镜像管理需要构建私有镜像仓库对平台中的公有基础镜像、自定义镜像进行统一管理，包括镜像创建、镜像下载、镜像删除等。本平台中镜像管理部分用户可以通过上传 Dockerfile 配置文件实现镜像的创建，并能对镜像实施下载和删除操作。

**4）可信数据管理模块**

可信数据管理模块通过底层的 Fabric 区块链实现，以区块链不可篡改的特性保证关键数据的真实可信。主要包含三个部分：关键信息存储、完整信息存储、信息查询。信息存储主要保存目标的关键字段，完整信息存储放在普通数据库当中，已存储的信息可通过主机的 IP 地址进行查询。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | | | | | | | | | 基于战场分析应用的K8s云平台 | | | | | | | | | | | | | | | | | | | | |  | | | | | | | | | | | |
|  | | | | | | | | | | |  | | | | | | | | | |
|  | | | | | | | | | | | | |  | | | | | | | | | | | | | | 镜像管理模块 | | | | |  | | | | |  | | | | |
| 平台总览模块 | | | | | | | | 应用部署模块 | | | | | | | | | | | | | | | | | | | | | | | 可信数据管理模块 | | | | | | | | | |
|  |  |  | |  | | | | | | | | | | | | |  | | | | | | | | | | | | | |  |  |  | | |  | | | | |
|  | | | |  | | |  | |  |  | | |  | | | |  | | |  | | |  | | |  | | |  | |  |  | |  | | |  | |  |  | |
| 集  群  信  息  展  示 | |  | | 节  点  信  息  展  示 | |  | 创  建  容  器  应  用 | | 容  器  应  用  查  看 | | |  | 容  器  应  用  编  辑 | |  | | 容  器  应  用  删  除 | |  | 调  度  结  果  查  看 | |  | 迁  移  结  果  查  看 | |  | 镜  像  创  建 | |  | 镜  像  下  载 | | 镜  像  删  除 | | |  | 关  键  信  息  存  储 | |  | 完  整  信  息  存  储 | | 信  息  查  询 | | |  |

图 2.2 云平台功能模块图

* 1. 相关技术基础
     1. **K8s 架构**

针对课题的需求本文采用 Docker 容器引擎，由于本文需要部署多个容器应用并对其进行资源监控，但仅以命令行的方式进行管理难度较大，所以需要容器编排工具实现对基础设施资源的整合和对容器的有效管理。如表 2.1 所示是三种开源容器编排工具的优缺点比较。

11

表 2.1 开源容器编排工具比较

容器编排工具 优点 缺点

Docker Swarm 功能简单，适合小集群管理[43] 结合 Docker

Compose 使用

没有集群统一监控和用户界面，只能基

于命令行操作

Marathon 支持并提供集群资源管理 没有提供容器自动伸缩，统一监控功能

K8s

提供资源调度、容器运行以及服务发现等功能

目前只支持CPU 单一指标自动伸缩 [44]。并且 Dashboard 只能提供简单的查看和

基础功能

通过比较分析，本文基于 K8s 架构实现对容器集群的基本管理，并结合课题实际所需，在此基础上实现云平台管理功能的二次开发和调度器的改造。下面对集群架构、功能组件和资源对象进行简单介绍。

**1）K8s 集群架构**

K8s 集群架构如图 2.3 所示，包含 Master、Worker 两种节点和七大组件，节点和集群的具体功能都是由以下七大组件实现的[45]：

**Node1/Worker1工作节点**

**Pod应用** **Pod应用**

container

Kubelet

命令行组件

container

Kube-proxy

请求转发组件

**External Network**

**外部网络**

**Client**

**Master主节点**

**客户端**

**Node2/Worker1工作节点**

**Pod应用** **Pod应用**

|  |  |  |
| --- | --- | --- |
|  | | API Server  集群管理接口 |
| Controller Manager  管理控制中心 |  |
|  |

container

container

|  |  |  |
| --- | --- | --- |
| Scheduler  调度器 |  | Etcd  键值对  数据库 |
|  |

图 2.3 K8s 集群架构

⚫ API Server：提供统一的访问接口，任何资源请求都要先交给它，由其进行统

一管理与分配；

⚫ Controller Manager：API Server 可以理解为“前端”工作，那么 Controller Manager

就可以理解为“后端”工作，负责维护应用具体运行时的状态；

⚫ Scheduler：为集群中应用的正常运行按调度策略寻找节点和资源；

⚫ Kubelet：命令行管理工具，是 Master 在 Node 节点上的代理，负责当前 Node

节点上容器的创建、启动、停止等任务，并封装容器引擎之间交互的 API；

12

⚫ Kube-proxy：请求转发控制组件，负责 Service 间的通信和负载均衡，完成将

客户端的访问请求转发到后端集群的多个 Pod 上运行的工作；

⚫ Etcd：用于存放集群内部信息（节点信息、资源信息）的键值对数据库；

⚫ Container Runtime：负责对容器的创建、运行、停止、暂停和删除五种状态的生命周期管理，当系统中的容器运行起来时，此时的 Container Runtime 就是一个具体的 Docker 应用。

Master 是管理集群和资源访问的主要节点[46]，由 API Server、Controller Manager、Etcd和 Scheduler 等四个管理组件构成。Worker 节点是集群中的实际工作节点，也可以称作Node 节点，由 Kubelet、Container Runtime 和 Kube-proxy 构成，负责应用容器的创建与删除，同时提供容器运行的具体环境。

**2）K8s 资源对象**

K8s 通过十多种资源使用对象实现对整个集群的控制，但与本课题相关的资源对象主要有以下三种：

⚫ Pod：K8s 资源调度的基本单位[47]，实际的应用服务就是运行在 Pod 中的。一个 Pod 中通常会运行一个或多个相关的 Docker 容器，通过进程间通信和文件共享方式来组合提供应用服务。

⚫ Replication Controller（RC）：RC 控制器负责确保 K8s 集群中多个节点上的应

用副本数正常。

⚫ Service（SVC）：SVC 由集群中多个 Pod 构成，并以服务的形式抽象出来，负责定义服务访问和请求转发策略，记录了 Pod 变化的 IP 地址，达到访问不同Pod 实例的目的。

虽然 K8s 的默认调度器在大多数场景下已经能够满足需求，但是针对战场分析场景下的应用来说资源分配的针对性、节点的负载均衡都较低，所以需要在默认调度策略的基础上扩展自定义调度策略，实现针对性的资源调度。

* + 1. **Docker 迁移技术**

针对现有容器迁移所面临的问题，很多学者对相关容器应用迁移的实现，其中大部分

方法都是在 CRIU 的基础上进行优化与改进。下面是 CRIU 技术的基础理论。

CRIU（Checkpoint/Restore In Userspace）[48]是作用于用户空间上的一种工具，如图

2.4 所示，CRIU 主要分为两个过程，一是冻结运行过程保存数据；二是恢复数据并继续运行进程。借助这个过程，可以通过扩展 CRIU 实现程序或容器的冻结，迁移以及远端调试等功能。

13

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 待迁容器 | |  | | | 容器恢复 | |  |
|  |  |  | |
| 插入寄生代码 | | | | |  | | |
| **CRIU恢复过程**  恢复上下文，进程恢复运行  恢复每个进程详细的资源使用状态  恢复容器命名空间  将容器进程树放入Docker Daemon  进行恢复  根据检查点文件解析资源 | | | |
| **CRIU冻结过程**  冻结容器内所有进程创建容器进程树结构  收集每个进程详细资源使用情况  存储为检查点文件 | | | | 传输检查点文件 |
|  | |  | |
|  | |  | |
|  | |  | |

图 2.4 CRIU 示意图

冻结进程运行过程中通过插入寄生代码开始保存运行状态和数据，具体过程如下：①寄生代码首先动态注入到待迁移容器，冻结容器并暂停进程执行；然后生成镜像文

件并收集进程的上下文信息。②收集进程的资源使用情况并存储为检查点文件。③删除寄生代码。

传输检查点文件，恢复数据并继续运行进程的具体过程如下：

①分析检查点文件获取资源使用情况。②根据检查点文件恢复容器。③恢复容器各个进程的基本资源使用情况，包括内存区域、使用的文件状态以及命名空间 Namespace 等。④恢复进程运行的上下文。

当前对 Docker 容器应用迁移的研究中有不少卓有成效的迁移机制，比如 Nadgowda S等人基于容器联合文件系统对容器运行时进行迁移的 Voyager[27]机制，以及 Ma Lele 等人[28]基于镜像复用和恢复 Daemon 管理的容器迁移技术。虽然这些研究都达到了不同物理机之间容器应用迁移的目的，但是 Voyager 方法仅从容器运行时角度入手，忽略了目标端Docker 架构对应用的重新管理能力；Ma Lele 等人的方法不适合由多物理机所组成集群下的容器迁移。所以本文从上述 Docker 容器迁移方案的不足之处吸取经验教训，针对 K8s架构集群环境中的容器应用，实现一种分阶段的迁移方案。在文件系统迁移方面，重复利用未变化的基础只读镜像以减少数据传输；在内存迁移方面，优化停机迁移条件以减少效率不高的重复迁移。

* + 1. **Fabric 区块链基础**

区块链通过共识可以保证数据的可信性，一般分为公有链、私有链和联盟链。公有链通过任意节点参与共识并向公众开放数据记录；私有链通过特定节点参与共识私密度更高；而联盟链介于两者之间，同时由于共识更快、可控节点更多的特点，在各方面都应用广泛[41]。本文重点研究联盟链中的 Hyperledger Fabric 技术[49]，从逻辑架构和运行时架构进行技术理论介绍，并重点分析智能合约与链码在一次数据存储中的作用。

**1）Fabric 逻辑架构**

图 2.5 是 Fabric 的逻辑架构，主要由成员服务、区块链服务以及链码服务[50]三个服务模块组成。其中最重要的是 Fabric 提供 API、SDK、CLI 等接口，方便用户对 Fabric 进行自定义的功能管理，课题通过编写智能合约对自定义数据进行可信存储就是通过 SDK 接口改造实现的。下面是对成员服务、区块链服务和链码服务的详细说明。

**API、SDK、CLI**

**成员管理** **区块链管理** **交易管理** **合约管理**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **成员服务**  会员注册  身份管理  认证服务 | **区块链服务**  共识管理 分布式账本  P2P网络协议 账本存储 | **链码服务**  安全容器服务安全注册服务 |
|  | **事件流** |  |

**服务**

图 2.5 Fabric 逻辑架构图

⚫ 成员服务：主要负责身份认证和权限管理，来保证链上访问的安全性；

⚫ 区块链服务：负责区块链管理和交易管理的核心部分。包括节点间的共识、分布式数据存储、以及节点间对等协议的实现；

①区块链管理：实现对网络内部交易区块的上链管理。首先从排序服务节点上接收交易区块，并根据智能合约与冲突策略判断该区块上的交易是否有效，然后将该区块添加到节点文件系统中的哈希链上，完成区块上链操作。

②交易管理：负责对链码相关交易的管理。一是进行创建、启动链码容器工作的链码部署交易管理，链码被安装在区块链上是交易成功的标志；二是进行账本数据修改的链码调用交易管理，通常将对数据的修改操作函数写入调用交易中，调用结束后返回操作结果。

⚫ 链码服务：链码是能够对链上数据进行操作的、用户可自定义的、运行在容器

中的一个组件程序。链码服务为智能合约程序的运行提供环境。

**2）Fabric 运行时架构**

Fabric 的运行时架构和重要角色如图 2.6 所示，可以看到在区块链网络运行情况下，有多个节点承担不同的功能角色以构成一个分布式的系统，通过共识保证交易数据安全。

