计算机科学
COMPUTER SCIENCE

面向通用一致性优化的通信高效的异步 ADMM 算法

王冬霞，雷咏梅，张泽宇

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211200006

面向通用一致性优化的通信高效的异步 ADMM 算法

王冬霞　雷咏梅　张泽宇
上海大学计算机工程与科学学院　上海 200444
(wangdongxia1983@126.com)

摘　要　分布式交替方向乘子法(Alternating Direction Method of Multipliers,ADMM)是求解大规模机器学习问题使用最广泛的方法之一。现有大多数分布式 ADMM 算法都基于完整的模型更新。随着系统规模及数据量的不断增长,节点间的通信开销逐渐成为限制分布式 ADMM 算法发展的瓶颈。为了减少节点间通信开销,提出了一种通信高效的通用一致性异步分布式 ADMM 算法(General Form Consensus Asynchronous Distributed ADMM,GFC-ADADMM),该算法通过分析高维稀疏数据集的特性,节点间利用关联模型参数代替完整模型参数进行通信,并对模型参数进行过滤以进一步减少节点间传输负载。同时结合过时同步并行(Stale Synchronous Parallel,SSP)计算模型、allreude 通信模型及混合编程模型的优势,利用异步 allreduce 框架并基于 MPI/OpenMP 混合编程模型实现 GFC-ADADMM 算法,提高算法计算与通信效率。文中利用 GFC-ADADMM 算法求解稀疏 logistic 回归问题,实验测试表明,与现有分布式 ADMM 算法相比,GFC-ADADMM 算法可减少 15%63%的总运行时间,且算法收敛时可达到更高的准确率。

关键词:分布式交替方向乘子法;通用一致性优化;稀疏 allreduce;混合编程模型;Logistic 回归
中图法分类号　TP391

Communication Efficient Asynchronous ADMM for General Form Consensus Optimization

WANG Dong-xia,LEI Yong-mei and ZHANG Ze-yu
School of Computer Engineering and Science,Shanghai University,Shanghai 200444,China

Abstract　The distributed alternating direction method of multipliers(ADMM) is one of the most widely used methods for solving large-scale machine learning applications. However,most distributed ADMM algorithms are based on full model updates. With the increasing of system scale and data volume,the communication cost has become the bottleneck for the distributed ADMM when big data are involved. In order to reduce the communication cost in a distributed environment,a general form consensus asynchronous distributed alternating direction method of multipliers(GFC-ADADMM) is proposed in this paper. First,in the GFC-ADADMM,the associated model parameters rather than full model parameters are transmitted among nodes to reduce the transmission load,and the associated model parameters are filtered according to the characteristics of high-dimensional sparse data sets to further reduce the transmission load. Second,the GFC-ADMM is implemented by an asynchronous allreduce framework,which combines the advantage of the asynchronous communication protocol and the allreduce communication mode. Third,combining the advantages of the stale synchronous parallel(SSP) computing model,allreduce communication model,and hybrid programming model,the asynchronous allreduce framework and MPI/OpenMP hybrid programming model are adopted to implement the GFC-ADADMM,which improves calculation efficiency and communication efficiency of the algorithm. Finally,the sparse logistic regression problem is solved by the GFC-ADADMM. Evaluation with large-scale datasets shows that compared with state-of-the-art asynchronous distributed ADMM algorithms,the GFC-ADADMM can reduce the total running time by 15%-63%,and has higher accuracy in convergence.

Keywords　Distributed alternating direction method of multipliers,General form consensus optimization,Sparse allreduce,Hybrid programming model,Logistic regression

1　引言

机器学习是人们从原始数据中提取有效信息的首要手段,随着数据规模的增长,单节点很难在有效时间内处理大规模的机器学习问题。因此,利用分布式系统来求解大规模机器学习问题是目前的主要趋势。交替方向乘子法(ADMM)是一种求解优化问题的有效算法,该方法可将大规模问题分解为若干个子问题求解,并通过协调子问题的解得到全局解。该

到稿日期:2021-12-01　返修日期:2022-05-09
基金项目:国家自然科学基金(U1811461)
This work was supported by the National Natural Science Foundation of China(U1811461).
通信作者:雷咏梅(lei@shu.edu.cn)

方法不仅适用于求解带约束项的优化问题，且可以有效地扩展到大型问题，因此被广泛应用于求解大规模回归、分类等监督式机器学习问题[1-3]。虽然相比随机梯度下降法，分布式ADMM算法具有更高的计算通信比。但随着数据集规模的增长，通信开销仍然是限制其性能的主要因素。

目前，已有研究从不同层面来减少分布式ADMM算法的通信开销。如采用通信审查机制和量化等方式减少节点间传输负载[4-6]，基于SSP计算模型实现分布式ADMM算法减少系统同步开销[7-8]。上述算法均是基于全局一致性或分散一致性优化问题，算法模型中每个节点均处理完整的模型参数。而在一些特殊应用领域，每个数据块所关联的模型参数往往较少。如处理文本分类问题时，一些单词只在部分文本中出现，因此每个处理器只能处理本地语料库中出现的单词。在这类应用场景中，每个节点只需要处理与之关联的模型参数，而不需要处理所有的模型参数。Boyd等[9]针对这类问题提出了通用一致性分布式ADMM算法。Zhu等[10]针对通用一致性优化问题，提出了一种分块异步分布式ADMM算法，该算法节点间只需要传输关联模型参数，因此可以有效减少节点间通信负载。但该算法基于参数服务器框架实现，节点间负载不均衡，在对称多处理机集群环境下会造成计算资源的浪费。本文基于allreduce对等模式实现异步分布式ADMM算法求解通用一致性优化问题；同时，本文利用数据集样本关联特征的频率确定关联模型参数，以进一步减少节点间通信负载。

