2022 届硕士专业学位研究生学位论文

分类号：_____ 学校代码：10269

密级：_____ 学　　号：00000000001

華東師範大學

East China Normal University

硕士专业学位论文

Master’s Degree Thesis (Professional)

论文题目：基于 Kubernetes 的通用云原生大数据架构

院　　系：计算机科学与技术学院

专业学位类别：工程硕士

专业学位领域：计算机技术

指导教师：XX 教授

学位申请人：XX

2022 年 5 月 8 日

Thesis for Master's Degree (Professional) in 2022

University code:10269

Student ID: 00000000001

East China Normal University

Title: General Cloud-native Big Data Architecture With Kubernetes

Department/School: School of Computer Science and Technology

Category: Master of Engineering

Field: Computer Technology

Supervisor: Prof. XXX Xx

Candidate: Xxx Xx

May 8, 2022

华东师范大学学位论文原创性声明

郑重声明：本人呈交的学位论文《基于 Kubernetes 的通用云原生大数据架构》，是在华东师范大学攻读硕士/博士（请勾选）学位期间，在导师的指导下进行的研究工作及取得的研究成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明并表示谢意。

作者签名 **日期** 2022 年 5月 7日

华东师范大学学位论文著作权使用声明

《 基于 Kubernetes 的通用云原生大数据架构》系本人在华东师范大学攻读学位期间在导师指导下完成的硕士/博士（请勾选）学位论文，本论文的著作权归本人所有。本人同意华东师范大学根据相关规定保留和使用此学位论文，并向主管部门和学校指定的相关机构送交学位论文的印刷版和电子版；允许学位论文进入华东师范大学图书馆及数据库被查阅、借阅；同意学校将学位论文加入全国博士、硕士学位论文共建单位数据库进行检索，将学位论文的标题和摘要汇编出版，采用影印、缩印或者其它方式合理复制学位论文。

本学位论文属于（请勾选）

（ ）1.经华东师范大学相关部门审查核定的“内部”或“涉密”学位论文*，于 年 月 日解密，解密后适用上述授权。

（√）2.不保密，适用上述授权。

导师签名 **作者签名**

日期 2022年 5月 7日

* “涉密”学位论文应是已经华东师范大学学位管理办公室或保密委员会审定过的学位论文（需附获批的《华东师范大学研究生申请学位论文“涉密”审批表》方为有效），未经上述部门审定的学位论文均为公开学位论文。此声明栏不填写的，默认为公开学位论文，均适用上述授权）。

杜森硕士学位论文答辩委员会成员名单

姓名	职称	单位	备注
			主席
		华东师范大学	
		华东师范大学	
以下空白			

摘要

随着数据量的不断膨胀，传统的数据处理技术，性能上逐渐不能满足需求。为了解决数据增长带来的挑战，逐渐发展出各种各样的数据处理技术，一定程度上提高了数据处理能力，却带来技术孤岛、数据分散、架构复杂、维护困难的问题，导致数据成本越来越高昂。由于云计算技术的蓬勃发展，“万物上云”成为时代常态，不仅普通系统可以借助云的优势，提高整体能力，数据处理技术更能依托云，提高性能降低成本。虽然一些新型数据处理技术，以云原生为基石，结合多种技术手段，融合事务处理和分析计算，提供统一的访问接口，但是由于其还处于发展初期，受限于基础设施、技术发展等多方面因素，短期还不能完全替换传统数据处理技术。

通过把传统数据处理技术与云进行结合，提高性能降低成本，是减少数据膨胀挑战、降低系统复杂性、节约数据成本的主要方法之一，但是对于该研究的相关工作并不多。数据处理技术与云融合，通常需要特殊改造，不能泛化，云存储性能不高，传统存储不灵活，整体复杂度较高，导致不能很好利用云的优势。本文为了解决这些问题，提出了一种基于 Kubernetes 的通用云原生大数据架构：

1. 基于成熟的云容器编排技术，设计了模块化低耦合的云原生运行环境，提高了数据处理技术上云的效率。

2. 基于云存储、数据加速中间件、容器存储接口等技术，设计了云存储加速策略，实现了存算分离结构，使云上数据系统具有较强的扩展性、灵活性，提高了数据存储规模，降低了数据成本。

3. 通过引入新的并行计算策略，规避了传统大规模并行计算的缺陷，提高了整体查询性能，使 TPS 响应更加稳定 。

4. 通过引入成熟的日志、监控、追踪等技术，设计了大规模云上资源的标准可观测性策略，提高了系统稳健性，降低了维护复杂性。

最后在分析型数据处理技术 ClickHouse 上，验证了本文提出的架构，通过大量的实验从查询性能、TPS 响应等方面与原架构进行对比，结果证明在查询性能上有 18%~60% 的提升，在数据成本上有 50%~90% 的下降，在数据规模、一致性、可靠性、系统扩展性上优于原架构，在运维复杂度上易于原架构。

关键词：大数据；云计算；大数据与云融合；存储计算分离；大规模并行计算；可观测性

ABSTRACT

With the continuous expansion of data volume, the performance of traditional data processing technology is gradually unable to meet the demand. In order to solve the challenges brought by data growth, various data processing technologies have been gradually developed, which has improved the data processing ability to a certain extent, but it has brought the problems of technical silos, scattered data, complex architecture and difficult maintenance, resulting in increasingly high data cost. With the vigorous development of cloud computing technology, "everything goes to the cloud" has become the normal state of the era. Not only ordinary systems could take advantage of the cloud to improve their overall capabilities, but data processing technology can moreover rely on the cloud to improve the performance to reduce the costs. Although some new data processing technologies, take cloud native as cornerstone, combine various technical means, integrate the transaction processing and analytical calculation, and provide a unified access interface, but as they are still in the early stage of development, and limited by many factors such as infrastructure and technological development, they can't completely replace the traditional data processing technologies in a short term.

Combining traditional data processing technology with cloud, in this way, the performance can be improved, the cost can be reduced. This is one of the main methods to reduce data inflation challenge, lower system complexity and obtain data cost savings. But there is not much related work on this research. The integration of data processing technology and cloud usually needs special transformation, which cannot be generalized. The facts that the cloud storage performance is not high, the traditional storage is inflexible, and the overall complexity is high, make it difficult to make good use of the advantages of cloud technology. In order to solve these

problems, this paper proposes a general primary cloud big data architecture based on Kubernetes.

1. Based on mature cloud container orchestration technology, the modular and low-coupling primary cloud operating environment is designed, which improves the efficiency of data processing technology on the cloud.

2. Based on cloud storage, data acceleration middleware, container storage interface and other technologies, the cloud storage acceleration strategy is designed, which realizes the structure of separation of storage and calculation. This makes the data system on the cloud have strong scalability and flexibility, improves the data storage scale and reduces the data cost.

3. By introducing the new parallel computing strategy, the defects of traditional large-scale parallel computing are avoided, the overall query performance is improved, and TPS response is more stable.

4. By introducing the mature technologies of logging, monitoring, tracking etc. the standard observability strategy of large-scale cloud resources is designed, which improves the system robustness and reduces the maintenance complexity.

Finally, on ClickHouse, the analytical data processing technology, the architecture proposed in this paper is verified. Through a large number of experiments come from aspects of query performance, TPS response, etc. compare with the original architecture, the result shows that the query performance is improved by 18%~60%, and the data cost is reduced by 50%~90%. It is superior to the original architecture in data scale, consistency, reliability and system scalability, and is easier to operate and maintain.

Keywords: *Big Data; Cloud Computing; Integration of Big Data and Cloud; Separation of Storage and Calculation; Massively Parallel Computing; Observability*

目录

插图

表格

第一章 绪论

1.1 研究背景与意义

十四五规划把“上云用数赋智[1]”，作为建设数字化时代的关键步骤。云计算属于大规模分布式计算技术，以及其相应商业模式演进的产物。近些年来，随着虚拟化、分布式并行计算、大规模数据管理、容器编排技术、信息安全等各项技术、产品的共同发展，以及服务托管、后向收费、按需交付等商业模式的快速演进，使云计算逐渐演变成，社会发展的基础设施能力，为“万物上云”提供了很好的契机。

数据处理技术经过几十年的发展，积淀了许多成熟可靠的系统和工具，并为各行各业广泛地应用。随着近些年数据量的不断激增，传统的数据处理技术，逐渐显得力不从心。新型的数据处理技术，为了更好的处理大规模数据，在设计之初通常会采用云原生的方式，以便于充分利用云近乎无限的存储、计算能力。

新型的数据处理技术虽然有很多优势，但是一方面由于其还处于发展中阶段，不够成熟有一定的缺点，另一方面由于传统数据处理技术的使用，已经非常广泛深入，直接换成新技术还存在困难。如图 1-1 所示，根据 DB-ENGINES 的全球数据库热度趋势，可以看出传统的数据处理技术，如 Oracle、MySQL、SQLServer、PostgreSQL 等，依旧占据主导地位。新型的数据处理技术，如 Snowflake、DynamoDB 等，还处于发展初期，使用普及率远低于传统数据处理技术。

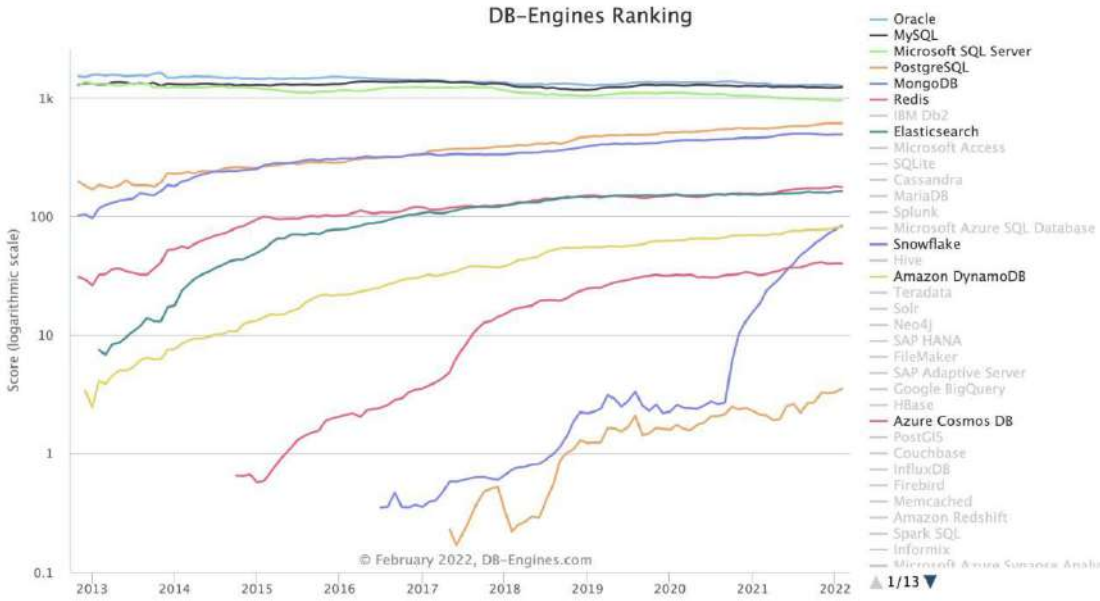

图 1-1：DB-ENGINES 全球数据库热度趋势[2]

如何通过技术化改造，使传统数据处理技术，也能借助云近乎无限的存储、计算能力，从而提高本身数据处理能力，就成为一个亟待解决的问题。

传统数据处理技术上云与云融合，有两种主要方式：一、把本地基础设施换成云上基础设施，利用云便捷的资源伸缩，简单的维护以及稳定的服务能力，数据处理技术的运行维护方式不变。二、以云的基础设施为本，采用容器管理技术，构建云原生的数据处理技术，充分利用云的弹性能力。

第一种方式目前业界用的相对较多，能够规避本地化资源固定的问题，享有一定灵活性，降低整体维护难度，不过数据处理技术的综合能力并得到太多改善。

第二种方式用的比较少，一方面因为传统的数据处理技术，在架构设计之初没有考虑云原生，改造成云原生的难度较高；另一方面因为云本身是动态的、灵活的、相对不稳定的，数据处理追求可靠、高效、稳定，数据是应用的基石，需要极强的稳定性、可靠性保证。这些因素限制了传统数据处理技术与云高效融合，共同发挥两者的优势。

为了使数据处理技术更好的与云进行结合，充分利用云的弹性计算存储能力，提高数据处理技术的性能，降低数据处理的成本。本文设计了一种通用的云原生大数据架构模型，带来四个方面的创新。

1. 通过把数据处理技术容器化，并结合业界成熟稳定的容器编排系统，提供一个简单、可靠、高效、稳定的云原生运行环境，解决数据处理技术上云的困难。

2. 结合云存储、数据加速中间件和容器存储接口等技术，首先提高云存储的读写性能，使其满足常规的数据处理操作。然后实现数据存储与计算的完全分离，让数据处理技术充分发挥云的弹性能力，享有云近乎无限的存储、计算能力，从而不仅提高了数据处理的能力，而且大幅度的降低了数据存储的成本。

3. 在云的赋能下，以及存算分离的支持下，实现了大规模并行计算（Massively Parallel Processing, MPP）模型的改进版，基于云的大规模并行计算(Cloud based Massively Parallel Processing, CMPP)，从而提高数据处理技术的并发能力，提高请求响应的稳定性。

4. 为了保障数据处理技术在云上稳健的运行，结合业界成熟技术，实现整体架构的可观测性。

1.2 国内外发展现状

1.2.1 数据处理技术的发展

1961 年 Charles Bachman 等人设计了，第一个网状结构的计算机数据库管理系统，1968年 IBM (International Business Machines Corporation) 开发了层次的数据库管理系统。这两种都是实验性的先驱。因为其并没有现在我们所熟悉的 SQL (Structured Query Language) 和关系表，只能使用指针的方式来操作数据，需要开发人员耗费大量的精力来管理和维护。

直到 1970 年，E. F. Codd[3] 建立了现在众所周知的关系数据模型，随后 IBM 开始了著名的 System R[4] 研究项目，该研究项目是第一个实现 SQL 和事务的数据库管理系统，System R 的设计对后来的各类数据库，都产生了非常积极的影响。关系模型摆脱了查询和数据存储之间的紧密耦合，查询独立于存储，数据库可以自由地在幕后进行优化。开发人员无需知道背后的存储方式，只需要通过 SQL 与数据库进行交互，这对于使用者非常友好，极大地促进了关系数据模型的普及。

随着后来的 Oracle、DB2、SQL Server 等商业数据库发布，数据库从成长期，慢慢地走向成熟，关系型数据库成为了主流，SQL 语言成为了标准查询语言，直到现在我们依然在使用。由于数据量的不断增长，商业数据库的使用成本越来越昂贵，一些开源关系型数据库：MySQL、PostgreSQL 也逐渐获得大量生机，并被大多数互联网企业所使用。

伴随着互联网的三次浪潮，网上冲浪的用户数量不断攀升，互联网应用也呈多元化局面，电商、网游、音视频、社交、娱乐等全面发展齐头并进。信息技术的快速发展，使得人们查询、存储、分析数据的能力不断提高，全球数据规模高速增长、海量集聚。传统的关系型数据库，已经无法满足互联网日益增长的数据处理需求。

于是人们开始在关系型数据库上尝试一些新方法，比如：设计反范式[5]，以减少多表连接；设计分库分区分表[6]，以减少数据扫描；设计主从复制[7]、读写分离[8]，以减少单机性能瓶颈的限制等。这些方法在一定程度上减缓了数据爆炸式增长带来的急迫性，不过并没有从根本上解决问题，反而为之后的系统维护带来更多困难。

当大多数人的关注点还聚集在单机上，思考如何提升单机的性能，寻找更快更强的服务器时。Google 在 2004 年前后发表了三篇论文，也就是业界常说的“三驾马车”，分别是分布式文件系统 (Google File System, GFS)[9]、分布式计算框架 MapReduce[10] 和 NoSQL 数据库系统 BigTable[11]，告诉我们可以部署

一个大规模的集群服务，采用分布式的手段，将海量的数据存储在这个集群上，然后利用集群上的所有机器资源，同时进行数据处理。

Lucene 开源项目的创始人 Doug Cutting，根据 Google 三篇论文的原理，初步实现了类似 GFS 和 MapReduce 的功能。后来把这些功能剥离出来，成立单独的大数据项目，这就诞生了赫赫有名的 Hadoop[12]。围绕着 Hadoop 有一系列的大数据处理技术诞生：HBase[13]、Hive[14]、HDFS[15]、Pig[16]、Storm[17]、Spark[18] 等，在 Hadoop 之外也有许多用于解决，细分领域问题的大数据技术：专注于文本搜索的 ElasticSearch[19]、专注于文档处理的 MongoDB[20]、专注于图数据处理的 Neo4j[21]、专注于并行计算的 Greenplum[22]，专注于内存键值存储的 Redis[23] 等。从此便开启了大规模数据处理的 NoSQL 时代。NoSQL 官方解释为不仅仅是 SQL，通过非关系型数据模型来组织数据，通过多种查询语言获取数据，消减传统数据库中事务控制能力，增强大规模数据的吞吐能力。

NoSQL 技术在一定程度上，解决了数据规模大的问题，互联网行业的多数企业，或多或少都会用到相关的技术。在这些技术的使用，更加深入之后，人们发现 NoSQL 并不能代替 SQL。当对数据准确性要求比较高时，还是需要使用关系型数据模型，使用有事务隔离，并发控制的数据库。这个时候业界有种观点是，关系型数据库发展了几十年，打磨出很多成熟的技术，不应该把这些技术丢掉，应该和现在的技术结合起来，创造出新的更强大的数据库。

2012 年 Google 发表了 Spanner[24]、F1[25]、Percolator[26] 等一系列的文献, 阐述了大规模分布式关系型数据库如何实现，提出了 NewSQL 概念。接踵而来，2014 年国外团队基于 Spanner/F1 的理念，开源了 CockroachDB[27] 数据库：标准的 SQL 接口、兼容 PostgreSQL 协议、强大的扩展能力、支持高并发访问、弹性扩容、多副本一致性、服务高可用、分布式事务、多区域部署等特性。2017年国内团队基于 Spanner/F1，开源了 TiDB[28] 数据库：支持事务处理与分析处理的融合型分布式数据库产品，具备水平扩缩容、金融级高可用、大规模分布式、兼容 MySQL 等优势。

1.2.2 云计算技术的发展

云计算技术是大规模分布式技术，以及其相应商业模式演进的产物。它的发展更多的依赖于资源虚拟化、数据分布式存储、容器大规模管理、信息安全等各项技术、产品的共同发展。

虚拟化方面，1998 年 VMware 公司推出 x86 虚拟化技术，次年推出 VMware Workstations。1999 年 IEEE(Institute of Electrical and Electronics Engineers) 颁布 VLAN(Virtual Local Area Network)[29] 标准，次年颁布 VPN(Virtual Private Network) 标准。2001年 VMware 公司发布 ESX(Elastic Sky X) 和 GSX，2005年 Citrix 公司发布 Xen 3.0，2006年 RedHat 公司发布 KVM(Kernel-based Virtual Machine)[30]，2009年 Innoteck 公司发布 VirtualBox。这些虚拟化技术发展与逐渐成熟，为云计算提供坚实的基础。

基于虚拟机技术，众多云厂商推出了云计算服务。2006 年亚马逊公司推出了S3(Amazon Simple Storage Service)[31] 分布式云存储，以及 EC2(Elastic Compute Cloud) 云虚拟机。2008 年谷歌公司发布 Google App Engine 服务。2009 年 HeroKu 公司推出第一款平台即服务产品。2010 年微软发布 Microsoft Azure 云平台服务。2011 年，Openstack 公司推出开源基础架构即服务产品，Privotal 发布 Cloud Foundry，作为第一个开源的平台即服务产品。随后 2013 年谷歌推出 Google Cloud Engine 云平台，2014 年亚马逊率先推出 Lambda，成为第一个函数即服务的产品。

随着虚拟化技术、云托管平台的不断发展，2013 年 DotCloud 公司发布 Docker，容器开始迅速普及。2014 年谷歌公司发布 Kubernetes[32]，用来管理云上的大规模容器服务。2015 年云原生计算基金会 CNCF(Cloud Native Computing Foundation) 成立。随着容器技术逐渐成熟，围绕容器编排多方进行技术较量，最终在 2017 年，Kubernetes 胜出并成为容器编排的标准，在之后逐渐形成云原生的生态。

1.2.3 数据处理技术与云计算技术的结合

云计算技术兴起之前，对于大多数企业来说，硬件的自行采配和机房租用，是主流的 IT 基础设施构建方式。除了服务器本身，机柜、带宽、交换机、网络配置、软件安装、虚拟化等许多底层事项，总体上需要相当专业的人员负责，资源调整的周期也比较长。云计算技术兴起之后，越来越多的企业倾向于使用云厂商的基础设施，逐渐替换掉本地私有化的机房，于是传统的数据处理技术也部署到了云平台，获得云平台的部分优势。

在云计算技术发展早期，国外的亚马逊和国内的阿里云，相继推出关系型数据库服务 (Relational Database Service, RDS)[33][34]，优化改良原数据处理技术的部分模块（权限认证、数据复制、主从协作等），使其能稳定可靠地运行在云基础设施（云服务器、云防火墙、高速云网络）之上，最终获得更高效的数据处理能力。虽然这种 RDS 化改造，提高了处理性能，但是由于改造的复杂度较高、改造之后的产品和原技术有一定差异（部分功能被限制，以满足云环境）、改造方案基本私有化，所以很难在更多场景中通用。

随着云计算技术的逐渐成熟，云原生概念的普及，一些基于云原生设计理念的数据处理技术，也通过云提供更加便捷服务。比如，以事务型为主的数据处理技术，有国外的 Spanner Cloud[35]、Cockroach Cloud[36]，国内的 TiDB Cloud[37]；以分析型为主的数据处理技术，有国外的的 Snowflake Cloud[38]、国内的 Kyligence Cloud[39]。这些数据处理技术通常采用全新的架构设计，引入一致性共识算法、多阶段事务提交、并行数据处理、灾难恢复、分区容错等新技术，能够解决传统数据处理技术，所不能解决的一些问题。但是，一方面由于处于发展初期，还不够成熟稳定，而且技术上普遍复杂度较高，同时需要消耗较多的云计算资源，导致各方面成本居高不下。另一方面由于传统数据处理技术使用广泛，且成熟稳定，新型数据处理技术处于不断完善发展阶段，并未普及。所以使传统数据处理技术结合云的能力，以提高本身的性能，也应运而生为热门研究方向。

