Computer Science and Application 计算机科学与应用, 2020, 10(7), 1422-1430
Published Online July 2020 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2020.107147

Hans汉斯

Cloud Data Center Network Multi-User Multi-Service Concurrency Algorithm

Haike Ren[1*], Fugui Yang[2], Hui Li[3], Xiaolong Chen[3]

[1]College of Electronic and Information Science, Fujian Jiangxia University, Fuzhou Fujian
[2]Mathmatical Institution, Fujian Jiangxia University, Fuzhou Fujian
[3]Ruijie Networks Co., Ltd., Fuzhou Fujian
Email: *rhk00@sina.com

Received: Jul. 10th, 2020; accepted: Jul. 23rd, 2020; published: Jul. 30th, 2020

Abstract

In the cloud data center network, massive service flows often have delays and packet loss in data transmission, but single-user performance testing cannot achieve multi-user multi-service flow concurrency detection. The transmission process of multi-user multi-service concurrent data on the network has discovered the factors that affect the bandwidth, delay, and packet loss during data transmission, in the case of multi-user concurrency issues. In this paper, the Parallel Matrix Multiplication algorithm is used to implement the RDMA Ib_write source port change, which solves the current test bottleneck of multi-user multi-service flow testing; the distributed and multi-threaded group function based on RDMA technology is used to achieve concurrent testing, and achieve the simulation effect of multi-user concurrency without adding a server, while also reducing resource consumption and time consumption.

Keywords

Cloud Data Center, RDMA, Multi-User, Multi-Service, Concurrency

云数据中心网络多用户多业务并发性算法

任海科[1*]，羊富贵[2]，李　辉[3]，陈小龙[3]

[1]福建江夏学院电子信息科学学院，福建　福州
[2]福建江夏学院数理部，福建　福州
[3]锐捷网络股份有限公司，福建　福州
Email: *rhk00@sina.com

收稿日期：2020年7月10日；录用日期：2020年7月23日；发布日期：2020年7月30日

*通讯作者。

文章引用: 任海科, 羊富贵, 李辉, 陈小龙. 云数据中心网络多用户多业务并发性算法[J]. 计算机科学与应用, 2020, 10(7): 1422-1430. DOI: 10.12677/csa.2020.107147

摘　要

在云数据中心网络中，海量的业务流在数据传输中往往会出现延迟、丢包等现象，但单用户性能测试无法实现多用户多业务流并发性的检测，为了能实时了解数据中心网络多用户多业务并发性数据的传输过程，发现数据传输过程中影响带宽、产生延迟、丢包现象的形成因素，解决了当前无法仿真多用户多条业务流测试瓶颈，解决不需增设服务器的情况下多用户并发性问题。文中采用Parallel Matrix Multiplication算法实现RDMA Ib_write源端口变化，解决了当前无法仿真多用户多条业务流测试瓶颈；采用基于RDMA技术的分布式和多线程组功能实现了并发测试，达到不增设服务器却能实现多用户并发的仿真效果，同时也降低了资源消耗及时间消耗。

关键词

云数据中心，RDMA，多用户，多业务，并发性

1. 引言

随着 Web2.0 时代的到来，基于互联网的网上交易应用迅猛发展，尤其出现类似每年的“双十一”等可以说是购物的狂欢节，随之而来的就是网络数据量的爆炸式增长，论文[1]概述了传统数据中心网络体系结构及其不足，指出了新的需求，指出了数据中心网络未来的发展方向；论文[2]建立了一种面向云计算的数据中心网络拓扑结构，仿真其拓扑结构的数据中心网络上进行云计算是可行性；论文[3]介绍了近年来利用 RDMA 加速分布式系统的工作处理器的利用率以及对 RDMA 的使用率；论文[4]描述了通过商用以太网来运行 RDMA 的一些挑战，这一挑战包括使用 RDMA 支持微软的高可靠、延迟敏感的网络服务。为了保障数据服务的稳定运行，在实现云数据中心运营平台产品上线前的能模拟百亿级别访问量的情景再现，设计基于 RDMA 技术的云数据中心网络多用户多业务算法，模拟多用户多条业务流，实现仿真多用户多条业务流测目的，本文通过基于 RDMA 技术的云数据中心网络进行数据建模，设计 Parallel Matrix Multiplication 算法实现 RDMA Ib_write 源端口变化，解决了当前无法仿真多用户多条业务流测试瓶颈，为应对百亿级数量访问量的数据中心的稳定运行提供运营前的测试保障。