15

**客户端**

**App**

登记注册

**CA**

图 2.6 Fabric 运行时架构图

其中具体的节点和功能角色在表 2.2 中进行说明。

表 2.2 Fabric 运行时架构角色说明

角色 说明

APP 客户端：创建交易且通过验证后向 Orderer 提交真正的交易请求

CA 认证：对 Fabric 网络中的所有节点和成员身份进行统一认证管理

Channel 通道：保证指定节点和 Orderer 进行保密通信与数据隔离的子网

Orderer 排序服务节点：负责验证交易、对交易排序、生成区块并在通道内广播

Org 组织：代表联盟中多个不同的组织

Cahincode 链码：以容器运行的方式通过 API 实现对账本数据的交互与操作

Ledger 账本：存储所有交易记录并保存在所有节点上的一个交易哈希链

Peer 对等节点：Fabric 网络中的节点都是对等节点，共同维护着哈希链上的数据

角色中 Peer 节点可以划分为 4 种：预先执行交易给出响应的背书节点；同步区块至其他节点的 Leader 节点；与同一通道内不同组织进行信息交互的锚节点；以及验证交易结果并保存到账本的提交节点。其中改造自定义的智能合约从背书节点上完成。由于课题需要将自定义数据存储在区块链的账本中，而对账本进行一次操作就是发起一次交易，

次交易就是一次数据存储，节点角色与完整交易的过程如图 2.7 所示，一共包含 8 个步骤。

**区块链网络**

**Org**

背书节点

2.模拟执行提案并进行背书

8.同步区块

8.同步区块

提交节点

9.保存区块到账本

**客户端**

**App**

Leader

7.保存区块到账本

锚节点

连接其他节点

6.广播区块给主节点

Orderer

5.对交易进行排序生成区块

图 2.7 交易流程图

①客户端向区块链网络提交交易请求。

②区块链网络在收到交易提案后通过背书节点进行身份验证、预先执行提案、生成读写集等操作，再附签名后将结果响应给客户端。

③返回模拟执行结果给客户端后，在签名验证后根据链码的具体操作，区分是否需要提交该响应结果给网络中的排序节点进行账本更新。

④生成交易。若链码执行的是调用操作，则要将交易请求和预先执行的结果提交给排序节点，确认所有预先执行的结果一致后再将前三步的信息打包生成一次交易。

⑤排序节点按照时间顺序产生区块。

⑥排序服务节点将上一步产生的区块广播同步给 Leader 节点。⑦保存区块至账本。记账节点将有效交易提交至账本。

⑧主节点在组织内部同步最新的区块给其他节点。

由上述 Fabric 的一次账本操作过程可以清晰了解数据不可篡改的原理。通过 Fabric实现数据可信存储已有大量研究，而对于 K8s 集群上的区块链服务来说，现有方案虽然通过一定的辅助技术部署链码，但并没有在 K8s 的管理环境中进行集成。所以本课题针对该问题，使用容器控制组件的方法完成将链码纳入 K8s 环境管理的目标。

* 1. 本章小结

本章首先从云计算在军事领域的应用展开，阐述国内外对现有军事应用云平台的需求。接着对课题要求构建的基于战场分析应用的 K8s 云平台进行需求分析。然后给出本文构建的基于战场分析应用的 K8s 云平台整体架构和云平台功能模块设计。最后对本文研究所用到的相关理论与技术进行介绍。

17

1. 基于战场分析的 K8s 资源调度优化设计与实现

本章主要介绍战场分析应用在云平台中资源调度策略的研究与优化过程。首先介绍K8s 调度基本功能与流程，并研究默认调度策略。其次针对战场分析应用的资源需求与特点，分析 K8s 默认调度策略在使用过程中的不足，针对 Pod 的优先级、GPU 资源、显存资源对调度策略进行改进设计。最后通过比较三种优化调度策略的实现原理确定本文的实现方法，并通过编码实现扩展的优化策略。

* 1. K8s 调度策略研究
     1. **K8s 默认调度概述**

K8s 作为一个容器集群管理工具，为了成功运行各类应用，就需要为应用容器合理分配平台资源。调度器在其中发挥着重要作用，它为具体的 Pod 应用寻找集群中可用 Node节点并进行绑定。具体调度框架如图 3.1 所示，整体调度过程如下：

① 客户端发起创建资源请求；

② API Server 接收用户请求后，先存储与请求相关的数据到 Etcd 数据库中；

③ Scheduler 调度器监听到 API Server 的用户请求后，查看待调度 Pod 列表和可用Node 节点列表，通过预选和优选过程为每个 Pod 分配 Node 节点。

④ 将优选阶段分数最高的 Node 节点与 Pod 进行绑定操作，并将结果存储到 Etcd 中；⑤ 被选中的 Node 节点的 Kubelet 负责执行 Pod 的创建、镜像下载以及容器启动操作，

完成绑定。

**客户端**

发起创建资源请求

存储相关数据

到Etcd

Pod1

**API Server** **Etcd**

**Node1**

监听到请求后准备分配资源

**Scheduler**

Pod2

分配完成后进行

绑定操作

**Node2**

**待调度Pod列表** Pod1 Pod2 Pod3 ...

**调度策略**

**调度算法** Pod3

**可选节点列表**

**Master**

Node1

Node2

Node3

...

**Node3**

图 3.1 K8s 调度框架

* + 1. **K8s 默认调度算法研究**

图 3.1 中的 Scheduler 就是 K8s 的默认调度器[52]，在进行具体的资源调度时，又细分为预选、优选两个阶段，在调度过程中这两个阶段所发挥的作用如图 3.2 所示：

首先在预选阶段过滤不可用节点。调度器通过预选策略循环遍历所有节点的资源情况，筛除不符合 Pod 应用资源请求的节点，并在没有符合资源请求节点的情况下，Pod 将排在队尾等待直到有节点满足资源条件。默认的预选策略主要有：PodFitsHostPorts、PodFitsHost、PodFitsResources 和 MatchNodeSelector。

然后在优选阶段为节点打分。调度器根据优选策略从预选过滤得到的节点中选择最优的一个。默认的优选策略主要有以下几个： LeastRequestedPriority、MostRequestedPriority以及 BalancedResourceAllocation。

Node节点列表

Node1 Node2 Node3

预选

Node节点列表

Node1 Node2 Node3

节点没有足够的资源

优选

Node节点列表

Node1 Node2

分数：5 分数：10

绑定

Node节点列表

Node2

最终选出Node2与Pod绑定

图 3.2 预选、优选示意图

主要的预选策略如下[51]：

（1）PodFitsHostPorts：确定 Node 节点上关于待调度 Pod 所请求的端口是否空闲。（2）PodFitsHost：确定待调度 Pod 是否指定了特定的 Node 节点。

（3）PodFitsResources：确定 Node 节点是否还存在剩余资源供待调度 Pod 使用。（4）MatchNodeSelector：确定 Node 节点的标签是否与 Pod 所要求的一致。

主要的优选策略如下[51]：

（1）LeastRequestedPriority：根据资源消耗程度打分，资源消耗越少，该 Node 节点分数越高，越能更多的部署应用。具体计算如公式(3.1)所示：

(

*score* =

*node* -*toMi*

*cpu* *cpu*

*node*

*cpu*

) 10 (

+

*nodemem* -*toMimem*

*node*

*mem*

) 10

(3.1)

其中 *nodecpu* 表示可用节点的 CPU 总量，*nodemem* 表示可用节点内存资源总量，*toMicpu* 表示可用节点上已调度 Pod 和待调度 Pod 的 CPU 资源请求量，*toMimem* 表示可用节点上已调度Pod 和待调度 Pod 的内存资源请求量。

（2）MostRequestedPriority：选择能最多满足资源请求的 Node 节点，这样可以将待调度 Pod 在完成任务的前提下，使用集群中更少的节点。

（3）BalancedResourceAllocation：根据资源使用均衡情况打分，资源使用越均衡，分数越高。具体计算如公式(3.2)所示。

*score* =10 - *math*.*Abs*









*toMi*

*cpu*

*nodecpu*

+

*toMi*

*mem*

*nodemem*









10

(3.2)

* 1. 基于战场分析的 K8s 资源调度优化设计
     1. **问题分析**

作为目前主流的容器管理工具，K8s 能够对集群中的容器应用进行统一管理和监控，同时调度器为 Pod 选择合适的 Node 节点进行绑定完成应用的运行。但是针对课题中的战场分析应用，默认调度器性能欠佳，在使用过程中存在许多限制：

（1）针对战场分析系统所面临的资源需求多样性（CPU、内存、磁盘、网络、GPU等）和战场分析服务的复杂性（目标检测、人体姿态估计、目标威胁评估以及目标跟踪），默认调度策略不具有针对性。

（2）没有根据 Pod 应用的实际资源请求情况为其设置合适的调度优先级。

（3）选择最优节点的过程中并没有突出 GPU 和显存指标对战场分析应用的重要程

度；没有考虑到应用的资源需求与 Node 节点上可用资源的匹配度

[10-11]

。

所以针对云平台中具体的战场分析应用，对默认调度器进行优化设计。在增加磁盘、网络指标的基础上优化预选阶段，增加了 GPU、显存指标；同时在预选阶段之前增加预处理，根据 Pod 优先级与节点负载情况排成有序队列；优化优选阶段重点突出 GPU、显存以及应用与节点匹配程度的权重，针对性的选出对战场分析应用最优的节点。

* + 1. **优化预选阶段**

优化预选阶段首先是在增加磁盘、网络指标的基础上，将 GPU、显存指标纳入调度的统一衡量中，同时结合节点的负载情况保证节点与集群的负载均衡；其次根据 Pod 的实际资源请求情况为 Pod 设置合适的调度优先级。

集群中的节点表示为集合 *N* = (*N*1 , *N*2 , *N*3 ,..., *Ni* ,..., *Nn* ) ，集群中待调度 Pod 表示为集合 *P* = (*P*1 , *P*2 , *P*3 ,..., *Pj* ,..., *Pm* ) ，根据公式(3.3)、公式(3.4)和公式(3.5)衡量节点的负载情况：

*LS* (*Ni* ) =

*Load* (*Ni* ) *Source*(*Ni* )

* + - 1. =1,2,3,..., *n*

(3.3)

*Load* (*Ni* ) = *L*(*N*

*i*

*cpu*

) + *L*(*N*

*i*

*mem*

) + *L*(*N*

*i*

*disk*

) + *L*(*N*

*i*

*network*

) + *L*(*Nigpu* ) + *L*(*N*

*i*

*vram*

)

(3.4)

*Source*( *Ni* ) = *a*  *S* ( *N*

*i*

*cpu*

) + *S* ( *N*

*i*

*mem*

) + *S* ( *N*

*i*

*disk*

) + *S* ( *N*

*i*

*network*

) + *b*  *S* ( *Nigpu* ) + *S* ( *N*

*i*

*vram*

)

(3.5)

公式(3.4)中 *L*( *Ni* ) 代表节点 *Ni* 上的 CPU 使用率，*L*( *Ni* ) 代表节点 *Ni* 上的内存使用

*cpu* *mem*

率，

*L*(*Ni* ) 代表节点 *Ni* 上的磁盘使用率，

*disk*

*L*(*N*

*i*

*network*

) 代表节点 *Ni* 上的网络带宽使用率，

*L*( *Ni* ) 代表节点 *Ni* 上的 GPU 使用率， *L*( *N*

*gpu*

*i*

*vram*

) 代表节点 *Ni* 上的显存使用率。

公式(3.5)中 *S* ( *Ni* ) 代表节点 *Ni* 上 CPU 的频率，*a* 是 CPU 的核数，*S* ( *Ni* ) 代表节点

*cpu* *mem*

*Ni* 上内存容量，

*S* (*Ni* ) 代表节点 *Ni* 上磁盘容量，

*disk*

*S* ( *N*

*i*

*network*

) 代表节点 *Ni* 上网络带宽，

*S* ( *Ni* ) 代表节点 *Ni* 上 GPU 的频率，*b* 是 GPU 流处理器的数量， *S* ( *N*

*gpu*

*i*

*vram*

) 代表节点 *Ni* 上

显存容量。

根据公式(3.6)设置 Pod 的优先级值：

*V* (*Pj* ) = *init*(*P* ) + *num*(*P* ) +*images*(*P* )

* + - 1. =1,2,3,..., *m*

(3.6)

其中*init*(*Pj* ) 是 Pod 的初始优先级，*num*(*Pj* ) 是 Pod 要运行的副本数量，*images*(*Pj* ) 是 Pod运行所用镜像的大小。

* + 1. **优化优选阶段**

通过预选阶段得到可用 Node 节点的集合，在优选过程中本文重点突出 GPU、显存的计算权重；同时充分考虑 Node 节点与 Pod 的匹配程度，针对性的实现优化。