文献[40]提出了一种基于 SaltSatck 的云数据处理方案，通过 OpenStack 相关技术组织管理云服务器，在云服务器之上运行数据处理技术，并通过对应的后台管理系统，进行数据处理服务的监控与维护，使传统数据处理技术能够做到云上高可用。从整体上来看，该文献提供了较为全面的云环境支持，使得数据处理技术有一定的性能提升。但是还存在的一些不足：1. 由于整体操作较为繁琐，共享存储为传统磁盘扩展性不高；2. 服务器的调度与管理不够灵活，所以只能提高部分处理能力，同时引入了较多复杂性。

文献[41]提出了一种云环境下大数据计算处理的动态调度框架，采用 Kubernetes 编排管理云环境，利用云存储来共享数据，使用 MapReduce 来计算数据，并引入机器学习调度计算任务，以提高整体性能。从整体来看，该文献利用了云编排系统的优势，进行服务调度，同时结合了大容量的云存储，解决容量问题，但是还存在的一些不足：1. 云存储相对传统磁盘读写吞吐能力较低，直接作为主存储使用，性能较差；2. 只使用 MapReduce 作为计算框架，限制了架构的通用性；3. 引入机器学习，提高了复杂性。

文献[42]通过容器化数据处理技术，以集群方式运行在云环境中。主要采用 Kubernetes 管理云资源，使用 NFS(Network File System)[43] 云存储共享数据，借助 MyCat 中间件管理数据处理技术的访问，最终提供统一的访问接口。从整体来看，该文献利用了云编排系统的优势，进行服务调度，同时结合了大容量的云存储，解决容量问题，但是还存在的一些不足：1. 使用中间件来管理数据访问，中间件将会成为并发访问的瓶颈；2. 使用了特定中间件，限制了架构的通用型，不能扩展给其它数据处理技术使用；3. 直接使用 NFS 云存储，性能会比较差。

文献[44][45][46]也提出，如何在云上运行传统数据处理技术，以提升原技术的能力。不过这些文献的研究，1. 通常只针对特定数据处理技术（MySQL、PostgreSQL、MongoDB 等），很难直接应用到其他技术上；2. 对云虚拟化管理技术研究不够深入，不能充分发挥云的弹性伸缩能力；3. 通常直接使用云存储，云存储自身读写性能较低，导致整体性能没有太多提升；4. 数据处理技术

上云之后，没有提供监控、通知、可视化、追踪这些观测能力，导致整体安全性、稳定性、可靠性无法衡量。

本文针对前人研究存在的不足，1. 首先采用低耦合高内聚的思想，模块化分层设计整体架构，以满足通用性的要求，能够支持较多的数据处理技术。2. 以 Kubernetes 云容器编排技术为基础，更多地使用其原生能力（存活探针检测、负载均衡、资源调度与自动伸缩、服务优先级、存储接口等），减少其他技术的引入，降低整体复杂度，提高性能；3. 集成云存储加速中间件（压缩存储、分散文件、本地缓存、并行处理等），充分发挥云存储的优点：大容量、高可靠，大带宽、高并发，弱化云存储的缺点：读写吞吐慢。4. 按照可观测性要求，引入成熟组件，提供标准监控、通知、可视化、追踪能力，满足数据处理技术上云后的维护问题。

1.3 本文的主要内容

通过分析数据处理技术和云计算技术的演进特点，深入研究数据处理技术与云计算技术的融合策略。结合前人的研究基础，为了使数据处理技术上云更加容易，为了使上云策略具有普适性，为了解决海量数据的存储计算难题，为了优化 MPP 并发计算效率，为了提高云上集群的容灾能力。

本文首先采用 Kubernetes 容器编排系统，按照模块化分层的方式，设计了一个通用的元原生数据处理技术架构。使得数据处理技术上云，只需要简单的容器化操作，而不需要复杂的架构改造，降低上云难度。同时由于架构的通用设计，没有特殊的依赖组件，能够适应较多的数据处理技术，使得上云策略更具有普适性。

然后基于该通用架构，设计了存储与计算分离的策略，使得数据处理技术上云后，不受磁盘存储能力与扩展能力的限制，能够充分利用云的弹性计算优势。设计了云存储优化加速策略，使得数据处理技术上云后，不仅拥有海量的存储空间，同时也能达到甚至超过，传统物理磁盘的读写吞吐能力。

然后基于该通用架构，设计了云上大规模并行计算策略，以避免传统大规模计算策略的并发能力低、集群规模小、稳定性差的问题。并结合抽象计算公式，评估分析了传统大规模计算策略的缺点，云上大规模并行计算的优点。

然后基于该通用架构，设计了标准的可观测性策略，使得云上数据处理技术，能够更加容易的被监控、测量、追踪、可视化，从而保证集群的健康运行与容灾恢复。

最后对 ClickHouse 应用本文提出的通用架构，结合传统集群架构、传统单机架构，进行科学实验分析，以验证本文架构的合理性与创新性。经过大量的性能压力测试，通过收集大量的实验结果，以及对结果的统计分析。最终得出：本文架构在查询性能上，相较于集群架构提高了 18%，相较于单机架构提高了 60%，同时 TPS 表现更加稳定；在存储成本上，相较于集群架构降低了 90%，相较于单机架构降低了 50%；在数据规模、一致性、可靠性、扩展性上，均优于集群和单机架构；在运维复杂度上，均低于集群和单机架构。

1.4 本文的组织结构

第一章，主要介绍了研究的背景与意义，以及国内外的发展现状、本文的解决方法和研究贡献。

第二章，主要介绍了本文提出的通用云原生大数据架构，所用到的相关技术：Kubernetes、JuiceFS、OSS、Prometheus、Fluentd 等，为后面章节的内容做铺垫。

第三章，首先分析了数据处理技术上云的困难，然后提出了数据处理技术上云的策略，最后设计了一个通用的云原生大数据架构，用于简化上云流程，提高上云效率，使云上数据处理更加稳定可靠。

第四章，首先分析了海量数据存储计算的困难，其次分析了云存储的优势与劣势，以及云存储读写性能优化问题，然后提出了云存储优化加速策略，最后在通用元原生架构和云存储加速的基础上，提出了存储计算分离策略。使得

云上数据处理，不仅享有高性能、易于水平扩展的海量存储，而且还大幅度降低数据存储的成本。

第五章，首先分析了传统大规模并行计算存在的问题，然后提出了一种基于云的大规模并行计算策略，最后通过计算公式评估两者的差异，证明新的计算策略，能够使用更大规模、更高并发的数据计算场景。

第六章，首先介绍了可观测性的内容，其次分析了可观测性的重要性，然后分析了系统达到可观测性标准的困难，最后提出了本文通用架构可观测性的实现策略。

第七章，选择一个具体的数据处理技术（ClickHouse），通过对比分析改造前和改造后的数据处理性能、数据扩展性、服务稳定性等七大维度，论证本课题的可行性以及创新性。

第八章，针对本文的研究，总结和提炼了主要贡献，以及提出了未来的展望。

第二章 相关技术

2.1 Kubernetes

Kubernetes[32] 是谷歌公司开源的容器编排管理系统，是谷歌多年大规模容器管理技术 Borg 的开源版本。旨在让使用者不必操心资源管理的问题，使其专注于核心业务，并且做到跨多个数据中心的资源利用率最大化。主要功能包括：基于容器的大规模应用部署、维护和滚动升级；负载均衡和服务发现；跨机器和跨地区的集群调度、自动伸缩；无状态服务和有状态服务管理；广泛的存储卷支持；插件机制保证扩展性；Kubernetes 发展非常迅速，属于容器编排领域的引领者。

如图 2-1 所示，Kubernetes 分为 Client、Master 和 Cluster 三部分，Client 提供和用户交互的接口，Master 负责管理整个系统，Cluster 负责接受 Master 的指令完成工作。Kubernetes 可以很好的运行在各大公有云平台、私有云平台，屏蔽了底层资源调度的细节，帮助应用上云，提供标准的访问、调度、管理支持。

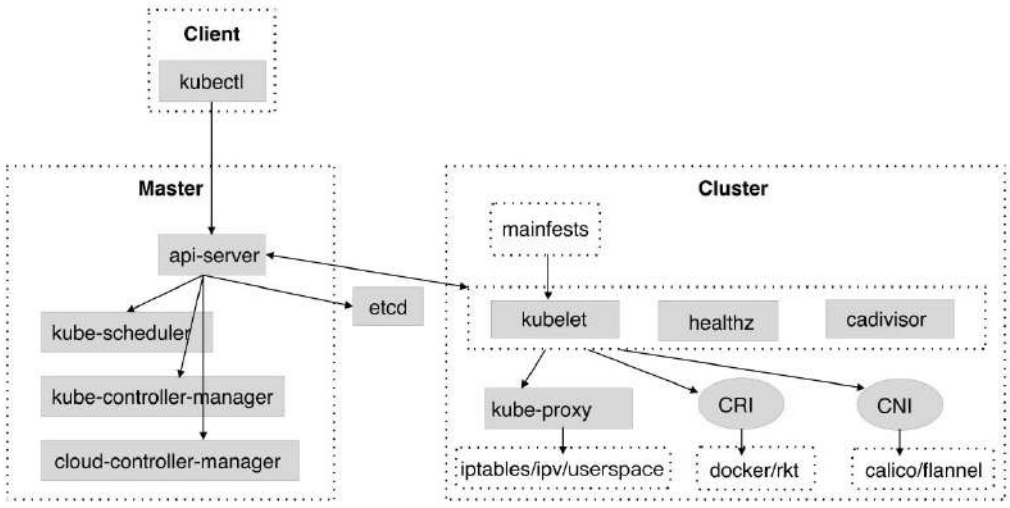

图 2-1：Kubernetes 架构示意图

2.2 Rancher

Kubernetes 容器管理系统，默认通过 Kubectl 终端命令方式进行容器管理操作，终端操作的优势是速度快、效率高，缺点是难度较大、容易出错。Rancher[47] 采用简单易用的 Web 管理界面，通过调用 Kubernetes API 操作接口，集成多种通用服务模版，大幅度降低了容器管理技术，部署容器应用的难度。使用 Rancher 作为容器管理终端，能简化数据处理技术在云上的运行维护。

2.3 JuiceFS

云存储设计之初是为了满足大容量、低成本、高可用的目标，所以常规的云存储读写吞吐能力并不高。容器应用的数据卷如果只使用本地磁盘，则没办法满足大容量、低成本、高可用的要求。在云存储之上再包装一层，使用缓存、压缩、元数据等技术加速云存储，则可以解决云存储的性能问题，更好地利用云存储的大容量、低成本、高可用优势。

如图 2-2 所示，JuiceFS[48] 是一款高性能 POSIX(Portable Operating System Interface for Computing Systems) 文件系统，专门针对云原生环境特别优化设计。使用 JuiceFS 存储数据，数据本身会被持久化在对象存储，比如：OSS、S3、Blob 等，而数据对应的元数据可以根据场景存放在 Redis、MySQL 等多种数据库中。JuiceFS 属于一个中间件，数据存储层可以使用多种存储引擎，数据应用层可以对接多种使用场景，使整体架构的扩展性更优良。

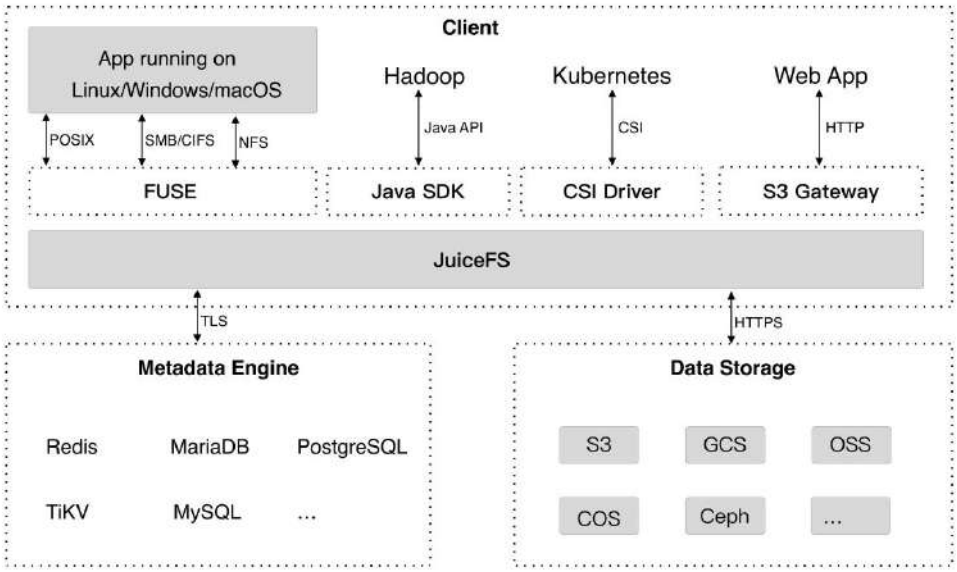

图 2-2：JuiceFS 架构示意图

2.4 Redis

Redis[49] 属于内存型数据库，拥有较高的读写速度，可以通过快照或异步写的方式，持久化数据到磁盘，便于灾难恢复。同时作为 JuiceFS 的元数据存储引擎之一，可以用来管理元数据，提供高效的访问性能。Redis 的系统架构如图 2-3 所示，主要包括键值数据处理、数据缓存及管理、用于系统扩展的主从复制/集群管理，以及插件化功能扩展的模块系统。

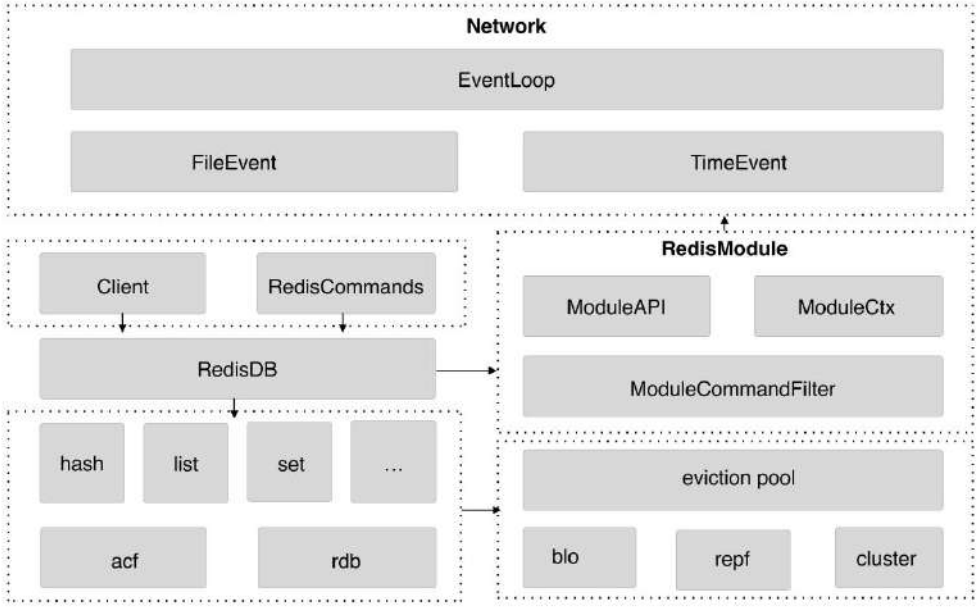

图 2-3：Redis 架构示意图

2.5 OSS

对象存储服务(Object Storage Service, OSS)[50] 是阿里云提供的海量、安全、低成本、高持久的云存储服务，支持版本控制、分区桶策略、跨区域复制、数据加密等特性，和其他同类产品：Amazon S3(Simple Storage Service)、Google GCS(Cloud Storage)、Azure Blob Storage 等，并列为工业界主流的云存储解决方案。

如图 2-4 所示，对象存储服务采用分层架构设计，对外提供简单易用的 API 访问接口，对内使用复杂的技术、算法保证数据一致性、高可用性。数据至少保存3个副本，分散在多个物理区域，中间通过高速光缆网络互连。对于数据的访问、更新，会通过特殊的算法，做相应的缓存处理、预读处理、并发执行，以保证合理的访问速度。

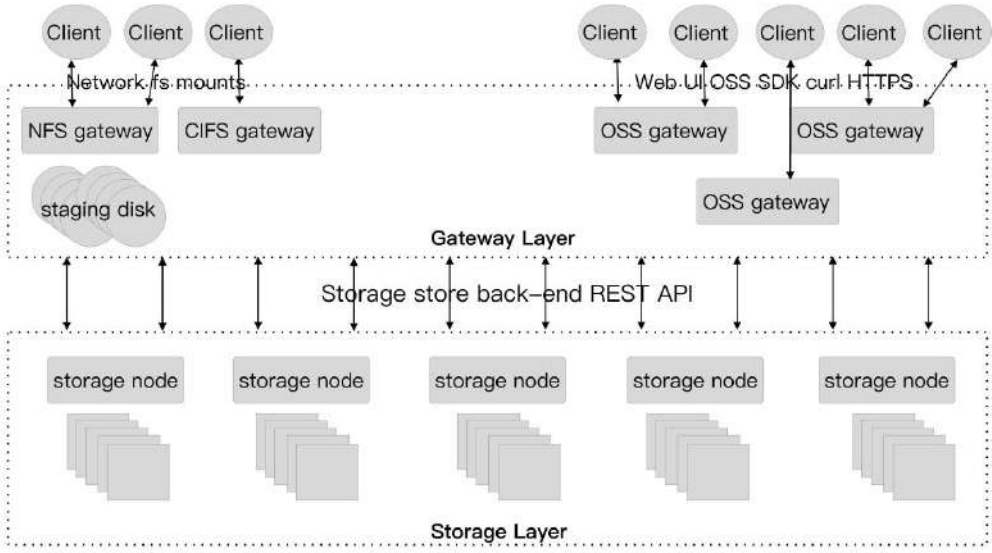

图 2-4：对象存储架构示意图

2.6 Prometheus、AlertManager、Grafana

Prometheus[51] 是一个开源的监控系统，最初由 SoundCloud 构建。自2012年成立以来，许多公司和组织都采用了 Prometheus，该项目拥有非常活跃的开发人员和用户社区。它通过推送或抓取两种方式来收集数据，收集到的数据有两个去向，一个是报警，另一个是可视化。

AlertManager[52] 主要用于接收 Prometheus 发送的告警信息，它支持丰富的告警通知方式，而且很容易对告警信息进行去重，降噪，分组，策略路由，属于相对成熟易用的告警通知系统。

Grafana[53] 是一个开源的度量分析与可视化套件，提供了丰富的可视化展示方式，包括快速灵活的客户端图表，拥有不同方式的可视化指标和日志的面板插件以及丰富的仪表盘插件，包括热力图、折线图、区域分布图等。

2.7 Fluentd、ElasticSearch、Kibana

Fluentd[54] 是一个日志处理器和转发器，从不同的来源采集多种数据，如度量和日志，用过滤器丰富它们，并将它们发送到多个目的地。由于 Fluentd

采用了非常多的 Kubernetes 元数据，来丰富自己的日志种类，所以能更好的与 Kubernetes 这样的容器化环境的结合。

ElasticSearch[55] 是一个实时、分布式、可扩展的搜索引擎，支持通过全文和结构化搜索进行分析，主要用于索引和搜索大量日志。ElasticSearch 通常与 Kibana[56] 一起组合使用，Kibana 是 ElasticSearch 的数据可视化前端和仪表板。Kibana 允许通过 Web 界面浏览 ElasticSearch 日志，并构建仪表盘和查询，从海量日志快速获取关键信息。

2.8 本章小结

Kubernetes 提供了大规模容器编排能力，使用户不必再关心底层服务器的运行细节，更多精力可以放在业务应用的研发上。在这基础上，Kubernetes 的数据卷技术让需要数据持久化的应用，有了数据落盘不丢失的保障。为了能一致可靠的处理数据、持久化数据，Kubernetes 的容器数据接口规范，使数据卷插件更加安全可靠。为了充分发挥云存储的优势，提高云存储的性能，JuiceFS 针对云原生做了特殊优化，采用元数据、压缩存储、多端共享等方式，将海量云存储的使用，变得像本地磁盘一样简单高效，让存算分离更加容易实现。Prometheus 的监控通知可视化组合、Fluentd 的日志收集分析组合，使系统的可观测性得以实现。

第三章 基于容器编排技术的通用数据处理技术上云策略

3.1 数据处理技术上云的困难

传统的数据处理技术，通常会运行在私有化机房的物理机上。为了满足高峰的系统负载，单机通常拥有极高的配置，以带来强大的计算能力；为了实现高可用，多台机器通常运行相同的服务，以满足单机的故障转移；为了实现高可靠，磁盘通常采用容错式磁盘阵列，以提供冗余存储的能力。这些策略虽然在一定程度上，解决了数据处理技术本地化的一些问题，但是不可否认其本身存在着资源严重浪费、可扩展性极低、维护难度居高不下的现象。随着云计算技术的成熟，逐渐有越来越多的企业和组织把私有化机房，迁移到了云上，享受云的弹性资源伸缩能力，减少私有化固定服务器带来的资源浪费。

新型的数据处理技术，往往在设计之初基于云考虑，可以很容易与云的能力结合。像 Spanner、TiDB 这类数据库基于共识算法 Raft[57] 、全局原子钟等，实现资源的动态调度与服务高可用的保障。不过这类数据库也还处于不断发展与进步阶段。就其本身所使用的 RaftStore IO 模型，还需要极大改进，才能解决大规模节点通信的问题；从传统数据库继承来的多版本并发控制存储模型，还需要针对分布式环境进行优化；由于需要维持高可用服务保障，运行系统所需要的资源普遍较多。

完全用新型的数据处理技术，替代传统的数据处理技术，还存在诸多不可行的因素，仅仅把传统数据处理技术的基础设施，更换成云资源，则无法充分利用云的弹性计算和存储能力。

于是结合云计算技术，对数据处理技术进行云原生改造，成为主要方向。如第一章绪论中列出的一些研究文献[40][41][42][43][44][45][46]，也在探索数据处理技术与云技术结合的方法。不过当前大多数研究，只针对单一技术进行改造，引入特定技术方法，导致研究所得的架构不能泛化；在存储上，使用了传统磁盘，导致水平扩展性能力较弱，使用了云存储，导致整体性能偏低；没有引入标准的可观测性方法，导致系统黑箱现象明显等问题。所以设计通用的架构模