MPI并行编程模型和MPI/OpenMP混合并行编程模型是高性能领域使用最多的两种编程模型。与MPI并行编程模型相比，针对层次存储结构设计的MPI/OpenMP混合并行编程模型可以更好地均衡节点间负载，节省内存和通信开销[11-12]。本文基于MPI/OpenMP混合并行编程模型实现分布式ADMM算法，主要工作总结如下：

(1)提出了一种通用一致性异步分布式ADMM算法(GFC-ADADMM)，该算法根据数据集特性确定每个节点关联模型参数，并通过特征维度的出现频率对关联模型参数进行选择，以减少节点间通信负载。

(2)基于allreduce通信模型实现GFC-ADADMM算法，一方面，节点间采用SSP计算模型减少节点间同步时间开销；另一方面，基于对等模式实现算法，可更好地均衡节点间负载，充分利用对称多处理机集群的优势来提高算法性能。同时结合多核分布式集群层次存储结构，基于MPI/OpenMP混合编程模型实现该算法，以提高算法的可扩展性。

(3)利用GFC-ADADMM算法求解稀疏logistic回归问题。实验测试表明，与现有的基于全局一致性的异步分布式ADMM算法相比，GFC-ADADMM算法可有效减少运行时间，提高算法的收敛速度。

本文第2节介绍通用一致性优化问题及求解该类问题的ADMM算法框架，并进一步提出基于通用一致性的异步分布式ADMM算法；第3节介绍GFC-ADADMM算法的具体实现；第4节利用GFC-ADADMM算法求解稀疏logistic回归问题，并对其收敛性、可扩展性等性能进行分析；最后总结全文并展望未来。

2 通用一致性优化与ADMM算法

大多数分布式机器学习任务可抽象为式(1)所示的优化问题。

$$\min \frac{1}{n}\sum_{i=1}^{n} l_i(\boldsymbol{A}_i^{\mathrm{T}}x-b_i)+r(x) \tag{1}$$

其中，n表示样本数，$\boldsymbol{A}_i$表示第i个样本，b_i为第i个样本的标签值，l_i为第i个样本的损失函数，$x\in R^p$为模型参数，p表示数据特征维度，$r(x)$为正则项。增加正则项的主要目的是避免模型过拟合。最常见的正则项有L2范数正则项和L1范数正则项。带L1范数正则项模型通常具有稀疏模型参数的功能，如Lasso、稀疏logistic回归等。本文主要关注稀疏logistic回归模型，该模型在机器学习[13]、医学[14-15]等领域都得到了广泛应用。稀疏logistic回归模型可抽象为式(2)所示的优化问题。

$$\begin{aligned}&\min \frac{1}{n}\sum_{i=1}^{N}\sum_{j=1}^{n_i}\log(1+\exp(-b_{ij}\boldsymbol{A}_{ij}^{\mathrm{T}}x_i))+\lambda\|z\|_1\\&\text{s.t. } x_i=z,i=1,\cdots,N\end{aligned} \tag{2}$$

求解式(2)所示问题常用的算法有近似梯度法及其变种、ADMM算法等。ADMM算法是一种基于对偶空间的梯度下降方法，近年来受到机器学习领域的普遍关注[16-18]。为了应用ADMM求解正则化问题，通常将问题(2)转化为式(3)所示的全局一致性问题进行求解。

$$\begin{aligned}&\min \frac{1}{n}\sum_{i=1}^{N}\sum_{j=1}^{n_i}\log(1+\exp(-b_{ij}\boldsymbol{A}_{ij}^{\mathrm{T}}x_i))+\lambda\|z\|_1\\&\text{s.t. } x_i=z,i=1,\cdots,N\end{aligned} \tag{3}$$

其中，$x_i\in R^p$表示局部变量，$z\in R^p$表示全局变量，λ为正则项系数。利用ADMM算法求解式(3)所示全局一致性问题的迭代公式如下：

$$\begin{aligned}x_i^{k+1}:=\arg\min_{x_i}(&\sum_{j=1}^{n_i}\log(1+\exp(-b_{ij}\boldsymbol{A}_{ij}^{\mathrm{T}}x_i))+\\&(\rho/2)\|x_i+y_i^k/\rho-z^k\|_2^2\end{aligned} \tag{4}$$

$$z^{k+1}:=ST_{\lambda/\rho N}(\sum_{i=1}^{N}\|x_i^{k+1}+y_i^k/\rho-z^k\|_2^2) \tag{5}$$

$$y_i^{k+1}:=y_i^k+\rho(x_i^{k+1}-z^{k+1}) \tag{6}$$

其中，$y_i\in R^p$表示x_i的对偶变量，ST表示软阈值算子。全局一致性分布式ADMM算法并行实现流程主要包括以下几个关键步骤：首先利用多个工作节点并行更新局部变量，然后聚合所有局部变量和对偶求解全局变量，最后各个工作节点再并行更新对偶变量。利用ADMM算法求解全局一致性优化问题时，每个节点处理的局部参数关联数据集的所有特性，即每个节点的局部变量和对偶变量均包含完整的模型参数。当数据特征维度较高时，节点间通信量会成为算法的性能瓶颈。而在许多机器学习应用中(如文本分类、疾病诊断)，每块数据集往往只关联部分特性。特别是当处理的数据集为高维稀疏数据集时，每个数据块关联的模型参数较少。基于这一特征，文献[9]中将式(2)所示问题转换为通用一致性问题来进行求解，如式(7)所示：

$$\begin{aligned}&\min \frac{1}{n}\sum_{i=1}^{N}\sum_{j=1}^{n_i}\log(1+\exp(-b_{ij}\boldsymbol{A}_{ij}^{\mathrm{T}}x_i))+\lambda\|z\|_1\\&\text{s.t. } x_i=z_{g_i},i=1,\cdots,N\end{aligned} \tag{7}$$

