**基于Spark比较采用数据分布与模型分布实现算法并行**

作者A，作者B

中国科学院计算技术研究所，北京市，100190

E-mail: 通讯作者电子邮件地址

摘 要： 随着大数据时代的降临, 分布式计算也广受关注。所谓分布式计算，其核心目标是把计算任务拆解成多个小的任务，分配到多个处理器上做计算。分布式计算或者分布式机器学习除了要把计算任务分布到多个处理器上，更重要的是把数据分布开来。除了使用数据分布式使机器学习模型并行以外，还可以将机器学习模型也分布在多台处理器上从而实现更细粒度的并行，简称模型并行方式。本文将会从编程范式，可扩展性，效率等方面比较两种算法并行方式在spark上实现的异同。

关键词：分布式计算；数据分布；模型分布；Spark

Comparision Between Algorithm Parallelization Implemented by Data distribution and Model Distribution

AUTHOR A, AUTHOR B, AUTHOR C

Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China

E-mail: [yiqunliu@tsinghua.edu.cn](mailto:yiqunliu@tsinghua.edu.cn)

**Abstract:** As with the big data era coming, distributed computing is wildly concerned. The target of distributed computing is to split the computing tasks into independent ones and detach them onto multiple computing nodes, such reducing the total running time cost. Besides data parallelization, model can also be distributed onto different computing nodes. This papaer will compare the differences on Programming paradigm，model scalability and training efficiency between machine learning algorighm parallelized by data distribution and model distribution based on spark platform.

**Keywords:** Distributed Computing; data distribution; model distribution; Spark

1 引言

近些年来，由于互联网普及度越来越高，数据量飞速地增长，大规模数据集加上高维数据类型使得传统的单机处理无论是运算能力还是时间效率都无已法胜任实际需要。面对大数据时代的数据处理任务，设计合理的并行系统，以及分布式计算框架成为了一个十分重大的挑战。传统并行方法有 MPI（Message Passing Interface）、PVM(Parallel Virtual Machine)，Hadoop MapReduce 并行框架；MPI 简单易用, PVM 实现了负载平衡和良好的容错性, Hadoop MapReduce 高度抽象容易编写。然而，PVM 和 MPI 的抽象度不高，Hadoop MapReduce 抽象程度高，适合离线批处理文件，能够实现机器学习算法的并行化，但是因为其高IO读写磁盘过于频繁，处理迭代计算不够灵活。

由 UC Berkeley AMP lab[1] 开发的通用并行分布式计算框架 Spark，支持内存技术、多迭代批量处理、流计算和图计算等多种范式，Spark 内存计算框架适合迭代式算法和交互式数据分能够提升大数据处理的实时性和准确性，现已得到很多企业的支持如阿里巴巴、百度、网易、英特尔等。相较于传统并行计算框架，其内存计算的特性能很好的适应迭代计算，同时在数据切分、并行处理、容错健壮性方面进行了封装，能良好的适应于并行化计算开发。

随着云计算概念的推广，为提高算法效率出现了多种并行处理数据方式，典型的并行方法有两种，分别是数据并行以及模型并行。它将海量数据集无差别的划分为 p 个数据子集，将数据子集发放到各个子处理机器上并行的对数据进行处理，最后汇合局部数据得出全局结果，最典型的应用是spark的Mllib；模型并行的代表是Graphlab[2], 它是一个基于BSP编程范式的机器学习平台，它将数据跟模型构成一个有向无环的二谱图进行迭代计算。相比于数据并行，模型并行是能够支持更细粒度的并行手段，它可以同时将数据子集跟模型子集分发到不同的处理机器上。

数据分布式并行实现简单，通信代价小，但是因为其将模型整个存在一个节点上，无法满足大规模模型并行化的需求；模型分布式相比之下编程要复杂，需要在训练模型之前将样本与模型抽象出一个二谱图，因为样本与模型都是图的节点，所以并行的通信开销较大，但是与此同时因为模型可以分布式存储，所以可以支持大规模模型的需求。为了验证这些猜想，本文会围绕两种并行方式在编程方式， 模型的可扩展性，以及计算效率三分方面进行比较。

本文结合以上的分析，在spark平台上分别用模型分布式跟数据分布式来实现并行计算，并根据实验结果在编程方式，通信效率，适用场景等方面对其进行比较。第2节详细介绍模型分布与数据分布，并以Logistic Regression 并行化作为样例介绍采用两种方式编程的一般方法；第3节详细叙述了在spark平台下采用不同并行方式训练NMF，LR等常用模型在效率上的差异；第4节概括比较模型并行跟数据并行的不同与优劣。

