增量式迭代计算模型研究与实现

宋 杰[1] 郭朝鹏[1] 张一川[1] 张岩峰[2] 于 戈[3]

[1](东北大学软件学院 沈阳 110819)

[2](东北大学计算中心 沈阳 110819)

[3](东北大学信息科学与工程学院 沈阳 110819)

摘 要 不动点迭代广泛存在于数据挖掘和机器学习算法中,这些算法已应用到诸如社会网络分析、高性能计算、推荐系统、搜索引擎、模式识别等诸多领域中.在云计算环境中,利用MapReduce编程模型所带来的便利,通过普通的PC集群运行相应的迭代算法,可以提高迭代算法的执行效率.但由于数据的快速变化,每当数据发生改变,整个迭代算法也需要重新运行,这将会导致大量的运算资源浪费和性能损失.文中研究基于原始迭代结果和新增数据的增量迭代计算DELTA(Delta data based incrEmentaL iTerAtive computing),并提出DELTA模型以解决上述问题.文中理论证明了DELTA模型的正确性,阐述了其适用范围,并列举了PageRank、K-means和Descendant Query算法在DELTA模型中的运用.文中还扩展HaLoop为ΔHaLoop框架,使其支持增量式的迭代计算.通过一系列的测试用例,对DELTA模型功能、性能进行了分析和讨论,实验结果表明DELTA模型在获得准确的迭代结果的基础上性能优势明显.文中提出的DELTA模型能够适应多数迭代算法,对云计算环境下的迭代计算的应用和优化起到推动作用.

关键词 云计算;大数据;MapReduce;迭代计算;增量迭代

中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2016.00109

Research and Implementation Incremental Iterative Model

SONG Jie[1] GUO Chao-Peng[1] ZHANG Yi-Chuan[1] ZHANG Yan-Feng[2] YU Ge[3]

[1](*Software College*, *Northeastern University*, *Shenyang* 110819)

[2](*Computing Center*, *Northeastern University*, *Shenyang* 110819)

[3](*School of Information Science and Engineering*, *Northeastern University*, *Shenyang* 110819)

Abstract The fixed point iterative algorithms widely exist in the area of data mining and machine learning, which have been applied in many fields, such as social network analysis, high-performance computing and recommended system. In cloud computing environment, we can utilize the convenience brought by MapReduce to improve the efficiency of iterative algorithms on big data through running the algorithm on larger PC-cluster. However, the entire iterative algorithm has to be re-executed when new data is introduced, which cause large amount of computing resource wastes and performance losses. In this paper, the original iterative results new data based incremental iterative computing, which is named as DELTA(Delta data based incrEmentaL iTerAtive computing), is well studied, and the corresponding DELTA model is proposed. We prove the correctness of the model, and describe the application scope. Then, the application cases of DELTA model applying on the iterative algorithms are enumerated, such like PageRank,

收稿日期:2014-09-07;在线出版日期:2015-04-21.本课题得到国家自然科学基金(61433008,61202088,61272179,61173028)、教育部博士点基金(20130042120006)、教育部-中国移动科研基金项目(MCM20125021)、中国博士后科学基金面上基金(2013M540232)、中央高校基本科研业务费专项资金(N130417001)和辽宁省博士启动基金(201403314)资助.宋 杰,男,1980年生,博士,副教授,中国计算机学会(CCF)高级会员,主要研究方向为大数据存储与管理、迭代计算.E-mail:songjie@mail.neu.edu.cn.郭朝鹏,男,1990年生,硕士研究生,主要研究方向为迭代计算.张一川,男,1981年生,博士,讲师,主要研究方向为迭代计算框架与算法.张岩峰,男,1981年生,博士,副教授,主要研究方向为迭代计算.于 戈,男,1962年生,博士,教授,主要研究领域为数据库理论.

K-means and Descendant Query. Finally, ΔHaLoop is implemented by extending HaLoop to support DELTA model. A series of test cases are designed to analyze the DELTA model on functionality and performance. The results show that the model improves the iteration performance without any loss of accuracy. The DELTA model proposed in this paper can adapt many iterative algorithms, which promotes the application and optimization of iterative algorithms in cloud computing environment.

Keywords cloud computing; big data; MapReduce; iterative computing; incremental iterative

1 引 言

不动点迭代广泛存在于数据挖掘和机器学习算法中,这些算法已应用到诸如社会网络分析、高性能计算、推荐系统、搜索引擎、模式识别等诸多领域中.例如:著名的 PageRank 算法根据网页之间的链接关系,从任意迭代初始值开始,根据迭代函数更新每个网页的 PageRank 值直至收敛[1];类似迭代算法还包括最大期望(Expectation-Maximization)算法、*K*-means 算法[2]、协同过滤(Collaborative Filtering)算法[3]、SVM 算法[4]等.由此可见迭代算法有着非常广泛的应用范围.

随着人类进入到信息化时代,互联网、传感器和生物信息学等领域的快速发展,数据量呈现爆炸型增长.当今时代是一个数据量爆发的时代[5-6],在大数据环境下,迭代算法出现诸多不适用性,其中由于普通服务器的计算能力有限,算法运行时间往往令人无法接受.大数据上的迭代算法的运行消耗完全超出了单服务器的承受极限,因此学术界开始寻找分布式的迭代算法执行环境[7].基于 Hadoop① MapReduce[7]的 HaLoop 框架[8]是目前为止较为成熟的迭代计算框架.利用 HaLoop 框架和 MapReduce 编程模型所带来的便利,可通过廉价的 PC 集群获得强大的计算能力,支持大数据环境下众多迭代算法.

然而,我们注意到,时变性是大数据的一个特点.大数据是数据量庞大且高速增长的数据,当数据量增加后,原始数据的迭代结果将不再适用,整个迭代算法也需要在数据全集上重新运行,这将浪费大量的时间和资源.若能够通过新增数据集和已知的迭代结果完成增量式的迭代计算,则可以很大程度上提高迭代计算的效率.而就我们目前的知识,现存少量模型和算法层面的增量式迭代计算研究,成熟的增量式迭代框架也屈指可数,文中将在相关工作部分加以分析和比较,此外较相关的工作是以增量的方式更新元数据或维护索引等.例如 Google 的 Percolator[9]系统可以以增量的方式更新索引,Incoop[10]系统可以增量的执行 MapReduce 作业.

本文研究增量式的迭代计算模型以及其实现框架,而非迭代算法本身,旨在提出适用于大部分算法的,基于原始迭代结果和新增数据的增量迭代计算(Delta data based incrEmentaL iTerAtive computing, DELTA)模型和框架.DELTA 模型基于新增数据集和原始迭代结果,进行新一轮迭代,在不损失精度的前提下提升迭代性能.本文采用复合函数和集合抽象迭代过程,定义了 DELTA 模型,阐述其适用范围,设计了 DELTA 模型中的关键算法,证明了其性能优势和迭代结果的正确性,并示例性地给出了其在 PageRank、*K*-means、Descendant Query 迭代算法中的应用;本文还扩展 HaLoop 使其支持 DELTA 模型;最后本文通过实验验证 DELTA 模型的正确性和性能优势.

本文第 2 节介绍相关工作;第 3 节定义 DELTA 模型,并详述模型的正确性证明和适用范围以及在 PageRank、*K*-means 和 Descendant Query 算法上的应用;第 4 节简要介绍支持 DELTA 模型的 HaLoop 扩展框架的结构以及在其他框架上的实现分析;第 5 节通过实验验证 DELTA 框架的功能和性能;最后,在第 6 节对本文的研究进行总结并提出进一步的工作.

2 相关工作

迭代算法,例如 PageRank[1]、HITS[11]、神经网络计算[12],都有一个共同的特点:数据的计算过程是迭代的,直至满足一个确切的收敛条件.国内外已有较多针对迭代算法的研究成果,相关技术上也比

① http://hadoop.apache.org/

较成熟.部分研究将迭代算法等价于图的相关算法，并给出理论证明.部分文献也涉及了增量式的迭代算法研究.例如，文献[13]给出了增量式的迭代计算的相关条件，同时给出了相关增量式的迭代计算的例子以及反例.文献[14]研究当迭代结构发生改变时，如何继续进行迭代.上述文献虽然对不动点迭代算法进行了深入的研究，但由于发表时间较早，尚未采用分布式计算环境，而且均针对特定算法.此外，若使用上述文献中的方法进行大数据迭代计算，会由于数据量以及运算量超过单服务器承受能力而无法即时获得准确的迭代结果.

然而，MapReduce 对于迭代计算并没有提供直接的支持.一种变通的方法是编程人员通过程序设计，精心地布置 MapReduce 任务，同时编写一些特殊的算法来支持迭代算法，例如，用于检测收敛条件是否满足的相关算法.这样做将会导致编写迭代算法的过程异常复杂，一些简单的作业也需要采用多个 Map 和 Reduce 任务来实现，且大量中间结果的存储和传递导致 I/O 资源和网络资源的浪费.

Bu 等人[8]在提出了 HaLoop 框架，基于云计算技术提供迭代算法的运行框架，使得迭代计算更高效.HaLoop 扩展了 Hadoop，以更适合迭代计算，包括：(1)编程接口更加适用于迭代算法；(2)框架实现迭代终止条件的检测；(3)任务尽量满足数据本地计算的特性，两次迭代任务尽量使用相同的数据；(4)采用缓存和索引两种优化机制.但是 HaLoop 的动态和静态数据无法分离，且没有一个客观的停止迭代的标准[5].此外，还存在一些迭代计算框架的实现.Twister[15]是一个基于流，支持迭代算法的 MapReduce 框架，它将全部数据存放在分布式缓存中，采用独立模块传递所有的消息和数据.但是数据驻留内存的限制使其难以实用，且其计算模型的抽象程度不高，支持的算法也很有限.iHadoop[16]实现了 MapReduce 的异步迭代，其任务分配器提供一种数据定位机制，从而减少数据的冗余传输以及 I/O 和网络资源的浪费.PrIter[17]是基于 Hadoop 的，支持带优先级的迭代计算框架，能够保证迭代的快速收敛，适合实时查询需求.iMapReduce[18]是一种基于 MapReduce 的迭代计算模型，它尽量地减少 MapReduce 作业的数量，通过缓存的方法除去 Shuffle 阶段中的静态数据，并允许 Map 任务的异步执行，以此优化迭代计算过程.但是它的静态调度策略和粗粒度的任务可能会导致资源利用不佳和负载不均衡.REX[19]结合了大数据环境中的计算平台和传统的 RDBMS 技术，提出了 RQL 在大数据环境中模拟 SQL 的特性如连接、聚集、子查询、递归查询等.其主要创新点在于：对于迭代计算而言，可以仅传递迭代步骤间的变化，从而大幅度的优化迭代(递归)查询.在上述迭代计算框架中均提供基于云计算技术的迭代算法执行环境，但都没有考虑增量数据的迭代问题，当新数据产生时，上述框架均需要重新进行迭代.此外，支持迭代计算的分布式框架还有 HaLoop[8]、Spark[20]、Naiad[21]等，后文会提及.本文基于 HaLoop 实现了支持 DELTA 模型的 ΔHaLoop，HaLoop 又是基于 Hadoop 分布式文件系统的，而其他主流的迭代框架，如 Spark、Twister 和 Naiad，或基于其他文件系统，或基于分布式缓存.但理论上，这些框架都可以扩展以支持 DELTA 模型.