定义 *scorecpu* 为 Node 节点上 CPU 的使用情况得分，具体计算如公式(3.7)所示，其中*Tcpu*为 Node 节点上 CPU 的资源总量；*U cpu* 为 Node 节点上已用 CPU 资源量。

*score*

*cpu*

=

1. -*U*

*cpu*

*T*

*cpu*

*cpu*

(3.7)

同理，定义 *scoremem* 为 Node 节点上内存使用情况得分； *scoredisk* 为 Node 节点上磁盘

使用情况得分；

*score*

*network*

为 Node 节点上网络情况得分；*scoregpu* 为 Node 节点上 GPU 使

用情况得分； *scorevram* 为 Node 节点上显存使用情况得分。具体计算分别对应公式(3.8)、公式(3.9)、公式(3.10)、公式(3.11)和公式(3.12)，其中*T* 为 Node 节点上某个资源的总量，

1. 为 Node 节点上某个资源的已使用量。

*score*

*mem*

=

*T* -*U*

*mem*

*T*

*mem*

*mem*

(3.8)

*score*

*disk*

=

*T* -*U*

*disk*

*T*

*disk*

*disk*

(3.9)

*score*

*network*

=

*T* -*U*

*network* *network*

*T*

*network*

(3.10)

*score*

*gpu*

=

*T* -*U*

*gpu*

*T*

*gpu*

*gpu*

(3.11)

*score*

*vram*

=

*T* -*U*

*vram*

*T*

*vram*

*vram*

(3.12)

除了计算该 Node 节点上各类资源的使用情况得分，还要考虑节点与 Pod 的匹配程度，采用相同的资源类型（CPU、内存、磁盘、网络、GPU、显存）分别表示 Pod 与 Node 上的资源情况，定义 Pod 向量、Node 节点向量如公式(3.13)、公式(3.14)所示，分别代表 Pod各类资源请求情况、Node 节点上各类资源剩余情况。

*Pod* = (*reqcpu* , *reqmem* , *reqdisk* , *reqnetwork* , *reqgpu* , *reqvram* ) (3.13)

*Node* = (*restcpu* , *restmem* , *restdisk* , *restnetwork* , *restgpu* , *restvram* ) (3.14)

使用夹角余弦度量两个向量之间的相似度，并以 *scorematch* 表示 Pod 应用请求资源与

Node 节点剩余资源的相似程度，即 Pod 应用与 Node 节点的匹配程度。

*score* 值越大

*match*

代表向量相似度越高，应用与节点的匹配度越高；反之匹配度越低。*scorematch* 具体计算如公式(3.15)所示：

*score*

*match*

=

*Pod Node Pod* *Node*

(3.15)

所以改进后的优选过程打分公式如(3.16)所示：

*score* = *w*

*cpu*



*score*

*cpu*

+

*w*

*mem*



*score*

*mem*

+

+

*w*

*disk*

*w*

*gpu*





*score* + *w*

*disk* *network*

*score* + *w* 

*gpu* *vram*

 *score*

*network*

*score*

*vram*

(3.16)

+

*w*

*match*



*score*

*match*

其中 *w* 代表各类资源对应的权重值，所有权重值的和为 1，本文将 CPU、内存、磁盘、网络这四类资源的权重设置为 *m* ，将 GPU、显存以及匹配程度的权重设置为 2*m* 。

* 1. 优化调度流程与实现
     1. **优化调度整体流程**

根据上述调度优化设计，确定改进后的整体调度流程如图 3.3 所示，具体步骤如下：首先，根据公式(3.6)计算待调度 Pod 列表中所有 Pod 的优先级值*V* (*Pj* ) ，根据*V* (*Pj* ) 的

值按从大到小的顺序存入 Pod 优先级队列 *PodQueue* 中；

其次*，根* 据公式(3.3)计算集群中所有 Node 节点的负载均衡情况 *LS* (*Ni* ) *，根 据 LS* (*Ni* )的值按从小到大的顺序存入 Node 节点队列 *NodeQueue*1 中；

然后，取 *PodQueue* 队列中队首 Pod，与 *NodeQueue*1 按照 K8s 优化预选策略进行节点过滤；若不存在可用 Node 节点，则将该 Pod 放入 *PodQueue* 队列队尾，等待下一次调度；

若存在可用 Node 节点，则得到 *NodeQueue*2 ，进行下一步；

最后，进入优选阶段，若 Pod 请求了扩展资源（GPU、显存资源）则调度扩展的优化方法；若没有，则从 *NodeQueue*2 中按照 K8s 默认优选策略，选出最优节点与 Pod 进行绑定操作；当前 Pod 调度结束，进行下一个 Pod 的调度。

开始

根据公式计算待调度Pod列

表中所有Pod的优先级值

根据优先级值从大到小存

入Pod优先级队列PodQueue

根据公式计算集群中所有

Node节点的负载情况

根据负载值按从小到大存

入节点队列NodeQueue1

针对所取PodQueue队首Pod，结合K8s默认预选策略对NodeQueue1进行过滤

否

是否存在可用Node节点？

是

调度失败。将该Pod放在

PodQueue队尾，等待下一次调度

得到节点队列NodeQueue2

根据K8s默认优选策略选出NodeQueue2

中最优节点，与Pod进行绑定

成功完成一次调度

否

PodQueue是否为空？

是

结束

图 3.3 优化调度整体流程图

* + 1. **优化调度器实现原理**

表 3.1 是 K8s 提供的实现自定义调度规则的方法，以满足不同应用场景的资源调度需求。根据表中对三种不同方法的具体做法与优缺点的描述，本文采用第三种调度扩展程序的方式，这种方式以外部扩展的形式实现自定义的优化调度策略，可以避免因为完全使用自定义调度器而导致缺少默认调度器内置基础调度算法的问题，且对默认调度器的源码无侵害，在默认调度组件的 yaml 文件中添加扩展调度程序的配置文件，自定义调度规则就能在集群中发挥作用。

表 3.1 添加调度规则方法对比

方法 具体过程 优缺点

1.修改源码

修改默认调度程序，重新编译后运行在

K8s 集群中

对原生代码侵入性高，不利于 K8s 的版

本升级和回滚

2.自定义调度器

编写自定义调度器，并添加

spec.scheduler 字段进行指定，作为独

立调度器运行在 K8s 集群中

满足业务场景需求但自定义调度器间无法同步通信，容易失去更佳的调度方案，

甚至调度失败

3.调度扩展程序

在默认调度器上进行外部扩展，

度器可以根据需求进行调用

对源码无侵害，既能选用默认调度器，也可以调用外部扩展程序满足业务需求

* + 1. **优化预选阶段实现**

使用 Golang 语言编写调度扩展程序。需要先在默认预选策略中添加优化预选策略的声明，如表 3.2 所示，该声明包含两个参数 name 和 func，name 参数指定扩展预选策略的名称，func 指定扩展预选策略的具体实现。

表 3.2 优化预选策略声明输入：优化预选策略声明

1. w\_GpuFilter = filter {
2. name:“w\_GpuFilter”,
3. func:func(pod Pod, node Node) {.} //优化预选策略细节
4. }

本文将优化的预选策略的名称定义为 w\_GpuFilter。func 方法中输入的参数为待调度的 Pod 和集群中 Node 节点集合，输出的参数为满足该过滤策略的 Node 节点集合，并将其加入优选队列中准备进行下一阶段的打分，不满足要求的节点被过滤掉。表 3.3 是具体策略的详细实现过程。

表 3.3 优化预选 w\_GpuFilter 策略输入：待调度 Pod、集群 Node 节点集合

输出：满足预选的 Node 节点集合

1. //计算所有 Pod 的优先级并存放在队列中
2. for i = 0:N-1
3. V(Pi) = init(Pi) + num(Pi) + images(Pi)
4. Pod\_Queue = put(Podj)
5. end for
6. //计算所有 Node 节点的负载均衡情况并排序
7. for j = 0:M-1
8. LS(Nj) = Load(Nj)/Soure(Nj)
9. Sort(LS)
10. end for
11. FilterResult //定义过滤结果集
12. //按顺序取 Pod 对 Node 进行过滤处理，结果存入结果集
13. for i = 0:N-1
14. NodeQueue[] = Filter\_Handler(Podi)
15. end for
16. //运行结束，http 返回状态
17. ResponseSet(Content - Type, application/json)
18. ResponseWrite(httpOK)
    * 1. **优化优选阶段实现**

优化优选阶段的实现同理，需要先在默认优选策略中进行声明，如表 3.4 所示，声明语句包括代表优选打分策略名称的 name 参数，以及代表优化优选策略具体实现方法的func 参数。

表 3.4 优化优选策略声明输入：优化优选策略声明

1. w\_MatchPriority = prioritize {
2. name:“w\_MatchPriority”,
3. func:func(pod Pod, node[] Node) {.} //优化优选策略细节
4. }

本文将优化优选的实现定义为 w\_MatchPriority。输入的参数为待调度的 Pod 应用和备选 Node 节点集合，经过优化优选阶段后输出最匹配的 Node 节点，与 Pod 进行匹配。表 3.5 是具体策略详细实现过程。

表 3.5 优化优选 w\_MatchPriority 策略输入：待调度 Pod、候选 Node 节点集合

输出：分数最高的 Node 节点

1. //计算各类资源使用情况总分数
2. for i = 0:N-1
3. for j = 0:M-1

4.

*match*

*Podi Nodej*

= *Podi* *Nodej*

5. weight[] //定义各个分数的权重值

6. Score\_LSpodi,podj = Cal(cpu, mem, disk, network, gpu, vram, match)

7. Pod\_Node\_Score(Score\_LSpodi,podj, weight[])//计算加权值

1. end for
2. end for
3. //对打分排序
4. Pod = Sort(Pod\_Node\_Score)
5. //运行结束，http 返回状态
6. ResponseSet(Content - Type, application/json)
7. ResponseWrite(httpOK)
   1. 本章小结

本章首先对 K8s 默认调度器及调度策略进行详细介绍；然后对战场分析系统应用在K8s 调度过程中存在的问题进行分析，针对 Pod 优先级、GPU 和显存资源进行重点改进；最后，进行调度优化后的整体流程进行描述，并通过比较选择出最合适的优化调度实现方法。

1. 基于 K8s 的容器应用迁移优化设计与实现

本章主要介绍基于 K8s 的容器应用迁移优化设计与实现过程。首先对现有的容器迁移方法进行分析。然后针对不足之处结合容器分层和内存迁移的特点，对集群容器迁移提出优化方案，在分阶段迁移（迁移准备阶段和正式迁移阶段）的基础上，对文件系统进行对比迁移，对内存迁移的条件进行优化改进，降低迭代次数。最后确定迁移的整体流程，并通过代码实现各个阶段的迁移。

* 1. 问题分析

在研究课题的过程中发现，容器云平台在运行和维护过程中[53]面临两个问题：一是具体的容器业务易受到战场复杂环境的影响，造成容器故障进而增加应用完成的时间，甚至造成整体业务中断；二是当主机维护、系统硬件升级时，需要平台管理员将容器应用进行转移以完成阶段性的维护工作。这些都可以借助容器迁移解决。当前针对虚拟机的在线迁移在学术和工业领域都有相对成熟的支持，但 Docker 容器迁移仍在不断发展完善中，并且由于两者虚拟化技术之间的差异，虚拟机的迁移方法并不能直接应用在容器迁移中。

现阶段对 Docker 容器迁移的研究中有不少卓有成效的迁移机制，比如 Nadgowda S等人的 Voyager 机制以及 Ma Lele 等人研究优化的迁移方案。虽然这些研究都达到了不同物理机之间容器应用迁移的目的，但是 Voyager 方法仅从容器运行时角度入手，忽略了目标端 Docker 架构对应用的重新管理能力；Ma Lele 等人的方法不适合由多物理机所组成集群下的容器迁移；一些针对 Pod 的迁移更是选择先关闭或挂起主机后再进行迁移操作，这种冷迁移的方式会对源主机中的各种服务造成非常大的影响，因为一个 Pod 所提供的服务可能由一个或多个容器共同完成，所以用户感受到的中断时间相对较长。

因此，针对上述问题，本文对容器文件系统层级关系和内存迁移方法进行深入分析，利用 CRIU 技术与 Pre-Copy 机制，结合 K8s 的集群特点提出优化方案，对 K8s 集群中容器应用迁移的现有技术与方法进行改进，提供一套适用于 K8s 云平台的 Docker 容器应用迁移方法。

