云计算环境中基于访问量和依赖性评价的数据分配算法

孙熙领[1] 陈 超[1] 丁治明[1] 许佳捷[1] 袁 栋[2]
(中国科学院软件研究所 北京 100190)[1] (澳大利亚斯文本科技大学信息传播技术学院 墨尔本 3122)[2]

摘 要 大量的大规模密集型数据需要存储在多个数据存储中心,而应用越来越广泛的云计算环境很好地解决了大规模密集型数据在分配中遇到的规模性问题。但是,云计算环境中多数据存储中心的数据分配会带来数据存储中心之间数据量的传输,从而导致数据访问效率低下。同时,单位时间上数据访问量的不平衡性会引起数据存储中心的访问瓶颈。以大规模密集型数据中的数据流为建模对象,提出了一种数据分配算法,它在保证数据存储中心负载平衡的基础上兼顾了密集型数据之间的依赖性。实验表明,相比于同类的数据分配算法,所提算法具有更好的综合表现,特别是在保证数据存储中心的负载平衡方面,效果突出。

关键词 数据分配,云计算,大规模密集型数据,负载平衡,数据依赖
中图法分类号 TP31 **文献标识码** A

Data Allocation Algorithm Based on Visit Capacity and Dependency Evaluation in Cloud

SUN Xi-ling[1] CHEN Chao[1] DING Zhi-ming[1] XU Jia-jie[1] YUAN Dong[2]
(Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)[1]
(Swinburne University of Technology, Swinburne University of Technology, Melbourne 3122, Australia)[2]

Abstract A huge number of large-scale intensive data have to be stored in distributed data centers. Nowadays, under the cloud environment, large-scale data storage can be better supported. However, a challenging issue is that the transmission of intensive data between cloud data centers may cause low efficiency of data access. Also, the bottleneck of access on data center may be derived from the imbalanced capacity of data visit in unit interval. We first proposed a model based on data flow between large-scale intensive data. Afterwards, a data allocation algorithm was presented to guarantee the load balance of data centers while considering dependencies between intensive data. Extensive experiments confirm that our solution has better performances than conventional approaches particularly in load balance.

Keywords Data allocation, Cloud computing, Large-scale intensive data, Load balance, Data dependency

1 引言

随着计算机技术和互联网技术的不断发展,分布式系统、网格计算和云计算等技术先后涌现出来。由于其良好的性能,近年来出现的云计算得到了越来越广泛的应用。Google公司的云计算平台[17]、IBM的"蓝云"计算平台、Amazon的弹性计算云等在各自的领域都获得了极大的成功[1,2,21]。云计算能够处理更大规模的数据,有着更为合理的价格,并能更为高效地完成计算需求。云计算中的计算对象多为大规模密集型的数据,如银行、交通统计、多媒体和科学工作流等应用都有着PB级别的数据[3]。面对如此大规模的密集型数据,合理而有效的数据分配机制是提高固有资源利用率和数据处理能力、避免产生系统瓶颈的前提条件[4]。

诸如科学工作流、数据流中的大规模密集型数据在应用于云计算环境时可以很好地解决数据的规模问题,同时满足用户的弹性计算需求[5]。但是大规模密集型数据分配于云计算平台中也存在着许多挑战,如何利用数据及云计算自身的特点是其中之一。不同于传统的网格计算和分布式系统中地域上离散的服务器[22],云计算中采取的是大规模集群系统甚至是多台服务器的廉价虚拟化集群。这种底层结构上的不同决定了传统以减少通信代价为主要目标的数据分配算法不能很好地利用云计算环境的特点[6]。云计算拥有着容量更大的廉价服务器,硬件不再适合作为数据分配算法考虑的主要限制因素。同时,大规模密集型数据之间存在着数据依赖性,如科学工作流中不同的任务使用相同的数据[7]、数据流中不同的数据之间存在着的传输数据量[16]。数据分配过程中保证数据之间的依赖性,可以降低数据使用中的传输代价和时间开销[3]。同时,数据在单位时间内的访问量是恒定的,但是在各个单位时间之间的访问量一般是波动变化的。如果不考虑数据访问量的因素,数据分配的结果可能存在数据存储中心访问量的不均衡性,造成系统瓶颈。所以,云计算环境中数据分配问题主要的挑战有以下几点:首先是如何改变数据分配

到稿日期:2011-06-16 返修日期:2011-10-13 本文受中科院知识创新项目(KGCX2-YW-174)资助。
孙熙领 硕士生,主要研究方向为云数据管理、数据库,E-mail:hixiaoxi@gmail.com;**陈 超** 硕士生,主要研究方向为云数据管理、移动计算;**丁治明** 博士,研究员,主要研究方向为数据库、移动计算、信息检索;**许佳捷** 博士,助理研究员,主要研究方向为数据库、工作流;**袁 栋** 博士生,主要研究方向为工作流、网格计算、云计算。

算法的主要评价目标，以更好地利用云计算环境的特点；然后是如何利用数据单位时间上访问量的不平衡取得数据存储中心的负载平衡；最后是如何利用大规模密集型数据具有依赖性的特点来提高数据的访问效率。

本文通过分析云计算环境下大规模密集型数据的特点，以数据流作为分析和建模对象，提出了数据分配算法 DOVE (Data allocation algorithm based On Visit capacity and dependency Evaluation in cloud)。针对单位时间上数据访问量不平衡的特点，DOVE 使用数据存储中心总体时间利用率指标 ET、系统单位时间均衡指标 ES、数据存储中心的最大波峰值[19]和传输数据量指标 EDT 等作为评价指标，并结合动态调整反馈方式，来保证数据存储中心在总体时间、单位时间上的负载平衡和较低的数据传输量。DOVE 算法在进行数据分配时，通过减少数据存储中心之间传输数据量的方式来提高密集型数据流的执行效率。相比于以往的工作，我们的贡献是，在已知数据单位时间访问量的基础上，保证了数据存储中心的负载平衡，同时兼顾了数据之间的依赖性，减少了数据存储中心之间的传输数据量，提高了数据的访问效率。

