第 14 卷 第 4 期 集 成 技 术 Vol. 14 No. 4
2025 年 7 月 JOURNAL OF INTEGRATION TECHNOLOGY Jul. 2025

引文格式：
周梦兵, 李秋彦, 吴欧, 等. 基于 Spark 与 MPI 集成的数据分析与处理平台 [J]. 集成技术, 2025, 14(4): 106-119.
Zhou MB, Li QY, Wu O, et al. Integrated data analysis and processing platform based on Spark and MPI [J]. Journal of Integration Technology, 2025, 14(4): 106-119.

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呈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、传输和容错，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。本文提出一种基于 Spark 和 MPI 的高性能混合大数据处理平台。该平台基于典型的大规模集群构建，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等 3 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，本文设计了基于 Spark 和 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，从而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类型数据的特征，本文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，构建了高性能存储系统。针对双密型计算的通信特点，本文提出一种高性能通信技术的集成方式，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，该平台可为双密型应用提供高效的双范式混合计算，与单一的 Spark 大数据平台相比，各种计算任务的性能提升了 4.2%～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
中图分类号 TP301.6 **文献标志码** A doi: 10.12146/j.issn.2095-3135.20241203002
CSTR: 32239.14.j.issn.2095-3135.20241203002

收稿日期：2024-12-03 **修回日期**：2025-04-11
基金项目：第三次新疆综合科学考察项目 (2021XJKK1300)；深圳市承接“人机物融合的云计算架构与平台”的产业化应用研究项目 (CJGJZD20230724093659004)；深圳市科技计划项目 (SGDX20220530111001003)
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，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。

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*)

[*]*Corresponding Author*: *yang.wang1@siat.ac.cn*

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 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 3 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 characteristics of different types of data during the computing process, a heterogeneous storage structure and a data-metadata separation strategy are designed. This optimizes data storage through classification, building a high-performance storage system. In response to the communication needs of dual-density computing, this paper proposes an integration approach that combines high-performance communication techniques, 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.

Keywords dual-intensive applications; big data processing; high performance computing; hybrid paradigm computing

Funding This work is supported by 3rd Xinjiang Scientific Expedition Program (2021XJKK1300); Industrial Application Research Project of Shenzhen for “Cloud Computing Architecture and Platform for Human-Machine-Thing Integration” (CJGJZD20230724093659004); and Shenzhen Science and Technology Plan Project (SGDX20220530111001003)

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与计算密集型并存的双重特性。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[1-3]。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，因此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和较高的性能等固有优势被广泛应用于各种领域[4]。Spark[5]是一种基于内存的分布式并行计算框架，其特有的数据结构——弹性分布式数据集 (resilient distributed datasets，RDD) 为其提供了比 Hadoop 的 MapReduce 框架更灵活的编程模型和更高的计算性能[6]。作为新一代大数据计算框架，Spark 在计算密集型应用中的性能虽然已显著提升，但与采用本地代码执行，并通过优化通信原语加速的高性能计算框架 (如消息传递接口 (message passing interface，MPI)[7]) 相比，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时，使用 MPI 比 Spark 快 4.6～10.2 倍[8]。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 框架比使用 Spark 的 GraphX 库快两个数量级[9]。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 提供了丰富的进程间通信和同步原语，能支持高效的并行算法设计，广泛应用于各类计算密集型任务中。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要求，传统集群均显现出各自的不足之处。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大数据的高效处理机制，又达到高性能计算技术的性能，是一个颇具前景的手段[10]。

然而，由于大数据框架和高性能计算框架的应用环境和设计目标不同，融合趋势还处于发展阶段，相关技术还不够完善，当前还存在许多亟待解决的挑战[11]。因此，本文提出了基于 Spark 和 MPI 的混合大数据平台，以充分利用 Spark 大数据计算框架提供的简易编程接口和容错等机制，以及 MPI 高性能计算框架提供的通信原语带来的性能提升，进而为人工智能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了大数据处理与高性能计算，形成了强大而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 专注于高性能计算和优化进程间通信，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，集成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 的集成方法进行分类讨论，主

