数据中心网络高效数据汇聚传输算法

陆菲菲[1),2)] 郭得科[3)] 方 兴[1)] 谢向辉[1)] 罗兴国[2)]

[1)](数学工程与先进计算国家重点实验室 江苏 无锡 214125)

[2)](解放军信息工程大学国家数字交换系统工程技术研究中心 郑州 450002)

[3)](信息系统工程国防科技重点实验室(国防科学技术大学) 长沙 410073)

摘 要 在数据中心中,类MapReduce的分布式计算系统在数据的混洗阶段产生巨大流量,令数据中心的东西向网络资源成为瓶颈.将这些高度相关的数据流在接收端进行聚合是分布式计算的通用处理方式,为了降低网络通信量并有效利用带宽,文中采用网内关联性流量的汇聚传输策略,将混洗和汇聚并行化,达到进一步降低东西向网络资源消耗、缩短混洗阶段延迟的目的.目前提出的IRS-based算法在适用场景上有一定局限性,为了解决这一问题,文中首先在以服务器为中心的代表结构BCube上建立incast最小树模型,分别提出MIB-based算法和MC-based算法,仅根据已知拓扑结构和发送节点编号即可快速生成一棵近似的最小代价incast树.MIB-based算法针对发送节点强关联的情况,使高层发送节点尽可能汇聚到已有的低层发送节点构建incast树;MC-based算法针对发送节点松散关联的情况,将节点进行最大程度上的聚合,通过增加最少的汇聚点完成incast树的构建.随后将上述两种算法结合起来进一步提出适用于各种场景的M2-based算法,通过推算时间复杂度证明该算法能够满足在线构建incast树的需求.最后,详细分析了M2-based算法对其他数据中心网络结构的适应性以及网内汇聚传输能够减少作业完成时间的原理.小规模实验结果表明,在不同网络规模下,M2-based比IRS-based节省了网络中约3%的数据量,整个作业在混洗和Reduce阶段的等待时间比不采用网内汇聚缩短约2/3;在不同传输节点规模下,M2-based比IRS-based节省了网络中约19%的数据量,整个作业在混洗和Reduce阶段的等待时间比不采用网内汇聚缩短约3/4.

关键词 数据中心;数据汇聚;网内聚合;混洗传输;incast树

中图法分类号 TP393 **DOI号** 10.11897/SP.J.1016.2016.01750

Efficient Data Aggregation Transfers in Data Center Networks

LU Fei-Fei[1),2)] GUO De-Ke[3)] FANG Xing[1)] XIE Xiang-Hui[1)] LUO Xing-Guo[2)]

[1)](*State Key Laboratory for Mathematical Engineering and Advanced Computing*, *Wuxi*, *Jiangsu* 214125)

[2)](*National Digital Switching System Engineering & Technological Research Center*, *The PLA Information Engineering University*, *Zhengzhou* 450002)

[3)](*Key Laboratory on Information System Engineering* (*National University of Defense technology*), *Changsha* 410073)

Abstract In data centers, distributed computing systems like MapReduce produce massive amount of traffic across successive processing stages. Such shuffle transfers make east-west network resource become a bottleneck. In many commonly used workloads, data flows from all senders to each receiver are typically highly correlated. Many state-of-the-practice systems thus already apply aggregation functions at the receiver side of a shuffle transfer to reduce the output data size. To lower down the network traffic and efficiently use network bandwidth, we introduce

收稿日期:2015-03-26;在线出版日期:2015-11-03.本课题得到国家"九七三"重点基础研究发展规划项目青年科学家专题项目(2014CB347800)、国家自然科学基金优秀青年基金(61422214)、国家自然科学基金(91430214)、国家"八六三"高技术研究发展计划项目基金(2013AA01A213)资助.**陆菲菲**,女,1981年生,博士研究生,工程师,中国计算机学会(CCF)会员,主要研究方向为分布式计算和数据中心网络.E-mail:lu.feifei@meac-skl.cn.**郭得科**,男,1980年生,博士,副研究员,国家自然科学基金优秀青年基金获得者,中国计算机学会(CCF)会员,主要研究方向为分布式计算、数据中心网络和无线通信系统.**方 兴**,男,1980年生,博士,助理研究员,中国计算机学会(CCF)会员,主要研究方向为计算机系统结构和高速模拟电路.**谢向辉**,男,1958年生,研究员,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为计算机系统结构、网络和分布式计算.**罗兴国**,男,1951年生,教授,博士生导师,主要研究领域为无线通信、软件无线电和高性能计算.

in-network aggregation for associated traffic and parallelize the shuffle and reduce phases. It can significantly reduce consuming the rare east-west network resource, and avoid long latency time produced by the shuffle phase in MapReduce jobs. IRS-based algorithm proposed currently has certain limitations. To solve this problem, we first built a model for incast minimal tree with BCube, a representative server-centric networking structure for future data centers, and propose two approximate incast tree construction methods named MIB-based and MC-based, respectively, solely based on the labels senders and the data center topology. MIB-based method is applied to the case of highly correlative senders. It can build an incast minimal tree by making an endeavor to aggregate the high-level senders to low-level senders. MC-based method is applied to the case of loose associative senders. It can build an incast minimal tree by aggregating nodes as far as possible and increasing the least nodes. Then we combined two methods and further proposed M2-based method for any case. It proved that the method we proposed can meet the demand of building the incast tree on line by calculating the time complexity of the M2-based incast tree building method. At last, we analyzed the adaptability of M2-based to other data center structures, and the principle of in-network aggregation in reducing the job execution time. The small-scale experimental results show that, in the different size of data center, M2-based saves the network traffic by 3% on average compared to IRS-based, and shortens about two-third waiting time of a job in the shuffle and reduce phase compared to the existing method which does not perform the in-network aggregation. In the different size of incast transfer, M2-based saves the network traffic by 19% on average compared to IRS-based, and shortens about three-forth waiting time of a job in the shuffle and reduce phase compared to the existing method.

Keywords　data center; data aggregation; in-network aggregation; shuffle transfer; incast tree

1　引　言

随着分布式数据处理技术和云计算的不断发展，大规模数据中心成为分布式计算系统(如 MapReduce[1]、Dryad[2]、CIEL[3]、Pregel[4] 和 Spark[5])处理和存储大数据的平台.在数据中心上运行的应用利用分布式计算框架将数据分发到成百上千台服务器上并行执行，从而达到在短时间内处理 Tb(Terabyte)级以上海量数据的目的.