本文第 2 节介绍相关工作；第 3 节是建模和问题定义；第 4 节为算法设计；第 5 节通过实验将 DOVE 算法和其他数据分配算法进行对比；最后是结论及未来的工作。

2 相关工作

传统网格计算或者分布式系统有很多成熟而有创新性的数据分配算法。Stork[8]是网格计算中的一个不错的数据分配算法，它改变了网格计算传统方法中数据分配依靠手工或简单的脚本解决数据分配的效率和可靠性的问题，实现了一个自动化的并且有着容错性的数据分配算法。Reactive nuca[9]是分布式缓存中的数据分配和复制策略，它根据缓存中的数据访问方式将数据分类，然后将不同的类别放置于合适的位置。在操作系统的支持下，Reactive nuca 可以实现数据的智能分配、数据迁移和数据复制。为了适应大规模并行磁盘系统上访问数据的高频性，Xie 提出了 SEA[10]算法。SEA 是一种节省能量和提高并行反应速度的数据分配算法，能够在 RAID 磁盘阵列的系统中，在提供快速分配的同时显著地节省能量。分布式系统的多重计算资源通常需要共享数据，Cope 等人提出了具有高鲁棒性的数据分配算法，同时该算法能够满足分布式系统对时间的要求[11]。混合方法 RSEDP[12]是有效地实现了大型网络存储系统中所要求的高可靠性、高扩展性、高效率的数据分配算法。在大型的异构网络存储系统中，相同的副本被分配到不同的网络设备中，实现了高冗余性和容错性。同时，RSEDP 算法根据数据的权重和规模在不同的网络节点之间分配数据。然而，上述数据分配算法并没有很好地利用云计算环境的结构特点，也没有利用大规模密集型数据之间存在的依赖性来优化数据分配算法。同时，上述数据分配算法并没有关注数据单位时间上访问量的变化来保证数据存储中心在总体时间和单位时间上的负载平衡。

随着云计算的快速发展，云计算环境中许多新的数据管理系统涌现出来。这些数据管理系统处在云计算平台的底层，其类型属于 Infrastructure as a Service[18]，例如 Google 公司的分布式文件管理系统 Google File System[13]、Google File System 及 MapReduce 的开源实现 Hadoop[14]软件和 Amazon 公司的弹性云 EC2[15]。这些数据管理系统都隐藏了基础结构来方便用户，使得数据的存储独立于用户。Google File System 主要应用于 Web 搜索，这些应用与大规模密集型数据如科学工作流、数据流，有着很大的不同。Hadoop 的文件管理系统会自动地将存储数据分割成数据块，并将这些数据块随机分配到数据存储中心。工业界中这些通用性较强的数据存储系统在存储大规模密集型数据的时候不能很好地利用数据的特点来保证数据之间的依赖性和存储节点的负载平衡。

Yuan 提出了一种针对科学工作流的数据分配策略[5]，其利用科学工作流中不同任务使用相同数据的依赖性来降低数据的移动次数，从而保证工作流的执行效率。这种方法首先构建依赖度矩阵，然后利用 BEA 算法划分依赖度矩阵，并为所得的划分分配存放位置。Yuan 的算法有效地利用了大规模密集型数据之间的一种依赖性，减少了数据在被访问过程中的移动次数，但它没有考虑数据之间的另一种依赖性传输数据量。文献[20]对 Yuan 的算法做了改进，即加入了传输开销的因素，利用遗传算法取得了不错的效果，但是它没有考虑单位时间上数据访问量的不平衡性。数据访问量在单位时间内是恒定的，但是在各个单位时间之间一般是变化的，如果不予考虑，则有可能造成数据存储中心的超额访问，从而造成数据访问的瓶颈。

综上所述，虽然网格计算和分布式系统中存在着很多成熟、高效的数据分配算法，但是这些数据分配算法并没有很好地利用云计算环境和大规模密集型数据的特点。同时，关于云计算环境下数据分配的研究大多没有考虑大规模密集型数据之间的依赖性，或者只关注了减少数据之间的移动次数、传输时间等，而没有关注数据在各个单位时间之间访问量的不均衡性。

3 建模和问题定义

本文选取大规模密集型数据中的数据流进行数据分配算法的建模和设计。本节将在云计算环境下完成算法的建模，具体包括数据的建模、数据流的建模、数据存储中心的建模。

3.1 数据的建模

数据是数据流中最小的原子单位。对于每一个数据，本文仅关注数据的大小、数据访问量和数据之间的传输数据量。分配在同一数据存储中心上的数据访问量决定了数据存储中心的访问量，这是数据存储中心负载平衡的关键。同时，在大规模密集型数据流中，数据之间存在着数据量的传输，传输数据量也是数据之间依赖度的一个方面。将数据之间传输数据量高的数据分配在一起，能够减少数据之间传输的数据量。本文将数据的访问量和数据之间的传输数据量作为数据分配算法的主要依据。而对于数据的访问量，本文尝试从总体时间上的访问量和单位时间上的访问量两方面进行探索。数据集合和数据的定义如下。

定义 1 数据集合 D 是由 $|D|$ 个数据组成的集合 $D=\{d_i \mid i=1,2,\cdots,|D|\}$，其中 d_i 是第 i 个数据，$|D|$ 是组成数据集合的数据个数。

定义 2 数据的原子形式定义为 $d=\langle ds, DV\rangle$，其中 ds 表示数据的大小；DV 表示数据在总体时间上的访问量，这是由数据在各个单位时间上的访问量组成的一个集合。即对于