* 1. 基于 K8s 的容器应用迁移优化设计

由于迁移过程中涉及到编排信息、容器配置信息以及镜像转移的问题，所以将迁移过程分为迁移准备和正式迁移两个阶段，并在数据传输结束后进行应用的恢复。

* + 1. **迁移准备阶段设计**

迁移准备阶段主要完成编排信息的转移、目标主机申请资源、创建新容器的工作。迁移准备阶段的重点是创建一个在资源上与源容器完全相同的容器，但该阶段容器并不运

行，只为后续的镜像与文件系统迁移提供配置好的环境。

由于 K8s 中的基础单位是 Pod，所以编排信息要从运行在集群中的 Pod 的 yaml 文件中收集，收集的信息主要包括 Pod 的名称、Pod 的 IP 地址、Pod 所在命名空间以及 Pod中的容器信息，后续通过该编排信息恢复应用。并额外添加一个标志位，用来判断该应用是否是“迁移应用”。如果是“迁移应用”，则目标主机创建容器应用的过程与 API Server创建容器应用的常规过程有些不同，也就是在资源申请以及创建容器完成后，并不马上运行容器；而 API Server 一般情况下创建的容器应用在分配好资源与运行环境后是直接运行的。

* + 1. **正式迁移阶段设计**

当前各类学者对迁移的深入研究已经能够实现容器应用迁移的目标，但在迁移过程中大量的容器文件数据进行了多次传输，这就造成了额外的迁移时间[28]。所以将正式迁移分为容器文件、容器内存两个部分来进行迁移优化，下面是对各部分迁移优化过程的详细说明。

1. **容器文件系统迁移设计**

容器是一种由各种镜像组成的分层结构。容器迁移重要的是将文件系统和内存状态传输到目标主机，而文件系统正好可以依据其分层的特性进行迁移优化。容器可支持多种文件系统，如 aufs、overlay 等[55]。文件系统由于类型不同在实现方式上有所差别，但都属于联合目录结构的文件系统，以分层的方式分为多个层级的文件，再以统一视角提供给用户进行查看使用。

本文 Docker 所采用的是 overlay2（默认的）文件系统驱动，其目录结构如图 4.1 所示，分为三层：存放代码运行所需环境的镜像只读层、存放容器具体运行状态或中间数据的读写层、以及存放文件系统根目录的联合挂载层。

容器结构 文件系统结构

联合挂载

读写层

(容器层)

只读层

镜像层1

(镜像层)

镜像层2

镜像层3

根目录

变化层

基础层

图 4.1 overlay2 文件系统分层与容器的对应结构图

针对容器文件系统的迁移通常有两种方式，一种是全部文件迁移，在目标端恢复时再

将内容全覆盖。这种方式虽然简单直接，但在这些文件中存在相当一部分的文件数据在容器运行时并不会发生改变，比如只读层的基础镜像数据，对其进行迁移只会造成数据的重复传输，增加网络开销和迁移时间。另一种是部分文件迁移，利用容器分层和中心镜像仓库对文件系统进行分层迁移。但是该方法设定的中心镜像仓库容易受网络带宽的影响。

所以本文在迁移时，利用文件系统的分层特性，将文件读写层直接在节点间传输，只读层通过占位符代替其在读写层的位置以保证结构的快速恢复。其中，只读层占位符的信息包括该位置的镜像 ID、镜像名称以及所在目录。当读写层迁移结束后，根据占位符恢复文件系统结构。

1. **容器内存迁移设计**

内存迁移阶段需要对容器内存进行快速并有效的迁移，“快速并有效”就要求应用服务的暂停时间尽可能的短，使用户尽可能感觉不到因迁移而造成的应用服务的中断。所以宕机时间和迁移时间是衡量迁移的重要指标。

从虚拟化技术上来说，容器迁移并不能直接使用虚拟机迁移的方法；但是从内存角度来说，内存拷贝策略可以应用于容器迁移中。针对不同的应用需求，大量学者已提供了各种各样的迁移方法和技术支持，但大多是针对 Pre-Copy 大量迭代问题的改进。例如Post-Copy 通过同步最小工作集的方式加快了应用服务的转移，达到减少内存重复传输的目的，但迁移后续剩余内存数据的时间较长影响了服务性能；CR/RT-Motion 通过记录和重放操作日志文件的方式实现迁移，避免了内存的大量拷贝，但增加了一定的文件传输量。

所以考虑到对大量迭代问题的解决，本文结合 CRIU 与 Pre-Copy 机制，优化停机迁移的条件，降低容器应用迁移时间。 CRIU 首先收集并备份内存数据，然后 Pre-Copy进行内存数据的循环拷贝。内存迭代拷贝示意图如图 4.2 所示，首先，Pre-Copy 把当前的内存页面传输到目标端主机，此时源主机的容器正常运行；然后，多次执行以下操作，即把继续运行所产生的脏页数据拷贝到目标端主机并进行内存状态的更新；最后，当待迁移Pod 应用满足停机迁移的条件时暂停运行，并进行 Restore 恢复。

Checkpoint

创建内存检查点

内存

停机迁移

**源端**

**目标端**

图 4.2 内存迭代预拷贝示意图

由于 Pre-Copy 的一系列内存拷贝及迭代同步操作，造成深度学习这类内存读写频繁的应用执行了大量重复的迁移，因此本文修改停机迁移条件以减少迭代次数，尽可能快的

让容器在目标主机恢复运行。执行停机迁移并减少重复迁移的条件如下：

①脏页面大小达到阈值：脏页面阈值要根据实际的磁盘读写速率、网络带宽以及待迁移内存大小进行计算。

②迭代次数达到最大：为了优化深度学习类等内存读写频繁应用的内存重复迁移问题，降低了原本的迭代最大值。

当容器应用的文件和内存状态都传输到目标端主机后，接下来要在目标端主机进行迁移容器应用的恢复。恢复时目标端主机利用 Kubelet 根据编排信息执行 Restore，并判断应用的运行是否正常，若应用恢复正常，则删除目标端的编排信息和源端的应用，结束迁移；若应用恢复非正常，则检查应用是否达到规定的副本数量，若没有则执行 Restore 继续恢复操作，若已达到，说明迁移失败，直接删除目标端的应用和编排信息，结束此次迁移。

* 1. 基于 K8s 的容器应用迁移实现
     1. **迁移整体流程**

本文面向 K8s 的 Docker 容器应用迁移的整体流程如图 4.3 所示。分别是迁移准备阶段和正式迁移阶段。首先迁移准备阶段根据编排信息在目标端主机上创建容器，并根据信息申请所需资源；然后进入正式迁移阶段，分别迁移容器文件系统和容器内存；最后进行应用的恢复，在目标端完成迁移应用的运行。

开始

源端Kubelet获取

被迁移应用的编排信息

迁移容器应用的

镜像文件

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 源端Kubelet对容器应用  循环执Checkpoint | |
|  |
|  |  |
| 内存脏页面是否达到阈值？  否  是否达到最大迭代次数？ | | | |
|  | | | 否 |

结束

图 4.3 K8s 的容器应用迁移整体流程图

* + 1. **迁移准备阶段实现**

迁移准备阶段主要完成编排信息的转移、目标端申请资源以及创建新容器的工作。由于容器运行在 Pod 内，所以本文在 K8s 中自定义编排信息 PodConfigMessage，用于源端待迁移应用信息的获取，数据成员信息如表 4.1 所示。其中 PodMig 有两种值，Y 代表“迁移应用”，N 代表“本地应用”，值的不同代表后续操作有所不同。

表 4.1 PodConfigMessage 编排信息

名称 说明

PodName Pod 名称

PodIP Pod 的 IP 地址

PodN-S Pod 所在的命名空间

PodMig 标识当前 Pod 是否是迁移过来的

ContainerMessage[] Pod 中的容器信息数组，包括容器名称、容器标识信息

本文通过 Golang 语言实现迁移准备阶段，定义为 Pre\_Mig 方法，具体如表 4.2 所示。该方法根据输入的待迁移应用，完成目标端主机上申请资源和创建容器的工作，而是否创建容器根据 PodMig 的值来判断。本文在 K8s 的管理组件中添加了针对 PodMig 字段的控制代码，如果是“迁移应用”，就将该信息发给目标端主机准备进行迁移；接着目标端执行迁移的准备工作，下载对应的编排信息文件并申请资源，完成容器的创建。如果监听到是“本地应用”，则由 API Server 创建容器并直接运行。

表 4.2 迁移准备 Pre\_Mig 函数输入：待迁移应用

输出：目标端的容器是否创建成功

1. //源端收集待迁移应用的编排信息
2. kubelet.getMessage
3. //根据 PodMig 字段信息判断是否是迁移应用
4. if(PodMig == Y)
5. //目标节点下载编排信息文件
6. kubelet.downMessage
7. //创建 Pod 并申请资源
8. kubelet.createPod(Message)
9. //创建容器，但并不启动
10. kubelet.createContainer(ContainerMessage[])
11. api\_server.createPod //否则正常创建 Pod，直接运行
    * 1. **正式迁移阶段实现**

正式迁移阶段分为容器文件系统迁移和容器内存迁移，在编码实现过程中，将方法分别定义为 For\_Overlay\_Mig 和 For\_Mem\_Mig。并通过 Con\_Restore 恢复应用。

1. **容器文件系统迁移实现**

容器镜像是容器运行的基础，文件系统驱动将镜像组成具有一定结构的文件系统。容器文件系统迁移主要完成读写层的迁移。容器文件系统迁移函数如表 4.3 所示，首先获取容器文件系统的各个分层；然后通过判断该层性质确定是否直接迁移，如果是读写层则直接迁移，如果是只读层则将其信息进行记录形成占位符，随读写层迁移；最后根据占位符恢复只读层的文件内容。

表 4.3 容器文件系统迁移 For\_Overlay\_Mig 函数输入：待迁移应用

输出：容器文件系统是否迁移成功

* 1. for i = 0:N-1 //循环获取容器文件系统
  2. //从容器中获取文件系统目录
  3. kubelet.getOverlay
  4. //判断是哪种类型层级
  5. if(lower)

6. //lower 类型

7. setFlag(lower) //占位符代表只读层在读写层的标识

1. end if
2. //upper 层
3. //读写层带上占位符直接迁移
4. upper\_Mig(Flag)
5. //恢复文件系统目录
6. kubelet.setOverlay(Flag)
7. end for
8. **容器内存迁移实现**

容器内存迁移函数如表 4.4 所示。迁移过程中的最大迭代次数设置为 15 次。内存脏页阈值的计算和设定具体如下：①计算出理论上的宕机时间，包括两次磁盘读写时间，一次网络传输时间；②取磁盘读写速率与网络带宽速率的平均值作为整体的迁移速率；③将理论宕机时间与整体迁移速率相乘，即为迁移中具体的脏页阈值大小。

表 4.4 容器内存迁移 For\_Mem\_Mig 函数

输入：待迁移应用

输出：容器内存是否迁移成功

* 1. for i = 0:N-1 //内存循环 Checkpoint
  2. CRIU.checkpoint(Container) //对当前容器创建内存检查点
  3. while(number<15 || current>value) //判断停机迁移条件

4. Dirty\_Mig() //不满足则内存脏页迭代迁移

1. end while
2. Stop\_Mig(rest) //满足则进行停机迁移
3. end for

容器文件和内存都传输到目标端后，目标端进行应用的恢复操作，并判断应用是否正常运行，恢复函数如表 4.5 所示。应用恢复成功后删除目标端下载的编排信息。

表 4.5 应用恢复 Con\_Restore 函数输入：传输的文件数据、内存数据

输出：应用是否迁移成功

1. for i = 0:N-1 //循环 Restore
2. CRIU.restore(Container, Overlay, Mem) //恢复应用文件和内存
3. if(num(Container) < N-1) //还未达到指定数量

4.

1. end for

continue

1. if(run(Pod, Container)) //恢复的应用是否正常运行

7.

