本科毕业论文（论文）

材料大数据分布式处理系统设计

学　　院：大数据与信息工程

专　　业：信息管理与信息系统

班　　级：信管 151

学　　号：1500890286

学生姓名：张可颖

指导教师：高廷红

2019 年 6 月 15 日

贵州大学本科毕业论文（设计）

诚信责任书

本人郑重声明：本人所呈交的毕业论文（设计），是在导师的指导下独立进行研究所完成。毕业论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。

特此声明。

论文（设计）作者签名：

日　　期：

目 录

材料大数据分布式处理系统设计

摘 要

近年来，云计算与大数据飞速发展，应用程序产生的数据爆炸式增长，一方面促进了云计算相关技术的发展如 Kubernetes、Storm 和 Hadoop 等，一方面技术带来的变革也影响到了各行各业，如材料大数据科学计算，通过分布式的数据处理能够极大提高计算效率。但于此同时，数据中心在方便了用户的同时也产生了巨大的不必要的成本，如当业务高峰消失，多余的资源不能及时自动释放，持续产生电力运营成本。云平台资源调度优化问题引起了学者们的广泛关注，如何优化云平台资源利用率也是学术界和工业界研究的热点。

本文首先介绍了云平台自适应系统的研究背景、意义、国内外研究现状以及相关理论。然后搭建基于 Kubernetes 的容器云平台并部署相关监控，根据材料二维坐标数据计算半径内点的个数的案例以 Map Reduce 思想设计分布式算法并在 Kubernetes 平台上编程实现，证明了分布式算法能显著提高材料科学计算的效率。

然后针对现有的自适应系统研究很少考虑 Qos 服务质量方面的不足，提出了一种基于 Qos 和成本权衡的自动扩展方案。该策略利用移动平均自动预测材料工作任务请求数目，并通过成本延迟权衡得到最佳云资源工作单位需求。由于虚拟机或容器工作单位启动需要一定时间，基于预测的需求自动提前对资源实现伸缩，能有效提高集群资源调度效率。

在云资源自动扩展方案的基础上提出一种基于 Q 学习 e 贪心算法的自适应性云资源最大利用率优化算法，选择虚拟机或容器工作单位的最优调度服务器，将无负载的服务器设置为休眠，从而提高集群资源利用率。

最后进行仿真调度实验，实验表明所提出的方案实现了资源自动扩展在满足成本延迟权衡和低 SLA 违约的条件下，能同时有效提高了服务器集群资源利用率。

关键词 材料数据分布式处理，云计算，自动伸缩，资源配置，自适应调度

Material big data distributed processing system design

ABSTRACT

In recent years, the rapid development of cloud computing and big data applications to produce the explosive growth of data, on the one hand, promoted the growth of cloud computing technology, such as Kubernetes, Storm, and Hadoop, etc., on the one hand, technology changes also impact to all walks of life, such as material scientific computing, big data through distributed data processing can greatly improve the computational efficiency. However, at the same time, the data center not only facilitates users, but also generates huge unnecessary costs. For example, when the business peak disappears, the redundant resources cannot be released automatically in a timely manner, thus continuously generating operating costs of electricity. The problem of resource scheduling optimization of cloud platform has attracted extensive attention of scholars, and how to optimize the resource utilization of cloud platform is also a hot topic in academia and industry.

This paper first introduces the research background, significance, domestic and foreign research status and relevant theories of cloud platform adaptive system. Then, a kubernet-based container cloud platform is built and relevant monitoring is deployed. A distributed algorithm is designed and programmed on the Kubernetes platform to calculate the number of points within the radius based on two-dimensional coordinate data of materials. It is proved that the distributed algorithm can significantly improve the efficiency of material scientific computing.

Then aiming at the deficiency of Qos service quality which is seldom considered in the existing adaptive system research, an automatic extension scheme based on Qos and cost tradeoff is proposed. This strategy USES mobile average to automatically predict the number of material work task requests, and obtains the best cloud resource work unit demand through cost delay tradeoff. Since it takes a certain time for virtual machine or container work unit to start up, the resources can be

automatically extended in advance based on the predicted demand, which can effectively improve the scheduling efficiency of cluster resources.

On the basis of the cloud resource automatic expansion scheme, an adaptive algorithm for optimizing the maximum utilization rate of cloud resources based on Q learning e-greedy algorithm was proposed. The optimal scheduling server of virtual machine or container unit was selected, and the unloaded server was set to sleep, so as to improve the utilization rate of cluster resources.

Finally, the simulation scheduling experiment is carried out, and the experiment shows that the proposed scheme can realize the automatic extension of resources and effectively improve the resource utilization of server cluster under the condition of meeting the cost delay tradeoff and low SLA default.

Keywords Distributed Processing of Material Data, Cloud Computing, Auto Scaling; Resource Allocation; Distributed System; Adapting System

第一章 绪 论

1.1 研究背景和意义

从小型企业到大型企业，现代 IT 公司越来越多地利用云计算来提高利润并降低成本。在所有软件即服务（Saas），平台即服务（Paas），和基础设施即服务（Iaas）级别中，云的一个显著特征就是弹性。在任务具有比较稳定和周期性的可预测场景中，配置的资源可以由专家预先大致指定。但是，对于其他情况，如意外的工作负载变化，弹性只能通过运行时自动缩放策略来实现[1]。在按需云环境中，材料科学计算程序承载的容器或虚拟机提供机构可能向上或向下扩展虚拟化资源，以实现资源高效利用的目的，然而按需的真正弹性目前尚未实现[2]。资源提供过度或供应不足的问题仍然存在与云服务的使用中[3]。在虚拟机和容器工作单位级别上，如果服务提供机构操作正确，将会降低成本和提高服务质量（如延迟或响应时间）[4]。在 2008 年 4 月，图像处理网络程序 Animoto 在短短三天经历了 50 个实例到 4000 个实例（Amazon EC2）的需求跳跃；在峰值之后，流量急剧下降到远低于峰值的正常水平。因此 Animoto 仅在高峰时段需求 4000 个虚拟机实例，当峰值消失，未使用的资源被释放。所以良好的自适应弹性调度策略能降低科研服务提供机构的成本，同时提高用户服务质量。

由于材料科研计算平台在全世界拥有大量的潜在用户，可以预测流量是高度动态的，因此材料应用程序可能会受到高度突发的请求并且变得不堪重负。当服务器过载时，用户会感受到更长的延迟甚至丢失服务；与此相对，服务器处于空闲状态或非高峰期，却没有释放资源，仍然在消耗电力，给科研服务机构增加成本。为了应对这些挑战，在本文中，提出一种新的方案，用于实现云资源的虚拟机(VM)或容器工作单位（POD）级别的自动调度，并为材料计算应用程序服务机构找到最佳的成本-延迟权衡。提出的方案致力于为应用程序分配足够的资源，以最大限度地减少资源浪费，同时避免服务级协议（SLA）违约，而无需人为干预。

1.2 国内外研究现状

云任务调度是 NP-hard 问题，不论是工业界还是学术界都无法给出一个十全十美的

银弹方案，现有的研究往往是针对某一个或几个优化方向，结合适合的算法，进行优化。例如工业界中的 Max-Min 算法虽然实现简单，执行快速，但是可扩展性差；Min-Min 算法同样实现简单，执行快速，但是负载均衡性能差[2]。在学术界，蚁群调度算法虽然在效率和鲁棒性上表现较强，但初期求解速度慢，搜索时间比较长[3]；遗传调度算法利于求解全局最优，但是效率低，稳定性差且容易过早收敛[4]；粒子群调度算法收敛快且精度高，但是易于陷入局部最优[5]。

工业界应用广泛的任务调度有先来先服务，最小执行时间，最小完成时间，Max-Min，Min-Min，和加权平均打分等。然而这些算法往往存在各自缺陷，在学术界，蚁群算法，粒子群算法，元启发算法，遗传算法等也被应用到云调度中[4]。

在学术界，随着强化学习蓬勃发展，许多强化学习和启发式算法也应用于云调度之中。在文献[5]中，针对数据流处理，设计了几种分布式自适应策略。第一种是传统的基于阈值的触发式策略。第二种是基于强化学习无模型 Q 学习策略。第三种在第二种的基础上，采用全备份模式，不再需要无模型 Q 学习中的强制探索机制。在文献[6]中，利用进化算法解决云资源调度中多目标优化问题，同时考虑多个 Qos 属性（如等待时间，吞吐量和成本），具体提出了四种播种策略，并用开源数据集证实了策略的有效性。在文献[7]中，提出了一套结合遗传算法的优化云资源配置系统，通过使用排队论理论和统计技术，建模并计算 SLA 违约度量，该度量被定义为优化算法中的适应度函数。该优化策略可以指导云用户购买适当的资源以最小的资源成本满足其各种工作负载要求。

目前学术界的研究中存在的问题和面临的挑战做了主要如下[3]：

（1）Qos 建模在自适应弹性调度系统中经常被忽略；

（2）大多数研究仅仅尝试在 Iaas 级别扩展硬件资源，未来成熟的自适应弹性调度系统应该考虑软件和硬件的联动；

（3）大所属研究的研究的控制粒度都是固定的（即服务或应用级别，虚拟机或容器级别，云），未来的自适应弹性调度系统应该考虑更灵活的粒度混合；

（4）缺少真实的案例和情景，大多数研究属于理论研究范畴。

同时对现有的自适应系统根据感知环境类型进行了如下分类：

（1）基于刺激的感知；（2）基于交互的感知；（3）基于周期性的时间感知；（4）基于目标和约束的感知；（5）元自我感知。

1.3 本文研究内容

本文首先对云平台自适应弹性调度策略的研究背景意义和国内外研究现状进行了归纳和总结，然后对 Kubernetes 容器云平台，Prometheus，Grafana，Docker 容器，Map Reduce，M/M/N 排队论理论，强化学习 Q 学习 e 贪心算法分别进行了介绍。然后，针对 Qos 建模在自适应弹性调度系统建模中经常被忽略的不足，提出了一种基于移动平均和排队论的云资源自动扩展方案。该策略自动预测工作任务请求数目，并通过成本延迟权衡得到最佳云资源工作单位需求。由于虚拟机或容器工作单位启动需要一定时间，基于预测的需求自动提前对资源实现伸缩，能有效提高集群资源调度效率。提出一种基于 Q 学习的自适应性云资源最大利用率优化算法，在云资源自动扩展方案的前提下，选择最优调度服务器，将无负载的服务器设置为休眠，从而提高集群资源利用率。进行仿真调度实验，实验表明所提出的方案实现了资源自动扩展在满足成本延迟权衡和低 SLA 违约，同时能有效提高了服务器集群资源利用率。本文的工作总结如下：

（1）大量阅读相关的国内外文献，对现有的云平台自适应弹性调度策略进行比较和总结，取长补短，并利用 Google 开源的容器云平台 Kubernetes，监控和可视化软件 Prometheus、Grafana 和容器虚拟化技术 Docker 组合搭建一个材料大数据分布式处理系统，并在此基础上研究材料云平台的自适应弹性调度优化问题。

（2）根据 Map Reduce 思想设计分布式算法计算材料二维坐标半径 R 内点的个数，并在所构建的材料数据分布式处理系统中实现成功，并与单机运行进行效率对比，证明材料大数据分布式处理系统的可行性，能显著提高材料科学计算的计算效率。