为了确保高可扩展性，这些应用通常采用分割-汇聚的操作模式：在分割阶段，主节点将作业或用户请求分成若干个子任务，发送到不同的工作节点并行执行，每个工作节点处理一个数据子集，并在本地生成部分中间结果；在汇聚阶段，由各工作节点生成的庞大的中间结果集被分割成不同子集，由一个或多个工作节点通过聚合处理得到最终的输出数据.例如，在 Hadoop 的 MapReduce 中，输入数据集被分割并分别发送到不同的 mapper 上处理，生成一系列中间结果；reducer 把具有相同键的中间数据合并从而得到结果数据.可见，汇聚阶段典型地包括一个数据流的混洗(shuffle)传输过程，大量工作节点之间相互通信，产生“多对多”的流量模式，Facebook 的数据中心显示在汇聚阶段产生的网络流量占总流量的 46%[6]，许多研究均表明，这一巨大的网络流量使网络传输成为 MapReduce 应用的性能瓶颈[7-9].例如，在 Facebook 的所有包含 Reduce 阶段的 MapReduce 作业中，网络传输占总运行时间的 33%，在 26%的作业中，传输占据 50%以上的运行时间，有 16%的作业，传输甚至占据其 70%以上的运行时间[6].产生上述瓶颈的原因主要有以下几点，首先，现代数据中心的内部流量已从传统的“南北流量”为主演变为“东西流量”为主，而 MapReduce 作业造成的大量流量令数据中心的东西向网络资源成为瓶颈；其次，传统数据中心网络的带宽收敛比较高，大大限制了混洗过程的数据传输率，如汇聚层通常为 5∶1，即在一些通信模式下汇聚层的可用带宽仅为服务器端的 20%，而核心层则高达 80∶1，甚至 240∶1[7]；再者，服务器端有限的可用带宽(通常最多为 1 Gbps 或 10 Gbps)成为“多对多”通信的瓶颈；最后，普通商业交换机的可用缓存空间较小，当多台服务器同时向一台服务器传输大量数据时往往造成

交换机的缓冲区溢出而发生丢包,从而导致网络吞吐量急剧下降,即典型的 TCP Incast[10] 问题. 现存的解决这些问题的方法主要包括采用全二分带宽拓扑结构消除网络的带宽收敛比以增加网络的可用带宽,如VL2[7]、PortLand[11]、BCube[12]、DCell[8] 等,或通过数据迁移避免网络热点,如 Hedera[9]. 然而,上述方法都未能减少网络的通信量,使得性能还是受限于终端服务器的可用带宽,尽管目前许多数据中心已经升级到 10G 网络,但鉴于成本因素和更高的带宽收敛比,全面更新到 40G 网络仍然需要相当长的一段时间. 再者,由于缺乏任务级的调度策略我们仍不能很好地处理流的聚合行为. 因为在 MapReduce 中,通常只有接收到 Map 阶段处理的所有数据,Reduce 阶段的处理过程方可开始,如果将一次传输定义为在任务的两个连续阶段中通过一组数据流,则整个任务的运行时间不是由单个流的持续时间决定,而是依赖于完成整个传输所花费的时间.

在分布式计算框架中,任务节点间的唯一通信过程发生在混洗传输阶段. 通过网络和计算资源联合实现网内关联性流量的聚合,能够极大降低对稀缺的东西向网络资源的消耗,避免 MapReduce 作业在混洗阶段产生太长的等待延迟. 由于"多对多"混洗传输由一组互不相关的"多对一"incast 传输组成,因此,我们通过优化 incast 传输实现高效混洗传输. 本文利用数据中心网络的拓扑特性建立高效的 incast 数据汇聚树,将混洗和汇聚并行化,在传输数据的同时聚合数据流,从而减少网络中传输的数据量并提高网络性能. 然而,最小代价 incast 树的构造是个 NP 难问题,最近提出的算法 IRS-based[13] 在 incast 成员节点的推算上仍具有一定的随机性,导致各层新增许多不必要的汇聚节点,从而增加了数据流和链路代价. 为了解决上述问题,本文首先提出了两种分别适用于不同场景的最小代价 incast 树构建算法 MIB-based 和 MC-based. MIB-based 算法针对发送节点强关联的情况,使高层发送节点尽可能汇聚到已有的低层发送节点构建 incast 树;MC-based 算法针对发送节点松散关联的情况,将节点进行最大程度上的聚合,通过增加最少的汇聚点完成 incast 树的构建. 将上述两种算法结合起来进一步提出了 M2-based 算法,在任何场景下能够在更大程度上减小网络中传输的数据量,且仍然能够在较短时间内完成数据中心的混洗传输. 同时,本文提出的方法同样适用于以交换机为中心的 FBFLY 和 HyperX 等基于 Generalized hypercube[14] 的网络结构.

2 研究背景及相关工作

2.1 数据中心网络结构

现有的数据中心网络主要依靠交换机、汇聚交换机、核心交换机/路由器将服务器连接起来构成树形结构,然而树形结构的高带宽收敛比(oversubscription)使其很难达到数据中心网络所追求的高可扩展、容错性好、高聚集带宽等目标. 因为树型结构的高层核心交换机/路由器构成网络的流量瓶颈,所以在扩展系统时,往往需要更换为更高端的交换机;而且树型结构的容错性也不理想,容易出现单点故障;与此同时,许多应用服务,如搜索引擎等对服务器间数据交换的带宽要求越来越高,因此,研究人员针对传统数据中心网络结构的固有缺陷,提出了一些新的架构设计方案. 目前提出的数据中心网络结构主要分为两大类,即以交换机为中心的结构和以服务器为中心的结构.

在以交换机为中心的结构中,网络连接和路由功能主要由交换机完成. Fat-Tree[15]、VL2 和 PortLand 中交换机彼此互连成各种树形结构,服务器仅通过一个 NIC 端口与接入层交换机相连,这种结构通过在树形结构上层横向增加更多的交换机来提供网络冗余,故称之为以交换机为中心的类树结构. 为克服类树结构的固有弊端,研究人员又提出了两种以交换机为中心的扁平结构 FBFLY[16] 和 HyperX[17],其基本思想是将高基交换机互连成一个 Generalized Hypercube 结构,每个交换机的剩余端口连接一些服务器. 由于所有链路和交换机被公平使用,所以与类树结构相比扁平结构本身不存在性能瓶颈.

