基于 Spark 与 MPI 集成的数据分析与处理平台

周梦兵[1,2]，李秋彦[1]，吴欧[3]，王洋[1,2,4]

[1]（中国科学院深圳先进技术研究院，深圳 518055）

[2]（中国科学院大学，北京 100049）

[3]（南京大学软件学院，南京 210093）

[4]（深圳理工大学，深圳 518107）

摘要： 目前以机器学习为代表的人工智能应用负载表现出计算密集型与数据密集型并存的双密特征。这类应用不仅需要支持海量数据的存储、传输与容错，还需优化复杂逻辑计算的性能。传统的单一大数据框架或高性能计算框架已无法应对这类应用带来的挑战。文章提出的基于 Spark 和 MPI 的混合大数据平台是一种高性能大数据处理平台。该平台以典型大规模集群为基础，针对以机器学习为代表的双密型应用的存储和计算特点，重点建设了双范式混合计算、异构存储和融合高性能通信三个模块。针对双密型应用既有数据密集型的大数据处理也有计算密集型的高性能计算的特点，设计了 Spark 范式混合 MPI 范式的计算模块，通过对任务进行分割和分类，将计算密集型任务卸载到 MPI 计算模块，提升双范式混合计算功能。针对双密型应用在计算过程中不同数据的数据特征，设计了异构的存储结构和数据与元数据分离的策略，通过对数据的分型优化存储，构建高可用、高性能的存储系统。针对双密型计算的通信特点，提出高性能通信技术的融合方式，为计算模块和存储模块提供高性能通信支持。测试结果表明，该平台可以为双密型应用提供高效的双范式混合计算，针对不同的计算任务，相较于单一的 Spark 大数据平台性能提升 4.2%至 17.3%。

关键词 双密型应用；大数据处理；高性能计算；混合范式计算

doi：10.12146/j.issn.2095-3135.20241203002

Integrated Data Analysis and Processing Platform Based on Spark and MPI

ZHOU Mengbing[1,2],LI Qiuyan[1],WU Ou[3],WANG Yang[1,2,4]

[1] (Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China)

[2] (University of Chinese Academy of Sciences, Beijing 100049, China)

[3] (Software Institute, Nanjing University, Nanjing 210093, China)

[4] (Shenzhen University of Advanced Technology, Shenzhen 518107, China)

Abstract: Currently, AI application workloads, represented by machine learning, exhibit a dual-density characteristic, combining both compute-intensive and data-intensive traits. These applications not only require support for the storage, transmission, and fault tolerance of massive data but also need to optimize the

收稿时间：2024-12-03 修回时间：2025-04-11

基金项目：第三次新疆综合科学考察（2021xjkk1304）；深圳市承接“人机物融合的云计算架构与平台 ”的产业化应用研究（CJGJZD20230724093659004）;边云智能协同计算方法与面向 C-V2X 的应用，深圳市科技计划项目（深港澳 C 类，No. SGDX20220 530111001003）

作者简介：周梦兵，硕士研究生，研究方向为云计算与大数据分析；李秋彦，硕士研究生，研究方向为并行计算与大数据整合；吴欧，南京大学助理研究员，研究方向为区块链、云计算、排队论、随机调度算法；王洋(通讯作者)，博士，研究员，研究方向为云存储、云计算，E-mail: yang.wang1@siat.ac.cn。

performance of complex logical computations. Traditional single big data frameworks or high-performance computing frameworks can no longer meet the challenges posed by these applications. The hybrid big data platform based on Spark and MPI proposed in this paper is a high-performance big data processing platform. This platform, built on a typical large-scale cluster, focuses on addressing the storage and computing characteristics of dual-density applications, such as those in machine learning, and includes three key modules: dual-paradigm hybrid computation, heterogeneous storage, and integrated high-performance communication. To address the dual-density nature of these applications, which involve both data-intensive big data processing and compute-intensive high-performance computing, a computational module combining the Spark and MPI paradigms is designed. By splitting and classifying tasks, compute-intensive tasks are offloaded to the MPI computation module, enhancing the dual-paradigm hybrid computation capability. To address the different data characteristics during the computation process, a heterogeneous storage structure and a data-metadata separation strategy are designed. This optimizes data storage through classification, building a highly available, high-performance storage system. In response to the communication needs of dual-density computing, a high-performance communication technique is proposed, providing strong communication support for the computing and storage modules. Test results show that this platform provides efficient dual-paradigm hybrid computation for dual-density applications, achieving performance improvements of 4.2% to 17.3% compared to a standalone Spark big data platform for various computation tasks.

Key words: Dual-intensive Applications; Big Data Processing; High Performance Computing; Hybrid Paradigm Computing

1 引 言

以机器学习为代表的人工智能新型大数据应用负载展现出显著的数据密集型和计算密集型双重特性。例如，对深度模型的训练在大量数据密集型的 I/O 操作之后又需要进行大量的张量计算[24][25][26]。这种双密型应用不仅需要处理海量数据，还对计算性能有着严格要求，从而对传统的大数据框架构成了新的挑战。

目前分布式大数据处理框架包括 Hadoop、Spark 和 Flink 等，其中 Hadoop 和 Spark 凭借方便的编程接口以及较高的性能等方面的固有优势，被广泛应用于各种领域[1]。Spark[23] 是基于内存的分布式并行计算框架，RDD（Resilient Distributed Datasets，弹性分布式数据集）作为其特有的数据结构，使其相比于 Hadoop 的 MapReduce 计算框架具备更灵活的表达方式和更高的计算性能[2]。尽管 Spark 作为新一代大数据计算框架，在计算密集型应用中的性能已有显著提升，但相比采用本地代码与优化通信原语的高性能计算框架，如 MPI（Message Passing Interface，消息传递接口）[3]，其性能仍存在较大差距。研究表明，在具有 100 个计算节点的高性能集群上运行大型矩阵分解算法，使用 MPI 比 Spark 快 4.6 到 10.2 倍[4]。另一项关于分布式图算法的研究表明，对于弱连通图，在具有 16 个节点的集群上，使用 MPI 框架可比 Spark 的 GraphX 库快两个数量级[5]。MPI 既是消息传递函数库的标准规范，又是一种消息传递并行程序设计模型。作为一种高性能计算框架，MPI 提供了大量进程之间相互通信和同步等操作的原语，可以实现更高效的并行算法，目前常用于计算密集型应用的计算。然而，MPI 无法像 Spark 那样有效简化资源管理、任务调度、并行处理和容错等复杂操作，并不能很好地应对双密型应用的数据密集这一特征。