其中，$x_i\in R^{p_i}$，$z\in R^p$，$z_{g_i}\in R^{p_i}$代表与局部变量x_i关联的

全局变量,且 z_{g_i} 是关于全局变量 z 的线性函数。由式(7)可知,每个局部变量只需与关联特性保持一致。利用 ADMM 算法求解通用一致性问题,其迭代公式可表示为:

$$x_i^{k+1}:=\arg\min_{x_i}\Big(\sum_{j=1}^{n_i}\log(1+\exp(-b_{ij}\boldsymbol{A}_{ij}^{\mathrm{T}}x_i))+(\rho/2)\parallel x_i+y_i^k/\rho-z_{g_i}^k\parallel_2^2\Big) \tag{8}$$

$$z_j^{k+1}:=\arg\min\Big(\lambda\parallel z_j\parallel_1+\sum_{i\in\Theta_j}(\rho/2)\parallel x_{ij}^{k+1}+y_{ij}^{k+1}/\rho-z_j\parallel_2^2\Big),j=1,\cdots,p \tag{9}$$

$$z_{g_i}^{k+1}:=\{z_j\mid\forall j\in\Gamma_i\} \tag{10}$$

$$y_i^{k+1}:=y_i^k+\rho(x_i^{k+1}-z_{g_i}^{k+1}) \tag{11}$$

其中,z_j 表示全局变量的第 j 维模型参数,Θ_j 表示与第 j 维模型参数关联的所有节点的集合,Γ_i 表示节点 i 关联的所有模型参数的集合。

由式(9)可知,求解全局量时需要聚合与之关联的所有节点的模型参数,而由式(8)可知,局部变量的求解仍然是一个优化问题。在大规模分布式系统中,由于数据分布的不均衡性,每次迭代时每个计算节点求解局部变量的时间会存在较大差异,这种差异性会大大增加节点间同步时间。为了减少节点间同步时间,本文提出通用一致性异步分布式 ADMM(GFC-ADADMM)算法,该算法的迭代公式如下:

$$x_i^{k+1}:=\arg\min_{x_i}\Big(\Big(\sum_{j=1}^{n_i}\log(1+\exp(-b_{ij}\boldsymbol{A}_{ij}^{\mathrm{T}}x_i))+(\rho/2)\parallel x_i+y_i^k/\rho-z_{g_i}\parallel_2^2\Big) \tag{12}$$

$$z_j^{k+1}:=\arg\min\Big(\lambda\parallel z_j\parallel_1+\sum_{i\in\Theta_j}(\rho/2)\parallel x_{ij}^{k+1}+y_{ij}^{k+1}/\rho-z_j\parallel_2^2\Big),j=1,\cdots,p \tag{13}$$

$$z_{g_i}:=\{z_j\mid\forall j\in\Gamma_i\} \tag{14}$$

$$y_i^{k+1}:=y_i^k+\rho(x_i^{k+1}-z_{g_i}) \tag{15}$$

其中,z_{g_i} 表示与第 i 个节点关联的最新的全局参数,x_{ij} 与 y_{ij} 表示第 j 维参数关联的第 i 个节点的最新的局部变量和对偶变量。

3 通用一致性异步分布式 ADMM 算法的实现

在全连通的网络拓扑结构中,基于参数服务器结构的主从模式和基于 allreduce 结构的对等模式是实现分布式 ADMM 算法的两种主要拓扑结构。基于参数服务器架构的主从模式提供了灵活的同步机制,可以很容易实现异步算法。但参数服务器结构需要利用专门的中心节点来聚合所有工作节点的局部参数,这在一定程度上会浪费计算资源,同时基于主从模式实现 ADMM 算法,中心节点需要存储所有节点的局部变量的值,内存开销大。而基于 allreduce 结构的对等模式,各节点间通信负载均衡,且各计算节点分散存储变量,节省了内存开销,但 allreduce 通信模式通常要求计算节点间同步。为了更好地均衡节点间负载,提高节点间通信效率,本文将基于 allreduce 对等模式实现 GFC-ADADMM 算法。

3.1 工作节点与模型参数的映射

由式(12)和式(15)可知,求解局部变量 x_i 和对偶变量 y_i 时,需要确定工作节点关联的模型参数 Γ_i。本文利用每个节点处理的数据集来确定其关联的模型参数,假设第 i 个工作节点读取第 i 块数据集 D_i。每个工作节点在读取数据集的同时统计各模型参数出现的频率,并将统计信息保存到向量 $\boldsymbol{V}_i$ 中。$\boldsymbol{V}_{ij}$ 表示 D_i 中第 j 维特性出现的频率。若 V_{ij} 小于阈值 ε,则表示 D_i 与第 j 维模型参数不相关。Γ_i 的表达式如式(16)所示:

$$\Gamma_i=\{\forall j\in\{1,\cdots,p\}\mid\boldsymbol{V}_{ij}>\varepsilon\} \tag{16}$$