要包括 Spark 调用 MPI、扩展 MPI 以实现 Spark 机制，以及通过第三方集群资源管理模块进行融合和其他融合方式。

(1) 通过在 Spark 程序执行过程中调用 MPI 库实现高性能计算是一种常见的融合方法。该方法通过利用 Spark 强大的数据处理能力和 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 通信等方式传递数据。例如， Spark＋MPI[12]通过共享内存进行数据传输，从而实现计算任务卸载。而 Alchemist[13-14]则利用 Socket 通信传输数据，避免了内存副本的开销，尽管其性能可能会受网络带宽限制。为克服这一问题，可引入更多高性能特性，如采用远程直接数据存取 (remote direct memory access，RDMA) 等高性能计算集群中的网络通信技术[15-16]。

(2) 通过扩展 MPI 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[17]，从而显著提升计算性能。但该方法通常需要大量额外代码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[18]是该方向的代表性工作，它基于 MPI 实现大数据框架的通信机制，并基于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[19]利用本地语言重写 Spark 的计算引擎，以提高效率，同时复用 Spark 已有的成熟特性； MPI4Spark[20]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[21]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，但灵活性较低。

(3) 通过第三方集群资源管理模块进行融合避免了对 Spark 和 MPI 框架的深度修改，减少了兼容性问题。该方法通过集群资源管理器 (如 Slurm 或 Hydra) 管理 Spark 和 MPI 间的通信和资源分配。例如，Spark-MPI 架构[22-23]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。该方法虽然简化了实现过程，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。

(4) 除了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，大数据负载通常利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。然而，数据转换编程框架和并行编程模型在融合过程中需协调不同应用组件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同时配置并维护大数据和高性能计算环境，操作烦琐、成本较高。为缓解上述复杂性，Mammadli 等[24]提出了一种基于任务的编程模型，用于统一协调双密型应用中的数据与计算任务。该模型虽然简化了系统集成，但在底层性能优化方面存在一定程度的折中。此外，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[25]，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，何晓斌等[26]提出了并行存储系统的解决方案，以提升融合系统在存储层面的性能表现和适配能力。

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。为此，本文提出了一套基于 Spark 与 MPI 集成的全新数据分析和处理平台。在实现过程中，本文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和多项前沿技术的融合实现。围绕计算、存储和通信三大核心模块，本文对平台进行了系统性优化，以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和计算效率。

3 融合框架设计

本节主要介绍系统的整体架构设计和模块划分，包括融合系统的各个组成模块和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，构建高可用和高并发的存储与计算一体化的高性能大数据计算平台。为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，因此可显著降低大数据计算中的延迟问题。

该平台的架构如图 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。

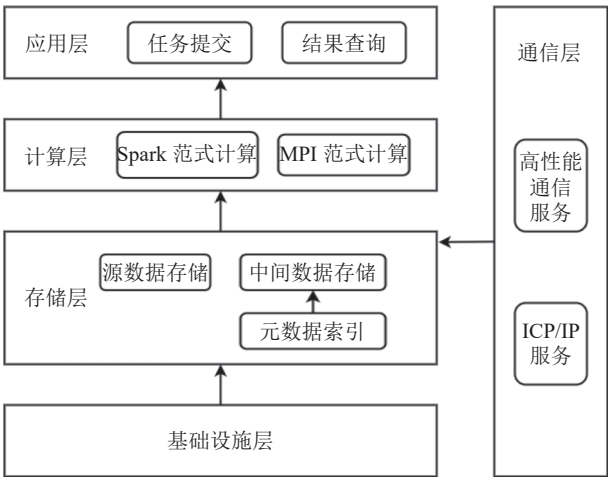

图 1 基于 Spark 和 MPI 的混合大数据平台架构

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

本文的设计工作主要围绕融合系统的计算层、存储层和通信层，分别实现双范式计算功能、异构存储功能和高性能混合通信功能。为此，本文将针对计算、存储和通信 3 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；存储层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和元数据存储模块；通信层细分为高带宽通信模块和低延时通信模块。融合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，重点针对双密型应用的数据密集性和计算复杂性，完成一套基于 Spark 和 MPI 双范式的一