总之，目前典型的大数据框架多以支持大数据的高效处理为目标，追求独立任务的高带宽 I/O 操作，缺少对高性能计算的有效支撑。而传统的高性能集群又多以支持科学工作流为主的高性能计算为核心，追求的是相互通信任务的低延迟 I/O 操作，缺少有效的大数据处理架构对高带宽计算的支持。面对双密型应用对存储系统的高容量、高可靠和计算系统的高性能的要求，传统集群都显现出了各自的不足之处。因此，将大数据框架和高性能技术相融合，既保留大数据框架对于大数据的高效处理机制，同时又达到高性能计算技术的性能，是一个颇具前景的手段[6]。

然而由于两种框架应用环境和设计目标的不同，融合趋势还处于发展阶段，相关技术还不够完善，当前还存在着许多亟待解决的挑战[7]。本文在这一背景下，提出了基于 Spark 和 MPI 的混合大数据平台，以充分利用 Spark 大数据计算框架所提供的简易编程接口以及容错等机制和 MPI 高性能计算框架所提供的通信原语带来的性能上的提升，为人工智能应用等双密型应用提供高性能大数据处理支持。该混合大数据平台将用于人工智能应用

数据的高效存储和高性能计算，以提升神经网络的分布式训练和推理性能。

2 研究现状

基于 Spark 和 MPI 的混合大数据平台深度融合大数据处理与高性能计算，构建了强大高效的大数据处理能力。Spark 作为通用内存计算引擎，擅长高效处理大数据，特别适用于数据密集型任务；而 MPI 专注于高性能计算，优化进程间通信，尤其适用于计算密集型任务。两者的融合可充分满足双密型应用的需求，即同时具备数据密集型和计算密集型特性的应用。具体来说，融合架构适用于既需处理大规模数据集又需执行高性能计算任务的场景。本节将对当前国内外基于 Spark 和 MPI 的融合方法进行分类讨论，主要包括 Spark 调用 MPI、扩展 MPI 以实现 Spark 机制、通过第三方集群资源管理模块进行融合，以及其它融合方式。

首先，通过在 Spark 程序执行过程中调用 MPI 库来实现高性能计算是一种常见的融合方法。这种方法能够利用 Spark 强大的数据处理能力与 MPI 在并行计算方面的优势，将计算密集型任务卸载到 MPI 模块中。然而，数据的传输问题是这种方法的关键挑战，尤其是在分布式环境中，通常需要通过文件 I/O、共享内存或 Socket 通信等方式来传递数据。例如，Spark+MPI [8]通过共享内存进行数据传输，从而实现计算任务卸载。而 Alchemist[9][10]则采用了 Socket 通信来传输数据，避免了内存副本的开销，尽管其性能可能会受网络带宽的限制。为了克服这一问题，可以引入更多高性能特性，例如 RDMA（Remote Direct Memory Access，远程直接数据存取）等高性能计算集群中的网络通信技术[11][12]。

其次，通过扩展 MPI 来实现 Spark 的数据处理机制是另一种融合方法，以避免集群间的通信开销和复杂的序列化操作[13]，从而显著提升计算性能。但该方法通常需要大量额外代码来实现大数据框架的功能，开发复杂度较高，且可能与现有高性能计算应用存在兼容性问题。DataMPI[14] 是该方向的代表性工作，它通过 MPI 实现大数据框架的通信机制，并结合线程并行化技术，实现计算与通信的高效重叠，从而提升整体性能。与 Hadoop 相比，DataMPI 在处理计算密集型和数据密集型任务时均展现出显著的性能优势。但其优化主要集中在通信层，尚未全面解决大数据框架中的任务管理与容错机制问题。

为降低开发复杂度，另一些工作选择在不重写大数据框架全部特性的前提下实现融合：例如，Chukonu[15] 采用本地语言重写 Spark 的计算引擎以提高效率，同时复用 Spark 已有的成熟特性；MPI4Spark[16]则将 MPI 通信机制嵌入 Spark 框架，用于替代其原生通信，特别是在计算密集型任务的 shuffle 阶段，显著缓解了性能瓶颈；而 JAMPI [27]以算法插件形式将 MPI 集成到 Spark 中，在保持框架改动最小化的同时，实现一定程度的性能提升，尽管其灵活性相对较低。

另一种可行方案则利用第三方集群资源管理模块进行融合，该方案避免了对 Spark 和 MPI 框架的深度修改，从而减少了兼容性问题。这种方法通过集群资源管理器（如 Slurm 或 Hydra）来管理 Spark 和 MPI 之间的通信和资源分配。Spark-MPI 架构[17][18]是这一思路的具体体现，它利用集群资源管理器的进程启动与信息交换功能，在几乎不改变原有框架的情况下实现 Spark 和 MPI 的融合。尽管该方法简化了实现，但引入的第三方依赖可能带来可移植性问题，同时该结构仍需面对数据序列化等问题。

除中间件层的融合方法外，当前还存在一些应用层与底层的融合方案。在中间件层方案中，大数据负载通常采用数据转换编程框架，而高性能计算负载则依赖并行编程模型。二者在融合过程中需协调不同应用组件之间的数据传输与执行逻辑，增加了系统设计的复杂性。此外，该类方案在部署和运行阶段需要同时配置并维护大数据与高性能计算环境，操作繁琐、成本较高。为缓解上述复杂性，Mammadli 等人提出了一种基于任务的编程模型[19]，用于统一协调双密型应用中的数据与计算任务，虽然简化了系统集成，但在低层次性能优化方面存在一定牺牲。另一方面，底层的存储系统适配性问题也是融合系统面临的关键挑战之一。大数据处理更关注存储系统的高吞吐能力[20]，而高性能计算则要求 I/O 的整体均衡性与稳定性。为同时满足这两类需求，国家并行计算工程技术研究中心提出了并行存储系统的解决方案[21]，以提升融合系统在存储层面的性能表现与适配能力。