1. else

迁移成功

1. 迁移失败
2. kubelet.deleteMessage
3. end if
   1. 本章小结

本章主要介绍基于 K8s 的 Docker 容器应用迁移的优化设计与实现。首先针对当前存在的问题与不足进行分析。然后针对迁移过程中遇到的文件数据和内存重复传输的问题，将迁移分为迁移准备阶段和正式迁移阶段完成迁移与应用恢复，并进行详细设计。最后确定迁移的整体流程，完成相应函数的实现。

1. 基于 K8s 的 Fabric 区块链高可信设计与实现

本章主要介绍基于 K8s 的 Fabric 区块链高可信存储的设计与实现过程。首先对基于战场分析应用的 K8s 云平台中实现区块链应用的现有方案进行分析。其次针对 Fabric 在K8s 集群中运行时组件启动的时间不同，从基础组件和可扩展组件两个部分进行云化部署设计。最后确定战场应用数据的数据结构，并编码实现数据存储和查询的扩展链码。

* 1. 问题分析

信息技术在战场领域得到广泛应用，战场信息处理数据的多样化（图片、视频数据等）使得信息管理对象的范围扩大，管理的复杂程度也随之增加。虽然云计算及虚拟化等技术的不断发展为军事信息处理提供了不同的解决思路，但是在战场分析应用的军事特殊场景下，怎样实现单个主机或服务器之间在保证安全的情况下同步数据并共享信息，以达到新加入主机获取信息的完整性，这是军事云计算的发展瓶颈，也是课题研究的一大难点。而区块链技术可以解决可信存储的问题，课题选用 Fabric 区块链作为底层技术支持，利用智能合约及链码的实例化实现扩展功能，将自定义数据存储在区块链上并能进行数据查询。同时将 Fabric 部署在 K8s 上对其具有监控管理、灵活配置和方便扩展的好处。

但是，Fabric 中链码是实现扩展功能的关键，它是由智能合约实例化运行后得到的。链码编写完成后并不能直接在区块链网络中发挥处理交易提案的作用，需要通过安装、编译和启动操作后使用。具体过程如图 5.1 所示：首先使用 install 命令将链码安装在区块链节点上；接着使用 instantiate 命令对已传入链码容器中的链码进行编译；然后对指定链码进行打包，生成链码镜像供后续重复使用；最后启动链码容器，完成实例化操作，时刻准备处理提交的交易提案。

开始

Peer节点调用

Docker Daemon

利用基础镜像构建链码容器作为准备

Peer将链码源码传入链码准备容器，完成链码编译

结束

启动并运行链码容器

Peer再次调用Docker Daemon生

成链码镜像供后续重复使用

图 5.1 Fabric 链码生命周期

从链码生命周期管理的过程中能够发现两个问题。一是智能合约容器的生命周期独立于 K8s 管理体系之外；二是链码容器的构建与启动在多节点 K8s 环境中存在跨节点调度使用的问题。虽然现有方案能够实现将 Fabric 部署在 K8s 中运行，但都是通过一定的辅助技术部署功能组件，并没有在 K8s 的管理环境中进行集成，尤其是最重要的链码部分。因此，本文深入研究了区块链的底层基础，根据 Fabric 组件的两种不同特性对云化集成

进行设计；通过对链码生命周期重新进行管理，解决了 K8s 管理 Fabric 组件以及链码跨节点实例化的问题；搭建了基于 K8s 的联盟链应用，为各个独立作战单元提供数据可信存储和数据查询服务。

* 1. 基于 K8s 的 Fabric 区块链高可信设计

根据 Fabric 组件两种不同的特性，可以将组件分为基础组件和可扩展组件。基础组件要保证区块链网络正常运行，比如 CA、Peer、Order 等需要在一开始就启动；可扩展组件指对区块链进行自定义操作的组件，比如链码在实例化智能合约时才真正启动。本文根据基础组件与可扩展组件的运行区别，将 Fabric 区块链的组件分开部署在 K8s 上运行。

* + 1. **基础组件部署设计**

Fabric 是一种组件多、云化集成难度大且复杂的区块链网络，其中基础组件可以从一开始就启动运行。由 2.3.3 小节所介绍的 Fabric 基础理论知识可知，在 Fabric 的运行架构中基础组件由多个虚拟节点共同发挥作用，与其他组织相互独立。所以在将基础组件进行云化时，为了继续保持各组织之间的独立性，将 Fabric 的组织与 K8s 的 Namespace 命名空间一一对应起来，使其工作互不干扰。

K8s 中的 Namespace 命名空间可以划分隔离开不同的虚拟集群。在将组件组织与

Namespace 进行对应时，可以利用域名进行划分，这样既能使组织之间继续保持相互独立，又充分利用了 K8s 的 DNS 域名服务。组件组织与 Namespace 的对应关系如图 5.2 所示。

**Pod**

Peer0

**Pod**

CA

SVC

SVC

**Pod**

Peer1

**Pod**

CLI

SVC

SVC

**Pod**

...

PeerN

Namesapce:org1

Namesapce:orgorderer1

...

...

Namesapce:orgN

Namesapce:orgordererN

图 5.2 Fabric 组件与 Namespace 对应关系图

根据对等节点组件和排序服务组件两种不同的组织，对应 K8s 的 Namespace 进行组件划分。将域名 org1 中的对等节点组件对应 K8s 中 org1 命名空间上的 Pod，一共 3 种类型：Peer-Pod（区块链网络中 Peer 对等节点的 Pod）、CA Server-Pod（区块链网络中 CA认证服务的 Pod）、CLI-Pod（区块链网络中操作 Peer 节点、Channel 通道或 Chaincode链码的 Pod）。将域名为 orderer1 的排序服务组件对应 K8s 中 orderer1 命名空间上的 Pod（负责排序工作）。

除上述组件对应划分之外，还需要提前下载好 Fabric 的镜像文件，一些关键镜像文件和其承担的作用如下表 5.1 所示。

表 5.1 Fabric 关键镜像列表

IMAGE TAG 说明

hyperledger/fabric-tools x86\_64-1.0.0 本地客户端镜像，执行基本操作

hyperledger/fabric-orderer x86\_64-1.0.0 排序服务镜像

hyperledger/fabric-peer x86\_64-1.0.0 Fabric 网络节点镜像

hyperledger/fabric-ccenv x86\_64-1.0.0 链码实例化的编译环境

hyperledger/fabric-ca x86\_64-1.0.0 服务器本地 CA Server，执行客户端相关操作

hyperledger/fabric-baseimage

x86\_64-0.3.1

基础镜像，内嵌了 JDK、Golang 等环境，用

于生成其他镜像

* + 1. **可扩展链码部署设计**

链码是一种动态的可扩展的功能组件，它的功能是通过实例化智能合约实现的，在一般应用环境下，链码和 Peer 节点的功能由同一个主机承担，这样链码的安装、编译与启动都在同一个 Docker 容器中。但是在多主机场景下，链码的完整生命周期可能被分割开，链码的安装、编译以及启动很有可能需要在不同主机上完成，所以在云上运行时需要提供链代码在不同节点之间传输的能力。本文在动态链码部署上主要解决两个问题，一是链码容器如何获取 Peer 节点 IP 地址；二是链码如何跨节点实例化。

**1）网络设计**

Fabric 网络中的链码运行在主机的 Docker 容器中，而链码容器的创建要通过 Peer 容器调用 unix:///var/run/docker.sock 接口实现，但通过该接口创建的容器在数据交互上存在两个问题：一是主机默认的 DNS 解析无法获得 Peer 的域名；二是该容器脱离了 K8s 的管理体系。所以在网络域名解析 DOCKER\_OPTS 中，为 K8s 的每一个 Node 节点添加 K8s的 DNS 和主机的 DNS。

**2）链码跨节点实例化设计**

由 5.1 小节可知，链码的作用是处理交易提案，实现对区块链账本的数据更新，而维护数据保障安全性正是区块链的核心部分，所以实现链码的跨节点实例化是区块链应用成功运行的重点。本节通过对 Fabric 组件的功能二次开发将链码列入到 K8s 管理中。

原本的 Peer 容器对链码容器进行整个生命周期的管理时，指定 Docker 引擎的服务地址并在该引擎上完成链码的编译和启动。但是在多主机的 K8s 环境中，链码编译和链码部署所在的主机有可能并不是同一个，即需要链码服务的主机上没有链代码。所以本文分离链码编译和链码部署，同时借助 Harbor 镜像仓库实现镜像转发。首先链码编译成镜像的过程仍由 Peer 容器指定的原始 Docker 引擎完成；然后将链码编译镜像上传至镜像仓库中；接着在集成了集群 SDK 的 Peer 中开始链码部署操作，拉取镜像进行实例化链码操作完成部署。对链码编译和部署的分离改造涉及到 Peer 客户端，具体的改造实现将全部封装在 K8s 的 controller 中，函数接口有 createConfigMap、deleteConfigMap、createDeployment、deleteDeployment、createSVC 和 deleteSVC。

* 1. 基于 K8s 的 Fabric 区块链高可信实现
     1. **云化部署整体流程**

基于 K8s 的 Fabric 区块链应用运行整体流程如图 5.3 所示。首先在搭建好的 K8s 集群中，利用控制节点的 Kubectl 启动 Fabric 网络，基础组件随网一同启动；然后客户端编写链码，并将链码打包安装在任意节点上；接着在该节点上进行链码编译，同时生成链码镜像，上传至镜像仓库保存；当某个节点发起一次交易时，也就是对账本数据进行操作时，该节点先从镜像仓库中拉取链码镜像，然后启动链码容器，执行链码功能完成数据操作，返回数据读写集；最后链码执行结束后直接退出链码容器，并返回操作结果给节点客户端。

开始

控制节点的Kubectl启动基

础组件

客户端C编写自定义链码并

安装在节点A上

节点A编译链码，生成镜像

链码镜像上传仓库保存

B节点从仓库拉取镜像，

启动链码容器

调用链码执行数据操作，

并返回读写集

否 退出链码容器，

另一节点B是否发起交易

是

执行结果返回给C

结束

图 5.3 基于 K8s 的 Fabric 区块链应用运行整体流程

* + 1. **基础组件部署实现**

**1）生成配置文件**

由于 K8s 中并没有 Fabric 各组件的环境配置，所以需要提前编写并生成 Fabric 关键组件的配置文件。直接编写 5 个模板文件，再由这些模板文件生成对应组件的配置文件。具体模板文件名称和内容分别如下：

①fabric\_template\_namespace.yaml：定义 Fabric 域名与 K8s 命名空间的对应关系。②fabric\_template\_cli.yaml：定义区块链网络中进行操作的 Pod 模板，用于对通道的

创建、链码的安装等操作。

③fabric\_template\_ca.yaml：区块链网络中进行 CA 认证服务的 Pod 模板，该 yaml 模板文件需要指定 selector 选择器将 deployment 与 service 进行绑定才能真正发挥认证作用。

④fabcric\_template\_orderer.yaml：Orderer 排序服务的 Pod 模板。⑤fabric\_template\_peer.yaml：区块链网络中对等节点的 Pod 模板。对等节点不仅要构

成基础区块链网络基础，还要将 K8s 集群中 Node 节点的 docker.sock 接口映射给 Peer 的容器引擎，以实现区块链网络在 K8s 集群环境中的运行。

**2）启动网络并查看**

使用 docker-compose v1.8 版本，在命令终端进入项目文件夹，执行 docker-compose up命令启动 Fabric Network，如图 5.4 所示是启动结果，从控制台可以看到各类配置文件的加载与启动，包括 CA 认证、Peer 服务等。

图 5.4 Fabric 网络启动结果图

网络启动成功后，在新的命令行终端下可以查看启动成功的基础组件。运行 docker ps命令查看基础组件的启动数量、类别、状态以及名称等信息，结果如图 5.5 所示。本文启动了一个 CA 认证服务，两个 Peer 对等节点服务以及一个 Orderer 排序服务，当前 Fabric网络正常，后续可以使用底层 SDK 完成链码编写。

图 5.5 Fabric 基础组件状态图

* + 1. **战场分析数据可扩展链码部署实现**

智能合约是动态链码运行的基础。有关数据存储链码的实现分为两个部分，首先确定所需存储的数据字段与智能合约的结构体；然后对存储和查询的关键代码进行阐述。

1. **网络实现**

网络部分主要是通过修改配置文件的参数以达到通过 DNS 查找 IP 的目的。以 IP 为10.0.0.10 的链码容器为例，它所在主机的 IP 地址是 192.168.1.10，为使得链码容器可以解析到主机，进行如下修改：