在以服务器为中心的结构中,主要的互连和路由功能由服务器完成. 这类拓扑结构中,服务器通常采用多个 NIC 端口接入并连接网络,使得网络具有大量冗余的链路和平行路径以支持各种类型的流量模式. 他们或者仅将交换机作为类似于 crossbar 的交换功能使用,如 DCell、BCube、FiConn[18] 和 HCN[19];或者不使用任何交换机构建网络,如 CamCube[20].

在上述两类结构中,FBFLY 和 BCube 最具代表性. FBFLY 是一个利用高基数交换机构建的易扩展、低直径的多维直接网络. FBFLY 中的所有交换机在各个维上与其他所有交换机互连,剩余端口连接服务器,任何两个服务器之间不直接相连. k 元 n 维的 FBFLY 同构于 k 元 n 立方网络 torus,不同的是,torus 中各维互连成一个环形结构,而 FBFLY 中各

维全互连. 例如，FBFLY(c,k,n)网络中共包括 ck^{n-1} 台服务器和 k^{n-1} 个 k 口交换机，每个交换机连接的服务器个数为 c. BCube 针对模块化数据中心设计，其典型规模为 1 K～4 K. BCube 结构以迭代方式构建，BCube(n,k)同构于 n 元 $k+1$ 维的 Generalized Hypercube 结构，不同的是，Generalized Hypercube 结构中的所有邻居服务器直接相连，而 BCube 中的邻居服务器之间均通过交换机相连. BCube(n,0)由 n 个服务器连接一个 n 口交换机构成，BCube(n,k)由 n 个 BCube_{k-1} 连接 n^k 个 n 口交换机构成. 每个 BCube(n,k)具有 $k+1$ 个网络端口. 具有多个网络端口的服务器连接到多个层次的小型交换机，任何两个服务器之间没有直接连接. 图 1 所示为 BCube(4,1)结构，包括 16 台服务器和两层交换机.

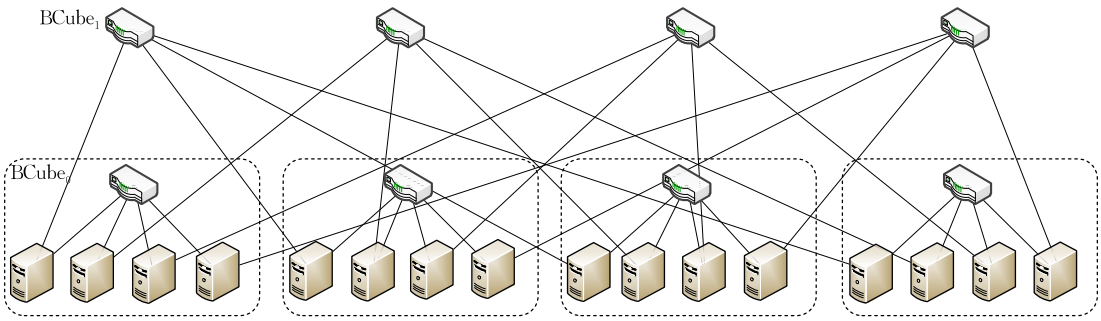


图 1　BCube(4,1)结构

2.2　数据中心网内数据汇聚

在许多分布式计算应用中，数据在接收端汇聚，输出数据的大小仅是输入数据的一小部分. 以 MapReduce 为例，在混洗传输阶段，从所有发送节点传输到某相应接收节点的数据流是高度相关的，也就是说，对每个 incast 传输，相同接收节点的 key/value 对拥有相同的 key 值，因此，通常在接收端应用一个汇聚函数（*sum*，*maximum*，*minimum*，*count*，top-k 和 KNN）把具有相同键的中间数据合并生成结果数据. 研究表明在 Google 的 MapReduce 中，平均输出数据占中间数据集大小的 40.3%[1]，同样的应用在 Facebook 中，从中间数据集到输出数据的数据量减少了 81.7%[21]，而在 Yahoo 中，数据量更是减少了 90.5%[21]. 这些数据说明在接收端执行数据汇聚操作能够缩减数据量的规模，受此启发，如果在混洗阶段执行相同的数据汇聚操作，还能够进一步减少网络中的流量，如图 2 所示.

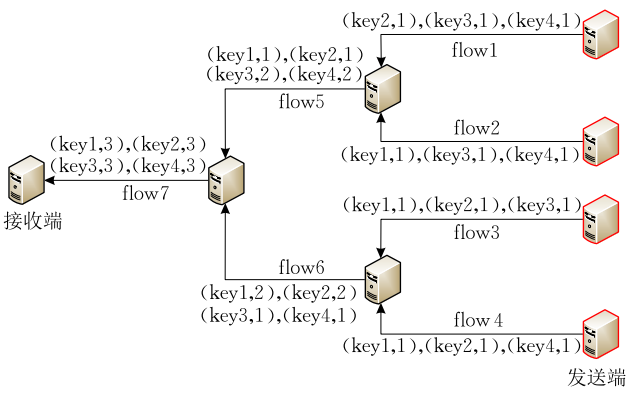


图 2　Incast 传输阶段构建数据汇聚树的示意图

特别是在以服务器为中心的数据中心网络中，服务器可使用集成有交换机芯片的 PCI 网卡 ServerSwitch，他不但具备传统交换机的能力而且能通过高速 PCI 口实现 CPU 和 ServerSwitch[22] 的高速互联，能借助服务器强大的计算能力甚至存储能力实现对网络流量的深入分析处理和对数据流的网内存储、聚合等功能.