由式(13)可知,在求解全局变量 z 时,需要确定每个模型参数关联的节点 Θ_j。实际上,求解全局变量只需要聚合每个工作节点关联参数,因此计算 z 时只需要确认与每一维模型参数关联的节点数,而不需要确定具体与哪些节点关联。本文将每一维模型参数所关联的节点数称为该维模型参数的权重。每个工作节点首先根据 Γ_i 确定其权重,若该节点与某一维模型参数相关,则将权重设置为 1,否则设置为 0。节点 i 的权重 η_i 的更新规则如式(17)所示。全局模型参数的权重 W 则通过聚合所有工作节点的权重得到,其更新公式如式(18)所示:

$$\eta_{ij}=\begin{cases}1, & \text{if } V_{ij}>\varepsilon\\ 0, & \text{if } V_{ij}\leqslant\varepsilon\end{cases} \tag{17}$$

$$W_j=\sum_{i=1}^{N}\eta_{ij},j=1,\cdots,p \tag{18}$$

其中,η_{ij} 表示第 i 个节点中第 j 维模型参数的权重,W_j 表示全局变量中第 j 维模型参数的权重。

3.2 GFC-ADADMM 算法各变量更新规则

由于局部变量与对偶变量只与关联模型参数相关,因此在更新 x_i,y_i 时也只需要更新其关联参数,x_i,y_i 的更新公式如式(12)和式(15)所示。在求解全局变量时可将 $x_{ij}+y_{ij}/\rho$ 看作一个整体,本文利用中间变量 ω_{ij} 代替 $x_{ij}+y_{ij}/\rho$ 更新全局变量。ω_{ij} 的更新表达式为:

$$\omega_{ij}^{k+1}:=\begin{cases}x_{ij}^{k+1}+y_{ij}^{k+1}/\rho, & \text{if } j\in\Gamma_i\\ 0, & \text{otherwise}\end{cases} \tag{19}$$

更新全局变量时只要求部分节点局部参数更新完成,对于未完成更新的工作节点,利用其前一次的值进行计算。全局变量更新表达式为:

$$\nu_{ij}^{k+1}:=\begin{cases}\omega_{ij}, & \forall i\in\Omega_k\\ \omega_{ij}^k, & \forall i\in\Omega_k^c\end{cases} \tag{20}$$

$$z_j^{k+1}:=ST_{\gamma/\rho W_j}\left(\frac{\sum_{i\in\Theta_j}\nu_{ij}^{k+1}}{W_j}\right)=ST_{\gamma/\rho W_j}\left(\frac{\sum_{i=1}^{N}\nu_{ij}^{k+1}}{W_j}\right) \tag{21}$$

其中,Ω_k 表示已完成计算的工作节点集合,Ω_k^c 为 Ω_k 的补集。由于工作节点间采用异步计算模型,因此每次更新全局变量时利用部分“旧值”进行计算。为了保证算法的收敛性,在计算时设置两个异步条件。1)部分障碍,即要求每次更新全局变量时至少有 S 个工作节点的参数是最新的值,也就是说,每次计算全局变量时 Ω_k 中的元素个数不少于 S。本文将 S 称为阈值,N 表示工作进程数,则有 $0<S\leqslant N$,当 $S=N$ 时,则算法等价于同步算法。2)有界延时,即要求所有工作节点间更新周期的差值不大于最大延时周期 τ。

由于变量 ω_{ij} 非关联参数为 0,因此对关联参数执行累加操作等价于对所有参数执行累加操作。求解全局变量时可

利用 allreduce 算法聚合参数。但与基于全局一致性 ADMM 算法不同的是，各节点局部参数中有许多“0”元素，因此在通信时可直接传输稀疏向量。同时，本文利用文献[19]中提出的 Calculator-Communicator-Manager(CCM)框架实现基于 allreduce 通信模型的 GFC-ADADMM，并采用 HSA 算法聚合参数。与文献[19]提出的 HSAC-ALADMM 算法类似，本文基于稀疏 allreduce 通信模型实现 GFC-ADADMM 算法，由于求解通用一致性优化问题时，每个节点关联的模型参数是固定不变的，因此无须每次通信时对参数进行选择，可直接利用集合 Γ_i 确定传输参数的索引。

3.3 GFC-ADADMM 算法流程

本文基于 CCM 框架及 MPI/OpenMP 混合编程模型实现 GFC-ADADMM 算法，系统中包含 Calculator，Communicator 和 Manager 3 种角色。Calculator 负责计算，Communicator 负责进程间通信，Manager 负责协调 Communicator 间的通信过程。每个计算节点只包含一个 Calculator，每个 Calculator 可创建多个线程并行求解子问题。线程间基于 OpenMP 实现并行化。

GFC-ADADMM 算法流程主要包括以下 8 步：

(1)各个 Calculator 读取数据集，并更新关联模型参数集合，更新完成后向 Communicator 发送通知；

(2)Communicator 接收到 Calculator 的更新完关联参数的通知后聚合所有节点的关联模型参数，并更新各模型参数的权重，然后唤醒 Calculator；

(3)Calculator 利用并行信赖域牛顿法[20]求解局部变量，更新完局部变量后向 Communicator 发送计算完成通知；

(4)Communicator 接收到计算完成通知后，向 Manager 请求执行 allreduce 操作；

(5)Manager 接收到 Communicator 的请求消息后，判断是否满足异步通信条件，若满足条件则通知所有 Communicator 执行 allreduce 操作，否则继续等待接收消息；

(6)Communicator 接收到 Manager 的执行 allreduce 操作的通知后，检查当前进程是否完成计算，若已完成，则利用最新的局部变量和对偶变量更新 ω_{ij}，否则利用局部变量及对偶变量前一次的值更新 ω_{ij}；

(7)所有 Communicator 利用 HSA 算法[15]聚合参数 ω_{ij}，执行完成后向 Manager 发送已完成 allreduce 操作的通知，并唤醒 Calculator；