数据集合 D 中的第 i 个数据 d_i，其在总体时间上的访问量为 DV_i，同时 $DV_i=\{dv_{ij}\mid j=1,2,\cdots,|T|\}$，其中 dv_{ij} 是第 i 个数据总体时间的访问量 DV_i 在单位时间 j 的大小，而 $|T|$ 表示总体时间中包含的单位时间的个数。

这里定义的数据是一个通称，它可以是一个普通的密集型数据，也可以是一个密集型的数据集，还可以是数据库中的一个很大的表，或其他类型的数据，如 xml 或者文件等。

图 1 反映的是数据在单位时间上的访问量，即定义 2 中的 dv_{ij} 在不同单位时间上的变化。ti 是一个单位时间，其中 $i\in[1,12]$ 且为整数。数据的访问量在各个单位时间之间一般是不相等的，如果不考虑数据访问量的变化，将数据分配到数据存储中心，可能会造成数据存储中心访问量的不均衡，如图 1 所示。这样可能会导致数据存储中心效率的低下及其负载的瓶颈。因此，本文将数据存储中心的负载平衡作为数据分配的主要目标。

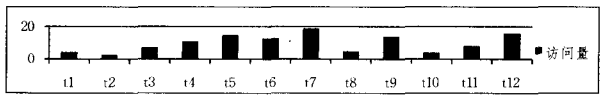

图 1　数据的访问量

数据不仅拥有总体时间的访问量和单位时间的访问量等属性而影响负载平衡，还存在与其他数据之间的传输数据量而影响数据流的读取效率。数据之间传输数据量的定义如下。

定义 3　数据之间的传输数据量 $s_{ij}=ds_{ij}$，其中 ds_{ij} 是数据 d_i 传输给数据 d_j 的数据的大小。

在大规模密集型数据流中，数据访问模式一般是固定不变的，所以数据之间的传输数据量一般也是固定不变的。

3.2　数据流的建模

数据流是一个只能以事先规定好的顺序被读取一次的数据的序列[16]。图 2 是一个简单的数据流例子，该数据流是由数据 $d_1,d_2,\cdots,d_8$ 组成的一个读取序列。而数据传输方向上的数值表示的是数据之间的传输数据量，例如 d_1 给 d_2 的传输数据量为 3，d_1 给 d_3 的传输数据量为 5。同时，两个传输数据量之间是没有关系的，即 d_1 给 d_2 的传输数据量 3 和 d_2 给 d_5 的传输数据量 5 之间没有关系。

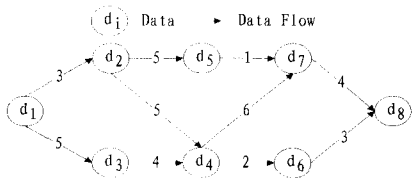

图 2　数据流示意图

数据库中对表的操作是一个典型的数据流的例子。数据库中的一个表是数据流示意图中的一个数据 d_i，而表自身的大小就是数据 d_i 的大小。同时在一个操作中，可能需要多个表的 join 操作。表之间的 join 操作都需要一个表向另外一个表传输数据量，这种现象就是数据流中数据之间传输数据量的现象。表之间传输数据量的大小就是数据流中数据之间传输数据量的大小。

数据流中，位于不同数据存储中心的数据之间传输的数据量越大，数据的传输对数据存储中心整体的性能造成的影响就越大。如果把有着大数据量传输的数据分配在同一个数据存储中心，就可以降低数据存储中心之间的传输数据量，这样就把数据之间大数据量的传输转化为数据存储中心内部的数据传输。数据存储中心内部的数据传输较之存储中心之间有带宽限制的数据传输，在传输速度和质量上都有着很大的提高，从而达到减少网络开销和全局数据传输的目的。

3.3　数据存储中心的建模

云计算环境中，数据存储中心是由大规模的廉价服务器集群构成的。对于每一个数据存储中心，本文仅关注存储空间大小、其负载平衡和数据与数据存储中心之间的传输数据量。数据存储中心内部的数据传输速度很快，传输时间可以忽略。但是由于带宽的限制和物理位置离散的可能，使得数据存储中心之间的数据传输时间不能被忽略。数据存储中心集合和数据存储中心的定义如下。

定义 4　数据存储中心集合 M 是由 $|M|$ 个数据存储中心组成的集合 $M=\{m_i\mid i=1,2,\cdots,|M|\}$，其中 m_i 是第 i 个数据存储中心，$|M|$ 是数据存储中心的个数。

定义 5　数据存储中心的原子形式定义为 $m=\langle ms,MV\rangle$，其中，ms 表示数据存储中心存储空间的大小。MV 表示数据存储中心在总体时间上的访问量，这是由数据存储中心在单位时间上的访问量组成的一个集合。即对于数据存储中心集合 M 中的第 i 个数据存储中心 m_i，其在总体时间上的访问量为 MV_i，同时 $MV_i=\{mv_{ij}\mid j=1,2,\cdots,|T|\}$，其中 mv_{ij} 是第 i 个数据存储中心总体时间的访问量 MV_i 在单位时间 j 的大小，而 $|T|$ 表示总体时间中包含的单位时间的个数。

数据存储中心的访问量是存储在该数据存储中心上数据的访问量在对应单位时间上加和叠加的效果：数据存储中心 m_i 在总体时间上的访问量为 $MV_i=\Sigma DV_j$ 且 $d_j\in m_i$，其中 DV_j 是分配在数据存储中心 m_i 中的数据 d_j 在总体时间上的访问量。数据存储中心 m_i 在单位时间 j 上的访问量 $mv_{ij}=\Sigma dv_{kj}$ 且 $d_k\in m_i$，其中 dv_{kj} 是分配在数据存储中心 m_i 中的数据 d_k 在单位时间 j 上的访问量。