然而当前数据中心的网内关联性流量聚合方面的研究工作依然很欠缺. Costa 等人[23] 针对 CamCube 结构设计了一个类 MapReduce 的系统 Camdoop，在 Camdoop 中，服务器使用 *getParent* 函数计算 incast 树拓扑，给定接收节点 R 和一个发送节点（或中间节点）S，*getParent* 函数返回 S 的 6 个邻居中距离 R 最近的一个，因此，在一棵 incast 树中，由每个发送节点传输的数据流各自沿着不相关的路由路径发送，类似于基于单播汇聚树传输的方法，这种方法难以实现中间数据流的有效汇聚，从而很难有效减少网络流量；Guo 等人针对 incast 汇聚传输提出了一个近似的优化算法 IRS-based. 在 incast 树中，每一个中间节点都有若干流输入和一个流输出，较少的中间节点则意味着较少的数据流，因此，IRS-based 算法试图在每一层中寻找包含最少节点的服务器集合. 然而，该算法在逐层推算汇聚点时没有考虑低层已有的发送节点，也就是说，IRS-based 算法通过对上一层节点的同一维进行调整来生成下一层节点，没有考虑上一层节点与位于下层的已有节点直接聚合的可能，从而导致新增一些不必要的节点. 为此，本文提出了一种新的计算最小代价 incast 树

的高效算法 M2-based，该算法充分考虑高层发送节点能够直接汇聚到已有的低层发送节点的情况，仅根据已知拓扑结构和节点编号即可快速生成一棵近似的最小代价 incast 树.

相对于 IRS-based 算法，M2-based 占用更少的数据中心资源，且能够在较短时间内进一步减小网络中传输的数据量.

3 数据中心关联性流量的网内聚合算法

3.1 问题描述

给定发送节点集 $\{S_1, S_2, \cdots, S_m\}$ 和接收节点 R，在 incast 传输阶段，从各发送节点到 R 的消息路由本质上形成一棵汇聚树. 在富连接数据中心网络(如 BCube)中存在许多棵路由成本不尽相同的汇聚树，我们面临的挑战是如何构建一棵使网络内传输的数据量尽可能最小的汇聚树，即最小代价 incast 树. 假设数据中心网络结构以图 $G=(V,E)$ 表示，其中，V 是网络中节点的集合，E 是连接节点的边的集合. 节点对应数据中心的交换机或服务器，边 (u,v) 定义一条从 u 到 v 的链路，则最小代价 incast 树的建立问题可形式化描述为在 $G=(V,E)$ 中寻找一个包括所有发送节点集和接收节点的最小代价连通子图，使得该连通子图上的链路代价之和最小.

以 BCube 结构为例，给出如下定义.

定义 1. 在 BCube(n,k) 中，如果两台服务器的编号 $x_k x_{k-1} \cdots x_1 x_0$ 和 $y_k y_{k-1} \cdots y_1 y_0$ 仅在第 $j(j \in [0,k])$ 维不同，则称它们为 j 维 1 跳邻居服务器，其中，$x_i, y_i \in \{0,1,\cdots,n-1\}$，$i \in [0,k]$. 每台服务器在每个维度都有 $n-1$ 个 1 跳邻居服务器，且所有 j 维 1 跳邻居服务器共同连接到第 j 层交换机上，因此，将任意两台编号在 j 个维度都不相同的服务器称为 j 跳邻居服务器.

令接收节点编号为 $r_k r_{k-1} \cdots r_1 r_0$，一个发送节点的编号为 $s_k s_{k-1} \cdots s_1 s_0$，其中 $r_i, s_i \in \{0,1,\cdots,n-1\}$，$i \in [0,k]$，则接收节点和每个发送节点间的海明距离最多为 $k+1$. 因此，所有发送节点和接收节点间的最短路径组成一个具有 $k+2$ 层的多层有向图，如图 3 所示. 第 0 层仅包括接收节点；第 $k+1$ 层仅包括发送节点；第 $j(j \in [1,k+1])$ 层的服务器节点是接收节点的 j 跳邻居. 一般地，只有发送节点和接收节点无法形成一个确定的连通子图，即无法构建一棵 incast 树，那么，如何在第 k 层到第 1 层中适当地选择一些中间节点使 incast 树的代价最小成为解决问题的关键. 文献[13]已经证明在 BCube 网络中建立最小 incast 树问题是一个 NP 难问题，因此，本文的目标是寻找一种高效的近似算法，仅根据已知拓扑结构以及发送节点和接收节点的编号即可快速生成一棵近似的最小代价 incast 树.

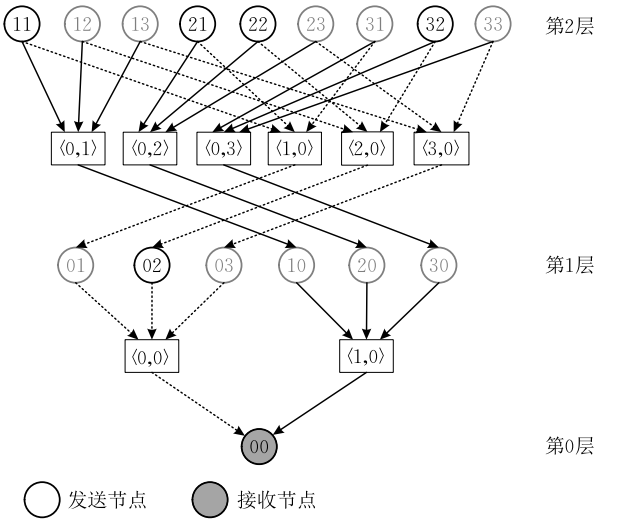


图 3 BCube(4,1)结构中，发送节点 11、21、22、32、02 与接收节点 00 构成 3 层有向图

3.2 MIB-based 算法

在 incast 树中，每新增一个汇聚节点，所有经过该节点的输入流将汇聚为一个输出流，那么，汇聚节点数量越少，输出的流的数量则越少，从而网络中传输的数据流的总数越少. 因此，一个近似的优化策略为使高层发送节点尽可能汇聚到已有的低层发送节点，即尽量使各层中除发送节点外的新增汇聚节点数最少. 首先给出如下定义.

定义 2. 发送节点共有 $l(l \in [1,k+1])$ 层，节点 A 位于第 $l'(l' \in [1,m))$ 层，则将第 $l'+1$ 层到第 l 层中与节点 A 的编号依次相差 $j(j \in [1, l-l'])$ 维的发送节点称为节点 A 的相关发送节点，其中，位于第 $l''(l'' \in [l'+1, l])$ 层的节点称为节点 A 的第 $l''-l'$ 层相关发送节点.

定义 3. 由节点 A 及其相关发送节点构成的节点集称为一组 incast 树分支.

定义 4. 在一组 incast 树分支节点中，如果除最低层外的其他节点都不能再形成新的 incast 树分支，则称该 incast 树分支为最小 incast 树分支.

