计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202330164
61(12): 2985−3004, 2024

面向深度神经网络大规模分布式数据并行训练的 MC^2 能耗模型

魏　嘉　张兴军　王龙翔　赵明强　董小社
（西安交通大学计算机科学与技术学院　西安　710127）
（weijia4473@stu.xjtu.edu.cn）

MC^2 Energy Consumption Model for Massively Distributed Data Parallel Training of Deep Neural Network

Wei Jia, Zhang Xingjun, Wang Longxiang, Zhao Mingqiang, and Dong Xiaoshe
（*School of Computer Science and Technology*, *Xi'an Jiaotong University*, *Xi'an* 710127）

Abstract　Deep neural network (DNN) have achieved state-of-the-art accuracy in many modern artificial intelligence (AI) tasks. In recent years, it has become increasingly popular to use high performance computing platforms for massively distributed parallel training of DNN. Energy consumption models have been crucial in designing and optimizing DNN for massively parallel training and restraining excessive energy consumption on HPC (high performance computing) platforms. Currently, most energy consumption models model the energy consumption of a single device or a cluster of multiple devices from a hardware perspective. From an energy consumption perspective, the need for disaggregated analysis of distributed parallel DNN applications has resulted in a dearth of energy consumption models that model the characteristics of distributed DNN applications. In this paper, we propose the "materials preprocessing-computing-communicating" three-stage MC^2 model from the perspective of the essential features of DNN model training for the most commonly used DNN distributed data parallel training model. The model is validated by training the classical VGG16, ResNet50 networks and the latest Vision Transformer network using up to 128 MT nodes and 32 FT nodes on the domestic E-class prototype Tianhe-3. The experimental results show that the difference between MC^2 and the actual energy measurements is only 2.84%. Compared with the four linear proportional energy models and the AR, SES, and ARIMA time prediction models, the accuracy of the model proposed is improved by 69.12%, 69.50%, 34.58%, 13.47%, 5.23%, 22.13%, and 10.53%, respectively. By using the models proposed in this paper, the energy consumption of DNN models at each stage and the overall energy consumption can be obtained on a supercomputer platform, which provides a basis for evaluating the efficiency of DNN energy-aware massively distributed parallel training and inference, as well as optimizing the strategies of task scheduling, job scheduling, model partitioning, and model pruning.

Key words　deep neural network (DNN); energy consumption model; massively distributed training; data parallel; supercomputer

摘　要　深度神经网络（deep neural network, DNN）在许多现代人工智能（artificial intelligence, AI）任务中取得了最高的精度. 近年来, 使用高性能计算平台进行大规模分布式并行训练 DNN 越来越普遍. 能耗模型在设计和优化 DNN 大规模并行训练和抑制高性能计算平台过量能耗方面起着至关重要的作用. 目

收稿日期: 2023-03-17; **修回日期**: 2023-08-15
基金项目: 国家自然科学基金项目(62172327)
This work was supported by the National Natural Science Foundation of China (62172327).
通信作者: 张兴军(xjzhang@xjtu.edu.cn)

前,大部分的能耗模型都是从设备的角度出发对单个设备或多个设备构成的集群进行能耗建模,由于缺乏从能耗角度对分布式并行DNN应用进行分解剖析,导致罕有针对分布式DNN应用特征进行建模的能耗模型.针对目前最常用的DNN分布式数据并行训练模式,从DNN模型训练本质特征角度出发,提出了“数据预处理(materials preprocessing)–前向与反向传播(computing)–梯度同步与更新(communicating)”三阶段MC^2能耗模型,并通过在国产E级原型机天河三号上使用最多128个MT节点和32个FT节点训练经典的VGG16和ResNet50网络以及最新的Vision Transformer网络验证了模型的有效性和可靠性.实验结果表明,MC^2与真实能耗测量结果相差仅为2.84%,相较4种线性比例能耗模型以及AR, SES, ARIMA时间预测模型准确率分别提升了69.12个百分点,69.50个百分点,34.58个百分点,13.47个百分点,5.23个百分点,22.13个百分点,10.53个百分点.通过使用的模型可以在超算平台得到DNN模型的各阶段能耗和总体能耗结果,为评估基于能耗感知的DNN大规模分布式数据并行训练及推理各阶段任务调度、作业放置、模型分割、模型裁剪等优化策略的效能提供了基础.

关键词 深度神经网络;能耗模型;大规模分布式训练;数据并行;超级计算机

中图法分类号 TP302

近年来,作为人工智能(AI)领域典型代表的深度神经网络(deep neural network, DNN)技术得到了突飞猛进的发展,DNN已经广泛应用于图像识别、自然语言处理、语音识别、计算机视觉等领域.DNN效能的提升一方面来源于模型结构的不断复杂和深度的不断增加,另一方面来源于数据集规模的不断增加和数据种类的不断丰富,二者对计算和存储开销都提出了极大的挑战.例如,何凯明团队提供的目前为止最强大的基于DNN的图像分类预训练模型ResNeXt WSL[1],就具有8亿个网络模型参数,使用了9.4亿张图片做训练.为了高效、快速地得到可用的DNN模型,使用高性能计算平台进行大规模分布式数据并行训练成为当今大型DNN模型训练的主流模式.

大规模DNN模型训练任务巨大的计算强度和频繁、大量的数据访问在类似如图1所示的高性能计算平台上产生了大量的能耗.国际数据公司(international data corporation, IDC)指出,随着大数据时代的到来,基于高性能计算(HPC)的大数据分析的HPDA应用成为HPC的下一个热点.目前67%的HPC资源用于HPDA,而DNN是其中的典型应用.高性能计算的未来趋势为AI+科学计算.一方面,利用高性能计算中心开展大规模DNN模型训练会产生大量的经济成本.一个典型的高性能计算中心的能耗相当于25 000个家庭能耗的总和,每单位面积的电量消耗是标准办公空间的100~200倍[2].随着用电量的急剧增加和电费的不断增涨,电费已经成为当今高性能计算平台的一项重大支出[3].另一方面,利用高性能计算中心开展大规模DNN模型训练还会产生许多的环境问题[4].例如,2005年美国仅数据中心的碳排放量就相当于类似阿根廷这种中等规模国家碳排放量的总和.基于此,如何利用节能技术在确保大规模DNN模型训练准确性和实效性的前提下,节省训练过程中的能耗已经成为业界和学界共同关注的关键问题.针对这一问题,首先就需要设计有效和可靠的能耗模型来准确地评估大规模神经网络模型训练任务在高性能计算平台下的能耗情况.

能耗模型是对真实任务或系统能耗的抽象,多种多样的能耗模型在2方面起着关键作用:首先,能耗模型可以引导组件和系统的初始设计.一般来说,为每一个组件设计都制作一个真实的物理系统以评估能耗成本和效率是不可行的[5],而能耗模型可以快速高效地评估能耗,vor dem Berge等人[6]就使用能耗模型设计超算中心的组件;其次,能耗模型可以帮助预测未来一段时间内的能耗趋势.在高性能计算中心中用户运营商往往都需要了解计算机系统和某个特定的应用场景的用电模式,以最大限度提高其效能.对单个物理设备进行测量的方法一方面因为无法预测场景而无法给出未来的能耗情况[7],另一方面也不能提供应用各个阶段或物理资源使用和能耗之间的具体联系.

使用真实的实验数据进行测量通常非常昂贵且灵活性差,使用能耗模型可以更加灵活地适应运行参数的变化且更加便宜.然而,现有大部分研究都是从单个硬件设备或多个硬件设备组成的系统角度出发研究系统利用率和能耗之间的线性或非线性关系,缺乏从应用角度出发研究特定硬件平台环境下的能耗情况.

本文认为,在数据并行和BSP同步模型下,分布式机器学习训练的本质特征就是将DNN模型分配给多个训练节点,每个节点迭代地使用总数据集的

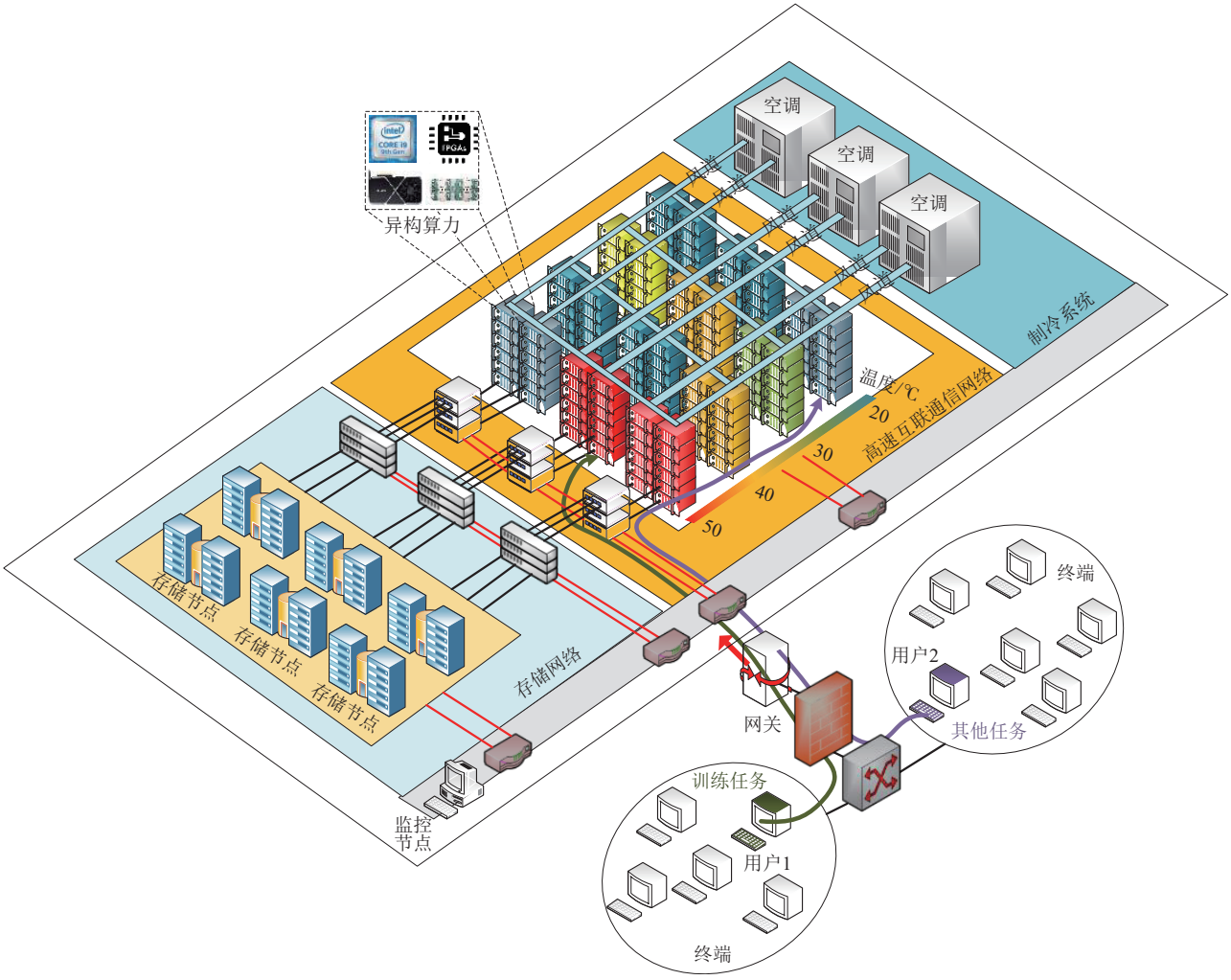


Fig. 1 System structure of HPC platform

图 1 高性能计算平台系统结构

$1/n$（n 为参与训练的节点个数）完成本地计算，并使用集合通信方法进行彼此之间的模型更新. 又因为在整个分布式训练过程中，不同阶段的各部分功耗表现有所差异，基于此，本文构建了 DNN 大规模分布式数据并行训练 MC^2 三阶段能耗模型，将上述分布式 DNN 训练本质映射到实际模型训练过程中的预处理、计算和梯度通信 3 个阶段. 为日后面向大规模智能训练任务的高性能计算中心设计、能耗预测、设计节能优化策略等方面提供了准确、快捷的方案.

本文贡献主要包括 3 个方面：

1）从 DNN 模型大规模分布式数据并行训练的任务特点出发，根据分布式 DNN 训练任务典型的“数据预处理（materials preprocessing）—前向与反向传播（computing）—梯度同步与更新（communicating）”三阶段建立 MC^2 模型. 该模型结合各阶段软硬件特征，提出了各阶段任务的能耗模型和总体任务的能耗模型.