型，充分融入云计算技术的优势，以适应更多的数据处理技术，简化整体构建运行流程，提高数据查询速度，增强并发处理能力，降低数据存储成本，就变得更加必要了。

3.2 数据处理技术上云的策略

为了充分发挥云的优势，需要借助云时代的容器编排技术，抛弃传统服务器运维管理的概念，把大数据应用当作一种容器化的服务，把云当作一种按需分配的服务资源，而不用考虑服务器应该怎么划分，以及每台服务器分多少份服务。

通过容器编排管理系统，统筹管理整个云服务资源。首先结合着资源动态调度策略，按需分配服务资源，使资源利用率达到最优。其次结合着资源弹性伸缩策略，判断服务器负载阀值，自动申请或释放更多资源，以达到削峰填谷。最后结合着存活探针、健康检查策略，保证服务失败时的自动容错与恢复。

通过把数据处理技术容器化，规避与传统服务器的重依赖关系，使其更轻量且易于被容器编排系统管理。然后结合容器数据卷挂载策略，分离存储与计算的紧密耦合，使其充分利用云的弹性计算能力。最后结合云存储、存储加速中间件、容器存储接口策略，解决数据高效持久化读写与访问难题，使其充分受益于云存储的大容量、低成本、高可靠等特性。

最终数据处理技术作为一种容器化的服务，在容器编排技术的基础上，可以不受服务器的限制，根据实际需要动态分配资源。分配较多的服务资源，以达到更高的性能。分配较少的服务资源，以降低成本。资源自动管理、服务动态调度和高可用保障，使服务可靠稳定。存储加速、海量云存储空间和标准化接口，使服务扩展灵活、降低成本、提高性能。

3.3 通用的云原生大数据架构

本文提出的“通用云原生大数据架构”，是一个用来帮助数据处理技术高效上云，充分利用云平台优势，提升自身存储计算能力，降低数据成本的架构模型。

通过 Kubernetes 容器编排技术，实现大数据应用在运行高效部署、维护和滚动升级。通过 JuiceFS+多种云存储技术，实现高性能、大容量、低成本的数据存储，同时结合 Kubernetes 数据卷挂载技术，实现存储与计算的完全分离。通过容器化部署大数据应用服务，结合负载均衡的动态网络请求，提供高并发访问，优化传统 MPP 计算模型的并发能力低、节点规模小、木桶效应明显的问题。

基于 Prometheus、AlertManager、Grafana 的通用监控策略、基于 Fluentd、ElasticSearch、Kibana 的通用日志处理、结合 Kubernetes 的健康检查机制、模块化低耦合设计，满足大数据技术上云后的可观测性。

如图 3-1 所示，整体架构采用模块化分层设计，满足低耦合高内聚的软件设计标准，总共有九大模块组成，各模块可以根据具体使用场景，替换为其它技术，满足通用型的要求。

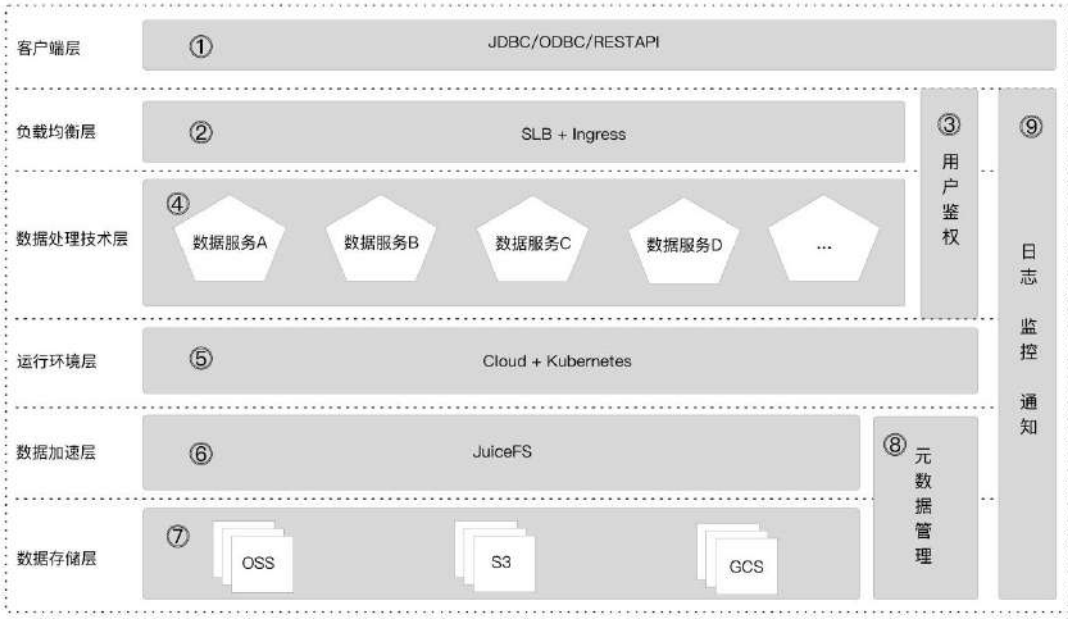

图 3-1：整体架构图

3.3.1 模块一：客户端

客户端通过标准的 JDBC(Java Data Base Connectivity Standard)、ODBC(Open Database Connectivity)、API(Application Programming Interface) 等方式接入系统。该架构不会修改客户端访问模式，对于用户或者应用来说，不需要做任何改变，通过现有的方式即可正常访问。

3.3.2 模块二：负载均衡

负载均衡层主要对客户端请求，做流量控制、流量分发、恶意访问拦截等工作。Ingress 是 Kubernetes 中的一个资源对象，用来管理集群外部访问集群内部服务的方式。可以通过 Ingress 资源来配置不同的转发规则，从而达到根据不同的规则设置访问集群内不同的 Service 后端 Pod 。

3.3.3 模块三：用户鉴权

用户鉴权，主要通过访问请求信息，判断当前用户否有权限访问资源。该模块，直接复用了大数据应用本身的权限模型，未做额外的修改。实际上客户端的访问请求，经过负载均衡层后，会直接到达大数据应用层做权限验证。

3.3.4 模块四：数据处理技术

数据处理技术以容器化的方式，部署在容器管理系统中，对外提供数据读写服务，对内使用相同数据源、相同运行策略。数据服务可以扩展无限多的副本，以提供强大的数据处理能力。

该模块默认分为两种服务，一种是支持数据写入的服务，一种是支持数据读取的服务。由于数据源是相同的一份，所以数据写入服务暂不能有多份，以避免写入冲突，数据读取服务可以有多份。所有数据服务通过配置服务存活检查探针，当节点不可用时自动删除旧 Pod 加载新 Pod ，满足不间断高可用服

务。挂载固定的 JuiceFS 数据目录，保证每次写入操作都在之前数据的基础上，实现存储与计算解耦。

多个相同服务的意义在于，能够使高并发请求转为低并发，提高整体性能。每一个数据服务可以处理一部分请求，当有多个服务时，请求会被均匀分流到每一个服务，原来的高并发请求，就被化解成低并发请求。云存储加速中间件，会自动缓存数据到容器服务中，所以每个服务都会存在高频数据的缓存。在多个服务共同工作的情况下，这是一个比较有效的优化策略（减少云存储的访问）。云存储依赖于网络，每台服务器的网卡能力是有限的，但是有多个服务之后，就能充分利用到多台服务器的网络资源。而且万兆网以及十万兆网的速度，通常可以达到单块磁盘的几倍甚至更多。

3.3.5 模块五：运行环境

运行环境是整个系统的基石，服务于容器化的大数据应用，提供简单易用、弹性灵活、稳定高效的运行时支持。云原生依托于云环境，云上有大量的虚拟机需要管理，Kubernetes 以其领先的大规模容器管理技术，从而被大多数企业和组织来管理云。使用 Kubernetes 作为运行环境，对整个架构有诸多益处。

首先在服务发现方面，使用域名或者 IP 地址开放容器网络访问。当进入容器的流量较高时，自动负载均衡并合理分配网络流量，从而使应用服务稳定运行。在存储编排方面，支持应用自动挂载所需要的存储系统，例如：本地磁盘阵列、公共云存储服务等。在自动部署和回滚方面，通过标识已部署容器服务的运行状态，能以稳定的操作速度，将实际运行状态更改为期望状态。例如，可以自动化为应用的部署创建新容器，删除现有容器，并将它们的所有资源用于新容器服务。

然后在资源分配方面，允许单独为每个容器，指定其所需要的 CPU 和内存资源。当容器设置了资源约束时，可以做出更好的决策，来管理整个集群的资源调度。在自动容错方面，支持重新启动失败的容器服务、替换容器、移除

不满足自定义运行状况检查的容器，并且在服务重新恢复成功之前，不将其开放给客户端调用。

最后在密钥与配置管理方面，支持存储和管理敏感信息，例如密码、OAuth(Open Authorization) 令牌和 SSH(Secure Shell) 密钥。可以在不重新创建容器镜像的情况下，部署和更新密钥，以及对应用程序进行配置，也无需在系统配置中暴露密钥。在扩展性方面，具体很高的可扩展性，这体现在整个架构，包括容器存储接口、自定义服务类型等等。使应用可以更好的扩展，使其更加适应实际场景。

3.3.6 模块六：数据加速

数据加速层，主要用于降低存储层与应用层的耦合，提高云存储的读写性能，发挥云存储大容量、低成本、高可靠的优势。比如 JuiceFS 这类云存储加速中间件，通常是兼容文件系统协议，模拟传统磁盘操作方式，对于使用者来说和用正常磁盘没有区别。

3.3.7 模块七：数据存储

云存储有近乎无限大的容量、低廉的成本、可靠的数据保障。大数据技术合理使用云存储，能很好的解决存储空间问题，极大地降低数据成本。随着万兆网和十万兆网的普及，云存储相比普通硬盘会更有优势。

3.3.8 模块八：元数据管理

数据加速模块为了提高云存储的访问速度，会把文件的描述信息存放成元数据。元数据的性能，极大的影响了实际的数据读写性能，所以选择一个合适的元数据管理数据库非常关键。

对于把 JuiceFS 作为数据加速层，可以选用的元数据管理数据库有很多种，比如：Redis、MySQL、PostgreSQL、SQLite、TiKV 等，不同数据库管理元数据性能会有差异。采用Redis作为元数据管理，可以达到较高的性能水平，

使访问延迟降到微秒级别。其它的数据库，则通常需要几百微妙甚至几十毫秒的延迟。采用 Redis 存放元数据，只占用较少的存储空间，用其他数据库会占用较多的空间。

3.3.9 模块九：日志、监控、通知

该模块主要满足大数据应用上云之后的可观测性。通过结合 Prometheus、AlertManager、Grafana 的技术，实现标准的监控通知可视化。通过结合 Fluentd、ElasticSearch、Kibana 的技术，实现标准的日志分析处理，满足异常问题的定位。通过 Kubernetes 的健康检查能力，对所有应用容器配置健康检查策略，及时排解应用服务异常问题。

3.3.10 各模块之间的交互关系

如图 3-2 所示，客户端通过 JDBC/ODBC/API 等方式进行服务访问时，首先经过云的网关，过滤掉非法访问、恶意攻击等请求。然后到达容器管理系统的负载均衡层。负载均衡根据用户的请求，决定访问大数据应用中的哪一类数据服务节点。数据写入有一个服务节点，写入类请求都会转发到该服务节点。数据读取有N个服务节点，所以会根据每个服务节点的繁忙情况来决定请求具体分配。无论是数据写入还是数据读取服务，本身并不存放数据，所有的数据访问都会通过数据加速层，由数据加速层根据元数据信息去云存储中获取数据。

数据服务、数据加速都处于 Prometheus 的监控、Fluentd 的日志收集、Kubernetes 的健康检查下。同时它们也归容器编排系统管理。Kubernetes、云存储、元数据都部署在云上。整个架构属于云原生设计，同时受益于模块化设计，可以灵活组合。

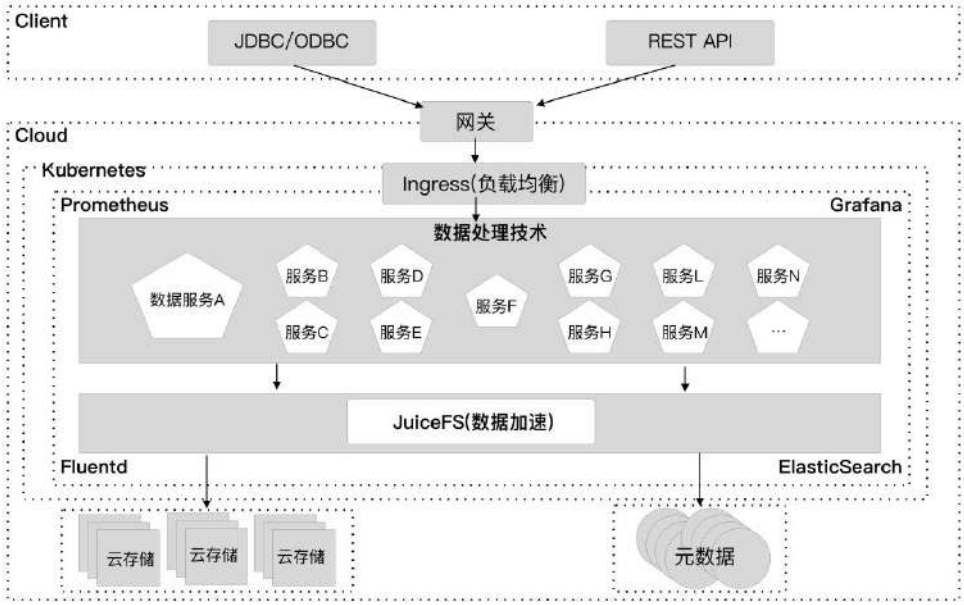

图 3-2：模块交互图

3.4 本章小结

通过分析传统数据处理技术运行环境的缺陷、云原生数据处理技术还存在的问题，引出了数据处理技术上云的必要性。为了解决传统数据处理技术的上云困难，以及上云后的运维困难，设计了一个通用的元原生大数据架构模型，提出了结合容器编排系统来实现快速上云、高效用云的策略。使用容器编排系统的意义在于，资源的灵活调度、弹性伸缩，服务的健康稳定运行，以及操作的简单性、维护的轻便性。数据处理技术容器化的意义在于，充分利用容器编排系统的管理能力，降低复杂度，提高灵活性，使大数据服务运行更高效。

第四章 基于云原生混合技术的存储加速以及存算分离方法

4.1 海量数据存储计算的困难

传统的大数据技术，在运行方式上，大致分为三种：单机、主从、集群。单机最为简单，一台服务器运行一个服务实例，性能不够，通过升级机器配置解决，同时这种方式的可用性、扩展性是最低的；主从有一主一从、一主多从的方式，主节点提供所有服务，从节点往往不直接提供服务，而是作为副本用于灾难恢复。

在后期的演化过程中，通过读写分离的策略，从节点也可以提供一定的服务能力。这种方式在一定程度上解决了服务可用性的问题，但是单机性能瓶颈的问题依旧存在。集群通过把数据分布到多台服务器，充分利用多台服务器的计算能力，在一定程度上缓解了单机性能瓶颈的问题。不过无论是主从还是集群，都没能很好解决资源横向扩展的问题。如果是主从，想加一个从节点，则需要搬移全部数据到从节点，数据过多时，时间成本非常高；如果是集群，想加一个节点，则需要对整个集群所有节点的数据进行重分布。由于服务器计算能力与存储能力发展不均衡，存储计算耦合还造成了资源的浪费。

新型的大数据技术，在设计之初就采用分层设计，计算层属于无状态服务，可以任意伸缩节点，存储层通过共识算法，采用多副本多节点保证高可用，节点数量和副本数量会在一定的策略下保持弹性。计算层经过网络接口与存储层进行交互，中间可能会通过中心节点获取元数据，也可能不通过中心节点而直接访问，这取决于具体设计。这种架构属于业界目前比较流行的设计，一个好的存储层设计能够为整体性能带来较大提升，为了保证整体架构完整可靠，该类系统通常相对复杂。

随着云技术的普及、数据量的激增、本地磁盘空间的捉襟见肘，业界逐渐发展出两种主要的主要存储解决方法。一种是采用分类存储（如图 4-1），按照实际业务场景，把数据划分成不同类型。对于事务型的数据，如交易，则放入普通数据库。对于文本型的数据，如日志，则放入全文检索类数据库。对于

文件类的数据，如图片，则放入网络文件系统。这种方式能解决一部分的数据存储问题，但是没能减轻海量事务型数据存储的压力，同时提高系统接入的复杂性。

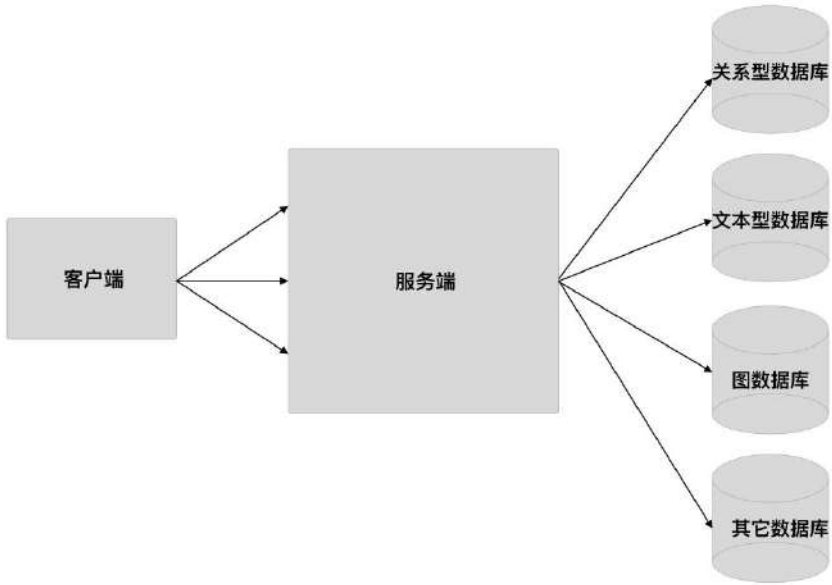

图 4-1：海量数据分类存储

另一种是采用冷温热分层存储（如图 4-2），热数据会使用高性能的本地磁盘，温数据会使用普通性能的本地磁盘，冷数据会放到云存储上。在业务逻辑层面划分数据的使用，如果查询几个月前的数据，那就慢速查询云存储，如果查询最新的数据，那就访问本地高速磁盘。然后设置定时任务，定期把数据从热存储搬移到温存储，在搬移到冷存储。这种方式一定程度上解决了问题，不过使用起来比较繁琐，而且限制了业务想象力，导致跨层的分析计算十分困难。

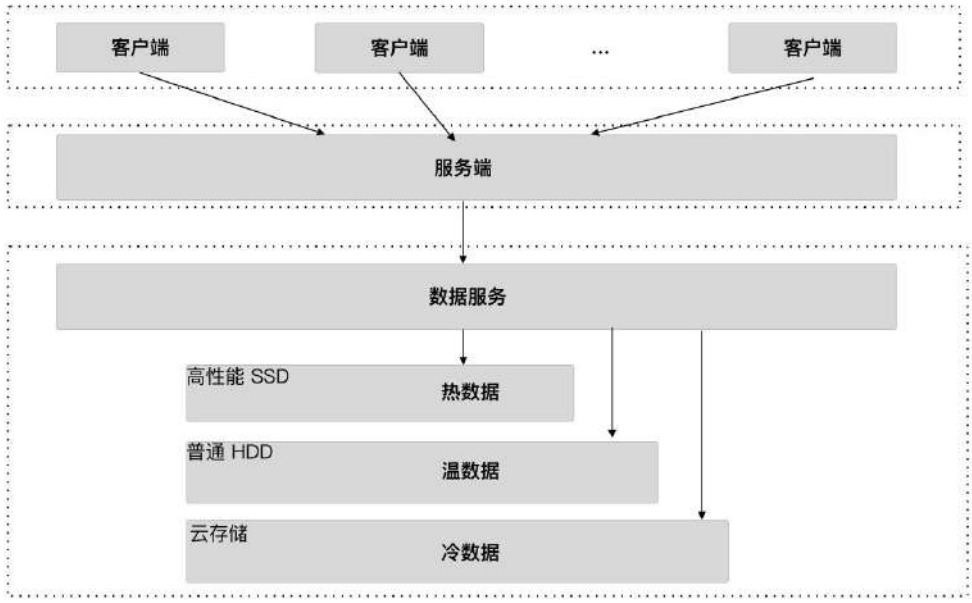

图 4-2：海量数据分层存储

4.2 云存储的优势与劣势

对于海量数据的存储，通常会把使用频率低的数据，或者把重要性低的数据放入云存储，其他数据放入高性能磁盘。造成这种情况的主要原因是，云存储拥有存放大规模数据的能力，但是其数据读写效率非常低，一般只能达到传统磁盘读写速度的十分之一。

如图 4-3 所示，云存储的整体架构通常分为四层。从下往上，第一层为存储单元层，用于存放具体数据。存储阵列设备由多块物理磁盘组成，采用 RAID(Redundant Array of Independent Disks) 技术，实现比单块磁盘更快的读写能力，同时满足一定的数据容灾备份能力。存储阵列设备通常会放置在多个数据中心，以满足海量的存储空间，以及跨区域的冗余备份能力。第二层为存储索引层，主要用于将众多存储单元，通过高速光纤网络连接在一起，以达到数据之间的快速、高效、可靠传输。第三层为数据分发计算层。对请求存储的数据，进行合理分割、计算，使其存储到合适的存储单元。对于请求查询的数据，从对应存储单元并行获取，合并加工之后返回。第四层为数据服务层，提供通用网络访问接口，供客户端存储、获取数据。