例如，图 3 中 11 和 21 是 01 的第 1 层相关发送节点，32 是 02 的第 1 层相关发送节点，因此节点集 {11,21,01} 和 {32,30} 构成 2 组 incast 树分支，且均为最小 incast 树分支. 由上述定义可知，最小 incast 树分支由一系列相关发送节点构成，位于不同层次的相关发送节点之间其编号相差的位数即为层次

差.因此,构建最小 incast 树分支的过程即为从最低层发送节点开始,自底向上寻找相关发送节点集的过程,如算法 1 所示.

算法 1. *BuildIncastMinimalBranch* 函数.

输入:接收节点 R,第 l 层第 x 个发送节点 $S_l[x]$

输出:第 l 层第 x 个发送节点的最小 incast 树分支数组 $I_l[x]$

```
1.  IF S_l>0
2.    THEN FindMutualNode(R,S)
3.      FOR i←0 to I_l.length DO
4.        BuildIncastMinimalBranch(R,I_l)
FindMutualNode(R,S)
1.  FOR i←0 to S_l.length DO
2.    FOR j←0 to S_l'.length DO
3.      IF S_l[i]是 S_l'[j]的(l'-l)-跳邻居;
4.        将发送节点 S_l'[j]添加到 I_l[i];
```

在算法 1 中,从给定发送节点集的最低层起,对每一个节点逐层查找其相关发送节点,这些具有高度相关性的发送节点构成一组 incast 树分支,并存入相应节点的 incast 树分支数组,最终将给定发送节点集分割成若干组最小 incast 树分支.因此,在每一组最小 incast 树分支中,如何通过增加最少的汇聚节点将这些具有高度相关性的发送节点连接起来,即将最小 incast 树分支构成一棵最小代价 incast 树分支成为解决问题的关键.定理 1 给出最小 incast 树分支中位于各个层次的发送节点的编号规律和汇聚关系.

定理 1. 在一组最小 incast 树分支中,假设接收节点为 $X(0)$,第 $l(l\in[1,k+1])$ 层中的服务器节点表示为 $X(l)$,有

(1) 当 $l'<l$ 时,$X(l')$ 中有 $l'-l$ 个维度与 $X(l)$ 不同.

(2) 当 $l'>l$ 时,若 $X(l)$ 与 $X(0)$ 在某一维不同,$X(l')$ 与 $X(l)$ 必然在该维相同.

(3) 当 $3\le l_a=l_b\le k+1, l_a,l_b>l+1$,若 $X(l_a)$ 和 $X(l_b)$ 在与 $X(l)$ 不同的维度上有 $j(j\in[1,l_a-l])$ 维相同,则 $X(l_a)$ 与 $X(l_b)$ 在第 $j+l$ 层可以汇聚到同一个节点;否则不能在第 $l+1\sim l_a-1$ 层实现汇聚.

(4) 当 $3\le l_a\ne l_b\le k+1, l_a>l_b>l$,若二者在与 $X(l)$ 不同的维度上有 $j(j\in[1,l_b-l])$ 维相同,则 $X(l_a)$ 与 $X(l_b)$ 在第 $j+l$ 层可以汇聚到同一个节点;否则不能在第 $l+1\sim l_b$ 层实现汇聚.

证明. (1) 由定义 1、2 可知,对一组 incast 树分支节点,$X(l')$ 是 $X(1)$ 的 $l'-1$ 跳邻居节点,$X(l)$ 是 $X(1)$ 的 $l-1$ 跳邻居节点,因此,$X(l')$ 中有 $l'-l$ 个维度与 $X(l)$ 不同,得证;(2) 已知 $X(l)$ 中有 l 个维度与 $X(0)$ 不同,假设 $X(l')$ 与 $X(l)$ 在这 l 个相应维不同,因为 $X(l')$ 中共有 $l'-l$ 个维度与 $X(l)$ 不同,所以除了这 l 个维度之外,$X(l')$ 中还有 $l'-2l$ 个维度与 $X(l)$ 不同,因此 $X(l')$ 中共有 $l'-l$ 个维度与 $X(0)$ 不同,而实际上,$X(l')$ 是 $X(0)$ 的 l' 跳邻居节点,故 $X(l')$ 中应有 l' 个维度与 $X(0)$ 不同,与原假设矛盾,故原假设不成立,$X(l')$ 与 $X(1)$ 必然在相应维相同,得证;(3) 因为 $X(l_a)$ 有 l_a-l 个维度与 $X(l)$ 不同,又 $X(l_a)$ 和 $X(l_b)$ 在与 $X(l)$ 不同的维度上有 j 维相同,所以 $X(l_a)$ 和 $X(l_b)$ 有 l_a-j-l 维不同,即 $X(l_a)$ 与 $X(l_b)$ 各自需调整 l_a-j-l 维可实现汇聚,故在第 $j+l$ 层可以汇聚到同一个节点.反之,若 $X(l_a)$ 和 $X(l_b)$ 在与 $X(l)$ 不同的维度上均不相同,即 $X(l_a)$ 与 $X(l_b)$ 各自需调整 l_a-l 维可实现汇聚,故只能在第 l 层汇聚到 $X(l)$,即不能在第 $l+1\sim l_a-1$ 层实现汇聚,得证;(4) 因为 $X(l_a)$ 有 l_a-l 个维度与 $X(l)$ 不同,$X(l_b)$ 有 l_b-l 个维度与 $X(l)$ 不同,又二者在与 $X(l)$ 不同的维度上有 j 维相同,所以 $X(l_a)$ 与 $X(l_b)$ 分别需调整 l_a-j-l 维和 l_b-j-l 维即可实现汇聚,故二者在第 $j+l$ 层可以汇聚到同一个节点.反之,若 $X(l_a)$ 和 $X(l_b)$ 在与 $X(l)$ 不同的维度上均不相同,即 $X(l_a)$ 与 $X(l_b)$ 分别需调整 l_a-l 维和 l_b-l 维可实现汇聚,故只能在第 l 层汇聚到 $X(l)$,即不能在第 $l+1\sim l_b$ 层实现汇聚,得证. 证毕.

由定理 1 可以很容易地推算出各层发送节点的下一跳节点,如推论 1.