2）针对在异构平台开展 DNN 模型大规模分布式数据并行训练的特性，提出了一种异构平台训练能耗补偿计算策略，能够更准确地评估异构平台下的训练能耗.

3）使用本文提出的 MC^2 能耗模型计算验证了在天河三号原型机上开展分布式 DNN 数据并行训练的能耗情况. 实验分别采用 32 FT 和 128 MT 节点使用 Cifar-10 数据集训练 VGG16 模型以及采用 128 MT 节点使用 ImageNet 数据集训练 ResNet50 模型.

本文后续相关工作中使用的关键变量含义如表 1 所示.

1 相关工作

1.1 DNN 并行化研究

1.1.1 模型并行与数据并行

模型并行和数据并行是当前最主流的 DNN 并

Table 1 Key Variables in Related Work

表 1 相关工作中的关键变量

符号	含义
W	神经网络权值
λ_{lr}	神经网络学习率
n	批大小
E	能耗
P	功率
T	时间
t	时刻
P_{total}	总功率
P_{dynamic}	动态功率
P_{static}	静态功率
A	每个时钟周期内的开关数
V	电压
C	电容
I_{static}	漏电电流
T_{d}	温度
K_i	技术常数
α_i	某个事件发生的次数
C_i	某个事件所花费的能耗
m	处理器核数
ρ	处理器核利用率
P_s	速度为 s 时的处理器功率
λ	任务到达率
R	每个任务平均执行指令数目
α_{ai}	活动因子
$f_{\mathrm{max}i}$	处理器组件最大频率
P_{idlesm}	空闲流处理组件功率
$E_{\mathrm{DRAM\ to\ SRAM}}$	从 DRAM 读写数据到 SRAM 的能耗
E_{SRAM}	SRAM 实现激活和权值缓冲区的能耗
E_{rest}	TPU 其余组件能耗
P_{DRAM}	动态随机存储器（DRAM）功率
μ_{write}	写吞吐量
μ_{read}	读吞吐量
E_{icache}	指令高速缓存能耗
E_{dcache}	数据高速缓存能耗
E_{buses}	高速缓存指令开销
E_{pads}	主存与外设间总线能耗
E_{mem}	内存能耗
$\alpha(A)$	算法 A 激活阶段指令周期数
$R(A)$	算法 A 激活阶段指令读次数
$W(A)$	算法 A 激活阶段指令写次数
P_{cke}	内存静态功率
P_{stby}	激活存储组件和等待指令功率
E_{act}	激活时能耗
P_{rdwr}	内存读写功率
P_{ser}	整个服务器的功率
P_{cpu}	服务器中 CPU 功率
C_{cpu}	每台服务器 CPU 的数量
C_{mem}	每个服务器中内存 DIMMs 的数量
C_{d}	每个服务器磁盘数量
P_{d}	每个磁盘的功率
$(P_{\mathrm{i}}, T_{\mathrm{i}})$	MapReduce 初始化阶段
$(P_{\mathrm{m}}, T_{\mathrm{m}})$	MapReduce 映射阶段
$(P_{\mathrm{shu}}, T_{\mathrm{shu}})$	MapReduce 洗牌阶段
$(P_{\mathrm{r}}, T_{\mathrm{r}})$	MapReduce 规约阶段
E_{layer}	每一层的能耗
E_{comp}	计算开销
E_{data}	数据输入、数据输出以及卷积核的访存开销

行化的 2 种方式[8]. 前者通过将深度模型纵向切分成若干子网络层并分解到各个计算节点上，然后通过多个节点间的协作完成整个模型的训练；后者是通过将训练数据进行切分，每个节点分别使用分片的数据训练完整的模型，并采用参数服务器或 All-Reduce 等交换策略. 通常，DNN 使用如式(1)所示的小批量随机梯度下降模型更新权值：

$$W_{t+1} = W_t - \frac{\lambda_{\mathrm{lr}}}{n}\sum_{i=1}^{n}\nabla W_i, \tag{1}$$

其中 λ_{lr} 和 n 分别代表学习率和批次大小，每批次训练结束后使用参数服务器或 All-reduce 交换权值梯度 $\sum_{i=1}^{n}\nabla W_i$ 更新权值. 由于在多数情况下，数据并行的通信开销和同步开销小于模型并行开销，因此其加速比优于模型并行，所以，通常使用如图 2 所示的数据

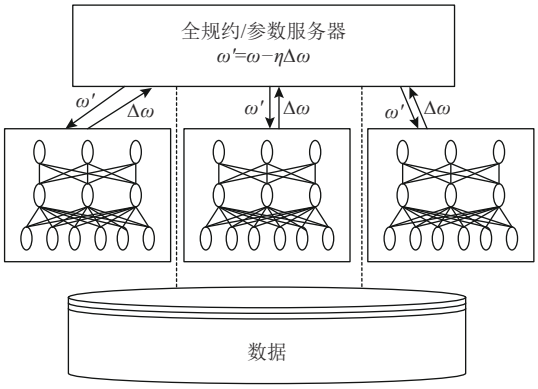


Fig. 2 Illustration of data parallel training of DNN model

图 2 深度神经网络模型数据并行训练示意图

并行来完成分布式并行 DNN 训练.

1.1.2 单节点并行研究现状

在单节点并行研究方面，往往利用多核和众核芯片，如 GPU, MIC, FPGA 对 DNN 模型进行并行加速.

1)在 GPU 研究方面，Raina 等人[9]使用 GPU 对深度置信网络训练进行了加速；Li 等人[10]使用单块 Tesla K40c GPU 优化深度置信网络预训练过程，最终相较于 Intel Core i7-4790K 实现了 14~22 倍加速比；Bottleson 等人[11]使用 OpenCL 优化了 GPU 卷积操作，相较 Intel CPU, CNN 训练得到了 2.5~4.6 倍加速比. Kaler 等人[12]使用领域采样器、共享内存并行优化、GPU 计算批传输流水线等技术将图神经网络训练速度相较于 Pytorch-Geometric 基准提升了 3 倍. Tan 等人[13]实现了基于 GPU 的隐私深度学习系统，相较于 CPU 平台的>最新算法提升了 8~36 倍的隐私深度学习训练加速.

2)在 MIC 研究方面，Viebke 等人[14]使用原生模式依托数据并行技术在 224 线程上达到了单线程版本 103.5 倍加速比；Liu 等人[15]以卸载模式同样采用数据并行技术在 240 线程下达到了单线程版本 131 倍加速比；Zlateski 等人[16]提出了 3 维卷积算法，在 Intel Xeon Phi 5110P 众核处理器上取得了 90 倍加速比.

3)在 FPGA 研究方面，Suda 等人[17]实现了 VGG 和 AlexNet 在 FPGA 平台的训练，在 VGG 中相较于 IntelI5 处理器达到 5.5 倍加速比. Zhang 等人[18]实现了性能可达近 1.8 TOPS 的 FPGA VGG 网络，相较于至强 E5-1530 处理器其加速比可达 4.4 倍. Aydonat 等人[19]在 FPGA 平台上将 Winograd 矩阵乘优化算法融入到 AlexNet 网络中，在 Arria 10 上达到了 1 382 GFLOPS 的峰值性能.

1.1.3 大规模分布式并行研究现状

随着 DNN 模型的不断复杂和加深，单节点已无法满足训练要求. 使用高性能计算平台实现 DNN 模型大规模分布式并行训练成为国内外研究者重点关注的领域.

1)在以 CPU 为主的高性能计算平台研究方面，Google 公司的 Dean 等人[8]在 CPU 集群上开发了 Google 深度学习软件框架 DistBelief；朱传家等人[20]针对“神威·太湖之光”超算平台对深度学习框架 Caffe 进行分布式扩展研究，其在特定模型上通信性能提高了 98 倍；魏嘉等人[21]在国产 E 级原型机天河三号上基于 Pytorch 平台实现了分布式并行 DNN 模型训练，在处理批次大小不变的情况下，在 VGG16 网络上实现了相较单节点 24.59 倍的加速比. Ji 等人[22]在 32 个节点的 Intel knights CPU 集群上利用共享和分布式内存实现了 Word2Vec 算法的跨节点分布式计算并实现了近线性加速比. Das 等人[23]优化了多节点同步 SGD 算法，分别在 64 个 Intel Xeon 节点 VGG-A CNN 模型和 16 个节点的语音识别深度学习模型上实现了 53 倍和 6.5 倍的加速比. Roy 等人[24]提出的非一致性内存访问(non-uniform memory access, NUMA)感知的分层并行化技术分别在 1~4 个 Intel 和 AMD NUMA 节点上相较于原始 Caffe 平台提升了吞吐量和拓展性[25].

2)在以 GPU 为主的高性能计算平台研究方面，Awan 等人[26]利用协同设计的技术在 12 个节点的集群中使用 80 张 Tesla K80 GPU 卡设计了分布式深度学习框架 S-Caffe；美国橡树岭国家实验室的 Yin 等人[27]在 SUMMIT 超算上使用容器技术，基于 Tensorflow 和 Pytorch 在 64 个节点上使用 384 张 Nvidia Tesla V100 GPU 卡，将 ResNet50 模型训练时间缩短至 12 min. Duan 等人[28]提出了 Mercury 调度策略在 MXNet 平台上针对 ResNet50 和 BERT 等模型进行 8 个节点 8GPU 分布式训练，相较基础 MXNet 平台分布式深度学习算法的加速比提升了 1.18~2.18 倍. Huang 等人[29]提出了 Gpipe 批间流水线并行训练算法，分别在 8 个 GPU 和 128 个 TPU 上基于 Lingvo 平台训练了用于图像分类的 AmoebaNet 模型和多语言机器翻译的 128 层 Transformer 模型. Narayanan 等人[30]提出了 Pipedream 批内流水线并行算法，在 Pytorch 平台下进一步流水线并行训练速度提升了 5.3 倍. Rajbhandari 等人[31]基于 DeepSpeed 平台提出了 ZeRO-Infinity 异构分布式训练系统，在 512 个 Nvidia V100 GPUs 集群上训练超过 20 万亿参数的 Transformer 模型.

1.2 能耗模型研究

1.2.1 能耗与功率

能耗(energy consumption)是一个系统在一段时间内所消耗的总能量(即系统做的总功)，而功率(power)则代表系统完成某项任务所消耗能量的速率[32]. 二者的关系可以用式(2)来表示：

$$E = PT, \tag{2}$$

其中 E 代表系统的能耗(单位为 J)，P 代表系统的功耗(单位为 W)，T 表示系统运行的时间(单位为 s). 如果 P 是随时间 T 变化的，那么在时刻 t_1 到时刻 t_2 时间内的总能耗进一步地可以增强地表示为

$$E = \int_{t_1}^{t_2} P_t \mathrm{d}t. \tag{3}$$

1.2.2 静态功率与动态功率

互补金属氧化物半导体(CMOS)材料由于其对

噪声的弹性和相比其他元件的低产热优势成为了当今计算机微处理器的主流选择. 数字 CMOS 电路功耗可以被划分为动态功率(晶体管正常工作时的功耗)和静态功率(晶体管关闭时漏电流产生的功率)2部分:

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{static}}, \tag{4}$$

其中动态功耗被认为是 CMOS 中功耗的主要来源. 可以用式(5)来表示[33-34]:

$$P_{\text{dynamic}} = A \times C \times V^2 \times f, \tag{5}$$

其中 A 代表开关活动(activity),即每个时钟周期内的开关数, C 表示物理电容, V 表示电压, f 代表时钟频率.

因为漏电流的存在,即使在晶体管关闭时,也会产生与晶体管数目正相关的静态功率,如式(6)所示:

$$P_{\text{static}} \propto I_{\text{static}} V, \tag{6}$$

其中 I_{static} 代表漏电流.

1.2.3 处理器能耗模型

处理器巨大的能耗已经成为了高性能计算平台服务器主要的能耗来源[35]. 目前对处理器能耗建模的方法主要为 2 种:一种是对处理器进行数字电路级建模,这种方法更加准确但成本更高;另一种的方法是使用统计模型,这种技术计算更快,但依赖于处理器评价计数器能快速地测量处理器主要组件的能耗和数据分析工具对这些能耗数据的有效数据处理. 本节分别介绍了单核中央处理器、多核中央处理器和图像处理器的能耗建模方法.

Shin 等人[36]使用式(4)加法模型表征单核处理器的功率,其中动态功率使用式(5)计算,而因为静态功率与温度高度相关,所以在静态功率中引入了温度 T_{d},表示为

$$P_{\text{s}}(T_{\text{d}}) = V_{\text{dd}}\left(K_1 T_{\text{d}}^2 \text{e}^{\frac{K_2 V_{\text{dd}} + K_3}{T_{\text{d}}}} + K_4 \text{e}^{(K_5 V_{\text{dd}} + K_6)}\right), \tag{7}$$

其中 $K_i (i = 1,2,3,4,5,6)$ 代表技术常数.