2 数据分布与模型分布

典型的并行方法有两种，分别是数据分布以及模型分布。数据分布将海量数据集无差别的划分为 p 个数据子集，将数据子集发放到各个子处理机器上并行的对数据进行处理，最后汇合局部数据得出全局结果，最典型的应用是spark的MLlib；模型分布式不仅将数据随机分发到各个计算节点上，还会将模型分发到各个节点上，实现更加细粒度的并行，这类的典型代表是GraphLab, 它是一个基于BSP编程范式的机器学习库。为了实现模型并行它将数据跟模型构成一个有向无环的二谱图进行迭代计算。

## 2.1 数据分布式

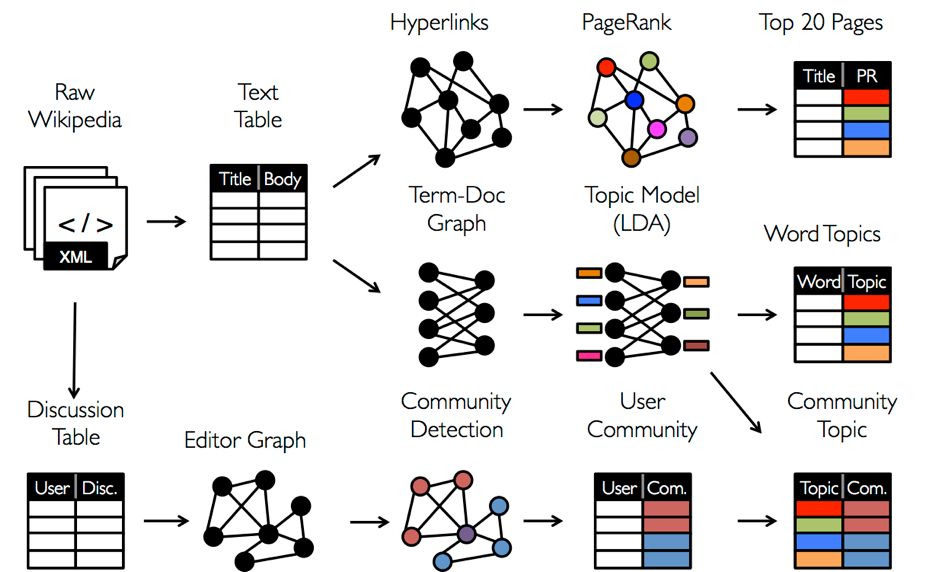
数据分布式是将数据分发到不同节点以实现算法并行的一类方式的统称。基于Hadoop MapReduce分布式架构实现的大规模数据挖掘与机器学习库Mahout就是通过数据并行来达到分布式计算的目的的。对于一个机器学习算法，MapReduce作业通常将输入的数据集划分为若干个相互独立的数据块, 然后以完全并行的方式在Map任务中计算, 接着在Reduce任务中对模型进行更新。

MLlib是建立在Apache Spark 上的分布式机器学习库，因为Apache Spark是一个基于内存的分布式计算框架，它使用RDD[3]来对分布式数据集进行抽象，所以在效率上mllib比基于MapReduce实现的分布式算法有100倍的性能提升。MLlib支持分类、回归、聚类、协同过滤、降维等主要机器学习算法。底层的数值优化算法主要包括SGD和L-BFGS (Limited-memory BFGS)，二者都是非常重要的机器学习参数优化算法。拿SGD 在Spark中的应用为例，MLlib中通过MiniBatchFraction参数指定用于计算梯度的样本比例，然后取平均值作为该点的随机梯度。跟Mahout类似，Mllib中也是采用数据分布实现模型并行。在MLlib中，数据集也是被分发到不同的计算节点上，每个计算节点计算不同的数据集得到梯度，然后汇总取平均作为该点的随机梯度，并更新模型。

## 2.2 模型分布式

模型分布式是将模型分布在不同的节点实现分布式计算的一类方式的统称。现有的比较常见的采用模型分布式来对机器学习算法并行化的库有Spark的graphx[4], Giraph[5], GraphLab等。这些支持大规模图数据的图计算库，采用的都是BSP(bulk synchronous *message passing* graph-parallel abstraction )编程范式。基于Spark Graphx使用Pregel[6]框架实现的常见算法包括Pagerank, LDA, Community Detection等，见图-1。