（3）对云平台自适应弹性调度系统原理进行深入的研究，分析各种现有研究的优缺点，然后针对针对 Qos 建模在自适应弹性调度系统建模中经常被忽略的不足，提出了一种云资源自动扩展方案。该策略能够在没有人为的干预下自动预测工作任务请求数目，并通过成本延迟权衡得到最佳云资源工作单位需求。由于虚拟机或容器工作单位启动需要一定时间，基于预测的需求自动提前对资源实现伸缩，能有效提高集群资源调度效率。

（4）基于云资源自动扩展策略，提出一种基于 Q 学习 e 贪心算法的自适应性云资源最大利用率优化策略。云资源自动扩展策略确定虚拟机或容器工作单位扩展的数目，Q 学习 e 贪心算法选择最优调度服务器，将无负载的服务器设置为休眠，从而提高集群

资源利用率。进行仿真调度实验，实验表明所提出的方案实现了资源自动扩展在满足成本延迟权衡和低 SLA 违约，同时能有效提高了服务器集群资源利用率。

1.4 本文组织架构

本文一共分为五个章节来阐述所研究的内容，论文的结构如下：

第一章 绪论：首先阐述了云资源弹性调度相关问题的研究背景和意义，并对其国内外研究现状进行了概述。接着介绍了本文的主要研究内容。最后介绍了本文的组织结构。

第二章 相关理论知识：对 Kubernetes 容器云平台管理系统、Map Reduce、M/M/N 排队论、强化学习 Q 学习 e 贪心算法进行了简单介绍，并给出了材料大数据分布式处理系统架构图。

第三章 材料大数据分布式处理策略研究：基于 Map Reduce 思想设计分布式算法计算材料二维坐标半径 R 内点的个数，并在所构建的材料数据分布式处理系统中实现成功，并与单机运行进行效率对比，证明材料大数据分布式处理系统的可行性，能显著提高材料科学计算的计算效率。

第四章 材料应用自适应资源弹性调度策略：对云平台自适应弹性调度系统原理进行深入的研究，分析各种现有研究的优缺点，然后针对 Qos 建模在自适应弹性调度系统建模中经常被忽略的不足，提出了一种云资源自动扩展方案。该策略能够在没有人为的干预下自动预测工作任务请求数目，并通过成本延迟权衡得到最佳云资源工作单位需求。由于虚拟机或容器工作单位启动需要一定时间，基于预测的需求自动提前对资源实现伸缩，能有效提高集群资源调度效率。

第五章 总结：对本文工作进行总结。

第二章 云平台与弹性调度技术研究

云计算的快速发展带来数据量的爆炸式增长，也推动了大数据相关技术的发展。从2000年的初步发展，到现在的日益成熟，技术在给人们带来方便的同时，也带来了一些挑战。云平台作为云计算与大数据处理的基础，其资源调度优化问题也收到了国内外的广泛关注，大量学者对此做出了贡献[8-12]。

本章首先介绍Kubernetes容器云平台管理系统及其相关技术、Map Reduce、M/M/N排队论、强化学习Q学习e贪心算法等理论，然后，进一步明确指出本文的研究内容。

2.1 云计算与Kubernetes

2.1.1 云计算概述

2006年，谷歌首次正式提出了云计算的概念。维基百科将云计算定义为基于互联网的计算方法，其中可以根据需要向各种计算机终端和其他设备提供共享的硬件和软件资源和信息。云计算自提出以来，就发展迅猛，给世界带来了一场席卷几乎所用传统行业的技术变革，给人们的生活带来了极大的便捷。国内外的IT巨头Google，Amazon，Microsoft，IBM，阿里巴巴，华为，腾讯，百度争先建立了各自的公有云，私有云和混合云云平台。按服务模式分类，云平台提供的服务由底至上可以分为基础设施即服务（Infastructure as a Service,IaaS），平台即服务（Platform as a Service,IaaS），软件及服务（Software as a Service,IaaS）[13]。如今主流的开源云平台操作系统有Openstack和Kubernetes。前者主要提供IaaS层服务，后者主要提供Paas层服务。

2.1.2 Kubernetes概述

2014年，Google正式成立Kubernetes项目，Kubernetes是Google Borg的开源版本，所以一出世便以迅雷不及掩耳之势打败Docker Swarm和同期竞争对手Apache Mesos，在2017年一统容器编排的市场。

Kubernetes的架构和其原型项目Borg十分相似，都为Master-Slave架构，对应角色分别为控制节点和计算节点，如下图所示：

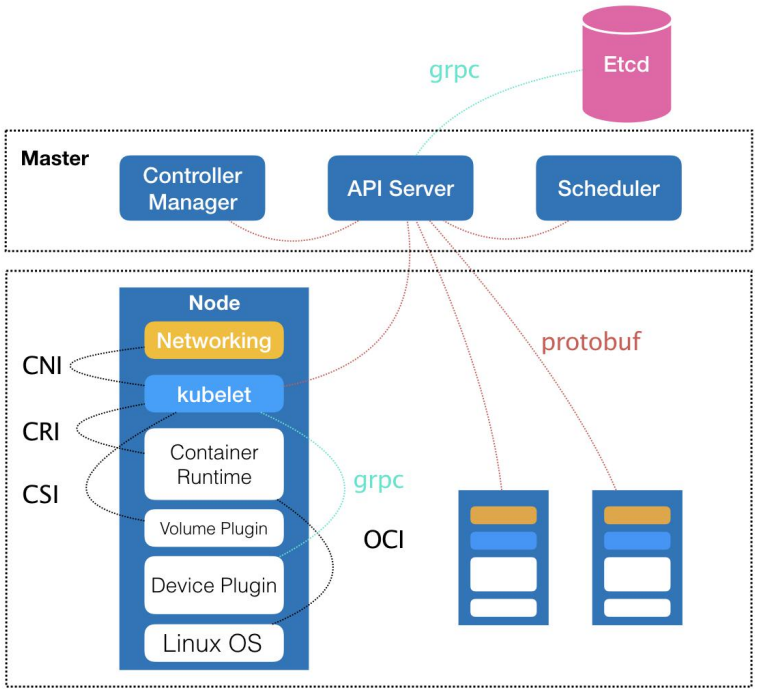

图 2.1 Kubernetes 系统架构图

其中，控制节点即 Master 节点由三个紧密协作的独立组件组合而成，它们分别是负责 API 服务的 kube-apiserver、负责调度的 kube-schedule，以及负责容器编排的 kube-controller-manager。整个集群的持久化数据，则由 kube-apiserver 处理后保存在 Etcd 中。

在计算节点上最核心的部分是 kubelet 组件。kubelet 主要负责与运行时的容器交互。这个交互依赖的是一个乘坐 CRI 的远程调用接口，这个接口定义了容器运行时的各项核心操作，比如:启动一个容器需要的所有参数。所以 Kubernetes 项目并不关心容器运行使用什么技术实现，只要运行的容器镜像是标准的，它就可以通过实现 CRI 接入到 Kubernetes 项目当中。

2.2 Map Reduce 概述

Map Reduce 是一种编程模型，是并行的大规模数据计算框架，基础数据结构是键值对（Key-Value），核心思想是分治算法[14]。Map Reduce 编程模型并不是 Google 原创，也不是 Hadoop 原创，但是 Google 和 Hadoop 创造性地将 Map Redece 编程模型应用到大数据计算上，极大的促进了大数据处理的发展。

Map Reduce 是一种简单又强大的编程模型。简单在于其编程模型只包含 Map 和 Reduce 两个过程，Map 的计算输入是一组键值对，经过 Map 计算后输出一组键值对；然

后将相同键合并，形成键值对集合，再将这个键值对集合输入 Reduce，经过计算输出零个或多个键值对。Map Reduce 十分强大，不管是关系代数运算（SQL 计算），还是矩阵运算（图计算），大数据领域几乎所有计算需求都可以通过 Map Reduce 编程实现。

2.3 M/M/N 排队论理论

2.3.1 排队论基本构成与指标

（1）输入过程：输入过程是根据什么样的规则描述客户如何到达排队系统。包括①顾客的数目：顾客总数是有限的还是无限的。②抵达的类型：顾客是单个抵达还是分批抵达。③顾客之间到达的间隔：通常设它们彼此之间独立同分布，一些以相等的间隔抵达，一些服从负指数分布，一些服从 k 阶 Erlang 分布。

（2）排队过程：排队规则是指客户按指定的顺序接受服务。通常，有等待系统，损失系统，混合系统和封闭系统。当客户到达时所有服务台都不闲置，客户排队等待直到他们离开服务，这称为等待系统。在等候系统中，可以使用先到先得的服务，例如排队买票;还有先到先得的服务，例如天气预报;还有随机服务，如电话服务;并且还有优先服务，例如重症患者，可以先治疗。当客户到达时，所有服务台都没有闲置，然后客户立即离开而不等待，称为损失系统。排队等候的客户数量有限，称为混合动力系统。当客户对象和服务对象相同并且已修复时，它们将被关闭。例如，几个维修工人修理了某个工厂的机器并关闭了。

（3）服务机构：服务机构的指标设置主要包括服务台的数量和服务时间服从的分布。常见的有定长分布、负指数分布、几何分布等。

2.3.2 排队论系统评价指标

（1）队长与等待队长：队长 L_s 是指系统中的平均客户数（包括正在接受服务的顾客）。等待队长 L_q 指系统中正在等待的客户的数量。总队长=等待队长+正在服务的客户数。

（2）等待时间：等待时间由客户的平均停留时间 W_s 和平均等待时间 W_q 两部分组成。客户的平均停留时间是客户进入系统的时间与离开系统的时间之间的时间，包括等待时间和接收服务的时间。客户的平均等待时间是客户进入系统接收服务的时间。

（3）忙碌时期：从客户到达空闲系统，服务立即启动，直到再次空闲。该时段是

系统持续忙碌的时段，称为系统的忙碌时段。它反映了系统中服务组织的强弱，是衡量服务系统利用效率的指标，即服务强度=忙时/总服务时间=（1 - 空闲时间）/总服务时间空闲时段和忙碌时段对应的系统空闲时间，即系统保持空闲的时间长度。

2.3.3 M/M/N 等待制模型[15]

该模型中顾客到达规律服从参数为λ的 Poisson 分布，在$[0,t]$，t时间内到达的顾客数$X(t)$服从的分布为：

$$P\{X(t)=k\}=\frac{(\lambda t)^k\cdot e^{-\lambda t}}{k!} \tag{2-1}$$

其单位时间到达的顾客平均数为λ，$[0,t]$时间内到达的顾客平均数为λt。

顾客接受服务的时间服从负指数分布，单位时间服务的顾客平均数为μ，服务时间的分布为：

$$f(t)=\begin{cases}\mu e^{-\mu t} & t>0\\ 0\end{cases} \tag{2-2}$$

每个顾客接受服务的平均时间为$\frac{1}{\mu}$。

当有多个服务台$N>1$时，当系统在稳定状态下有i个顾客的概率为$p_i(i=0,1,2...)$。

p_0表示系统空闲的概率。因此：

$$\sum_{i=0}^{\infty}p_i=1 \quad p_i\geq 0, i=1,2,\cdots,K \tag{2-3}$$

平衡方程为;

$$\begin{cases}\mu P_1=\lambda P_0\\ (k+1)\mu P_{k+1}+\lambda P_{k-1}=(\lambda+k\mu)P_k & 1\leq k\leq s-1\\ s\mu P_{k+1}+\lambda P_{k-1}=(\lambda+s\mu)P_k & k\geq s\end{cases} \tag{2-4}$$