Merkel 等人[37]通过计算特定时间段内发生事件的数量来确定处理器在该时段所消耗的总能量:

$$E = \sum_{i=1}^{n} \alpha_i C_i, \tag{8}$$

其中 α_i 表示某个事件在一段时间内出现的次数, C_i 代表某个事件所花费的能耗, n 表示所有可能的事件总数.

接下来,本文将从典型的功率组件分解建模和基于排队论的功率建模来介绍多核 CPU 功耗模型的发展.

Bertran 等人[38]提出类似单核的模块分解功率模型,假设每个核的性能相同,功率模型为:

$$P_{\text{total}} = \left(\sum_{j=1}^{n_{\text{core}}}\left(\sum_{i=1}^{m} A_{ij} P_i\right) + P_{\text{static}}\right), \tag{9}$$

其中, P_i 代表处理器核中每个组件的功率, A_{ij} 表示第 j 个核的第 i 个模块的活动频率.

Li 等人[39]将多核 CPU 建模为一个具有多台服务器的 M/M/m 排队系统[40],处理器的总功率表示为

$$P_{\text{total}} = m(\rho P_s + P^*) = \lambda R P_s + m P^*, \tag{10}$$

其中 m 表示处理器核数, ρ 表示处理器核利用率, P_s 表示在速度为 s 时处理器核的功率, P^* 表示处理器核不工作时的功率, λ 表示任务到达率, R 表示每个任务平均执行指令数目.

图像处理单元(GPUs)正逐步成为高性能计算平台的常见组件,鉴于其支持大规模硬件并发操作,尤其适合 DNN 训练等人工智能技术,许多现代高性能计算平台都配备了通用图像处理单元[41]. 然而,对图像处理器的功耗测量、建模和预测仍处于起步阶段.

现有大部分研究都是基于性能评测计数器对 GPU 功率进行建模,典型的 GPUWattch 就是用式(11)进行功率建模.

$$P = \sum_{i=1}^{N} (\alpha_{ai} P_{\text{max}i}) + P_{\text{idlesm}} + P_{\text{static}}, \tag{11}$$

其中 GPU 的动态功率被记为每个组件的活动因子 α_{ai} 与每个组件峰值功率的乘积, P_{idlesm} 和 P_{static} 分别代表空闲流处理器组件功率和静态功率.

除此之外, Google TPU[42]等神经网络专用加速设备逐步成为高性能计算平台的重要组件. Zhang 等人[43]针对类似 TPU 的处理设备提出了能耗模型:

$$E = E_{\text{MAC}} + E_{\text{DRAM to SRAM}} + E_{\text{SRAM}} + E_{\text{rest}}, \tag{12}$$

其中 E_{MAC} 代表 MAC 阵列计算能耗, $E_{\text{DRAM to SRAM}}$ 代表 DRAM 能耗(即从 DRAM 读写数据到 SRAM 的能耗), E_{SRAM} 代表 SRAM 实现激活和权值缓冲区的能耗, E_{rest} 代表其余组件能耗,包括控制、激活、池化、归一化等组件的能耗.

1.2.4 内存能耗模型

虽然传统上处理器被认为是服务器平台能耗的主要来源,但随着数据密集型应用的不断增加和相应的数据密集型高性能平台的不断发展,内存在高性能计算中心中能耗比重大幅度增加. 本节中,我们介绍内存能耗模型.

作为服务器中第 2 大功耗来源的内存[44],在高性能计算平台中,主存功耗也占到了服务器功耗的 30%[45], DRAM 存取容量和带宽的快速增长导致了 DRAM 子系统消耗了高性能计算平台很大一部分能量. 目前,

大部分对内存能耗研究集中于对 DRAM 能耗的研究，主要方法与处理器类似，也是使用数字电路组件级建模和基于内存性能评价计数器.

Lin 等人[46] 提出的基于静态功率和动态功率累加的计算模型公式为

$$P_{\mathrm{DRAM}} = P_{\mathrm{staticDRAM}} + \alpha_1 \mu_{\mathrm{read}} + \alpha_2 \mu_{\mathrm{write}}, \tag{13}$$

其中α_1是取决于处理器的常数，DRAM 的静态功率用$P_{\mathrm{staticDRAM}}$来表示，读写的吞吐量分别使用μ_{read}和μ_{write}来表示.

Vijaykrishnan 等人[47] 提出的另一种类似的累加模型存储系统的全部能耗被建模为

$$E = E_{\mathrm{icache}} + E_{\mathrm{dcache}} + E_{\mathrm{buses}} + E_{\mathrm{pads}} + E_{\mathrm{mem}}, \tag{14}$$

其中，E_{icache}和E_{dcache}分别代表指令高速缓存和数据高速缓存能耗，E_{buses}代表数据高速缓存和指令高速缓存之间总线开销，E_{pads}代表主存与外设之间的总线开销，E_{mem}[48] 是中描述的内存开销.

Roy 等人[49] 提出了一种针对某个算法 A 的完全不同的基于内存性能评估计数器的内存能耗建模方法：

$$E_{\mathrm{mem}}(A) = P_{\mathrm{cke}} T(A) + P_{\mathrm{stby}} T_{\mathrm{act}}(A)\alpha(A) + E_{\mathrm{act}}\alpha(A) + (R(A) + W(A)) T_{\mathrm{rdwr}} P_{\mathrm{rdwr}}, \tag{15}$$

其中$\alpha(A)$, $R(A)$,$W(A)$ 分别代表算法 A 激活阶段指令周期数、读次数和写次数. 内存总功率包括：P_{cke}代表静态功率，即所有存储单元均未激活时的功率，P_{stby}指用于激活存储组件和等待指令的功率，E_{act}代表激活阶段捕获各种指令的能耗，P_{rdwr}代表内存进行读写的功率. 各阶段功率与对应时间之积再累加就是内存系统的总能耗.

1.2.5　应用系统能耗模型

除了针对各种元器件的物理特征建立能耗模型之外，考虑各种不同的应用和工作负载的类型建立针对性的能耗模型也是同样重要的研究问题.

Poess 等人[50] 针对典型的数据仓库应用，基于 TPC-H 基准，开发了企业数据仓库功耗模型. 该模型旨在估计系统的峰值功率：

$$P_{\mathrm{ser}} = (C_{\mathrm{cpu}} P_{\mathrm{cpu}} + 9C_{\mathrm{mem}} + C_{\mathrm{d}} P_{\mathrm{d}}) \times 1.3 + 100, \tag{16}$$

其中P_{ser}代表整个服务器的功率，C_{cpu}代表每台服务器 CPU 的数量，P_{cpu}是他们研究的处理器设计功率，C_{mem}是每个服务器中内存 DIMMs 的数量，C_{d}是每个服务器中的磁盘数量，P_{d}代表每个磁盘的功率. Poess 等人[50] 增加了 30% 的上述组件功率和额外的 100 W 功率来解释机箱的电源开销.

Feng 等人[51] 针对分布式集群系统中常见的 MapReduce 应用提出了能耗加法模型：

$$E = PT = P_{\mathrm{i}} T_{\mathrm{i}} + P_{\mathrm{m}} T_{\mathrm{m}} + P_{\mathrm{s}} T_{\mathrm{s}} + P_{\mathrm{r}} T_{\mathrm{r}}, \tag{17}$$

将 MapReduce 的能耗定义为各部分功率与运行时间乘积的和. $(P_{\mathrm{i}}, T_{\mathrm{i}})$代表初始化阶段，$(P_{\mathrm{m}}, T_{\mathrm{m}})$表示映射阶段，$(P_{\mathrm{s}}, T_{\mathrm{s}})$代表中间数据洗牌阶段，$(P_{\mathrm{r}}, T_{\mathrm{r}})$代表规约阶段. 同时提出了 4 个影响 MapReduce 应用能耗的因素，分别是工作负载的 CPU 密集程度、I/O 密集度、副本数量和块大小.

Yang 等人[52] 针对单机 DNN 尤其是在嵌入式设备上进行推理时的能耗进行了建模. 他们并未考虑具体的硬件功耗，而是使用归一化的乘加操作能耗作为能耗的单位. 每一层的能耗为

$$E_{\mathrm{layer}} = E_{\mathrm{comp}} + E_{\mathrm{data}}, \tag{18}$$

其中E_{comp}和E_{data}分别代表每一层的计算开销和数据输入、数据输出以及卷积核的访存开销，并使用网络中所有卷积层和全连接层（全连接层可以视为特殊的卷积层）的能耗代表该 DNN 的总能耗.

1.2.6　能耗模型研究现状总结

综上所述，目前的能耗模型主要分为针对硬件的能耗模型和针对应用的能耗模型 2 种.

针对硬件的能耗模型只给出了器件在不同状态下的能耗情况而未考虑上层应用的情况[37]，因而无法获得某个应用具体的内耗情况，而针对应用的能耗模型往往将程序运行时的服务器当成整体，采用测量方式来得到某个应用的能耗[51]，这种测量的方法成本很高，每一次对应用的微调都要重新测量整个应用；或者将某种能耗用归一化乘加操作表征[51]，无法与具体的硬件能耗开销相对应.

而本文所提出的 MC^2 模型通过分析大规模 DNN 分布式训练时的运行特点，通过转化底层硬件的能耗表现，将上层 DNN 应用程序状态变化与底层硬件能耗变化结合起来，只需要执行少量的步骤，无需执行完整的训练任务，且无需额外的能耗测量就可以较为精准地计算得到具体 DNN 分布式训练任务的总能耗.

2　MC^2 能耗模型

针对现有应用程序能耗获取方法存在的与底层硬件关联不紧密、统计成本高等问题，本文提出了 MC^2 能耗模型. 该模型分析了大规模 DNN 分布式训练时的运行特点，并将这些特点转化为底层硬件的能耗表现. MC^2 能耗模型把上层 DNN 应用程序状态变化映射为底层硬件能耗变化，只需要进行少量实

验, 无需运行整个程序, 就可以计算得到具体的 DNN 分布式训练任务总能耗.

如图 3 所示, MC^2 能耗模型将 DNN 模型大规模分布式数据并行训练分为数据预处理、前向与反向传播以及梯度同步与更新 3 个阶段.

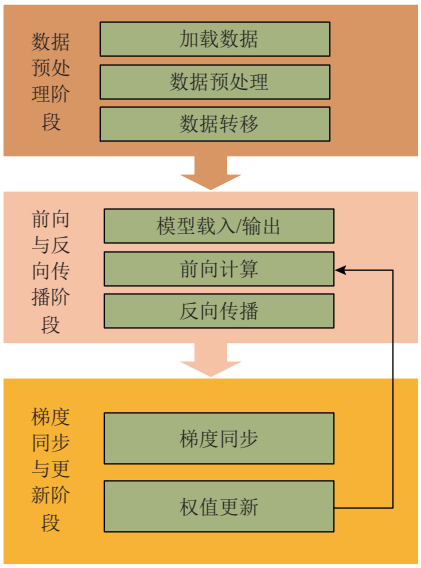


Fig. 3 Structure of MC^2 energy consumption model

图 3 MC^2 能耗模型结构

各阶段的功耗都映射为 CPU、内存、硬盘和网络设备功耗的组合, MC^2 能耗模型并不对这些设备组件的功耗信息来源进行约束, 组件的能耗来源既可以是真实环境下的测量值, 也可以通过第 2 节中对应的设备功率模型获取, 组件功耗的准确性将很大程度影响 MC^2 能耗模型最终的准确性.

2.1 数据预处理阶段

良好的数据质量是人工智能算法成功的重中之重. 数据预处理阶段是深度学习最初始的阶段, 也是最为关键的阶段之一. 由于数据集的来源多种多样, 数据本身可能存在数据缺失、数据噪声、数据不一致、数据冗余、数据集不均衡、数据离群点、数据重复等问题, 因此在数据被输入模型进行训练之前, 必须进行一系列的数据预处理操作.

本文所述的数据预处理包括数据集成、数据规约和数据变换, 这是 DNN 模型训练前最常用的数据预处理手段. 虽然数据清理也是一项关键且耗时的工作, 但由于数据清理时间高度依赖于数据来源和数据质量, 往往这一过程在整个深度学习开始之前就预先完成, 因此本文进行预处理的数据已经完成了数据清理.