图 3 表示的是数据存储中心在单位时间上的访问量，即 mv_{ij} 在不同单位时间之间的变化。ti 是一个单位时间，其中 $i\in[1,12]$ 且为整数。图中假定某一数据存储中心中有 4 个数据，则该数据存储中心在单位时间上的访问量就是 d_1、d_2、d_3 和 d_4 这 4 个数据在相对应单位时间上访问量加和叠加的结果。对于大规模密集型数据，本文试图寻找一个对于数据存储中心在单位时间之间访问量上相对平衡的一个数据分配结果，即寻找一个对于图 3 中的 4 个数据在对应单位时间上的访问量加和叠加之和在各个单位时间之间的变化趋于平缓的数据分配结果。这样就将数据在单位时间上访问量的不平衡性转化为数据存储中心在单位时间上访问量的平衡性。

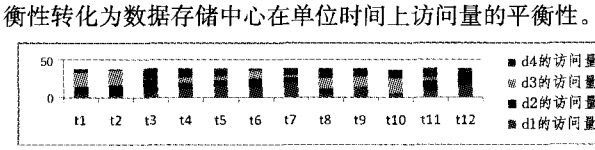

图 3　数据存储中心的访问量

数据存储中心不仅拥有总体时间的访问量和单位时间的访问量等属性，还存在与数据之间的传输数据量的关系。对于每一个数据存储中心 m_k 和不存储在 m_k 上的数据 d_i，数据与数据存储中心之间的传输数据量 $S_{ik}=\Sigma s_{ij}$ 且 $d_j\in m_k$，其中 $k=1,2,\cdots,|M|$，s_{ij} 表示数据 d_i 与数据 d_j 之间的传输数据

量。在进行数据分配的时候，尽量保证传输数据量大的数据存放在一个数据存储中心，这也是本文关注的主要问题。

4 算法设计

云计算环境中，大规模密集型数据流应用为 $A=\langle M, D\rangle$，其中 M 是数据储存中心的集合，D 是待分配的数据的集合。DOVE 算法的流程如图 4 所示。

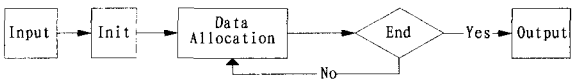

图 4 算法流程图

DOVE 算法的输入为 Input=$\langle M, D\rangle$，输出为 Output=$\langle R\rangle$，其中 M 是数据存储中心集合，D 是数据集合，R 是数据分配的结果集合。Init 模块的主要工作是对数据和数据存储节点进行初始化处理，包括从数据集合中按照一定的规则找到数据存储中心中的初始化数据、将数据存储中心进行初始化的数据分配等。Data Allocation 模块是整个算法的核心模块，其主要工作是在保证数据存储中心负载平衡和减少数据存储中心之间传输数据量的基础上进行数据分配。End 模块是算法的反馈模块，用来判断分配结果是否符合结束条件。如果分配结果不符合结束条件，则 End 模块要对参数进行调整并重新进行数据分配。

4.1 初始化阶段

在数据分配之前，数据存储中心要初始化一些数据，以保证数据存储中心的存储模式和数据分配算法中的距离的计算。对于输入 Input=$\langle M, D\rangle$，初始化算法从数据集合 D 中找到符合初始化方式的 $|M|$ 个数据，并将其分别分配到个数为 $|M|$ 的数据存储中心中。

Algorithm 1 Initialization Algorithm

```
Input: D: set of data
       M: set of data centers
Output: set of data centers with initial data
        set of data without initial data
1. ID=∅ // ID: set of initialized data
2. F=∅ // F: set of initializing foundation of data
3. RD= D−ID // RD: set of residual data
4. for d_i in D:
5.    f_i=compute_foundation(d_i)
6.    f_i=F ∪ f_i
7. sort f_i in F
8. for m_i in M:
9.    for f_i in F:
10.      if size(d_i of f_i)<size(m_i):
11.         allocate d_i to m_i
12.         ID=ID∪d_i
13. RD=D−ID
14. D=RD
```

初始化算法中，compute_foundation 函数(Algorithm 1 第 5 行)是按照选择的初始化方式计算每个数据的初始化依据。而初始化选取数据的方式有很多种，比如按照数据之间传输数据量的大小选取、按照数据单位时间之间访问量方差的大小选取、按照数据的大小选取或者随机选取等。本文尝试最小访问量方差的初始化方式，其定义如下。

定义 6 最小访问量方差的数据集合为 $D_{minv}=\{d_i \mid i=1, 2, \cdots, |M|\}$，其中 $|M|$ 为数据存储中心的个数，且对于 $\forall d_i \in D_{minv}$ 和 $\forall d_j \notin D_{minv}$，有 $Dvisit(d_i)<Dvisit(d_j)$，$Dvisit(d_i)$ 表示第 i 个数据在所有单位时间上访问量的方差。

4.2 数据分配阶段

初始化阶段完成之后，就要对剩余数据集中的 $|D|-|M|$ 个数据按照数据存储中心的容量、访问量和依赖度的约束进行数据分配。

Algorithm 2 Data Allocation Algorithm

```
Input: D: set of data without initial data
       M: set of data centers with initial data
Output: set of data centers with allocated data
        set of data with residual data
1. for d_i in D:
2.    distance=compute_distance(M, d_i, α, β, γ, δ)
3.    sort m_i by distance in M
4.    for i=1,2,…,|M|:
5.       if size(d_j in m_i+d_i)<size(m_i):
6.          allocate d_i to m_i
7.          update_dependency(m_i)
8.          update_visit(m_i)
9.          delete d_i in D
10. return D
```