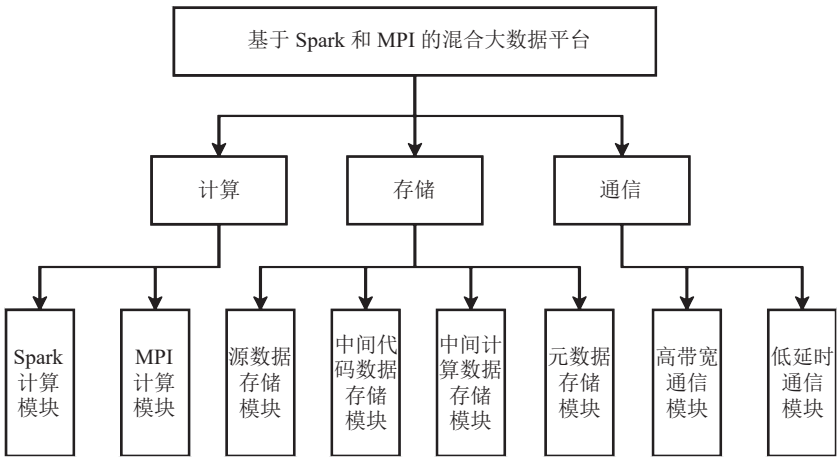

图 2　融合系统功能模块

Fig. 2　Integrated system functional modules

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。

本文设计的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克服以下 3 个关键难点。

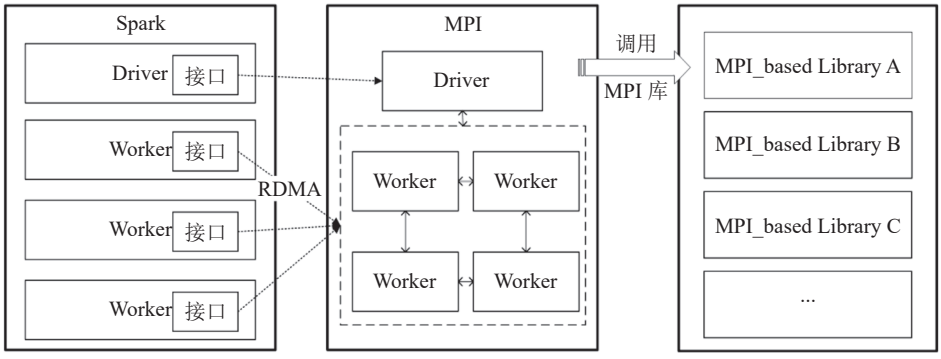

图 3　计算模块整体架构

Fig. 3　Overall architecture of the computing module

(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。通信过程涉及分布式数据和非分布式数据两类，系统需分别实现针对这两类数据的高性能传输机制。

(2) 数据序列化：由于两个范式计算时使用的数据结构具有独特性，因此在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：用户正常提交作业后，需在脱机情况下实现自动调用。

集群间数据通信主要包括通过共享文件、通过共享内存和通过 Socket 3 种方式，且各有利弊。通过共享文件进行数据通信可以很容易地实现数据持久化和容错，且文件系统具有完备的命名空间树，简化了对共享数据的管理，但基于磁

盘存储的共享文件会造成计算过程中额外的磁盘I/O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I/O性能，进而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进行数据通信可以避免额外的磁盘I/O操作，并支持点对点的分块数据通信，但其性能仍会受到网络带宽的限制。因此，本文的计算模块采用了内存分布式文件存储与高性能通信技术相结合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能解析的格式。数据转化时需考虑如下问题：

(1) 计算密集型应用通常依赖分层数据格式组织文件和数据集(如主流的HDF5[27]文件和NetCDF[28]数据)，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特征，很难将大数据框架的数据格式转化为所有数据密集型应用都能处理的格式[29]。

(3) 需要存储的数据与中间数据存储模块的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实现，即先将Spark数据结构转化为中间数据存储模块可以识别的格式，再将存储模块中的数据转化为适配MPI计算的数据格式。一方面，该方法可通过拆解Spark和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，该方法可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，计算模块需有效地进行任务调度。为此，本平台采用预先定义的工作流，基于任务分割和任务类型判断，实现任务在运行过程中的自动化分型调度。具体而言，任务分割依赖人工预先定义的工作流，而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，并基于任务的计算逻辑、数据传输量和计算复杂度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，并基于历史运行数据和统计模型，可正确识别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的存储和传输，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和调度方式，平台能实现更高效的计算资源分配和性能优化。