在 DNN 模型大规模分布式数据并行训练时, 数据预处理在每个节点中单独完成了, 每个节点获取所有数据集的 $1/N$(N 代表参与训练的节点数量), 然后使用各节点的 CPU 完成数据预处理操作. 具体操作为:

首先, 每个节点都需要对自己的数据集建立索引, 以便 CPU 能够找到数据集存放位置的起始地址以及获取数据集的长度, 在这一过程中仅仅为数据集建立了索引而没有把任何训练数据导入内存中, 因为现代 DNN 训练数据集规模往往非常庞大, 内存无法也无需同时容纳所有的数据集, 只需要每次提取 1 个批次的数据即可, 这一部分的功率为

$$P_{\text{setdataset}} = P_{\text{gs}} + P_{\text{sm}} + P_{\text{sCPU}}, \tag{19}$$

其中 P_{gs} 为 2 级存储寻找数据集起始位置和读取数据集长度的开销, P_{sm} 是将起始位置索引和长度写入内存的开销, P_{sCPU} 表示 CPU 在这一阶段发出读写等指令的功率.

随后, 由于模型使用小批量随机梯度下降进行训练, 每次需要每个节点从存储介质(SSD/HDD)中取出 1 个批次(大小为 B)的数据到内存中, 这一过程的功率为

$$P_{\text{loaddata}} = P_{\text{rs}} + P_{\text{wm}} + P_{\text{lCPU}}, \tag{20}$$

其中 P_{rs} 代表从底层 2 级存储读取数据的功率(包括空闲、寻道、读取等功耗), P_{wm} 表示将硬盘数据写入内存的功率, P_{lCPU} 表示 CPU 在这一阶段发出读写等指令的功率.

最后, 节点 CPU 对自己内存中的需要预处理的数据分别执行数据集成、数据规约、数据变换操作, 此部分功率为

$$P_{\text{preprocess}} = P_{\text{pm}} + P_{\text{pCPU}}, \tag{21}$$

其中 P_{pm} 表示数据预处理阶段 CPU 读写内存时内存产生的功率, P_{pCPU} 表示 CPU 在执行各种数据预处理操作时的功率.

如果进行大规模训练的高性能计算平台使用了 GPU, TPU, DPU 等其他设备, 还需要将内存中预处理完的数据转移到对应的设备上, 这一部分可选的开销为

$$P_{\text{transfer}} = P_{\text{rm}} + P_{\text{wgtd}} + P_{\text{tCPU}}, \tag{22}$$

其中 P_{rm} 代表内存读功率, P_{wgtd} 代表 GPU, TPU, DPU 等加速设备内存的写功率, P_{tCPU} 表示 CPU 在这一阶段发出读写指令的功率.

因为高性能计算平台通常使用同构节点完成同一个给定的应用, 在数据并行情况下, 每个节点处理数据量相同, 所以可以假设每个节点上进行上述数据读取的时间 T_{loaddata}、CPU 进行数据预处理的时间

$T_{\text{preprocess}}$、内存转移到其他加速设备时间T_{transfer}几乎是相同的，所有节点全部执行完上述操作的总能耗为

$$E_{\text{pre}}=\frac{n}{NB}\sum_{i=1}^{N}\left(\frac{BN}{n}\int_{T_{\text{setdataset}}^{\text{begin}}}^{T_{\text{setdataset}}^{\text{end}}}P_{\text{setdataset}}^{i}\mathrm{d}t+\int_{T_{\text{loaddata}}^{\text{begin}}}^{T_{\text{loaddata}}^{\text{end}}}P_{\text{loaddata}}^{i}\mathrm{d}t+\int_{T_{\text{preprocess}}^{\text{begin}}}^{T_{\text{preprocess}}^{\text{end}}}P_{\text{preprocess}}^{i}\mathrm{d}t+\int_{T_{\text{transfer}}^{\text{begin}}}^{T_{\text{tranfer}}^{\text{end}}}P_{\text{transfer}}^{i}\mathrm{d}t\right).\tag{23}$$

当各节点功率稳定，不随时间发生剧烈变化时，可以进一步近似表示为

$$E_{\text{pre}}=\frac{n}{B}\times P_{\text{loaddata}}T_{\text{loaddata}}+P_{\text{preprocess}}T_{\text{preprocess}}+P_{\text{transfer}}T_{\text{transfer}}+N\times P_{\text{setdataset}}T_{\text{setdataset}},\tag{24}$$

其中n代表数据集的样本个数，$\frac{n}{B}$表示总共训练执行的批次数量. 类似地，如果各节点功耗稳定，下文2个阶段也可以使用类似的近似表示.

2.2 前向传播阶段和反向传播阶段

在DNN模型大规模分布式训练过程中，前向传播和反向传播是模型训练最核心的部分，也是整个训练中最耗时、最耗能的部分. 整个前向传播过程需要根据每一层的输入使用网络权值计算出每一层神经网络的输出，每层网络类型包括卷积层、池化层、全连接层、批量规约层、激活层等，每一层都对应着若干的参数；类似地，反向传播中需要根据前向传播计算得到的损失值和中间状态参数逐层计算出各层的权值梯度，然后执行后续的梯度同步与更新工作.

在这一阶段中，主要能耗可分为模型存取部分（包括模型权值、模型的超参数等需要存储至2级存储中的必备参数）、前向传播和反向传播每一层的输入读取和输出计算部分.

首先，模型存取部分，该部分类似于数据预处理的第1阶段，如果不使用加速设备，这部分功耗仅包括模型从硬盘读入内存的功耗和训练完成后将模型存入硬盘的功率：

$$P_{\text{inmodel}}=P_{\text{rs}}+P_{\text{wm}}+P_{\text{iCPU}},\tag{25}$$

$$P_{\text{outmodel}}=P_{\text{ws}}+P_{\text{rm}}+P_{\text{oCPU}}.\tag{26}$$

如果考虑使用加速设备，则还需要增加将模型转移到加速设备内存的功耗以及将模型参数从加速设备写回内存的功率：

$$P_{\text{ingtd}}=P_{\text{rm}}+P_{\text{wgtd}}+P_{\text{iCPU}},\tag{27}$$

$$P_{\text{outgtd}}=P_{\text{wm}}+P_{\text{rgtd}}+P_{\text{oCPU}}.\tag{28}$$

使用小批量梯度下降训练模型不仅是处理器能耗最多的部分，网络中每一层的输入和输出也是整个模型中消耗内存（CPU内存或加速设备内存）最严重的部分，例如在VGG16模型中能耗占到了90.41%.

在前向传播过程中，每一层的训练都包括：从内存（本节及下节的内存指广义的内存，包括显存等其他加速设备内存）中读取上一层的输出（第1层为读取训练数据）、读取本层神经网络权值、计算本层神经网络的输出、将本层神经网络输出存储至内存4部分，其中计算本层神经网络输出时CPU高速运转，而内存相对空闲，此部分功率为P_{f1}，该部分功率是前向传播功率的主要来源；而其余3部分内存高速运转而CPU相对空闲，可以合并为一个部分计算功率，用P_{f2}来表示：

$$P_{\text{f1}}=P_{\text{cgtdf1}}+P_{\text{mf1}},\tag{29}$$

其中P_{cgtdf1}代表在第1部分使用CPU或其他加速设备进行网络计算的功率，P_{mf1}表示第1部分的内存功率.

$$P_{\text{f2}}=P_{\text{cgtdf2}}+P_{\text{mf2}},\tag{30}$$

类似地，P_{cgtdf2}和P_{mf2}分别代表在第2阶段的加速设备和内存的功率.

与前向传播类似，反向传播是自最顶层至输出层链式求导计算各层梯度值以进行权值更新，反向传播每一层的训练也包括：从内存中读取上一层中间变量的输出、读取本层神经网络权值、计算本层梯度值、读取后一层梯度值和输出本层反向传播计算出的梯度值到内存5部分，由于不同部分处理器和内存表现的差异，反向传播也可以将计算梯度部分剥离功率记为P_{b1}，该部分功耗也是反向传播功耗的主要来源，其余部分功率记为P_{b2}.

$$P_{\text{b1}}=P_{\text{cgtdb1}}+P_{\text{mb1}},\tag{31}$$

$$P_{\text{b2}}=P_{\text{cgtdb2}}+P_{\text{mb2}},\tag{32}$$

因此，前向传播与反向传播阶段的总能耗为

$$E_{\text{fb}}=\frac{n}{NB}\sum_{i=1}^{N}\left(\frac{BN}{n}\int_{T_{\text{inoutmodel}}^{\text{begin}}}^{T_{\text{inoutmodel}}^{\text{end}}}P_{\text{inoutmodel}}^{i}\mathrm{d}t+\frac{BN}{n}\int_{T_{\text{inoutgtd}}^{\text{begin}}}^{T_{\text{inoutgtd}}^{\text{end}}}P_{\text{inoutgtd}}^{i}\mathrm{d}t+\int_{T_{\text{f1}}^{\text{begin}}}^{T_{\text{f1}}^{\text{end}}}P_{\text{f1}}^{i}\mathrm{d}t+\int_{T_{\text{f2}}^{\text{begin}}}^{T_{\text{f2}}^{\text{end}}}P_{\text{f2}}^{i}\mathrm{d}t+\int_{T_{\text{b1}}^{\text{begin}}}^{T_{\text{b1}}^{\text{end}}}P_{\text{b1}}^{i}\mathrm{d}t+\int_{T_{\text{b2}}^{\text{begin}}}^{T_{\text{b2}}^{\text{end}}}P_{\text{b2}}^{i}\mathrm{d}t\right).\tag{33}$$

2.3 梯度同步与更新阶段

梯度同步技术是实现DNN模型大规模分布式训练的核心技术，在这一阶段中，如式(34)所示，整个分布式集群整合各个节点反向传播得到的梯度值，然后进行全规约操作，各个节点获取到更新后的梯度后，再在本地执行类似于式(35)所示的本地权值更新，随后开始新一批次数据的训练，直至训练轮次结束或损失值满足跳出结束条件. 在本阶段中，主要能耗来源于梯度全规约过程和本地权值更新过程.

$$\nabla W_{\text{average}} = \frac{1}{n}\sum_{i=1}^{n}\nabla W_i, \tag{34}$$

$$W_{t+1} = W_t - \lambda \nabla W_{\text{average}}. \tag{35}$$

全规约过程主要执行的是网络通信操作, 现有高性能集群通常使用高吞吐量、低延迟的高性能InfiniBand等网络, 这一部分功率主要来源于网络链路功率P_{cn}和网络设备功率P_{cd}, 以及CPU收发指令的功率P_{cCPU}与读写内存的功率P_{cm},

$$P_{\text{communication}} = P_{\text{cn}} + P_{\text{cd}} + P_{\text{cCPU}} + P_{\text{cm}}. \tag{36}$$

本地权值更新是使用网络获取到的平均梯度更新各节点的权值, 这一部分开销主要来源于CPU或加速设备的计算开销P_{ucgtd}和对内存的读写开销P_{um}:

$$P_{\text{update}} = P_{\text{ucgtd}} + P_{\text{um}}. \tag{37}$$

因此梯度同步与更新阶段的总能耗为

$$E_{\text{cu}} = \frac{n}{NB}\sum_{i=1}^{N}\left(\int_{T^{\text{begin}}_{\text{communication}}}^{T^{\text{end}}_{\text{communication}}} P^i_{\text{communication}}\,\mathrm{d}t + \int_{T^{\text{begin}}_{\text{update}}}^{T^{\text{end}}_{\text{update}}} P^i_{\text{update}}\,\mathrm{d}t\right). \tag{38}$$

2.4 同步等待功耗补偿

当计算节点出现性能表现差异时, 就需要进行同步等待功耗补偿. 各节点计算性能差异主要有2个原因: 一个原因是节点之间本身就是异构的, 异构节点执行相同任务的处理时间肯定不同, 所以那些计算较快的节点就需要等待计算较慢的节点完成计算任务; 另一个原因是即使是完全同构的节点, 仍然可能因为制造工艺水平、主频稳定性、节点负载情况等不同导致处理相同任务的完成时间略有差异, 我们的"同步补偿"针对的是这2种情况.

具体地, 在如果在进行DNN大规模分布式数据并行训练且使用BSP一致性协议时, 分布式系统中的快节点就要在梯度同步阶段等待慢节点完成本地的权值梯度计算, 各节点的等待时间取决于该节点与最慢节点的计算效率之差, 在这段时间内快节点内存和慢节点内存等计算设备均处于空闲等待状态, 同步等待补偿功率为

$$P_{\text{add}} = \sum_{i=1}^{N}\left(P_{\text{cgtdidle}} + P_{\text{midle}}\right), \tag{39}$$

P_{cgtdidle}和P_{midle}分别代表计算设备和内存设备空闲状态的功率. 本阶段的总能耗为

$$E_{\text{add}} = \frac{n}{NB}\sum_{i=1}^{N} T_i(P_{\text{cgtdidle}} + P_{\text{midle}}), \tag{40}$$

其中T_i表示每一批次训练过程中节点i等待最慢节点完成计算的等待时间.