此外我们也参考了特定算法的增量式迭代优化，本节以 K-Means 算法为例.文献[22]基于 FGKA(Fast Genetic K-means Algorithm)提出了 IGKA(Incremental Genetic K-means Algorithm)，用于基因数据的聚类；文献[23]提出基于图割理论的 Kernel K-means 算法，同时进一步提出了增量式加权 Kernel K-means 算法；文献[24]研究了 K-means 收敛的情况，并基于 Distortions Reduction 提出增量式 K-means 算法，解决了 K-means 收敛到局部极值问题；文献[25]提出了基于手机轨迹数据的紧凑表示法和轨迹相似性度量，基于上述理论该研究又提出了增量式的聚类方法用于发现空间中相似的移动终端；文献[26]基于现有的增量式神经网络(IGNG)提出了其改进算法 I2GNG 用于证券分类(Invoice Classification)；文献[27]提出了增量式分类方法用于在线文档分类，但是其研究重点是相似性度量和文档结构的抽取.本文提出的 DELTA 模型并非针对 K-means 算法，并且和上述研究也有较大的区别：一部分算法是单机算法，难以扩展到分布式环境，即使用 MapReduce 成功改写这些算法，其执行性能也很低，如文献[22-23]；另一部分算法采用“增量迭代”技术，其目的是为了研究算法收敛性，或特定领域的聚类算法，与本文所定义的 DELTA 目标不同，如文献[24-27].在相关文献中，文献[28]属于增量式的 K-means 算法，且可以移植到 MapReduce 框架中，且移植后的算法与本文实验中所采用的 K-means 全量迭代算法是一致的，本文 5.4 节对其进行性能比较.

通过分析可以看出，大多数的迭代计算框架都

优化迭代算法的执行过程，从而提高计算效率. 具体的措施如下：(1) 使用缓存和索引的方法，尽量地减少数据传输过程中本地和网络 I/O 代价；(2) 使用内存优化的相关技术，加速迭代算法的执行；(3) 使用缓存技术优化迭代计算中终止条件判定操作，从而减少迭代算法执行时间；(4) 优化 Map 任务和 Reduce 任务之间的通信方法，从而实现异步迭代；(5) 引入 RDBMS 的递归查询技术支持迭代更新. 在迭代框架的研究方面，当前研究基于算法运行流程角度更新迭代结果，但现有研究均未涉及基于原始迭代结果和新增数据的 DELTA 模型和框架. DELTA 作为迭代计算的一种重要的优化手段，能够利用原始迭代结果以及新增数据在较短时间内得到新的迭代结果. 本文从算法和迭代框架层面出发，研究内容与现有研究有着本质不同.

3 增量迭代计算模型

DELTA 模型是迭代算法和计算过程的抽象，该模型利用已有迭代结果以及新增数据集，计算新的迭代结果，该模型是增量迭代计算框架的基础. 在本节中，首先定义迭代、迭代轮、增量迭代等相关概念；随后给出 DELTA 模型的形式化表达；最后介绍模型证明、差集算法和模型应用.

3.1 模型定义

定义 1. 迭代. 迭代是用计算机解决问题的一种基本方法. 计算机对一组指令或一定步骤重复执行，在每次执行这组指令或步骤时，由变量的原值推出它的新值.

根据定义 1，一个迭代算法实际上是函数的重复执行，在执行过程中对一组变量进行更新，直到达到某种终止条件为止，由此可得迭代算法的 3 个要素为迭代变量、迭代函数和迭代的终止条件.

(1) 迭代变量. 在迭代算法中，至少存在一个直接或间接地不断由旧值递推出新值的变量，这个变量就是迭代变量.

(2) 迭代函数. 由迭代变量的前一个值推出其下一个值的函数.

(3) 迭代终止条件. 当迭代终止条件满足时，迭代算法运行结束，输出迭代结果，此时也称之为迭代算法收敛.

定义 2. 迭代轮. 迭代变量通过迭代函数更新一次的过程称之为一次迭代轮，或一轮迭代. 整个迭代算法可以由 $n(n\geq 1)$ 个迭代轮(n 轮迭代)组成，记作 $t_1,t_2,\cdots,t_n$，$\forall i\in[1,n-1]$，t_i 的输出是 t_{i+1} 的输入，t_1 的输入定义为初始化迭代变量以及参与迭代的数据，t_n 的输出即为最终的迭代结果.

设迭代算法 $x=f_n(x_0,L)$：根据某迭代初始变量 x_0 和迭代数据 L，经过迭代计算得到迭代结果 x. $f(x,L)$ 是迭代函数，亦可视为某轮迭代. $f(x,L)$ 的参数为迭代变量的初始值 x_0 以及迭代数据 L. 记函数 $f(x,L)$ 的 n 次自身复合函数 $f(f(\cdots f(f(x,L),L)\cdots,L),L)$ 为 $f_n(x,L)$. $f_n(x_0,L)$ 表示迭代函数作用于迭代数据 L 上，经过 n 轮达到收敛状态.

当基于迭代数据 L 的迭代计算完成后，由于业务增长而产生的新的迭代数据 ΔL 即为增量数据. 例如，PageRank 算法的网页链接关系数据和社会网络的图状态数据每天都会增加，于是，需要根据全量数据更新迭代结果. 在后文 3.2 节会详细描述，我们定义 L 为原始数据，ΔL 为增量数据，$\Delta L-L$ 为差集数据，$\Delta L/L$ 为关联数据，$\Delta L\cup L$ 为全量数据，一般的，$|\Delta L|\ll|L|$. 在原始数据上的迭代称为原始迭代，在全量数据上的迭代称为全量迭代. 两者算法相同但针对的数据不同. 全量迭代能够得到更新后的迭代结果但消耗大量资源. 本文的 DELTA 模型将依赖定义 3 中“增量迭代”的定义. 后文将证明增量迭代结果与全量迭代结果相同，但性能优于全量迭代.

定义 3. 增量迭代. 增量迭代是根据新增数据 ΔL 和原始迭代结果获得新的迭代结果的迭代方法，且可由式(4)表达.

$$R=f_\alpha(R_0,L) \tag{1}$$

$$\Delta R=f_\beta(\Delta R_0,\Delta L-L) \tag{2}$$

$$R^*=f_\gamma(R\cup\Delta R,L\cup\Delta L) \tag{3}$$

所以

$$R^*=f_\gamma[f_\alpha(R_0,L)\cup f_\beta(\Delta R_0,\Delta L-L),L\cup\Delta L] \tag{4}$$

根据式(1)～(3)，增量迭代的步骤定义为：

(1) 原始步. 在原始数据 L 上进行 α 轮迭代直至收敛，初始迭代变量集为 R_0，收敛后得到原始数据上的迭代结果集 R，如式(1)所示，该步骤在增量数据 ΔL 到来时已经完成；

(2) 增量步. 在差集数据 $\Delta L-L$ 上进行 β 轮迭代直至收敛，初始迭代变量为 ΔR_0，收敛后得到差集数据上的迭代结果集 ΔR，如式(2)所示；

(3) 合并步. 在全量数据 $L\cup\Delta L$ 上继续 γ 轮迭代直至收敛，初始迭代变量集为 $R\cup\Delta R$，收敛后得到最终迭代结果集 R^*，如式(3)所示.

定义 3 即定义了 DELTA 模型采用的迭代方法，且定义 3 同样适用于原始迭代的情况. 按式(4)，

在初次迭代时，原始的迭代结果 $R=\varnothing$，原始数据 $L=\varnothing$，这时所有的迭代数据被当作增量数据 ΔL 处理，$\alpha=\gamma=0$.

3.2　迭代数据

迭代数据是由数据以及数据间的关系所组成的集合. 如 PageRank 算法中迭代数据是由页面和页面之间的链接关系所组成的集合. 设迭代数据符合图关系 L. 在 L 中节点的集合为 $V(L)$，边的集合为 $E(L)$，其中 $E(L)$ 是节点所组成序偶的集合. 在 L 中，$\forall u,v\in V(L)$，如果 u,v 是有关系的，则有 $\langle u,v\rangle\in E(L)$. 图关系 ΔL 同理.

如图 1 所示，将图 L 和 ΔL 使用邻接矩阵的方式表示. 在图 1 中，$V(L)=\{A,B,C,D\}$，$V(\Delta L)=\{E,F,G\}$. 如果任意两个节点有关系，则在矩阵中标识为 1，否则为 0. 矩阵被分为 4 个区域，如图所示编号分别是 M_1,M_2,M_3,M_4. 其中，原始数据 $L=M_1$；增量数据 $\Delta L=M_2+M_3+M_4$；全量数据 $\Delta L\cup L=M_1+M_2+M_3+M_4$；定义差集数据（$\Delta L-L=M_4$）为 ΔL 的与 L 无关的最大子图；定义关联数据（$\Delta L/L=M_2+M_3$）为 ΔL 的与 L 关联的最大子图，$\Delta L=(\Delta L-L)\bigcup(\Delta L/L)$，差集数据和关联数据计算方法如下：

$$\begin{aligned}&\Delta L-L=\langle V(\Delta L-L),E(\Delta L-L)\rangle,\\&V(\Delta L-L)=V(\Delta L)-V(L),\\&E(\Delta L-L)=\\&\quad\{\langle u,v\rangle\mid\langle u,v\rangle\in E(\Delta L)\wedge u,v\in V(\Delta L-L)\}\end{aligned}\tag{5}$$

$$\begin{aligned}&\Delta L/L=\langle V(\Delta L/L),E(\Delta L/L)\rangle,\\&V(\Delta L/L)=V(\Delta L)\bigcap V(L)\bigcup\\&\quad\{v\mid\langle u,v\rangle\in E(\Delta L)\wedge u\in V(\Delta L)\bigcap V(L)\},\\&E(\Delta L/L)=\{\langle u,v\rangle\mid\langle u,v\rangle\in E(\Delta L)\wedge u,v\in V(\Delta L/L)\}\end{aligned}\tag{6}$$

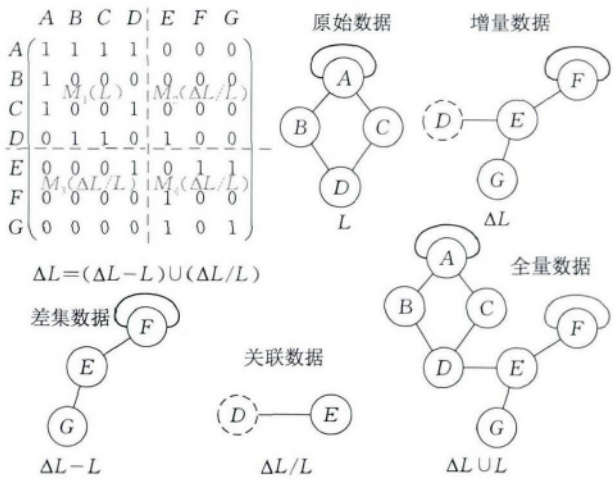

图 1　迭代数据的邻接矩阵表达

本文更多地考虑式(5)的差集计算，我们称为差集算法. 在图 1 中，M_1 和 M_4 是相互独立的，M_4 是增量数据中与原始数据独立的部分，增量迭代分别对 M_1 和 M_4 运行迭代算法，对应式(1)和(2). 然而还需要考虑增量数据和原始数据的关联部分 M_2+M_3，因此需要将 M_1 的迭代结果 R 和 M_4 的迭代结果 ΔR 合并，并在全量数据上继续迭代直至收敛.