通过对上述主要融合方向的探讨可以看出，Spark 与 MPI 的深度融合在提升大数据处理与高性能计算整体效率方面展现出巨大潜力，为双密型应用提供了切实可行的解决方

案。然而，现有研究仍存在诸多亟待突破的局限性。为此，本文提出了一套基于 Spark 与 MPI 集成的全新数据分析与处理平台。在实现过程中，我们着重在实现复杂度与系统性能之间寻求平衡，选择了中间件层的第一种融合方法，即在 Spark 程序中调用 MPI 模块。针对数据通信与序列化过程中存在的性能瓶颈，本文提出了多项创新性解决方案，包括高效的数据分类传输机制、新型数据序列化方法，以及多项前沿技术的融合实现。围绕计算、存储与通信三大核心模块，本文对平台进行了系统性优化，全面满足双密型应用的高性能需求，有效提升了资源利用率与计算效率。

3 融合框架设计

本节主要介绍系统的整体架构设计与模块划分，包括融合系统的各个组成模块以及模块间的协作关系设计等，这些设计是建立合理可用且满足需求的融合平台的基础。

3.1 系统概要

基于 Spark 和 MPI 的混合大数据平台旨在以典型大规模存算一体集群为基础，融合并集成现有的数据存储与计算技术，针对双密型应用的存储计算特点，打通大数据存储与高性能计算之间的壁垒，构建高可用、高并发的存储与计算一体化的高性能大数据计算平台。为了应对双密型应用中大量数据带来的计算挑战，该高性能大数据计算平台以融合 Spark 和 MPI 各自特点的计算范式为核心进行设计。通过这一融合，平台既能够充分发挥 Spark 提供的可靠用户界面和成熟的大数据处理能力（包括高效的数据存储管理和容错机制），又能利用 MPI 在复杂计算中的高效性能，从而显著降低大数据计算中的延迟问题。

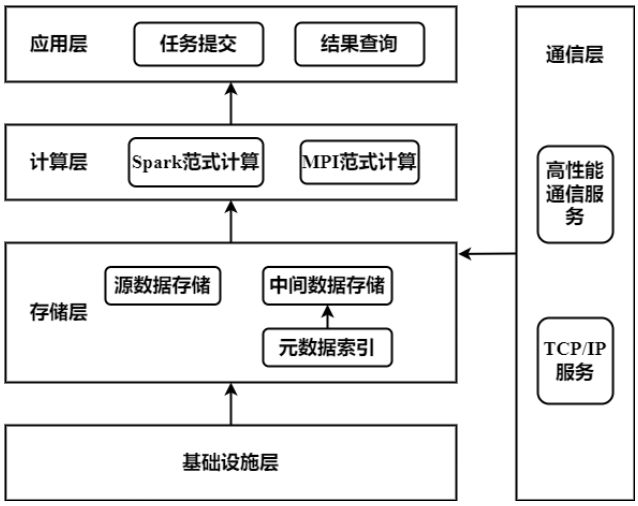

图 1　基于 Spark 和 MPI 的混合大数据平台架构图

Fig. 1　Architecture of the hybrid big data platform based on Spark and MPI

该平台的架构图如图 1 所示。平台采用分层的架构，从上到下分别为应用层、计算层、存储层、通信层和基础设施层。应用层主要提供用户接口，实现与用户的交互；计算层主要提供双范式计算功能；存储层则根据计算过程中产生的不同数据的特点，提供异构存储功能；通信层则为存储和计算提供通信保障；基础设施层则是具体的硬件集群设施，为存储、计算和通信提供硬件支持。

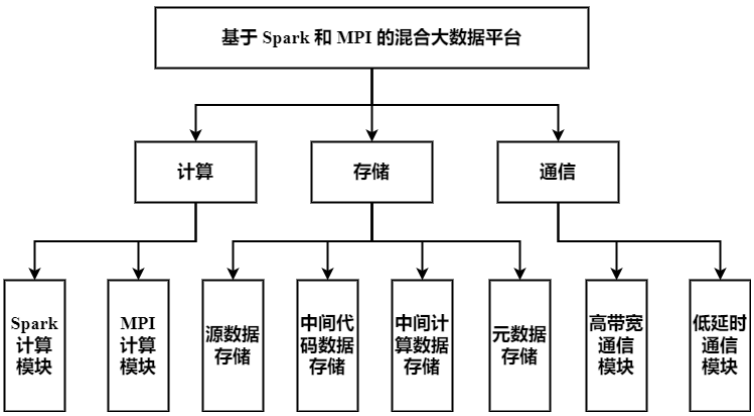

图 2　**系统功能模块图**

Fig. 2　System functional modules

本文的设计主要围绕系统的计算层、存储层和通信层，分别实现双范式计算功能、异构存储功能以及高性能混合通信功能。为此，本文将针对计算、存储和通信三个部分的设计与实现进行详细阐述。其中，计算层进一步划分为 Spark 计算模块和 MPI 计算模块；通信层细分为高带宽通信模块和低延时通信模块；存储层则包括源数据存储模块、中间代码数据存储模块、中间计算数据存储模块以及元数据存储模块。系统的模块化结构如图 2 所示。

3.2 计算模块

计算模块是混合大数据平台中最重要的一个模块，重点是针对双密型应用的数据密集性和计算复杂性，完成一套结合 Spark 和 MPI 双范式的一体化高性能数据计算系统。为此，计算模块的核心思路是在运行过程中根据预先判定的任务类型，分别采用 Spark 或 MPI 计算范式，优化任务调度，提升资源利用效率，最大化双密型应用的数据处理性能。