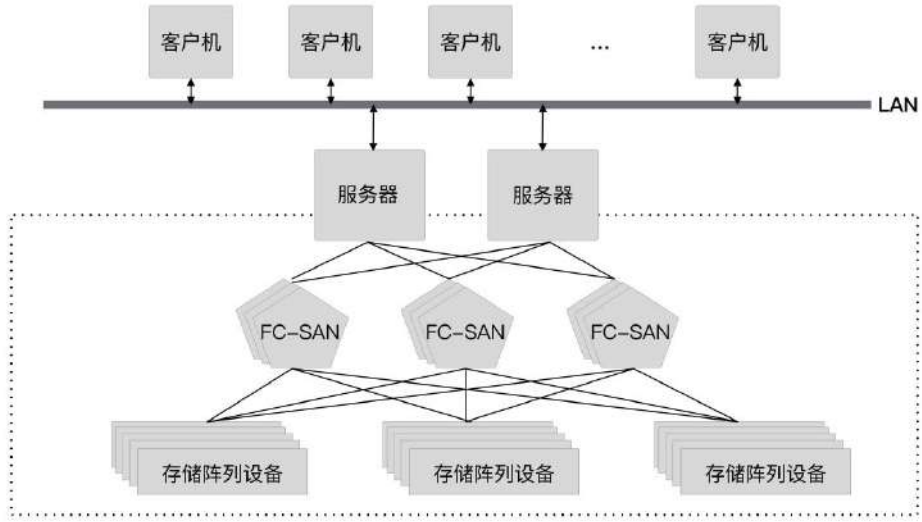

图 4-3：云存储整体架构

从云存储整体架构可以看出，优点：云存储拥有大量的存储单元，所以能提供海量的数据存储能力。云存储拥有较多冗余备份，所以能提供更加安全可靠的数据容灾能力。云存储提供了统一访问接口，所以能提供客户端便捷的访问使用。缺点：数据的存储和访问经过了多个设备链路，导致云存储的读写吞吐能力，会低于传统磁盘。因为传统磁盘的数据读写，只用经过内部的 I/O 总线，数据传输距离较近。

除此之外，云存储还分为公有云、私有云、混合云。不同的类型，对应不同的实际场景，并不能严格划分孰优孰劣。比如，公有云依托于各大云厂商的技术实力，整体上能做到比私有云更高的可靠性。但是从反面分析，公有云数据在公网传输，更容易遭遇网络攻击，所以其安全性并不一定高于私有云。如图 4-4 ，从 Google Trends 可以看出，近一年来，公有云和私有云的热度差别不大，略高于混合云。本文提出的通用架构，可以兼容所有类型的存储，根据实际需要，引入对应云存储即可，所以此处不延伸探讨云存储的差异问题。

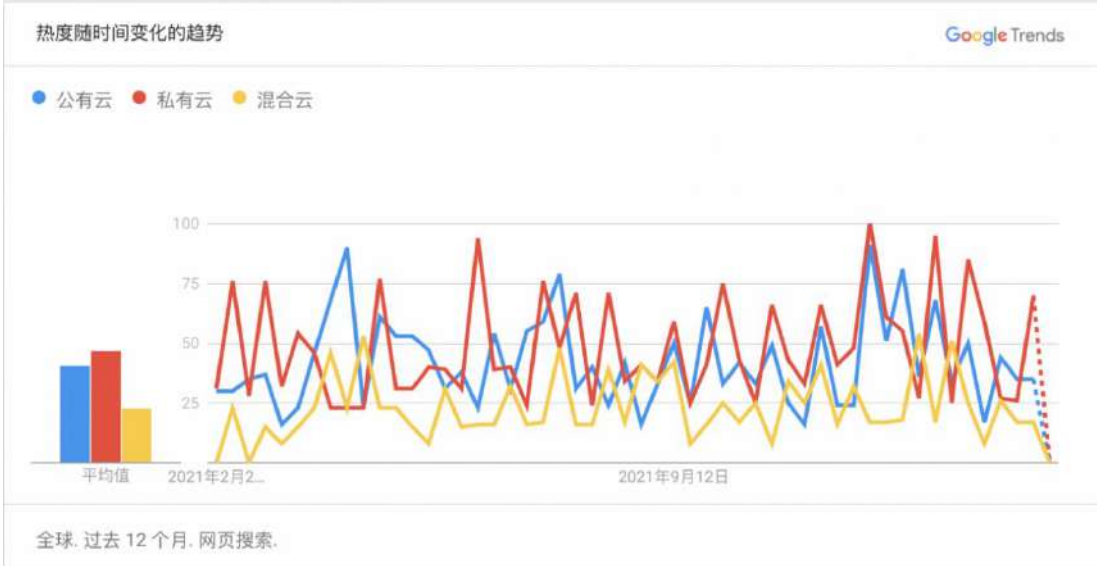

图 4-4：不同云存储类型的热度趋势[58]

4.3 存储优化问题分析

从 4.1 节，海量数据存储计算的困难中，可以看出传统数据处理技术，受限于物理磁盘的机械性质，很难容纳较多的数据，受限于系统架构的设计，很难水平扩容存储容量。导致，整体数据处理能力较低，整体系统架构过于复杂，使用与维护很困难。从 4.2 节，云存储的优势和劣势中，可以看出云存储有着超大容量的存储空间、高可靠的数据安全性，不过由于架构设计的原因，整体的读写能力受限。

基于这些问题，分析之后，可以总结为两个主要问题：1. 如何提供易于扩展的大容量存储？2. 如何使该存储达到足够性能？

针对问题1，满足易于扩展的性质，从技术发展程度来看，只能选择云存储。传统物理磁盘，受限于物理性质无法做到水平扩容。易于扩容的存储作为基础设施，数据处理技术才能在其之上，达到易于扩容的能力。也就是实现存储计算分离技术，达到存储能力的水平扩容，在下一小节介绍。

针对问题2，决定数据传输性能的本质原因，无非有两个方面，距离和大小。通过缩短数据传输的距离，可以提高数据传输的速度。通过减小数据传输的大小，可以提高数据传输的速度。云存储的特点，就是需要连接众多数据单

元，以满足几乎无限的存储容量。所以，云存储本身所需要的传输距离很难被压缩。而且因为，云存储通常由云厂商进行技术维护，很难直接对云存储技术进行改动。文献[59]提出一种针对 Amazon S3 云存储优化读写性能的策略(S3FS)，如图 4-5 所示。在云存储之上再加入一层技术，通过异步缓存机制，把频繁使用的数据缓存在内存，或者是磁盘，间接缩短数据传输距离。并行处理读写请求，提高整体效率，从而减轻云存储广域网延迟，造成的性能损失。

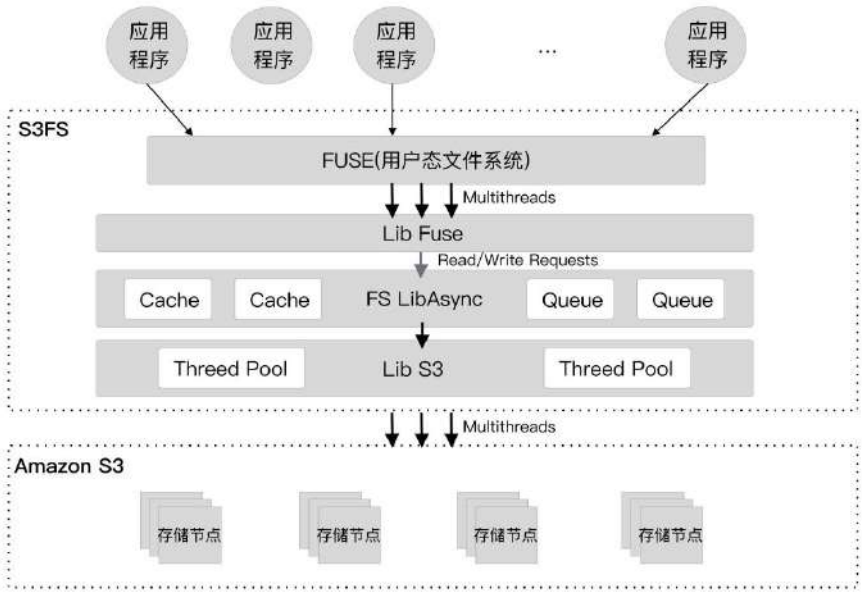

图 4-5：S3FS 技术优化云存储的流程

上面这种优化策略，提高了部分读写能力，不过存在需要完善的地方。首先，S3FS 只针对 Amazon S3 优化，不能扩展到其它云存储，限制了通用能力。其次，S3FS 只利用了并行处理和缓存，并没有减少数据传输的总大小，对于大数据量场景的提高并不太明显。最后，S3FS 基于 Python 语言实现，Python 的性能较低、稳定性欠佳，导致整体优化性能受限。

4.4 云存储优化加速策略

受 S3FS 的启发，同时为了规避 S3FS 这些问题，带来更高的性能提升。工业界的开源技术 JuiceFS[48]，基于 POSIX 接口协议，通过高性能的 Go 开发语言，引入元数据概念，使用缓存、并行处理、数据分片、压缩等技术，实现

了支持多种平台，支持多种云存储，减小了数据传输总大小，大幅度提高了云存储的读写能力，达到甚至超过传统磁盘的读写速度。（整体架构可参见 2.3 小节）

如图 4-6 所示，能够加速云存储的核心原理在于，当客户端向云存储写入数据时，首先经过 JuiceFS 优化加速中间件。该中间件会利用服务器的计算能力，通过一些算法，提取原始数据的文件信息，把文件信息作为元数据存放到云数据存储中，对原始文件进行并行分块、并行压缩、然后并行上传到云存储。对于经常频繁访问的数据，会缓存在当前服务器的内存和磁盘中。

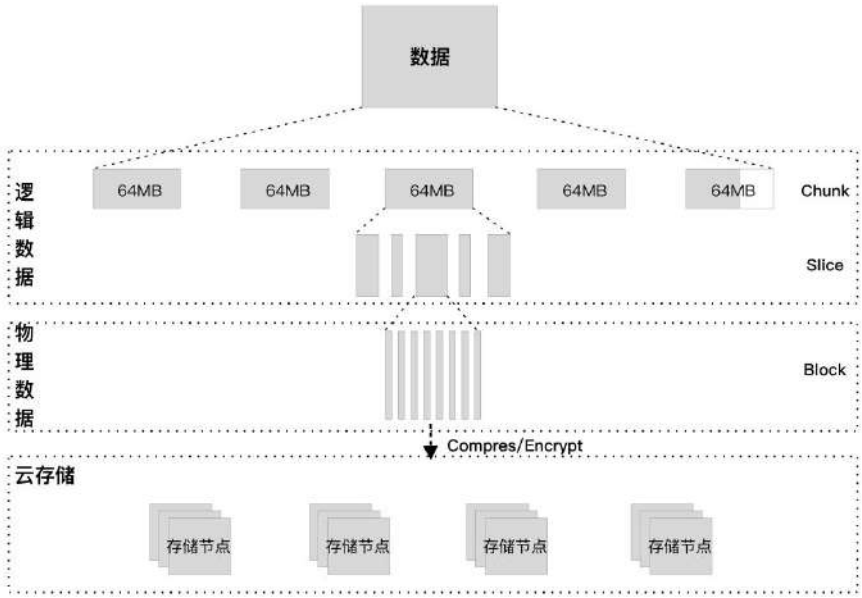

图 4-6：数据加速之数据分片、压缩

如图 4-7 所示，当用户从云存储中读取数据时，该中间件会先访问元数据数据库，获得文件描述信息，然后去云存储中并行下载数据。元数据通常很小（100GB 真实数据，产生 20MB 左右元数据），而且原数据被切成多块，每一块都放在不同位置（打乱后上传到云存储，避免热点现象），能够充分利用并行技术，所以通常会比直接访问云存储快十倍到百倍。除此之外，这类中间件还会把经常访问的数据，存放到当前的容器的内存以及磁盘，这对于数据加速非常有效的。

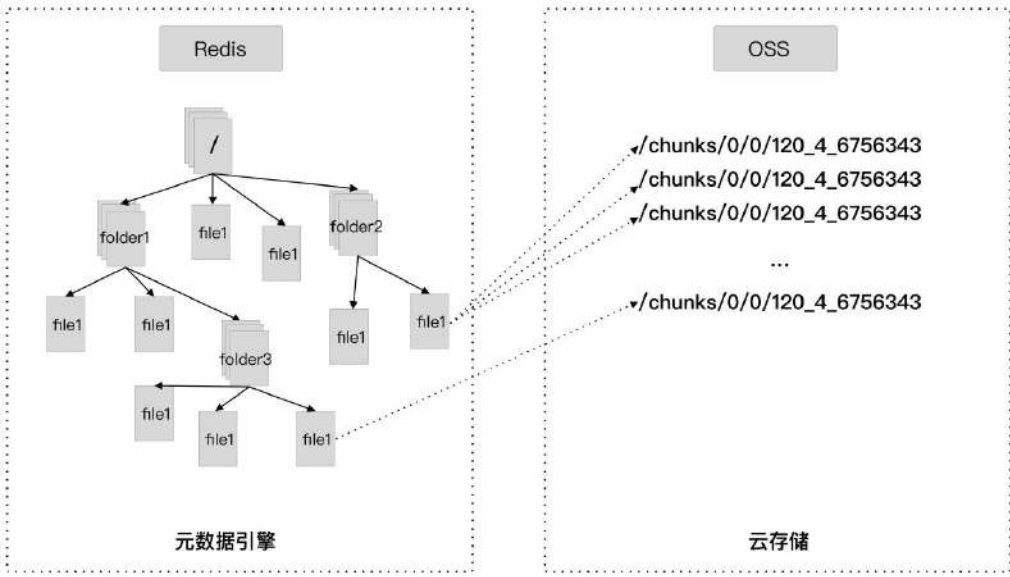

图 4-7：数据加速之提取元数据、分散访问热点

JuiceFS 的优化加速策略，改善了 S3FS 的缺点，带来了更高的性能提升。通过基准测试，如图 15 所示，数据写入速度可以达到 627 MB/s，数据读取速度可以达到 752 MB/s 。普通的机械硬盘，读写能力在 100~300 MB/s，高性能的固态硬盘，读写能力可以达到 500~1000MB。那么经过 JuiceFS 加速的云存储，整体性能已经超过传统磁盘，更接近于高性能磁盘。

```
$ ./juicefs bench -p 4 /mnt/jfs/
  WriteBig 4096 / 4096 [==============================================================]  done
   ReadBig 4096 / 4096 [==============================================================]  done
WriteSmall 400 / 400 [================================================================]  done
 ReadSmall 400 / 400 [================================================================]  done
      Stat 400 / 400 [================================================================]  done
Benchmark finished!
BlockSize: 1 MiB, BigFileSize: 1024 MiB, SmallFileSize: 128 KiB, SmallFileCount: 100, NumThreads: 4
Time used: 17.3 s, CPU: 203.3%, Memory: 796.8 MiB
+------------------+------------------+--------------+
|       ITEM       |       VALUE      |     COST     |
+------------------+------------------+--------------+
|   Write big file |     627.79 MiB/s |   6.52 s/file |
|    Read big file |     752.91 MiB/s |   5.44 s/file |
| Write small file |     98.9 files/s | 40.44 ms/file |
|  Read small file |    822.7 files/s |  4.86 ms/file |
|        Stat file |  25602.0 files/s |  0.16 ms/file |
|   FUSE operation | 71252 operations |    0.43 ms/op |
|      Update meta |  1677 operations |    1.69 ms/op |
|       Put object |  1424 operations |   99.67 ms/op |
|       Get object |  1101 operations |   62.05 ms/op |
|    Delete object |    14 operations |   18.20 ms/op |
| Write into cache |  1059 operations |    5.15 ms/op |
|  Read from cache |   323 operations |    0.05 ms/op |
+------------------+------------------+--------------+
```

图 4-8：JuiceFS 基准测试表现

如图 4-9 所示，在整体的读写吞吐效率上，相对于 S3FS 和其它同类型技术，有 10 倍以上的提升。如图 4-10 所示，在元函数调用上，相对于 S3FS 和其它同类型技术，有 15 倍左右的提升。由此可见，使用 JuiceFS 作为云存储加速技术，是较为合理的策略。

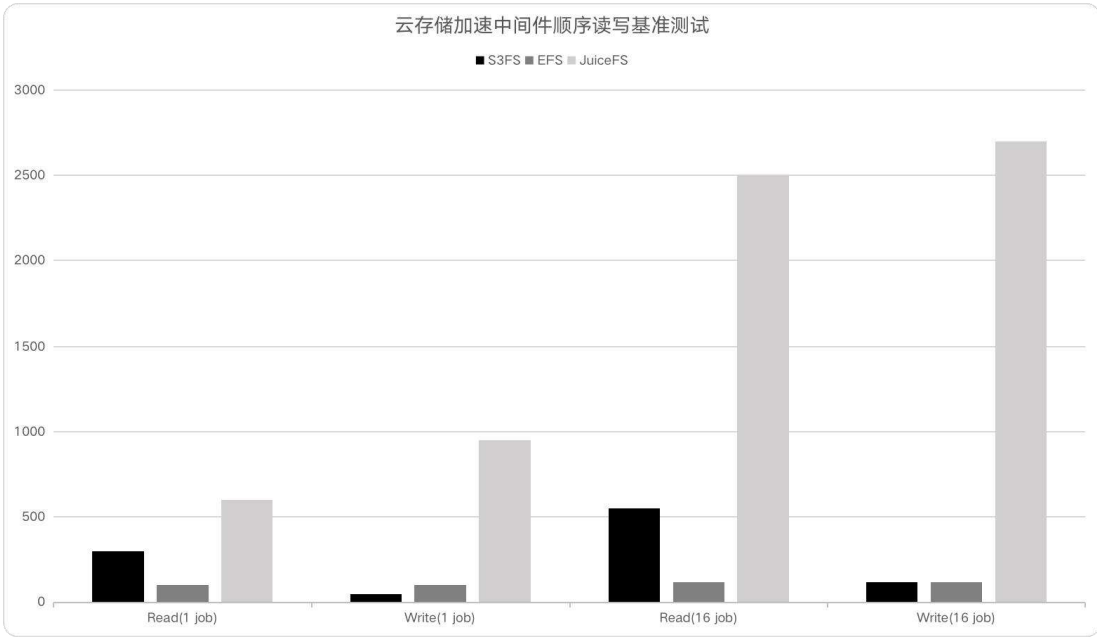

图 4-9：云存储加速中间件顺序读写基准测试

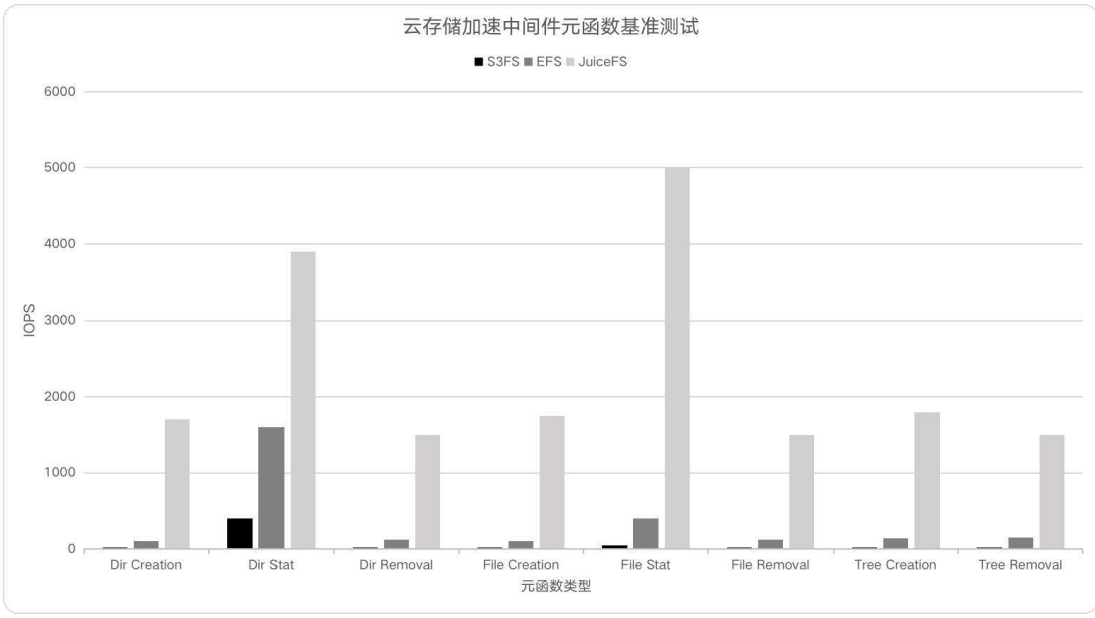

图 4-10：云存储加速中间件元函数基准测试

4.5 存储计算分离策略

对于 4.3 提出的问题：1. 如何提供易于扩展的大容量存储？2. 如何使该存储达到足够性能？通过 4.4 的云存储优化加速策略，已经解决了一部分。加速后的云存储已经超过传统磁盘的读写吞吐能力，接近于高性能磁盘的读写吞吐能力。同时由于，云存储本身拥有近乎无限的存储容量，整体灵活的数据访问，可以满足易于扩展的特性。

但是如果直接在数据处理技术，与磁盘之间加入 JuiceFS 中间件，将导致应用服务运行和维护复杂度升高。原因是，JuiceFS 正常运行时，将会在服务器中启动几个服务线程，用于数据压缩、分片等操作。对于新增的服务进程，则必须需要考虑容错、恢复等操作，如果 JuiceFS 出现异常，数据处理服务也将异常，从而影响整体服务可靠性。

在本文提出的通用元原生大数据架构上，基于大规模容器编排技术，首先容器化了数据处理技术，使其更轻量化易于服务调度，从而能够充分利用云的计算能力。然后，使用 JuiceFS 对云存储进行优化加速，使其满足正常的数据读写吞吐效率，从而为通用架构提供弹性海量的存储能力。

为了使容器编排技术、数据处理技术、云存储和存储加速技术，这四者更好的结合，需要实现存储与计算分离，来降低技术连接的复杂性。通过借助容器编排系统的标准存储接口 CSI 和数据卷挂载技术 PV/PVC(Persistent Volume Claim)，可以实现存储分离，从而实现四者之间的高效协作。

如图 4-11 所示， Node 对应云上的单台服务器资源，DaemonSet Pod 对应云上的数据处理服务，StatefulSet Pod 对应数据存储服务。JuiceFS 云存储加速技术，可以通过实现 CSI 协议，以功能插件[60]的形式成为数据存储服务，并按照 StatefulSet Pod 的服务形式运行。通过集成 JuiceFS 功能到容器编排系统中，JuiceFS 服务将被自动调度运行在每一个服务器节点中，对于服务中断、异常等现象，容器编排服务会自动容错恢复，保证服务持续正常运行。JuiceFS 服务将按照配置参数，进行云数据库的连接与访问。