图-1 Graphx上的机器学习算法



Spark Graphx是Spark中专为大规模图计算设计的一个库，它使用spark中RDD抽象出一个有向图，并自带了一些常用的图算法，包括Pagerank, Connected Components, Triangle Counting等。Spark Graphx使用的编程框架是Pregel[3], 它是谷歌提出的一个用于分布式图计算的计算框架，该计算框架采用BSP编程范式来实现顶点之间计算、消息传递以及同步。虽然Graphx的优势并不在于解决机器学习算法的并行化问题，但是不妨碍其实现一般机器学习算法的分布式训练。基于Spark Graphx将机器学习模型并行化的通用做法是将样本跟模型构成一个二谱图，如上图中LDA算法的输入，Term-Doc Graph。由于样本还有模型参数都表示成了二谱图中的顶点，所以基于Graphx实现的机器学习算法不仅能够数据并行还能够模型并行化。

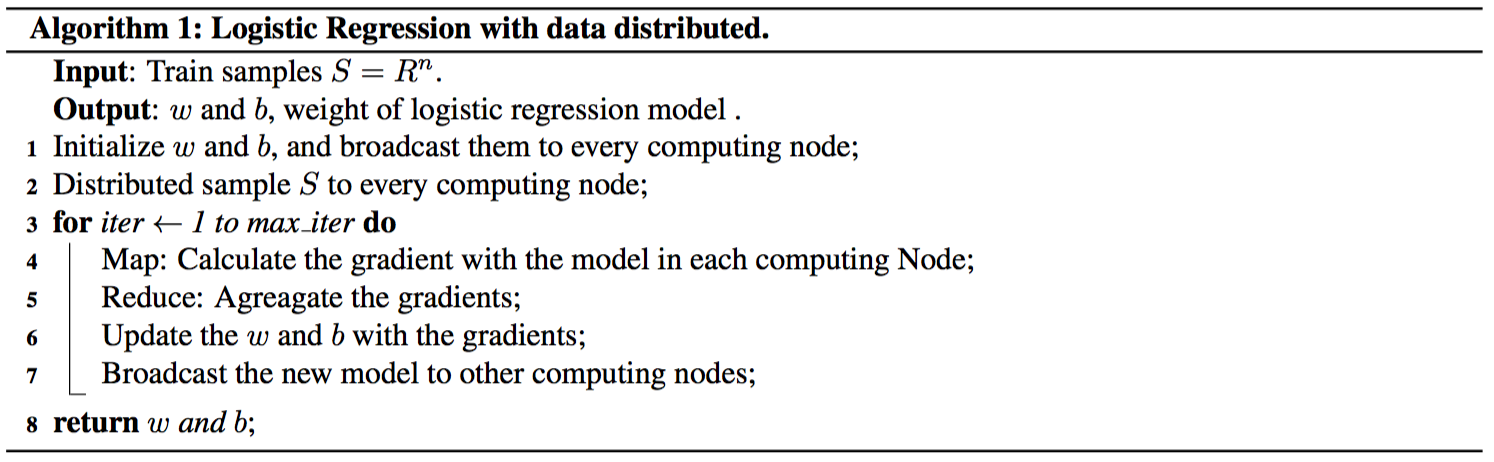
## 2.3 Logistic Regression的例子

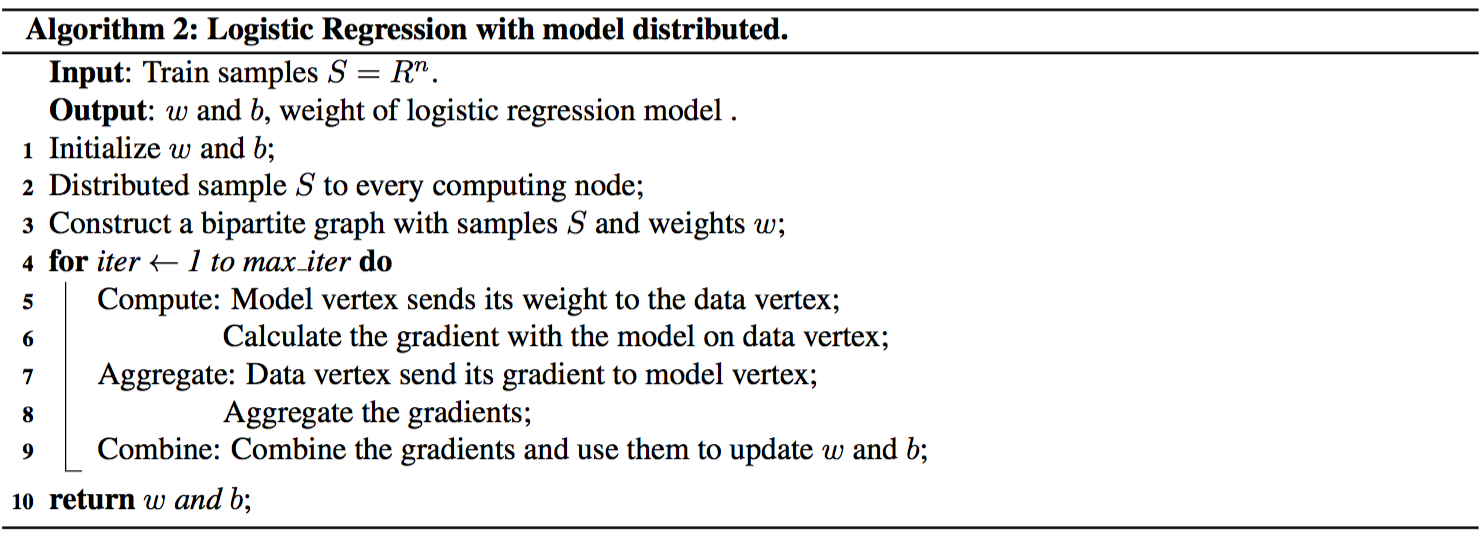
为了更进一步明确数据分布式跟模型分布式的区别，下面介绍在spark上用以上两种方式实现 Logitstic Regression。Logistic Regression 的目标函数由negative log-likelihood 跟L2正则项组成，见式(1), 其中表示第n个样本的真是label, 表示使用模型对第n个样本预测到的预测值。