推论 1. 在一组最小incast树分支中,令第 $l(l\in[1,k+1])$ 层节点为 $X(l)=l_k l_{k-1}\cdots l_{e_1}l_{e_2}\cdots l_{e_j}\cdots l_0$,$X(l')=l'_k l'_{k-1}\cdots l'_{e_1}l'_{e_2}\cdots l'_{e_j}\cdots l'_0$ 和 $X(l'')=l''_k l''_{k-1}\cdots l''_{e_1}l''_{e_2}\cdots l''_{e_j}\cdots l''_0$ 分别是 $X(l)$ 的第 $l'-l$ 层和第 $l''-l$ 层相关发送节点.若 $X(l')$ 和 $X(l'')$ 在与 $X(l)$ 不同的维度上有 $j(j\in[1,\min(l',l'')-l])$ 维相同,即 $\begin{cases} l'_{e_1}=l''_{e_1}\ne l_{e_1} \\ l'_{e_2}=l''_{e_2}\ne l_{e_2} \\ \cdots \\ l'_{e_j}=l''_{e_j}\ne l_{e_j} \end{cases}$,那么 $X(l')$ 与 $X(l'')$ 可分别通过逐一调整除 j 维的其他任意维度为 $X(l)$ 的相应维得到其下一跳节点,并在第 $j+l$ 层汇聚到节点 $X(j+l)=l_k l_{k-1}\cdots a_{e_1}a_{e_2}\cdots a_{e_j}\cdots l_0$.汇聚点 $X(j+l)$ 可通过逐一调整维度 $l'_{e_1}l'_{e_2}\cdots l'_{e_j}$ 为 $X(l)$ 的相应维 $l_{e_1}l_{e_2}\cdots l_{e_j}$ 得到其下一跳节点,将所有节点相连则构成一棵以 $X(l)$ 为根节点的最小代价 incast 树分支.

推论 1 给出了将最小 incast 树分支自顶向下构建最小代价 incast 树分支的方法.其中下一跳节点

的生成方法使得位于高层的发送节点尽可能汇聚到已有的低层发送节点，最大程度上确保各层新增的汇聚节点数最少. 假设最高层为 H，共有 m_H 个发送节点，距离最高层最近的低层发送节点位于第 $l(l\in[1,H])$ 层，那么根据上述推论可以构建出以 $X(l)$ 为根，以 m_H 个发送节点为叶子节点的部分最小代价 incast 树分支；同理，可以继续构建出以距离 $X(l)$ 最近的低层发送节点 $X(l')(l'\in[1,l))$ 为根，以 $X(l)$ 为叶子节点的部分最小代价 incast 树分支，直到形成一棵完整的最小代价 incast 树分支. 我们将这种方法命名为 MIB-based(Minimal Incast Branch based)汇聚树构建方法.

3.3 MC-based 算法

针对多层次、强关联的发送节点，采用自底向上推算相关发送节点，自顶向下建立汇聚路径的方法构建最小代价 incast 树. 那么，对于不能构成一组 incast 树分支的节点将通过寻找各层节点之间的汇聚关系将节点进行最大程度上的合并，即针对松散关联的发送节点，我们采用自顶向下聚类并建立汇聚路径的方法，通过计算最小聚类节点组(Minimal Clustering Node Groups，MCNG)使高层节点逐层合并到低层节点构建最小代价 incast 树. 首先给出最小聚类节点组的定义.

定义 5. 将发送节点按照如下要求进行分组，即组内成员之间互为 $j(j\in[1,k+1])$ 跳邻居且各组之间无交集，使得组合化程度最高而分组数目最小，由此形成的节点组定义为最小聚类节点组.

最小聚类问题(Minimal Clustering Problem)也是一个 NP 难问题，本文提出了一种近似的优化算法，如算法 2 所示. 令图 $G'=(V',E')$ 中 V' 表示所有待聚类发送节点，对任意两个节点 $u,v\in V'$，若 u 和 v 互为 $j(j\in[1,k+1])$ 跳邻居，则存在 $(u,v)\in E'$. 首先，计算图 G' 中所有节点的度并找出具有最大度的节点 u，由于 u 和不同的邻居节点可能在不同的维度上相差 j 维，因此计算出在相同维度上相差 j 维的邻居节点的最大集合构成一组；然后，从图 G' 中将该组内的所有节点和相关的边移除，对于剩下的节点重复以上过程直到图 G' 为空. 最终，该算法将所有待汇聚的发送节点分成若干无交集的组.

算法 2. *MinClustering* 函数.

输入：图 $G'=(V',E')$

输出：最小聚类节点组

1. *Groups*={};
2. WHILE $G'\neq\varnothing$ DO
3. 计算 V' 中每个节点的度；
4. 在 G' 中寻找具有最大度的节点 u；
5. 将 u 及其同一维度的 j-跳邻居构成一组，并加入 *Groups*；
6. 从 G' 中移除该组中的所有元素及其相关边；

因此，在由非 incast 树分支组成的节点集中构建最小代价 incast 树的方法为，首先，在第 $k+1$ 层的所有节点中寻找 k 维相同的 MCNG，得到位于第 k 层的汇聚节点；第二，在第 $l(l\geq k)$ 层的剩余节点中寻找在与 $X(0)$ 不同维度上 $k-1$ 维相同的 MCNG，得到位于第 $k-1$ 层的汇聚节点；在第 $l(l\geq k-1)$ 层的剩余节点中寻找在与 $X(0)$ 不同维度上 $k-2$ 维相同的 MCNG，得到位于第 $k-2$ 层的汇聚节点；以此类推，直到在第 $l(l\geq 2)$ 层的剩余节点中寻找在与 $X(0)$ 不同维度上 1 维相同的 MCNG，得到位于第 2 层的汇聚节点；第三，对从第 $k+1$ 层到第 2 层的所有不能汇聚的节点逐一调整相应维度为 $X(0)$ 的相应维得到其下一跳节点，最后，将所有节点相连则构成一棵以 $X(0)$ 为根节点的最小代价 incast 树. 我们将这种构建最小代价 incast 树的方法命名为 MC-based(Minimal Clustering based)汇聚树构建方法.