3.3　迭代函数

迭代函数 f 的设计取决于迭代算法，且 f 需满足以下条件：f 在迭代轮中保持不变，且可以分解为若干变换函数 h 的代数运算. 针对某一个迭代变量 x_i，有形如：$f(x_i,L)=h_1(x_1,L)\oplus h_2(x_2,L)\oplus\cdots\oplus h_n(x_n,L)(i\in[1,n],n$ 为迭代变量个数），其中 $\oplus$ 为代数运算，h 函数是某迭代变量对当前迭代变量的变换函数，h 函数对 $\oplus$ 运算是分布的，$h(a\oplus b)=h(a)\oplus h(b)$. 为描述简便，我们将 L 视为常量，在 f 函数和 h 函数中省略.

若仅考察一个迭代变量 r 在整个迭代过程的变化，在一轮迭代中，该迭代变量吸收了其他迭代变量对它的变换，将这些变换聚集起来改变自身. 如果在某轮迭代中某变量由 r_0 变换为 r^*，其他变量为 r_1 和 r_2，迭代函数为 f，而变换函数为 h，聚集运算为 $\oplus$，那么一轮迭代可以表达成 $r^*=f(r_0,r_1,r_2)=h(r_0)\oplus h(r_1)\oplus h(r_2)$，一个变量的多轮迭代可以看作 $\sum_{i=1}^{n}\oplus h(r_i)$ 递归调用. 因此，有如下定义.

定义 4.　迭代计算的分解描述. 迭代计算可以分解为每一个迭代变量的变化，而一个迭代变量的变化可以分解为该变量每一轮变化的积累，而每一轮变化的积累又可以分解为其他变量对该变量的变换的代数聚集. 在这种分解描述下，某个迭代变量的收敛条件为变化为零，整个迭代计算的收敛条件为所有的迭代变量均收敛. 为描述简单，令迭代变量变化为零时仍可以继续迭代，以此保证所有变量同时收敛.

定义 4 中，“变化”表示某迭代变量自身的改变，这种改变来源于其他迭代变量对该变量的“变换”，h 函数则为“变换函数”. 对于某一轮迭代，设由 n 个迭代变量所组成的集合为 $R=\{r_1,r_2,\cdots,r_n\}$. 其中迭代函数为可以分解为 $f_1,f_2,\cdots,f_n$. 迭代变量 r_i 第 k 轮（k 为自然数）迭代可以表示为 $r_i^k=fi(r_1^{k-1},r_2^{k-1},\cdots,r_n^{k-1})$. 由定义 4 可知 fi 可以继续分解为式(7)形式：

$$r_i^k=h_i^1(r_1^{k-1})\oplus h_i^2(r_2^{k-1})\oplus\cdots\oplus h_i^n(r_n^{k-1})\tag{7}$$

式(7)可以解释为迭代变量 r_i^k 吸收其他迭代变量对其的变换,累积这些变换,并变化自身值.式(7)中函数 h_i^j 表示 r_j 对 r_i 的变换函数.设 n 个迭代变量的初始值为 $\{r_1^0, r_2^0, \cdots, r_n^0\}$,等同于第 0 轮迭代变量值,那么,当 $i \neq j$ 时 $h_i^j(r_j^{k-1})$ 由 f 函数确定;当 $i=j$ 时,$h_i^j(r_j^{k-1}) = r_j^{k-1}$;运算⊕为不同迭代变量对当前迭代变量变换的聚集操作,例如在 PageRank 算法中⊕为加法(+),K-means 算法中⊕为集合并(∪).

本文提出的 DELTA 模型要求其迭代函数能按定义 4 和式(5)分解描述,且⊕为代数运算,h 函数对⊕运算是分布的.

3.4 理论证明

本节将从理论上证明 DELTA 模型在性能上的优势,并证明增量迭代和全量迭代的迭代结果是一致的.

(1)性能优势证明

我们首先假设迭代算法的执行时间仅与迭代数据大小有关,则原始迭代的执行时间为 $\tau(|L|)$;增量迭代的执行时间为 $\tau(|\Delta L - L|) + \tau(|\Delta L \cup L|)$,而全量迭代的执行时间为 $\tau(|\Delta L \cup L|)$,显然增量迭代的性能无法优于全量迭代.分析知迭代算法的执行时间还与迭代变量初始值和迭代数据有关.根据定义 4,一个迭代轮可以分解"迭代变量吸收其他迭代变量对它的变换,将这些变换聚集起来改变自身"这一过程,因此,若迭代变量初始值越不合理,迭代数据越紊乱,变量间相互变换越多,迭代轮数越大,迭代时间越长.

原始迭代,增量迭代的原始步、增量步、合并步,以及全量迭代都是迭代计算.对于迭代计算的性能,取决于迭代轮数和每轮迭代代价(I/O 代价和运算量).按定义 4,我们从"迭代轮数"和"每轮迭代代价"两个角度定性分析各种迭代方法的执行时间,以原始迭代执行时间为参照,具体分析如下:① 对于增量迭代的增量步,$\Delta L - L$ 虽然远小于 L,但 $\Delta L - L$ 中迭代变量尚未吸收其他变量对它的变换,因此,增量迭代的增量步迭代轮数与原始迭代的迭代轮数大体相同,$\alpha \approx \beta$(参见式(1)和(2)),但是由于数据量很小,因此增量迭代的增量步的每轮迭代代价要远小于原始迭代的每轮迭代代价,进而增量迭代的增量步执行时间远小于原始迭代时间;② 对于增量迭代的合并步,由于迭代数据仅需要收集关联数据($\Delta L/L$)对其变换,迭代轮数小于原始迭代,$\gamma < \alpha$(参见式(1)和(3)),因此增量迭代合并步迭代时间小于原始迭代时间;③ 对于全量迭代,由于 $\Delta L - L$ 的存在,全量迭代轮数和原始迭代轮数大致相同,但是由于全量数据略大于原始数据,全量迭代的每轮迭代代价略大于增量迭代的每轮迭代代价,因此全量迭代的迭代时间略大于原始迭代的迭代时间;此外还考虑一种特例,增量迭代的迭代变量初始值是原始迭代结果,当迭代算法对初始值敏感时,全量迭代性能会优于原始迭代性能.综上所述,增量迭代的迭代时间小于全量迭代的迭代时间.表 1 定性比较了 3 种迭代的迭代轮数、每轮执行代价和执行时间.

表 1 迭代的执行定性比较

迭代种类	迭代数据	迭代轮数	每轮执行代价	执行时间
原始迭代	s_0	n_0	e_0	t_0
增量迭代增量步	s_1	n_1	e_1	t_1
增量迭代合并步	s_2	n_2	e_2	t_2
全量迭代	s_3	n_3	e_3	t_3
比较	$s_1 < s_0 < s_2 = s_3$	$n_2 < n_0 \approx n_1 \approx n_3$	$e_1 < e_0 < e_2 < e_3$	$t_1 < t_2 < t_0 \approx t_3$

进一步分析知,影响增量迭代性能优势的因素有两个:① 增量数据更新原始迭代结果的程度,称为更新度.更新度影响增量迭代性能优化效果,更新度越小,优化效果越好,反之则反.按 h 函数的定义 $h(r, L)$(L 在前文公式中省略),增量数据对原始迭代结果的更新程度体现在两个方面:一是增量数据中的迭代变量对原始数据中的迭代变量的变换,即 h 函数的 r 部分;二是增量数据对原始迭代结果的变换,即 h 函数的 L 部分;② 原始数据量大小,由于增量迭代的增量步和全量迭代的迭代轮数相同,但前者是在小数据集上执行,而后者在大数据集上执行,数据量越大,每轮迭代代价就越大.全量数据由增量数据和原始数据两部分组成,当增量数据增加时,增量迭代的增量步每轮迭代代价和全量迭代每轮迭代代价都增加,因此相互抵消;当原始数据增加时,增量迭代的增量步每轮迭代代价不变,全量迭代每轮迭代代价增加,因此增量迭代的增量步优势明显,增量迭代的性能优势增大.

(2) 迭代结果一致性证明

我们利用定义 4 迭代计算的分解描述来证明增量迭代和全量迭代的结果一致性，证明思路如下：根据迭代的分解表示，由于该迭代变量的初始值是相同的，若证明两种方法迭代结果一致，只要证明任意一个迭代变量在两种方法的所有轮迭代中积累(⊕)的变换一致. 我们可以证明：① 全量迭代中某迭代变量的积累变换要大于该变量在增量迭代的原始步中的积累变换；② 当增量迭代进入合并步，该变量的积累变换逐渐变大，最终会等于全量迭代中该变量的积累变换. 由①和②可知，采用这两种方法迭代，同一迭代变量会收敛到同一状态.

在证明过程中增量和全量迭代采用一致的序列 $\{1,2,\cdots,k,\cdots,n\}$ 来表示每一轮迭代. 不失一般性，设：① 增量迭代和全量迭代同时开始；② 增量迭代的原始步和增量步同时开始(实际原始步在增量数据到来时已经结束)，且当原始步和增量步都收敛后开始合并步，先收敛的迭代变量则执行空轮；③ 增量迭代原始步收敛于 k_1 轮；④ 增量迭代增量步收敛于 k_2 轮；⑤ 增量迭代合并步开始于 k_m+1 轮，由于无法确定 k_1 和 k_2 的大小，因此令 $k_m=\max(k_1,k_2)$；⑥ k_3 轮是增量迭代合并步中任意一轮，以及对应的全量迭代中迭代轮，$k_3>k_m$.

$k_1>k_2$ 时迭代计算的示意图如图 2 所示，当 $k_2>k_1$ 时同理.

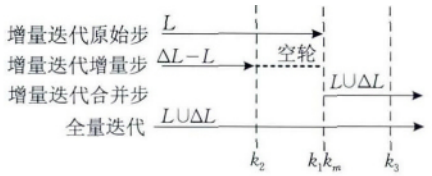

图 2　迭代轮 k_1,k_2,k_m,k_3

证明.　若⊕为代数运算，h 函数对⊕运算是分布的，则增量迭代的迭代结果等同与全量迭代的迭代结果.

因为 $h(a\oplus b)=h(a)\oplus h(b)$

因为⊕符合交换律、结合律和零律；

又因为参照定义 4 和式(7)，迭代变量 r_i 的第 k 轮迭代可以表示为式(8).

$$r_i^k=h_i^1(r_1^{k-1})\oplus h_i^2(r_2^{k-1})\oplus\cdots\oplus h_i^n(r_n^{k-1})=\sum_{x=1}^{n}\oplus h_i^x(r_x^{k-1}) \tag{8}$$

所以某一迭代变量 r_i 从第 1 轮迭代至 k 轮的值如式(9).

$$\begin{aligned}r_i^1&=\sum_{x_1=1}^{n}\oplus h_i^{x_1}(r_{x_1}^0)\\ r_i^2&=\sum_{x_2=1}^{n}\oplus h_i^{x_2}(r_{x_2}^1)=\sum_{x_2=1}^{n}\oplus h_i^{x_2}\Big(\sum_{x_1=1}^{n}\oplus h_i^{x_1}(r_{x_1}^0)\Big)\\ r_i^3&=\sum_{x_3=1}^{n}\oplus h_i^{x_3}(r_{x_3}^2)=\sum_{x_3=1}^{n}\oplus h_i^{x_3}\Big(\sum_{x_2=1}^{n}\oplus h_i^{x_2}(r_{x_2}^1)\Big)\\ &=\sum_{x_3=1}^{n}\oplus h_i^{x_3}\Big(\sum_{x_2=1}^{n}\oplus h_i^{x_2}\Big(\sum_{x_1=1}^{n}\oplus h_i^{x_1}(r_{x_1}^0)\Big)\Big)\\ &\cdots\\ r_i^k&=\sum_{x_k=1}^{n}\oplus h_i^{x_k}\Big(\sum_{x_{k-1}=1}^{n}\oplus h_i^{x_{k-1}}\Big(\cdots\sum_{x_1=1}^{n}\oplus h_i^{x_1}(r_{x_1}^0)\Big)\Big)\\ &=\mathop{\mathrm{H}}_{j=1}^{k}\Big(\sum_{x_j=1}^{n}\oplus h_i^{x_j}(r_{x_j}^0)\Big)\end{aligned} \tag{9}$$

$$r_i^k=\mathop{\mathrm{H}}_{j=1}^{k}\Big(\sum_{x_j=1}^{n}\oplus h_i^{x_j}(r_{x_j}^0)\Big) \tag{10}$$

式(10)中 x_k 表示第 k 轮任意一个迭代变量的下标. 符号 H 表示函数的复合. 第 k 轮的迭代中，任意一个迭代变量都是 k 轮积累变换的聚集.