系统中有s个服务台，系统服务能力为$s\mu$，服务强度为$\rho=\frac{\lambda}{s\mu}$。

系统中的平均队长为：

$$L_s=s\rho+\frac{(s\rho)^s\rho}{s!(1-\rho)^2}\cdot p_0 \tag{2-5}$$

其中，$p_0 = \left[\sum_{k=0}^{s-1}\frac{(s\rho)^k}{k!} + \frac{(s\rho)^s}{s!(1-\rho)}\right]^{-1}$ 表示所有服务台都空闲的概率。

系统中顾客逗留时间为：

$$W_s = \frac{L_s}{\lambda} \tag{2-6}$$

系统中顾客的平均等待时间为：

$$W_q = W_s - \frac{1}{\mu} \tag{2-7}$$

系统中顾客的平均等待队长为：

$$L_q = \lambda W_q \tag{2-8}$$

2.4 强化学习 Q 学习 e 贪心算法

2.4.1 马尔可夫决策过程(MDP)

大多数强化学习算法是以 MDP 为基础,MDP 是以马尔可夫随机过程为理论基础[19],定义为五元组(S, A, p, c, γ)。S 为有限的状态集合，$A(s)$ 是有限的动作集合，$p(s'|s,a)$ 是状态与状态之间转移的概率。在马尔可夫决策中,当前状态 s 转移到下一状态 s' 的的概率和获得的即时汇报只与当前状态 s 和在此状态下选择的动作 a 有关。$a \in A(s), c(s,a)$ 是在状态 s 下执行动作 a 的成本。$\gamma \in [0,1]$ 是折扣因子。

2.4.2 Q 学习 e 贪心算法

强化学习是用来解决在状态转移函数和奖赏函数都未知的情况下，智能体怎样通过学习得到最优行为策略的问题,而 Q 学习是强化学习的经典算法。Q 学习是在时间差分的基础上利用状态-动作函数 $Q(s,a)$ 进行值的函数迭代的一种无模型强化学习算法。Q 学习的迭代公式如下：

$$Q(s_t, a_t) = (1-a)Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t)] \tag{2-9}$$

Q 学习首先会建立一张 Q_table，Agent 通过不断与环境交互，得到反馈，对智能体的状态-动作矩阵形成奖赏值，通过不断地迭代进行蟹盖，会使得选择正奖赏动作的概率持续增加，而与之相对的，选择负奖赏的动作的概率会一直降低。最后，随着迭代次数不断增加，会使智能体动作趋于最优动作集合。

探索和利用的平衡问题是强化学习的重要研究方向[16],e 贪心探索算法对贪心算法的一种改进,与贪心算法相比,e 贪心探索算法能有效跳出局部最优。当 Agent 在与环境进行交互进行动作策略选择时，会以概率 e 进行随机选择，即为探索，以此来保证遍历所有的状态空间，而以 1-e 概率进行利用，选择最优奖赏。其他常用的探索平衡策略又波尔曼分布和启发式动作选择等[17]。

2.5 本章小结

本章主要介绍了云平台和大数据处理所涉及到的相关技术，为第三章材料大数据分布式处理策略研究奠定基础，同时还介绍了排队论和强化学习相关理论，为第四章材料应用自适应资源弹性调度策略的研究奠定基础。

第三章 材料大数据分布式处理策略研究

3.1 问题描述

Kubernetes 容器云平台具有轻量级，高可扩展性和高自修复等特点，可以为材料大数据处理平台提供以容器为中心的基础架构，包括计算，存储，网络，监控，认证等。在 Kubernetes 的 Pods 中可以构建材大数据料分布式处理 PaaS 平台和 SaaS 应用来提高材料科学计算的效率，而目前为止，少有学者研究容器云平台在材料大数据处理方面的应用。

3.2 材料大数据分布式处理系统架构

Master node 作为控制节点,对系统进行调度管理,Worker node 作为真正的工作节点，运行编写好的 Map Reduce 材料应用程序。监控软件 Prometheus 和 Grafana 以及材料应用程序皆以 Pod 的形式运行在 Kubernetes 容器云平台上，进行统一管理。

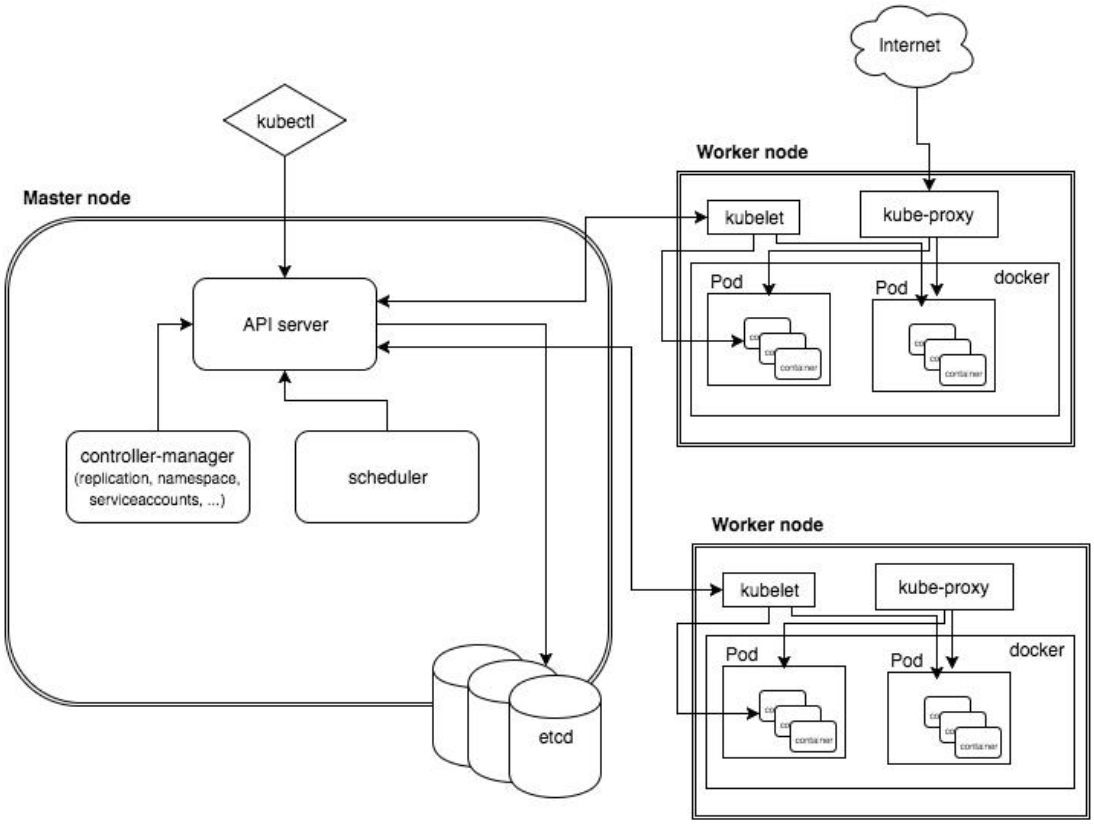

图 3.1 Kubernetes 材料大数据分布式系统架构图

3.3 分布式处理流程

第一步:将数据进行 Map 过程，进行数据分块，本次实验将数据区块设置为 16 个核心区块，计算每个点在核心区块 R 半径内的所有点，只需要计算区块边界值+R 的区间范围，如下图所示：

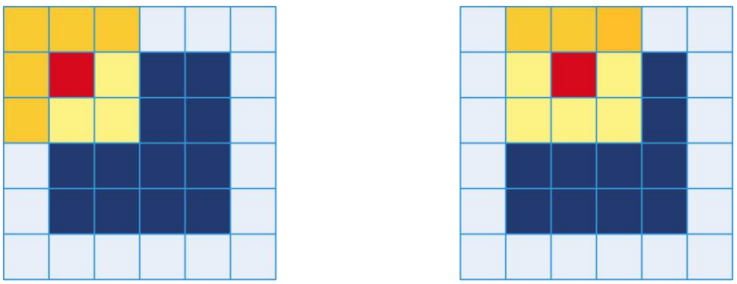

图 3.2 材料数据 Map 过程

对于超过样本边界的范围，即图中深黄色的范围，区域内是没有点的，所以这样划分边界不会影响计算结果。

第二步:将 Map 过程得到的 16 个区块均匀分配到本次实验的 2 个节点中。算法为假设有 n 个服务器节点， m 个任务，利用求余算法 $m\%(n+1)$ 确定任务分配的服务器节点。

第三步：所有 Node 节点运行 Reduce 程序计算区块 R 半径内所有节点数目。并将计算的结果发送回 Master 节点。

第四步：Master 节点对各个节点传回的结果进行合并，并输出计算以后的结果。

3.4 实验结果与分析

3.4.1 实验环境

为了验证和评估该算法的有效性，搭建了 Kubernetes 云平台进行了实验。实验一共用到 Virtualbox 的三台虚拟机每台虚拟机各有 4G 内存，2 核 CPU,32G 硬盘,采用 Centos7.6（64 位）操作系统,使用的 Kubernetes 版本为 1.13.1。使用的监控的时序数据库软件 Prometheus 版本为 2.9.0,监控可视化软件为 Grafana5.2，Python 编程版本为 Python2.7.x。

3.4.2 实验数据来源

实验数据来源于老师科研的二维材料坐标数据，样本有 39429 条。

```
39424  240.5903931 244.6746979
39425  239.7319031 246.9584046
39426  241.8605042 245.3106079
39427  244.3490906 245.1430054
39428  245.5553894 246.2346954
39429  247.0164948 245.9237976
```

图 3.3 数据样本结构

3.4.3 实验功能实现截屏

第一步：搭建 Kubernetes 容器云平台，配置实验基础环境如网络，监控，集群之间的免密登录等。

（1）搭建了一个三节点的 Kubernetes 集群。

```
[root@master01 ~]# kubectl get nodes
NAME       STATUS   ROLES    AGE    VERSION
master01   Ready    master   149d   v1.13.1
node01     Ready    <none>   149d   v1.13.1
node02     Ready    <none>   149d   v1.13.1
[root@master01 ~]#
```

图 3.4 三节点集群展示

（2）安装时序数据库 Prometheus 和可视化界面 Grafana。

```
[root@master01 ~]# kubectl get svc -n kube-system
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
grafana                NodePort    10.105.42.136   <none>        3000:32622/TCP   122m
kube-dns               ClusterIP   10.96.0.10      <none>        53/UDP,53/TCP    149d
kubernetes-dashboard   NodePort    10.111.105.26   <none>        443:30001/TCP    149d
node-exporter          NodePort    10.105.206.40   <none>        9100:31672/TCP   123m
prometheus             NodePort    10.105.141.85   <none>        9090:30003/TCP   123m
[root@master01 ~]#
```