最终，计算模块的整体执行流程如图4所示，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，则调用Spark模块采用Spark范式执行；若为计算密集型任务，则调用MPI模块，采用MPI范式进行计算；

(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和将计算数据写入中间数据存储模块，并预先为计算结果分配存储空间；

(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，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，进入新一轮任务类型判断与调度。

3.3　存储模块

存储模块是实现存算一体化的核心基础，可为双密型应用提供高性能的大数据存储服务。为

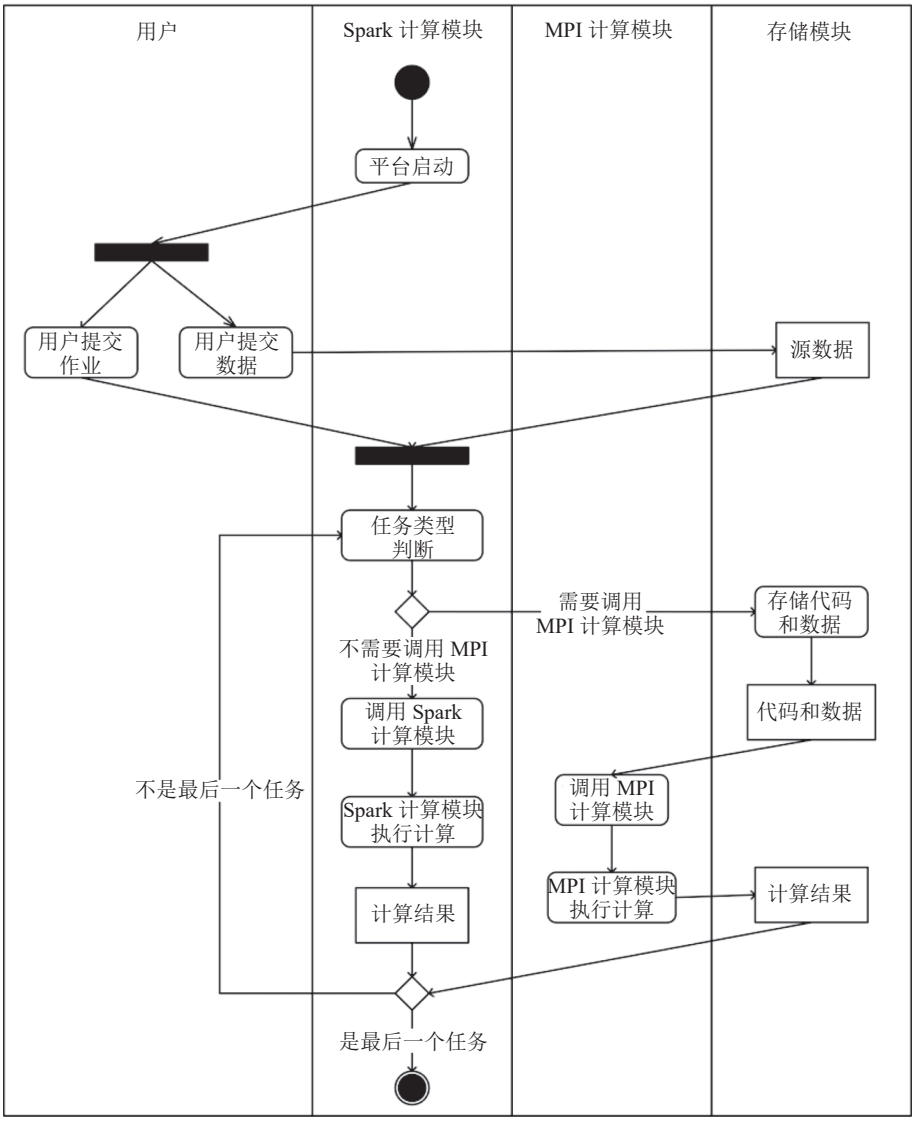

图 4　基于 Spark 和 MPI 的混合大数据平台活动图

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

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的特性，提供针对性的存储方式，从而在吞吐量与延时之间取得平衡，以最大化存储性能。