算法的时间复杂度是 $O((|D|-|M|)\times|M|\log(|M|))$，其中 $|D|$ 为数据的个数，$|M|$ 为数据中心的个数。Data Allocation 模块的输出是已经分配好数据的数据存储中心集合。整个算法的下一个模块即 End 模块会对输出的分配结果进行反馈。compute_distance 函数(Algorithm 2 第 2 行)是按照规则计算数据到每个数据存储中心的距离，而参数 α、β、γ 和 δ 是距离函数传入的 4 个权重值，这 4 个权重值是由 End 模块中的 Adjust Algorithm 进行调整的。

Algorithm 3 Compute Distance Algorithm

```
Input: M: set of data centers
       d_i: data
       α, β, γ, δ: weighing for computing distance
Output: set of distance between M and d_i
1. distance = ∅ // set of distance between M and d_i
2. for m_i in M:
3.    distance_i=0 // distance_i: distance between m_i and d_i
4.    V=compute_variance(m_i, d_i)
5.    // V: the evaluation of variance
6.    E=compute_expect(m_i, d_i)
7.    // E: the evaluation of expect
8.    EM=compute_dis_EM(m_i, d_i)
9.    // EM: the evaluation of EM
10.   Dep=compute_dis_dep(m_i, d_i)
11.   // Dep: the evaluation of dependency
12.   distance_i=α×V +β×E + γ×EM + δ×Dep
13.   distance=distance∪distance_i
14. return distance
```

具体实现中，数据到数据存储中心之间的距离是由数据存储中心访问量的变化和数据与数据存储中心之间的传输数据量决定的。而数据存储中心访问量的变化是由 compute_variance 函数(Algorithm 3 第 4 行)、compute_expect 函数(Algorithm 3 第 6 行)和 compute_dis_EM 函数(Algorithm 3 第 8 行)进行量化的，数据与数据中心的传输数据量是由 compute_dis_dep 函数(Algorithm 3 第 10 行)进行量化的。compute_variance 函数的主要计算依据是待分配数据 d_i 模拟分配到数据存储中心 m_i 之后，m_i 在单位时间的访问量变化

方差。compute_expect 函数的主要计算依据是待分配数据 d_i 模拟分配到数据存储中心 m_i 之后，m_i 在单位时间的访问量期望。compute_dis_EM 函数的主要计算依据是待分配数据 d_i 模拟分配到数据存储中心 m_i 之后，m_i 在单位时间上的最大波峰值。最后，compute_dis_dep 函数的主要计算依据是待分配数据 d_i 与数据存储中心 m_i 之间的依赖度。在本文的模型中，依赖度是数据之间的传输数据量。

Algorithm 3 将数据存储中心负载平衡和数据与数据存储中心传输数据量转化为数据到数据存储中心之间的距离(Algorithm 3 第 12 行)。Algorithm 2 选择距离最小且符合存储节点硬件限制的数据放入存储节点中。这样既能保证数据存储中心的负载平衡，又能减少不同数据存储中心之间的传输数据量。

4.3 反馈阶段

算法流程的 End 模块是反馈模块。该模块利用 ET、ES、EM[19] 和 EDT 等 4 个指标对 Data Allocation 模块的数据分配结果进行评价。

ET 指标是每个节点的总体时间使用率指标。$ET=\Sigma Var(MV_i)/|M|$ 且 $i=1,2,\cdots,|M|$，其中 $|M|$ 是数据存储中心的个数，$Var(MV_i)$ 是数据存储中心 m_i 在总体时间范围内各个单位时间之间访问量变化的方差。

ES 指标是系统每个单位时间上的均衡指标。$ES=\Sigma Var(MV_i)/|T|$ 且 $i=1,2,\cdots,|M|$，其中 $|T|$ 是单位时间的个数，$Var(MV_i)$ 指的是数据存储中心 m_i 在总体时间范围内各个单位时间之间访问量变化的方差。

EM 指标是系统出现的最大波峰值。$EM=\max(mv_{ij})$，其中 $i=1,2,\cdots,|M|$，$j=1,2,\cdots,|T|$，$|M|$ 是数据存储中心的个数，而 $|T|$ 是单位时间的个数。

EDT 指标是传输数据量指标。$EDT=\Sigma s_{ij}$ 且 d_i 和 d_j 不属于同一个数据存储中心。该指标统计了系统中数据存储中心之间传输数据量的总和，表示了数据分配结果与依赖度之间的关系。EDT 指标越好，数据分配算法在数据之间依赖度上的利用就越好。

End 模块首先计算出数据分配结果在 ET、ES、EM 和 EDT 等 4 个指标上的收益，计算公式如下：

$$Benefit(E)=\begin{cases}\dfrac{E_{best}-E_{new}}{E_{best}}, & \text{if } E_{best}>E_{new}\\ 0, & \text{if } E_{best}=E_{new}\\ \dfrac{E_{best}-E_{new}}{E_{new}}, & \text{if } E_{new}>E_{best}\end{cases} \quad (1)$$

式中，E_{best} 指 4 个指标各自最好的一次分配结果得出的指标值，而 E_{new} 指的是当前数据分配结果得出的对应指标值。End 模块根据式(1)分别计算出 4 个评价指标的收益 $Benefit(ET)$、$Benefit(ES)$、$Benefit(EM)$ 和 $Benefit(EDT)$。然后 End 模块会根据这 4 个不同指标上的收益结果计算对数据分配结果的评价值，计算公式如下：

$$Evaluation=\mu\times Benefit(ET)+\sigma\times Benefit(ES)+\varphi\times Benefit(EW)+\psi Benefit(EDT) \quad (2)$$

式中，μ、σ、φ 和 ψ 是权重值。End 模块将根据式(2)计算出的上次数据分配结果的评价值和之前最好的评价值做对比，保留一个较好的评价值，同时记录下较好的评价值对应的距离函数参数值，即 Algorithm 3 中的 α、β、γ 和 δ。最后 End 模块根据分配结果对这 4 个参数进行动态调整，同时判断其是否满足算法结束的条件。