设：在原始数据 L 中含有 m 个迭代变量，全量数据 $L\cup\Delta L$ 中含有 n 个迭代变量，则差集数据 $\Delta L-L$ 中含有 $n-m$ 个迭代变量. k_1,k_2,k_3 轮的关系见图 2，$k_m=\max(k_1,k_2)$.

所以第 k_1 轮，全量迭代下 L 中的任意迭代变量 r_p，式(11)成立.

$$r_p^{k_1}=\mathop{\mathrm{H}}_{j=1}^{k_1}\Big(\sum_{x_j=1}^{m}\oplus h_i^{x_j}(r_{x_j}^0)\Big) \tag{11}$$

因为第 k_1 轮，增量迭代原始步中 L 内与 r_p 所对应的迭代变量 r_q，式(12)成立.

$$r_q^{k_1}=\mathop{\mathrm{H}}_{j=1}^{k_1}\Big(\sum_{x_j=1}^{m}\oplus h_i^{x_j}(r_{x_j}^0)\Big) \tag{12}$$

又因为 $n>m$，r_p 每一轮积累变换大于每一轮 r_q 积累变换.

所以 $r_p^{k_1}>r_q^{k_1}$ 成立.

又因为 L 中与 ΔL 完全没有关联的迭代变量其变换始终为 0，即 $h=0$，所以 $\exists r_p^{k_1}=r_q^{k_1}$.

所以 $r_p^{k_1}\geq r_q^{k_1}$

同理，第 k_2 轮，全量迭代下 L 中的任意迭代变量 r_p 和增量迭代增量步下 $\Delta L-L$ 中与 r_p 所对应的迭代变量 r_q，由于 $n>n-m$，r_p 每一轮积累的变换大于每一轮 r_q 积累的变换.

所以 $r_p^{k_2}\geq r_q^{k_2}$

结论①，在增量迭代合并步开始之前，$r_p^{k_m}\geq r_q^{k_m}$.

所以第 k_3 轮，全量迭代下 L 中的任意迭代变量 r_p，式(13)成立.

$$r_p^{k_3}=\mathop{\mathrm{H}}_{j=1}^{k_3}\Big(\sum_{x_j=1}^{m}\oplus h_i^{x_j}(r_{x_j}^0)\Big) \tag{13}$$

因为 1 至 k_3 轮可以分解为 1 至 k_m 轮和 k_m+1 至 k_3 轮，且迭代变量仍然为 n 个.

所以式(13)可以变换为式(14)

$$r_p^{k_3}=\mathop{\mathrm{H}}_{j=1}^{k_m}\Big(\sum_{x_j=1}^{n}\oplus h_i^{x_j}(r_{x_j}^0)\Big)\oplus\mathop{\mathrm{H}}_{j=k_m+1}^{k_3}\Big(\sum_{x_j=1}^{n}\oplus h_i^{x_j}(r_{x_j}^0)\Big) \tag{14}$$

因为式(14)的 k_m+1 至 k_3 轮(后项)，迭代变量的 r_p 积累变换又来自于又来自于 L 中 m 个迭代变量和 ΔL 中 $n-m$ 个迭代变量.

所以式(14)可以变换为式(15)

$$\begin{aligned}r_p^{k_3}=&\mathop{\mathrm{H}}_{j=1}^{k_m}\Big(\sum_{x_j=1}^{n}\oplus h_i^{x_j}(r_{x_j}^0)\Big)\oplus\\&\mathop{\mathrm{H}}_{j=k_m+1}^{k_3}\Big(\sum_{x_j=1}^{m}\oplus\mathop{\mathrm{H}}_{l=1}^{k_1}\Big(\sum_{x_l=1}^{m}\oplus h_i^{x_l}(r_{x_l}^0)\Big)\Big)\oplus\\&\mathop{\mathrm{H}}_{j=k_m+1}^{k_3}\Big(\sum_{x_j=m+1}^{n}\oplus\mathop{\mathrm{H}}_{l=1}^{k_2}\Big(\sum_{x_l=m+1}^{n}\oplus h_i^{x_l}(r_{x_l}^0)\Big)\Big)\end{aligned} \tag{15}$$

式(15)中，第 1 行是前 k_m 轮迭代全量数据中迭代变量对 r_p 的积累变换；第 2 行是 k_m+1 轮至 k_3 轮迭代原始数据 L 中迭代变量对 r_p 的积累变换；第 3 行是 k_m+1 轮至 k_3 轮迭代增量数据 ΔL 中迭代变量对 r_p 的积累变换.

所以第 k_3 轮，增量迭代下合并步中与 r_p 所对应的迭代变量 r_q，若 $r_q\in L$，则式(16)成立.

$$\begin{aligned}r_q^{k_3}=&\mathop{\mathrm{H}}_{j=1}^{k_m}\Big(\sum_{x_j=1}^{m}\oplus h_i^{x_j}(r_{x_j}^0)\Big)\oplus\\&\mathop{\mathrm{H}}_{j=k_m+1}^{k_3}\Big(\sum_{x_j=1}^{m}\oplus\Big[\mathop{\mathrm{H}}_{l=1}^{k_1}\Big(\sum_{x_l=1}^{m}\oplus h_i^{x_l}(r_{x_l}^0)\Big)\oplus\\&\mathop{\mathrm{H}}_{l=1}^{k_2}\Big(\sum_{x_l=m+1}^{n}\oplus h_i^{x_l}(r_{x_l}^0)\Big)\Big]\Big)\oplus\\&\mathop{\mathrm{H}}_{j=k_m+1}^{k_3}\Big(\sum_{x_j=m+1}^{n}\oplus\Big[\mathop{\mathrm{H}}_{l=1}^{k_2}\Big(\sum_{x_l=m+1}^{n}\oplus h_i^{x_l}(r_{x_l}^0)\Big)\oplus\\&\mathop{\mathrm{H}}_{j=1}^{k_1}\Big(\sum_{x_l=1}^{m}\oplus h_i^{x_l}(r_{x_l}^0)\Big)\Big]\Big)\end{aligned} \tag{16}$$

式(16)中，第 1 行是原始步下 L 中各迭代变量对 r_q 的积累变换；第 2 行和第 3 行是合并步下 L 中各迭代变量对 r_q 的积累变换，其中各迭代变量的积累变换又来自于 L 中 m 个迭代变量(方括弧中前项)和 ΔL 中 $n-m$ 个迭代变量(方括弧中后项)；第 4 行和第 5 行是合并步下 ΔL 中各迭代变量对 r_q 的积累变换，其中各迭代变量的积累变换又来自于 ΔL 中 $n-m$ 个迭代变量(方括弧中前项)和 L 中 m 个迭代变量(方括弧中后项).

与式(15)对比，根据结论①，式(15)第 1 行小于式(16)的第 1 行；式(15)第 2 行等于式(16)第 2 行中方括弧中前项部分；式(15)第 3 行等于式(16)第 3 行中方括弧中前项部分；式(16)的第 2 和第 3 行比式(15)的第 2 和第 3 行均多出了方括弧中后项. 因此，当 k_3 足够大时，式(16)的值大于等于式(15)的值.

结论②，增量迭代合并步下，$r_q^{k_3}\geq r_p^{k_3}$，若 $r_q\in\Delta L-L$，结论②同理可得.

所以根据结论①和结论②，在合并步开始后，$r_p^{k_3}$ 和 $r_q^{k_3}$ 的差距逐渐变小. 由于迭代序列的无穷性，那么最终会有 $r_q^{k_3}\geq r_p^{k_3}$. 可见二者积累变换相同，最终收敛到同一点.

所以增量迭代的迭代结果等同于全量迭代的迭代结果. 证毕.

3.5 差集算法

根据 3.4 节分析，增量迭代对比全量迭代的性能优势在于增量迭代的增量步的每轮迭代代价小，而增量迭代的合并步迭代轮数小. 此外，增量迭代需要求取增量数据和原始数据的差集，差集算法也会引入额外的时间开销. 对于迭代数据结构较为简单的迭代算法而言，差集算法并不复杂. 例如 K-means 算法，所有的数据均为记录的形式，记录是数据点的名称和其坐标所组成的序偶. 而求取差集的过程则看作是求取在增量数据中距离当前所有中心点最小距离超过阈值的点的集合. 但是对于关系较为复杂的算法而言，求取的差集和原始数据集应不存在公共的数据和关系，即无论是数据还是关系，差集与原始数据集是相互独立的. 例如 PageRank 算法中，数据为网页，关系为网页间的连接关系，L，ΔL 和 $\Delta L-L$ 如图 3 所示. 图 3 中节点表示网页，边表示网页间的连接关系.

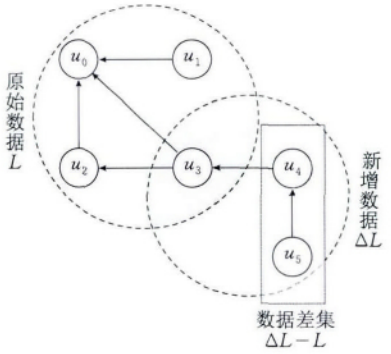

图 3 差集示意图

当数据关系较为复杂时候，差集的求取将变得复杂，在本节给出一个包含二元关系的数据集的差

集算法.在算法 1 中,使用两个 MapReduce 作业完成差集运算.