图 3.5 监控软件展示

（3）安装并配置系统基础管理软件如 Dns 解析 coredns，键值数据库 etcd，容器网络 flannel，负载均衡 proxy 等。

```
[root@master01 ~]# kubectl get pods -n kube-system -o wide
NAME                                    READY   STATUS    RESTARTS   AGE    IP              NODE       NOMINATED NODE   READINESS GATES
coredns-78d4cf999f-k98vf                1/1     Running   7          149d   10.244.0.17     master01   <none>           <none>
coredns-78d4cf999f-v6pdb                1/1     Running   7          149d   10.244.0.16     master01   <none>           <none>
etcd-master01                           1/1     Running   7          149d   172.16.8.100    master01   <none>           <none>
grafana-core-78df886fb9-qwfqj           1/1     Running   1          124m   10.244.2.111    node02     <none>           <none>
kube-apiserver-master01                 1/1     Running   7          149d   172.16.8.100    master01   <none>           <none>
kube-controller-manager-master01        1/1     Running   8          149d   172.16.8.100    master01   <none>           <none>
kube-flannel-ds-amd64-dqq4k             1/1     Running   17         149d   172.16.8.102    node02     <none>           <none>
kube-flannel-ds-amd64-r5t6l             1/1     Running   8          149d   172.16.8.100    master01   <none>           <none>
kube-flannel-ds-amd64-z4jk6             1/1     Running   16         149d   172.16.8.101    node01     <none>           <none>
kube-proxy-8mj2                         1/1     Running   7          149d   172.16.8.100    master01   <none>           <none>
kube-proxy-h2rrc                        1/1     Running   16         149d   172.16.8.102    node02     <none>           <none>
kube-proxy-v449d                        1/1     Running   16         149d   172.16.8.101    node01     <none>           <none>
kube-scheduler-master01                 1/1     Running   8          149d   172.16.8.100    master01   <none>           <none>
kubernetes-dashboard-847f8cb7b8-ndgb2   1/1     Running   18         149d   10.244.2.112    node02     <none>           <none>
node-exporter-76jtz                     1/1     Running   1          125m   10.244.1.95     node01     <none>           <none>
```

图 3.6 系统基础管理组件展示

（4）Grafana 监控界面展示，可以监控集群内所有物理服务器节点的 CPU，内存，网络，文件系统利用率；所有容器的 CPU，内存，网络，文件系统利用率，以及其他上百项指标，详细参数可去官网查看，除此之外，还可以自定义图形化界面展示的指标。

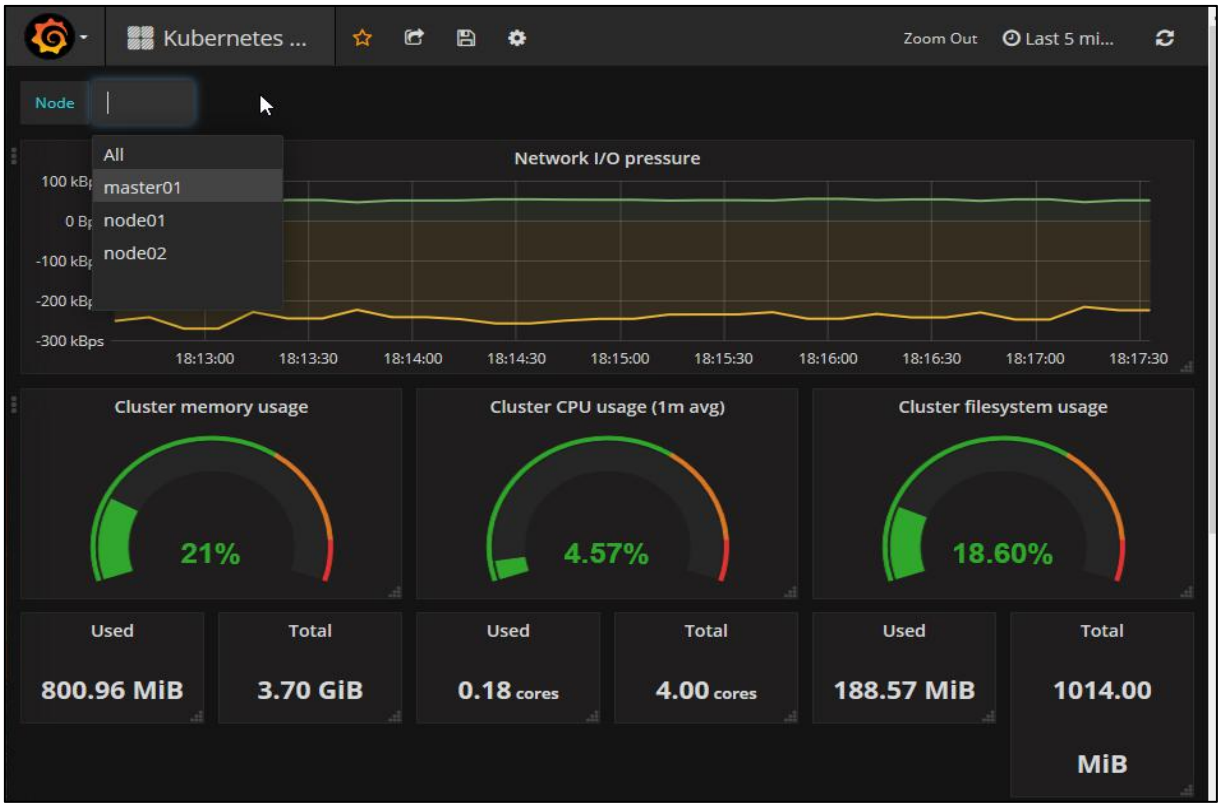

图 3.7 Grafana 监控界面展示（一）

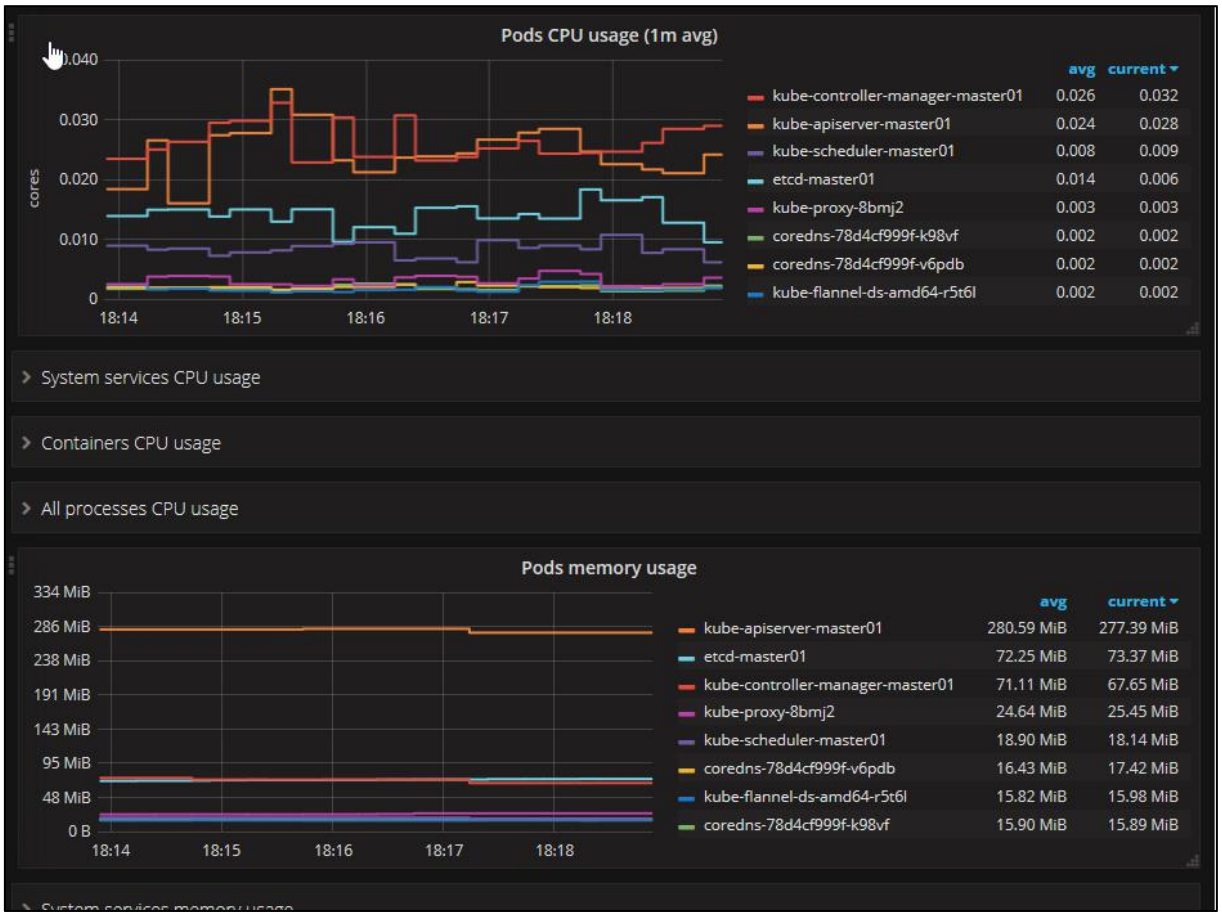

图 3.8 Grafana 监控界面展示（二）

（5）Prometheus 时序数据库实时抓取的各个节点的参数展示。

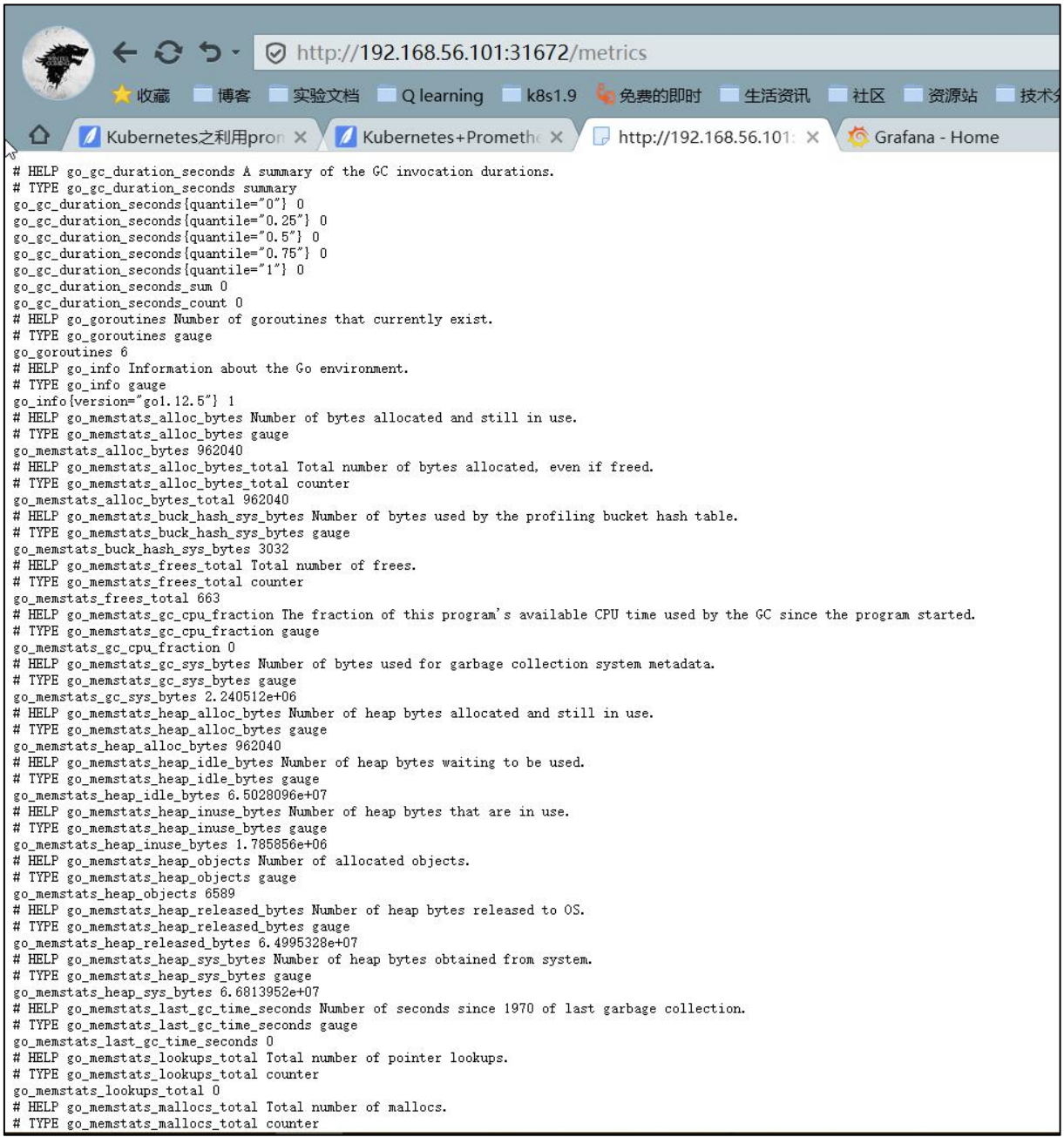

图 3.9 Prometheus 时序数据库实时抓取数据展示

（6）Prometheus 的监控参数有上百项，可以根据需求选择需要的参数进行简单的可视化，Grafana 是专业的监控数据可视化软件，可以弥补 Prometheus 在可视化方面的缺陷。