Algorithm 4 Adjust Algorithm

Input: λ_i: weighing for computing distance

Output: end or λ_i, λ_{new1}, λ_{new2}

1. Init(dilate) //dilate: the parameter of computing
2. Init(shrink) //shrink: the parameter of computing
3. if λ_i is the last one in S_λ:
4. $\quad\lambda_{new1}=\min(\lambda_i\times dilate, high)$
5. else:
6. $\quad\lambda_{new1}=(\lambda_i+\lambda_{i+1})/shrink$
7. if λ_i is the first one in S_λ:
8. $\quad\lambda_{new2}=\max(\lambda_i/dilate, low)$
9. else:
10. $\quad\lambda_{new2}=(\lambda_{i-1}+\lambda_i)/shrink$
11. if $|\lambda_i-\lambda_{new1}|<\varepsilon$ or $|\lambda_{new2}-\lambda_i|<\varepsilon$:
12. $\quad$return end
13. else:
14. $\quad$return λ_i, λ_{new1}, λ_{new2}

Algorithm 4 是一个类二分查找的动态调整参数算法。其中参数 λ_i 可以指代 α、β、γ 和 δ 中的任何一个，而 ε(Algorithm 4 第 11 行)是指调整参数与原参数的最小间隔。因为数据分配算法中有 4 个参数，所以 Algorithm 4 在整个算法中是被循环调用的。Algorithm 4 尝试在一个较好的评价所对应的参数 λ_i 附近找到一个更好的参数值，从而得到一个更好的数据分配结果。

图 5 是说明参数调整中 Algorithm 4 是怎样被循环调用的一个例子。为了例子的简单性，我们以 2 个参数的调用为例进行说明。假设参数 α 有 3 个可能的取值常量 i、j 和 k，参数 β 也有 3 个可能的取值常量 o、p 和 q。End 模块先调用 Algorithm 4，开始调整参数 α。同时循环调用 Algorithm 4，开始调整参数 β。调整过程中，假设参数 α 先选定一个值 i，评价函数会在 $\alpha=i$ 的情况下对 β 的 3 个取值进行评价，从中选出最好的评价结果所对应的 β 的参数值 o。这样就有一组评价良好的参数 $\alpha=i$、$\beta=o$。然后 α 选择下一个参数值 j，评价函数会在 $\alpha=j$ 的前提下对参数 β 的 3 个取值进行评价，从中选出最好的评价结果所对应的 β 的参数值 q。这样就又有一组评价良好的参数据 $\alpha=j$、$\beta=q$。此时 End 模块会对比这两组良好的参数值，从中选择最好的一组反馈给 End 模块。

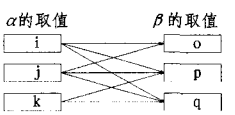

图 5 Algorithm 4 的调用

5 实验及其结果分析

5.1 实验设置与环境

本节将对 DOVE 数据算法与 Yuan 提出的基于 BEA 聚类的数据分配算法[5](以下简称 cluster 算法)进行对比实验。实验选取的对象是数据定义中的一个子集密集型数据集。对于随机生成的数据集，分别使用 DOVE 与 cluster 在模拟的云环境中进行数据分配，然后对数据分配结果分别用 ET、ES、EM 和 EDT 4 个指标进行评价对比。在相同参数和相同数据的情况下，每次实验运行 10 次，统计 10 次实验结果各指标的平均值。

实验环境为 Intel Pentium Dual E2200，CPU 频率为 2.2GHz和 2.19GHz，内存为 2GB。

编程语言为 Python 和 Java。

5.2 实验结果和分析

DOVE 算法和 cluster 算法在 ET、ES、EM 和 EDT 4 个指标上的实验对比结果分别如图 6—图 9 所示。

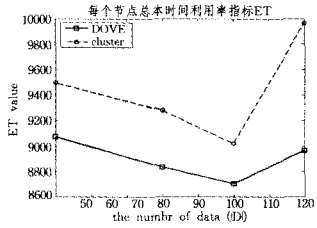

图 6 DOVE 和 Cluster 的 ET 指标对比

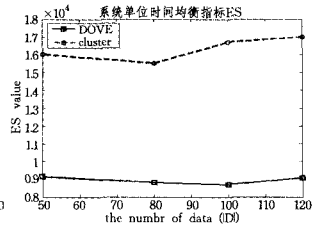

图 7 DOVE 和 Cluster 的 ES 指标对比

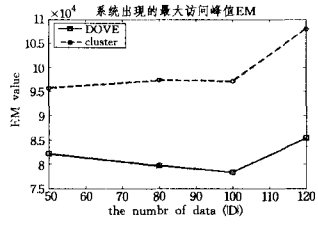

图 8 DOVE 和 Cluster 的 EM 指标对比

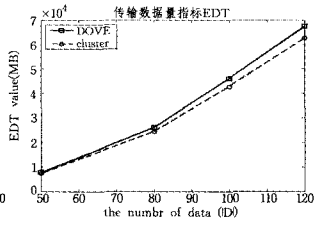

图 9 DOVE 和 Cluster 的 EDT 指标对比

如图 6 所示，随着数据集个数的增加，两种数据分配算法的 ET 评价指标都呈现出先减少再增加的趋势。但是 DOVE 算法在 ET 指标方面一直优于 cluster 算法，并且在一定数据集个数之后，DOVE 算法的 ET 指标的增速明显比 cluster 算法的 ET 指标的增速要慢。在数据集是 50 的时候，DOVE 算法的 ET 指标比 cluster 算法低 4.48%；在数据集是 80 的时候低 4.83%；在数据集是 100 的时候低 3.53%；在数据集是 120 的时候低 10.03%。这说明 DOVE 算法较之 cluster 算法在每个数据存储中心的总体时间上取得了较好的负载均衡结果。