2. 研究背景及相关工作

2.1. 数据中心网络拓扑结构

基于云的数据中心网络，2017 云栖社区运维/DevOps 在线技术峰会上，阿里云专家云登针对基于云的数据中心计算网络集群总结有两类架构[5]。传统情况下云计算网络架构会分为三层：接入层、汇聚层和核心层。如图 1 [1]。

另外一种比较常见的云计算网络集群架构，在 Spine 节点和 Leaf 节点之间可能会存在三层连接，而 Spine 节点和 Core 节点之间也可能会存在三层连接，这种网络架构相比于前面提到的架构而言，其扩展粒度要更细，可以细化到一组或者多组进行接入。如图 2 [1]。

2.2. 数据中心网内数据分派机制

无论选用那种数据中心网络集群架构，海量的业务流数据在传输中往往会出现延迟、丢包等现象，但单用户性能测试无法实现多用户多业务流并发性的检测，为了能实时了解数据中心网络多用户多业务并发性数据的传输过程，发现数据传输过程中影响带宽、产生延迟、丢包现象的形成因素，解决了当前无法仿真多用户多条业务流测试瓶颈，解决不需增设服务器的情况下多用户并发性问题。

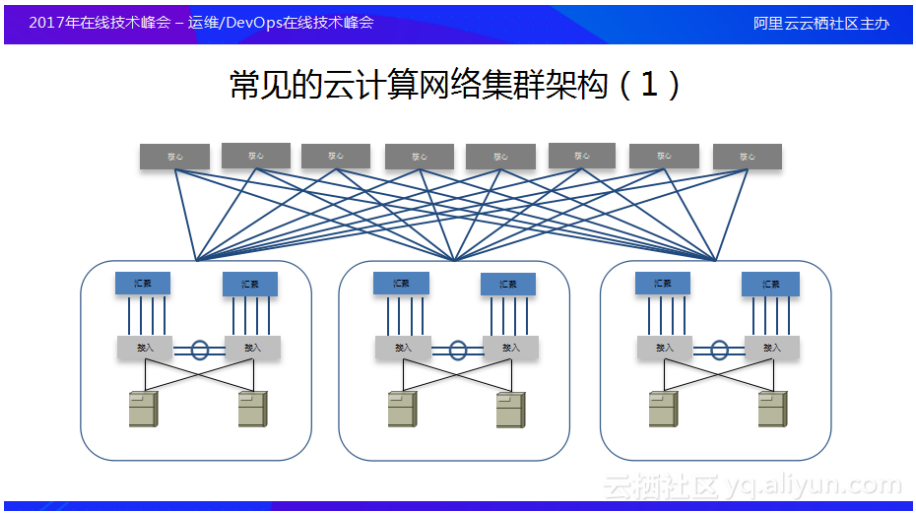

Figure 1. Traditional cloud computing three-layer network architecture
图 1. 传统云计算三层网络架构

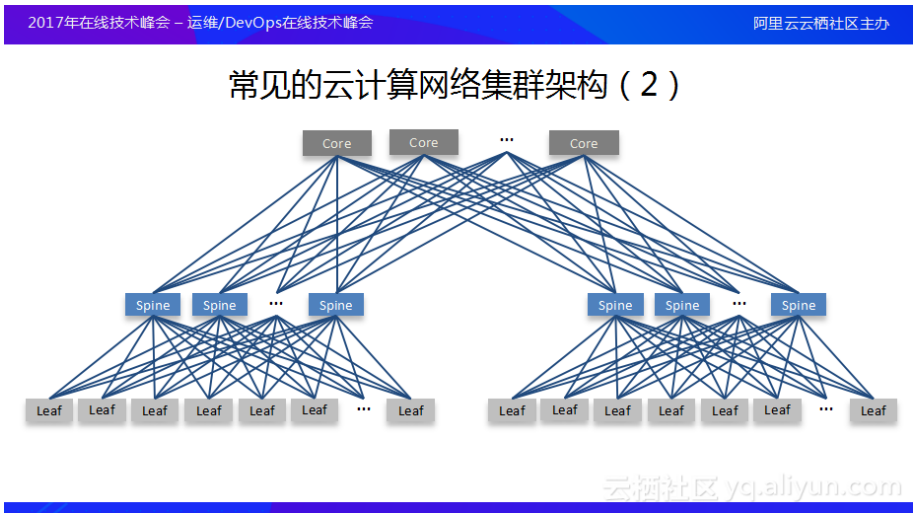

Figure 2. Traditional cloud computing network cluster architecture
图 2. 传统云计算网络集群架构