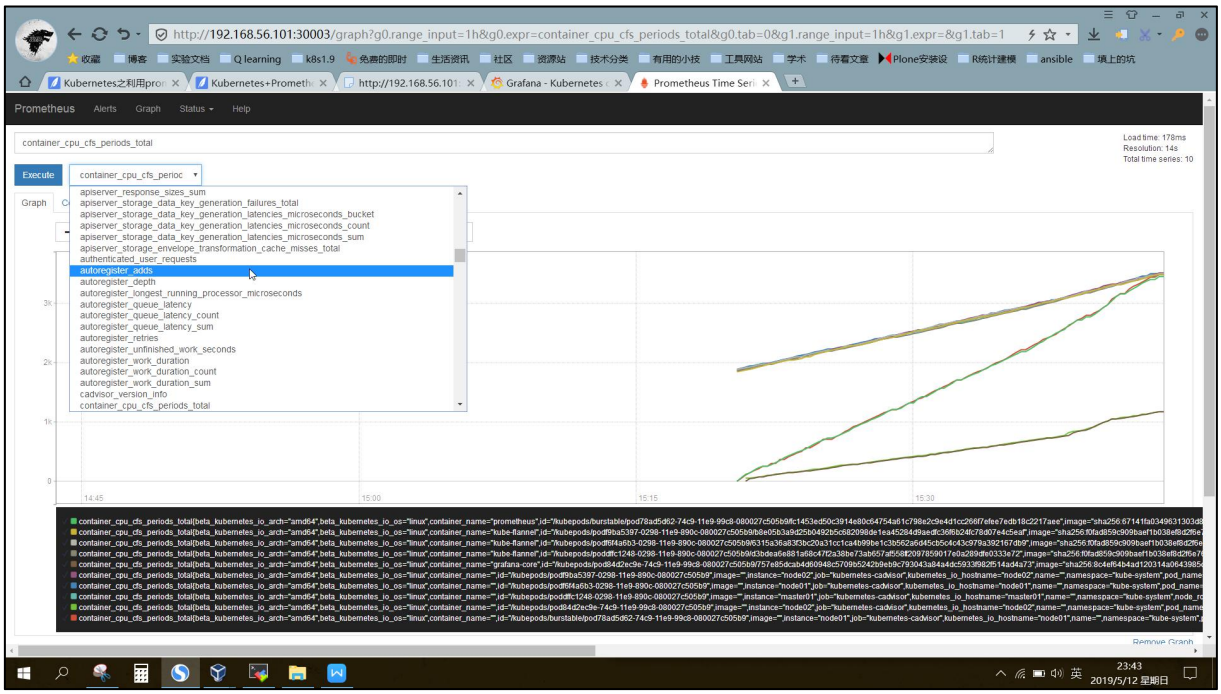

图 3.10 Prometheus 监控指标可视化

第二步：在 Master 和 Node 节点分别编写 Map Reduce 分布式程序，节点程序目录结构如下图所示。

```
[root@master mapreduce-code]# tree
.
|-- code
|   |-- clear.sh
|   |-- format-check.sh
|   |-- help.sh
|   |-- map.sh
|   |-- map-single.sh
|   |-- pythoncode
|   |   |-- format-check.py
|   |   |-- map.py
|   |   `-- merge.py
|   |-- reducer.sh
|   `-- reducer-singer.py
|-- data
|   |-- numx.txt
|   `-- numy.txt
|-- mapped
|-- merge
|-- process
`-- processed

7 directories, 12 files
```

图 3.11 Master 节点程序目录结构

```
[root@node01 mapreduce-code]# tree
.
|-- code
|   |-- reducer.py
|   |-- SendtoMaster.sh
|   `-- test.py
|-- mapped
|-- process
`-- processed

4 directories, 3 files
```

图 3.12 Node 节点程序目录结构

第三步：编写文件目录数据清理脚本。

```
[root@master code]# cat clear.sh
#!/bin/bash
rm -rf /root/mapreduce-code/mapped
rm -rf /root/mapreduce-code/process
rm -rf /root/mapreduce-code/processed
rm -rf /root/mapreduce-code/merge
mkdir /root/mapreduce-code/mapped
mkdir /root/mapreduce-code/process
mkdir /root/mapreduce-code/processed
mkdir /root/mapreduce-code/merge
echo master done!

ssh node01 > /dev/null 2>&1 << eeooff
rm -rf /root/mapreduce-code/mapped
rm -rf /root/mapreduce-code/process
rm -rf /root/mapreduce-code/processed
mkdir /root/mapreduce-code/mapped
mkdir /root/mapreduce-code/process
mkdir /root/mapreduce-code/processed
exit
eeooff
echo node01 done!

ssh node02 > /dev/null 2>&1 << eeooff
rm -rf /root/mapreduce-code/mapped
rm -rf /root/mapreduce-code/process
rm -rf /root/mapreduce-code/processed
mkdir /root/mapreduce-code/mapped
mkdir /root/mapreduce-code/process
mkdir /root/mapreduce-code/processed
exit
eeooff
echo node02 done!
```

图 3.13 文件目录清洗脚本

第四步：编写输入数据的格式检查脚本。

```
[root@master code]# cat format-check.sh
#!/bin/bash
cd /root/mapreduce-code/code/pythoncode/
python format-check.py
[root@master code]# cd pythoncode/
[root@master pythoncode]# ls
format-check.py  map.py  merge.py
[root@master pythoncode]# cat format-check.py
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-
from fractions import Fraction
#导入x坐标
x = []
#for line in open("F:/numx.txt","r"):
for line in open("/root/mapreduce-code/data/numx.txt","r"):
        line = line[:-1]
        x.append(line)
#导入y坐标
y = []
#for line in open("F:/numy.txt","r"):
for line in open("/root/mapreduce-code/data/numy.txt","r"):
        line = line[:-1]
        y.append(line)
# 将字符类型的list转换成数值类型
x = map(float, x)
y = map(float, y)
a = len(x)
b = len(y)
if a == b:
    print("Format Pass!")
elif a != b:
    print("Format Wrong!")
```

图 3.14 数据输入格式检查

第五步骤：编写交互式可视化 Help 帮助的脚本。

```
[root@master code]# cat help.sh
#!/bin/bash
OPTION=$(whiptail --title "Help" --menu "How to use the sofeware?" 15 60 4 \
"step0" "position of the code: cd /root/mapreduce-code/code" \
"step1" "check data format: sh format-check.sh" \
"step2" "clear previous data: sh clear.sh" \
"step3" "map process: sh map.sh" \
"step4" "reduce process: sh reduce.sh"  3>&1 1>&2 2>&3)

exitstatus=$?
if [ $exitstatus = 0 ]; then
    echo "Have a great day!"
else
    echo "You chose Cancel."
fi
```

图 3.15 可视化 Help 代码

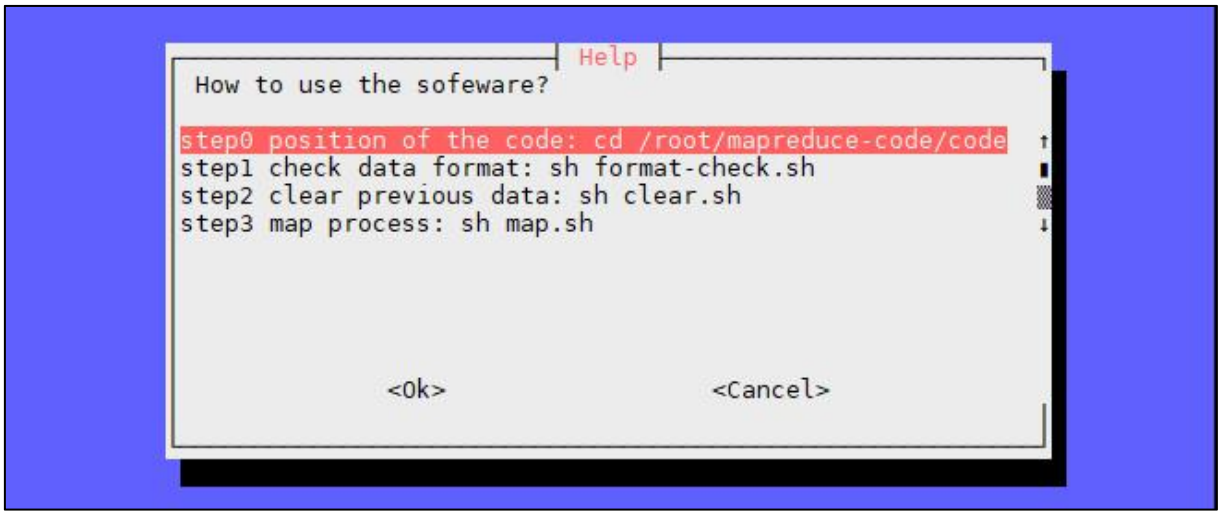

图 3.16 可视化 Help 图形界面

3.4.4 实验结果以及分析

为了证明材料大数据分布式处理系统在计算结果正确的基本条件下，能显著提高材料科学计算的效率，本文将同样的数据进行单机版运行和双节点分布式运行，结果如下图所示：

```
39419 235.2583923 246.5712891 4
39420 236.6407013 247.2406006 5
39421 237.5932923 245.2692871 5
39422 239.1965027 245.622406  7
39423 241.3092957 246.8255921 4
39424 240.5903931 244.6746979 4
39425 239.7319031 246.9584046 5
39426 241.8605042 245.3106079 5
39427 244.3490906 245.1430054 4
39428 245.5553894 246.2346954 1
39429 247.0164948 245.9237976 2
```

图 3.17 材料计算程序运行结果

```
[root@master code]# sh reducer-singer.py
13.879982233
14.0890409946
13.6707820892
13.518447876
14.4468028545
14.5115590096
14.0575959682
13.8111879826
14.4261500835
14.5096540451
13.9933960438
13.7993609905
13.8469820023
13.8317241669
13.4478909969
13.5256547928
Completed!
本次运行时间： 224s
```