与高性能计算中常用的并行文件系统相比，传统的 Hadoop 分布式文件系统（Hadoop distributed file system，HDFS）在速度上存在显著差距。为解决这一问题，本文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。最终，本文存储系统以 HDFS 为核心，结合分布式内存文件系统，从数据安全性和高性能读写两个维度满足双密型应用的需求。其中，HDFS 用于源数据存储模块，分布式内存文件系统则对应中间数据存储模块。

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更好地适配各自的使用场景，还能有效提升内存空间的利用效率。

为进一步优化存储系统的整体读写性能，本文存储系统还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) 和元数据存储模块。

3.3.1 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，源数据存储模块是基于传统 HDFS 实现的，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。

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，增强了系统的可扩展性。通过这种设计，混合大数据平台能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。

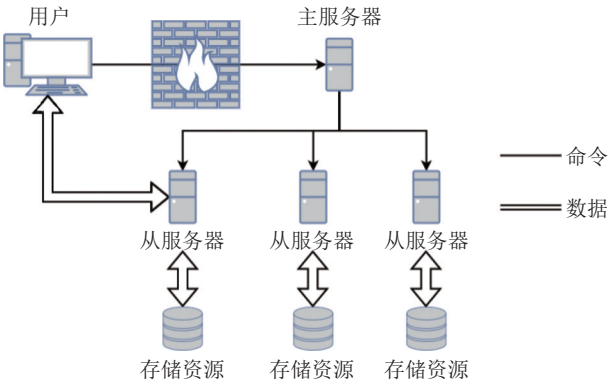

图 5 源数据存储模块的集群架构

Fig. 5 Cluster architecture of source data storage module

3.3.2 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 计算范式融合的关键。其主要功能在于存储代码和在调用 MPI 计算模块之前、经过预处理生成的序列化数据，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，且需在 MPI 集群中所有参与该任务的节点间共享，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 3 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。服务器接到请求后，将该文件发送给 MPI 计算模块，如图 6 所示。任务完成后，Spark 计算模块可通过删除共享文件接口删除服务器中存储的共享文件。

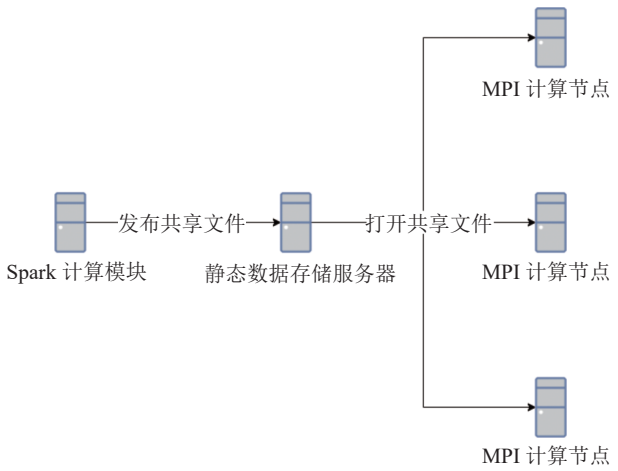

图 6　中间代码数据存储模块集群架构

Fig. 6　Cluster architecture of intermediate code data storage module

Spark 计算范式运行在 Java 虚拟机上，主要用 RDD 作为数据结构；而 MPI 计算范式在更低水平运行，以 C＋＋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是通过第三方分布式内存文件系统实现的，主要优点如下：

(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) 与磁盘相比，内存提供更高的 I/O 性能，更适应计算模块的高速处理需求；

(3) 分布式架构可分散数据访问压力，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) 分布式内存文件系统将数据以文件形式组织，利用树形目录结构管理分布式数据，从而高效简化数据管理，减少额外的数据访问开销。

由于本文的测试规模较小，节点数有限，且对高可用性要求不高，因此，在原型系统中，本文采用较轻量的方案：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和稳定性。

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，结合通信模块可获得高效的存储性能。

3.3.3　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，具体如下：

(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。

针对问题 (1)，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，为中间计算数据存储模块提供元数据服务。元数据存储模块采用 Redis 数据库，可高效处理中间存储的高频访问和短期特性。对于问题 (2)，本文提出数据与元