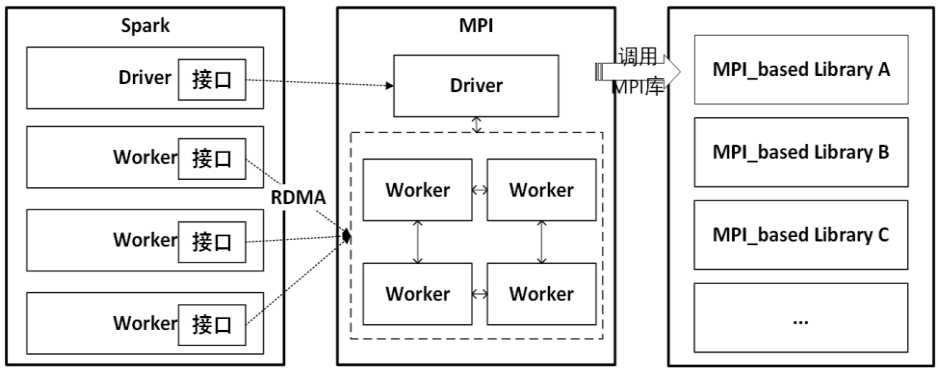

图 3　**计算模块整体架构图**

Fig. 3　Overall architecture of the computing module

计算模块的整体架构如图 3 所示，采用了 Spark 计算范式与 MPI 计算范式的水平融合策略。在 Spark 程序执行过程中，调用 MPI 库以实现高性能计算，需要克服以下三个关键难点：

1. 数据通信：水平融合过程中，两个范式相互调用时需要进行数据通信。通信的数据包括分布式数据和非分布式数据，需要实现两种数据的高性能传输方式。

2. 数据序列化：由于两个范式计算时所使用的的数据结构具有独特性，数据传输之前需要将数据先进行序列化。而高性能计算数据类型繁多，如何实现统一的序列化方式是一个需要解决的难点。

3. 调用时机：用户正常提交了作业之后，需要在脱机情况下实现自动调用。

对于数据通信问题，集群间数据通信一般有通过共享文件、通过共享内存和通过

Socket 通信三种方式，并且这三种方式各有利弊。通过共享文件进行数据通信，可以很容易的实现数据持久化和容错，而且文件系统具有完备的命名空间树，简化了对于共享数据的管理，但基于磁盘存储的共享文件造成计算过程中额外的磁盘 IO 次数，降低了性能。通过共享内存进行数据通信，可以获得更高的 IO 性能，缓解数据通信的性能瓶颈问题，但内存容量有限，需要额外设计合理的内存分配算法和数据存储接口，造成了计算过程的额外开销。通过 Socket 通信可以避免多余的磁盘 IO 操作，而且可实现点对点的分块数据通信，但 Socket 通信也会受网络带宽影响。因此，本文的计算模块采用了内存分布式文件存储结合高性能通信技术的混合数据通信方式。

Spark 计算范式依赖 Java 虚拟机（JVM）运行，以 RDD 为核心数据结构，而 MPI 计算范式位于更底层，主要采用 C++语言编程。因此，在两个计算模块间进行数据交互时，首先需要进行数据序列化处理，即将 Spark 中的数据转换为 MPI 能够解析的格式，数据转化时需要考虑：(1) 计算密集型应用通常依赖分层数据格式来组织文件和数据集（例如主流的 HDF5[22]文件和 netCDF 数据[29]），不能简单地当成字符串进行序列化。(2) 数据密集型应用数据一般具有多元异构的特征，将大数据框架数据格式转化为所有数据密集型应用所能处理的格式几乎是不可能的[28]。(3) 需要存储的数据与中间数据存储模块存储格式存在一定程度的不匹配。针对问题 (1) 和 (2)，本文采用模板的方式实现数据序列化，具体的序列化方法由用户自定义，并通过参数传递给系统。模板模式将序列化操作与框架分离，使用户能够设计满足不同应用需求的序列化方法，从而既解决了异构数据的序列化问题，又在性能优化上具有针对性，减少序列化操作带来的性能损耗。针对问题 (3)，数据序列化采用两段式的结构来实现，首先将 Spark 数据结构转化为中间数据存储模块可以识别的格式；其次，再将存储模块中的数据转化为适配 MPI 计算的数据格式。该方法一方面通过拆解 Spark 与 MPI 的数据格式，减少不必要的序列化方法定义，提升代码的可重构性；另一方面也兼顾存储模块存储格式的优化。

融合模块旨在解决双密型应用中的双范式融合计算问题。在脱机批处理过程中，计算模块需要有效地进行任务调度。为此，本平台采用预先定义的工作流，结合任务分割与任务类型判断，实现任务在运行过程中的自动化分型调度。具体而言，任务分割依赖于人工预先定义的工作流，该工作流基于任务的计算逻辑和数据依赖关系进行合理划分。在任务类型判断方面，本平台通过人工规则对任务进行分类，结合任务的计算逻辑、数据传输量及计算复杂度，来判断每个子任务是数据密集型还是计算密集型。通过静态分析任务的计算公式和数据传输模式，并结合历史运行数据及统计模型，足以使我们正确识别出任务的主要瓶颈是数据处理还是计算处理，从而准确分类任务。数据密集型任务侧重于大规模数据存储与传输，而计算密集型任务则集中于复杂的计算过程。基于任务分类结果，本平台将数据密集型任务调度至 Spark 计算模块，计算密集型任务调度至 MPI 计算模块。通过这种基于工作流的任务分类与调度方式，平台能够实现更高效的计算资源分配与性能优化。

最终，计算模块的整体执行流程可概述如下：（1）用户提交作业到 Spark 计算模块；（2）Spark 计算模块加载作业后，根据作业需求初始化 Spark 计算环境；（3）将作业拆分为多个独立任务，并保存至任务集中；（4）从任务集中选取一个任务，判断其类型：若为数据密集型任务，调用 Spark 范式执行；若为计算密集型任务，调用 MPI 模块执行 MPI 范式计算；（5）在执行 Spark 范式计算前，模块通过通信接口从源数据存储模块获取所需数据，随后根据任务需求生成 RDD 并完成计算；（6）在执行 MPI 范式计算前，需完成调用准备，包括将任务代码保存至中间代码存储模块，将计算数据存储至中间数据存储模块，并为结果返回分配存储空间；（7）MPI 模块启动后，加载任务代码并通过任务 ID 从元数据存储模块提取计算所需的元数据信息；（8）MPI 模块依据元数据信息与 Spark 节点建立高效数据通路以获取相关数据；（9）获取数据后，MPI 模块正式执行计