3.4 M2-based 算法

在实际应用中，发送节点的分布往往具有随机性，因此，需要把 MIB-based 和 MC-based 结合起来共同构建一棵最小代价 incast 树，我们将这两种方法统称为 M2-based. 综上所述，给定发送节点集和接收节点，最小代价 incast 树的构建过程分为以下 5 步：

步骤 1. 根据发送节点与接收节点的编号计算出每个发送节点在 incast 树中的层次. 即位于第 $j(j\in[1,k+1])$ 层的发送节点是接收节点的 j 跳邻居，它们的编号相差 j 维.

步骤 2. 从最低层发送节点开始，自底向上逐层构建 incast 树分支. 假设最低层发送节点为第 l 层，节点个数为 m_l，对第 l 层中的每一个发送节点，自底向上查找其相关发送节点，构成 m_l 组 incast 树分支. 对余下各层中的剩余发送节点依次采用相同的方法构建 incast 树分支.

步骤 3. 对每一组 incast 树分支，再次应用上一步中的方法递归构建 incast 树分支，直到所有 incast 树分支均为最小 incast 树分支. 对每一组最小 incast 树分支，可根据推论 1 直接构建出最小代价 incast 树分支.

步骤 4. 对没有构成 incast 树分支的剩余节点则通过计算 MCNG 自顶向下构建最小代价 incast

树分支.

步骤 5. 将位于第 1 层的发送节点与接收节点相连得到最终的最小代价 incast 树.

例如，令接收节点 $R=000$，发送节点集为{002，003，010，011，031，121，202，211，221，300，301，321，322，323}. 根据发送节点与接收节点的编号将发送节点分为 3 层，分别为 $l_1=\{002,003,010,300\}$，$l_2=\{011,031,202,301\}$，$l_3=\{121,211,221,321,322,323\}$. 对第 l_1 层到第 l_3 层的每一个节点，自底向上查找其相关发送节点，构成 7 组 incast 树分支，如图 4(a)所示. 对第一组 incast 树分支递归构建最小 incast 树分支，由此形成 8 组最小 incast 树分支，如图 4(b)所示. 对第 1、2 组最小 incast 树分支可根据推论 1 直接构建出其最小代价 incast 树分支，如图 4(c)所示. 对没有汇聚到第 l_1 层的剩余节点{031，121，221}自顶向下逐层计算 MCNG 并合并构建最小代价 incast 树，如图 4(d)所示. 最后将位于第 1 层的发送节点与接收节点相连得到一棵最小代价 incast 树.

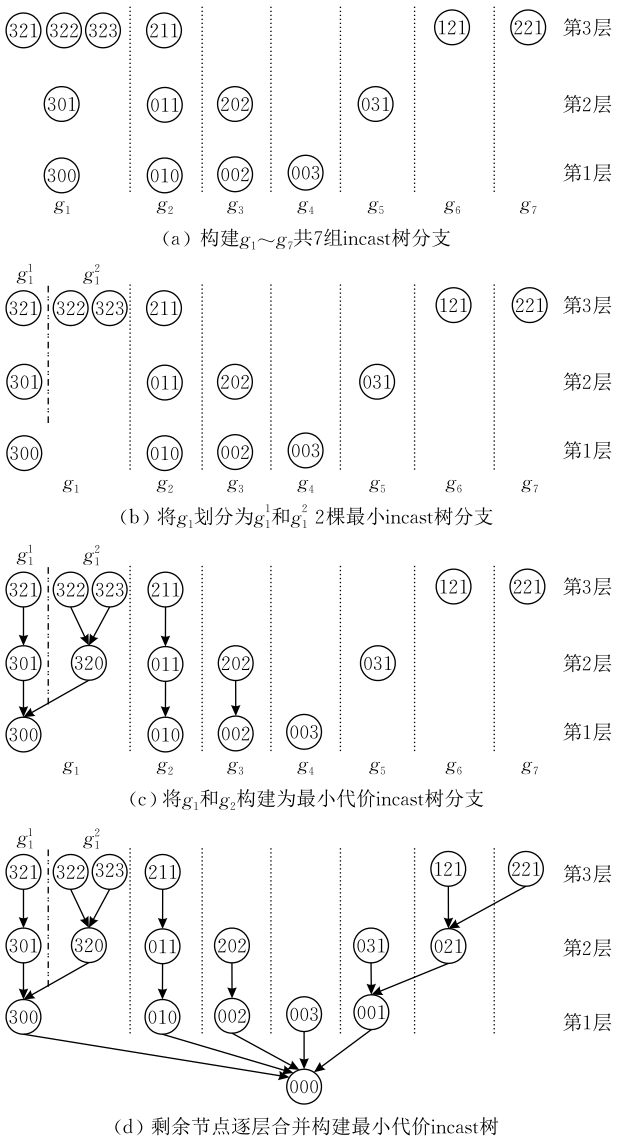


图 4　BCube(4，1)结构中最小代价 incast 树的构建过程

3.5　算法复杂性分析

假设 incast 树中所有边的权重为 1，树的代价为所有边权重之和，则最小代价 incast 树的构建问题可等效为 Steiner 树问题，是一个 NP 难问题，目前已经提出的在一般图中构建 Steiner 树的近似算法其时间复杂度为 $O(mN^2)$，m 表示发送节点的个数，N 表示网络规模. 然而，大规模商用数据中心通常拥有成千上万台服务器，这一较高的复杂度显然无法满足在线构建 incast 树的需要，而且这些算法也无法高效利用诸如 BCube 等富连接网络的拓扑特性. 本文提出的 M2-based 算法利用 BCube 的多等价路径特性构建最小代价 incast 树大大降低了通用算法的复杂度.

定理 2.　假设 incast 传输由 m 个发送节点组成，位于第 l 层的发送节点的个数为 $m_l(l\in[1,k+1])$，则 $m=m_1+m_2+\cdots+m_{k+1}$，那么最小代价 incast 树构建算法的时间复杂度为 $O(m^2(\lg m+\lg N))$.

证明.　最小代价 incast 树构建算法的计算量主要花费在 2 个阶段，即自底向上计算最小 incast 树分支(算法 1)和对不能构成 incast 树分支的剩余节点计算 MCNG(算法 2). 第 1 阶段中，从第 1 层到第 $k+1$ 层逐层递归构建最小 incast 树分支，形成一棵深度为 $O(\lg m)$ 的递归树，其中，每一层的代价为 $O(m_j\times(m_{j+1}+m_{j+2}+\cdots+m_{k+1}))=O(m_j\times(m-m_1-m_2-\cdots-m_j))=O(m^2)$. 因此，第 1 阶段的时间复杂度为 $O(m^2\lg m)$. 第 2 阶段中，逐层计算 MCNG 的时间复杂度为 $O((m_{k+1}+m_k+\cdots+m_j)^2)=O(m^2)$，最多需要计算 k 次，因此，第 2 阶段的时间复杂度为 $O(km^2)=O(m^2\lg N)$，综上，总的时间复杂度为 $O(m^2(\lg m+\lg N))$.　　证毕.