如图 7 所示，随着数据集个数的增加，两种数据分配算法的 ES 评价指标都呈现出比较平缓的变化趋势。但是 DOVE 算法在 ES 指标上一直优于 cluster 算法，并且两者的差距比较明显。在数据集是 50 的时候，DOVE 算法的 ES 指标比 cluster 算法的低 42.9%；在数据集是 80 的时候低 43.26%；在数据集是 100 的时候低 47.94%；在数据集是 120 的时候低 46.63%。这说明 DOVE 算法较之 cluster 算法在整个数据存储中心的单位时间上取得了较好的负载均衡结果。

如图 8 所示，随着数据集个数的增加，两种数据分配算法的 EM 指标都呈现出先减少再增加的趋势。但是 DOVE 算法在 EM 指标方面明显优于 cluster 算法，即 DOVE 算法的最大波峰值比 cluster 算法小。在数据集是 50 的时候，DOVE 算法 EM 指标比 cluster 算法低 14.17%；在数据集是 80 的时候低 18.21%；在数据集是 100 的时候低 19.33%；在数据集是 120 的时候低 20.99%。EM 指标说明 DOVE 算法较之 cluster 算法有着较小的系统最大波峰值，数据访问不易出现瓶颈。

如图 9 所示，随着数据集个数的增加，两种数据分配算法的 EDT 指标都呈明显上升趋势，即两种数据分配算法的传输数据量都随着数据集个数的上升而增加。与 cluster 算法相比，DOVE 算法所对应的传输数据量略多，但二者差距非常小。在数据集是 50 的时候，DOVE 算法的 EDT 指标比 cluster 算法高 5%；在数据集是 80 的时候高 1.8%；在数据集是 100 的时候高 7%；在数据集是 120 的时候高 7.55%。EDT 指标说明 DOVE 算法在利用数据依赖性方面不如 cluster 算法，但是两者的差距很小。

综上所述，虽然 cluster 算法在传输数据量的评价指标 EDT 中的实验结果好于 DOVE 算法，但是两者的 EDT 指标值很接近。同时，DOVE 算法在负载平衡的 3 个评价指标 ET、ES 和 EM 的实验结果中均好于 cluster 算法。而且在 ES 和 EM 指标的对比中，DOVE 取得了很大的优势。这说明 DOVE 算法能够有效地保证数据存储中心的负载平衡。同时 DOVE 兼顾了密集型数据之间的依赖性，减少了数据存储中心之间的传输数据量，提高了数据的访问效率。

结束语 本文首先分析了云计算环境中进行大规模密集型数据分配所面临的挑战。通过大规模密集型数据中的数据流完成了建模，并提出了云计算环境中基于访问量和依赖度的数据分配算法 DOVE。DOVE 采用了不同于传统数据分配算法的约束条件，更好地适应了大规模密集型数据以及云计算环境的特点。通过与其他数据分配算法的对比实验，说明了 DOVE 有着良好的综合性能，特别是在保证数据存储中心负载平衡方面，DOVE 算法的优势十分明显。同时 DOVE 兼顾了密集型数据之间的依赖性，降低了数据存储中心之间的传输数据量。

DOVE 算法还有许多需要改进的地方，例如数据存储中心的负载平衡与数据之间传输量的权值如何平衡、如何利用副本机制将依赖度大的数据复制并分配在多个数据存储中心、如何在纯分布式环境中实现 DOVE 算法、如何在考虑复杂网络拓扑结构的前提下更加精确地计算数据传输所要付出的代价等，这些都需要在将来的工作中进一步研究与探讨。

参 考 文 献

[1] Brantner M, Florescuy D, Graf D, et al. Building a Database on S3[C]// SIGMOD. Vancouver, BC, Canada, 2008: 251-263

[2] Moretti C, Bulosan J, Thain D, et al. All-Pairs: An Abstraction for Data-intensive Cloud Computing[C]// IEEE International Parallel & Distributed Processing Symposium (IPDPS'08). 2008: 1-11

[3] Deelman E, Chervenak A. Data Management Challenges of Data-intensive Scientific Workflows[C]// IEEE International Symposium on Cluster Computing and the Grid. 2008: 687-692

[4] Zeng Wen-ying, Zhao Yue-long, Ou Kai-ri. Research on cloud storage architecture and key technologies[C]// Proceedings of the 2nd International Conference on Interatction Sciences. 2009

[5] Yuan D, Yang Y, Liu X, et al. A data placement strategy in scientific cloud workflows[J]. Future Generation Computer Systems, 2010, 26(6): 1200-1214

[6] Foster I, Yong Z, Raicu I, et al. Cloud Computing and Grid Computing 360-degree Compared[C]// Grid Computing Environments Workshop(GCE '08). 2008: 1-10

[7] Barga R, Gannon D. Scientific versus Business Workflows[M]. Workflows for e-Science, 2007: 9-16

[8] Kosar T, Livny M. Stork: making data placement a first class citizen in the grid[C]// Proceedings of 24th International Conference on Distributed Computing Systems(ICDCS 2004). 2004: 342-349

(下转第 171 页)

第三，在这10组训练和测试集上分别评估约简算法的分类精度，并计算平均值。针对每一组训练和测试集，首先根据约简算法所生成的约简来删除训练集中的冗余属性；然后利用RSES软件中提供的LEM2算法来提取决策规则；最后，利用这些规则来对测试集进行分类。RSES中相关的参数设置如下：Shortening ratio：0.9；Conflicts resolve by：STANDARD VOTING[14]。

表4给出了不同算法所得到的约简的分类精度。

表4 分类精度比较

数据集	分类精度(%)					
	算法1	POSAR	CEAR	DISMAR	GAAR	PSORSAR
Vote	95.2	94.33	92.33	93.67	94.0	95.33
Mushroom	100	100	90.83	100	100	99.7
Lymphography	79.1	85.71	72.14	74.29	70.0	75.71
SOYBEAN-Small	100	100	100	100	97.5	100
Zoo	97.9	96.0	94.0	94.0	92.0	96.0