算，结果存储在分配的结果返回地址中；（10）MPI 模块完成计算并退出计算环境；（11）Spark 计算模块继续从任务集中选取任务，进入新一轮类型判断与调度。

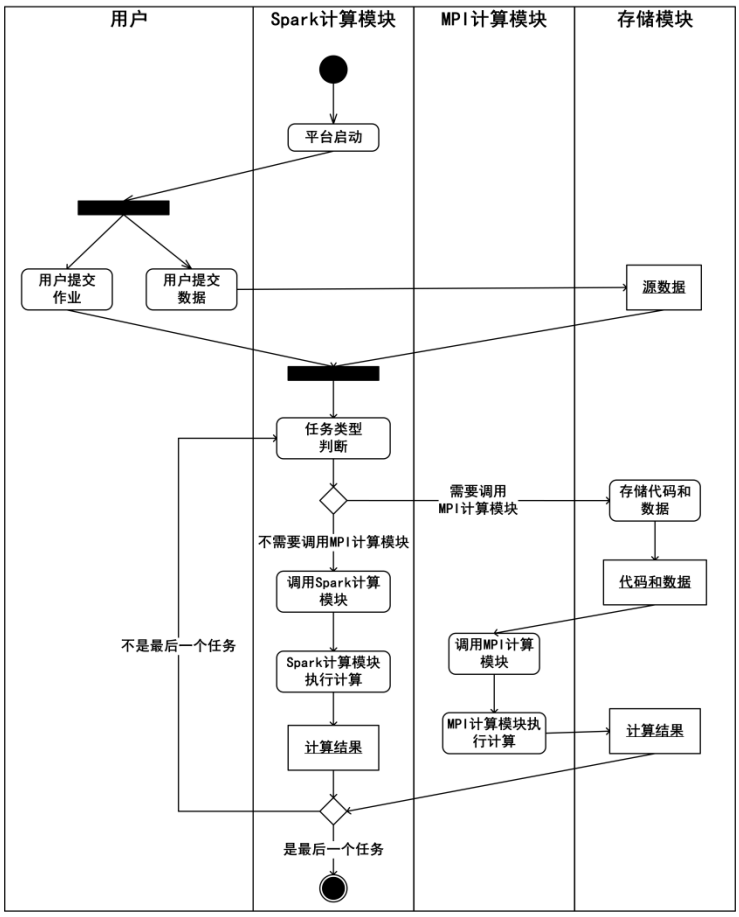

图 4　基于 Spark 和 MPI 的混合大数据平台活动图

Fig. 4　Workflow of the hybrid big data platform integrating Spark and MPI

3.4 存储模块

存储模块是实现存算一体化的核心基础，为双密型应用提供高性能的大数据存储服务。为了适应混合计算范式的需求，存储模块设计为异构存储系统，根据双密型应用计算过程中不同数据类型的特性，提供针对性的存储方式，在吞吐量和延时之间取得平衡，从而最大化存储性能。

传统分布式文件系统 HDFS 相较高性能计算中常用的并行文件系统在速度上存在显著差距。为解决这一问题，存储系统将中间计算数据重定向至内存，实现了读写速度的显著提升和性能的优化。该存储系统以 HDFS 为核心，结合分布式内存文件系统，从数据安全性与高性能读写两个维度满足双密型应用的需求。这两类文件系统分别对应源数据存储模块和中间数据存储模块。

中间数据按照特性分为静态数据和动态数据，因此中间数据存储模块进一步划分为中间代码（静态）数据存储模块和中间计算（动态）数据存储模块。静态和动态数据的独立存储不仅能够更好地适配各自的使用场景，还有效提升了内存空间的利用效率。

为了进一步优化存储系统的整体读写性能，设计中引入了数据与元数据分离的存储策略，同时增加了元数据存储模块。元数据存储模块通过 Redis 数据库，针对双密型应用计算过程中产生的海量小文件，显著提升了读写性能并提高了内存空间的利用率。

具体来说，存储模块被细分为源数据存储模块、中间数据存储模块（包括代码数据存储模块和计算数据存储模块）以及元数据存储模块。下文将详细说明各子模块的功能与实现。

3.4.1 源数据存储模块

源数据存储模块主要负责保存用户提交计算任务中涉及的原始数据。这些数据通常未经处理，具有杂乱无序且数据量巨大的特点，因此需要源数据存储模块具备高吞吐量和良好的容错能力。为此，该模块基于传统 HDFS 大数据文件系统实现，采用分布式存储架构，组织为客户端-服务器模式，如图 5 所示。

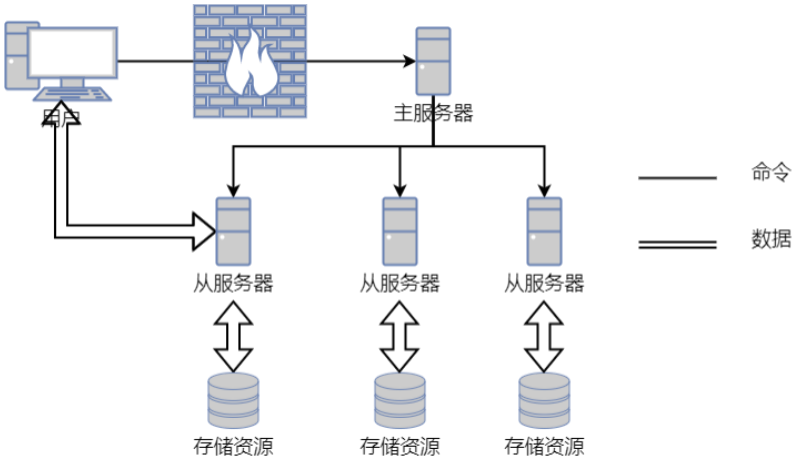

图 5　源数据存储模块集群架构图

Fig. 5　Cluster architecture of source data storage module

通过客户端-服务器模式，将命令处理与数据传输任务分离，可以有效降低存储模块服务器的负担，缓解单点瓶颈问题，并提升整体存储容量和性能。同时，该模块采用主从分离的集群架构，支持从节点的动态加入和退出，从而增强了系统的可扩展性。通过这种设计，混合大数据平台能够根据实际应用需求和集群规模，灵活调整源数据存储容量，以满足多样化场景的要求。