3. 云数据中心多业务并发性算法

3.1. 问题描述

当今时代存在数据大爆炸的特征。信息的高速增长要求领域技术和软件工具处理大量的数据。为了应对云数据中心的多业务大数据挑战，一些分布式内存的并行执行模型已经被提出：MapReduce [6]，迭

代 MapReduce，图处理和数据流图处理并行执行模型。MapReduce 编程模型已经应用到很大范围的计算应用，因为它能提供简单的使用和处理大规模数据时的效率。但是 MapReduce 有其局限性。比如 MapReduce 处理多个相关的异构数据集合时的效率不高。MPI [7] [8]是一个开放的通讯协议，用于编写并行程序。MPI 的目标是高性能，大规模，和可移植性。

3.2. MPI 计算架构

mpirun 是 MPI 程序的启动脚本程序，它的提供简化了并行程序的启动，隐藏了底层的实现细节，为用户提供了一个通用的 MPI 处理机。mpirun 在执行并行程序时，参数-np 指明了需要并行运行的进程个数。mpirun 首先在本地机器上启动一个进程，然后根据 machines 文件中所列出的主机，为每个主机启动一个进程，如图 3。一般会给每个节点分一个固定的标号，类似于身份 ID，在消息通信中会用到。

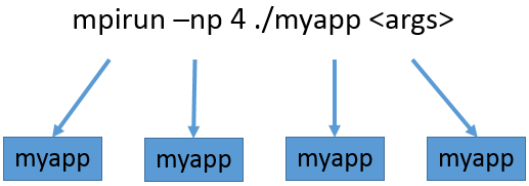

Figure 3. MPI computing architecture
图 3. MPI 计算架构

3.3. 多业务并发性算法

由于并行矩阵乘法有着良好的通信和计算开销模式，因而选择并行矩阵乘法程序来评估多用户多任务算法的性能。

矩阵乘矩阵的性能模型已经被研究了好几十年。并行矩阵乘法的计算复杂度以 N 的立方增长，而它的内存开销以 N 的平方增长。并行矩阵乘法的工作量已经被分为许多同构的子任务并且被并行执行。并行矩阵乘法良好的计算和通信模式使它成为研究多任务多用户算法程序的良好应用。我们提出了一个精确的分析模型来分析 MPI 并行矩阵乘法。多用户多任务的并行矩阵乘法的算法开销为：

$$T(N)=\frac{\sqrt{N}(N+1)}{2}*\text{Tscheduling}+\frac{\left(1+\log_2^{\sqrt{N}}\right)*M^2}{2*\sqrt{N}}*\text{Tcomm}+2*\left(M^3/N\right)*\text{Tflops}$$

MPI 并行矩阵乘法算法代码如下：

```
int main(intargc, char* argv[]) {
int     p;
intmy_rank;
        PROCESS_INFO_T    grid;
double*    local_A;
double*    local_B;
double*    local_C;
double*    temp_mat;
int n;
int m;
int option;
```

```
intn_threads;
intgot_local_memory = 1;
intgot_all_memory = 0;
double t1, t2, Mflops;
voidSetup_grid(PROCESS_INFO_T* grid);
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
Setup_grid(&grid);
if (my_rank == 0) {
    (void) sscanf(argv[1], "%d", &n);
      (void) sscanf(argv[2], "%d", &n_threads);
    (void) sscanf(argv[3], "%d", &option);
       }//if
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&option, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&n_threads,1,MPI_INT,0, MPI_COMM_WORLD);
if (my_rank == 1)
printf("process[1] n:%d, option:%d, threads:%d\n",n,option,n_threads);
      m = n/grid.s;
if (m*grid.s != n) {
if (grid.my_world_rank == 0) {
printf("\nn must be multiple of s\n");
printf("Instead you gave n = %d and s = %d. Try \n", n, grid.s);
printf("it again, but *first* read the documentation, dipshit.\n");
           };
MPI_Finalize();
exit(-1);
        }
local_A    =Initialize_Matrix(grid, m);
local_B    =Initialize_Matrix(grid, m);
local_C    = (double *) malloc(m*m*sizeof(double));
temp_mat = (double *) malloc(m*m*sizeof(double));
BMR(n, grid, local_A, local_B, local_C, temp_mat,n_threads,option);
MPI_Finalize();
return 0;
}
```