算法 1.　差集算法

输入：〈α,β〉：数据项和数据项所组成的关系,以 Key-Value 格式表示；

out：Map 任务输出的中间结果收集器；

数据项包含布尔类型的 E,Δ 和 O 属性,分别表示该数据项是否为原始数据(Exist),增量数据和 Reducer 输出结果数据(Output)

输出：$Reduce_2$ 输出结果：〈α,β〉

Map_1 阶段：

1. Init $\beta.E$ and $\beta.\Delta$ //初始化数据的 E 和 Δ 属性
2. $out.collect(\alpha,\beta)$

$Reduce_1$ 阶段：

1. $original$=false, new=false;
2. $result$=new $List()$;
3. For each β in βs　　//对于同一个 α 的所有 β
4. If $\beta.E$　　//如果 β 是原始数据
5. $original$=true
6. Else If $\beta.\Delta$ //如果 β 是增量数据
7. new=true
8. $result.add(\beta)$　　//将 β 加入结果集
9. End If
10. If $original$ and new
11. Return;
12. End If
13. End For
14. For each β in $result$　　//对于结果集中所有 β
15. $\beta.R$=true;　　//设置 β 为输出结果
16. $out.collect(\alpha,\beta)$;
17. End For

Map_2 阶段：

1. $out.collect(\beta,\alpha)$ //将 α 和 β 调换,输出给 $Reduce_2$

$Reduce_2$ 阶段：

1. $Reduce_2$=$Reduce_1$//重复执行 $Reduce_1$

算法 1 的核心思想是通过过滤数据项的方式,过滤掉数据间的关系.在 Map_1 中按照〈key,$value$〉序列正常输出,在 $Reduce_1$ 的 $values$ 中如果均出现了原始数据和增量数据标记,则表示该数据点重复出现在了两个数据集中,故将其过滤掉.而 Map_2 将〈key,$value$〉序列反向输出,从而使得 $Reduce_2$ 过滤掉关联数据 $\Delta L/L$.

3.6　模型应用

迭代算法均可满足定义 4 的分解描述,但若适用于本文提出的 DELTA 模型,则要求迭代函数 f 满足 3.4 节首段的描述;此外,根据 3.4 节迭代性能定性分析,当数据量越大,或增量数据对原始迭代结果的更新越少,增量迭代较全量迭代的性能优化效果越好.在前文形式化表达的基础上,本节举例描述常用的迭代算法 PageRank、K-means 和 Descendant Query 的 DELTA 模型实现.

(1) PageRank 算法

PageRank 是 Google 用来标识网页等级(重要性)的一种方法.设网络的链接关系存于集合中,该集合每个元素是二元组〈url_source,url_dest〉,表示在URL为 url_source 的页面中,有一个指向 url_dest 页面的超链接.初始化的迭代变量集为 R_0,每个迭代变量 u 为二元组〈url,$rank$〉,表示每个页面的URL和其 PageRank 值.若原始数据 L 中共有 n 个页面,则初始化时 R_0 内共有 n 条数据.

PageRank 的迭代函数为 fp 如式(17)所示.

$$fp(u_i,L)=\frac{1-q}{N}+q\sum_{u_j\in B(u_i)}\frac{R(u_j)}{|B(u_j)|} \tag{17}$$

在式(17)中 u 指任意 url,i,j 均为页面下标,其中 $B(u_i)$ 为指向 u_i 链接的页面的集合,$|B(u_i)|$ 为集合 $B(u_i)$ 的大小,$R(u_j)$ 为 u_j 当前的 PageRank 值,N 为页面总数,q 为阻尼常数,取值为 0.85.fp 的终止条件为 R_i 不发生变化,或者 R_i 和 R_{i+1} 的差值小于某个阈值,算法收敛,通常阈值可以取得 0.2.

PageRank 算法满足 DELTA 模型：当 u_j 有指向 u_i 的链接时 h 函数为式(18),当 u_j 没有 u_i 的链时 h 为 0;$\oplus$ 运算为加法运算,表示 PageRank 值的增长,满足代数运算要求;h 函数在“+”运算上是分布的,及每个网页对目标网页 PageRank 值的贡献之和等同于网页集对目标网页的 PageRank 值的贡献;采用算法 1 可与求取 $\Delta L-L$.

$$h_i^j(u_j^{k-1})=q\frac{R(u_j^{k-1})}{|B(u_j^{k-1})|},\ i\neq j \tag{18}$$

按式(18),R 和 ΔR 中分别包含了 L 以及 $\Delta L-L$ 内所有网页的 PageRank 值,显然 $R\cap\Delta R=\varnothing$,因为在 L 和 $\Delta L-L$ 中并没有公共网页存在.若简单的将 R 和 ΔR 合并,则丢失了关联数据集 $\Delta L/L$ 对 PageRank 值的贡献,这是因为在背景数据 ΔL 有可能包含指向 L 中页面的超链接,但是这些链接在 $\Delta L-L$ 中都被过滤掉了.R 和 ΔR 分别可以看作背景数据 L 和 $\Delta L-L$ 的局部极值,因此增量迭代的合并步可以看作是求全局极值的过程,所以合并步会很快收敛.

(2) K-means 算法

K-means 算法是典型的基于距离的聚类算法,采用距离作为相似性的评价指标,即认为两个对象

的距离越近，其相似度就越大. 设所有数据经过格式化并转化为坐标的形式，存于集合 P 中，P 内任意元素 p 是元组 $\langle id, x, y\rangle$，id 为数据的唯一标识，x 和 y 为数据经过转换后的坐标. 初始化的迭代变量为 k 个簇 $S_1^0, S_2^0, \cdots, S_k^0$，簇 S 是数据点 p 的集合，S 集有一个中心点 c. 通过迭代后求得最终的迭代结果 $\{S_1^*, S_2^*, \cdots, S_k^*\}$ 是 P 的一个划分.

K-means 的迭代函数 fk 如式(19)所示.

$$fk(S_i, P)=\{p \mid p\in P, \forall S_j\in R, dis(p, c_i)\leq dis(p, c_j)\} \tag{19}$$

在式(19)中，P 为所有数据点的集合，p 为某数据点，R 是当前迭代结果，包含了 k 个簇，第 i 个簇 S_i 的中心点为 c_i，$dis(p, c_i)$ 为 p 和 c_i 间的距离. fk 的终止条件为 S_i 不发生变化，算法收敛.

K-means 算法满足 DELTA 模型：对于某轮迭代，原属于簇 S_j^{k-1} 的数据点若变化为属于簇 S_i^k 的数据点，h 函数如式(20)所示，若簇 S_j^{k-1} 的数据点没有向簇 S_i^k 发生移动，对应的 h 为 0，当 S_i^k 确定后，c_i^k 根据 S_i^k 重新计算；$\oplus$ 运算为集合并运算，表示 S_i^k 的扩充，满足代数运算要求；h 函数在“$\cup$”运算上是分布的，及每 $k-1$ 轮每个簇移动到某簇的数据点等同于所有数据点集合移动到该簇的数据点；采用算法 1 可以求取 $\Delta L-L$.

$$h_i^j(S_j^{k-1})=\{p \mid p\in S_j^{k-1}, \forall S_j\in R, dis(p, c_i^{k-1})\leq dis(p, c_j^{k-1})\},$$

$$c_i^k=\frac{\sum_{p\in S_i^k} dis(p, c_i^{k-1})}{|S_i^k|},\ i\neq j \tag{20}$$

K-means 算法的增量迭代实现中：原始步在原始数据集 L 中初始化 k 个中心点并获得 k 个簇；定义差集为在增量数据中距离这 k 个簇的中心点最小距离均大于某一阈值的数据点；增量步在差集数据 $\Delta L-L$ 中初始化 k' 个中心点并获得 k' 个簇($k'\ll k$)；合并步之前将 L 中 k 个中心点和差集数据 $\Delta L-L$ 中 k' 个中心点视为数据点，执行一次微缩的 K-means 算法并获取 k 个新的中心点；合并步将采用这 k 个新的中心点重新迭代. R 和 ΔR 分别可以看作背景数据 L 和 $\Delta L-L$ 的预先聚类，因此增量迭代的合并步可以看作是初始点良好的聚类过程，所以合并步会很快收敛.

(3) Descendant Query 算法

Descendant Query 算法用于计算社交网络中与某人相识的所有人的列表，用以分析用户的交友信息等情况. 设社交网络信息存于集合 F 中，F 即为迭代数据，F 中每个元素为序偶 $\langle p_x, p_y\rangle$，p_x 和 p_y 均为人名，表示名字为 p_x 的人与名字为 p_y 的人是朋友关系. 迭代变量为集合 R，初始 R 仅包含一条数据，即查询对象. 通过迭代计算找出与查询对象相关的所有人.

Descendant Query 的迭代函数为 fd 如式(21)所示.

$$fd(R)=R\cup\{p_y \mid \forall\langle p_x, p_y\rangle\in F, p_x\in R\} \tag{21}$$

在式(21)中 R 是唯一的迭代变量，fd 的终止条件为 R 不发生变化.

Descendant Query 算法满足 DELTA 模型：当 R 和对 R 自身的 h 函数为式(22)；$\oplus$ 运算为集合并运算，表示 R 的扩充，满足代数运算要求；h 函数在“$\cup$”运算上是分布的，$h(R\oplus\varnothing)=h(R)\oplus h(\varnothing)=h(R)$.

$$h(R^{k-1})=\{p_y \mid \forall\langle p_x, p_y\rangle\in F, p_x\in R^{k-1}\} \tag{22}$$

Descendant Query 算法的特点导致其增量迭代和全量迭代相同，但前者有略微的优势. 全量迭代会以原始迭代结果作为初始迭代变量，在全量数据上迭代. 增量迭代实现中：原始步在原始数据集 L 查找与当前查询对象有关系的所有人，最终迭代结果集为 R；定义差集为在增量数据中与原始对象中所有人均没有关系的人，根据算法(1)可以求得 $\Delta L-L$，根据该差集定义可以知道 $\Delta R=\varnothing$，无须增量步迭代；合并步以 R 为初始迭代变量，只需在原始数据集和关联数据上迭代，最终当收敛后得到新的迭代结果. 增量迭代合并步数据量少于全量迭代数据量，因此增量迭代合并步每轮迭代代价小于全量迭代每轮迭代代价，又因为两者迭代轮数相同，因此，Descendant Query 算法增量迭代性能略优于全量迭代.

4 增量迭代计算框架

我们对 HaLoop 框架进行修改和扩展，使其支持 DELTA 模型，称为 ΔHaLoop 框架. 本小节将简述 ΔHaLoop 框架的系统架构，介绍各个部分的用途和相应的改进. 我们选择 Hadoop-0.20.2 版本实现 ΔHaLoop，在该版本仅包含 3 个模块，即 MapReduce、HDFS 和 Common，代码结构较为简单，且易于扩展.

ΔHaLoop 系统架构如图 4 所示. 图 3 中的软件模块可以分为 4 类：① Hadoop 模块，这些模块是与 HDFS 相关的文件系统模块以及任务队列模块；② HaLoop 模块，这些模块是 HaLoop 对原始 Hadoop

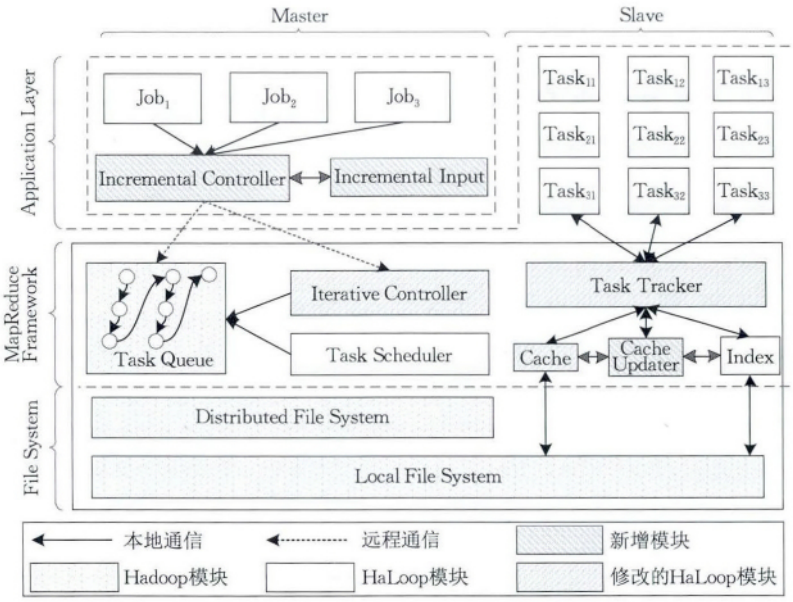

图 4　ΔHaLoop 系统架构图

的修改或者扩展部分，主要实现了迭代计算、缓存、任务调度等相关功能；③ 本文对 HaLoop 的修改模块，这些模块根据 DELTA 模型和 HaLoop 现存缺陷，对 HaLoop 模块进行了修改，主要包括 Loop Control（迭代控制）和 Task Tracker 模块；④ ΔHaLoop 新增模块，用于支持 DELTA 模型. ΔHaLoop 各个模块的简要描述如下：

(1) File System. 沿用 Hadoop 的文件系统，支持计算框架对分布式文件系统以及每个节点的本地文件系统的读写操作.