根据(1)式，可求得b以及w的梯度，如下：

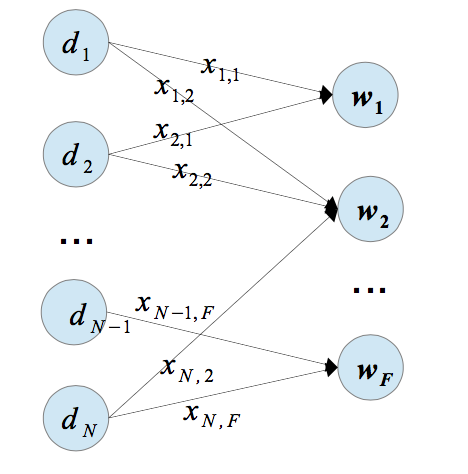
采用GD(Gradient Descent)的参数优化的方式, 可得w以及b的更新方式见下式：

下面分别是使用数据分布跟模型分布实现Logistic Regression的伪代码。





下图是使用模型分布方式将样本跟模型构造二谱图的示意图，见图[2]:



## 2.4 数据分布与模型分布

从编程方式上看，数据分布式不需要提前构造样本到模型的二谱图，而且从实现细节上讲，模型是通过广播分发到其他计算节点的，实现起来比较简单；相比之下，模型分布式不仅需要先初始化二谱图，而且每一个样本的梯度的计算都会涉及到计算节点的通信问题，所以为了提高效率需要在编程的过程中考虑到图的存储、图的划分等问题，不然模型的效率会大大降低。

从模型的可扩展性上分析，因为数据分布中所有的计算节点都需要保存一个模型的拷贝，所以很明显它对大规模模型的支持是非常有限的；但是模型分布式因为可以将模型分布存储到所有的计算节点上，所以它可以支持大规模参数的模型训练。

上面的分析尚不能推断出基于两种并行方式实现的模型的训练效率，论文在接下来的内容中会通过实验进行进一步的比较。

3 实验设计

## 3.1 实验设置

本文实验的环境是7台Spark服务器，系统的配置为6核主频为1.2GHz的GenuineIntel处理器，128G的RAM和2TB的硬盘，使用的操作系统版本为CentOS release 6.6(Final)。

实验中将分别统计基于spark平台使用数据分布和使用模型分布实现的NMF，Logistic Regression两个算法在在不同样本规模上模型训练效率方面的差异。

本次实验的环境配置信息见表-1：

表1 实验环境参数

|  |  |  |  |
| --- | --- | --- | --- |
| Spark.cores.max | Spark Excutor memory | Spark Excutor nums | partition |
| 50 | 50G | 20 | default |

## 3.2 数据集

表-1中为本次实验使用数据详细描述，前四个数据集分别是movie-lens中用户给电影打分的历史数据，其中每个样本有三个字段，分别是User\_id, Item\_id 以及Rate，本文将用这四个数据集测试使用模型分布式和使用数据分布式实现的NMF(Non-negative Matric Factorization)算法的训练效率。表-2中包含了movie-lens数据集中用户数目，电影数目，以及数据稀疏性的统计信息。A1a数据集来自于UCI的一个人口普查收入数据集，其中有8个分类特征6个连续特征，本实验中使用的是libsvm官网提供的转换后的a1a数据集的结果。实验将四个样本规模不同的数据集分别用来测试使用模型分布与使用数据分布实现的Logistic Regression算法的训练效率。