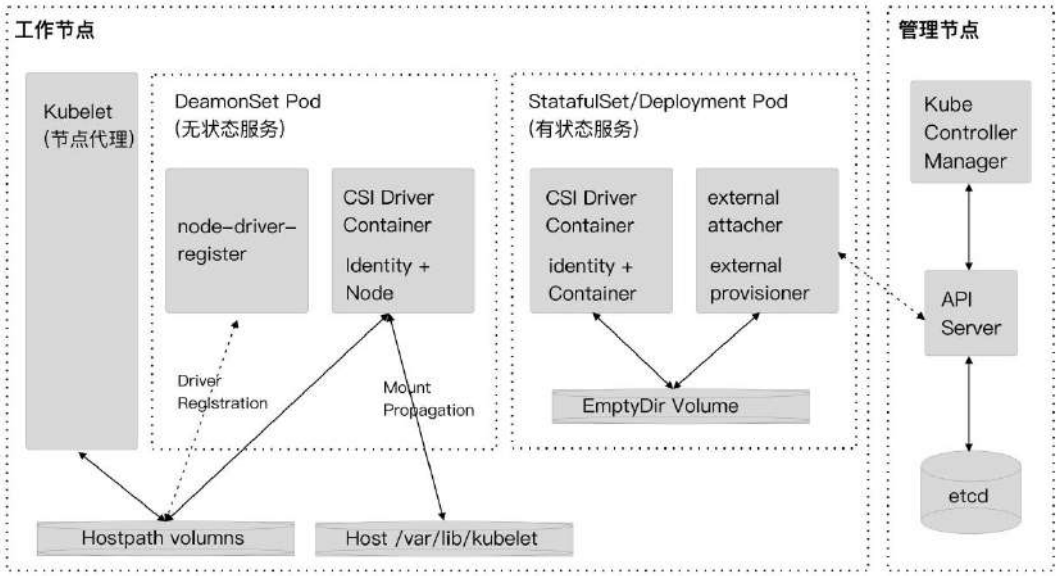

图 4-11：云环境下的容器存储接口工作流程

容器编排系统的 CSI 能力，为 JuiceFS 的服务高可用提供了保障。通过把 CSI、JuiceFS和云存储三者连接，在整个通用系统中，已经存在了高可用、高性能、简单灵活的海量存储。接着，如图 4-12 所示，首先通过容器编排系统的 PV 技术，连接到系统中已经存在的海量存储。然后通过 PVC 技术连接到 PV。最后数据处理技术的服务连接到 PVC。

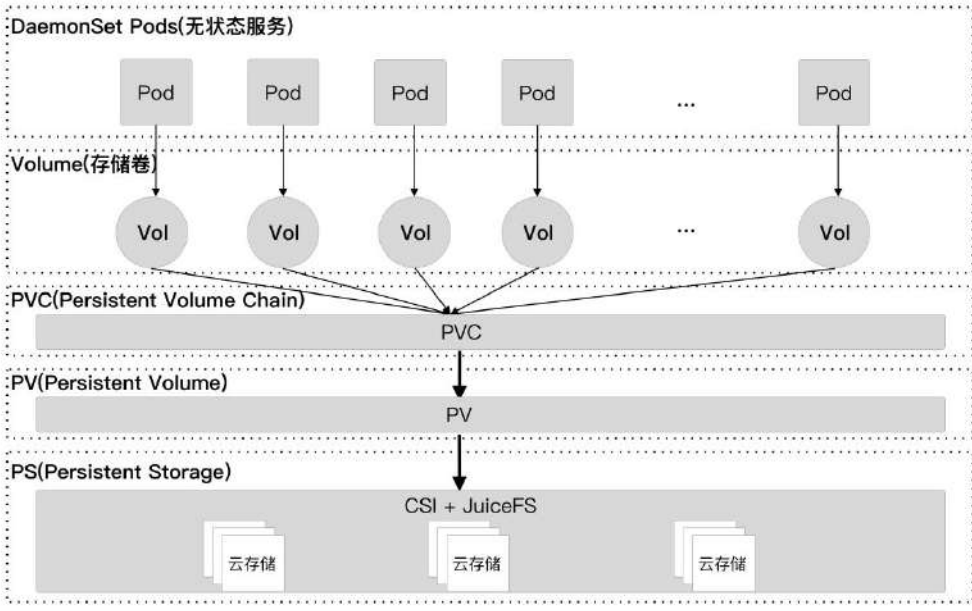

图 4-12：数据处理技术服务与云存储加速服务的连接流程

在整个集群中，只需要初始化一个 PVC 和 PV，所有数据处理技术容器连接相同的 PVC，从而实现计算与存储的分离，实现容器编排技术、数据处理技术、云存储和存储加速技术的联通。

整体上，通过分层架构设计、引入数据加速层、借助数据卷挂载技术，存储与计算分离只需要在 Kubernetes 中声明一个持久化存储卷，该存储卷会访问相同的元数据、访问相同的云存储，大数据应用容器统一挂载该存储卷，数据变动都会通过该存储卷传递到所有容器服务。

当有容器服务异常，Kubernetes 自动移除异常容器，创建新容器，即可保证服务不间断。当现有服务计算能力不足，则Kubernetes 创建更多新容器，即可提升服务计算能力。当现有服务计算资源冗余，则 Kubernetes 自动移除部分容器，即可降低成本自动缩容资源规模。

由于数据源存储在统一的云存储，而且云存储内部会将数据多副本保存以满足高可用，提高计算能力只用简单挂载数据源，所以通过这种方式既能接近新型大数据技术的优良设计，又能规避高可用存储对系统带来的复杂性，还可以增强系统的弹性伸缩能力、充分发挥云的优势、降低数据存储成本。

4.6 本章小结

传统大数据技术受限于存储计算的耦合，很难实现水平扩展，而且对服务器的资源利用也比较低效。新型大数据技术，为了实现存储计算分离，为系统带来了较高的复杂性。立足于云，通过把 JuiceFS 与云存储融合，使得云存储的读写能超过传统磁盘，接近于高性能磁盘，为我们带来了高性能、高可靠、易扩张的海量存储能力。通过把 Kubernetes、JuiceFS 和云存储融合，为我们带来了简单高效的存储计算分离设计，为云上大数据技术带来更多赋能。存储与计算的完全分离，真正使大数据技术变得灵活，从而能更有效的受益云近乎无限的存储能力、计算能力，受益容器编排系统的动态调度、弹性伸缩、自动容错等特性。

第五章 基于容器化节点的大规模云上并行计算策略

5.1 传统大规模并行计算的问题

早期 MPP(Massively Parallel Processor) 的代表性技术是 Greenplum，它构建于 PostgreSQL[61] 技术之上。通过在多台服务器部署 PostgreSQL 实例，按照 Segment 划分数据，构成一主多从的集群。计算任务由主节点分发到从节点，从节点上报计算后的结果，主节点收集结果并二次处理后返回给计算请求发送方。这种计算策略，充分展现 MPP 模型大规模并行计算的特点，任务分发到所有节点，利用所有节点的资源完成计算，再由主节点处理上报。由于主节点既负责任务分发、汇总，又负责任务计算，所以主节点容易成为性能瓶颈；每一个计算任务都需要集群所有节点共同处理，受限于单节点计算能力，所以不支持高并发场景。而且，Greenplum 是对 PostgreSQL 的包装，PostgreSQL 本身 OLTP 型数据库，数据处理相对 OLAP 较弱。

新型 MPP 的代表性技术是 ClickHouse，它吸收了众多 OLAP 技术的优点，列式压缩存储、向量化计算引擎、代码动态编译等，同时基于新版 C++ 编程语言做了大量优化，整体性能非常优异。集群采用多主架构，每一个节点既可以作为主节点，也可以作为从节点。计算任务可以由任意节点接收，收到任务的节点作为当前阶段的主节点，并把任务分发到其它节点，其他节点上报计算结果，当前节点二次处理并返回任务请求方。通过多主的方式消除了单主的性能瓶颈问题，同时借助许多优化策略，极大提高了数据处理效率，把 MPP 的特性发挥的淋漓尽致。

综上所述，MPP 计算模型有其天生的缺陷，首先需要所有节点共同完成计算，单节点的计算能力有限，所以不支持任务并发量大的场景；其次任务计算的时间，取决于集群中计算最慢的节点，短板效应明显；再次需要收集节点的计算结果，进行二次加工处理，导致集群间网络开销极高；最终随着集群规模增大，节点故障率也会随之增高，集群可用性就会急剧下降。

5.2 基于云的大规模并行计算策略

本文提出的 CMPP(Cloud based Massively Parallel Processing) 计算策略，属于“通用云原生架构”的一部分，依赖于云以及存储计算分离策略。主要逻辑是：当接到大量并发计算任务时，会均匀分配到所有计算节点，每个节点独立处理收到的任务，完成计算任务后直接返回结果。和单机的区别在于，有多个计算节点，数据完全一致，可以承担大量并发计算任务，而不会受限于单机处理器和磁盘性能瓶颈。和 MPP 的区别在于，每个节点都为主节点，且能够独立完成计算任务不需要其他节点协作，而不会受限于短板效应、二次数据计算等问题。

CMPP 的优势在于，由于存储和计算完全分离，结合容器化以及容器编排技术，计算能力的水平扩张得到了解决。由于云存储+数据加速中间件的组合能力，数据容量、高可用、性能也得到了很好的解决。由于在服务器上磁盘的性能瓶颈明显与处理器，通过把相同服务部署到多个节点（挂载同一份云存储数据），每个节点可以独立完成整个计算任务。

当有大量计算任务，通过负载均衡策略均匀分配任务到空闲节点，每个节点充分发挥计算能力的优势，完成所有计算任务。在这个过程中，消除了大量节点互相通信带来的延迟、消除了短板效应以及主节点二次计算带来的延迟。并且受益于万兆以及云存储加速插件，云存储的并发读写能力优于传统磁盘性能，带来整体性能的增强。

5.3 两种大规模并行计算策略的分析对比

通过抽象 MPP 和 CMPP 的计算处理过程，引入计算公式，评估两者之间的差异。

假设有 N 台服务器，内网带宽平均速度为e、最大速度为E。磁盘平均速度为f、最高速度为F，云存储平均速度为cf、最高速度为CF(受限于内网带宽)，CPU计算速度为S。当查询计算数据量为M时，中间结果数据量为

m(M*0.2)、计算结果总数据量为r(m*0.1)。当并发数量C逐渐增大时，网络带宽、磁盘速度会被耗尽。

令 e 为100MB/s，E 为1000MB/s，f 为150MB/s，F 为150MB/s，cf 为80MB/s，CF 为max(E, 2000MB/s) 按25GBps/s计算，并减去了一些传输损耗，S 为100MB/s(8c16gb服务器)，M 为100MB，m 为20MB，r 为2MB。忽略网络延迟、上下文切换、性能抖动等问题，以便于对比不同并发下MPP架构和CMPP架构的查询耗时T。（上述指标数据采集自云厂商文档和实验所得）

在 MPP 架构下，用户发起一次查询请求，会到达其中 1 个计算节点(假设为A)，然后 A 节点，转发查询请求到其它 N-1 个节点。 N 个节点同时计算 M/N 大小的数据量，然后 N-1 个节点上传 m 大小的中间结果集到A节点，A节点计算 m 大小的中间数据，最终返回给客户端。

阶段一公式:

$$\begin{aligned} T1(\text{查询耗时}) = {} & M(\text{总计算数据量}) / N(\text{服务器数量}) \\ & / \min(f(\text{磁盘平均速度}), F(\text{磁盘最高速度}) \\ & / C(\text{并发数量})) + M(\text{总计算数据量}) \\ & / N(\text{服务器数量}) / S(\text{CPU计算速度}) \end{aligned} \tag{5.1}$$

阶段二公式:

$$\begin{aligned} T2(\text{查询耗时}) = {} & m(\text{中间结果数据量}) / N(\text{服务器数量}) \\ & / \min(e(\text{网络平均速度}), E(\text{网络最高速度}) \\ & / C(\text{并发数量})) \end{aligned} \tag{5.2}$$

阶段三公式:

$$\begin{aligned} T3(\text{查询耗时}) = {} & m(\text{中间结果数据量}) \\ & / \min(f(\text{磁盘平均速度}), F(\text{磁盘最高速度}) \\ & / C(\text{并发数量})) \\ & + m(\text{中间结果数据量}) / S(\text{CPU计算速度}) \end{aligned} \tag{5.3}$$

总公式：

$$\begin{aligned} T(\text{查询耗时}) &= T1(\text{查询耗时}) + T2(\text{查询耗时}) \\ &+ T3(\text{查询耗时}) \\ &= (M / N / \min(f, F / C) + M / N / S) \\ &+ (m / N / \min(e, E / C) + (m / \min(f, F / C) \\ &+ m / S) \end{aligned} \tag{5.4}$$

在 CMPP 架构下，用户发起一次查询请求，会到达其中一个计算节点，该计算节点直接处理计算任务，并返回结果给客户端。

阶段一公式：

$$\begin{aligned} T1(\text{查询耗时}) &= M(\text{总计算数据量}) \\ &/ \min(cf(\text{云存储平均速度}), CF(\text{云存储最高速度}) \\ &/ \max(1, C(\text{并发数量}) \\ &/ N(\text{服务器数量}))) + M(\text{总计算数据量}) \\ &/ S(\text{CPU计算速度}) \end{aligned} \tag{5.5}$$

总公式：

$$\begin{aligned} T(\text{查询耗时}) &= T1(\text{查询耗时}) \\ &= M / \min(cf, CF / \max(1, C / N)) + M / S \end{aligned} \tag{5.6}$$

如图 5-1 所示，按照上述公式计算了 MPP 和 CMPP 在 5 台服务器下，随着并发数量的增加，查询响应时间的变化趋势。可以看出当并发较低时，MPP 查询速度快于 CMPP。当并发越来越高时，CMPP 查询速度逐渐快于 MPP，而且 CMPP 查询时间相对稳定，MPP 查询时间在急剧上升。

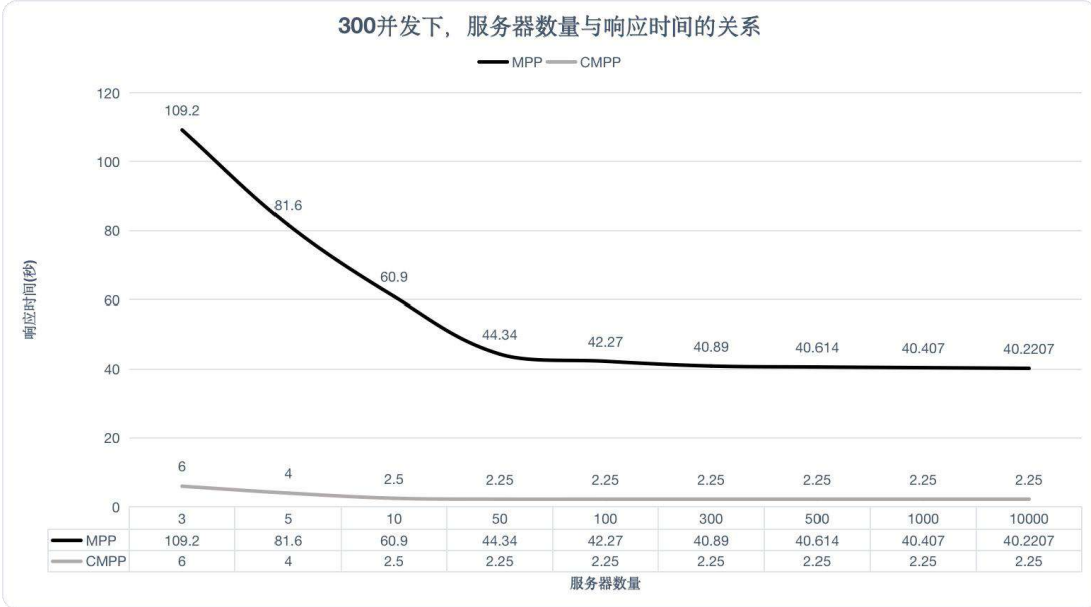

图 5-1：5台服务器下，并发数量与响应时间的关系

如图 5-2 所示，按照上述公式计算了 MPP 和 CMPP 在 300 并发下，随着服务器数量的增加，查询响应时间的变化趋势。可以看出在高并发下，即使服务器数量不断增加，MPP 的响应时间也不能变得更快，而且远慢与 CMPP 的响应时间。

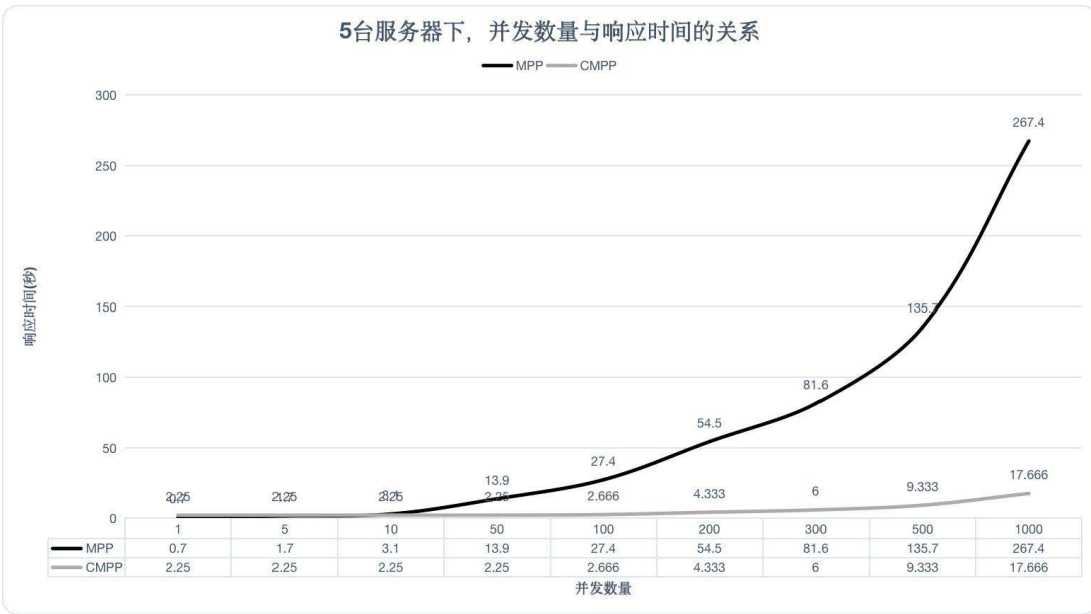

图 5-2：300并发下，服务器数量与响应时间的关系

5.4 本章小结

通过分析 MPP 技术的数据处理流程，阐述其存在的问题，提出 CMPP 技术，基于云和存储计算分离的优势，采用并行独立计算，消除了大量节点互相通信带来的延迟、消除了木桶效应以及主节点二次计算带来的延迟。最后，通过抽象 MPP 和 CMPP 的计算处理过程，引入计算公式，评估两者的差异。可以发现当并发越来越高时，CMPP 查询速度逐渐快于 MPP，而且 CMPP 查询时间相对稳定，MPP 查询时间在急剧上升。在高并发下，即使服务器数量不断增加，MPP 的响应时间也无法更快，而且远慢于 CMPP 的响应时间。

第六章 融合日志、度量、追踪技术的标准化可观测性设计

6.1 可观测性的定义

Google 根据内部多年的分布式系统架构实践，提出了著名的 SRE(Site Reliability Engineering)[62] 体系，把跟踪、度量、日志和健康状况作为高可用系统架构的关键因素。并指出可观测性的基本理念，如果仅使用系统的输出，可以在有限的时间段内，确定系统的当前状态，则该系统是可观察的。对于这样的系统，系统的所有行为和活动，都可以根据系统的输出进行评估。相反，其输出传感器提供的数据或信息不足，无法让操作员确定系统行为的系统，将被视为不可观测的。这些基本理念，为可观测性奠定了坚实的理论基础。

在 2017 年的分布式追踪峰会，Peter Bourgon 的研究文献《Metrics, Tracing, and Logging》[63] 系统地阐述了，度量、追踪和日志这三者的定义、特征，以及它们之间的关系与差异，如图 6-1 所示。随着分布式技术、云计算技术的发展，事件日志、链路追踪和聚合度量，逐渐成为了可观测性的三个主要纬度。

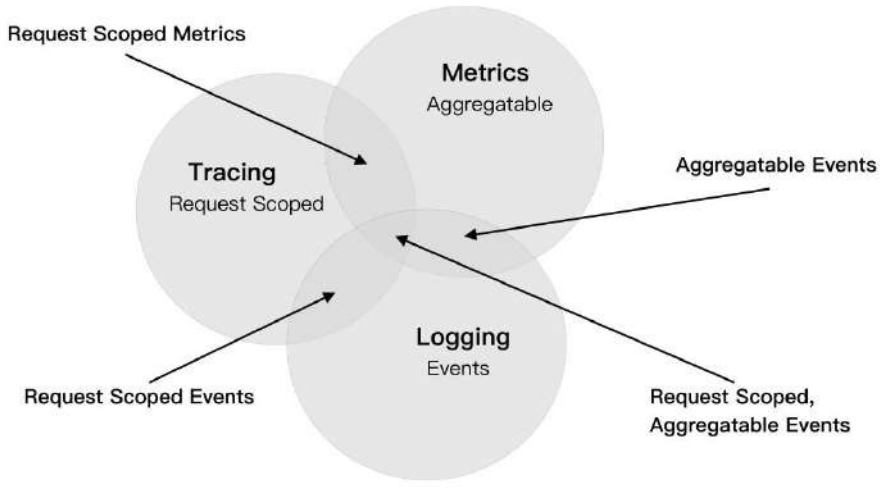

图 6-1：可观测性的定义

事件日志。日志主要用于记录离散性的事件，通过收集这些事件记录，可以用来分析程序的行为。

链路追踪。在单机系统时代，链路追踪的范围，只局限于堆栈追踪。比如：调试程序时，在程序开发工具中加入断点，看到的调用栈视图上的内容便是追踪。在微服务时代，追踪不只局限于调用栈，对于一个外部请求，则需要内部若干服务的联动响应，那么完整的调用轨迹将跨越多个服务，同时包括服务间的网络传输信息，与各个服务内部的调用堆栈信息。

聚合度量。聚合度量是指对系统中某一类信息的统计聚合。对于软件系统而言，比如高峰的请求数量、内存的使用情况、网络的吞吐量、磁盘空间、CPU 的利用率等，都可以看作一种度量指标信息。度量的主要目的是监控和预警，比如当某些度量指标达到风险阈值时，触发报警事件，以便自动处理或者提醒程序管理者介入。

6.2 可观测性的重要性

随着云计算技术、分布式技术的快速发展，可观测性逐渐成为，高可用系统不可缺少的部分。如图 6-2 所示，根据 Google Trends 的趋势分析，可以看出伴随这云计算的普及，可观测性的热度在不断上升。