(2) Cache. HaLoop 支持迭代计算中的缓存策略，并设计了 3 种缓存，分别为 Map 输入缓存、Reduce 输入缓存、Reduce 输出缓存. 在 DELTA 模型中，缓存信息不仅有益于原始迭代，也有益于增量迭代的差集求取和结果合并过程，且在下一次增量数据到来时，上一次迭代的缓存仍有效，因此 ΔHaLoop 新增了 Cache Updater 模块用于支持在增量迭代中更新缓存，尽量延长缓存的生命周期.

(3) Task Scheduler. 沿用 HaLoop 系统任务调度功能，任务调度支持迭代缓存功能. 为了使用上一轮迭代的缓存数据，任务调度器将当前迭代轮中任务数据调度到上一轮迭代计算所在的节点，以便读取上一轮迭代所缓存的数据，从而解决静态数据的冗余传输问题和重复计算问题.

(4) Task Tracker. Task Tracker 监督并执行某个任务（Map Task 或 Reduce Task），同时与 Job Tracker 通信，时刻汇报任务的执行情况，并获取新任务. 在 HaLoop 中，Task Tracker 根据任务配置创建缓存数据和索引，同时维护缓存数据. 在 ΔHaLoop 中对 Task Tracker 进一步进行了完善，加强了 Task Tracker 缓存数据的能力，使得缓存数据的管理粒度更细致，生命周期更长，适用于 DELTA 模型的要求.

(5) Loop Control. 提供迭代控制功能，使得整个迭代算法可以重复地执行直至收敛. 在 ΔHaLoop 系统中对 HaLoop 原有的 Loop Control 功能进行扩展，支持“针对于每轮迭代输出数据特点”的迭代收敛条件.

(6) Incremental Iterative Control. 提供增量迭代支持，包括基本的编程接口、执行流程、差集算法、增量迭代流程控制等. 其中：① 增量输入模块用以区分增量数据，完成增量输入和原始数据差集的计算过程. 同时增量输入模块还对当前节点所包含的缓存信息进行扫描，如果可用缓存数据，则使用缓存数据进行求差集操作；② 增量控制模块是 ΔHaLoop 为了支持增量迭代所引入的独立模块，模块又可以分为 MapReduce 作业的创建和增量迭代的流程控制两个子模块. 前者完成增量迭代的参数初始化工作，后者则按第 3 节描述的模型和方法执行迭代算法.

理论上，本文所提出的 DELTA 模型适用于其他大数据下的计算框架，如 Spark、Naiad 等. 由于本研究没有逐一在这些框架中实现 DELTA 模型，因此本节以 Spark 为例给出实现方法的分析. 首先，我们将 DELTA 模型的实现划分为 3 个不同的任务，包括计算差集（数据筛选）、增量迭代（在差集上运行

迭代算法)和迭代合并(在全量数据上运行迭代算法).计算差集的本质为数据筛选的过程,主要用于删除增量数据和原始数据间的关系,本文所提出的差集算法基于数据连接技术,连接算法也是计算框架中常见算法.如 Spark 可以通过 Map 和 Broadcast 的组合实现 Map 端连接算法,也可以通过 Join 算子实现 Reduce 端连接算法.增量迭代和迭代合并实质上都是迭代算法的实例.现存迭代框架均可以支持迭代算法,只是性能有所差别.如 MapReduce 可以通过任务的组合,将迭代算法的迭代步骤拆分成不同的 Map 任务和 Reduce 任务,从而执行迭代算法;Spark 中可以通过使用 RDD(Resilient Distributed Dataset)的不断变化来实现迭代算法.增量迭代和迭代合并区别在于以下两点:(1)增量迭代的数据来源于计算差集步骤所得到的差集,迭代合并的数据为全量数据;(2)增量迭代的初始点为随机值或符合迭代算法要求初始值,迭代合并的初始点来源于原始的迭代结果和增量迭代的迭代结果.能够支持迭代计算的框架均可以灵活的控制迭代数据范围和迭代初始点的选择.

综上所述,DELTA 模型适用于常见的分布式计算框架,但仍然需要对框架的部分模块加以扩展.表 2 分析了常见的分布式计算框架实现 DELTA 而需要的扩展.

表 2 常见分布式计算框架实现 DELTA 模型的分析

框架名称	流程控制模块	任务调度模块	任务监控模块	数据访问模块
Yarn	重新定义主节点的流程控制和任务调度,将多个 MapReduce 任务视为一个整体,实现增量迭代流程控制功能.			修改基于的HDFS文件系统的数据读写方法,增加按数据新旧程度访问数据的功能,可以区分历史数据和新增数据.
Spark	增加增量迭代流程控制模块,以控制计算差集、增量迭代和迭代合并任务中的执行顺序切换.	根据缓存机制修改现有的任务调度,将差集计算和迭代任务分发到一个节点,后者可以利用前者的缓存.	将原有的单任务监控机制,修改成计算差集、增量迭代和迭代合并3个任务为一体的监控机制.	
PrIter				
Twister				修改数据传播模块,区分新旧数据.

5 实验分析

本节对 ΔHaLoop 框架进行测试和评价,以证明本文提出的 DELTA 模型的正确性.本节首先描述了实验的测试环境以及测试用例.随后验证了 PageRank 和 K-means 两种算法的 DELTA 模型的正确性和性能优势,并加以分析.

5.1 实验环境

我们在真实的集群环境中对 ΔHaLoop 进行测试,实验集群由同构的 13 台 PC 机所组成,具体的实验环境如表 3 所示.

表 3 实验集群环境

项	描述
集群	品牌:清华同方超翔 Z900 计算机; CPU:Inter Core i5-2300 2.80 GHz,100 MHz 外频,28 X 倍频; 存储:8 GB 内存,1 TB 硬盘
操作系统	CentOS 6.3,Linux 2.6.32
节点	主节点:1 个,运行进程有 *namenode* 和 *job tracker*; 作业节点:12 个,运行进程有 *data node*、*task tracker*
平台版本	基于 HaLoop-1.0
JVM 版本	Java 7-updated-u9
网络带宽	1000 Mbps
开发 IDE	Eclipse 4.2

由于 ΔHaLoop 基于 HaLoop 实现,而 HaLoop 是基于 Hadoop-0.20.2 版本实现.所以在 ΔHaLoop 运行过程中,其运行参数与 Hadoop-0.20.2 版本是一致的.表 4 列出部分参数.

表 4 ΔHaLoop 运行的重要参数

参数	值	说明
HDFS block size	64 MB	HDFS 中文件块的大小
Hadoop internal memory	4 GB	Hadoop 运行过程中可用内存
Number of map slot	2	单一节点 Map 的任务最大数量
Number of reduce slot	2	单一节点 Reduce 的任务最大数量
Heartbeat interval	3 s	Hadoop 心跳时间

5.2 测试用例

为简化描述,本节使用 $S_i(i\in\{1,2,3\})$ 表示数据集的大小,$U_j(j\in\{1,2,3,4\})$ 表示增量数据对原始迭代结果的更新度,则 S_i 和 U_j 可以组合成多组实验数据.我们用 $T_1(S_iU_j)$ 表示原始迭代时间,$T_2(S_iU_j)$ 表示增量迭代时间,$T_3(S_iU_j)$ 表示全量迭代时间.为了探究增量迭代的性能,将使用 $[T_3(S_iU_j)-T_2(S_iU_j)]/T_3(S_iU_j)$ 作为增量迭代优化程度的度量,记作函数 $\omega(S_iU_j)$.

PageRank 测试数据集采用文献[17]的数据集.数据集中节点的入度服从对数正态分布($\mu=-0.5$,$\sigma=2.3$),其中参数来自于真实图数据集①.设增量数据对原始数据迭代结果的更新程度为 U.在增量数据生成过程中,增量数据中网页链接原始数据中网页的概率为 U,链接增量数据中网页的概率为 $1-U$.PageRank 测试数据的相关信息如表 5 所示.

① http://snap.stanford.edu/data

表 5　PageRank 测试数据集

名称	原始数据量/GB	原始节点数/千万	更新度	新增数据量/MB	新增节点数/万
S_1U_3	6.70	4	0.6	59.48	60
S_2U_3	13.50	8	0.6	59.48	60
S_3U_3	27.25	16	0.6	59.48	60
S_3U_1	27.25	16	0	59.48	60
S_3U_2	27.25	16	0.4	59.48	60
S_3U_4	27.25	16	0.8	59.48	60

K-means 数据集采用 DBPedia 数据集①. 从 DBPedia 数据集中抽取其中经度和维度字段组成了 K-means 基础数据集(20 MB)，并通过随机抽取的方式，将其分解成 K-means 种子数据集(18 MB)和 K-means 增量数据集(2 MB)，以上两个数据分别用于生成 K-means 测试中的原始数据和增量数据. 为适应大数据，在数据生成过程中，遵循文献[29]中数据模拟方法，在基础数据集点周围生成多倍的额外数据点来扩展数据集，其扩展倍数根据 K-means 测试用例选取. 设增量数据对原始数据迭代结果的更新程度为 U. 由于 K-means 将数据点分为 k 簇，新增数据若能够归入到原始 k 簇，则更新度小，新增数据全部无法归入到原始 k 簇而需要对原始簇改变，则更新度大. 数据生成时，设原始数据的值域为$[\alpha,\beta]$，在增量数据生成过程中，增量数据中的数据点值域在$[\alpha,\beta]$的概率为 U，在$[\alpha+10^2,\beta+10^2]$概率为 $1-U$. K-means 测试数据的相关信息如表 6 所示.

表 6　K-means 数据集说明

名称	原始数据量/GB	原始节点数/千万	增量数据量/MB	增量节点数/万	更新度 U
S_1U_1	4.13	12	56.78	800	0
S_2U_1	8.26	24	56.78	800	0
S_3U_1	16.51	48	56.78	800	0
S_2U_2	8.26	24	56.78	800	0.2
S_2U_3	8.26	24	56.78	800	0.6
S_2U_4	8.26	24	56.78	800	1

需要指出的是，当 $U=0$ 时，理论上，增量数据不会对原始聚类产生影响时，可以认为增量数据自成 k'簇，由于全量迭代的初始点是原始数据的 k 簇，因此理论上在全量迭代中这 k 个簇不需要变换，只需要找出增量部分的 k'簇即可. 但实际上由于 K-means 算法始终要保持聚类个数为 k，即使 $U=0$ 时全量迭代也需要将 $k+k'$个簇聚成 k 个簇. 因此，尽管 $U=0$，增量数据对原始迭代结果也会产生更新. 在本实验用例中，我们取 $k=12$，$k'=2$.