表2 实验数据集详细描述

|  |  |  |  |
| --- | --- | --- | --- |
| 数据集 | 任务 | 样本规模 | 特征维度 |
| Movie-lens 100k | 回归 | 100,000 | 2 |
| Movie-lens 1M | 回归 | 1,000,209 | 2 |
| Movie-lens 10M | 回归 | 10,000,054 | 2 |
| Movie-lens 20M | 回归 | 20,000,263 | 2 |
| A1a 30K | 分类 | 32561 | 122 |
| A1a 100K | 分类 | 99288 | 122 |
| A1a 1M  A1a 12M | 分类  分类 | 1025441  12305292 | 122  122 |

表3 Movie-lens 数据集统计

|  |  |  |  |
| --- | --- | --- | --- |
| 数据集 | 用户数目 | 电影数目 | 稀疏比 |
| Movie-lens 100k | 462 | 1682 | 12.8% |
| Movie-lens 1M | 6040 | 3952 | 4.1% |
| Movie-lens 10M | 71567 | 65133 | 0.21% |
| Movie-lens 20M | 138493 | 131262 | 0.11% |

## 

## 3.3 实验结果

在实验中，所有模型使用梯度下降的方式迭代50次进行训练，最后统计使用数据分布式与模型分布式实现的两种算法的平均每轮的迭代时间。图-2与图-3是分别对LogisticRegression 与NMF两个算法的测试结果。

图2 NMF平均迭代时间

图-3 LR平均迭代时间

根据分析结果可以发现，在模型的训练效率上数据分布式相比于模型分布式高了很多，这个在LR的平均迭代时间上尤为明显。这也验证了之前的猜想，模型分布式由于模型与特征都被抽象成图中的顶点，并且存储在不同的计算节点上，所以并行的通信开销较大；相比之下，在计算节点的内存足够模型存储以及计算的情况下，数据分布的方式通信开销并不大，更多的时间是花费在样本计算梯度上。关于数据分布与模型分布的通信开销可以从平均迭代时间的增幅来看，当数据集变大，模型并行的平均迭代时间增加的比较平缓，而数据分布的平均迭代时间几乎是随着样本数量线性增长的。

在Movie-lens-20M数据集上，采用模型分布式训练NMF模型要比采用数据分布训练效率高，并因此注意到模型分布的时间跟数据分布的时间比随着样本数据的稀疏程度在不断降低，如下图-4所示。

图-4 NMF两种分布方式迭代时间比

由上图所示，随着数据越来越稀疏降低，模型分布式与数据分布实现的模型的训练时间比也在不断降低。考虑到模型分布式是将样本跟模型构成了一个二谱图分布式存在各个计算节点上，可以推断出随着图越来越稀疏，样本特征的稀疏性所带来的通信时间的减少已经超过了模型分布的通信开销。

## 3.3 实验结论

实验结果显示，对于非稀疏性训练样本，采用数据分布式的模型训练效率比较高；对于稀疏性样本，在模型迭代过程中样本和模型会构成一个稀疏的二谱图，它所减少的计算开销可能会超过因为模型分布带来的通信开销，最终情况由样本的稀疏性决定。

4 总结与展望

比较基于spark平台使用不同并行手段去实现机器学习算法的差异性能更好地了解不同并行方式的应用环境，这样能使算法工程师更好结合实际应用的特点利用spark平台的优势设计并实现高效的分布式机器学习算法。实验结果表明，对于大规模稀疏数据集采用模型分布式不仅可以完美解决模型规模大，单个节点无法存储的问题，还能有效降低通信开销；对于小规模模型，使用数据并行的方式可以使模型训练更加快速。

参 考 文 献

1. Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster Computing with Working Sets[J]. HotCloud, 2010, 10: 10-10.
2. Low Y, Bickson D, Gonzalez J, et al. Distributed GraphLab: a framework for machine learning and data mining in the cloud[J]. Proceedings of the VLDB Endowment, 2012, 5(8): 716-727.
3. Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2-2.
4. Xin R S, Gonzalez J E, Franklin M J, et al. Graphx: A resilient distributed graph system on spark[C]//First International Workshop on Graph Data Management Experiences and Systems. ACM, 2013: 2.
5. Avery C. Giraph: Large-scale graph processing infrastructure on hadoop[J]. Proceedings of the Hadoop Summit. Santa Clara, 2011.
6. Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010: 135-146.