(8)Calculator 利用所有聚合后的参数更新全局变量 z，并判断算法是否满足停止条件，若满足，则算法停止，否则更新对偶变量，再循环执行步骤(3)－步骤(8)。

GFC-ADADMM 算法的详细描述如算法 1－算法 3 所示。

算法 1 GFC-ADADMM：Manager

1. 初始化：$k=0, 0<S\leqslant N, \tau, t_i=0(i=0,\cdots,N-1)$
2. Repeat
3. 等待接收到 S 个 Communicator 的执行 allreduce 操作的请求且 $t_i<\tau(i=0,\cdots,N-1)$；
4. 通知所有 Communicator 执行 allreduce 操作；
5. 等待接收所有 Communicator 执行完成 allreduce 操作的报告；
6. 将集合 Ω_k 中所有 Communicator 对应的计数器 t_i 清零，集合 Ω_k^c 中所有 Communicator 对应的计数器 t_i 增加 1；
7. $k\leftarrow k+1$；
8. 若接收到任意 Communicator 的停止程序请求，则向所有 Communicator 发送停止程序命令，然后等待接收 Communicator 的已停止程序报告；
9. Until 接收到所有 Communicator 的已停止程序报告。

算法 2 GFC-ADADMM：Communicator

1. 初始化：$k_i=0, \omega_i^0, x_{old}, y_{old}$；
2. 等待从 Calculator 接收 ω_i 更新完成通知；
3. 调用 MPI_Allreduce 接口规约 ω_i，更新 W，唤醒 Calculator；
4. Repeat
5. if 从 Calculator 接收到计算完成通知
6. 向 Manager 发送执行 allreduce 操作请求
7. end if
8. if 从 Manager 接收到执行 allreduce 操作命令
9. if 接收到 Calculator 计算完成通知
10. $\omega_i^{k_i+1}=x_i^{k_i+1}+y_i^{k_i+1}/\rho$；
11. $x_{old}=x_i^{k_i+1}$；
12. $y_{old}=y_i^{k_i+1}$；
13. else $\omega_i^{k_i+1}=x_{old}+y_{old}/\rho$
14. end if
15. 利用 HSA 算法聚合 $\omega_i^{k_i+1}$；
16. $k_i\leftarrow k_i+1$；
17. 唤醒 Calculator；
18. 向 Manager 发送 allreduce 执行完成的报告；
19. end if
20. if 从 Calculator 接收到停止程序通知
21. 向 Manager 发送停止程序请求；
22. 等待接收 Manager 停止程序命令；
23. 向 Manager 发送已停止程序报告；
24. 向 Calculator 发送停止程序消息并将停止标志设置为真；
25. end if
26. Until 停止标志为真。

算法 3 GFC-ADADMM：Calculator

1. 初始化：$k_i=0, x_i^0, y_i^0, z^0$；
2. 读取数据集；
3. 更新 Γ_i；
4. 更新 η_i；
5. 向 Communicator 发送 η_i 更新完成通知，并等待 Communicator 唤醒；
6. Repeat
7. 并行更新 $x_i^{k_i+1}$；
8. 向 Communicator 发送计算完成通知；
9. 等待 Communicator 唤醒；
10. 更新 z^{k_i+1}；
11. $k_i\leftarrow k_i+1$；
12. if 程序满足停止条件
13. 向 Communicator 发送停止程序通知；
14. 等待接收停止程序消息；
15. else 更新 $y_i^{k_i+1}$；
16. end if
17. 读取停止标志；
18. Until 停止标志为真。

4 实验结果与分析

本节利用本文提出的 GFC-ADADMM 算法来求解式(2)所示的稀疏 logistic 回归问题。实验平台为上海大学高性能集群“自强 4000”，该集群每个节点拥有两颗 Intel E5-2690 CPU，共计 16 个核。实验共使用 5 个计算节点，节点间由千兆以太网连接。操作系统为 Ubuntu Linux2.6.32，gcc 版本为 4.8.4，MPI 使用的是 MPICH3。实验数据集为公共数据集 KDDB 和 URL。数据集的相关信息如表 1 所列。

表 1 测试数据集信息

Table 1 Summary of datasets

DataSet	Number of features	Number of training samples	Number of testing samples
KDDB	1 163 024	19 264 097	748 401
URL	3 231 961	2 000 000	396 130

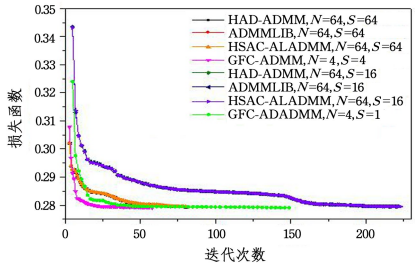
本文首先对算法的收敛性进行测试，然后从算法运行时间和准确率等方面对其性能进行测试，最后测试每个节点使用不同处理器数时算法的性能，分析算法的纵向可扩展性。并将 GFC-ADADMM 与 HAD-ADMM，ADMMLIB，HSAC-ALADMM 这 3 种算法进行对比。HAD-ADMM 为 Wang 等[17]基于层次化参数服务器结构实现的异步分布式 ADMM 算法；ADMMLIB 为 Xie 等[18]基于 allreduce 通信模型实现的异步分布式 ADMM 算法；HSAC-ALADMM 算法为 Wang 等[19]基于 CCM 计算框架实现的异步分布式 ADMM 算法。HAD-ADMM，ADMMLIB 及 HSAC-ALADMM 这 3 种算法均是基于 MPI 并行模式实现，其子问题采用信赖域牛顿法[21]求解。本文基于 MPI/OpenMP 混合并行模式实现 GFC-ADADMM 算法，并利用并行信赖域牛顿法[20]求解子问题。惩罚项参数 ρ 均设置为 1.0，正则项参数 λ 设置为 0.5。GFC-ADADMM 算法中 ε 设置为 2。算法的停止条件为原始残差 r 和对偶残差 s 满足以下条件：

$$\| r^k \|_2^2 = \sum_{i=1}^{N} \| x_i^k - z^k \|_2^2 \leqslant 10^{-2}\sqrt{Np} + 10^{-3}\max\{\sum_{i=1}^{N} \| x_i^k \|_2^2, N \| z \|_2^2\} \tag{22}$$