4. 讨论

我们实现了支持多用户用任务的 MPI 并行矩阵乘法算法程序。使用分析模型精确的分析了这一算法程序的通信开销和计算开销。该分析模型使我们能够实时了解数据中心网络多用户多业务并发性数据的

传输过程，发现数据传输过程中影响带宽、产生延迟、丢包现象的形成因素。

5. 仿真实验

5.1. 实验环境设计

1) 搭建包含以下之一 Mellanox 网卡适配器的服务器：

- ConnectX®-5 Ex (VPI, IB, EN) (firmware: fw-ConnectX5)
- ConnectX®-5 (VPI, IB, EN) (firmware: fw-ConnectX5)
- ConnectX®-4 Lx (EN) (firmware: fw-ConnectX4Lx)
- Innova IPsec 4 Lx EN
- ConnectX®-4 (VPI, IB, EN) (firmware: fw-ConnectX4)
- ConnectX®-3 Pro (VPI, IB, EN) (firmware: fw-ConnectX3Pro)
- ConnectX®-3 (VPI, IB, EN) (firmware: fw-ConnectX3)
- Connect-IB® (IB) (firmware: fw-Connect-IB)

2) 1GB 以上大小硬盘

3) 最新的 Mellanox device IDs

4) Linux 操作系统

5) root 安装权限

5.2. 实验过程

5.2.1. 软件工具

软件方面，OEM 厂商 Dell 将 Mellanox 网卡固件程序升级至最新版本：Network_Firmware_TPTR3_LN_14.20.18.20_01.BIN；以确保支持 RDMA 源端口变化等高级特性。此外 MPI 驱动程序采用的是 Mellanox 原厂驱动：MLNX_MPI_LINUX-4.2-1.0.0.0-rhel7.4-x86_64。

1) 确认系统装有网络适配器(HCA/NIC)，采用下面的命令可以显示 Mellanox HCA 的安装信息：

lspci -v | grepMellanox

86:00.0 Network controller [0207]: Mellanox Technologies MT27620 Family

Subsystem: Mellanox Technologies Device 0014

86:00.1 Network controller [0207]: Mellanox Technologies MT27620 Family

Subsystem: Mellanox Technologies Device 0014

2) 下载 tgz 压缩文件

文件的格式 MLNX_MPI_LINUX-<ver>-<OS label><CPU arch>.tgz.注：下载前查看 Linux 系统版本：

下载路径 http://www.mellanox.com --> Products --> Software-->InfiniBand/VPI Drivers -->MellanoxMPI Linux (MLNX_MPI)。

5.2.2. 硬件工具

测试使用的 RDMA 和 RoCE 协议中，以太网交换机需要支持 PFC，ECN，ETS 功能；网卡需要支持 RoCE 协议。网卡主要使用了 Mellanox ConnectX-4 LX [9]系列双 25 Gpbs 网口网卡。在测试中，主要测试了锐捷 S6500-4C 交换机，在部分测试中也有使用到 S6510 交换机。

5.2.3. 测试过程

1) 以 root 权限登陆到系统

2) 运行安装脚本#./mlnxMPIinstall；运行前如果安装过程中需要升级网络适配器固件ConnectX-3/ConnectX-3 Pro-重启驱动、ConnectX-4/ConnectX-4 Lx-重启系统、ConnectX-5/ConnectX-5 Ex-重启系统；否则重启驱动# /etc/init.d/openibd restart。

3) 运行 hca_self_test.MPI 实用程序来验证 InfiniBand 链接是否 up。该实用程序还检查和显示额外的信息，例如：HCA 固件版本、内核体系结构、驱动程序版本、与它们的状态一起活动的 HCA 端口的数量、节点 GUID 等信息。如需了解有关 hca_self_test. MPI 的更多细节，查看文件：docs/readme_and_user_manual/hca_self_test. readme。

4) 安装完成后，运行命令/ etc/infiniband/ info 来获取关于安装的 Mellanox 的信息，例如前缀、内核版本和安装参数，在启动后自动加载的模块列表可以在/ etc/infiniband/ openib. conf 文件中找到。