数据分离存储的策略：在 MPI 节点读取计算数据之前，由元数据存储模块为 MPI 节点提供元数据信息，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。

上述元数据存储设计方案具有多种优势，具体如下：

(1) 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 I/O 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；

(2) Redis 提供了持久化机制，可保证 Redis 遇到异常情况时的数据安全性和可靠性；

(3) Redis 的 I/O 多路复用和事件驱动机制能高效支持并发访问，有效应对中间存储的突发式访问；

(4) 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间和计算模块内各计算节点间提供通信服务，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，可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 计算范式的融合依赖两者间的数据传输，是实现协同计算的核心。模块间需要传输的数据可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。分布式数据通常存在文件 I/O、共享内存和 Socket 通信 3 种传输方式，但这 3 种方式均不能很好地满足本系统的需求。非分布式数据通过建立高吞吐通信的方式进行传输。若采用文件 I/O 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，则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，则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，则会进行数据的多次封装与解封装，造成计算资源浪费，当数据量达到 GB，甚至 TB 级别后，通信时延会大大增加，导致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 计算模块间的传输和计算模块与存储模块间的通信均采用高吞吐模式。相比之下，Spark 计算模块与 MPI 计算模块间的分布式数据传输和 MPI 计算模块的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

4.1 测试环境与评价指标

本实验在由 3 台服务器组成的集群上进行开发和测试。该集群包括计算集群 (Spark 计算集群和 MPI 计算集群) 和存储集群 (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 个 Intel Xeon E5-2630 v3 处理器，提供 16 核 32 线程的计算能力，并搭载 64 GB (4×16 GB) DDR4 R-ECC 内存。存储部分由 5 块 5 TB SATA 硬盘组成，总容量达 25 TB。网络环境采用 1 Gbps 以太网和 4 路 25 Gbps RDMA (RoCE) 链路。Spark 计算集群、MPI 计算集群和数据库集群的软件环境如表 1 所示。

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 和 Scala 2.12.12 集群环境：Spark2.4.3 和 HDFS2.7.3	编译环境：gcc、g++和 gfortran 9.3.1 集群环境：mpich-3.4.3	数据库：Redis 3.0.4

本小节对比了基于 Spark 和 MPI 的混合大数据平台与传统的大数据计算框架 Spark，评估了其在不同任务中的计算性能提升情况。性能提升利用“响应时间提升百分比”衡量，表示如下：

$$\mu=\frac{T-t}{T}\times 100\% \tag{1}$$

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

4.2　实验结果

本文选取线性代数和图论中比较基础和经典的算法测试本平台相较于 Spark 的性能表现。本文分别在相同规模的 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，对应的测试方案如表 2 所示。

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 $10^4\times10^4$，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，提取其主要特征信息	随机生成的密集型矩阵，输入矩阵的维度为 $10^4\times10^4$，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，并利用 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]

为避免随机因素的干扰，每个用例测试 3 次，并取结果的平均值。各方案的测试结果如表 3 所示。根据不同计算密集型任务占比下的性能平均提升情况，对于选取的 3 种测试算法，随着计算密集型任务占比的增加，基于 Spark 和 MPI 的混合大数据平台在性能提升方面表现出明显增强，因此该平台更适合双密型应用的计算。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，各方案的性能提升均大于 10%。

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%

5　结　论

面对双密型应用对存储系统高容量、高可靠性和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均存在一定的局限性。因此，本文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提供的简易编程接口、容错机制和高性能计算框架提供的通信原语带来的性能提升，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。在此背景下，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平台进行了系统性测试。测试结果表明，该平台能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] Korkmaz E. Adversarial robust deep reinforcement learning requires redefining robustness [C] // Proceedings of the AAAI Conference on Artificial Intelligence, 2023: 8369-8377.

[2] Chen A, Gisolfi N, Dubrawski A. Data-driven discovery of design specifications (student abstract) [C] // Proceedings of the AAAI Conference on Artificial Intelligence, 2024: 23449-23450.

[3] Goyal S, Maini P, Lipton ZC, et al. Scaling laws for data filtering—data curation cannot be compute agnostic [C] // Proceedings of the 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024: 22702-22711.