从表4可以看出，算法1的分类性能在大多数情况下都优于或等于其他算法。除了在Vote数据集上低于PSORSAR算法，在Lymphography数据上低于POSAR算法之外，我们的算法都具有最高的分类精度。因此，从总体上来看，我们的算法在这些数据集上具有更好的分类能力。

结束语 传统的基于正区域的约简算法往往忽略了对决策表中的不一致对象以及属性约简过程中所产生的不一致对象的处理。通过分析，我们发现删除这些不一致对象不会影响正区域的计算和属性约简的结果，但是不一致对象的存在却会降低算法的执行效率。因此，本文提出了一种基于重构相容决策表的属性约简算法。在本算法中，首先删除决策表和决策子表中的不一致对象，再通过计算正区域来得到约简，从而提高了算法的执行效率。在UCI数据集上的实验结果表明，我们的算法能得到较小的属性约简集，并且约简后所生成的规则具有较高的分类精度。

参考文献

[1] 王国胤. Rough集理论与知识获取[M]. 西安：西安交通大学出版社，2001

[2] 刘清. Rough集及Rough推理[M]. 北京：科学出版社，2005

[3] 王国胤，于洪，杨大春. 基于条件信息熵的决策表约简[J]. 计算机学报，2002，25(7)：759-766

[4] 刘少辉，盛秋戬，吴斌，等. Rough集高效算法的研究[J]. 计算机学报，2003，26(5)：525-529

[5] Pawlak Z. Rough sets[J]. Int. Journal of Computer and Information Sciences，1982，11(5)：341-356

[6] Wang X Y，Yang J，Teng X L，et al. Feature Selection Based on Rough Sets and Particle Swarm Optimization[J]. Pattern Recognition Letters，2007，28(4)：459-471

[7] Hu K Y，Lu Y C，Shi C Y. Feature ranking in rough sets[J]. AI Communication，2003，16(1)：41-50

[8] Wroblewski J. Finding minimal reducts using genetic algorithms [C]// Proc. of the 2nd Annual Joint Conf. on Information Sciences. 1995，186-189

[9] Jensen R，Shen Q. Finding rough set reducts with ant colony optimization[C]// Proc. of the 2003 UK Workshop on Computational Intelligence. 2003：15-22

[10] Hu X H. Knowledge discovery in databases：an attribute-oriented rough set approach[D]. Regina University，1995

[11] Wang G Y，Zhao J. Theoretical study on attribute reduction of rough set theory：comparison of algebra and information views [C]// Proc. of the 3rd IEEE Int. Conf. on Cognitive Informatics. 2004：148-155

[12] Bazan J. A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision table[M]. Rough Sets in Knowledge Discovery，1998：321-365

[13] Bazan J，Nguyen H S，Nguyen S H，et al. Rough set algorithms in classification problem[M]. Rough Set Methods and Applications，2000：49-88

[14] Skowron A，et al. RSES 2.2 User's Guide[M]. Warsaw University，2005

[15] Bay S D. The UCI KDD repository[EB/OL]. http://kdd.ics.uci.edu，1999

[16] 徐章艳，刘作鹏，杨炳儒，等. 一个复杂度为max(O(|C‖U|)，O(|C|^2|U/C|))的快速属性约简算法[J]. 计算机学报，2006，29(3)：391-399

(上接第146页)

[9] Hardavellas N，Ferdman M. Reactive NUCA：near-optimal block placement and replication in distributed caches[C]// Proceedings of the 36th Annual International Symposium on Computer Architecture(ISCA '09). Austin，TX，USA，2009：184-195

[10] Xie T. SEA：A Striping-based Energy-aware Strategy for Data Placement in RAID-Structured Storage Systems [J]. IEEE Transactions on Computers，2008，57：748-761

[11] Cope J M，Trebon N，Tufo H M，et al. Robust data placement in urgent computing environments[C]// IEEE International Symposium on Parallel & Distributed Processing (IPDPS 2009). 2009：1-13

[12] Xiao Nong，Chen Tao，Liu Fang. RSEDP：An Effective Hybrid Data Placement Algorithm for Large-scale Storage Systems[J]. 2009，55(1)：103-122

[13] Ghemawat S，Gobioff H，Leung S-T. The Google file system[J]. SIGOPS Oper. Syst. Rev，2003，37：29-43

[14] Hadoop[EB/OL]. http://hadoop.apache.org/2011-03-25

[15] Amazon Elastic Computing Cloud[EB/OL]. http://aws.amazon.com/ec2/，2011-03-30

[16] Henzinger M，Raghavan P，Rajagopalan S. Computing on Data Streams[R]. TR-1998-011. Digital Equipment Corp，Aug 1998

[17] Barroso L A，Dean J，Hölzle U. Web search for a planet：The Google cluster architecture[J]. IEEE Micro，2003，23(2)：22-28

[18] Lenk A，Klems M，Nimis J，et al. What's inside the cloud? An architectural map of the Cloud landscape[C]// Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing. May 2009：23-31

[19] 陈超，蒋建春，丁治明. 基于时序片段评价的数据分配算法[J]. 计算机研究与发展，2010，47(z1)：385-393

[20] 郑湃，崔立真，王海洋，等. 云计算环境下面向数据密集型应用的数据布局策略与方法[J]. 计算机学报，2010(8)：1472-1480

[21] 文明波，丁治明. 适用于云计算的面向查询数据库数据分布策略[J]. 计算机科学，2010，37(9)：168-172

[22] 王道祥. 基于分布式数据库的综合信息系统关键技术研究[D]. 长沙：国防科学技术大学，2006