此外，在 K-means 算法执行过程中，本文使用“迭代变量变化小于某阈值”这一条件作为迭代终止条件，且阈值设为极小的 10^{-6}，确保迭代结果的精确性. 并且为了对比各用例的迭代结果，各用例初始化的 k 个中心点是相同的，确保迭代结果能够合理地进行比较.

5.3　PageRank 算法

根据 3.4 节的理论证明，无论是通过增量迭代还是全量迭代，最终的结果数据是一致的. 由于 PageRank 迭代算法的运行结果本身不具有唯一性，因此即使同样数据下多次计算 PageRank，其计算结果也会存在微小的差异. 令每个网页误差为每次迭代 PageRank 值的差距，即 $|rank-rank'|/rank$，然后以所有网页的误差均值作为误差评价标准. 分析实验结果知，多次全量迭代结果的平均误差在[0.101,0.201]区间内，而增量迭代和全量迭代之间的误差在[0.117,0.209]区间内. 为进一步证明迭代结果的一致性，我们参考文献[30]，计算增量迭代结果和全量迭代结果的均方根误差. 均方根误差是常用的统计分析方法，用于分析预测值和真实值间的差距，其值越接近于 0，差异越小，其计算方法如式(23)所示，S 和 T 分别表示增量迭代结果和全量迭代结果，s_i和 t_i分别表示增量迭代结果和全量迭代结果中编号为 i 的页面的 PageRank 值，n 为页面数量. 计算得增量迭代和全量迭代结果的均方根误差平均值为 4.08×10^{-10}，该值小于文献[30]的实验结果 1.24×10^{-9}. 基于误差对比和均方根误差分析，我们可以认为，对于 PageRank 算法增量迭代和全量迭代的结果是一致的.

$$RMSE(S,T)=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(s_i-t_i)^2} \tag{23}$$

我们首先验证了本文提出的基于 MapReduce 的差集算法，用于求取原始数据和增量数据的差集. 表 7 给出了 3 组实验数据的增量数据集的大小，所求的差集大小以及差集占增量数据的比率.

表 7　差集求取结果

	增量数据大小/MB	差集大小/MB	差集所占比例
S_3U_1	59.48	59.48	1
S_3U_2	59.48	35.395	0.59507
S_3U_4	59.48	11.98	0.20141

为了方便表述，差集所占增量数据集的比率使用函数 p 进行表述. 根据 5.2 节的数据生成算法，则理论上 $p(S_3U_1)=1$，$p(S_3U_2)=0.6$，$p(S_3U_4)=0.2$. 在表 5 中的差集比例与更新度定义一致. 同时按大数定律，随着数据集的增大，$p(S_iU_j)$的值也越

① http://dbpedia.org

接近理论值. 实际值和理论值的极小差异证明了差集算法的正确性.

我们测试了相同更新度,不同数据量下的 3 种迭代计算的性能. 选用数据集为 S_1U_3、S_2U_3 和 S_3U_3,实验结果如图 5 所示.

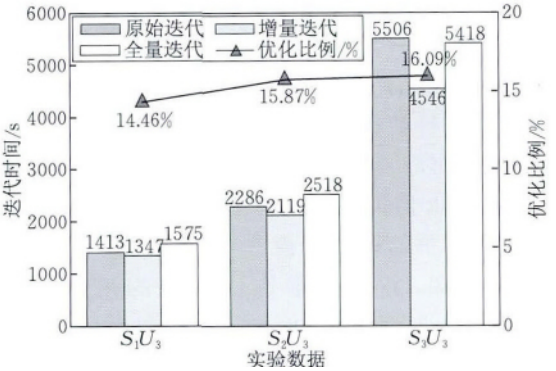

图 5 PageRank 在不同数据量的数据集下实验结果

由图 5 可知,$T_i(S_xU_3)<T_i(S_yU_3)(x,y,i\in\{1,2,3\}$ 且 $x<y)$,即同种迭代方法的迭代时间随迭代数据集大小单调递增. 文献[11]指出大部分数据集上的 PageRank 算法都会在 10 轮内收敛,但每轮的迭代代价不同,数据量不会影响迭代轮数,但会影响每轮迭代时间. 此外,$T_2(S_iU_3)<T_1(S_iU_3)<T_3(S_iU_3)$,增量迭代的性能优于原始迭代,而前两者均优于全量迭代.

$T_2(S_iU_3)<T_3(S_iU_3)$ 是我们研究的重点,增量迭代对比全量迭代的性能优势在于:增量迭代增量步的每轮迭代代价小于全量迭代的每轮迭代代价,而增量迭代的合并步迭代轮数小于全量迭代. 对于前者,全量迭代和增量迭代增量步的迭代轮数相同,当原始数据量增加时,全量迭代每轮迭代代价增加,而因为增量数据未发生变化,增量迭代的增量步每轮迭代代价不变,性能优化效果增加. 对于后者,当数据量增加而数据特征不变时,增量迭代和全量迭代的合并步的迭代轮数差距不变,即使每轮迭代代价增加,性能优化效果不变. 由于 $|\Delta L|\ll|L|$,因此 $\Delta L-L$ 是一个更小的数据,增量迭代增量步的代价在整个增量迭代代价中所占比例很小,因此,增量步的优化效果很小. 综上所述,在数据特征不变的情况下增加数据量,增量迭代的优化效果会有较小的提升. 图 5 中优化比例折线清晰地展示了上述特征. $\omega(S_1U_3)<\omega(S_2U_3)<\omega(S_3U_3)$,折线斜率很小,略有上升.

随后,我们测试了相同数据量,不同更新度下的 3 种迭代计算的性能. 选用数据集为 S_3U_1 至 S_3U_4,实验结果如图 6 所示.

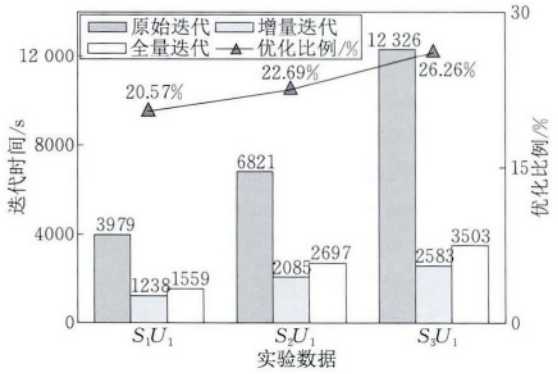

图 6 PageRank 在不同更新度的数据集下实验结果

由图 6 可以看出,对于全量迭代,无论更新度如何,$T_3(S_3U_i)(i\in\{1,2,3,4\})$ 均相差不大;对于增量迭代,除了 $T_2(S_3U_1)$ 很小,其他均相差不大;对于性能优化比例,$\omega(S_3U_1)>\omega(S_3U_2)>\omega(S_3U_3)>\omega(S_3U_4)$,但 $\omega(S_3U_1)$ 优势明显,其他数据集下增量迭代的优化效果随更新度增大而略微减小,折线下降缓慢.

增量迭代对比全量迭代的性能优势在于增量迭代的合并步迭代轮数小于全量迭代. S_3U_1 的更新度为零,因此增量迭代的合并步仅执行了 1 轮,而全量迭代执行了 7 轮,因此增量迭代明显优于原始迭代;但由于 $|\Delta L|\ll|L|$,无论更新度为多少,关联数据集都很小,不足以影响增量迭代合并步的轮数,因此,S_3U_2,S_3U_3 和 S_3U_4 下增量迭代的合并步迭代轮数均为 4 轮,因此从合并步迭代轮数角度优化比例不变. $\omega(S_3U_2)>\omega(S_3U_3)>\omega(S_3U_4)$ 的原因是:当关联数据集增加,增量迭代的合并步的每一轮参与计算的数据增加,因此每轮迭代代价增加,增量迭代执行时间增加,因此优化比例减少.

综上所述,通过多组数据实验,我们可以认为 PageRank 算法增量迭代较全量迭代的性能优化比例达到 15%.

5.4 *K*-means 算法

首先,我们验证 K-means 算法增量迭代结果和全量迭代结果的一致性. 实验证明,两者的迭代初始点相同,迭代后的 k 个簇内包含的数据也相同. 进一步地,我们判断增量迭代后 k 个簇的中心点与全量迭代 k 个簇的中心点是否重合. 我们对 k 个中心点进行比较,并计算对应簇的中心点间的欧氏距离. 经过分析,全量迭代聚类结果和增量迭代聚类结果中心点距离均值为 2.135×10^{-7},该值小于迭代终止阈值 10^{-6},即误差的控制范围. 因此,可以认

为K-means算法增量迭代和全量迭代的结果是一致的.

我们测试了相同更新度，不同数据量下的3种迭代计算的性能. 选用数据集为S_1U_1、S_2U_1和S_3U_1，实验结果如图7所示.

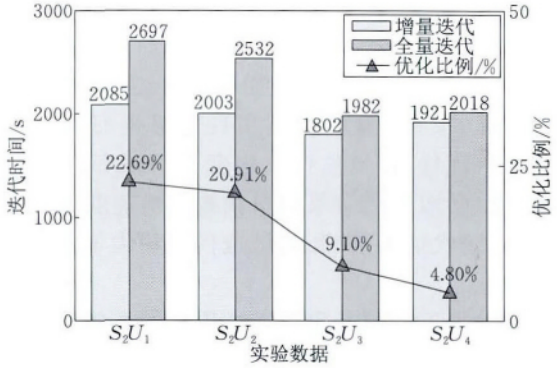

图7　K-means在不同数据量的数据集下实验结果

可知对于相同更新度且不同数据量的数据集，K-means算法(图7)和PageRank算法(图5)在3种迭代方法下的迭代时间规律类似：同种迭代方法的迭代时间随迭代数据集大小单调递增；增量迭代的性能优于全量迭代；在数据特征不变的情况下数据量增加，增量迭代的优化效果有所提升. 但K-means算法全量迭代执行时间平均为原始迭代的35%. 增量迭代较全量迭代的优化效果明显提高，在大数据量下性能优化达22%，且当原始数据增加时，增量迭代性能有所提升. 这是因为K-means算法迭代轮数对初始点的选择非常敏感，良好的初始点选择可以极大地减少迭代轮数：(1)原始迭代的初始中心点是随机选取，因此迭代轮数很大；(2)对于增量迭代的增量步，采用较少的随机初始点进行迭代，且差集数据很小，因此迭代代价较小；对于增量迭代的合并步，由于初始点来自于原始数据的k个中心点和差集数据的k'个中心点(求$k+k'$个点的k个中心点)，因此中心点选择良好，迭代轮数很小；(3)对于全量数据，采用原始数据k个中心点作为初始点，初始点选择较好，迭代轮数少于原始迭代轮数；(4)当数据量增加时，增量迭代合并步和全量迭代的迭代轮数差距更加明显，优化效果越好；(5)增量迭代增量步每轮迭代代价的优势对性能的影响不明显，而迭代轮数起决定性影响，迭代轮数则与初始点的选择相关.

随后，我们测试了相同数据量，不同更新度下的3种迭代计算的性能. 选用数据集为S_2U_1至S_2U_4，实验结果如图8所示.