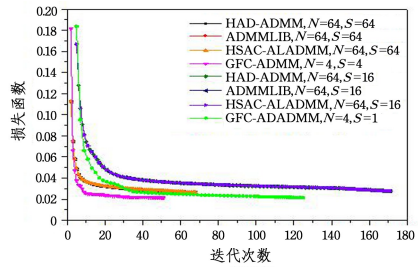
$$\| s^k \|_2^2 = N\rho^2 \| z^k - z^{k-1} \|_2^2 \leqslant 10^{-4}\sqrt{Np} + 10^{-3}\sum_{i=1}^{N} \| x_i^k \|_2^2 \tag{23}$$

4.1 收敛性测试

本节对算法收敛性进行测试。GCADMM 与 HSAC-ALADMM 算法使用 4 个计算节点作为工作节点，每个计算节点使用 16 个工作进程，即 $N=64$。GFC-ADADMM 算法也使用 4 个计算节点作为工作节点，每个计算节点使用 1 个工作进程，即 $N=4$，每个进程创建 16 个线程。也就是说 3 种算法工作节点使用的处理器数均为 64。同时，GFC-ADADMM 与 HSAC-ALADMM 算法中利用独立的一个节点作为 Manager 进程。图 1 给出了 4 种算法的收敛曲线。由图 1 可以看出，1)在对等情况下(阈值与工作进程数相同时)，GFC-ADADMM 算法比其他几种算法收敛时的迭代次数更少，且算法收敛时损失函数的值更小。这是因为分布式 ADMM 算法收敛时迭代次数受工作进程数的影响，采用 MPI/OpenMP 混合编程模型实现分布式 ADMM 算法，在保证单节点计算能力不变的情况下最小化了工作进程数。2)随着阈值 S 的减小，算法收敛时迭代次数增加，这是因为阈值减小，每次更新全局变量时使用新的局部参数个数减少，因此算法需要更多迭代次数才能达到收敛。



(a)KDDB



(b)URL

图 1 不同算法的收敛曲线

Fig. 1 Convergence curves of different algorithms

4.2 运行时间和准确率测试

本节实验相关设置与 4.1 节一致。表 2、表 3 分别列出了数据集为 KDDB，URL 时几种算法使用不同阈值时的运行时间与准确率。运行时间包括计算时间和等待时间，计算时间为各个变量更新时间，以所有 Worker 计算时间的平均值作为系统计算时间，等待时间包括数据传输时间及节点间同步时间。准确率(ACC)定义如式(24)所示：

$$ACC=(N_{tp}+N_{tn})/N_t \tag{24}$$

其中，N_{tp} 表示预测正确的正样本数，N_{tn} 表示预测正确的负样本数，N_t 表示总样本数。

由表 2、表 3 可以看出，1)当阈值与工作进程数相同时，GFC-ADADMM 算法比其他几种算法收敛时的系统时间更少，且收敛时准确率更高。如数据集为 URL，阈值与工作进程数相同时，相比 HAD-ADMM，ADMMLIB 以及 HSAC-ALADMM，GFC-ADADMM 算法的系统时间分别减少了 63%，55%，25%。一是因为 GFC-ADADMM 算法收敛时迭代次数更少，二是 GFC-ADADMM 节点间只有传输关联模型参数，节点间传输负载更小，因此可以有效减少数据传输时间。2)随着阈值的减少，HAD-ADMM，HSAC-ALADMM 以及 GFC-ADADMM 这 3 种算法的等待时间大大减少，主要是因为阈值减小，节点间每次通信的同步时间减少。随着阈值减小，ADMMLIB 算法中节点间同步时间也减少，但其每次通信时通信量恒定不变，阈值减小，算法收敛时迭代次数和数据传输时间增加，因此等待时间时间也可能会增加。该实验表明采用异步计算模型可有效减少系统运行时间。

表 2 数据集为 KDDB 时不同算法的运行时间及准确率

Table 2 Running time and accuracy of different algorithms on KDDB

算法	N	S	运行时间/s	计算时间/s	等待时间/s	准确率/%
HAD-ADMM	64	64	271.92	67.41	204.15	89.12
		32	180.57	56.01	124.56	89.10
		16	139.36	50.15	89.21	89.09
ADMMLIB	64	64	250.20	68.10	182.10	89.12
		32	159.23	56.23	103.00	89.11
		16	165.48	50.57	114.91	89.11
HSAC-ALADMM	64	64	202.54	68.85	133.69	89.12
		32	117.26	56.08	61.18	89.11
		16	105.33	50.87	54.46	89.12
GFC-ADADMM	4	4	153.85	46.35	107.50	89.17
		2	102.96	42.23	59.73	89.16
		1	73.18	40.91	32.27	89.16