图 3.18 单机运行程序效率

```
[root@master code]# sh reducer.sh
13.9032709599
13.6332259178
14.4649150372
14.1005837917
14.4509441853
14.0211591721
13.7757790089
13.5329611301
Completed!
14.2871460915
13.5923788548
14.6377179623
13.9388849735
14.5184109211
13.942043066
14.0216889381
13.6751170158
Completed!
本次运行时间： 114s
```

图 3.19 双节点分布式程序运行效率

从上图中的对比可以看到，分布式材料大数据处理系统能显著提高材料科研计算的程序运行效率。在本次实验中双节点分布式运行相比于单机，计算效率提高了 49.1%。值得注意的是，经过 map 过程得到的区块，计算量是比较均匀的，而这种应用程序消耗资源的均匀性有助于下一章中自适应弹性调度建模。

3.5 本章小结

本章建立了一个材料大数据分布式处理平台，配置了基础的云平台环境，并在此之上根据材料二维坐标数据计算半径内点的个数的案例以 Map Reduce 思想设计分布式算法并在云平台上编程实现，证明分布式算法能显著提高材料科学计算的效率，有广泛的应用前景。

第四章 材料应用自适应资源弹性调度策略

4.1 问题描述

在按需的云环境中，材料科学计算应用程序的负载到达的时间往往是高度动态的。服务器集群可能会收到高度突发的请求并且变得不堪重负，当服务器过载时，其用户会感觉到更长的延时甚至丢失服务；而与之相对的，在冷峰期，大量服务器处于空闲或者低负载状态，却持续保持运行状态消耗电能，产生大量数据中心电力运营成本。为了应对这些挑战，在本文中，提出一种新的策略，用于实现材料大数据平台的虚拟机或容器工作组级别的最优资源利用率的自动弹性调度，以最大程度减少资源浪费，同时避免服务等级协议（SLA）违约，无需人为干预整个过程。为实现优化目标，需要解决三个主要问题：

（1）预测每一个重新资源分配的单位时间需要多少资源；

（2）根据预测的资源需求自适应调整资源上限；

（3）设计优化算法，在成本与延迟之间做出权衡决策，同时满足延迟指标的成本约束和 SLA。

通过移动平均算法对材料应用程序请求进行分析预测下一个单位时间的平均请求数目。将预测的这个平均数考虑为未来一个单位时间内云平台收到的请求分布，作为 M/M/N 排队论的输入，建立虚拟机或容器工作组实例的数目与延迟（或响应时间）之间的关系。真实的材料应用程序请求数目会添加到新的预测中。同时考虑 SLA 违约和最大成本为约束条件，设计了成本和延迟之间的优化模型。

4.2 移动平均预测模型

移动平均预测模型的公式如下：

$$F(t)=\left(A_{t-1}+A_{t-2}+A_{t-3}+\ldots+A_{t-n}\right)/n \tag{4-1}$$

其中 $F(t)$ 是对下一个单位时间的预测，n 是移动平均采集的单位时间的个数，A_{t-1} 是上一个单位时间的实际任务请求数目，$A_{t-2}, A_{t-3}, A_{t-n}$ 分别代表前两个单位时间，前三个单位时间以及前 n 个单位时间的真实任务请求数目。

4.3 Qos 和成本权衡模型

在材料大数据分布式处理系统中的Qos虚拟机或容器工作单位的调度问题可以抽象为一个M/M/N排队问题，如下图所示。

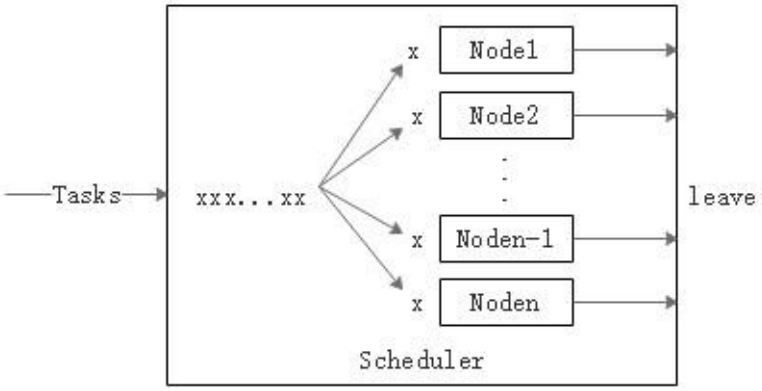

图 4.1 M/M/N 等待制调度模型

在此模型中，单位时间任务请求到达的数目服务参数为$\lambda(\lambda = E(F(t)))$的泊松分布。单位时间到达的平均任务请求数为$\lambda$，在$[0,t]$内到达的任务请求数为$\lambda t$。虚拟机或容器工作单位完成任务的时间服从指数分布，平均服务率为μ，假设集群中一共有w台虚拟机或容器工作单位，则整个集群的平均服务率为$w\mu$。

$$\mu_k = \begin{cases} k\mu & 1 \le k < m \\ w\mu & k \ge m \end{cases} \qquad (4\text{-}2)$$

在本章的自适应资源弹性调度模型中，假设有m台服务器。由于材料应用数据的特殊性，进行Map过程均匀拆分后，每一个区块理论上都有相似的计算量，所以根据不同的任务可以假设每台服务器，平均能同时运行n台虚拟机或容器工作单位，故整个集群针对某种类型的任务总共有$w = m \cdot n$个工作单位，整个集群的平均服务率为$w\mu$。现有的研究[17]已经证明当$\lambda/w\mu < 1$时，系统存在平稳分布，$\rho_1 = \lambda/\mu, \rho = \lambda/w\mu$。

由于模型没有对系统容量进行限制，所以系统的可能状态 $E = \{0,1,2,3,...\}$ 为，如下图所示。

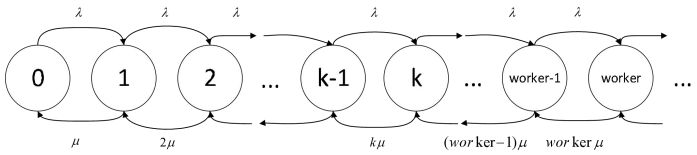

图 4.2 M/M/N 等待制模型状态流程图

其中，状态$k(0 \le k \le w)$表示系统内有k个虚拟机或者容器工作单位忙着运行任务，其余$w-k$个虚拟机或容器工作单位空闲；当状态$k > w$（即到达系统的顾客k超过w）时，w个虚拟机或容器工作单位均忙着运行任务，而余下的$k-w$个顾客排队等候服务。又约定只有一个等待队列，哪个虚拟机或者容器工作单位空闲时，等候中的任务按先后顺序前往空闲的服务工作单位接受服务。平稳态分布如下：

$$p_k = \begin{cases} \dfrac{\rho_1^k}{k!} p_0 = \dfrac{w^k}{k!} \rho^k p_0, & 0 \le k < w \\ \dfrac{\rho_1^k}{w! w^{k-w}} p_0 = \dfrac{w^w}{w!} \rho^k p_0, & k \ge w \end{cases} \tag{4-3}$$

由正则性条件$\sum_{k=0}^{\infty} p_k = 1$，当$\rho < 1$时，有

$$1 = (\sum_{k=0}^{w-1} \frac{\rho_1^k}{k!} + \sum_{k=w}^{\infty} \frac{\rho_1^k}{w! w^{k-w}}) p_0$$

$$= (\sum_{k=0}^{w-1} \frac{\rho_1^k}{k!} + \frac{\rho_1^w}{w!} \frac{1}{1-\rho}) p_0$$

所以，有

$$p_0 = (\sum_{k=0}^{w-1} \frac{\rho_1^k}{k!} + \frac{\rho_1^w}{w!} \frac{1}{1-\rho})^{-1} \tag{4-4}$$

4.3.1 任务队列分析与延迟分析

材料大数据分布式处理系统中的平均等待任务数目：

$$L_q = \sum_{k=n}^{\infty} (k-w) p_k = \frac{\rho_1^{n+1}}{(w-1)!(w-\rho_1)^2} \rho_0 \tag{4-5}$$

材料大数据分布式处理系统中正在工作的工作单位的数目：

$$L_{服} = \bar{k} = \sum_{k=0}^{n} k p_k + w \sum_{k=n+1}^{\infty} p_k = w\rho \left(\sum_{k=0}^{n-1} p_k + \sum_{k=n}^{\infty} p_k \right) = w\rho = \rho_1 \tag{4-6}$$

材料大数据分布式处理系统队长的均值

$$L_s = L_q + L_{服} = L_q + \rho_1 = \frac{\rho \rho_1^w \rho_0}{w!(1-\rho)^2} + \rho_1 \tag{4-7}$$

由 Little 公式可知任务平均排队等待时间和任务平均服务时间分别如下：

$$W_q = \frac{L_q}{\lambda} = \frac{\rho_1^w \rho_0}{\mu w \cdot w!(1-\rho)^2} \tag{4-8}$$

$$W_s = \frac{L_s}{\lambda} = W_q + \frac{1}{\mu} \tag{4-9}$$

通过等待时间W_q可以计算预期的任务响应时间：

$$L(\lambda,\mu,w) = W_q + W_s = \frac{2\rho_1^w \rho_0}{\mu w \cdot w!(1-\rho)^2} + \frac{1}{\mu} \tag{4-10}$$

4.3.2 多约束优化模型

（1）目标函数：优化目标是在最小化材料科学计算服务提供机构的成本（最大化资源利用率)的同时利用最少的SLA违约次数来提高Qos服务质量(如最小化响应时间)。但是，最小化成本与最大化 Qos 服务质量是冲突的。模型中的成本函数C是根据正在运行的工作单位消耗的电费来预测，定义成本函数为：

$$C = f(w)=0.1w \tag{4-11}$$

其表示代表每个工作单位每小时的计算成本为 0.1 元。

可以减少正在服务器节点中运行的工作单位从而休眠服务器节点达到减少成本的目的，但有可能会导致材料计算任务等待的队列过长。与之相反，可以通过激活休眠中的服务器来增加工作单位数目，从而减少等待延时，提高 Qos 服务质量，但可能会导致成本增加。针对这个问题，提出一个成本延迟权衡优化函数如下：

$$\arg\min_{w,\lambda,\mu} V(w,\lambda,\mu) = \alpha \cdot f(w) + (1-\alpha)\cdot L, \alpha \in [0,1] \tag{4-12}$$

α反映成本和延迟的重要性程度的系数。

由于工作单位的数目与延迟的量纲不同，需要对对延迟进行标准化处理：

$$G = \frac{L}{T} \tag{4-13}$$

其中T是 SLA 中定义的延迟阈值，为了规范化虚拟机或容器工作单位的数目，进行归一化处理：

$$C' = F(w) = \frac{f(w) - f_{\min}(w)}{f_{\max}(w) - f_{\min}(w)} \tag{4-14}$$

其中 $f(m)$，$f_{\max}(m)$，$f_{\min}(m)$ 分别表示单位时间的工作单位的成本，最小可能成本和最大可能成本。然后推导出下面目标函数进行优化：

$$\arg\min_{w,\lambda,\mu} V(w,\lambda,\mu)=\alpha\cdot f(w)+(1-\alpha)\cdot G \tag{4-15}$$