5) 端口设置

创建一个包含信息(源端口、目的端口、源 IP、目的 IP、协议)的五元组，利用 Parallel Matrix Multiplication [10]算法实现 RDMA Ib_write 源端口变化，实现多用户、多业务端口变化的目的，达成亿级的用户量和数据量。设置过程如下：

Step 1：五元组信息(源端口、目的端口、源 IP、目的 IP、协议)初始化；

1) 源 mac 无变化(24:8a:07:9a:28:83)目的 mac 无变化(58:69:6c:14:ca:01)

2) 源 ip 无变化(10.10.10.177)目的 ip 无变化(30.0.0.182)

3) 源端口变化随机，目的端口随机

4) 协议：RoCE

5) 报文长度：1082

Step 2：利用 Parallel Matrix Multiplication 算法实现 RDMA MPI 源端口变化，根据用户五元组中的源端口的变化，确定业务服务数量；当业务服务数量小于预设阈值 3000 时，不进行任何处理，当业务服务数量大于预设阈值 3000 时，转到 Step 3 执行。从而实现多用户、多业务端口变化的目的，达成亿级的用户量和数据量；

Step 3：在 MPI 业务下，从接收的用户多类业务请求中获取 MPI 矩阵乘法链路带宽等参数信息，根据这些信息给链路分配带宽，实现链路负载均衡。当矩阵的行和列较大时，如行，列都大于 1024，此时矩阵直接相乘会导致内存空间不够，并出现多次读取的问题。把整个矩阵乘法的动作，切割成很多局部小矩阵的乘法，这样就可以把两个小矩阵加载到共享内存中，小矩阵本身的乘法就不需要再加载其它数据到内存了。把大的整体矩阵 A、B 分成小的局部矩阵时，矩阵 $C = A \times B$，A、B、C 分别为 $M \times K$、$K \times N$、$M \times N$ 的矩阵，其中，$M = wp$，$N = hq$，$K = kr$。具体分割是将某些连续行和某些连续列的交叉部分划分为一块。确保每个元素都在唯一一个局部矩阵里。将矩阵 A 和 B 分割成 A 和 B 的子矩阵。从而达到优化程序，节约缓存的目的。具体为：

1) 定义 MPI 矩阵行和列的数据结构，分别是 MPI_Commrow_comm 和 MPI_Commcol_comm；The BLAS matrix*matrix multiply 算法，当 $M > 1024$，$N > 1024$ 情况下，启动 Parallel Matrix Multiplication 算法(并行矩阵)，并开启多线程调用；

2) 从接收的用户业务请求中获取 MPI 矩阵乘法的参数信息，将矩阵分成局部小矩阵，并给局部要相乘的矩阵分配内存，分配具体方法如下：

a) 首先将矩阵分块，将矩阵 C 分解为 $p \times q$ 的分块矩阵 C_{ij}，每个 $C_{ij} = (A \times B)_{ij}$ 是一个 $w \times h$ 的小矩阵，A 分解成为 $p \times k$ 个分块矩阵 A_{ij}，每个 A_{ij} 是一个 $w \times r$ 的局部矩阵，B 分解成为 $k \times q$ 的分块矩阵 B_{ij}，每个 B_{ij} 是一个 $r \times h$ 的局部矩阵。则分块矩阵乘法的定义为：

$$A=\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1k} \\ A_{21} & A_{22} & \cdots & A_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{pk} \end{pmatrix},\ B=\begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1q} \\ B_{21} & B_{22} & \cdots & B_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ B_{k1} & B_{k2} & \cdots & B_{kq} \end{pmatrix}$$

b) mpi_bcast 行块，本地矩阵块乘法：

$$C=A\times B=\begin{pmatrix} (AB)_{11} & (AB)_{12} & \cdots & (AB)_{1q} \\ (AB)_{21} & (AB)_{22} & \cdots & (AB)_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ (AB)_{p1} & (AB)_{p2} & \cdots & (AB)_{pq} \end{pmatrix}$$

c) mpi_send_receive 列块，mpi_bcast，mpi_send_receive，使用 mpirdmaucx 通信库，实现源端口变化发送和接受行和块数据：

$$(AB)_{ij}=\sum_{k=1}^{m}A_{ik}B_{kj}$$

Step 4：根据矩阵 BMR (broadcast-multiply-roll algorithm)算法结果，计算出运行时间，最终输出用户数量，线程数量，计算出网络延迟和带宽。