表 3 数据集为 URL 时不同算法的运行时间及准确率

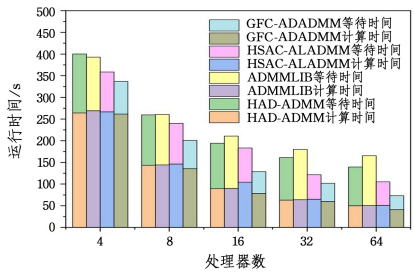
Table 3 Running time and accuracy of different algorithms on URL

算法	N	S	运行时间/s	计算时间/s	等待时间/s	准确率/%
HAD-ADMM	64	64	580.57	126.51	453.96	98.99
		32	493.75	117.95	375.80	98.98
		16	300.67	110.24	198.43	98.96
ADMMLIB	64	64	476.53	126.69	349.84	98.99
		32	488.00	118.41	369.59	98.97
		16	319.52	110.72	208.80	98.96
HSAC-ALADMM	64	64	283.58	127.26	156.32	98.99
		32	265.76	119.86	145.90	98.98
		16	200.85	111.72	89.13	98.98
GFC-ADADMM	4	4	212.59	107.24	105.35	99.23
		2	194.42	102.68	90.74	99.22
		1	169.93	96.81	73.12	99.22

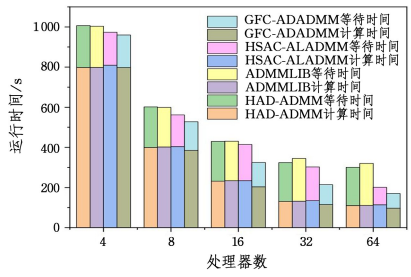
4.3 可扩展性测试

本节实验中几种算法的工作节点数仍然为 4，测试每个节点参与计算的处理器数分别为 1，2，4，8，16 时算法的运行时间变化情况。阈值设置为工作进程数的 1/4，其他参数设置与 4.1 节一致。图 2 给出了采用不用处理器数时几种算法的系统时间变化情况。图 3 给出了采用不用处理器数时几种算法的准确率变化情况。

由图 2 可知，1）随着单节点处理器数的增加，几种算法的计算时间减少，这是因为在数据集规模不变的情况下，单个处理器的计算任务减少。2）随着处理器个数的增加，GFC-ADADMM 等待时间减少明显，因为 GFC-ADADMM 工作进程数不会随着单节点处理器数的增加而增加，算法收敛时迭代次数不变，增加单节点处理器数不会改变数据传输时间，但单次迭代计算时间减少，所以等待时间减少。其他 3 种算法工作进程数随着处理器的增加而增加，因此算法收敛时迭代次数增加。HAD-ADMM 与 ADMMLIB 节点间数据传输量基本保持不变，增加工作进程数后单次计算等待时间减少，但总迭代次数增加，因此等待时间变化不明显。HSAC-ALADMM 算法中随着工作进程数增加，传输参数的密度降低，因此随着处理器的增加，节点间传输量减少，但工作进程数增加使算法收敛时总迭代次数增加。3）当处理器个数为 4 时，GFC-ADADMM 算法的等待时间比 HSAC-ALADMM 算法等待时间更短，虽然 GFC-ADADMM 与 HSAC-ALADMM 算法采用相同的通信算法，但 GFC-ADADMM 算法中关联参数的个数是不变的，在通信时可减少参数选择时间，因此节点间传输时间更短。



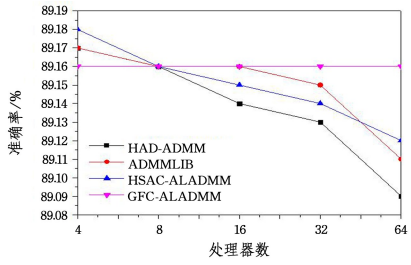
(a) KDDB



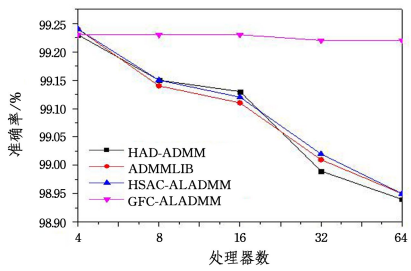
(b) URL

图 2 不同算法运行时间随处理器数的变化情况

Fig. 2 Running time of different algorithms vs. different number of processors



(a) KDDB



(b) URL

图 3 不同算法准确率随处理器数的变化情况

Fig. 3 Accuracy of different algorithms vs. different number of processors

由图 3 可知，1）当处理器数为 4 时，GFC-ADADMM 算法的准确率比其他 3 种算法的准确率更低，这是因为 GFC-ADADMM 算法只处理关联模型参数，而忽略了出现频率较低的特性，所以会在一定程度上损失算法准确率。2）随着处理器数的增加，GFC-ADADMM 算法的准确率基本不变，而其他 3 种算法的准确率随着处理器数的增加而降低，这是因为 GFC-ADADMM 算法节点内采用共享内存方式实现并行化，工作进程数不会随着单节点处理器数的增加而增加，而其他 3 种算法节点内采用多进程实现并行化，工作进程数随着单节点处理数的增加而增加，从而会影响算法收敛性。

由本节实验结果可以得出以下结论：1）与 HAD-

ADMM，ADMMLIB 及 HSAC-ALADMM 算法相比，基于 MPI/OpenMP 混合并行模式实现的 GFC-ADADMM 算法在不损失算法准确率的情形下可有效提高算法纵向可扩展性；2）当数据集规模较大时，采用异步计算模型可有效减少节点间同步时间，从而减少系统时间；3）GFC-ADADMM 算法基于数据集特性保持模型参数的稀疏性，传输参数关联特性保持不变，使用稀疏 allreduce 算法聚合参数时可达到更高的通信效率。