[4] Mostafaeipour A, Jahangard Rafsanjani A, Ahmadi M, et al. Investigating the performance of Hadoop and Spark platforms on machine learning algorithms [J]. The Journal of Supercomputing, 2021, 77(2): 1273-1300.

[5] Zaharia M, Xin RS, Wendell P, et al. Apache Spark: a unified engine for big data processing [J]. Communications of the ACM, 2016, 59(11): 56-65.

[6] 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 HL, Pi JY. Big data processing platform comparison and analysis [J]. Microcomputer & Its Applications, 2015, 34(11): 7-9.

[7] Walker DW, Dongarra JJ. MPI: a standard message passing interface [J]. Supercomputer, 1996, 12: 56-68.

[8] 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] // Proceedings of the 2016 IEEE International Conference on Big Data, 2016: 204-213.

[9] Slota GM, Rajamanickam S, Madduri K. A case study of complex graph analysis in distributed memory: implementation and optimization [C] // Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium, 2016: 293-302.

[10] Mei SZ, Guan HT, Wang QL. An overview on the convergence of high performance computing and big data processing [C] // Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems, 2018: 1046-1051.

[11] Usman S, Mehmood R, Katib I. Big data and HPC convergence for smart infrastructures: a review and proposed architecture [M]. Mehmood R, See S, Katib I, et al. Smart infrastructure and applications: foundations for smarter cities and societies. Cham: Springer International Publishing, 2020: 561-586. DOI: 10.1007/978-3-030-13705-2_23.

[12] 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.

[13] Gittens A, Rothauge K, Wang SS, et al. Alchemist: an apache Spark ⇔ MPI interface [J]. Concurrency and Computation: Practice and Experience, 2019, 31(16): e5026.

[14] Gittens A, Rothauge K, Wang SS, 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.

[15] Zhang ZY, Liu ZT, Jiang QC, et al. RDMA-based apache storm for high-performance stream data processing [J]. International Journal of Parallel Programming, 2021, 49(5): 671-684.

[16] 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 XJ, Sun Q, Cai LZ. Application of RDMA technique in data center [J]. Computer Applications and Software, 2021, 38(3): 22-25.

[17] Li ZL. Geospatial big data handling with high performance computing: current approaches and future directions [M]. Tang WW, Wang SW. High performance computing for geospatial applications. Cham: Springer International Publishing, 2020: 53-76. DOI: 10.1007/978-3-030-47998-5_4.

[18] Lu X, Liang F, Wang B, et al. DataMPI: extending MPI to Hadoop-like big data computing [C] // Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, 2014: 829-838.

[19] Yu BW, Feng GY, Cao HQ, 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, 2022, 15(4): 872-885.

[20] Al-attar K, Shafi A, Abduljabbar M, et al. Spark meets MPI: towards high-performance communication framework for Spark using MPI [C] // Proceedings of the 2022 IEEE International Conference on Cluster Computing, 2022: 71-81.

[21] Foldi T, Von Csefalvay C, Perez NA. JAMPI: efficient matrix multiplication in Spark using barrier execution mode [J]. Big Data and Cognitive Computing, 2020, 4(4): 32.

[22] Malitsky N, Chaudhary A, Jourdain S, et al. Building near-real-time processing pipelines with the Spark-MPI platform [C] // Proceedings of the 2017 New York Scientific Data Summit, 2017: 1-8.

[23] Malitsky N, Castain R, Cowan M. Spark-MPI: approaching the fifth paradigm of cognitive applications [Z/OL]. arXiv Preprint, arXiv: 1806.01110, 2018.

[24] Mammadli N, Ejarque J, Alvarez J, et al. DDS: integrating data analytics transformations in task-based workflows [J]. Open Research Europe, 2022, 2: 66.

[25] Chen Q, Chen K, Chen ZN, 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.

[26] 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 XB, Jiang JH. Sunway parallel storage system for big data heterogeneous system [J]. Big Data, 2020, 6(4): 30-39.

[27] 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.

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

[29] Liu JL, Racah E, Koziol Q, et al. H5Spark: bridging the I/O gap between Spark and scientific data formats on HPC systems [C] // Proceedings of the Cray Users Group, 2016: 1-8.