步骤一，编辑修改 Docker 默认配置文件，在 DOCKER\_OPTS 下设置网络参数，添加K8s 使用的 DNS，这一步是非常重要的，是解析 Peer 域名的关键：

--dns=10.0.0.10 --dns=192.168.0.1 --dns-search \ default.svc.cluster.local--dns-search svc.cluster.local步骤二，运行以下命令重启 Docker：

systemctl daemon-reload systemctl restart docker systemctl restart docker.service

1. **战场分析关键数据结构体**

战场分析数据实现可信存储的关键在于如何将数据存储在区块链中，本文结合智能合约的实现，将战场分析应用处理后得到的关键数据放在智能合约的数据结构当中。除了处理后的关键数据，智能合约中还需要存放处理该任务的主机 IP。将该结构体定义为 Object，其结构体内容如表 5.2 所示。

表 5.2 Object 结构体设计表

名称 数据类型 说明

TargetID String 当前目标编号

HostTime String 当前目标处理时间

TargetType String 当前目标类型

TargetBox String 目标框大小

TargetIntention String 目标意图

TargetThreat String 目标威胁度

HostIP String 处理目标的主机 IP

PreData PreArray[] 之前存放的所有数据记录

其中最后一个数据字段 PreData 代表在此之前存放的所有数据记录，以 PreArray 结构体存放，具体内容如表 5.3 所示。其中 PreID 代表当前交易块的序号，与 PreArray[]数组位置对应。

表 5.3 PreArray 结构体设计表

名称 数据类型 说明

PreID String 序号

Object Object 之前已经存放的数据记录

1. **战场分析数据可信存储关键代码**

实现数据存储与查询，也就是通过重新编写 Init、Invoke 函数的链码接口达到扩展功能的目的。其中 Init 函数的作用是对账本数据和链码进行初始化；Invoke 函数实际上是实现对账本数据的操作，对账本的操作包括保存数据和查询数据两种。重写的 Init 函数具体内容如下表 5.4 所示。其步骤是首先获取传入的参数；然后判断参数长度是否为 2，若是则将状态写入账本中，若不是则直接结束；最后返回状态信息，显示是否操作成功。

表 5.4 重写 Init 函数输入：数据自定义结构体，链码通用接口

输出：Peer 节点响应结果

* 1. //获取当前参数
  2. args = GetStringArgs()
  3. //判断参数长度是否为 2
  4. if(args != 2)
  5. //长度不为 2，返回错误
  6. Error(“参数不正确”)
  7. end if
  8. //长度为 2，将数据保存在账本
  9. putstate(args[0], []byte(args[1]))
  10. //初始化成功
  11. Success()

重写 Invoke 函数具体内容如下表 5.5 所示。其步骤是首先获取传入的参数；然后判断参数长度是否为 1，若参数长度为 1 则根据参数名称执行具体数据操作函数，若不是则直接结束；最后调用具体方法完成操作，并返回状态信息，显示是否操作成功。

表 5.5 重写 Invoke 函数输入：数据自定义结构体，链码通用接口

输出：Peer 节点响应结果

1. args = GetStringArgs()//获取当前参数
2. if(args != 1) //判断参数长度是否为 1
3. Error(“参数不正确”) //长度不为 1，返回错误直接结束
4. end if
5. //获取链码调用参数
6. name = GetFunctionAndParameters()
7. if(name == “w\_query”)
8. w\_query(args)
9. Success()//实例化链码成功
10. end if
11. Error(“操作失败”)

其中 w\_query()方法的实现如表 5.6 所示。首先根据参数长度判断参数是否指定正确，若错误则直接退出，若正确则根据指定的 Key 值调用方法对账本进行操作，并将操作结果返回。

表 5.6 w\_query 函数输入：args[]参数

输出：数据查询结果

1. if(args != 1) //判断参数长度是否为 1
2. Error(“参数指定错误”)
3. end if
4. result = getState(args[0]) //根据关键字进行数据查询
5. Success(result)
   1. 本章小结

本章主要完成基于 K8s 集群的 Fabric 区块链高可信存储的设计与实现过程。首先对基于战场应用的 K8s 云平台中的数据存储进行问题分析。其次针对 Fabric 基础组件与可扩展组件的不同特点，分两部分对 Fabric 在 K8s 上的运行进行设计。最后确定战场分析数据的数据结构，实现数据存储和查询的自定义扩展链码。

1. 平台实现与测试分析

本章将通过实验验证第三章改进的基于战场分析的 K8s 资源调度、第四章改进的基于 K8s 的容器应用迁移以及第五章基于 K8s 的 Fabric 数据可信存储。本章首先详细介绍了平台的实现环境，包括平台拓扑结构、节点配置和引擎所用版本。然后对云平台进行功能测试。最后进行性能测试，通过实验验证了本文第三章改进的调度策略的优异性和第四章基于 K8s 的容器应用迁移方法的有效性。

* 1. 平台环境配置

本文设计并搭建基于战场分析应用的 K8s 云平台，一共 5 个主机，采用虚拟机与物理机相结合的方式进行部署，集群拓扑结构如图 6.1 所示。各个节点的具体信息如表 6.1所示，整个集群配置 1 个 Master 管理主节点（server002），负责集群统一管理和资源调度；配置 3 个计算节点（server001、server003 以及 server006），运行具体的应用；配置1 个 Harbor 镜像仓库（server005），实现镜像存储。CPU 为 Intel(R) Xeon(R) W-2123 CPU @3.60GHz×8，GPU 为 Tesla P100。

K8s 管理平台

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| K8s | 集群  管理 | | |  | | |  | | | | 管理 | | | 上传镜像  拉取镜像 |
|  | | | | | | |
|  | | | 管理 | | | |
| Master |  |  |
|  | |  |  | | |  |  | | |  |
|  | |  |  | | |  |  | |  |
|  |
|  |  | |  |  |  |  | |
|  |  | |  |  |  |  | |
|  | |  | | |  | |

Worker1 Worker2 Worker3

图 6.1 K8s 集群物理拓扑结构图

表 6.1 集群各节点配置信息

主机名称

CPU

核数

内存

（GB）

GPU

数量

显存

（GB）

磁盘容量

（GB）

网络带宽（Mbps）

节点

角色

IP 地址

**server001** 2 4 0 -- 20 1000 Worker 192.168.1.201

**server002** 1 2 0 -- 20 1000 Master 192.168.1.202

**server003** 2 4 0 -- 20 1000 Worker 192.168.1.204

**server005** 4 8 0 -- 500 1000 Harbor 192.168.1.179

**server006** 4 8 1 32 40 1000 Worker 192.168.1.177

每个节点都配置 Ubuntu16.04 版本操作系统，Docker 配置 17.09.1-ce 稳定版本，K8s选择 v1.9 稳定版本，分别如图 6.2、图 6.3、图 6.4 所示。

图 6.2 K8s 版本

图 6.3 Docker 版本

图 6.4 Ubuntu 版本

* 1. 云平台功能测试

本文搭建面向战场分析应用的 K8s 云平台，主要包括平台总览模块、应用部署模块、镜像管理模块以及可信数据管理模块。如图 6.5 是可视化管理的登录界面，在浏览器上通过 <https://192.168.1.201:88/user/login> 的 URL 进入登陆页面，输入账号和密码登录后访问平台。

图 6.5 登录界面

* + 1. **平台总览模块功能测试**

平台总览模块为用户提供平台下集群的资源情况信息、集群中节点信息的展示。如图6.6 所示，用户通过“平台总览-资源情况”菜单栏，可以看到平台下集群中 CPU、内存的实时使用情况，集群中工作节点的名称、IP 地址、运行时间以及就绪状态等信息。

图 6.6 平台总览页面

* + 1. **应用部署模块功能测试**

应用部署模块为用户提供容器应用的创建、容器编辑、停止以及迁移功能。如图 6.7所示，用户通过界面能够看到平台内所部署容器应用的相关信息，包括应用名称、所在主机、应用创建时间、运行状态等，并能对应用进行相关操作；可以创建新的容器应用，如图 6.8 所示通过“创建”按钮指定所用镜像名称和自定义代码位置进行应用创建操作；可以对容器应用进行迁移操作，如图 6.9 所示通过“迁移”按钮输入目标主机的 IP 地址进行应用迁移操作。

图 6.7 应用部署界面

图 6.8 创建容器应用

图 6.9 迁移容器应用

* + 1. **镜像管理模块功能测试**

镜像管理模块为用户提供自定义镜像创建、下载和删除功能。图 6.10 是平台镜像管理界面，可以看到镜像的名称、类型（基础镜像还是自定义镜像）和当前镜像版本，可以通过“创建”按钮根据配置文件生成新的自定义镜像；可以通过“删除”按钮删除本地镜像；通过“下载”按钮可以从镜像仓库下载所需镜像。

图 6.10 镜像管理界面

* + 1. **可信数据管理模块功能测试**

可信数据管理模块为用户提供关键数据信息存储和信息查询功能。点击可信数据管理，跳转至基于区块链的数据存储与查询部分。图 6.11 是添加信息页面，用户可以根据关键字段进行数据存储，包括检测到目标的时间、类型、目标意图、目标框大小、威胁度和该数据来源的主机 IP 地址。

图 6.11 可信数据管理界面

图 6.12 是关键数据 IP 查询界面，图 6.13 是查询结果显示界面，根据所存入数据主机的 IP 地址查询链上存储的关键数据信息，并在页面进行展示。可以通过“返回首页”按钮进入系统首页。

图 6.12 关键数据 IP 查询界面

图 6.13 数据查询结果界面

图 6.14 是区块链后台操作日志的记录过程，从日志的记录过程可以看到数据上链和查询数据时的变化，图 6.14 中“接收到链码事件”表示数据存入区块链账本的操作成功，借助区块链不可篡改的特性可以保证数据信息安全可靠。当对链上数据进行查询时，通过查询函数中的 GetState(args[0])判断链上是否存在与该 args[0]相匹配的数据。若可以找到匹配数据，并返回如图 6.14 所示的“根据 IP 查询信息成功了”、数据的具体信息、当前哈希和历史数据等内容，说明数据没有发生变化且是安全可信的，成功查询到数据；否则返回“根据 IP 查询信息失败”，说明不存在与之对应的数据或数据被篡改。

图 6.14 区块链后台操作日志记录

同时，通过相同任务量下链码调用完成时间的长短来评估本文区块链数据参与计算的时延情况。实验利用脚本封装链码调用，通过参数指定调用次数，并分别记录调用次数为10、20、30、40、50、60、70、80、90 及 100 下的完成时间，测试共进行 5 次并求取平均值得到如图 6.15 所示的对比图。能够看到虽然本文改进的链码调用完成时间整体略高于改进前本机链码调用的情况，但是在调用次数达到 30 次之后，时间增加趋势逐渐减缓，甚至在 80 次后出现时间低于改进前的情况。这是因为本文考虑到非本机链码调用的问题，且只有在有调用需求时才会启动链码容器的创建，前期花费时间较长，但在后期所用链码都已启动完全的情况下，完成时间低于改进前，在达到关键数据区块链存储的基础上，该时延情况是可以接受的。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 20  18  16  14  12  10  8  6  4  2  0  0 |  |  | 改进后 |  |
|  | 改进前 |
|  | 100 |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
| 10 20 30 40 50 60 70 80 90  调用链码次数 | |

图 6.15 链码调用完成时延对比图

* 1. 性能测试与结果验证
     1. **云平台应用调度优化效果验证**

为了评估本文优化改进后的调度器，将本文优化调度器（简称 G-调度器）与文献[10]

改进的调度器（简称 T-调度器）进行对比。根据文献[9]，采用节点均衡效率来衡量调度器改进前后节点负载均衡效果。节点的均衡效率是指该节点上各类资源实际使用率的平均值与该节点上各类资源理想使用率的平均值的比值，定义公式如下所示：

*E*(*Ni* ) =

*Load* (*Ni* ) / 6 *Ideal*(*Ni* ) / 6

(6.1)

其中 *Ideal*(*Ni* ) = 0.75  *Source*( *Ni* ) 节点各类资源理想负载平均值。

实验搭建好 K8s 集群后，部署 T-调度器，在集群中运行由 2 种模型不同 batch\_size组成的共 50 个深度学习类 Pod 应用，然后获取节点的各类资源使用数据；再清除所有 Pod应用，加载扩展优化的 G-调度器后，再次在集群中运行相同的 50 个 Pod 应用，获取节点的各类资源使用数据。根据公式(6.1)分别计算两种调度器下集群中各节点的均衡效率，实