3.4.2 中间数据存储模块

中间数据存储模块是存储系统中的关键模块，是 Spark 与 MPI 计算范式能够融合的关键。其主要功能是存储代码以及调用 MPI 计算模块之前、预处理之后的序列化数据，这部分数据根据自身特点，分为静态数据和动态数据两类。

MPI 计算模块需要的计算任务代码，一经提交就不会再更改，且需要 MPI 集群中所有涉及到该任务的节点所共享，这类数据为静态数据。中间代码数据存储模块主要实现数据共享功能，提供发布共享、删除共享和打开共享三个接口。在预处理工作阶段，Spark 计算模块通过发布共享文件接口，向服务器发出共享文件请求。服务器接受请求之后从 Spark 计算模块读取该文件并保存到服务器节点所在的文件系统，之后向 Spark 计算模块回复发布成功通知。MPI 计算模块启动之后，根据调用时传入的参数信息，通过打开共享文件接口，向服务器发出读取共享文件请求。服务器接到请求之后，将该文件发送给 MPI 计算模块。任务完成之后，Spark 计算模块可以通过删除共享文件接口删除服务器中存储的共享文件。

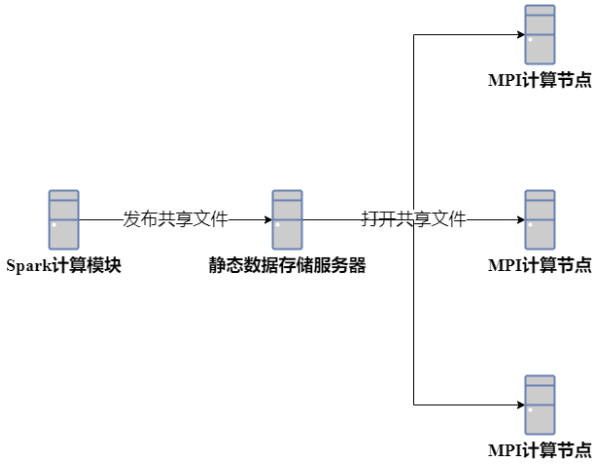

图 6　中间代码数据存储模块集群架构图

Fig. 6　Cluster architecture of intermediate code data storage module

Spark 计算范式运行在 JVM 上，主要采用 RDD 作为数据结构；而 MPI 计算范式在更低水平运行，以 C++ 作为主要编程语言。因此，Spark 计算模块在将计算数据传递给 MPI 计算模块之前，需先对数据进行序列化，并为序列化数据开辟专用存储空间。这些数据属于动态数据，具有临时性和突发性访问的特点。中间计算数据存储模块通过第三方分布式内存文件系统实现，其主要优点包括：1、内存文件系统在内存中构建，断电后数据消失，符合动态数据的临时性需求；2、相较于磁盘，内存提供更高的 IO 性能，更能适应计算模块的高速处理需求；3、分布式架构分散数据访问压力，通过负载均衡优化数据分布，能有效应对动态数据的突发性访问；4、数据以文件形式组织，利用树形目录结构管理分布式数据，高效简化数据管理，减少额外的数据访问开销。考虑到我们的测试规模较小、节点数有限且对高可用性要求不高，在原型系统中，我们采用了较为轻量的方案：将各节点的内存通过挂载 tmpfs 构建本地内存文件系统，并通过 NFS（Network File System，网络文件系统）将其共享，初步实现了集中式的内存文件共享功能。这一组合在简化部署的同时，基本满足了系统对临时、高速数据交换的性能需求。在未来的实际生产部署中，该方案可无缝替换为成熟的分布式内存文件系统（如 Alluxio 或 Ignite IGFS），以进一步提升系统的可扩展性与稳定性。

中间数据存储根据不同的数据特征，设计了非分布式和分布式的存储方式，合理地存储静态和动态中间数据，结合通信模块可以获得高效的存储性能。

3.4.3 元数据存储模块

元数据存储模块作为存储系统的重要补充，主要为融合架构中的存储系统提供元数据管理服务。本平台的融合架构在存储层面需解决两个关键问题：（1）中间计算数据存储模块采用分布式文件系统结构，由服务器进行数据管理，因此需要记录数据与物理节点间的映射；（2）由于负载均衡的实现方式，每次执行任务的 Spark 节点和 MPI 节点事先未知。在这种情况下， MPI 节点每次都需要与所有的 Spark 节点连接以判断是否存在自己所需的数据。随着集群规模扩大，连接数量迅速增加，导致极大的带宽浪费和中间数据存储性能降低。因此，必须优化连接策略，针对性地建立连接。

针对第一个问题，平台设计中让元数据存储模块辅助中间计算数据存储模块，为其提高元数据服务。元数据存储模块采用 Redis 数据库，可以高效处理中间存储的高频访问和短期特性。对于第二个问题，提出分别存储数据与元数据的策略：在 MPI 节点读取计算数据之前，由元数据存储模块为其提供元数据信息，再通过这些元数据定位所需数据对应的存储节点，精准建立通信连接。

这一设计方案具有多方面优势。Redis 数据库将数据存储在内存中，以提供快速的读写访问速度,可为中间数据存储模块的 IO 需求和计算模块的高速计算提供有效支持，避免了存储性能瓶颈。Redis 提供了持久化机制，可以保证 Redis 在遇到异常情况时的数据安全性和可靠性；其 IO 多路复用和事件驱动机制能够高效支持并发访问，有效应对中间存储的突发式访问；此外，Redis 可组织为高可用集群，避免服务器单点失效，提升系统整体可靠性。

3.4 通信模块

通信模块的主要功能是为混合大数据平台中的计算模块与存储模块之间，以及计算模块内部各计算节点之间提供通信服务。其具体实现细节分散在其他模块的实现中。通信模块提供高吞吐和低延迟两种通信模式，根据模块间的需求选择合适的模式，以兼顾吞吐量与性能要求。