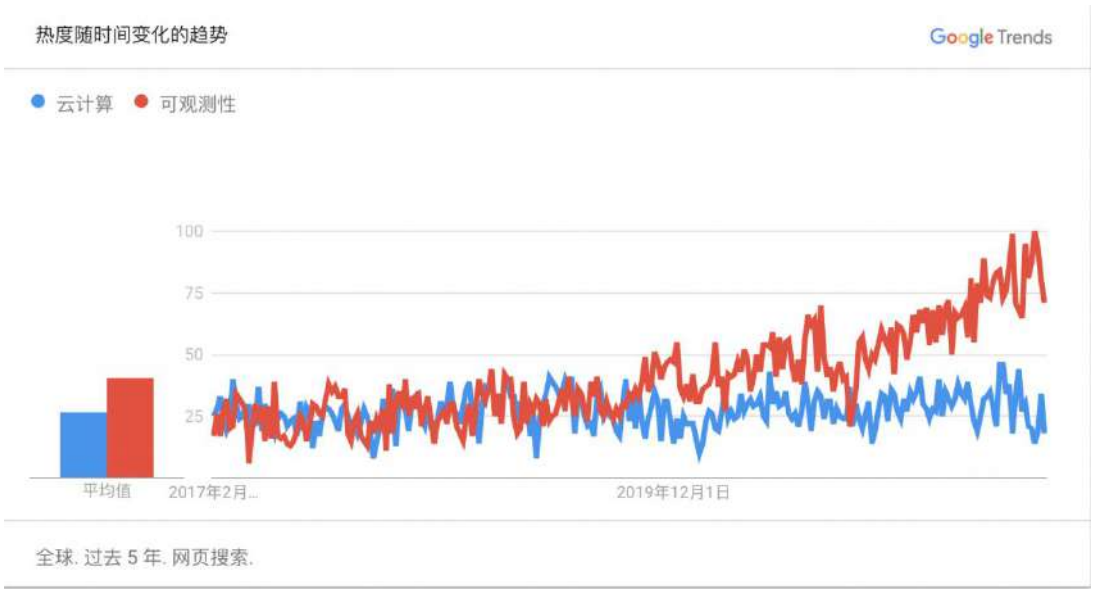

图 6-2：随着云计算的普及、可观测性热度的变化趋势[64]

在复杂的分布式系统中，可观测性能帮助系统维护者，理解和解决有关高度分布式系统中，正在发生的事情的具体问题。可观测性使系统维护者，能够

了解是什么导致系统逐渐缓慢，是什么导致损坏的，以及需要做什么来提高性能。有了可观测性解决方案，系统维护者可以收到有关问题的警报，并在问题影响用户之前主动解决这些问题。

由于现代云环境是动态的，并且在规模和复杂性方面不断变化，因此大多数问题既不为人知，也不被监控。可观测性解决了“未知的未知数”这一常见问题，使系统维护者能够在，新类型问题出现时，持续自动地理解问题的原因。

可观测性也是用于人工智能的一项关键能力。随着越来越多的组织采用云原生架构，他们也在寻找实施人工智能运维的方法，利用 人工智能作为，在整个维护生命周期中，自动化更多流程的一种方式。通过将人工智能应用于一切——从收集遥测数据，到分析整个技术堆栈中发生的事情——系统维护者可以获得对自动化应用程序监控、测试、持续交付、应用程序安全和事件响应，至关重要的可靠解决方法。

可观测性的价值并不仅限于系统维护者。一旦开始收集和分析可观测性数据，就有了一个了解数字服务，对业务影响的宝贵窗口。这种可见性能够优化转换、验证软件版本，是否满足业务目标、衡量用户体验满意度的结果，并根据最重要的事项，确定业务决策的优先级。

6.3 可观测性的困难

云环境会生成大量测试数据，尤其是在涉及微服务和容器化应用程序时。同时还产生了比过去必须解释的更多种类的数据。最后，所有这些数据到达的速度使得跟上信息流变得更加困难，更不用说及时准确地解释它以解决性能问题了。还经常遇到以下可观测性挑战：

数据孤岛：多个代理、不同的数据源和孤立的监控工具使得很难理解应用程序、多个云和数字渠道（如 Web、移动和物联网）之间的相互依赖关系。

数量、速度、多样性和复杂性：几乎不可能从不断变化的现代云环境（例如 AWS、Azure 和 Google Cloud Platform）中每个组件收集的大量原始数据中获得结论。对于 Kubernetes 和可以在几秒钟内启动和关闭的容器也是如此。

手动检测和配置：当 IT 资源被迫为每种新型组件，或代理手动检测和更改程序时，导致大部分时间都在尝试设置可观测性，而不是根据可观测性数据的见解进行创新。

故障排除时间成本高：对于不能从外部观测其内部变化的系统，当发生系统故障，那么就需要从系统外部，一个环节一个环节的排查问题，导致整体效率低下，时间成本非常高昂。

然后，还存在多种不同的应用服务的问题。虽然单个工具可以对，其应用程序架构的一个特定区域具有可观测性，但该工具可能无法提供，对所有可能影响应用程序性能的应用程序，和系统的完全可观测性。

此外，并非所有类型的日志数据，对于确定问题的根本原因，或了解其对系统体验的影响都同样有用。因此，仍然需要在多个解决方案中，寻找答案并解释日志所呈现的现象。

6.4 可观测性的实现策略

可观测性，对于云原生系统，对于拥有大规模资源的系统是非常重要的。同时由于云上系统的复杂性，导致可观测性存在诸多困难。本文通过对多种主流成熟技术的研究，并按照通用云原生大数据架构的特点，组合了一套标准的可观测性策略，以实现整个系统的可观测性要求。

在事件日志方面，随着大数据技术的发展、工业界的大量实践，日志收集和分析基本被统一到 EFK（ElasticSearch、Fluentd、Kibana）技术栈上。通过 Fluentd 采集日志，通过 ElasticSearch 存储 Fluentd 采集的日志，通过 Kibana 操作经 ElasticSearch 分类整理、优化存储的日志，从而实现事件日志端到端的收集与应用，最终满足可观测性的事件日志需求。

如图 6-3 所示，通过结合 ElasticSearch、Fluentd 和 Kibana 技术，构建一套完善的监控通知可视化系统。这种组合，为数据处理技术上云，提供了良好的可观测性支持。对于使用者，通过配置统一的可视化模版，则可以实现标准的监控观测能力。

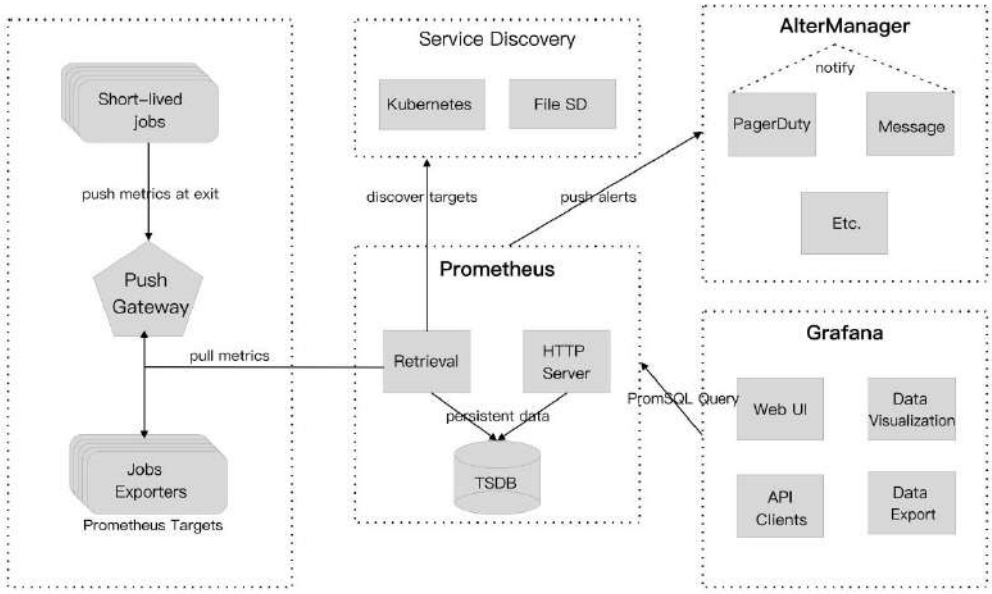

图 6-3：监控通知可视化流程

在链路追踪方面，追踪的主要目的是排查故障，从技术的发展以及工业界的大量实践来看，分布式环境下的链路追踪，主要是通过程序开发者在程序中嵌入链路追踪的接口，程序运行时会向这些接口提供数据，从而实现分布式容器服务下的链路追踪。本文架构采用模块化设计，各个模块不仅是独立的，而且属于专项功能（云存储元数据属于数据存储类、数据加速中间件属于数据协调类、数据处理属于任务计算类），所以网络IO、磁盘IO、CPU、内存为一个完整链路，通过可视化监控，既可分析出对应的链路现状用于排错与优化。

在聚合度量方面，随着 Kubernetes 统一容器编排的流行，Prometheus 的发展程度也超过了度量领域内，以 Zabbix 为代表的众多技术，成为云原生时代度量监控的事实标准。从社区活跃度上看，Prometheus 已占有绝对的优势。Prometheus 能够收集分析监控数据成为聚合度量指标，Grafana 把这些指标数据用可视化的方式呈现出来，AlertManager 则通过一些自定义规则完成实时的监控告警通知，其三者互相配合很好的解决了可观测性的聚合度量问题。

如图 6-4 所示，通过 Fluentd采集 Kubernetes 容器的日志，然后存储到 ElasticSearch ，当在监控上发现应用处于不健康状态时，则可通过 Kibana 直接

搜索 ElasticSearch 加速后的日志，准确定位问题所在，极大的提升了可观测性的能力。

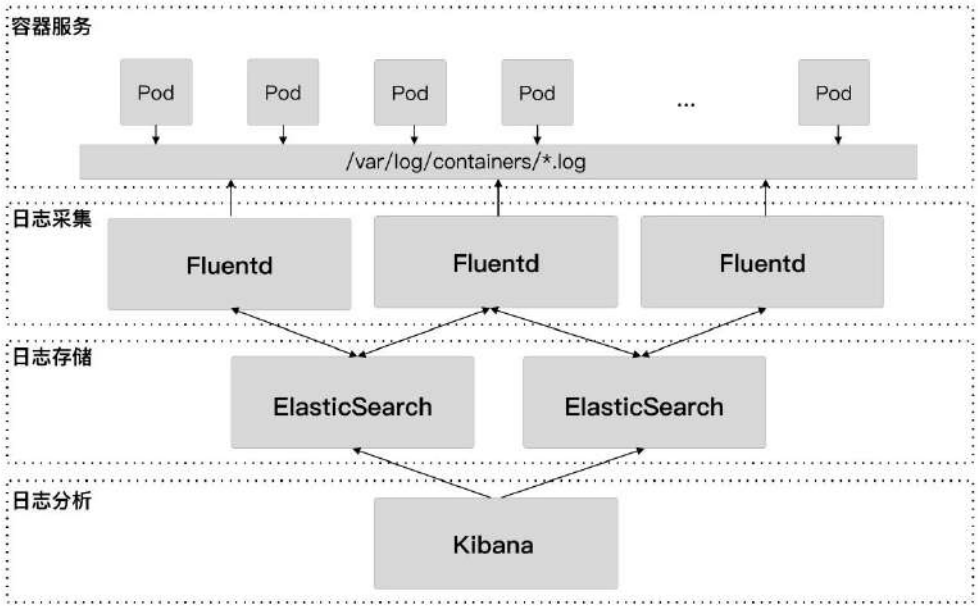

图 6-4：日志采集分析流程

6.5 本章小结

可观测性是了解整个系统状态的一种重要且有用的方法，能够更有效地监控现代系统，并帮助找到连接复杂链中的影响，并将其追溯到原因。通过 Prometheus、Grafana、AlertManager 解决可观测性的聚合度量问题；通过 Fluentd、ElasticSearch、Kibana 解决客观性的事件日志问题；通过模块化低耦合高内聚的架构特点，结合监控与日志，最终为可观测性提供很好的解决思路。

第七章 实验案例：ClickHouse 在不同架构下的性能表现

7.1 实验简介

本课题提出了一种通用的云原生架构模型，助力大数据技术快速上云，利用云的优势，获得更加强大可靠的数据处理能力，降低数据存储的成本。

ClickHouse[65] 是近几年非常流行 OLAP(On-line Analytical Processing) 型数据库，各方面性能指标均属行业领先地位。不过由于架构设计的限制，在扩展性、并发能力上有所欠缺。

本实验旨在对 ClickHouse，应用本文提出的通用架构模型，使 ClickHouse 实现存储计算的完全分离、资源的弹性伸缩，获得更加强大的数据处理能力，大幅度降低数据存储成本。最后结合严谨的科学评估方法，论证本课题的可行性以及创新性。

7.2 实验准备

本实验在国内最大的云平台（阿里云）环境下操作，实验所使用的云服务器、云存储、云数据库、云网络、云服务等资源均来自该平台，具体用到的云资源见表7-1。

表 7-1：云资源列表

资源名称	资源种类	备注/说明
云服务器	4c8g（ecs.c6.xlarge） 8c16g（ecs.c6.2xlarge） 16c32g（ecs.c6.4xlarge） 40c96g（ecs.hfc6.10xlarge） 80c192g（ecs.hfc6.20xlarge）	c代表CPU核心数量，g代表内存容量，括号内的是服务器型号，磁盘均为高效磁盘500GB或100GB
云存储	OSS	
云数据库	Redis	256MB内存 主从架构版
云网络	公网IP（按量计费100M） 负载均衡SLB（slb.s2.small） 企业级安全组 NAT网关	
云服务	Kubernetes 托管版标准服务	Kubernetes版本: 1.20.4-aliyun.1 Containerd 版本: 1.4.8

实验的测试对象为三种：集群版（cluster）、单机版（standalone）和本课题版（cmpp），见表7-2，为了保证实验公平，三种对象所使用的环境资源总成本相同，比如：集群版和本课题版同时使用5台服务器，单机版会使用一台性能为单台5倍的服务器。

表 7-2：实验对象列表

实验对象	架构类型	备注/说明
集群版（cluster）	五分片一副本	ClickHouse版本：21.9.2.17
单机版（standalone）	单分片单副本	
本文版（cmpp）	多分片单副本	

实验所采用的数据集和查询条件来自 SSB(Star Schema Benchmark)[66]，它是麻省州立大学的研究人员定义的基于现实商业应用的数据模型，业界公认用来模拟决策支持类应用，比较公正和中立。学术界和工业界普遍采用它来评价决策支持技术方面应用的性能。本次实验，基于 SSB 标准生成了 s100 规模的数据集，见表7-3。并对 SSB 提出的12种查询语句进行调整，使其适配 ClickHouse，为了避免缓存采用了动态过滤条件。

表 7-3：SSB数据集列表

数据集名称	数据集种类	备注/说明
s100	customer（300万）：客户表 part（140万）：零部件表 supplier（20万）：供应商表 lineorder（6亿）：商品订单表 lineorder_flat（6亿）：商品订单宽表	括号里面的是数据量，冒号后面的是表名称。

实验所采用的测试工具为 Jmeter，它是 Apache 组织开发的基于 Java 的压力测试工具，专用于对软件做压力测试，通过对服务器、网络或对象模拟巨大的负载，来在不同压力类别下测试它们的强度和分析整体性能。

7.3 实验方法

首先在不同实验环境上，分别部署 ClickHouse 集群版（cluster）、单机版（standalone）、本文版（cmpp）架构。集群版用的是工业界标准架构（ClickHouse+Zookeeper），五个分片，每台服务器一分片，磁盘均 500GB（为了提高集群部署效率，基于 Ansible 开发了集群快速部署工具[67]）。单机版非常简单，只包含一台服务器，磁盘 500GB，单服务器性能是集群版单服务器的5倍。本文版部署在 Kubernetes 之上，其中 Kubernetes 属于托管版服务，服务器性能等同于集群版，因为数据放在云存储，所以服务器磁盘只分配了 50GB。

然后根据 SSB 文献提供的开源数据集生成工具[68]，生成 s100 规模数据集（事实表 6 亿条记录），并导入到不同环境的不同实验对象。实验对象作为服务端，数据导入程序会单独运行在 4c8g 的客户端服务器上。数据导入时，通过 ClickHouse 自带的 clickhouse-client 工具，读取本地数据集文件进行导入。

最后在实验环境、实验对象准备完成后，使用 Jmeter 调用实验对象的服务接口，按照 10、30、50、100 的并发策略，分别发送 12 种查询条件，并把 Jmeter 记录的查询响应结果保存下来，以供评估分析。在每种测试项开始前，会等待几分钟，以便服务器负载恢复正常。每种测试项，都会重复执行三遍，以保证结果更加稳定。所有查询均设有 180s 超时，超过 180s 则该查询被自动取消。为了保证结果分析的科学性，会采用 90th percentile（百分位数）计算总的查询响应时间，百分位数可以更加客观的衡量样本的整体情况。具体实验策略，见表 7-4。

表 7-4：实验策略

实验对象	实验数据集	实验环境
集群版（cluster）	s100	8c16g * 5（台） 16c32g * 5
单机版（standalone）	s100	40c96g * 1 80c192g * 1
本文版（cmpp）	s100	8c16g * 5 16c32g * 5

7.4 实验结果

按照科学合理的实验方法，对 ClickHouse 集群版、单机版、本文版做了大量实验，并收集到了许多实验结果。为了便于复现操作环境，本实验所用到的相关代码、辅助工具、操作手册，以及完整的测试结果、分析结果等，均整理存放在开源平台 GitHub[69]，以便随本文同期公开。为了使实验结果客观且具有

代表性，选取了中等并发（50并发）和高并发（100并发）两种压力环境下、服务器配置为 16c32g 的测试结果。

图 7-1、图 7-2 展示了不同并发下的不同架构的查询响应时间。从查询响应时间来看，本文架构在 12 种查询场景上，总体查询性能相较于集群版有 18% 的提升，相对于单机版有 60% 的提升。部分查询场景则有 2 到 8 倍的性能提升。随着并发量从 50 增加到 100 ，查询优势逐渐增高，本文架构表现更优，说明本文架构提高了原技术的性能，更适应于高并发、大数据量的场景。

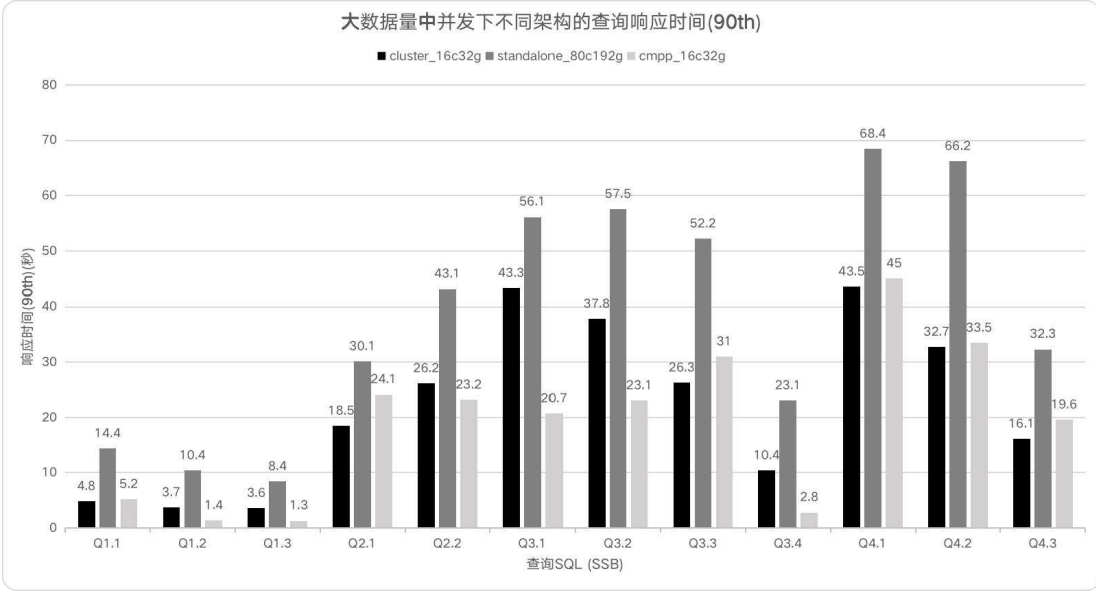

图 7-1：大数据量中并发下不同架构的查询响应时间

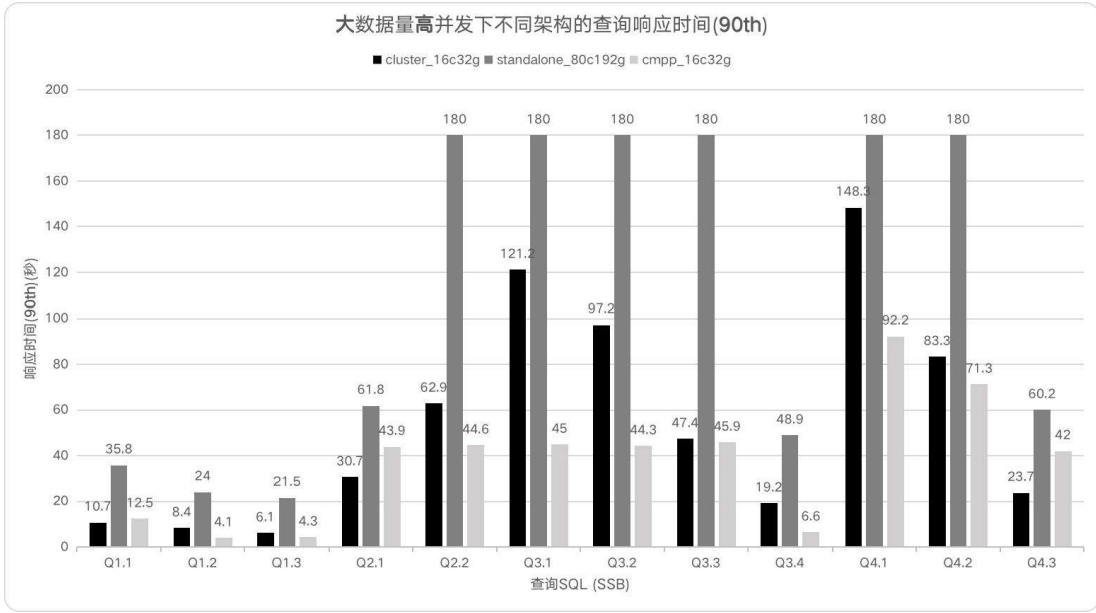

图 7-2：大数据量高并发下不同架构的查询响应时间

图 7-3、图 7-4 展示了不同并发下的不同架构的 TPS 响应情况。从 TPS 响应情况来看，本文架构每秒钟能够处理的事务数量，多于集群版和单机版，且更加稳定，证明其并发能力与可靠性较高。单机版出现较多的峰针，说明其稳定性不足。集群版的曲线紧邻底部，说明集群版的并发能力不佳，而且随着并发数的提高，情况则更加突出。