综上所述, DNN模型大规模分布式数据并行训练总能耗为

$$\begin{aligned}
E_{\text{total}} = E_{\text{pre}} + E_{\text{fb}} + E_{\text{cu}} = \frac{n}{NB}\sum_{i=1}^{N}\Bigg(&\frac{BN}{n}\int_{T^{\text{begin}}_{\text{setdataset}}}^{T^{\text{end}}_{\text{setdataset}}} P^i_{\text{setdataset}}\,\mathrm{d}t + \int_{T^{\text{begin}}_{\text{loaddata}}}^{T^{\text{end}}_{\text{loaddata}}} P^i_{\text{loaddata}}\,\mathrm{d}t + \int_{T^{\text{begin}}_{\text{preprocess}}}^{T^{\text{end}}_{\text{preprocess}}} P^i_{\text{preprocess}}\,\mathrm{d}t + \int_{T^{\text{begin}}_{\text{transfer}}}^{T^{\text{end}}_{\text{tranfer}}} P^i_{\text{transfer}}\,\mathrm{d}t + \\
&\frac{BN}{n}\int_{T^{\text{begin}}_{\text{inoutmodel}}}^{T^{\text{end}}_{\text{inoutmodel}}} P^i_{\text{inoutmodel}}\,\mathrm{d}t + \frac{BN}{n}\int_{T^{\text{begin}}_{\text{inoutgtd}}}^{T^{\text{end}}_{\text{inoutgtd}}} P^i_{\text{inoutgtd}}\,\mathrm{d}t + \int_{T^{\text{begin}}_{\text{f1}}}^{T^{\text{end}}_{\text{f1}}} P^i_{\text{f1}}\,\mathrm{d}t + \int_{T^{\text{begin}}_{\text{f2}}}^{T^{\text{end}}_{\text{f2}}} P^i_{\text{f2}}\,\mathrm{d}t + \\
&\int_{T^{\text{begin}}_{\text{b1}}}^{T^{\text{end}}_{\text{b1}}} P^i_{\text{b1}}\,\mathrm{d}t + \int_{T^{\text{begin}}_{\text{b2}}}^{T^{\text{end}}_{\text{b2}}} P^i_{\text{b2}}\,\mathrm{d}t + \int_{T^{\text{begin}}_{\text{communication}}}^{T^{\text{end}}_{\text{communication}}} P^i_{\text{communication}}\,\mathrm{d}t + \int_{T^{\text{begin}}_{\text{update}}}^{T^{\text{end}}_{\text{update}}} P^i_{\text{update}}\,\mathrm{d}t + T_i(P_{\text{cgtdidle}} + P_{\text{midle}})\Bigg) = \\
&\frac{n}{NB}\sum_{i=1}^{N}\Bigg(\frac{BN}{n}\left(\int P^i_{\text{setdataset}}\,\mathrm{d}t + \int P^i_{\text{inoutmodel}}\,\mathrm{d}t + \int P^i_{\text{inoutgtd}}\,\mathrm{d}t\right) + \int P^i_{\text{loaddata}}\,\mathrm{d}t + \int P^i_{\text{preprocess}}\,\mathrm{d}t + \\
&\int P^i_{\text{transfer}}\,\mathrm{d}t + \int P^i_{\text{f1}}\,\mathrm{d}t + \int P^i_{\text{f2}}\,\mathrm{d}t + \int P^i_{\text{b1}}\,\mathrm{d}t + \int P^i_{\text{b2}}\,\mathrm{d}t + \int P^i_{\text{communication}}\,\mathrm{d}t + \int P^i_{\text{update}} + T_i(P_{\text{cgtdidle}} + P_{\text{midel}})\Bigg).
\end{aligned} \tag{41}$$

3 实验结果与分析

3.1 实验对象

3.1.1 实验硬件配置

本文的实验平台天河三号E级原型机是由国防科技大学和天津超算中心联合开发的国产E级原型超级计算机之一[53]. 其计算节点包括Phytium FT-2000+(FT)和Phytium MT-2000+(MT)两种, 每个FT包含32个armv8架构的FTC662处理器核和64 GB内存, 处理器工作主频在2.4 GHz, 片上集成了32 MB的2级cache, 可提供204.8 Gbps访存带宽, FT节点典型工作能耗约为96 W; 而MT计算节点, 每个节点包含32个处理器核和16 GB内存, 主频最高可达2.0 GHz, MT节点典型工作功耗为135 W.FT计算节点和MT计算节点处理器的典型功耗开销分别约为50 W和60 W[54]. 值得一提的是, 这2种处理器都支持128 b的SIMD向量或标量浮点操作, 这对于深度学习而言非常关键. 天河三号软硬件基本情况如表2所示.

本文实验中分别使用32个FT节点(1 024处理器核和2 048 GB内存)和128个MT节点(4 096个处理器核和2 048 GB内存)进行DNN模型大规模分布式数

Table 2 Basic Information of Tianhe-3 Prototype

表 2 天河三号原型机基本信息

类别	属性	FT-2000+	MT-2000+
硬件	节点数	128	512
	单节点核数	32	32
	频率/GHz	2.4	2.0
	内存容量/GB	64	16
	网络互联带宽/Gbps	200	
软件	操作系统	kylin 4.0-1a OS with kernel v4.4.0	
	文件系统	lustre	
	MPI 版本	mpich v3.2.1	
	编译器	gcc v4.9.1/v4.9.3	
	库	Boost，BLAS，openBLAS，scalapack 等	

据并行训练实验并对实验能耗进行建模分析.

3.1.2 实验数据集

本文用于目标识别任务的数据集为 Cifar-10 和 ImageNet. Cifar-10 是由 Hinton 的学生 Alex Krizhevsky 和 Ilya Sutskever 整理的一个用于识别普适物体的小型数据集. 一共包含 10 个类别的 RGB 彩色图片：飞机、汽车、鸟类、猫、鹿、狗、蛙类、马、船和卡车. 图片的尺寸为 32×32，数据集中一共有 50 000 张训练图片和 10 000 张测试图片. ImageNet 是一个用于视觉对象识别软件研究的大型可视化数据库，超过 1 400 万的图像 URL 被 ImageNet 手动注释，以指示图片中的对象；为至少 100 万个图像提供了边界框.

3.1.3 实验模型

本文选择的 DNN 模型为 VGG16[55]，ResNet50[56]和 Vision Transformer（ViT）[57]. 本文实验中采用的网络模型均来自开源 Torchvision 库[58]，在使用时只需要根据实际分类数量设置参数 *num_classs*.

VGG16 模型结构如图 4 所示，VGG16 沿用了 AlexNet[59] 的基本结构，但深度比起 AlexNet 有所提升，其有 13 个卷积层和 3 个全连接层，具体由 5 组“多卷积层-池化层-Batchnorm+ReLU”加上全连接层构成，使用 softmax 获得平均结果. 在 VGG16 模型中，用 3 个 3×3 卷积核代替了 7×7 卷积核，用 2 个 3×3 卷积核代替了了 5×5 卷积核，从而保证在相同感知野的条件下提升了网络的深度，在一定程度上提升了神经网络的效果，也因此需要大量的计算开销，适合于使用大规模分布式环境进行网络模型训练.

通过对 VGG16 模型进行分析可以得出，在使用包含动量的随机梯度下降算法对 VGG16 模型进行训练时，模型参数（包括前向传播和反向传播中的参数）占用的内存为 1.54 GB，每张图片所有层次的输入输出中间状态内存占用为 116.24 MB，本实验单节点批次大小为 64，所以实验过程中每批次训练的内存开销约为 8.81 GB.

ResNet 模型为了解决更深层次的神经网络难以训练的问题，提出了新型的 building block 结构的去克服退化问题. 如图 5 所示，ResNet 在几个堆叠层之间并没有采用直接映射，而是添加了一条额外分支并显式地拟合了残差映射. 图 5(a)是 ResNet34 中所用的构建块结构；图 5(b)是在 ResNet50 模型中为了有效减少训练时间，在原来的基础上修改得到的构

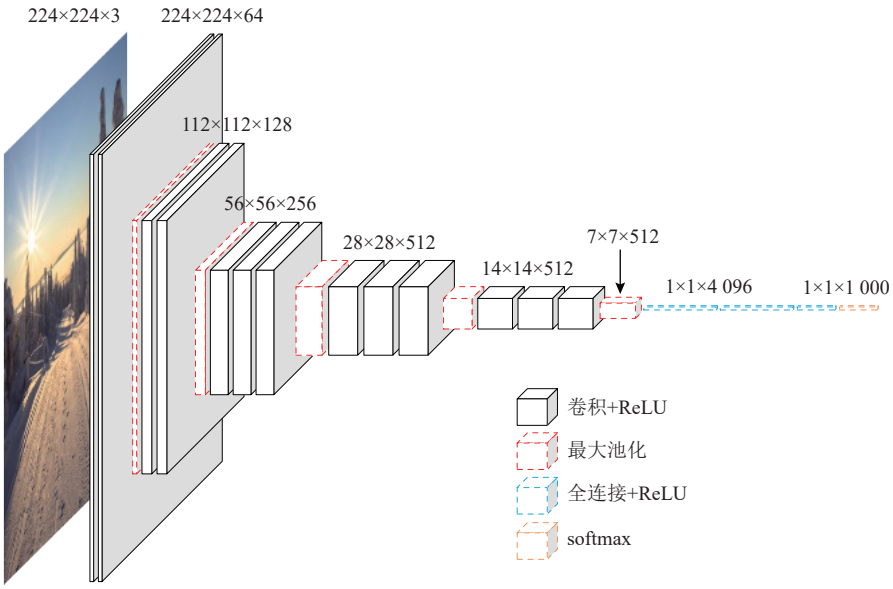
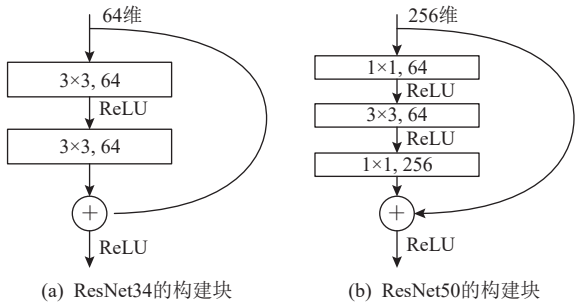


Fig. 4 Structure of VGG16 neural network model[55]

图 4 VGG16 神经网络模型结构[55]



(a) ResNet34的构建块 (b) ResNet50的构建块

Fig. 5 Building blocks in ResNet[56]

图 5 ResNet 中的构建块[56]

建块结构. 即对于每个残差函数使用 3 层的堆叠去拟合, 其中各层应用的分别是 1×1, 3×3, 1×1 的卷积核.

如图 6 所示, 在 ResNet34 与 ResNet50 模型中都拥有 4 个 3 层构建组, 每个构建组分别有 3 个、4 个、6 个、3 个块, 4 个构件组相加为 48 块. 除此之外, 在每个构建组的第 1 层都有 1 个单独的卷积层, 所以共 49 块卷积层. 使用平均池化以及 softmax 获得最终结果. 在极大地增加网络深度之后, 网络的效果得到极其显著的提升, 同时伴随而来的也包括高额的计算开销. 因此, 同样适合于使用大规模分布式环境进行网络模型训练.

自从深度学习暴发以来, CNN 一直是计算机视觉(CV)领域的主流模型, 而且取得了很好的效果, 相比之下 Transformer 却独霸自然语言处理(NLP)领域, Transformer 在 CV 领域的探索正是学术界想把 Transformer 在 NLP 领域的成功借鉴到 CV 领域. 视觉变压器(vision Transformer, ViT)模型是 Google 公司在 2020 年提出的直接将 Transformer 应用在图像分类的模型, 后续很多研究都基于 ViT 进行改进.ViT 原理如图 7 所示, 图像被分成固定大小的分片, 然后通过线性变换得到分片嵌入, 这就类比 NLP 的词和词嵌入, 由于 Transformer 的输入就是一系列的符号嵌入(token embeddings), 所以将图像的分块嵌入送入 Transformer 后就能够依据特征提取而分类了.ViT 模型已经在许多 CV 领域超越了基于卷积神经网络的模型达到了最优的效果.