Spark 计算模块的内部通信主要发生在模块启动阶段和执行 Spark 范式计算阶段，主要采用高吞吐通信模式以支持批处理计算。MPI 计算模块的内部通信则集中于模块启动和执行 MPI 范式计算阶段，采用低延迟通信模式以满足并行计算需求。Spark 计算范式与 MPI 范式的融合依赖于两者之间的数据传输，这是实现协同计算的核心，模块间需要传输的数据可以分为分布式和非分布式两类，两类数据需要不同的传输方式。对于非分布式数据，采用建立高吞吐通信的方式进行传输。对于分布式数据，通常存在文件 IO、共享内存和 Socket 通信三种数据传输方式，但是这三种方式都不能很好的满足本系统的需求。若采用文件 IO 的方式，即输入端将数据以分布式文件形式存入文件系统，接收端根据索引信息从文件系统中读取数据，大量与文件系统的交互操作会降低系统的整体性能。若采用共享内存的方式，会在内存中产生大量的数据副本，造成大量不必要的内存开销。若采用 Socket 通信的方式，会进行数据的多次封装与解封装，造成计算资源的浪费，当数据量达到 GB 甚至 TB 级别之后，通信时延会大大增加，使得数据通信成为整个系统的瓶颈。因此，分布式数据传输采用通信模块支持的低延迟模式（内存分布式文件系统结合低延迟连接）进行处理。

如前所述，Spark 计算模块内部通信、非分布式数据在 Spark 计算模块与 MPI 计算模块之间的传输，以及计算模块与存储模块之间的通信均采用高吞吐模式。相比之下，Spark 计算模块与 MPI 计算模块之间的分布式数据传输和 MPI 计算模块内部通信则使用低延迟模式，以满足性能需求。

4 结果分析与评估

3.4 测试环境与评价指标

本实验在由三台服务器组成的集群上进行开发与测试，该集群包括计算集群（Spark 计算集群和 MPI 计算集群）以及存储集群（数据库集群）。在测试环境中，存储与计算集群合并为一体，而在实际部署中，可根据数据规模的增长对存储与计算资源进行解耦。

每台服务器配备 2 颗 Intel Xeon E5-2630 v3 处理器，提供 16 核 32 线程的计算能力，并搭载 64GB（4 × 16GB）DDR4 R-ECC 内存。存储部分由 5 块 5TB SATA 硬盘组成，总容量达 25TB。网络环境采用 1Gbps 以太网及 4 路 25Gbps RDMA（RoCE）链路。Spark 计算集群、MPI 计算集群及数据库集群的软件环境见表 1。

表 1　配置环境

Table 1　Configuration Environment

集群	Spark 计算集群	MPI 计算集群	数据库集群
软件	操作系统：CentOS 7.5	操作系统：CentOS 7.5	操作系统：CentOS 7.5

环境	编译环境：jdk8、scala 2.12.12 集群环境：spark2.4.3、hdfs2.7.3	编译环境：gcc、g++、gfortran 9.3.1 集群环境：mpich-3.4.3	数据库：redis 3.0.4

本小节检测基于 Spark 和 MPI 的混合大数据平台与传统的大数据计算框架 Spark 相比，对于不同类型的任务的计算性能提升。性能提升通过响应时间提升百分比这一量化的参数来表征。响应时间提升百分比计算公式为公式（1）。

$$\mu = \frac{T-t}{T} \times 100\% \quad ...(1)$$

其中 μ 表示响应时间提升百分比，T 表示任务在 Spark 集群上的响应时间，t 表示任务在基于 Spark 和 MPI 的混合大数据平台上的响应时间。

3.4 实验结果

本文选取了线性代数和图论中比较基础和经典的算法来测试本平台相较于 Spark 的性能表现。我们分别在相同规模的 Spark 集群和混合大数据平台运行矩阵乘法算法、奇异值分解算法和单源最短路径算法，并统计平台对各个算法的响应时间，对应的测试方案见表 2。

表 2　计算能力测试方案

Table 2　Computational performance testing plan

测试方案	测试算法	测试数据
方案一	矩阵乘法算法：计算两矩阵的乘积，采用标准复杂度的矩阵乘法实现。	随机生成的密集型矩阵，两个输入矩阵的维度均为$10^4 \times 10^4$，元素类型为浮点型，取值范围为[0,100]。
方案二	奇异值分解 (SVD) 算法：对矩阵执行奇异值分解，将输入矩阵A分解为$A = U\Sigma V^T$	随机生成的密集型矩阵，输入矩阵的维度为$10^4 \times 10^4$，元素类型为浮点型，取值范围为[0,100]。
方案三	单源最短路径 (SSSP) 算法：在有向加权图上计算给定源点到所有其他节点的最短路径，采用 Dijkstra 算法进行测试。	随机生成的有向加权图，节点数量为10^4，边数量为1.5×10^6，边权重类型为浮点型，取值范围为[0,100]。

为了避免随机因素的干扰，每个测试用例测试三次，并取结果的平均值。各个方案的测试结果分别如表 3 所示。从不同计算密集型任务占比所获得的性能平均提升结果来看，针对所选取的三种测试算法，随着计算密集型任务占比的增加，基于 Spark 和 MPI 的混合大数据平台对于性能的提升就随之增加，因此该平台更适合于双密型应用的计算。另外，对于所选取的算法来说，当计算密集型任务占比达到 50% 时，性能提升都达到了 10% 以上。

表 3　测试结果

Table 3　Test results

测试方案	计算密集型任务占比	响应时间提升百分比
方案一	20%	5.1%

	50%	10.4%
	80%	17.3%
方案二	20%	4.2%
	50%	11.8%
	80%	15.2%
方案三	20%	6.9%
	50%	11.0%
	80%	14.6%

6 结 论

面对双密型应用对存储系统高容量、高可靠性以及计算系统高性能的需求，单一的大数据处理系统或高性能计算方式都存在一定的局限性。因此，本文旨在将传统的大数据计算框架与高性能计算框架相结合，充分利用大数据计算框架所提供的简易编程接口，以及容错等机制和高性能计算框架所提供的通信原语所带来的性能提升，构建一个高性能大数据处理平台，以支持双密型应用的存储和计算。在这一背景下，本文提出了基于 Spark 和 MPI 的混合大数据平台，旨在为人工智能应用提供高性能计算支持，提升深度学习大模型在大规模数据集上的分布式训练效率。最后，本文在实际环境中对该系统进行了系统性测试。测试结果表明，该平台能够为用户提供大数据存储和双范式混合计算功能，并在计算性能上获得显著提升，可有效优化实际环境中双密型应用的计算。

参考文献

[1] Mostafaeipour, A., Jahangard Rafsanjani, A., Ahmadi, M., & Arockia Dhanraj, J. (2021). Investigating the performance of Hadoop and Spark platforms on machine learning algorithms. The Journal of Supercomputing, 77(2), 1273–1300. https://doi.org/10.1007/s11227-020-03328-5

[2] 何海林, 皮建勇. 大数据处理平台比较与分析[J]. 微型机与应用, 2015, 34(11): 7-9.