根据单位时间 t 内的请求，给出 λ 和 μ，同时可以将 t 内队列等待函数写为：

$$L_t(\lambda,\mu,w)=L_t(w) \tag{4-16}$$

由于资源是有限的，需要设置最大资源上限作为约束。同时 SLA 违约也需要设置上限进行约束。在单位时间 t 内最终的成本和延迟权衡目标函数如下：

$$\arg\min_{w} V(w,\lambda,\mu)=\alpha\cdot f(w)+(1-\alpha)\cdot G_t(w) \tag{4-17}$$

约束条件：

$$\forall_t : C_t(w)\le C_t \max : \Pr\{L_t(w)>T\}\le K\% \tag{4-18}$$

对于材料任务请求，SLA 违约 K 参考其他应用研究，被定义为 $K\in[2,5]$。

假设有 z 台服务器，每台服务器有 w 个工作单位，则在时间 T 内一共可以处理 zw 个请求。这就意味着当队列长度小于 zw 时候，满足 SLA，因为所有请求可以在时间 T 内解决。通过图 4.3.2 我们知道只有在 zw 稳态才能满足 SLA，而其他状态会违反 SLA，所以 SLA 违约可以描述如下：

$$P_r\{L_t(w)\le T\}=\sum_{i=0}^{zw} p_i>(1-K\%) \tag{4-19}$$

表 4.1 确定最优工作单位数目

算法 1 计算最优工作单位数目

输入：

λ - 到达率　　μ - 每个虚拟机或者容器工作单位的处理速度

α - 成本系数　　K - SLA 违约的阈值

$1:\min V=\infty$

$2:$ *for* $n=1,2,...,N$

3: $l_t=L$//公式（4-11）

4: $l_t^{'}=$标准化（l_t）//公式（4-13）

5: $n^{'}=$标准化(n)//公式（4-14）

6: $V=\alpha\cdot n^{'}+(1-\alpha)\cdot l_t^{'}$//公式（4-17）

7: $if\,(n$满足$constra\,\mathrm{int}(K))\quad and\quad (V<previous\,V)$//公式（4-19）

8: $w=n$

9: $\min V=V$

输出：

w－最优的工作单位数目

为了最小化目标函数，找到最优权衡成本和延迟的服务器数目，并同时满足成本和SLA的约束。由于这是复杂的非线性函数问题，并且难以通过数学方法简化。考虑到材料科学计算服务机构提供的服务器数量有限，本章穷举搜索计算所有的w和V，找到最小成本相关的w，如算法1所示。

4.4 最优资源利用率调度模型

定义:在材料大数据分布式处理系统模型中MDP是一个四元组：$Model=\{S,A,R,\gamma\}$。

S是有限状态集合，在本章中$lowload$对应状态0，normalload对应状态1，healthy对应状态2，$overload$对应状态3，dead对应状态4：

$$S=\{lowload,normalload,healthy,overload,dead\}$$

$f_{state}(x,y,z)$是对材料大数据分布式处理系统的状态进行离散化的分段函数，将系统状态有限化。在模型中x,y,z分别代表系统中CPU、内存和网络的利用率。

$$f_{state}(x,y,z)=\begin{cases}0 & 0\le x\le 30and & 0\le y\le 30and & 0\le z\le 30\\ 1 & & & \\ 2 & 60\le x,y\le 80or & 60\le y,z\le 80or & 60\le x,y\le 80\\ 3 & 80\le x\le 100or & 80\le y\le 100or & 80\le x\le 100\\ 4 & x>100or & y>100or & z>100\end{cases}\tag{4-20}$$

A是有限行为集合，在本章材料大数据分布式处理系统模型中Action是一个状态转移的动作矩阵。

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} & A_{03} & A_{04} \\ A_{10} & A_{11} & A_{12} & A_{13} & A_{14} \\ A_{20} & A_{21} & A_{22} & A_{23} & A_{24} \\ A_{30} & A_{31} & A_{32} & A_{33} & A_{34} \\ A_{40} & A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

$R_a(s,s')$是每一次状态转移得到的及时奖励矩阵。

$R_a(s,s')$通过 Q 学习的 e 贪心训练得到，由于与前文中定义的动作矩阵一致，所以$R_a(s,s')$可以作为 Q 学习中的 Q-table。

$\gamma \in [0,1]$是折扣系数。

令$Q(\mathrm{k})$记录调度的奖赏平均值，如果调度了k次，则获得的奖赏序列为$v_1, v_2, ..., v_n$，平均奖赏为：

$$Q(k) = \frac{1}{n}\sum_{i=1}^{n} v_i \tag{4-21}$$

采取对均值进行增量计算的方法更高效的计算奖赏均值，将公式 21 优化为：

$$\begin{aligned} Q_n(k) &= \frac{1}{n}((n-1)\times Q_{n-1}(k) + v_n) \\ &= Q_{n-1}(k) + \frac{1}{n}(v_n - Q_{n-1}(k)) \end{aligned} \tag{4-22}$$

不管任务调度了多少次只需要计算和记录两个值：调度过的次数$n-1$和平均奖赏$Q_{n-1}(k)$。

Q 学习训练过程中，定义的奖励函数为$R(x,y,z)$，$r_i(i=0,1,2,3,4,5)$为各个状态的奖赏值：

$$R(x,y,z) = \begin{cases} r_0 & 0 \le x \le 30 and \quad 0 \le y \le 30 and \quad 0 \le z \le 30 \\ r_1 & \\ r_2 & 60 \le x, y \le 80 or \quad 60 \le y, z \le 80 or \quad 60 \le x, y \le 80 \\ r_3 & 80 \le x \le 100 or \quad 80 \le y \le 100 or \quad 80 \le x \le 100 \\ r_4 & x > 100 or \quad y > 100 or \quad z > 100 \end{cases} \tag{4-23}$$

$R(x,y,z)$倾向于将每台服务的负载保持在 60%到 80%，这样的设计考虑了几个方面。第一，资源利用率如果大于 80%，代表服务器负载过大，资源利用率虽高，但是会影响

服务器性能，导致延迟增加；第二，资源利用率过低小于 30%，虽然能保证任务性能，但造成了数据中心的能耗增加，增加运营成本；第三，在负载保持在 60%-80%范围内，能够在在资源利用率与延迟之间达到平衡，既不会浪费数据中心资源，留出一定资源应对突发峰值进行缓冲，又不会导致延时的过多增加。

Q 学习 e 贪心最优资源利用率算法描述如下：

表 4.2 确定最优工作单位数目

算法 2 确定最优调度的服务器节点

输入：

K - 服务器　　R - 奖赏函数

T - 训练的调度次数　　e - 探索概率

过程：

$1: r = 0$

$2: \forall i = 1,2,...K : Q(i) = 0, count(i) = 0$

$3: for\ t = 1,2,...,T$

4: $if\ random() < e$

5: $k =$ 从1,2，...,K 中均匀分布随机抽取

6: $else$

7: $v = R_{\max}(k)$

8: $r = r + v$

9: $Q(t) = \dfrac{Q(t) \times \text{count}(t) + v}{\text{count}(t) + 1}$ //公式（23）

10: $\text{count}(t) = count(t) + 1$

输出：

调度的最优服务器节点 k

4.5 实验结果与分析

4.5.1 实验环境

实验环境为仿真实验，仿真实验软件为 Pycharm2018，Python3.5.x。实验操作系统

为 Windows10，16G 内存。

4.5.2 实验过程

第一步：利用泊松分布仿真单位时间（每小时）到达的材料计算任务数量，利用指数分布仿真材料计算任务的运行时间。实验假设每台服务器能开 10 台虚拟机或容器工作单位，利用均匀分布产生三次 6 到 10 范围内的随机数分别模拟每个材料计算任务实际运行时消耗的服务器的 CPU，内存和网络负载率。本次实验给定的参数为：$\mu=5, T=0.05, K=5\%, \alpha=0.8$，任务到达速率单位为分钟，最大任务等待时间单位为小时。

第二步：根据算法 1 预测下一单位时间的工作单位（虚拟机或容器工作单位）的数目，将实验预测过程进行可视化呈现。

第三步：根据算法 3 判断增加的工作单位应该调度到哪一台服务器上，并在调度过程中，对集群 CPU，内存和网络进行可视化展示。

第四步：将算法与企业界 Openstack 和 Kubernetes 默认的加权打分策略进行对比。

4.5.3 实验结果以及对比分析

（1）定 r 为实际的材料计算任务数目。w 是策略得到的预测的最佳工作单位的数量。由于要满足 SLA，所以预测结果会略高于实际结果，但预测的总体趋势基本吻合。

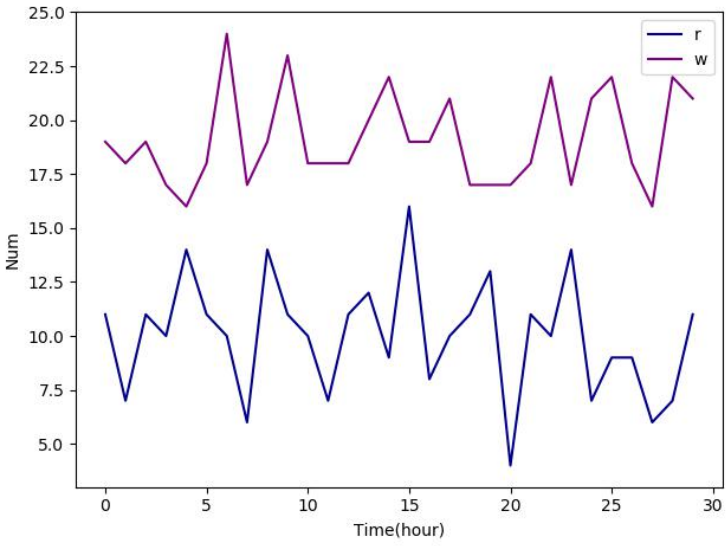

图 4.3 策略预测分配的工作单位数目与实际需求的工作数目对比

（2）30 小时的任务调度仿真过程展示。可以看到实际系统中实际运行的工作单位的数目在 80 个左右达到饱和。

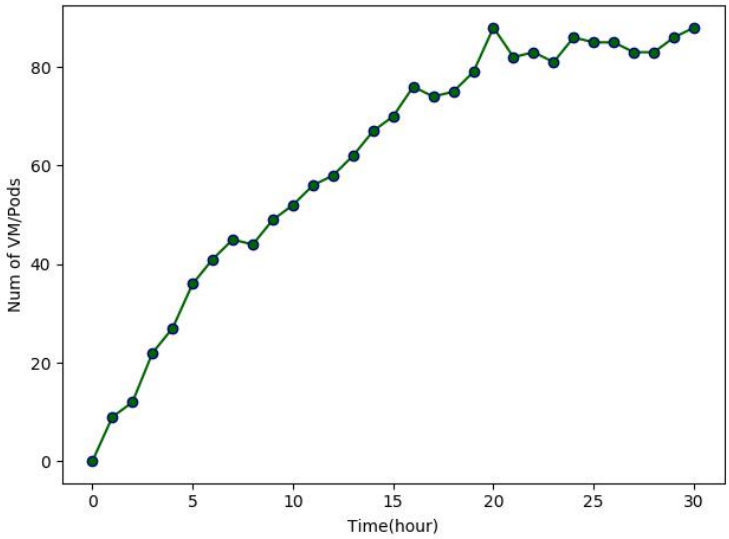

图 4.4 单位时间内正在运行的任务数目

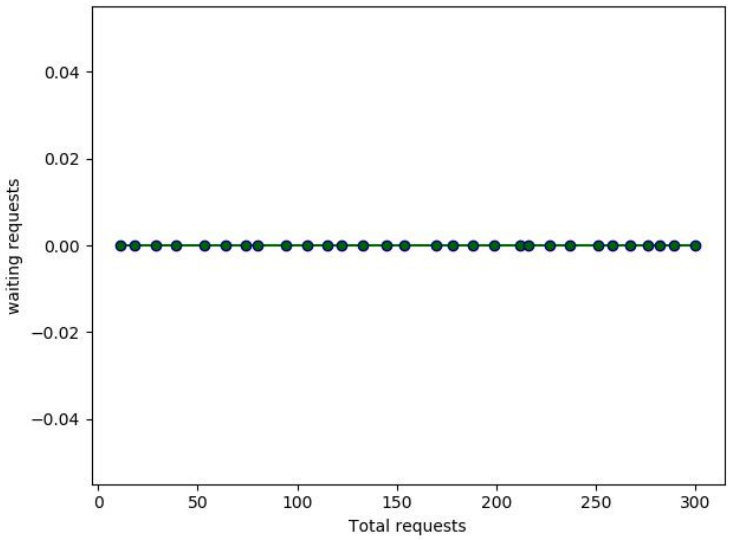

图 4.5 总任务请求数目与队列中等待数目

（3）Q 学习 e 贪心资源利用率优化算法与 Openstack 和 Kubernete 中默认的加权打分资源利用率对比。（a）为加权打分调度过程，（b）为 Q 学习 e 贪心算法调度算法。