结束语 本文基于 allreduce 通信模型实现了通用一致性异步分布式 ADMM 算法。首先利用关联模型参数代替完整模型参数进行计算和通信，减少节点传输负载；然后根据数据集特征维度出现频率对每个节点关联参数进行过滤，以进一步减少节点间传输负载；最后，结合混合编程模型和异步 allreduce 框架实现通信高效的 GFC-ADADMM 算法。本文在实现 GFC-ADADMM 算法时，虽然每个计算节点只关联部分模型参数，但是在求解子问题时仍然采用稠密向量进行计算，并没有充分利用稀疏向量的优势。后续将进一步研究利用稀疏向量实行子问题的并行求解，以进一步提高算法的可扩展性。

参 考 文 献

[1] LI Y，WANG X，FANG W，et al. A Distributed ADMM Approach for Collaborative Regression Learning in Edge Computing[J]. Computers，Materials and Continua，2019，58(2)：493-508.

[2] BALAMURUGAN P，POSINASETTY A，SHEVADE S. ADMM for Training Sparse Structural SVMs with Augmented l1 Regularizers[C]// Siam International Conference on Data Mining. 2016.

[3] DHAR S，YI C，RAMAKRISHNAN N，et al. ADMM Based Scalable Machine Learning on Spark[C]// 2015 IEEE International Conference on Big Data. Santa Clara：IEEE，2015：1174-1182.

[4] LI W，LIU Y，TIAN Z，et al. Communication-Censored Linearized ADMM for Decentralized Consensus Optimization[J]. IEEE Transactions on Signal Processing，2020，6：18-34.

[5] LIU Y，YUAN K，WU G，et al. Decentralized Dynamic ADMM with Quantized and Censored Communications[C]//53rd Asilomar Conference on Signals，Systems，and Computers. 2019.

[6] ISSAID C B，ELGABLI A，PARK J，et al. Communication Efficient Distributed Learning with Censored，Quantized，and Generalized Group ADMM[J]. arXiv：2009. 06459，2020.

[7] ZHANG R，KWOK J. Asynchronous distributed ADMM for consensus optimization[C]// International Conference on Machine Learning. 2014：1701-1709.

[8] CTANG T H，HONG M，LIAO W C，et al. Asynchronous Distributed ADMM for Large-Scale Optimization—Part I：Algorithm and Convergence Analysis[J]. IEEE Transactions on Signal Processing，2016，64(12)：3118-3130.

[9] BOYD S，PARIKH N，CHU E，et al. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers[J]. Foundations & Trends in Machine Learning，2010，3(1)：1-122.

[10] ZHU R，NIU D，LI Z. A Block-wise，Asynchronous and Distributed ADMM Algorithm for General Form Consensus Optimization[J]. arXiv：1802. 08882，2018.

[11] JIN H，JESPERSEN D，MEHROTRA P，et al. High Performance Computing Using MPI and OpenMP on Multi-core Parallel Systems[J]. Parallel Computing，2011，37(9)：562-575.

[12] CHORLEY M J，WALKER D W. Performance Analysis of a Hybrid MPI/OpenMP Application on Multi-core Clusters[J]. Journal of Computational Science，2010，1(3)：168-174.

[13] GENKIN A，MADIGAN L D. Large-scale Bayesian Logistic Regression for Text Categorization[J]. Technometrics，2007，49(3)：291-304.

[14] ALGAMAL Z Y，LEE M H. A Two-stage Sparse Logistic Regression for Optimal Gene Selection in High-dimensional Microarray Data Classification[J]. Advances in Data Analysis and Classification，2019，13(3)：753-771.

[15] SHEVADE S K，KEERTHI S S. A Simple and Efficient Algorithm for Gene Selection Using Sparse Logistic Regression[J]. Bioinformatics，2003，19(17)：2246-2253.

[16] ELGABLI A，PARK J，BEDI A S，et al. GADMM：Fast and Communication Efficient Framework for Distributed Machine Learning[J]. Journal of Machine Learning Research，2020，21(76)：1-39.

[17] WANG S，LEI Y. Fast Communication Structure for Asynchronous Distributed ADMM under Unbalance Process Arrival Pattern[C]// 27th International Conference on Artificial Neural Networks. 2018：362-371.

[18] XIE J，LEI Y. ADMMLIB：A Library of Communication-Efficient AD-ADMM for Distributed Machine Learning[C]// IFIP International Conference on Network and Parallel Computing. Cham：Springer，2019：322-326.

[19] WANG D，LEI Y，XIE J，et al. HSAC-ALADMM：An Asynchronous Lazy ADMM Algorithm Based on Hierarchical Sparse Allreduce Communication[J]. The Journal of Supercomputing. 2021，77(8)：8111-8134.

[20] WANG D，LEI Y，ZHOU J. Hybrid MPI/OpenMP Parallel Asynchronous Distributed Alternating Direction Method of Multipliers[J]. Computing，2021，103(12)：2737-2762.

[21] HSIA C Y，ZHU Y，LIN C J. A Study on Trust Region Update Rules in Newton Methods for Large-scale Linear Classification[C]// Proceedings of the Ninth Asian Conference on Machine Learning. 2017：33-48.

WANG Dong-xia，born in 1983，Ph. D candidate，is a member of China Computer Federation. Her main research interests include distributed machine learning，parallel and distributed computing.

LEI Yong-mei，born in 1965，professor，Ph.D supervisor，is a member of China Computer Federation. Her main research interests include parallel and distributed computing，big data analysis，etc.

（责任编辑：杨雪敏）