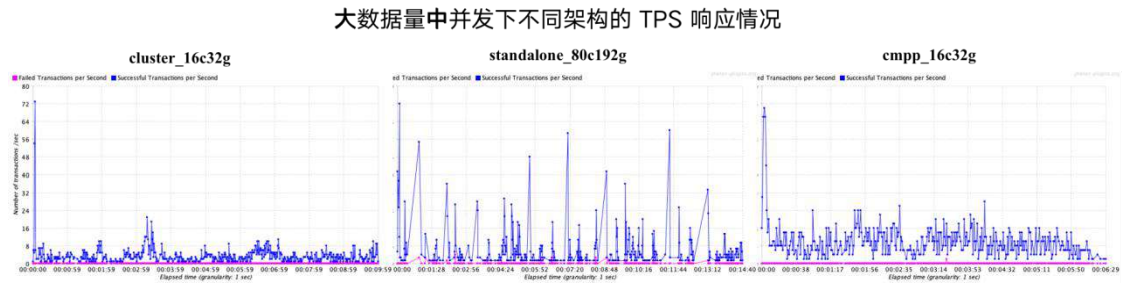

图 7-3：大数据量中并发下不同架构的 TPS 响应情况

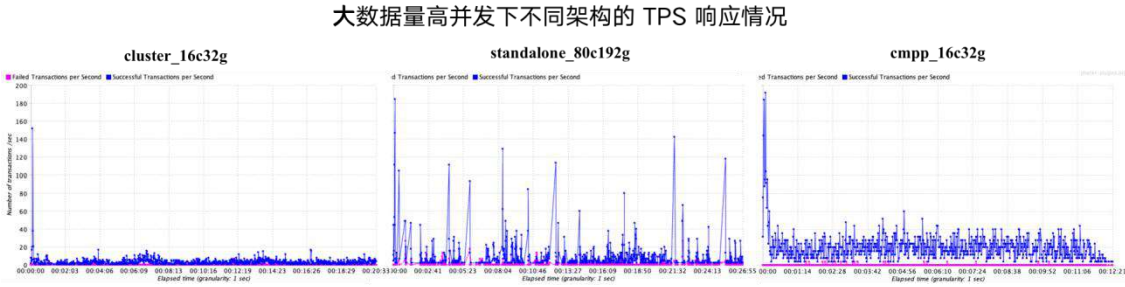

图 7-4：大数据量高并发下不同架构的 TPS 响应情况

图 7-5 展示了在云原生环境下，数据处理技术便捷的弹性资源伸缩。通过服务负载管理界面，使用加减既可调整服务数量，提高服务能力或者是降低服务能力。从中可以看出，5 台 16 核的服务器，运行了 69 个容器化后的数据计算实例（少部分资源被容器管理服务占用，实际可运行更多实例），使服务器的资源得到充分使用。

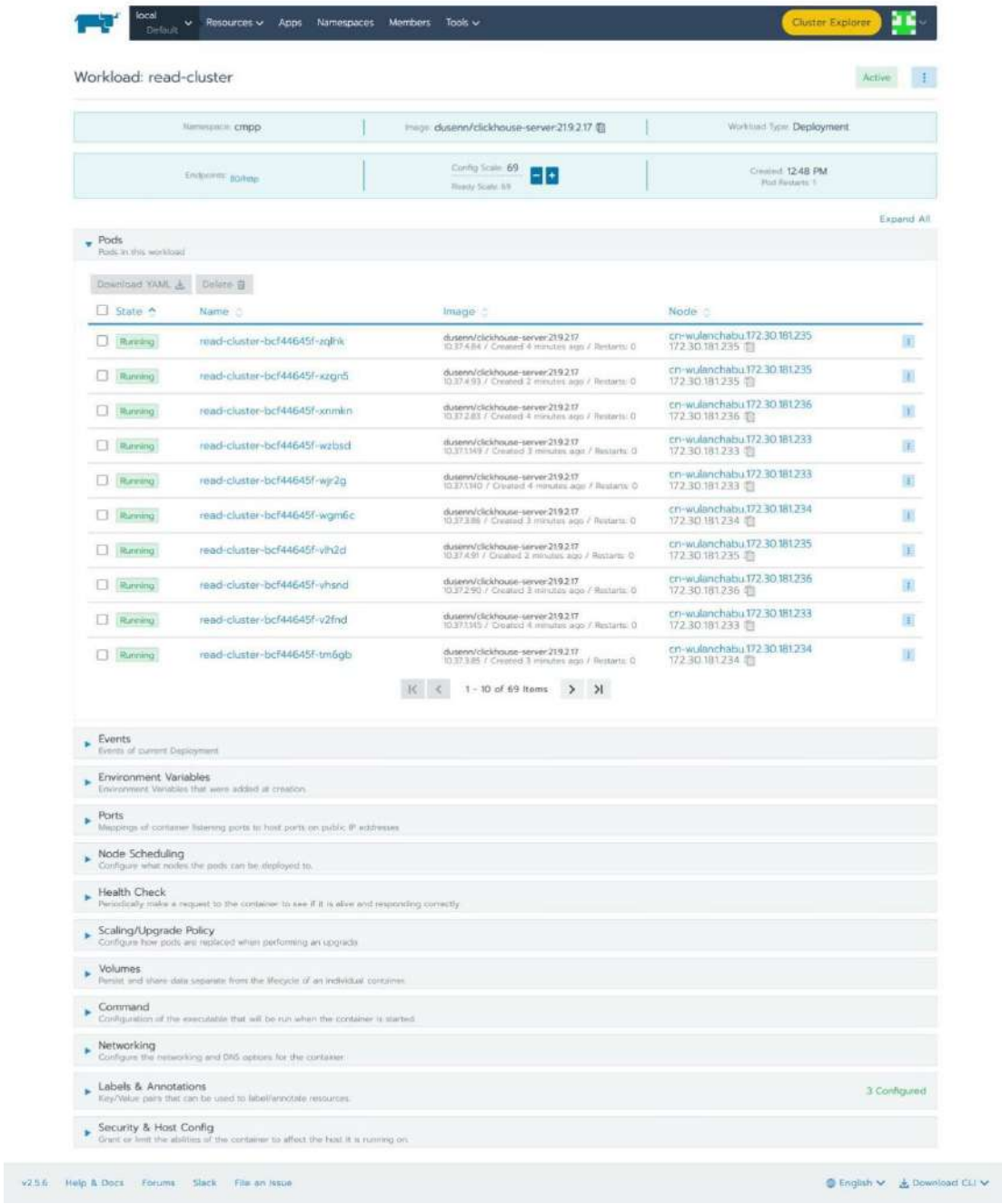

图 7-5：资源的弹性伸缩

图 7-6 展示了本文架构可观测性能力的可视化部分。在可视化的数字大屏中，除了呈现所有服务器的基本信息（CPU 核心数、内存容量、IP 地址等），还实时地提供了云资源的整体负载情况、CPU 使用率、内存使用率、网络带宽使用率等信息。根据这些信息能够洞察整体系统健康状况，快速定位问题发生地点。由于系统的所有运行信息都被收集，并提供了对应的访问接口，所以通过定制可视化展示模版，能够获得更加全面的监控能力。

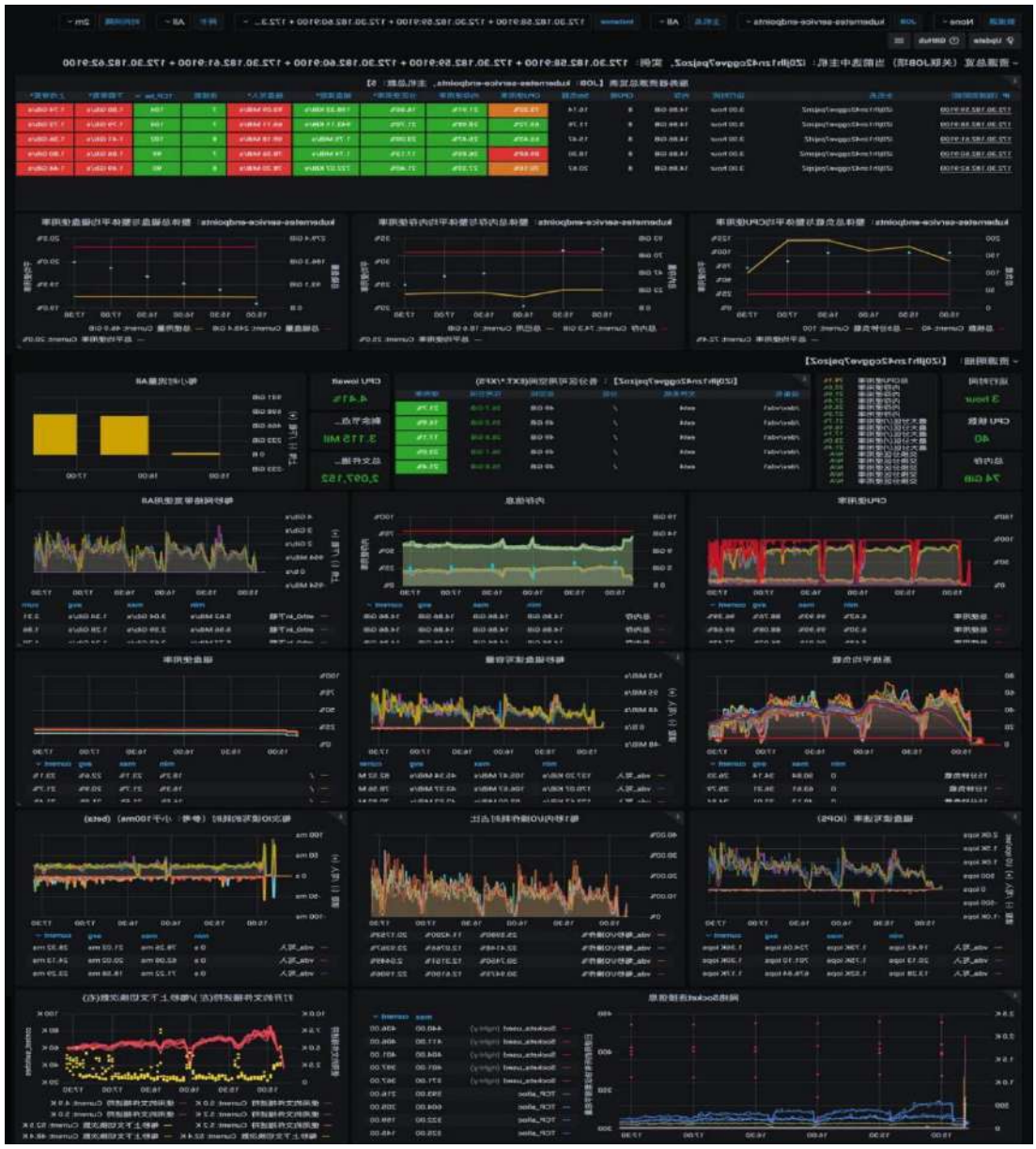

图 7-6：系统的可视化监控

图 7-7，展示了压力测试环境下的云存储性能表现。由于结合了云存储加速中间件，云存储的吞吐性能可以达到 10GB 每秒，整体读写能力远快于传统磁盘。本次实验的 s100 数据集，单表有 6 亿条数据，在云存储上占用了 130 GB 的空间。由于云存储的按使用量计费模式，且云存储价格低廉，每月数据存储成本只有 50 元左右。而集群版每台服务器要保证 500GB 容量的磁盘，一方面因为云服务器磁盘容量越低，读写速度越慢；另一方面，集群版的扩容相当于集群重装，为了避免扩容，所以需要留有相对的空余。所以，集群版 5 块磁盘每月成本 850 元左右。最终，通过合理使用云存储，能够减少 90% 以上的数据存储成本。

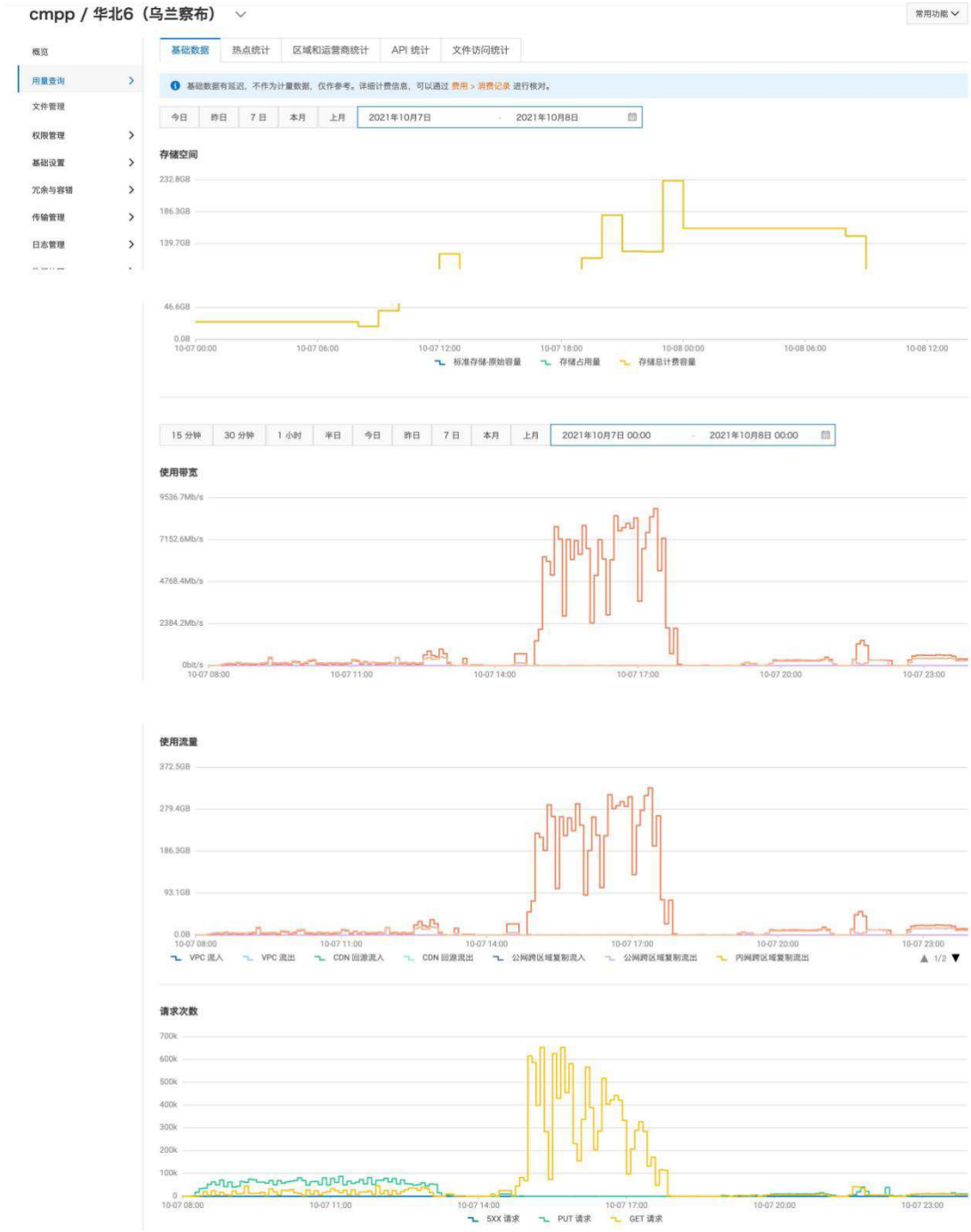

图 7-7：云存储的吞吐能力

7.5 实验结果分析

为了更加客观、且全面的研究不同架构的优劣，将按照表 7-5 列出的七大评估纬度进行分析，分别是查询性能、数据成本、数据规模、数据一致性、数据可靠性、系统扩展性、运维复杂性。

表 7-5：七大维度定义

维度名称	维度定义
查询性能	相同查询环境（资源、并发），不同架构的响应表现
数据成本	存储相同大小数据，需要的经济成本
数据规模	系统正常运行时能够容纳的数据总量
数据一致性	所有节点在同一时间的数据是否完全一致
数据可靠性	节点宕机，数据是否可以恢复正常
系统扩展性	系统的计算能力、存储能力是否容易扩展
运维复杂性	系统的版本升级、性能监控、日常维护的复杂度

7.5.1 查询性能

从查询性能维度来看，本文架构相较于集群架构有 18% 的提升，相较于单机架构有 60% 的提升。且拥有更高的并发能力，更稳定的 TPS 响应。带来这些提升的主要原因是：

一、采用了容器化技术隔离资源，以云原生的方式运行服务。这种方式使数据处理技术更加轻量，在相同的服务器资源中能够运行更多实例，且不会有资源冲突。能够充分利用服务器的算力、网络等各方面资源。

二、受益于容器管理技术的负载均衡，查询任务会被均匀且合理的分配到不同服务实例，使得每个服务实例被充分利用，不会产生热点现象，有助于 TPS 更加稳定。

三、采用了存算分离、共享存储、云存储加速，减少了数据的重复计算、重复传输开销，避免了服务器的磁盘瓶颈现象。集群架构的多阶段计算、数据聚集，产生了较多额外的磁盘 IO、网络 IO，最终会导致磁盘性能成为瓶颈。单机架构只有一块数据盘，即使服务器算力较高，过多的数据计算任务，也将很快耗尽磁盘性能。本文架构，根据云存储底层原理（多数据副本、多数据节

点等），采用数据分片、压缩、乱序命名等方式，使数据合理分散到云存储的多个数据节点，在查询时通过并行技术，充分利用多数据节点的 IO、缓存能力，最终使得整体数据读写能力远大于传统磁盘。

7.5.2 数据成本

从数据成本维度来看，本文架构相较于集群架构减少了 90% 以上的数据存储成本，相较于单机架构减少了 50% 以上的数据存储成本。且拥有更高的数据安全性，更多的数据副本，更快的整体读写速度。带来这些提升的原因是：

一、集群或单机架构使用传统物理磁盘，很难支持灵活扩缩容量。对于数据处理技术，数据增长非常快，在架构设计初期则需要预置一定冗余空间。云存储完全按使用量计费，而且拥有几乎无限的总容量，即不用担心数据持续增长导致的存储空间不足问题，还避免了闲置存储空间的浪费。

二、集群或单机架构使用的存储介质，是相对不可靠的。数据本身是宝贵的资产，需要进行安全性设计，避免丢失、损坏。采用基础的数据备份逻辑，只有一个副本，那么集群或单机也将使用双倍的存储磁盘。云存储底层通常采用 3 到 6 副本，进行数据冗余存储，同时屏蔽了复杂的数据备份细节，更便于使用。

三、主流云厂商的云存储产品，经过了大量严格的功能、性能测试，以及真实生产环境的磨练，在数据安全性上远高于普通集群的多副本架构。云存储的数据存放在世界各地专业的数据中心，有着较为完善的维护与管理能力。良好的基础设施带来了更快的网络连接、更稳定的服务，通常云存储的接入网络，可以达到万兆、十万兆以上，提高了数据访问速度。云存储的每一份数据，通常会保存 3 到 6 个副本到不同的数据管理区，单个区域遇到不可抗力自然因素，数据也能正常使用，并自动容错、恢复，提高了数据安全性。

四、对于真实生产情况，集群或单机架构通常需要 3 副本的数据备份，采用多份的磁盘空间与计算资源，而且数据量普遍按 TB、PB 计算。由于本文架

构采用了廉价且高可靠的云存储，最终能够减少的成本，相对于集群远高于90%，相对于单机远高于 50%。

7.5.3 数据规模

从数据规模纬度来看，

一、单机受限于单个存储介质的容量，单个存储介质最多只有几 TB 的容量。在大数据时代，数据通常以 TB、PB、ZB 计算，单机很难存储大规模的数据。

二、集群有多个计算节点，通过把数据分散到多个计算节点，相当于同时使用多个存储介质。不过由于并行计算架构，节点过多时数据过于分散，将耗费更多的数据传输开销，以及受限于处理性能最差的机器。所以即使采用集群架构，数据存储的容量也是有限的，数据存储过多将导致集群无法正常工作。

三、本文架构采用云存储，云存储拥有几乎无限的存储空间，可以存储非常多的数据。云存储对外提供了统一的访问接口，同使用单块存储介质无差异，对内把数据分散到多个数据节点，自动完成数据容错、复制等复杂操作。

7.5.4 数据一致性

从数据一致性纬度来看，

一、单机架构采用单个存储介质，对于数据读取、数据写入、数据更新等操作，通过原子性事务、隔离级别等技术，能够保障数据读写完全一致。

二、集群架构包含多个节点、多个存储介质，简单的原子性事务、隔离级别，不能保障数据读写完全一致。需要借助更复杂的技术，比如：异步复制、多阶段事务、共识算法，这些策略最终可以保证数据完全一致，但是为了提高数据处理技术的性能，多数分析型数据处理技术未实现完全的一致性。所以集群在数据一致性上，低于单机架构。

三、本文架构采用共享存储，全部数据处理服务共享同一份存储介质，等同于单机的单个存储介质。所以本文架构，能够保障数据读写完全一致。

7.5.5 数据可靠性

从数据可靠性维度来看，

一、单机架构，如果不采用特殊处理，那么数据只会在当前节点存储一份，当节点故障，数据则无法恢复。如果采用主从复制，那么数据不仅会在当前节点存储一份，还会在从节点存储一份，当其中一个节点发生故障，数据还可恢复，当两个节点同时发生故障，数据不可恢复。一般来说，可通过添加更多的从节点，提高数据可靠性。但是，主从节点之间，存在网络延迟，在数据还未复制到从节点时，主节点出现故障，数据则无法完全恢复。物理世界中不仅单个节点会出现故障，整个机房也可能出现故障，所以，单机架构很难完全保障数据可靠性。

二、集群架构，和单机类似，如果没有副本策略，很容易丢失数据。如果设计副本策略，则需要考虑数据已经分片处理，每个子节点都需要一个副本，总体会需要更多的副本节点，相比单机，复杂度将高出许多。所以，集群架构也很难完全保障数据可靠性。

三、本文架构，采用云存储。云存储内部设计，自动冗余 3 副本以上，分散到多个数据中心，提供 11 个 9 的高可靠性保障。相对于单机和集群架构，拥有更高的数据可靠性。