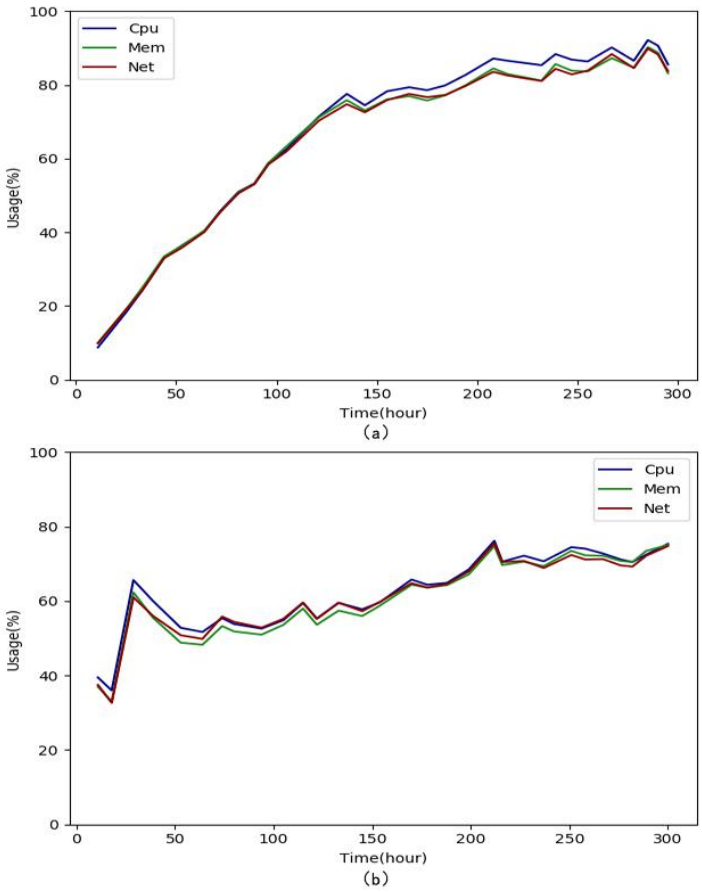

图 4.6 加权打分调度算法与 Q 学习自适应调度资源利用率对比

可以看到相比于加权打分调度算法，Q 学习 e 贪心算法能有效使得服务器集群资源利用率保持在 60%到 80%，集群内基本不会出现资源利用率过低或过高的状况。

4.6 本章小结

针对 Qos 建模在自适应弹性调度系统建模中经常被忽略的不足，提出了一种云资源自动扩展方案。该策略能够在没有人为的干预下自动预测工作任务请求数目，并通过成本延迟权衡得到最佳云资源工作单位需求。由于虚拟机或容器工作单位启动需要一定时间，基于预测的需求自动提前对资源实现伸缩，能有效提高集群资源调度效率。

相比与相关其他研究，本章创新点主要如下：

（1）在虚拟机或容器工作单位的级别上，将预测的请求数目考虑为一个分布，而不是简单使用平均值，这样做能有效减少预测的错误；

（2）提出了一种新的资源自动伸缩方案，决定虚拟机或容器工作单位当前时候是应该扩展数量，保持不变，或减少数量。实验证明该方案能有效权衡预测的成本和延时，并且无需人为干预。

（3）在（2）的基础上提出了一种基于 Q 学习 e 贪心算法的资源利用率优化调度算法，当（2）决定了虚拟机应该扩展或减少数量的决策之后，以集群资源利用率最优为优化目标，决定虚拟机或容器工作单位应该调度到数据中心的那一台服务器上，算法倾向于优先调度任务到某一台服务器上，使得服务器资源利用率保持在 60%-80%，将没有运行工作任务的服务器节点进行休眠，从而提高整个集群的服务器资源利用率，达到节能减成本的目的。

第五章 总结工作

云计算技术必将成为下一代计算技术的基础。材料数据分析往往数据量巨大，个人PC硬件难以承受，即使能运行，也往往效率低下，浪费大量时间。而借助云平台和大数据技术，材料科研机构相关研究人员能够更高效，更方便的进行科研，这是一个值得关注的问题。而与传统云服务提供厂商不同，材料科研分布式平台难以盈利，主要依赖于科研经费，以组织机构为单位，以私有云的形式存在。所以在预算方面往往并不充裕的情况下，如何在保证服务质量的同时提高服务器的资源利用率和降低使用成本显得十分重要。针对以上挑战，本文所做的主要工作有：

1.研究和学习了目前的自适应弹性调度系统相关算法，并进行了对比分析。

2.然后针对现有的自适应系统研究很少考虑Qos服务质量方面的不足，提出了一种基于Qos和成本权衡的自动扩展方案。该策略利用移动平均自动预测材料工作任务请求数目，并通过成本延迟权衡得到最佳云资源工作单位需求。由于虚拟机或容器工作单位启动需要一定时间，基于预测的需求自动提前对资源实现伸缩，能有效提高集群资源调度效率。

3.在云资源自动扩展方案的基础上提出一种基于Q学习e贪心算法的自适应性云资源最大利用率优化算法，选择虚拟机或容器工作单位的最优调度服务器，将无负载的服务器设置为休眠，从而提高集群资源利用率。

4.最后进行仿真调度实验，实验表明所提出的方案实现了资源自动扩展在满足成本延迟权衡和低SLA违约的条件下，能同时有效提高了服务器集群资源利用率。

参考文献

[1] Zhiliang Zhu, Jing Bi, Haitao Yuan, and Ying Chen. 2011. SLA based dynamic virtualized resources provisioning for shared cloud data centers[C]. In 2011 IEEE International Conference on Cloud Computing (CLOUD). 630–637. DOI: http://dx.doi.org/10.1109/CLOUD.2011.91

[2] Jiang, Jing & Lu, Jie & Zhang, Guangquan & Long, Guodong. (2013). Optimal Cloud Resource Auto-Scaling for Web Applications[J]. Proceedings - 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013. 58-65. 10.1109/CCGrid.2013.73.

[3] 马小晋,饶国宾,许华虎.云计算中任务调度研究的调查[J].计算机科学,2019,46(03):1-8.

[4] Cardellini, Valeria & Lo Presti, Francesco & Nardelli, Matteo & Russo Russo, Gabriele. (2018). Decentralized self-adaptation for elastic Data Stream Processing. Future Generation Computer Systems[J]. 87. 10.1016/j.future.2018.05.025.

[5] Chen, Tao. (2016). Self-Aware and Self-Adaptive Autoscaling for Cloud Based Services[J].

[6] Chen, Tao & Li, Miqing & Yao, Xin. (2018). On the effects of seeding strategies: a case for search-based multi-objective service composition[J].

[7] Chen, Tao & Bahsoon, Rami & Yao, Xin. (2018). A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems[J]. ACM Computing Surveys. 51. 10.1145/3190507.

[8] Chen, Tao & Bahsoon, Rami. (2016). Self-Adaptive and Online QoS Modeling for Cloud-Based Software Services[J]. IEEE Transactions on Software Engineering. 10.1109/TSE.2016.2608826.

[9] Chen, Tao & Bahsoon, Rami & Wang, Shuo & Yao, Xin. (2018). To Adapt or Not to Adapt?: Technical Debt and Learning Driven Self-Adaptation for Managing Runtime Performance[J]. 48-55. 10.1145/3184407.3184413.

[10] R. Han, L. Guo, M. Ghanem, and Y. Guo, “Lightweight resource scaling for cloud applications,”[C] in Cluster, Cloud and Grid Computing (CCGrid), 12th IEEE/ACM International Symposium on, 2012, pp. 644–651.

[11] Cardellini V , Presti F L , Nardelli M , et al. Auto-Scaling in Data Stream Processing Applications: A Model-Based Reinforcement Learning Approach[J]. 2017.

[12] Chen, Tao & Bahsoon, Rami & Yao, Xin. (2018). A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems[J]. ACM Computing Surveys. 51. 10.1145/3190507.

[13] Dean, Jeffrey & Ghemawat, Sanjay. (2004). MapReduce: Simplified Data Processing on Large Clusters[C]. Communications of the ACM. 51. 137-150. 10.1145/1327452.1327492.

[14] Kumar, Satish & Bahsoon, Rami & Chen, Tao & Li, Ke & Buyya, Rajkumar. (2018). Multi-Tenant Cloud Service Composition Using Evolutionary Optimization[J]. 972-979. 10.1109/PADSW.2018.8644640.

[15] 陆传赉.排队论[M]，北京：北京邮电大学出版社，2009

[16] Chen, Tao & Bahsoon, Rami. (2017). Chapter 6. Bridging Ecology and Cloud: Transposing Ecological Perspective to Enable Better Cloud Autoscaling[J]. 10.1016/B978-0-12-805467-3.00006-5.

[17] 周志华.机器学习[M]，北京：清华大学出版社，2016

致 谢

本科四年时光转眼已接近尾声。2015 年秋天，对大学充满憧憬的我，自信满满的走进贵州大学西校区，遇见了我的指导老师高廷红老师和我的室友们，开始了我的本科生时光。

四年转眼而过，在这段时间里，我非常感谢我的指导老师高廷红老师，他在讲台上是那么认真负责，幽默风趣，博学多才。同时他对学术严肃的科学观态度和谨密的治学精神以及精益求精的工作作风也深深地感染和激励着我。感谢四年时光中高老师在生活上的耐心指导和学术上的谆谆教导，他的无私奉献把我一步一步指引进科研的殿堂，传授给我严谨坚持的学术态度。我将永远尊敬和感激高老师。

接下来我要感谢我的室友们，感谢她们的陪伴，陪伴我走过本科生中的大部分时光，我们在一起讨论学业上的问题，在一起诉说生活中的烦恼，时不时在一起打游戏放松。就这样我们相互促进，相互鼓励，共同进步。虽然即将离别，但这份友情将刻骨铭心，永远珍藏在记忆之中。

最后，我要感谢我的父母，感谢他们一直以来对我的鼓励与支持，即使在我人生的低谷也对我不曾放弃，耐心陪伴。