验对比如图 6.16 所示。从图 6.16 可以看出，扩展优化的 G-调度器相比于 T-调度器，负载均衡率平均提升 3.2%，对各个节点的负载均衡都有不同程度的提升，其中对 server006 节点的提升较为明显，这是因为本文改进的调度器充分考虑了各类资源的分配使用。

74 T-调度器 G-调度器

72

70

68

66

64

62

60

58

56

server001 server003 server006

节点

图 6.16 节点负载均衡效率实验对比图

除了各个节点的负载均衡情况，整个集群的负载均衡也是衡量调度器改进效果的重点。本文采用各节点之间的标准差计算各个节点之间负载的差异程度，公式如(6.2)所示：

*SDK* 8*s* =

1 *n*

*~~n~~* (*Load* ( *Ni* ) - *Avg*

*i* =1

)2

(6.2)

其中 *Avg*

1 *n*

= 

*n i*=1

*Load* (*Ni* ) 是集群中各节点负载的平均值。

从公式(6.2)可以看出，标准差越大，集群各个节点之间负载差别越大，集群越不均衡。所以定义 *Balanced* =1- *SDK* 8*s* 来衡量集群的均衡程度。

实验搭建好 K8s 集群后，使用 T-调度器，在集群中依次部署 10 个、20 个、30 个、40 个、50 个 Pod 应用，分别获取不同 Pod 数量下各个节点上各类资源的使用数据；使用扩展优化的 G-调度器，进行同样的实验。根据公式计算不同 Pod 数量下所对应的集群均衡程度 *Balanced* ，实验对比如图 6.17 所示，当集群中所运行 Pod 应用较少时，集群以及各节点的资源充足，T-调度器和本文的 G-调度器对集群所造成的负载均衡效果都比较好；随着不断增多集群内运行 Pod 应用的数量，本文带有扩展优化调度策略的改进调度器对集群产生的负载均衡效果更好，主要原因在于扩展改进的 G-调度器充分考虑到了 Pod 应用和节点资源的匹配度，实现对资源的有效利用。

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1  0.9  0.8  0.7  0.6  0.5  0.4  0.3  0.2  0.1  0 |  | | | | |  | | | | | |
| T-调度器 G-调度器 | | | | | |
|  | | | | |  | | | | | 100 |
|  |  |  |  |  |  |  | | | |
|  | | | | | | |  | | |
|  | | | | | | | | | |
|  | | | | | | | |  |  |
|  | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
| 0 10 20 30 40 50 60 70 80 90  Pod数量（个） | | | | | | | | | |

图 6.17 集群负载程度对比图

* + 1. **云平台容器应用迁移效果验证**

本节对面向 K8s 的容器应用迁移进行性能测试，选取几个不同容器应用进行实验。为验证本文容器应用迁移方法的可行性及有效性，选取 3 个典型的容器应用和 1 个自定义应用进行实验，分别是：由 wordpress:latest（408MB）和 mysql:5.7（250MB）镜像构成的个人博客系统，代表多镜像应用的 WordPress；由 nginx:latest（423MB）镜像直接构成，并在运行后可以访问默认主页，代表多请求应用的 Nginx；由 ubuntu16.04（44.5MB）基础镜像构成的提供 Ubuntu 操作系统环境，代表少请求应用的 Ubuntu；以及由“每隔一秒就按顺序输出一个数字”的代码生成的自定义镜像（235MB）构成的自定义应用。

以可查看输出数字的自定义应用为例，说明迁移测试过程：首先，运行自定义应用后，在部署管理界面选择待迁移的容器应用，点击“迁移”按钮准备迁移；然后，确认待签容器名称，输入目标节点的主机 IP 地址，进行迁移，并记录开始时间、冻结时间、结束时间以及传输的数据量大小；最后，迁移完成并正常运行后，对比所输出数字是否可以正确衔接上。自定义应用在迁移前后所输出的数字情况如图 6.18 所示，可以看到应用在容器停止前输出的数字为 23，迁移完成后应用继续之前的顺序输出数字为 24，说明容器应用的内存数据和文件都进行了迁移。

(a) (b)

图 6.18 应用迁移前后结果输出对比图

实验结果从容器应用进行 5 次相同的迁移实验中计算平均值得到，并从容器应用的文件传输数据量、迁移时间和宕机时间三个方面与当前容器迁移系统（即徐波的 X-Sle 迁移方法）和传统迁移方法进行对比分析。

**1）文件传输数据量分析**

文件传输数据量的对比用来验证本文是否减少了重复镜像的迁移，从而进一步的减少迁移时间。如图 6.19 所示，直接迁移该文件系统上层目录的时候，WordPress、Nginx、Ubuntu 以及自定义应用的文件数据量分别是 658MB、423MB、102MB、235MB；X-Sle迁移时文件数据量分别是 197MB、120.5MB、36.72MB、47.6MB；本文改进迁移方法传输的数据量分别是 186.8MB、116.2MB、35.1MB、45.4MB。测试结果显示，本文改进迁移方法相较 X-Sle 迁移方法减少了 4.45%的文件传输数据量，在一定程度上进一步减少了只读层文件的重复传输。

700

600

500

400

300

200

100

0

WordPress Nginx Ubuntu 自定义应用

容器应用

图 6.19 文件传输数据量对比图

**2）迁移时间分析**

迁移时间是对整个迁移过程效率高低的衡量。从图 6.20 能够看到，由于容器层本身镜像大小的不同，导致迁移时间有所差别，这是因为文件传输在整个迁移过程中占很大一部分，且镜像越大，迁移时间越长。其中直接迁移花费的时间最长，这是因为直接迁移不考虑应用的文件系统中基础镜像复用的问题，所以文件数据的完全迁移要花费大量时间。本文改进迁移时间与 X-Sle 的单容器迁移时间相比略高，但明显低于直接迁移，分析可能的原因在于应用恢复阶段需要花费一定的时间解决冲突问题，这是可以接受的。

西安工业大学专业硕士学位论文

140

120

100

80

60

40

20

0

WordPress Nginx Ubuntu 自定义应用

容器应用

图 6.20 容器应用迁移时间对比图

**3）宕机时间分析**

宕机时间是应用迁移过程中一个重要的衡量指标，“宕机”是指用户完全访问不到应用的情况，宕机时间与容器应用运行所占用的内存大小有关。本文迁移方法的宕机时间=冻结时间+恢复时间。如图 6.21 所示是宕机时间与应用运行所占用内存大小之间的关系，由于容器配置文件中内存限制大小的设置范围为 100MB-700MB，所以对该范围的宕机时间进行测试分析，能够看到冻结时间和结束时间都有不同程度的增加，但整体宕机时间不超过 7s。

|  |  |  |
| --- | --- | --- |
| 6  5  4  3  2  1  0  0 |  | 800 |
| 冻结时间 结束时间 |
|  |
|  |
|  |
|  |
|  |
| 100 200 300 400 500 600 700  内存大小 |

图 6.21 宕机时间与内存对应关系

* 1. 本章小结

本章首先介绍了平台的物理拓扑结构和各个节点的硬件环境配置。然后通过二次开发K8s 实现本文的云平台，并对其进行功能测试与性能验证。最后通过实验验证表明，优化扩展的资源调度策略提升了单个节点中 3.2%的负载均衡效率，集群负载更加稳定；改进的针对集群的容器应用迁移方法减少了容器应用迁移时 4.45%的传输数据量。

1. 结论与展望
   1. 结论

本文依托科研项目，在分析当前国内外 K8s 云平台和可视化管理功能的基础上，结合项目需求设计并实现了适用于战场分析应用的 K8s 云平台，保证云平台中应用的高可用与关键数据的高可信。主要实现了以下内容：

**1）针对战场分析应用的 K8s 云平台调度策略优化**

本文通过分析并运行 K8s 默认的调度策略，发现对战场分析应用来说，默认策略并没有突出 GPU 和内存在各类资源当中的重要程度；并且不同应用对资源的需求和与节点的匹配程度也有不同要求，这导致了资源未被充分利用和资源利用不均衡的问题。所以针对上述问题提出突出 GPU、内存以及节点匹配程度的优化调度策略，提升节点和集群的资源利用情况。

**2）针对 K8s 集群中的 Docker 容器应用迁移优化**

通过研究虚拟机迁移和单个主机的容器迁移，分析容器和 K8s 集群的结构特点，发现容器的分层结构容易导致重复数据的迁移。所以根据容器分层特性，对容器文件系统进行分层对比迁移，读写层直接迁移，只读层以占位符形式迁移镜像索引；同时优化内存迁移中停机迁移的条件。改进迁移方法的实现能够有效减少迁移中传输的数据量。

**3）面向 K8s 的 Fabric 区块链数据可信服务**

本文要求对应用处理后得到的数据进行可信存储，通过分析 K8s 结构与 Fabric 组件启动原理，实现基于 K8s 的 Fabric 可信服务。根据 Fabric 中基础组件与可扩展组件的不同特点，基础组件修改符合 K8s 的配置文件并匹配 Namespace 后直接启动；可扩展组件优化生命周期，划分编译与运行时间，实现链码的跨节点运行，保证每个节点都可以进行数据的可信存储。

**4）平台实现和性能分析**

本文基于 K8s 技术，搭建具有优化调度策略、Docker 容器应用迁移和 Fabric 区块链服务的，并能实现对战场分析应用进行管理的云平台，使用多种类型的容器应用对优化的调度策略、容器应用迁移方法进行效果验证。通过实验验证了本文提出的调度优化策略的优异性和面向 K8s 的容器应用迁移方法的有效性，改进的调度器对单个节点的负载均衡效率提升 3.2%，集群负载程度在应用逐渐增多时更加稳定；改进迁移方法的实现能够减少迁移中 4.45%的传输数据量，且内存范围在 100MB-700MB 之间的应用宕机时间不超过7s。

* 1. 展望

由于自身处理问题和项目经验的不足，加上时间仓促的原因，课题完成的云平台在管理功能和性能指标达上虽然达到了要求，但仍存在一些可继续优化之处。今后可以就以下几个方面继续深入研究：

**1）考虑多节点间调度的通信问题。**本文提出的调度优化策略考虑将战场分析应用调度到更加合适的节点上，仅对将应用调度到一个节点上的过程进行了优化。对于需要多个节点运行的大规模复杂应用，本文没有考虑到多个节点之间的通信需求。在后续工作中，需要研究针对大规模复杂应用分布式运行的节点调度问题，并考虑跨节点运行时通信代价的大小。

**2）考虑复杂应用的资源共享问题。**本文的容器应用迁移涉及到 K8s 中的资源对象

Pod，当应用的复杂程度不断升级时，多个容器或镜像共同协作构成一个 Pod 应用。此时应用迁移需要考虑多个容器共享资源的情况，例如 Pod 内多个容器间的通信和 Pod 的数据挂载模式等。因此能够感知网络的迁移技术和能够考虑 Pod 本地数据卷挂载模式的迁移技术是今后研究的方向。

**3）更加灵活的 Fabric 云化方案。**本文实现了 K8s 上的区块链应用，采用组件分类启动运行的方式。虽然链码使用更加灵活，但是分离链码编译和链码运行增加了启动的时间。所以针对链码动态启动相对费时的问题，今后的研究可以考虑采用划分使用需求高低的形式：使用需求高的链码以传统方式直接运行启动；使用需求低的链码则以动态方式运行启动。划分使用需求程度以及如何分类启动是今后 Fabric 在云上应用的研究方向。

参考文献

1. 牛 青 . 云 计 算 平 台 在 舰 队 协 同 作 战 体 系 建 设 中 的 研 究 [J]. 舰 船 科 学 技术,2016,38(12):160-162.
2. 陆平 , 赵培 , 王 志坤 , 等 . 云计算基础 架构 及关键应用 [M]. 北京 : 机械工业出版社,2016:10-14.
3. 李乔,才智.一种军用云计算平台架构及性能评估[J].信息化研究,2018,44(01):50-54.
4. 龚正,吴治辉,崔秀龙,等.Kubernetes 权威指南：从 Docker 到 Kubernetes 实践全接触:纪念版[M].北京:北京电子工业出版社,2019:6-27.
5. 浙江大学 SEL 实验室.Docker-容器与容器云（2 版）[M].北京:人民邮电出版社，

2016:230-240.