5.3. 实验结果

根据上文中所述的方案及实验过程，MPI 矩阵乘法并行开销。MPI 矩阵乘法的并行开销定义如下：

$$f=\frac{\sqrt{N}*\left(1+\log_2^{\sqrt{N}}\right)}{4*M}*\frac{\text{Tcomm}}{\text{Tflops}}$$

使用 16 台服务器测试得到的并行矩阵乘法的并行开销与 $\frac{\sqrt{N}*\left(1+\log_2^{\sqrt{N}}\right)}{4*M}$ 的比率如图 4，显示线性函数的斜率与 Tcomm/Tflops 的测量之间的误差在 5%之内。证明该并行矩阵乘法的加速比成线性增长模式。

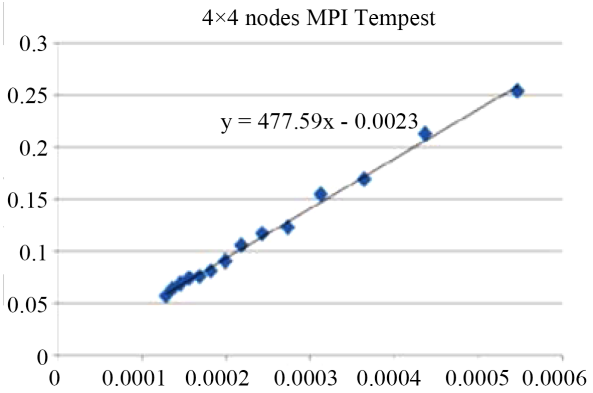

Figure 4. MPI matrix multiplication parallel overhead
图 4. MPI 矩阵乘法并行开销

6. 结论

本论文设计了云数据中心网络多用户多业务并发性算法，并行矩阵乘法的加速比呈线性增长模式，

解决了当前无法仿真多用户多条业务流测试瓶颈，同时通过矩阵乘法解决业务性能仿真无法实现问题。

基金项目

福建江夏学院青年科研人才培育基金重点项目(JXZ2017001)，福建江夏学院校级教育教学改革项目(J2020B016)，福建省自然科学基金面上项目(2018J01587)。

参考文献

[1] 魏祥麟, 陈鸣, 范建华, 等. 数据中心网络的体系结构[J]. 软件学报, 2013(2): 295-316.

[2] 丁泽柳, 郭得科, 申建伟, 罗爱民, 罗雪山, 等. 面向云计算的数据中心网络拓扑结构[J]. 国防科技大学学报, 2011, 33(6): 1-6.

[3] 魏星达, 陈榕, 陈海波, 等. 大数据驱动的智能计算体系架构——基于 RDMA 高速网络的高性能分布式系统[J]. 大数据, 2018(4): 1-14.

[4] Zhu, Y.B., *et al*. (2015) Congestion Control for Large-Scale RDMA Deployments. *ACM SIGCOMM Computer Communication Review*, **45**, 523-536.

[5] [云行] 云计算网络基础架构的实践和演进——打造云计算网络基石[Z/OL]. https://yq.aliyun.com/articles/74431, 2017-04-24.

[6] Dean, J. and Ghemawat, S. (2008) MapReduce: Simplified Data Processing on Large Clusters. *Communications of the Association for Computing Machinery*, **51**, 107-113. https://doi.org/10.1145/1327452.1327492

[7] Snir, M. (2018) The Future of MPI. *Communications of the ACM*, 105. https://doi.org/10.1145/3264415

[8] 赵宝琦, 李卫东, 邹佳恒, 林韬, 颜田. 基于 MPI 的分布式数据处理系统[J]. 计算机工程, 2019(45): 20-25.

[9] Li, H., Chen, X., Song, T., Chen, H. and Chen, H. (2019) Performance of the 25 Gbps/100 Gbps Fullmesh RoCE Network Using Mellanox ConnetX-4 Lx Adapter and Ruijie S6500 Ethernet Switch. *Workshops of the International Conference on Advanced Information Networking and Applications*, **927**, 757-767. https://doi.org/10.1007/978-3-030-15035-8_73

[10] Akbudak, K., Selvitopi, O. and Aykanat, C. (2018) Paritioning Models for Scaling Parallel Sparse Matrix-Matrix Multiplication. *ACM Transactions on Parallel Computing*, **4**, 13. https://doi.org/10.1145/3155292