层名称	ResNet34	ResNet50
Conv1	7×7, 64, 步长为2	
Conv2_x	3×3最大池化, 步长为2	
	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
Conv3_x	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$
Conv4_x	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1\,024\end{bmatrix}\times6$
Conv5_x	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2\,048\end{bmatrix}\times3$
全连接层	平均池化、100维全连接、softmax	

Fig. 6 Structures of ResNet34 and ResNet50[56]

图 6 ResNet34 与 ResNet50 结构[56]

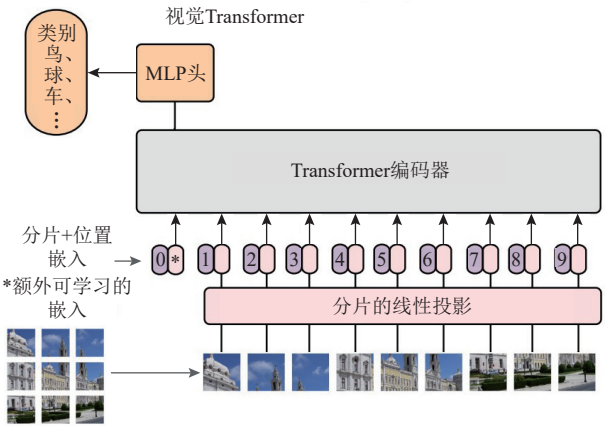


Fig. 7 Principle diagram of ViT model[54]

图 7 ViT 模型原理图[54]

为了进一步凸显本文提出的 MC^2 能耗模型的优越性, 本文还使用常用的与时间成比例的能耗模型(基准时间分别为 1 s, 4 s, 8 s, 16 s)和经典的 AR[60], SES[61], ARIMA[62] 模型, 3 种模型的训练时间步均为 500 作为参照的基线.

3.1.4 实验流程与测试目的

本文实验中使用 VGG16 模型和 ViT 模型对 Cifar-10 数据集; ResNet50 模型对 ImageNet 数据集分别进行了 1 个 epoch 完整训练, 并且每种配置下的实验均进行了 10 次, 在将各阶段的结果使用夏皮洛-威尔克检验[63] 证明 10 次实验的结果满足正态分布后(检验内容见附录 A), 使用 10 次实验的均值代表本文最终的实验结果.

本文实验部分旨在验证 MC^2 模型在大规模分布式数据并行 DNN 训练中能耗预测的有效性和可靠性. 预期通过使用 MC^2 模型、线性扩展的能耗模型以及 AR, SES, ARIMA 等经典能耗预测方法对 DNN 模型训练开销进行预测, 并与真实的训练开销进行对比, 从而证明 MC^2 能够实现更加精准的能耗预测.

3.2 实验设置

由第 2 节可知，各部件精准的功耗模型需要对部件各个子系统功耗进行详实的了解，在此基础上进行建模或使用功耗性能评价计数器进行精准观测. 本文使用 4.1 节观测到的天河三号节点功率值作为各部件正常工作时的功率. 基于对天河三号原型机平台处理器频率的实测结果和 SPECpower 基准[64]对广泛的 CPU 平台功耗测量的普遍情况，本文将天河三号处理器正常运行功率值的 25% 视为待机功率来计算. 受限于观测器的粒度，本文将前向阶段和反向阶段都视为 1 个阶段并令处理器和内存处于活动状态. 本文实验面向的是 CPU 超算平台，没有额外加速设备，所以不存在计算节点内存到加速设备内存的转移能耗开销.

另外，在本文实验中忽略了磁盘到内存的写入开销和网络设备通信开销. 一方面，由于保密原因，天河三号原型机的存储节点和网络设备信息不能被公开地获取到且测量困难. 另一方面，相对于计算节点的计算开销和内存读写开销而言，底层存储节点磁盘设备的功耗更低(以 2022 年 12 月 TOP500 第 1 名 Frontiers 使用的 Toshiba AL14SX Series Hard Disk Drive 为例，功耗仅为 5.6 W). 同时，我们通过实验进一步表明，如图 8 所示，在 Cifar-10 数据集训练过程中读写硬盘所带来的时间开销占总训练时间开销不到 8%；同样地，天河三号原型机节点间采用高速双向互联且通信时间占比很少，约占训练时间的 3.6%，按照数据中心级网络设备 Cisco Nexus X9536PQ 计算每个设备平均功率为 10 W. ImageNet 相较 Cifar-10 具有更多的图片数量和更大的图像尺寸，但相较模型的总时间开销而言，硬盘读写时间开销仍不到总开销的 10%，而网络开销也仅为总时间开销的 3.9% 左右. 因此，通过理论分析和实测结果可以证明硬盘读写开销和网络通信开销相较于庞大的计算开销确实可以忽略.

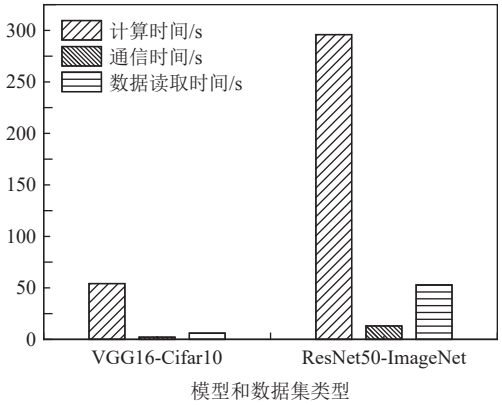


Fig. 8 Time for MTP 128 calculation, communication, and data I/O

图 8 MTP 128 计算、通信和数据读取时间

因此，在天河三号原型机平台使用 32 FT 节点和 128 MT 节点对 VGG 模型大规模分布式数据并行训练过程中各组件功耗情况如表 3 所示.

Table 3 Power Consumption of Each Component

表 3 各组件功耗

组件类型	功率/W
MT-P_{ca}	60
MT-P_{ci}	15
MT-P_{ma}	75
MT-P_{mi}	18.75
FT-P_{ca}	50
FT-P_{ci}	12.5
FT-P_{ma}	46
FT-P_{mi}	11.5

注：P_{ca}和P_{ci}分别表示处理器正常工作和空闲时的开销；P_{ma}和P_{mi}分别表示内存设备正常工作和空闲时的功率.

根据式(5)和 Nishtala 等人[65]、Jahanshahi 等人[66]和 Zhao 等人[67]的相关研究表明处理器频率的变化可以用来表征功率的变化. 因此本文使用 CPU 频率表征功率情况，通过对比各种能耗模型对 CPU 频率变化的预测情况与真实 CPU 频率变化情况来验证本文提出的 MC² 能耗模型的有效性和准确性.

3.3 实验结果

3.3.1 在 Cifar-10 上的实验结果

使用 MT128 节点各阶段的运行时间如表 4 和表 5 所示，使用 FT32 节点各阶段的运行时间如表 6 所示.

因此，实验总能耗可近似表示为

$$\begin{aligned}E_{\text{total}}=N\Big(&P_{\text{ca}}(T_{\text{preprocess}}+T_{\text{f}}+T_{\text{b}}+T_{\text{update}}+T_{\text{communication}})+\\&P_{\text{ci}}(T_{\text{setdataset}}+T_{\text{loaddata}}+T_{\text{inoutmodel}}+T_{\text{add}})+\\&P_{\text{ma}}(T_{\text{inoutmodel}}+T_{\text{preprocess}}+T_{\text{setdataset}}+T_{\text{loaddata}}+T_{\text{f}}+\\&T_{\text{b}}+T_{\text{update}}+T_{\text{communication}})+P_{\text{mi}}(T_{\text{add}})\Big),\end{aligned}\tag{42}$$

其中T_{f}、T_{b}分别表示前向传播和反向传播的时间，P_{ca}和P_{ci}分别表示处理器正常工作和空闲时的开销，P_{ma}和P_{mi}分别表示内存设备正常工作和空闲时的功率. T_{add}表示各节点等待慢节点完成计算以进行梯度平均的平均等待时间. 值得注意的是，式(41)(42)中时间变量 T 的含义有所不同. 在式(41)中，$\int_{T_{\text{operator}}^{\text{begin}}}^{T_{\text{operator}}^{\text{end}}} P_{\text{operator}}^{i}\,\text{d}t$ 指得是节点 i 执行 1 次某个算子操作所产生的能耗. 而在式(42)中，$P_x T_{\text{operator}}$指得是单个节点在整个训练过程中执行某个算子操作的处理器或内存的总能耗.

Table 4 Training Time of Each Stage in VGG16-MT128
表 4 VGG16-MT128 各阶段训练时间

类型	训练时间/s
$T_{\text{setdataset}}$	4.560 3
T_{loaddata}	0.189 1
$T_{\text{preprocess}}$	0.730 2
$T_{\text{inoutmodel}}$	0.563 6
T_{f}	21.291 0
T_{b}	28.678 2
$T_{\text{communication}}$	2.181 5
T_{update}	0.242 5
T_{add}	1.846 8

Table 5 Training Time of Each Stage in ViT-MT128
表 5 ViT-MT128 各阶段训练时间

类型	训练时间/s
$T_{\text{setdataset}}$	13.655 6
T_{loaddata}	1.023 7
$T_{\text{preprocess}}$	1.822 1
$T_{\text{inoutmodel}}$	0.921 3
T_{f}	140.046 5
T_{b}	323.535 9
$T_{\text{communication}}$	21.255 1
T_{update}	16.721 3
T_{add}	20.064 0

Table 6 Training Time of Each Stage in VGG16-FT32
表 6 VGG16-FT32 各阶段训练时间

类型	训练时间/s
$T_{\text{setdataset}}$	3.294 0
T_{loaddata}	0.193 2
$T_{\text{preprocess}}$	0.450 8
$T_{\text{inoutmodel}}$	0.254 6
T_{f}	76.444 7
T_{b}	100.170 2
$T_{\text{communication}}$	6.313 5
T_{update}	0.657 9
T_{add}	4.797 1

由表 3~5 以及式(42)可知, MT128 完成本实验设置下的大规模分布式训练任务的 VGG 总能耗和各阶段能耗如表 7 所示, ViT 各子阶段能耗和总能耗如表 8 所示.

进一步地, 当扩大训练的 epoch 次数时, 只需要对应扩大第 2 部分和第 3 部分时间就可以算出最终的能耗情况, 例如, 当使用 MT128 训练 VGG 网络 100 个 epoch 时的能耗约为 25.39 kW·h, 使用 FT32 训练 100 个 epoch 时的能耗约为 15.77 kW·h.以 MT128 为例, 参考 Cisco Nexus X9536PQ 和 Toshiba AL14SX Series Hard Disk Drive 计算得到 100 个 epoch 的网络能耗和存储能耗分别为 0.096 kW·h 和 0.077 kW·h, 占比均不超过 0.4%, 因此, 这 2 部分能耗确实是可以忽略的.

Table 7 Energy Consumption in VGG MT128 Training
表 7 VGG MT128 训练中的能耗

阶段	能耗/(kW·h)	总能耗/(kW·h)
MT128-数据预处理	0.020 5	
MT128-前向与反向传播	0.239 9	0.274 3
MT128-梯度同步与更新	0.013 9	
FT32-数据预处理	0.002 3	
FT32-前向与反向传播	0.150 7	0.160 0
FT32-梯度同步与更新	0.007 0	

Table 8 Energy Consumption in ViT MT128 Training
表 8 ViT MT128 训练中的能耗

类别	能耗/(kW·h)
$E_{\text{setdataset}}$	0.065 546 674
E_{loaddata}	0.003 275 884
$E_{\text{preprocess}}$	0.008 746 08
$E_{\text{inoutmodel}}$	0.002 948 16
E_{f}	0.672 223 157
E_{b}	1.552 972 412
$E_{\text{communication}}$	0.102 024 48
E_{update}	0.080 256
E_{total}	2.512 0

3.3.2 在 ImageNet 上的实验结果

使用 MT128 节点在各阶段的运行时间如表 9 所示. ImageNet 的总实验能耗仍然使用式(42)的 MC^2 能耗模型, 其基于表 10 所示的各组件能耗开销情况计算得到. 如表 10 所示, 使用 128 MT 节点在 ImageNet 上训练 1 个 epoch 的 ResNet50 模型的能耗为 77.44 kW·h, 类似地, 使用 100 个 epoch 的训练能耗约为 7 122.28 kW·h.