1. 刘敖迪, 杜学绘, 王娜, 等. 区块链技术及其在信息安全领域的研究进展[J]. 软件学报,2018,29(07):2092-2115.
2. Brendan Burns, Brian Grant, David Oppenheimer, et al. Borg, Omega, and Kubernetes[J]. Communications of the ACM,2016,59(5):50-57.
3. Rodriguez M, Buyya R. Container orchestration with cost-efficient autoscaling in cloud computing environments[M]//Handbook of research on multimedia cyber security. IGI Global, 2020:190-213.
4. 张 玉 芳 , 魏 钦 磊 , 赵 膺 . 基 于 负 载 权 值 的 负 载 均 衡 算 法 [J]. 计 算 机 应 用 研究,2012,29(12):4711-4713.
5. 谭莉,陶宏才.一种基于负载均衡的 Kubernetes 调度改进算法[J].成都信息工程大学学

报,2019,34(03):228-231.

1. 杜军.基于 Kubernetes 的云端资源调度器改进[D].浙江大学,2016.
2. 杨鹏飞.基于 Kubernetes 的资源动态调度的研究与实现[D].浙江大学,2017.
3. 徐敏, 李明, 郑建忠, 等. 基于 OpenStack 的 Swift 负载均衡算法[J]. 计算机系统应用,2018,27(01):127-131.
4. 卢 胜 林 , 倪 明 , 张 翰 博 . 基 于 Docker Swarm 集 群 的 调 度 策 略 优 化 [J]. 信 息 技术,2016(07):147-151+155.

[15]Hussein Mohamed K., Mousa Mohamed H., Alqarni Mohamed A.. A placement architecture for a container as a service (CaaS) in a cloud environment[J]. Journal of Cloud Computing,2019,8(1):1-15.

[16]Xu Zhigang, Xiao Limin, Zhan Weidian, et al. An VM Scheduling Strategy Based on Hierarchy and Load for OpenStack[C] //2016 7th International Conference on Cloud Computing and Big Data (CCBD). 2016,pp.58-63,doi:10.1109/CCBD.2016.022.

[17] 宋霖.基于 Kubernetes 的资源调度与监控系统的设计与实现[D].北京邮电大学,2019. [18]Zhang Dong, Yan Bing-Heng, Zhen Feng, et al. Container oriented job scheduling using

linear programming model[C] //International Conference on Information Management(ICIM). IEEE, 2017:174-180.

[19]Liu Bo, Li Pengfei, Lin Weiwei, et al. A new container scheduling algorithm based on multi-objective optimization[J]. Soft Computing,2018,22(23): 7741-7752.

[20]Liu Zhaobin, Qu W., Yan Tao, et al. Hierarchical Copy Algorithm for Xen Live Migration[C] //International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery,IEEE,2010: 361-364, doi:10.1109/CyberC.2010.71.

[21]Feng Xiujie, Tang Jianxiong, Luo Xuan, et al. A performance study of live VM migration technologies: VMotion vs XenMotion[J]. Shanghai Jiao Tong Univ. (China), 2012,8310-8311.

* 1. 彭丽苹, 吕晓丹, 蒋朝惠 , 等. 基于 Docker 的云资源弹性调度策略 [J]. 计算机应用,2018,38(02):557-562.
  2. 房锦章, 武延军. 基于 Docker 技术的 GUI 应用的在线迁移研究[J].计算机系统应用,2016,25(10):246-251.
  3. 毛 祺 , 卢 胜 林 . 基 于 Docker Swarm 集 群 的 容 器 迁 移 策 略 的 实 现 [J]. 信 息 技术,2016(09):156-160.
  4. 严康.CntrMig：高性能容器在线迁移系统[D].华中科技大学,2019.
  5. 聂虎卿.云环境下容器动态迁移技术研究[D].南京邮电大学,2018.

[27]Nadgowda S, Suneja S, Bila N, et al. Voyager: Complete Container State Migration[C]//

2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2017:2137-2142.

[28]Ma Lele, Yi Shanhe, Li Qun. Efficient service handoff across edge servers via docker container migration[P]. Edge Computing,2017:1-13.

[29]Zhang Liang, Litton J, Cangialosi F, et al. Picocenter: Supporting long-lived, mostly-idle applications in cloud environments[C] //Proceedings of the Eleventh European Conference on Computer Systems. 2016:1-16.

[30]Yu Chenying, Fei Huan. Live Migration of Docker Containers through Logging and Replay[C] //Advances in Computer Science Research, International Conference on Mechatronics and Industrial Informatics. 2015:23-28.

[31]Barbalace A, Karaoui M L, Wang W, et al. Edge computing: The case for heterogeneous-isa container migration[C] //Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. 2020: 73-87.

[32]Rattihalli G., Govindaraju M., Lu Hui., et al. Exploring Potential for Non-Disruptive

Vertical Auto Scaling and Resource Estimation in Kubernetes[C] //2019 IEEE 12th International Conference on Cloud Computing (CLOUD), IEEE,2019:33-40.

[33]Truyen E, Lagaisse B, Joosen W, et al. Flexible Migration in Blue-Green Deployments within a Fixed Cost[C] //Proceedings of the 2020 6th International Workshop on Container Technologies and Container Clouds. 2020:13-18.

* + 1. 朱 昱 锦 , 姚 建 国 , 管 海 兵 . 区 块 链 即 服 务 : 下 一 个 云 服 务 前 沿 [J]. 软 件 学报,2020,31(01):1-19.DOI:10.13328/j.cnki.jos.005891.
    2. 刘宏宇,梁秀波,吴俊涵.基于 Kubernetes 的 Fabric 链码管理及高可用技术[J].计算机应用,2021,41(04):956-962.
    3. 刘 楠 , 刘 露 . 区 块 链 与 云 计 算 融 合 发 展 BaaS 成 大 势 所 趋 [J]. 通 信 世

界,2017(17):61-62.DOI:10.13571/j.cnki.cww.2017.17.037.

* + 1. 才丽.面向 BaaS 平台的资源调度算法研究与实现[D].浙江大学,2018.

[38]Zheng Weilin, Zheng Zibin, Chen Xiangping, et al. NutBaaS: A Blockchain-as-a-Service Platform.[J]. IEEE Access,2019,PP(99):1-1.

[39]Asheralieva Alia, Niyato Dusit. Distributed Dynamic Resource Management and Pricing in the IoT Systems With Blockchain-as-a-Service and UAV-Enabled Mobile Edge Computing[J]. IEEE Internet of Things Journal,2020,7(3):.

[40]Singh J., Michels J.D. Blockchain as a Service (BaaS): Providers and Trust[C] //2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 2018:67-74.

[41]Ma Zhaofeng, Zhao Weizhe, Luo Shoushan, et al. TrustedBaaS: Blockchain-Enabled Distributed and Higher-Level Trusted Platform[J]. Computer Networks,2020,183:.

[42] 石山. 军事信息云的网络架构设计及作战指挥辅助决策研究[D]. 西安电子科技大学,2012.

[43]Naik N. Building a virtual system of systems using docker swarm in multiple clouds[C]

//2016 IEEE international symposium on systems engineering (ISSE). IEEE, 2016:1-3.

[44] 杨茂, 陈莉君. 基于 Kubernetes 的容器自动伸缩技术的研究[J]. 计算机与数字工程,2019,47(09):2217-2220+2232.

[45]Shah J., Dubaria D. Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform[C] //2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), IEEE,2019:0184-0189.

[46]Brewer E A. Kubernetes and the path to cloud native[C]//Proceedings of the sixth ACM symposium on cloud computing. 2015:167-167.

[47]Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture[J]. IEEE Software,2016,33(3): 42-52.

* + - 1. 黎跃春,韩小东,付金亮. Hyperledger Fabric 菜鸟进阶攻略[M].北京:机械工业出版社,2019:21-27.
      2. 李军让,邓元豪,冉计全,等. 基于区块链的军事数据标准管理研究[C] //第八届中国指挥控制大会论文集.,2020:793-798.DOI:10.26914/c.cnkihy.2020.014717.

[50]Thakkar P., Nathan S., Viswanathan B. Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform[C] //2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE,2018:264-276.

[51]Kubernetes. Scheduling,Preemption and Eviction[EB/OL].2021, https://kubernetes.io/zh/docs/concepts/scheduling-eviction/.

[52] 王一凡,王中华,李亚晖.一种面向战术云环境的轻量计算迁移方法[J].信息技术与信息化,2021(03):86-88.

[53]Docker Inc. Docker Images and Containers[EB/OL].2020, https://docs.docker.com/storage/storagedriver/imagesandcontainers/#images-and-layers.

[54] 刘 伟 . 基 于 动 态 预 取 策 略 的 容 器 写 时 复 制 机 制 优 化 研 究 [D]. 华 中 科 技 大学,2019.DOI:10.27157/d.cnki.ghzku.2019.001386.

[55]Microsoft Azure[EB/OL]. 2020, https://www.microsoft.com/azure/windowsazure.mspx.

致 谢

寒冬远去，春的气息已遍布校园，三年的研究生生活接近尾声。回首过往，许多事情我都历历在目，不论是教师节与团队大家庭初相识紧张的我，还是参加组会时懵懂的我，以及最后可以独立思考努力完成任务的我，这一路走来都是我成长的痕迹。在这三年的时间里，不论是外出四天从零学习区块链技术，还是埋头加深对基础理论与技术的研究，以及着手尝试搭建简单的云平台，这些反复摸索的过程，不仅让我获得了多方面的知识，更让我懂得了控制自我情绪、排遣内在压力的重要性；懂得了与人相处、互帮互助的重要性；这些奋斗的时光让我变得更加坚强勇敢，更加自信的面对过去和未来的自己。

三年的成长离不开帮助和支持我的人。

首先我想感谢我的导师高武奇副教授。高老师虽然工作繁忙，但是对学生的关心与帮助却从不缺席，在项目任务上不断鼓励提供解决问题的新思路，在课题研究上为我指明正确的方向，在技术学习上提供宝贵的机会并予以支持，在问题的讨论上以平易近人的方式使我豁然开朗，老师在对待问题时的思考与做法，使我收获颇多。

同时要感谢 211 实验室团队的所有老师，分别是王鹏老师、李晓艳老师、吕志刚老师、许韫韬老师、杨永侠老师以及董绵绵老师。他们与学生朝夕相处，一同攻克难题，一起打球放松，老师们在项目上的指导、讲座上的分享甚至是日常的交谈都带给我知识上的进步和思想上的变化。更要感谢实验室的规范管理，让我养成良好的习惯。

其次要感谢吴娇学姐、贺咪咪学姐、黄闪闪学姐对我学习研究上的答疑解惑；感谢同窗好友符惠桐、郭嘉和郭翔宇同学，大家积极分享信息资源，学习上互帮互助，项目上共同进步，为干饭小分队带来许多欢乐；感谢同门的穆武斌学弟，在完成项目查找资源，实际操作上提供了重要的帮助；感谢 433 宿舍我可爱的舍友们，大家虽然专业有所不同，但在彼此遇到压力时给予了最温暖的陪伴，闲暇时的锻炼休闲娱乐，都是值得我深藏的记忆。

最后想感谢我的父母，他们总是在我做出选择时给予默默的支持和最大的帮助，是我最坚强的后盾。愿未来的自己越努力越幸运，越幸运越努力。

学位论文独创性与知识产权声明

秉承学校严谨的学风与优良的科学道德，本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，学位论文中不包含其他人已经发表或撰写过的成果，不包含本人已申请学位或他人已申请学位或其他用途使用过的成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了致谢。

学位论文与资料若有不实之处，本人承担一切相关责任。

本人完全了解西安工业大学有关保护知识产权的规定，即：研究生在校攻读学位期间学位论文工作的知识产权属于西安工业大学。本人保证毕业离校后，使用学位论文工作成果或用学位论文工作成果发表论文时署名单位仍然为西安工业大学。学校有权保留送（提）交的学位论文，并对学位论文进行二次文献加工供其他读者查阅和借阅*；学* 校可以在网络上公布学位论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存学位论文。（保密的学位论文在解密后应遵守此规定）