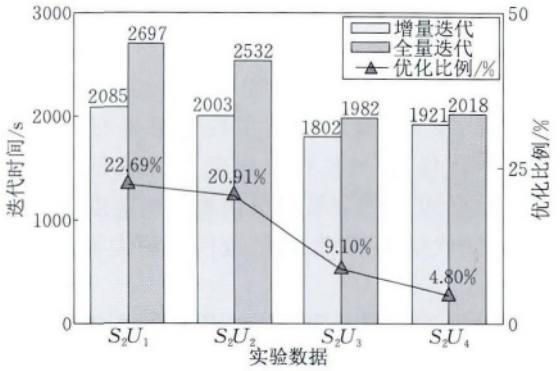

图8　K-means在不同更新度的数据集下实验结果

图8的规律和图6基本一致：$\omega(S_2U_1)>\omega(S_2U_2)>\omega(S_2U_3)>\omega(S_2U_4)$，但$\omega(S_2U_1)$优势明显，其他数据集下增量迭代的优化效果随更新度增大而减小.

首先，我们分析$S_2U_1(U=0)$的情况. 由于全量迭代的初始点是原始迭代结果，理论上其迭代次数也受初始点是否良好的影响. 当$U=0$时是增量迭代最佳情况，却是全量迭代的最坏情况. 前者会在增量步将增量数据(差集数据)聚成簇，并在合并步调整原始聚类结果，减少簇个数；后者则要将原始聚类结果不断变化以适应独立成簇的增量数据. 后者的迭代轮数大于前者的迭代轮数，因此，增量迭代性能优化效果很好. 其次，我们分析$S_2U_4(U=1)$，此时因为差集数据为空，增量迭代已经退化为全量迭代，两者的性能应该相差无几，图中优化4.8%的结果具有偶然性.

综上所述，通过多组数据实验，我们可以认为K-means算法增量迭代较全量迭代的性能优化比例平均为22%.

6　结论和进一步工作

本文提出了一种基于新增数据的增量迭代计算模型，又称DELTA(Delta data based incrEmentaL iTerAtive computing)模型，利用原始迭代结果和增量数据求解新的迭代结果，避免在数据全集上进行全量迭代. 本文将增量迭代分为3个步骤，即“原始步(已完成)”、“增量步”和“合并步”. 本文定义了DELTA模型、描述了其计算步骤，证明了DELTA模型的性能优势和结果正确性，同时列举了PageRank、K-means和Descendant Query算法在DELTA模型下的例子. 本文还描述了DELTA框架的实现，重点阐述了迭代框架ΔHaLoop中对HaLoop所作出

的改进.

本文对 DELTA 模型进行了评价,分别从功能正确性和性能优势两个角度进行阐述,最终得出如下结论:DELTA 模型及其实现能够很好的适应大数据的迭代分析,当数据发生变化后,DELTA 模型能够利用已知迭代结果和增量数据快速获得新的迭代结果,迭代结果准确.增量迭代明显优于全量迭代性能.

进一步工作包括研究通过更多的迭代算法来验证 DELTA 模型,以及讨论适用于增量迭代的迭代变量初始化算法.此外,目前的增量迭代中还包括一些数据预处理操作,如差集算法以及作业调度时间,我们拟通过提高并行性等方式减少这些操作的时间,探讨是否存在"增量迭代和原始迭代的执行时间之和能够接近全量迭代的执行时间"的可能性.

参 考 文 献

[1] Page L, Sergey B, Motwani R, Winograd T. The PageRank citation ranking: Bringing order to the web. Stanford InfoLab, USA: Technical Report 422, 1999

[2] Kiri W, Claire C, Seth R, Stefan S. Constrained *K*-means clustering with background knowledge//Proceedings of the 18th International Conference on Machine Learning. Williamstown, USA, 2001: 577-584

[3] Breese J S, Heckerman D, Kadie C. Empirical analysis of predictive algorithms for collaborative filtering//Proceedings of the Uncertainty in Artificial Intelligence. San Francisco, USA, 1998: 43-52

[4] Mingmin C. Bruzzone L. A novel transductive SVM for semisupervised classification of remote sensing images. IEEE Transactions on Geoscience and Remote Sensing, 2006, 44(11): 3363-3373

[5] Meng Xiao-Feng, Ci Xiang. Big data management: Concepts, techniques and challenges. Journal of Computer Research and Development, 2013, 50(1): 146-169(in Chinese)
(孟小峰, 慈祥. 大数据管理:概念、技术与挑战. 计算机研究与发展, 2013, 50(1): 146-169)

[6] Wang Shan, Wang Hui-Ju, Qin Xiong-Pai, Zhou Xuan. Architecting big data: Challenges, studies and forecasts. Chinese Journal of Computers, 2011, 34(10): 1741-1752(in Chinese)
(王珊, 王会举, 覃雄派, 周烜. 架构大数据:挑战、现状与展望. 计算机学报, 2011, 34(10): 1741-1752)

[7] Dean J, Sanjay G. MapReduce: Simplified data processing on large clusters//Proceedings of the 6th Symposium on Operating Systems Design and Implementation. Berkeley, USA, 2004: 137-149

[8] Bu Yingyi, Howe B, Balazinska M, Ernst M D. The HaLoop approach to large-scale iterative data analysis. The International Journal on Very Large Data Bases, 2012, 21(2): 169-190

[9] Daniel P, Frank D. Large-scale incremental processing using distributed transactions and notifications//Proceedings of the 9th Symposium on Operating Systems Design and Implementation. Berkeley, USA, 2010: 137-149

[10] Bhatotia P, Wieder A, Rodrigues R, et al. Incoop: MapReduce for incremental computations//Proceedings of the 2nd ACM Symposium on Cloud Computing. Cascais, Portugal, 2011: 7

[11] Kleinberg J M. Authoritative sources in a hyperlinked environment//Proceedings of the 1998 9th Annual ACM SIAM Symposium on Discrete Algorithms. San Francisco, USA, 1998: 668-677

[12] Krogh A, Vedelsby J. Neural network ensembles, cross validation, and active learning//Proceedings of the NIPS94-Neural Information Processing Systems: Natural and Synthetic. Cambridge, USA, 1995: 231-238

[13] Ryder B G, Marlowe T J, Paull M C. Conditions for incremental iteration: Examples and counterexamples. Science of Computer Programming, 1988, 11(1): 1-15

[14] Burke M. An interval-based approach to exhaustive and incremental interprocedural data-flow analysis. ACM Transactions on Programming Languages and Systems, 1990, 12(3): 341-95

[15] Fleischmann E, Forler C, Gorski M, Lucks S. TWISTER—A framework for secure and fast hash functions. International Journal of Applied Cryptography, 2010, 2(1): 68-81

[16] Elnikety E, Elsayed T, Ramadan H E. iHadoop: Asynchronous iterations for MapReduce//Proceedings of the IEEE 3rd International Conference on Cloud Computing Technology and Science. Los Alamitos, USA, 2011: 81-90

[17] Zhang Y, Gao Q, Gao L, Wang C. PrIter: A distributed framework for prioritizing iterative computations. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(9): 1884-1893

[18] Yanfeng Z, Qinxin G, Lixin G, Cuirong W. iMapReduce: A distributed computing framework for iterative computation//Proceedings of the IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum. Piscataway, USA, 2011: 1112-1121

[19] Mihaylov S R, Ives Z G, Guha S. REX: Recursive, delta-based data-centric computation. Proceedings of the VLDB Endowment, 2012, 5(11): 1280-1291

[20] Matei Z, Mosharaf C, Tathagata D, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing//Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. Berkeley, USA, 2012: 2-2

[21] Murray D G, McSherry F, Isaacs R, et al. Naiad: A timely dataflow system//Proceedings of the 24th ACM Symposium on Operating Systems Principles. Farmington, USA, 2013: 439-455

[22] Yi L, Shiyong L, Fotouhi F, et al. Incremental genetic *K*-means algorithm and its application in gene expression data analysis. BMC Bioinformatics, 2004, 5(1): 172

[23] Inderjit D, Yuqiang G, Brian K. A unified view of kernel *K*-means, spectral clustering and graph cuts. University of Texas at Austin, USA: Technique Report TR-04-25, 2004

[24] Pham D. T, Dimov S S, Nguyen C D. An incremental *K*-means algorithm. Journal of Mechanical Engineering Science, 2004, 218(7): 783-795

[25] Elnekave S, Last M, Maimon O. Incremental clustering of mobile objects//Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop. Piscataway, USA, 2007: 585-592

[26] Hamza H, Belaid Y, Belaid A, Chaudhuri B B. Incremental classification of invoice documents//Proceedings of the 19th International Conference on Pattern Recognition. Piscataway, USA, 2008: 1-4

[27] Khy S, Ishikawa Y, Kitagawa H. A novelty-based clustering method for on-line documents. World Wide Web, 2008, 11(1): 1-37

[28] Chakraborty S, Nagwani N K. Analysis and study of incremental *K*-means clustering algorithm//Proceedings of the International Conference on High Performance Architecture and Grid Computing. Chandigarh, India, 2011: 338-341

[29] Brin S, Page L. The anatomy of a large-scale hypertextual Web search engine//Proceedings of the 7th International World Wide Web Conference. Netherlands, 1998: 107-117

[30] Abdullah I B. Incremental PageRank for Twitter Data Using Hadoop [M. S. dissertation]. University of Edinburgh, School of Informatics, 2010

SONG Jie, born in 1980, Ph. D., associate professor. His research interests include big data storage and management, iterative computing.

GUO Chao-Peng, born in 1990, M. S. candidate. His current research focuses on data iterative computing.

ZHANG Yi-Chuan, born in 1981, Ph. D., lecturer. His current research focuses on framework and algorithm of iterative computing.

ZHANG Yan-Feng, born in 1981, Ph. D., associate professor. His research focuses on iterative computing.

YU Ge, born in 1962, Ph. D., professor. His research interest is database theory.

Background

Big data era brings new challenges to data analysis. The traditional data analysis technology is difficult to analyze large data accurately and efficiently. In the cloud computing environment, distributed file system and MapReduce programming model is a new technology to deal with large data analysis. Search algorithm, PageRank algorithm, EM (Expectation-Maximization) algorithm, *K*-means algorithm, Collaborative Filtering algorithm and intelligent optimization algorithm are iterative type analysis algorithm commonly used in large data sets. These algorithms have been applied to such as social network analysis, high performance computing, recommendation system, search engine, pattern recognition and other fields. So, the iterative calculation is the focus research in the field of cloud computing and data analyze.

At present, there have been some studies on the optimization for the iterative algorithm. Such as support for distributed computing environment iterative model, computing framework and task allocation and data layout optimization method. Incremental iteration is a necessary means to optimize iterative algorithm. Using previous iterative results and new data obtained new iteration results in a shorter time and less resource consumption. Through the new incremental data set and the known iterative results to complete the incremental iterative calculation, it can greatly improve the efficiency of iterative computation. This paper proposed an incremental fixed point iteration model and framework; design the key algorithm and application in PageRank algorithm; Extends HaLoop to support the incremental iterative function. Experiment shows the correctness and efficiency. This research, which has been proved to be efficient on the incremental iteration, has certain theoretical and practical value.

This work is mainly supported by the National Research Foundation for the Doctoral Program of Higher Education of China (No. 20130042120006). The project adopt cloud computing techniques and iterative computing methods to provide high reliability and high performance iterative computing framework. This paper can provide the theoretical basis and technical support for the research of the project development and deeply. This paper is also supported by the National Natural Science Foundation of China (Nos. 61433008, 61202088, 61272179, 61173028). Our group has working in the area of iterative computing for 3 years and published many paper.