本文在典型小数据集和大数据集上对经典模型和最新模型使用 MC^2 能耗模型进行训练能耗预测的结果表明本文提出的能耗模型可以快速、准确、有效预测各种 DNN 在各种数据集上开展大规模分布式数据并行训练的能耗情况. 本文在 3.4 节进一步对比了 MC^2 和其他能耗模型预测情况与真实能耗情况. 结果表明, MC^2 能耗模型在各种模型和数据集上相较于其他能耗预测模型都能够更逼近真实的能耗情

Table 9 Training Time of Each Stage in MT128 ResNet50

表 9 MT128 ResNet50 各阶段训练时间

阶段	训练时间/min
$T_{\text{setdataset}}$	40.135 2
T_{loaddata}	6.092 0
$T_{\text{preprocess}}$	4.332 2
$T_{\text{inoutmodel}}$	2.266 2
T_{f}	115.512 6
T_{b}	157.619 8
$T_{\text{communication}}$	13.080 0
T_{update}	4.903 4
T_{add}	17.900 0

Table 10 Energy Consumption in ResNet50 Training

表 10 ResNet50 训练能耗

阶段	能耗/(kW·h)
MT128-数据预处理	6.28
MT128-前向与反向传播	64.69
MT128-梯度同步与更新	6.47
MT128-总能耗	77.44

况，具有较强的准确性和普遍适用性.

使用 MC² 能耗模型能够帮助研究者快速、准确地定位被预测模型训练时的能耗瓶颈，指导后续高能效网络模型的优化研究.

3.4 结果分析

如图 9 所示，我们捕获了在天河三号原型机环境下使用 Cifar-10 数据集分布式并行训练 VGG 网络 1 个 epoch 的 CPU 频率变化情况，按照本文的超参数设置，每个 epoch 将会执行 6 个批次，为了更好地展示 CPU 频率变化情况，我们将每个时间步下的频率进行了标准化(在当前频率为主频时，频率为 1). 通过图 9 中 CPU 频率变化(图 9 中实线)可以看出，在数据集初始化阶段 CPU 频率较低，该阶段时长占总训练时长约 8%，随后频率随批次呈周期性变化，每个周期又可分为 2 个阶段：第 1 个阶段主要执行密集计算，CPU 频率维持在主频附近，该部分占每个周期的 96%；第 2 个阶段主要以执行通信和每批次数据读取为主，该部分频率保持在最低频率附近. CPU 频率的变化曲线很好地吻合了 MC² 模型曲线(图 9 中虚线)对实验结果的预想，图中高频、低频出现的位置和持续时间与 MC² 模型预测的处理器高能耗与低能耗出现的位置和持续时间基本一致.

另外，图 9 显示了 CPU 主频在训练过程中可以分为非周期性部分和周期性部分，并且频率的周期

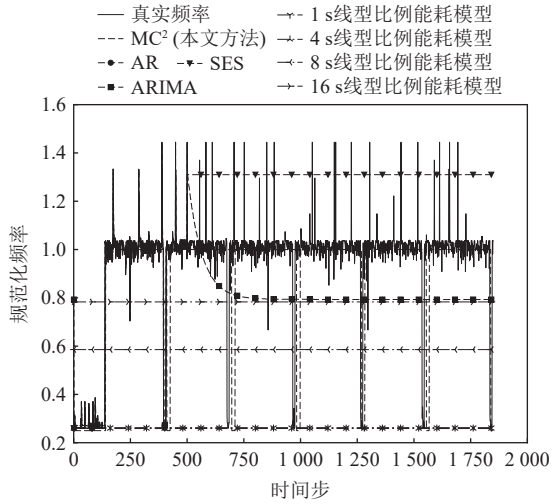


Fig. 9 CPU frequencies change with time during training

图 9 训练过程中 CPU 频率随时间变化

很难在不使用 MC² 进行建模前得到. 为了对比 MC² 能耗模型和与计算时间成线性比例的能耗模型，我们同时绘制了 MC² 能耗模型预测曲线和使用 1s, 4s, 8s, 16s 进行训练并与时间成比例的能耗模型和 AR[60], SES[61], ARIMA[62] 模型预测曲线. 实验结果表明，MC² 的预测准确率达到了 97.16%，相较于其余 7 种模型分别高出了 69.12, 69.50, 34.58, 13.47, 5.23, 22.13, 10.53 个百分点. 另外我们也使用类似的方法评估了 ResNet50 模型的能耗预测结果，预测准确率为 97.53%.

MC² 能耗模型曲线与真实结果更加一致，使用成比例线性放大能耗和时间序列预测模型均不能很好地拟合真实的 CPU 频率变化图像. 因此，使用 MC² 能耗模型将比使用计算时间成线型比例更加准确. 值得注意的是，本文进行了重复多次测试，虽然每次实验都会有个别突发的频率尖峰，但每次尖峰的出现时刻和持续时间都是不同的. 本文认为这种突发的“超频”可能是由于 Linux 系统内核态异常导致的，这种异常对于用户态是不可知的，也不是由于程序所引起，因为即使不运行任何程序，仍然会有某些时刻突发的超频. 本文提出的 MC² 模型是面向实际应用行为和底层硬件的普遍情况进行建模的，很难预测底层这种突发的异常. 由于“超频”时间较短，对总体功耗影响不大，我们会在后续的研究中进一步地观测和确定这种突发的“超频”出现的原因.

由于 MC² 模型对大规模分布式 DNN 任务的每一个阶段所对应的硬件表现情况都进行了细致地分析，只要使用准确的各组件能耗信息就可以得到一个与真实开销十分接近的能耗模型，效果要远胜于仅仅考虑处理器开销，且未考虑到处理器在不同时

期表现的现有静态处理器能耗模型.

除此之外, 在大型分布式计算中心使用功率计对某一个具体的任务进行能耗测量是十分困难的, 而我们的 MC^2 模型只需要知道硬件组件的静态功耗和动态功耗情况就可以得到较为准确的结果. 如果采用公开发行的处理器、内存、磁盘、网卡等设备, 这些功率信息就可以从生产商处容易地获取到.

如第 2 节所述, 目前常用的针对特定任务的能耗度量方法主要有 2 种: 一种是采用功率计进行直接测量; 另一种主要根据 CPU 功率乘以运行时间来估计能耗. 我们提出的 MC^2 能耗模型对于这 2 种方法的对比结果如表 11 所示. 由于上述 2 种方法均将分布式 DNN 训练任务和底层硬件情况视为黑盒, 不了解应用本身特点, 所以必须通过完整地执行整个训练任务才能得到最终的能耗情况. 相较而言, MC^2 根据 DNN 训练任务运行特点和底层硬件能耗表现, 仅计算 1 个 epoch 的能耗即可得到总训练能耗, 成本更低、耗时更少.

Table 11 Comparison of Energy Consumption Model

表 11 能耗模型对比

类别	速度	成本	部署难易程度	准确性
测量方法	慢	高	困难	高
CPU 估算	慢	高	简单	低
MC^2	快	低	中等	较高

4 结论与展望

本文提出了一种面向 DNN 大规模分布式数据并行训练的 MC^2 能耗模型, 该模型细致地将训练任务拆解为数据预处理、前向和反向传播、梯度同步与更新 3 个阶段, 结合各阶段处理器及加速设备、内存、硬盘等设备的硬件运行特征建立各阶段功耗模型, 仅需要各阶段基本的组件功耗, 就可以根据训练轮次要求建模得到整体训练能耗; 并且针对异构平台提出了能耗补偿策略, 比将训练任务看成单一任务仅考虑处理器能耗的模型得到更准确的能耗评估. 本文通过在国产 E 级超算原型机天河三号原型机上使用典型的 VGG16 和 ResNet50 网络模型, 得到了在不同规模下 2 种计算节点上的训练能耗, 证明了本文算法的有效性和扩展性.

本文所提出的 MC^2 模型用于评估在大规模 DNN 分布式训练过程中 3 个阶段的能耗情况以及总能耗情况. 一方面, 科研人员可以利用 MC^2 模型轻松地得到自己对裁剪后的 DNN 模型的能耗情况, 从而评估自己的裁剪算法在能耗优化上的优劣情况; 另一方面, 科研人员也可以利用 MC^2 模型获取到自己的训练能耗瓶颈在 DNN 任务的哪个阶段以及所对应的哪个(或哪几个)设备中, 针对性地使用各种优化方法来减少自己的训练能耗.

未来, 我们将在其他数据中心、计算中心、云服务中心等分布式计算集群上进一步地验证和优化 MC^2 能耗模型. 为了实现上述平台的部署, 我们将对代码进行分布式多机多 GPU 数据并行训练的适配, 并支持 NCCL 和 GLOO 等更多的通信底层框架. 我们会在后续的研究中进一步考虑网络的拓扑、路由、流控及应用通信过程对网络能耗的影响以增强 MC^2 模型的准确性. 我们也将尝试预测和拟合图 8 中出现的“尖峰”点.

另外, 除了节点计算性能差异导致的节点同步等待之外, 还有一类节点同步等待是分布式节点网络导致的, 这一类网络问题出现的原因包括网络拓扑结构异常、偶发的网络异常、网络阻塞等, 针对这一类原因导致的等待产生的能耗开销我们正在探索中, 并会在之后的研究中进一步讨论.

最后, 我们将进一步基于本文提出的能耗模型, 设计和验证新的基于能耗感知的 DNN 剪枝、任务调度优化和模块卸载等节能策略.

作者贡献声明: 魏嘉提出了研究思路、完成主要实验并撰写论文; 张兴军提供思路指导并修改论文; 王龙翔提出了实验设计方法; 赵明强完成了部分实验和文章校对; 董小社优化了论文研究思路.

参 考 文 献

[1] Orhan A E. Robustness properties of Facebook's ResNeXt WSL models[J]. arXiv preprint, arXiv: 1907.07640, 2019

[2] Gao Yongqiang, Guan Haibing, Qi Zhengwei, et al. Quality of service aware power management for virtualized data centers[J]. Journal of Systems Architecture, 2013, 59(4/5): 245−259

[3] Bilal K, Malik S U R, Khan S U, et al. Trends and challenges in cloud datacenters[J]. IEEE Cloud Computing, 2014, 1(1): 10−20

[4] Whitehead B, Andrews D, Shah A, et al. Assessing the environmental impact of data centres part 1: Background, energy use and metrics[J]. Building and Environment, 2014, 82(4): 151−159

[5] Rivoire S M. Designing Energy-Efficient Computer Systems[M]// Models and Metrics for Energy-Efficient Computer Systems. Ann Arbor, MI: ProQuest, 2008: 29−65

[6] vor dem Berge M, Da Costa G, Kopecki A, et al. Modeling and

simulation of data center energy-efficiency in coolemall[C]//Proc of the 1st Int Workshop on Energy Efficient Data Centers. Berlin: Springer, 2012: 25−36

[7] Floratou A, Bertsch F, Patel J M, et al. Towards building wind tunnels for data center design[J]. Proceedings of the VLDB Endowment, 2014, 7(9): 781−784

[8] Dean J, Corrado G S, Monga R, et al. Large scale distributed deep networks[C]//Proc of the 26th Neural Information and Processing System. Cambridge, MA: MIT, 2012: 1223−1231

[9] Raina R, Madhavan A, Ng A Y. Large-scale deep unsupervised learning using graphics processors[C]//Proc of the 26th Int Conf on Machine Learning. New York: ACM, 2009: 873−88

[10] Li Teng, Dou Yong, Jiang Jingfei, et al. Optimized deep belief networks on CUDA GPUs[C]//Proc of the 27th Int Joint Conf on Neural Networks. Piscataway, NJ: IEEE, 2015: 1688−1696

[11] Bottleson J, Kim S Y, Andrews J, et al. clCaffe: OpenCL accelerated Caffe for convolutional neural networks[C]//Proc of the 30th IEEE Int Parallel and Distributed Processing Symp Workshops. Piscataway, NJ: IEEE, 2016: 50−57

[12] Kaler T, Stathas N, Ouyang A, et al. Accelerating training and inference of graph neural networks with fast sampling and pipelining[C]//Proc of the 5th Machine Learning and Systems. New York: ACM, 2022: 172−189

[13] Tan Sijun, Knott B, Tian Yuan, et al. CryptGPU: Fast privacy-preserving machine learning on the GPU[C]//Proc of the 42nd IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2021: 1021−1038

[14] Viebke A, Memeti S, Pllana S, et al. CHAOS: A parallelization scheme for training convolutional neural networks on Intel Xeon Phi[J]. The Journal of Supercomputing, 2019, 75(1): 197−227

[15] Liu Junjie, Wang Haixia, Wang Dongsheng, et al. Parallelizing convolutional neural networks on Intel many integrated core architecture[C]//Proc of the 28th Int Conf on Architecture of Computing Systems. Berlin: Springer, 2015: 71−82