7.5.6 系统扩展性

从系统扩展性维度来看，

一、单机架构，如果需要提升系统性能，只能升级 CPU、内存、磁盘等硬件设备，采用垂直扩展的方法。单 CPU、内存、磁盘的性能总是有限的，将很快遇到扩展的瓶颈，所以单机架构的系统扩展性较低。

二、集群架构，可以通过添加节点的方式，水平扩展系统性能。不过集群架构的水平扩展是有限的，而且比较相对复杂。一方面，受限于系统架构与计算模式，集群在一定范围的添加节点，可以提升整体性能。当达到临界点之

后，添加节点反而会降低整体性能。因为数据查询任务，会同时使用集群中的所有节点，这些节点中会产生大量数据传输交互，当节点数过多，节点间的通信成本将会急剧上升，导致边界条件的收益递减。另一方面，一个运行中的集群，数据已经合理分配到所有节点，当新加入节点时，会导致集群数据的不均衡。这种情况，或者触发数据均衡，导致短期的集群性能下降。或者只均衡未来数据，历史数据不考虑，导致数据不均衡持续累积，集群总体性能不稳定。这些可能情况，都会导致加入新节点，是复杂的。所以集群架构比单机架构优良，可以拥有中等的系统扩展性。

三、本文架构，基于云原生设计，系统的扩展性非常高。首先数据处理技术已经被容器化，非常轻量，易于调度。其次由于使用了存储计算分离设计，共享存储，系统的扩展不需要考虑数据的重新均衡、移动操作。最后由于结合了成熟的容器编排技术，云资源的管理、容器服务的调控，能够很容易完成。可以在后台管理系统中，点击加减扩缩系统性能。也可以配置一定的规则，系统负载达到一定阀值，自动扩展系统性能。所以本文架构的系统扩展性，远优与集群和单机架构。

7.5.7 运维复杂性

从运维复杂性纬度来看，

一、单机架构，需要考虑服务的高可用、数据的高可靠，则需要引入服务容错处理、灾难恢复机制、主从复制策略等技术，还需要引入监控通知技术，以实时保障服务可用。所以单机架构的整体运维复杂性是比较高的。

二、集群架构，除了单机架构需要考虑的问题，还需要考虑更多问题，比如：多节点的数据一致性、多节点的负载均衡、节点间的网络通信、单一节点故障数据的自动均衡等。所以集群架构比单机架构的运维复杂性还要高。

三、本文架构，容器化运行在云原生环境中，通过容器编排技术，配置服务存活探针检测、异常自动恢复、调度优先级策略，能够自动保障系统正常运行。再结合本文架构，集成的标准化可观测性技术，提供统一的可视化监控、

预警通知、问题追踪、日志分析等，能够实时了解系统状态，快速定位故障问题。所以本文架构的运维复杂性较低。

7.5.8 总结

本文架构相对于集群和单机，不仅提升了查询性能，而且大幅度降低了数据成本，在数据规模、一致性、扩展性等方面也做出了较大改善，使得数据处理技术更容易上云，发挥更大的价值。最后，表 7-6 展示了本文架构多维度评估与分析的概览情况。

表 7-6：架构评估与分析

纬度名称	不同架构的能力水平		
	集群版	单机版	本文版
查询性能	★★★★☆	★★☆☆☆	★★★★★
数据成本	★★★★★	★★★☆☆	★☆☆☆☆
数据规模	★★★★☆	★★☆☆☆	★★★★★
数据一致性	★★★☆☆	★★★★★	★★★★★
数据可靠性	★★★☆☆	★★☆☆☆	★★★★★
系统扩展性	★★★☆☆	★☆☆☆☆	★★★★★
运维复杂性	★★★★★	★★☆☆☆	★☆☆☆☆

7.6 本章小结

为了验证，本文提出的通用云原生大数据架构的合理性、创新性，改造了大数据领域流行的 ClickHouse 技术，引入麻省理工大学提出的 SBB 评估方法，把 ClickHouse 集群版、单机版作为参考对象，使用 Jmeter 完成了 12 种查询场景的压力测试。

根据压力测试结果，结合七大纬度进行评估分析。从查询性能的情况来看，本文架构相对于集群版提高了 18%，相对于单机版提高了 60%，部分查询场景有 2 到 8 倍的提升。从数据成本的情况来看，本文架构相对于集群版减少了 90% 以上，相对于单机版减少了 50% 以上。从数据一致性、数据一致性、系统可扩展性的情况来看，本文架构均优于集群版和单机版。从运维复杂性的情况来看，本文架构比集群版和单机版更加容易。

第八章 总结与展望

数据量的不断膨胀，数据处理技术的日新月异，云计算技术的逐渐成熟，带来机遇的同时，也带来了不少挑战。 伴随着摩尔定律的逐渐失效，单机性能垂直扩展能力几乎停滞，传统的数据处理技术面临淘汰的厄运。为了解决数据处理能力的不足，众多细分领域的数据处理技术应运而生，或专注于文档搜索，或专注于键值查询，或专注于数值计算。在不断深入使用的过程中，虽然解决了部分性能问题，但是也带来了较多数据问题，依旧需要配合使用传统的数据处理技术。随着云原生概念的普及，新型数据处理技术旨在融合多种数据处理技术，成为事务分析混合型产品。在新型数据处理技术发展完善之前，改进传统数据处理技术，使其能够借助云的能力，提高数据处理能力，降低数据成本，发挥更大的作用，成为数据处理技术发展的一种新思路。

8.1 本文总结

本文通过分析数据处理技术和云计算技术的演进特点，深入研究了数据处理技术与云计算技术的融合策略，提出了一种基于 Kubernetes 的通用云原生大数据架构。此外为了该架构的通用性、云原生性、降低成本、提高性能，本文还提出了一种云存储优化策略、存算分离方法，改进了数据并行处理方式。最后为了验证该架构的合理性、创新性，本文对 ClickHouse 进行了架构改造、开发了集群自动化构建工具、改进了 SSB 基准测试场景、收集与分析了大量实验结果、提出了七大评估纬度等。本文工作可总结为以下十点：

1. 对数据处理技术的发展进行了研究分析。本文通过对数据处理技术进行研究后，发现其演进过程分为三个阶段，合-分-合。最初数据量不多，单机单数据库就可以容纳所有数据。随着数据量的快速增长，演变出了多种数据处理技术，呈现为百花齐放、百家争鸣的局面。多样性的数据处理技术，造成了诸多复杂性，带来了数据孤岛现象。云计算技术的成熟，使得传统数据处理技术，有了提高自身能力的机会，更促进了分析事务混合型数据库的发展。从最

新的数据库领域的发展可以看出，整合多种数据处理技术、对外暴露统一接口、对内实现复杂技术、自动容错调度恢复、云原生化的混合型数据产品属于前沿热点方向。

2. 对云计算技术的发展进行了研究分析。本文通过对云计算技术进行研究后，发现云计算技术并不是单一的技术，而是由多种技术组合而成，其发展演进多依赖于基建设施、基础领域的发展。随着虚拟化、分布式数据存储、大规模容器管理、信息安全等各项技术、产品的共同发展，云存储、云服务、云管理、云算力成为这个时代的基础能力，使得更多传统技术，以及新的技术获得赋能与提高。

3. 对数据处理技术与云计算技术融合的发展进行了研究分析。本文通过对数据处理技术与云计算技术进行研究，发现在云技术发展早期，数据处理技术与云的融合是相对简单的，产生了 RDS 这类融合技术。在云技术发展中期，逐渐深入结合多种云基础设施，构建大规模分布式集群，产生了众多细分领域的数据处理技术。云技术发展到现在，多领域的数据处理技术开始进行融合，以统一标准接口的方式提供使用。随着数据处理技术、云计算技术的发展，融合的程度将会更高。在此分析事务融合型技术成熟前，助力传统数据处理技术上云，提高性能、降低成本，则属于亟待解决的问题。

4. 设计了一种通用的云原生大数据架构。本文通过深入研究前沿成熟技术（Kubernetes、Rancher、JuiceFS、OSS、Prometheus、Fluentd等），以模块化低耦合的方式，组合设计了一种通用架构。相对于其它文献研究[39][40][41][42][43][44][45]，能够适应更多的数据处理技术，简化了整体构建运行流程，提高了数据查询速度，增强了并发处理能力，大幅度降低了数据存储成本。

5. 设计了一种云存储加速策略。本文通过深入分析云存储特性，研究对比多种存储缓存技术（JuiceFS、EFS、S3FS等），最终根据通用架构的特点，结合 CSI 容器存储接口技术，引入 JuiceFS 缓存加速层，使得加速后的云存储性能，相比之前有几十倍的提升。

6. 设计了一种存储计算分离方法。本文通过深入研究 Kubernetes 的 PV、PVC 技术，以及云存储加速缓存层的特点，使数据处理技术在云上，能够以高速共享存储、存算分离的方式运行。最终本文架构，相对于集群架构能够减少 90% 以上的存储成本，相对于单机架构能够减少 50% 以上的存储成本。

7. 设计了一种基于云的并行计算方法。本文通过深入分析 MPP 计算架构的不足（重复计算、节点通信、短板效应等），结合本文通用云原生架构的特点（轻量化容器服务、动态负载均衡、海量无状态计算节点等），提出了 MPP 的云上改良版 CMPP，减少了节点间数据多次传输开销，避免了集群中常见的短板效应，支持更多的并发查询请求，提高了响应请求的稳定性。最终，本文架构在基准测试上，相较于集群架构提高了 18% 的查询性能，相较于单机架构提高了 60% 的查询性能，同时 TPS 更加稳定。

8. 设计了一种云上服务的可观测性策略。本文通过研究可观测性的主要理论（事件日志、链路追踪、聚合度量等），结合主流成熟工具（Prometheus、AlertManager、Grafana、Fluentd、ElasticSearch、Kibana 等），组合出一种标准的端到端的监控通知可视化、运维管理自动化的模块系统。最终根据本文架构的特点，以插件化模版的形式引入，实现数据处理技术上云后的可观测性要求，降低本文架构的维护使用复杂性。

9. 设计实现了一种集群自动化构建工具。为了更高效的完成基准测试，根据 Ansible-Playbook 技术，实现了一个 ClickHouse 集群和单机的快速自动化构建工具，并开源至 GitHub 平台[64]。

10. 完成了不同实验环境、系统架构、数据规模、压力范围的大量组合基准测试，并对实验结果进行收集与分析。为了验证本文架构的合理性和创新性，针对不同实验环境（4c8g、8c16g、16c32g、80c192g等服务器配置），不同系统架构（本文架构、集群架构、单机架构），不同数据规模（单表 6000 万、单表 6 亿），不同压力范围（10、30、50、100、200 等并发数），进行大量组合测试。累计进行了 100 多种环境构建、500 多项基准测试、404,400 多次压力测试、收集到了 492,000 多条测试结果（所有测试脚本与工具、实验结

果，均已开源至 GitHub 平台[66]）。然后通过对结果进行统计分析，提出七大评估纬度，全面地评估分析了本文架构、集群架构、单机架构之间的优劣。最终得出本文架构在查询性能上高于集群和单机，在数据成本上少于集群和单机，在数据规模、一致性、可靠性上优于集群和单机，在系统扩展性上易于集群和单机，在运维复杂度上低于集群和单机。

8.2 未来展望

本文为了促进数据处理技术与云计算技术融合、提高查询性能、降低存储成本、增强整体能力，提出了一种基于 Kubernetes 的通用云原生大数据架构。在未来还可以针对以下问题，进一步提高通用云原生大数据架构的能力：

1. 引入统一的权限认证服务。本文实现了数据处理技术快速上云，无需复杂的架构改造，只用容器化数据处理技术，结合 Kubernetes 即可完成部署运行。访问请求经过负载均衡后，会直接转发到容器服务。权限任务与数据处理，会由收到访问请求的容器服务。通过引入统一的权限认证模块，一方面，数据处理技术的容器服务，只用处理数据请求，可以减轻服务压力，提高整体性能。另一方面，剥离权限认证，可以更细粒度的控制访问权限，提供标准的访问接口。最终可以做到对外提供统一接口，内部屏蔽技术处理细节，使整体架构更加完善，且易用。

2. 引入抽象语法树(AST)预解析能力。本文提出的架构，对于数据读写请求，将按照网关路由层的定义，转发至读服务或写服务。通过引入 AST 预解析模块，对请求内容进行一次简单语法解析，分离出 SELECT、INSERT、UPDATE、DELETE、ALTER 等操作，智能判断请求类型，转发至读写服务。最终只提供单独的访问接口，而不用通过多个接口区分读写服务，提高数据访问效率，降低操作复杂性，提高架构通用性。

3. 引入动态数据字典技术。本文提出的架构，能够很好的处理大数据量高并发查询访问，相对于单机或集群都有较大提升。不过由于采用共享存储，所有服务访问相同数据。为了避免智能负载均衡导致的写入冲突，写服务默认只

运行一个。从整体写入性能的角度来看，本文架构的写入性能等同于单机，低于集群。通过引入动态数据字典技术，自动提取并缓存数据库与表之前的关系。借助抽象语法数预解析能力，智能判断用户的写入请求。若可能发生数据写入冲突，则转发请求到相同容器服务（单服务内部会自动排队处理）。若不会发生写入冲突，则按照负载均衡策略自动转发。最终写服务将和读服务保持一致，支持任意水平扩展，预计比当前的写入性能，可以达到数倍的提升。

参考文献

[1] 中华人民共和国国民经济和社会发展第十四个五年规划和 2035 年远景目标纲要 website[EB/OL]. 2021. http://www.gov.cn/xinwen/2021-03/13/content_5592681.htm.

[2] DB-ENGINES. website[EB/OL]. 2022. https://db-engines.com/en/ranking_trend.

[3] CODD, Edgar F. A relational model of data for large shared data banks. In: Software pioneers. Springer, Berlin, Heidelberg, 2002. p. 263-294.

[4] ASTRAHAN, Morton M.. , et al. System R: Relational approach to database management. ACM Transactions on Database Systems （TODS）, 1976, 1.2: 97-137.

[5] 徐兴雷, 汪婵婵. 反范式在海量数据库设计中的应用[J]. 科技传播, 2011(5):2.

[6] 谢振华. 基于分布式的数据库分库与分表策略研究[J]. 电脑知识与技术：学术版, 2020, 16(14):2.

[7] Sacha J, Dowling J. A Gradient Topology for Master-Slave Replication in Peer-to-Peer Environments[J]. Springer-Verlag, 2005.

[8] 练振兴. MySQL 读写分离的技术原理[J]. 福建电脑, 2019, 35(8):3.

[9] GHEMAWAT, Sanjay; GOBIOFF, Howard; LEUNG, Shun-Tak. The Google file system. In: Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003. p. 29-43.

[10] DEAN, Jeffrey; GHEMAWAT, Sanjay. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2008, 51.1: 107-113.

[11] CHANG, Fay, et al. Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems （TOCS）, 2008, 26.2: 1-26.

[12] SHVACHKO, Konstantin, et al. The hadoop distributed file system. In: 2010 IEEE 26th symposium on mass storage systems and technologies （MSST）. Ieee, 2010. p. 1-10.

[13] George, Lars. HBase: The Definitive Guide[J]. Andre, 2011, 12(1):1 - 4.

[14] Thusoo A , Sarma J S , Jain N , et al. Hive - a petabyte scale data warehouse using Hadoop[C]// Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA. IEEE, 2010.

[15] Borthakur D . HDFS architecture guide. 2008.

[16] 周洪斌. Hadoop 大数据平台 Pig 应用研究[J]. 沙洲职业工学院学报, 2019, 22(3):5.

[17] Evans R . Apache Storm, a Hands on Tutorial[C]// 2015 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2015.

[18] Zaharia M , Chowdhury M , Franklin M J , et al. Spark: Cluster computing with working sets. 2010.

[19] C Gormley. Elasticsearch: The Definitive Guide[J]. Oreilly Media, 2015.

[20] Chodorow K , Dirolf M . MongoDB: The Definitive Guide. O'Reilly Media, Inc. 2013.

[21] Partner J , Vukotic A , Watt N . Neo4j in Action[J]. Pearson Schweiz Ag, 2014.

[22] Gollapudi S . Getting started with Greenplum for big data analytics[J]. better crops, 2013.

[23] 曾超宇, 李金香. Redis 在高速缓存系统中的应用[J]. 微型机与应用, 2013, 32(12):3.

[24] CORBETT, James C., et al. Spanner: Google's globally distributed database. ACM Transactions on Computer Systems （TOCS）, 2013, 31.3: 1-22.

[25] SHUTE, Jeff, et al. F1: A distributed SQL database that scales. 2013.

[26] PENG, Daniel; DABEK, Frank. Large-scale incremental processing using distributed transactions and notifications. 2010.

[27] TAFT, Rebecca, et al. Cockroachdb: The resilient geo-distributed sql database. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020. p. 1493-1509.

[28] HUANG, Dongxu, et al. TiDB: a Raft-based HTAP database. Proceedings of the VLDB Endowment, 2020, 13.12: 3072-3084.

[29] Rajaravivarma V . Virtual local area network technology and applications[C]// Symposium on System Theory. IEEE, 1997.

[30] Hirt T . KVM - The kernel-based virtual machine[J]. Proc Linux Symposium, 2010.

[31] Palankar M R , Iamnitchi A , Ripeanu M , et al. Amazon S3 for science grids: a viable solution?[J]. Faculty Publications, 2008.

[32] DK Rensin. Kubernetes - Scheduling the Future at Cloud Scale[J]. 2015.

[33] Amazon Relational Database Service (RDS) website[EB/OL]. 2015. https://aws.amazon.com/rds/.

[34] 云数据库 RDS. website[EB/OL]. 2016. https://help.aliyun.com/document_detail/26092.html.

[35] Cloud Spanner. website[EB/OL]. 2019. https://cloud.google.com/spanner.

[36] CockroachDB Cloud. website[EB/OL]. 2019. https://www.cockroachlabs.com/product/.

[37] TiDB Cloud. website[EB/OL]. 2021. https://en.pingcap.com/tidb-cloud/.

[38] Snowflake Cloud. website[EB/OL]. 2020. https://www.snowflake.com/.

[39] Kyligence Cloud-Native Architecture. website[EB/OL]. 2021. https://kyligence.io/cloud-native-architecture/.

[40] 徐涛. 基于 SaltSatck 的云数据库高可用方案的设计与实现[D].南京邮电大学,2016.

[41] LIU Wanggen. A scheduler system for large-scale distributed data computing in cloud. Big Data Research[J], 2020, 6(1): 2020007-1- doi:10.11959/j.issn.2096-0271.2020007

[42] 严丽云,何震苇,杨新章,张凌,侯韶新.基于 Kubernetes 的容器化数据库及其集群方案[J].电信科学,2018,34(12):163-171.

[43] SunMicrosystems. NFS : Network File System Protocol Specification[J]. Rfc, 1989.

[44] Sharma B , Bansal P , Chugh M , et al. Benchmarking geospatial database on Kubernetes cluster[J]. EURASIP Journal on Advances in Signal Processing, 2021, 2021(1).

[45] Truyen E , Bruzek M , Landuyt D V , et al. Evaluation of Container Orchestration Systems for Deploying and Managing NoSQL Database Clusters[C]// 2018:468-475.

[46] Yan L , Zhenwei H E , Yang X , et al. Containerizing solution based on Kubernetes of database and its cluster[J]. Telecommunications Science, 2018.

[47] Rancher 2.5. website[EB/OL]. 2021. https://rancher.com/docs/rancher/v2.5/en/.

[48] JuiceFS Cloud Service. website[EB/OL]. 2021. https://juicefs.com/docs/cloud/.

[49] Redis. website[EB/OL]. 2015. https://redis.io/documentation.

[50] 对象存储 OSS. website[EB/OL]. 2014. https://help.aliyun.com/document_detail/31817.html.

[51] Prometheus. website[EB/OL]. 2016. https://prometheus.io/docs/introduction/overview/.

[52] AlertManager. website[EB/OL]. 2021. https://prometheus.io/docs/alerting/latest/alertmanager/.

[53] Grafana. website[EB/OL]. 2018. https://grafana.com/grafana/.

[54] Fluentd. website[EB/OL]. 2015. https://www.fluentd.org.

[55] Elasticsearch. website[EB/OL]. 2015. https://www.elastic.co/elasticsearch/.

[56] Kibana. website[EB/OL]. 2016. https://www.elastic.co/kibana/.

[57] ONGARO, Diego; OUSTERHOUT, John. In search of an understandable consensus algorithm. In: 2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14). 2014. p. 305-319.

[58] Google Trends. Website[EB/OL]. 2022. https://trends.google.com/trends/explore?q=公有云,私有云,混合云.

[59] Alvaro L E . S3FS in the wide area. 2011.

[60] JuiceFS CSI Driver. website[EB/OL]. 2016. https://github.com/juicedata/juicefs-csi-driver.

[61] STONEBRAKER, Michael; ROWE, Lawrence A. The design of Postgres. ACM Sigmod Record, 1986, 15.2: 340-355.

[62] BEYER, Betsy, et al. Site reliability engineering: How Google runs production systems. " O'Reilly Media, Inc.", 2016.

[63] Metrics, Tracing, and Logging. website[EB/OL]. 2017. https://peter.bourgon.org/blog/2017/02/21/metrics-tracing-and-logging.html.

[64] Google Trends. Website[EB/OL]. 2022. https://trends.google.com/trends/explore?q=云计算,可观测性

[65] ClickHouse - Fast Open-Source OLAP DBMS. website[EB/OL]. 2014. https://clickhouse.com.

[66] O’NEIL, Patrick E.; O’NEIL, Elizabeth J.; CHEN, Xuedong. The star schema benchmark (SSB). Pat, 2007, 200.0: 50.

[67] Ansible Playbook for ClickHouse Cluster. website[EB/OL]. 2021. https://github.com/dusennn/clickhouse-ansible.

[68] Ssb-Dbgen. website[EB/OL]. 2010. https://github.com/vadimtk/ssb-dbgen.

[69] CMPP. website[EB/OL]. 2021. https://github.com/dusennn/cmpp.

致谢

时间如流水，不舍昼夜，转眼之间，硕士生活即将结束，在这里我收获非常多，我衷心的感谢大家。

古人云....

Xxx

二零二二年二月

攻读硕士学位期间科研情况

■ 已申请的软著

[1] Xxxxv1.0 登记号：Xxxx4

■ 已发表的论文

[1] Xxxxvxxx