He Hailin, Pi Jianyong. Comparison and analysis of big data processing platforms[J]. Microcomputer and Application, 2015, 34(11): 7-9.

[3] Walker D W, Dongarra J J. MPI: a standard message passing interface[J]. Supercomputer, 1996, 12: 56-68.

[4] GITTENS A, DEVARAKONDA A, RACAH E, et al. Matrix factorizations at scale: A comparison of scientific data analytics in spark and c+ mpi using three case studies[C]//2016 IEEE International Conference on Big Data (Big Data). IEEE, 2016: 204-213.

[5] SLOTA G M, RAJAMANICKAM S, MADDURI K. A case study of complex graph analysis in distributed memory: Implementation and optimization[C]//2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2016: 293-302.

[6] MEI S, GUAN H, WANG Q. An overview on the convergence of high performance computing and big data processing[C]//2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2018: 1046-1051.

[7] USMAN S, MEHMOOD R, KATIB I. Big data and hpc convergence for smart infrastructures: A review and proposed architecture[J]. Smart Infrastructure and Applications: Foundations for Smarter Cities and Societies, 2020, 2020: 561-586.

[8] ANDERSON M, SMITH S, SUNDARAM N, et al. Bridging the gap between hpc and big data frameworks[J]. Proceedings of the VLDB Endowment, 2017, 10(8): 901-912.

[9] GITTENS A, ROTHAUGE K, WANG S, et al. Alchemist: An apache sparkff mpi interface[J]. Concurrency and Computation: Practice and Experience, 2019, 31(16): e5026.

[10] GITTENS A, ROTHAUGE K, WANG S, et al. Accelerating large-scale data analysis by offloading to high-performance computing libraries using alchemist[C]//Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018: 293-301.

[11] ZHANG Z, LIU Z, JIANG Q, et al. Rdma-based apache storm for high-performance stream data processing[J]. International Journal of Parallel Programming, 2021, 49(5): 671-684.

[12] 涂晓军, 孙权, 蔡立志. RDMA 技术在数据中心中的应用研究[J]. 计算机应用与软件, 2021, 38(3): 22-25.

Tu Xiaojun, Sun Quan, Cai Lizhi. Research on the application of RDMA technology in data centres[J]. Computer Applications and Software, 2021, 38(3): 22-25.

[13] LI Z. Geospatial big data handling with high performance computing: Current approaches and future directions[J]. High Performance Computing for Geospatial Applications, 2020, 2020: 53-76.

[14] LU X, LIANG F, WANG B, et al. Datampi: extending mpi to hadoop-like big data computing [C]//2014 IEEE 28th International Parallel and Distributed Processing Symposium. IEEE, 2014: 829-838.

[15] YU B, FENG G, CAO H, et al. Chukonu: a fully-featured high-performance big data framework that integrates a native compute engine into spark[J]. Proceedings of the VLDB Endowment, 2021, 15(4): 872-885.

[16] AL-ATTAR K, SHAFI A, ABDULJABBAR M, et al. Spark meets mpi: Towards highperformance communication framework for spark using mpi[C]//2022 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2022: 71-81.

[17] MALITSKY N, CHAUDHARY A, JOURDAIN S, et al. Building near-real-time processing pipelines with the spark-mpi platform[C]//2017 New York Scientific Data Summit (NYSDS). IEEE, 2017: 1-8.

[18] MALITSKY N, CASTAIN R, COWAN M. Spark-mpi: approaching the fifth paradigm of cognitive applications[A]. 2018.

[19] MAMMADLI N, EJARQUE J, ALVAREZ J, et al. Dds: integrating data analytics transformations in task-based workflows[J]. Open Research Europe, 2022, 2: 66.

[20] CHEN Q, CHEN K, CHEN Z N, et al. Lessons learned from optimizing the sunway storage

system for higher application i/o performance[J]. Journal of Computer Science and Technology, 2020, 35: 47-60.

[21] 何晓斌, 蒋金虎. 面向大数据异构系统的神威并行存储系统[J]. 大数据, 2020, 6(4): 30-39.

He Xiaobin, Jiang Jinhu. A Shineway parallel storage system for big data heterogeneous systems[J]. Big Data, 2020, 6(4): 30-39.

[22] KRIJNEN T, BEETZ J. An efficient binary storage format for ifc building models using hdf5 hierarchical data format[J]. Automation in Construction, 2020, 113: 103134.

[23] ZAHARIA M, XIN R S, WENDELL P, et al. Apache spark: a unified engine for big data

processing[J]. Communications of the ACM, 2016, 59(11): 56-65.

[24] Korkmaz E. Adversarial robust deep reinforcement learning requires redefining robustness[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2023, 37(7): 8369-8377.

[25] Chen A, Gisolfi N, Dubrawski A. Data-Driven Discovery of Design Specifications (Student Abstract)[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2024, 38(21): 23449-23450.

[26] Goyal S, Maini P, Lipton Z C, et al. Scaling Laws for Data Filtering--Data Curation cannot be Compute Agnostic[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2024: 22702-22711.

[27] FOLDI T, VON CSEFALVAY C, PEREZ N A. Jampi: Efficient matrix multiplication in spark using barrier execution mode[J]. Big Data and Cognitive Computing, 2020, 4(4): 32.

[28] LIU J, RACAH E, KOZIOL Q, et al. H5spark: bridging the i/o gap between spark and scientific data formats on hpc systems[C]//Cray user group. 2016.

[29] Rew R, Davis G. NetCDF: an interface for scientific data access[J]. IEEE Computer Graphics and Applications, 1990, 10(4): 76-82.