[16] Zlateski A, Lee K, Seung H S. Scalable training of 3D convolutional networks on multi-and many-cores[J]. Journal of Parallel and Distributed Computing, 2017, 106(7): 195−204

[17] Suda N, Chandra V, Dasika G, et al. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks[C]//Proc of the 24th ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2016: 16−25

[18] Zhang Jialiang, Li Jing. Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network[C]//Proc of the 25th ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2017: 25−34

[19] Aydonat U, O'Connell S, Capalija D, et al. An OpenCL™ deep learning accelerator on Arria 10[C]//Proc of the 25th ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2017: 55−64

[20] Zhu Chuanjia, Liu Xin, Fang Jiarui. Research on distributed extension of Caffe based on "Light of Taihu Lake" in Shenwei[J]. Computer Applications and Software, 2020, 37(1): 15−20 (in Chinese)
(朱传家, 刘鑫, 方佳瑞. 基于"神威太湖之光"的 Caffe 分布式扩展研究[J]. 计算机应用与软件, 2020, 37(1): 15−20)

[21] Wei Jia, Zhang Xingjun, Ji Zeyu, et al. Performance evaluation and optimization of distributed parallel deep neural networks on the Tianhe-3 prototype[J]. Computer Engineering and Science, 2021, 43(5): 782−791
(魏嘉, 张兴军, 纪泽宇, 等. 天河三号原型机分布式并行深度神经网络性能评测及调优[J]. 计算机工程与科学, 2021, 43(5): 782−791)

[22] Ji Shihao, Satish N, Li Sheng, et al. Parallelizing Word2Vec in shared and distributed memory[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(9): 2090−2100

[23] Das D, Avancha S, Mudigere D, et al. Distributed deep learning using synchronous stochastic gradient descent[J]. arXiv preprint, arXiv: 1602.06709, 2016

[24] Roy P, Song S L, Krishnamoorthy S, et al. Numa-caffe: Numa-aware deep learning neural networks[J]. ACM Transactions on Architecture and Code Optimization, 2018, 15(2): 1−26

[25] Mittal S, Rajput P, Subramoney S. A survey of deep learning on CPUs: Opportunities and co-optimizations[J]. IEEE Transactions on Neural Networks and Learning Systems, 2021, 33(10): 5095−5115

[26] Awan A A, Hamidouche K, Hashmi J M, et al. S-Caffe: Co-designing MPI runtimes and Caffe for scalable deep learning on modern GPU clusters//Proc of the 22nd ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2017: 193−205

[27] Yin J, Gahlot S, Laanait N, et al. Strategies to deploy and scale deep learning on the summit supercomputer[C]//Proc of the 3rd IEEE/ACM Workshop on Deep Learning on Supercomputers. Piscataway, NJ: IEEE, 2019: 84−94

[28] Duan Qingyang, Wang Zeqin, Xu Yuedong, et al. Mercury: A simple transport layer scheduler to accelerate distributed DNN training[C]//Proc of the 41st IEEE Conf on Computer Communications. Piscataway, NJ: IEEE, 2022: 350−359

[29] Huang Yanping, Cheng Youlong, Bapna A, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism[J]. Advances in Neural Information Processing Systems, 2019, 32(1): 103−112

[30] Narayanan D, Harlap A, Phanishayee A, et al. PipeDream: Generalized pipeline parallelism for DNN training[C]//Proc of the 27th ACM Symp on Operating Systems Principles. New York: ACM, 2019: 1−15

[31] Rajbhandari S, Ruwase O, Rasley J, et al. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning[C]//Proc of the 34th Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2021: 826−840

[32] Dayarathna M, Wen Yonggang, Fan Rui. Data center energy consumption modeling: A survey[J]. IEEE Communications Surveys & Tutorials, 2015, 18(1): 732−794

[33] Ge Rong, Feng Xizhou, Cameron K W. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters[C]//Proc of the 18th ACM/IEEE Conf on Supercomputing. Piscataway, NJ: IEEE, 2005: 34−34

[34] Yeo S, Lee H H S. Peeling the Power Onion of Data Centers[M]// Energy Efficient Thermal Management of Data Centers. Berlin: Springer, 2012: 137−168

[35] Gao Yongqiang, Guan Haibing, Qi Zhengwei, et al. Quality of service aware power management for virtualized data centers[J]. Journal of Systems Architecture, 2013, 59(4): 245−259

[36] Shin D, Kim J, Chang N, et al. Energy-optimal dynamic thermal management for green computing[C]//Proc of the 22nd IEEE/ACM Int Conf on Computer-Aided Design-Digest of Technical Papers. Piscataway, NJ: IEEE, 2009: 652−657

[37] Merkel A, Bellosa F. Balancing power consumption in multiprocessor systems[J]. ACM SIGOPS Operating Systems Review, 2006, 40(4): 403−414

[38] Bertran R, Becerra Y, Carrera D, et al. Accurate energy accounting for shared virtualized environments using pmc-based power modeling techniques[C/OL]//Proc of the 11th IEEE/ACM Int Conf on Grid Computing. Piscataway, NJ: IEEE, 2010[2023-07-20].https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5697889

[39] Li Hui, Casale G, Ellahi T. SLA-driven planning and optimization of enterprise applications[C]//Proc of the 1st Joint WOSP/SIPEW Int Conf on Performance Engineering. New York: ACM, 2010: 117−128

[40] Li Keqin. Optimal configuration of a multicore server processor for managing the power and performance tradeoff[J]. The Journal of Supercomputing, 2012, 61(1): 189−214

[41] Kim S, Roy I, Talwar V. Evaluating integrated graphics processors for data center workloads[C]//Proc of the 46th Workshop on Power-Aware Computing and Systems. New York: ACM, 2013: 41−45

[42] Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit[C]//Proc of the 44th Annual Int Symp on Computer Architecture. New York: ACM, 2017: 1−12

[43] Zhang Boyu, Davoodi A, Hu Yuhen. Exploring energy and accuracy tradeoff in structure simplification of trained deep neural networks[J]. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2018, 8(4): 836−848

[44] Ellison B, Minas L. The problem of power consumption in servers[J]. Energy Efficiency for Information Technology, 2009, 41(2): 1−17

[45] Giridhar B, Cieslak M, Duggal D, et al. Exploring DRAM organizations for energy-efficient and resilient exascale memories[C]// Proc of the 26th Int Conf on High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2013: 277−289

[46] Lin Jiang, Zheng Hongzhong, Zhu Zhichun, et al. Thermal modeling and management of DRAM memory systems[C]//Proc of the 34th Annual Int Symp on Computer Architecture. New York: ACM, 2007: 312−322

[47] Vijaykrishnan N, Kandemir M, Irwin M J, et al. Energy-driven integrated hardware-software optimizations using SimplePower[J]. ACM SIGARCH Computer Architecture News, 2000, 28(2): 95−106

[48] Shiue W T, Chakrabarti C. Memory exploration for low power, embedded systems[C]//Proc of the 36th Design Automation Conf. Piscataway, NJ: IEEE, 1999: 140−145

[49] Roy S, Rudra A, Verma A. An energy complexity model for algorithms[C]//Proc of the 4th Conf on Innovations in Theoretical Computer Science. New York: ACM, 2013: 283−304

[50] Poess M, Othayoth Nambiar R. A power consumption analysis of decision support systems[C]//Proc of the 1st Joint WOSP/SIPEW Int Conf on Performance Engineering. New York: ACM, 2010: 147−152

[51] Feng Boliang, Lu Jiaheng, Zhou Yongluan, et al. Energy efficiency for MapReduce workloads: An in-depth study[C]//Proc of the 33rd Australasian Database Conf. Canberra, Australian: ACS, 2012: 61−70

[52] Yang T, Chen Y, Sze V. Designing energy-efficient convolutional neural networks using energy-aware pruning[C]//Proc of the 30th IEEE Conf on Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2017: 5687−5695

[53] Wang Ruibo, Lu Kai, Chen Juan, et al. Brief introduction of TianHe exascale prototype system[J]. Tsinghua Science and Technology, 2020, 26(3): 361−369

[54] Wei Jia, Zhang Xingjun, Ji Zeyu, et al. Deploying and scaling distributed parallel deep neural networks on the Tianhe-3 prototype system[J]. Scientific Reports, 2021, 11(1): 1−14

[55] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint, arXiv:1409.1556, 2014

[56] He Kaiming, Zhang Xiangyu, Ren Shaoqing, et al. Deep residual learning for image recognition[C]//Proc of the 29th IEEE Conf on Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2016: 770−778

[57] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An image is worth 16×16 words: Transformers for image recognition at scale[J]. arXiv preprint, arXiv: 2010.11929, 2021

[58] Marcel S, Rodriguez Y. Torchvision the machine-vision package of torch[C]//Proc of the 18th ACM Int Conf on Multimedia. New York: ACM, 2010: 1485−1488

[59] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Advances in Neural Information Processing Systems, 2012, 25(1): 1097−1105

[60] Akaike H. Autoregressive model fitting for control[M]//Selected Papers of Hirotugu Akaike. Berlin: Springer, 1998: 153−170

[61] Gardner Jr E S. Exponential smoothing: The state of the art[J]. Journal of Forecasting, 1985, 4(1): 1−28

[62] Box G E P, Pierce D A. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models[J]. Journal of the American statistical Association, 1970, 332(65): 1509−1526

[63] Hanusz Z, Tarasinska J, Zielinski W. Shapiro-Wilk test with known mean[J]. REVSTAT-Statistical Journal, 2016, 14(1): 89−100

[64] Henning J L. SPEC CPU2000: Measuring CPU performance in the new millennium[J]. Computer, 2000, 33(7): 28−35

[65] Nishtala R, Petrucci V, Carpenter P, et al. Twig: Multi-agent task management for colocated latency-critical cloud services[C]//Proc of the 26th IEEE Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2020: 167−179

[66] Jahanshahi A, Yu Nanpeng, Wong D. PowerMorph: QoS-aware server power reshaping for data center gegulation service[J]. ACM

Transactions on Architecture and Code Optimization, 2022, 19(3): 1−27

[67] Zhao Laiping, Yang Yanan, Zhang Kaixuan, et al. Rhythm: Component-distinguishable workload deployment in datacenters[C]// Proc of the 15th European Conf on Computer Systems. New York: ACM, 2020: 153−170



Wei Jia, born in 1997. PhD candidate. His research interests include computer architecture, high performance computing, and deep learning.

魏　嘉, 1997 年生. 博士研究生. 主要研究方向为计算机体系结构、高性能计算、深度学习.



Zhang Xingjun, born in 1969. PhD, professor, PhD, supervisor. Senior member of CCF. His main research interests include computer architect-ure, high performance computing, big data storage system, and computer networks.

张兴军, 1969 年生. 博士, 教授, 博士生导师, CCF 高级会员. 主要研究方向为计算机体系结构、高性能计算、大数据存储系统、计算机网络.



Wang Longxiang, born in 1988. PhD. Member of CCF. His main research interests include data deduplication, big data storage system, and artificial intelligence.

王龙翔, 1988 年生. 博士. CCF 会员. 主要研究方向为数据去重、大数据存储系统、人工智能.



Zhao Mingqiang, born in 1998. Master candidate. His main research interests include job scheduling, high performance computing, and reinforcement learning.

赵明强, 1998 年生. 硕士研究生. 主要研究方向为作业调度、高性能计算、强化学习.



Dong Xiaoshe, born in 1963. PhD, professor, PhD supervisor. Member of CCF. His main research interests include high performance computer system, storage system, and cloud computing.

董小社, 1963 年生. 博士, 教授, 博士生导师, CCF 会员. 主要研究方向为高性能计算系统、存储系统、云计算.

附录 A. 各阶段节点训练时间 W 检验结果.

为了增强实验的的可靠性和有效性，本文对 10 次实验的结果进行了夏皮洛-威尔克检验，结果如表 A1 所示，所有时间的 p 值都大于 0.05，即原假设不能被拒绝，训练各阶段的时间均服从正态分布，因此表 4、表 5、表 6、表 9 中的结果都用 10 次训练的均值来表示.

Table A1 Result of Shapiro-Wilk Test

表 A1 夏皮洛-威尔克检验结果

阶段	FT-p 值	MT-p 值
$T_{\text{setdataset}}$	0.750	0.647
T_{loaddata}	0.124	0.229
$T_{\text{preprocess}}$	0.065	0.330
$T_{\text{inoutmodel}}$	0.113	0.120
T_{f}	0.255	0.061
T_{b}	0.222	0.382
$T_{\text{communication}}$	0.650	0.579
T_{update}	0.772	0.867
T_{add}	0.237	0.425