4　讨　论

4.1　扩展到其他网络结构

在以交换机为中心的结构中，传统交换机不提供可编程数据层，而软件定义的网络技术仅支持控制层的可编程. 网内汇聚要求网络设备能够缓存和处理数据流. 因此，将用户定义的汇聚函数放在传统交换机上是不可行的，以交换机为中心的结构不能直接支持网内汇聚. 近年，新型的思科 ASIC 和 arista

应用交换机提供可编程的数据层,如果数据中心利用这种基于软件的或基于 FPGA 的新型交换机,在以交换机为中心的结构中实现网内汇聚将成为可能.在以交换机为中心的结构中构建最小 incast 树的工作将在下一步开展.

目前,以服务器为中心的结构已经能够支持数据中心的网内汇聚.尽管本文基于 BCube 结构进行研究,文中提出的方法仍可应用到其他以服务器为中心的结构中.然而,在不同的网络结构上构建 incast 树需要利用不同结构的不同拓扑特性,关于在其他以服务器为中心的结构上的汇聚传输策略也将作为我们后续工作之一.

如果采用具有可编程数据层的新型交换机,本文提出的方法可直接应用于以交换机为中心的 FBFLY 和 HyperX 结构,因为 BCube 拓扑本质上属于 Generalized Hypercube 结构,而 FBFLY 和 HyperX 在交换机的互连层面上也同构于 Generalized Hypercube.

4.2 对作业完成时间的影响

给定一个类 MapReduce 作业,其执行时间取决于 3 个阶段,即 map、混洗和 reduce.网内汇聚操作仅影响混洗和 reduce 阶段的完成时间,而对 map 阶段无影响.混洗阶段的完成时间主要取决于网络的通信量及可用的网络资源.

在数据中心中,类 MapReduce 的分布式计算框架存在 map 倾斜问题[24-28],由于不同 map 任务上负载分布的不均衡性,使得 map 任务的运行时间高度可变.当这种倾斜发生时,一些 map 任务会花费更长时间处理输入数据,从而延缓了整个作业的完成时间.同理,reduce 任务也存在类似问题.近年来,研究人员提出了许多方法解决或缓解 map 倾斜问题,这些策略与使用网内汇聚优化传输是正交的,因此,我们采用这些方法使所有 map 任务尽可能同时完成以更好地支持网内汇聚策略的应用.

在上述设定下,incast 传输中的每一个发送节点沿着 incast 树向接收节点传送数据流.当到达某汇聚节点,该流的所有数据包将被缓存,一旦有新的数据流到来,汇聚节点即可执行汇聚操作,即将所有 key-value 对根据 key 值分组,并对每组应用汇聚函数.如果汇聚节点上的可用缓存很小或者等待所有流到达的时间超出某个阈值,则生成一个新的数据流并继续沿着 incast 树向接收节点发送.原理上类似于文献[29]中的 MapReduce 延迟调度技术.因此,与目前不采用网内汇聚策略的方法相比,网内汇聚直接减少了混洗阶段传输的网络流量,从而缩短了混洗阶段的持续时间.

在这种方式下,由于存在拖后腿的 map 任务,最后到达的数据流可能无法在汇聚节点与其他流聚合,将以 cut-through 的方式转发,等同于现存的无网内汇聚的方法.因此,采用网内汇聚算法后,混洗阶段的完成时间在最坏情况下与现存方法相同.同时,与现存方法相比,网内汇聚策略使接收端接收的数据量大大减少,从而减少了接收端进行 reduce 计算的时间.综上,我们的方法不会增加混洗阶段的完成时间,且能够减少 reduce 计算的时间,因此,与现存方法相比能够减少整个作业的完成时间.

5 仿真实验

本节对 M2-based 算法与目前不采用网内汇聚的方案(以下简称现存方法)、单播算法以及 IRS-based 算法进行对比实验,通过调整网络规模和 incast 传输节点的规模评估如下 3 个指标:混洗传输的数据量即 incast 树中所有边流量的总和、汇聚服务器个数、新增服务器个数、incast 树中的有效链路个数以及接收端的数据量.由于实验条件所限,我们仅搭建了小型的测试床,在大规模真实网络上的验证工作拟留待下一步完成.

5.1 实验设计

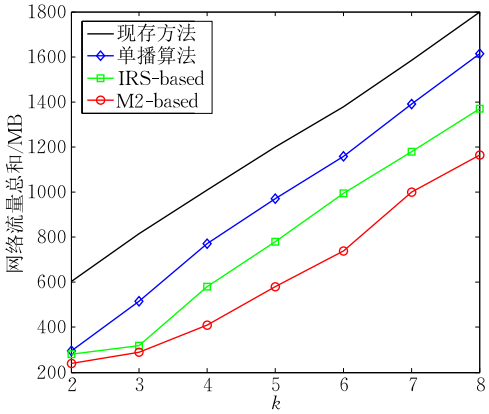
实验平台采用 3 台服务器通过 1 GB 以太网交换机互连.服务器的硬件配置包括一个 2 路 8 核 2.50 GHz Intel Xeon E5420 处理器,24 GB 内存和 1 TB SATA 硬盘.Hadoop 分布式实验环境由 3 台服务器上运行的 31 台 Red Hat 虚拟机构成,其中,1 台服务器上运行 11 台虚拟机,1 台为 Hadoop 的管理节点,10 台为数据节点;其余 2 台服务器上各运行 10 台虚拟机均作为 Hadoop 的数据节点.每个数据节点支持 4 个 map 任务和 1 个 reduce 任务.每台服务器使用一块 Intel 网卡,该服务器上的所有虚拟机通过虚拟交换机共享该网卡.

修改 Hadoop 使其支持网内数据包缓存和汇聚功能.MapReduce 作业采用 Hadoop 0.21.0 中的样例程序 *Wordcount*,包括 120 个发送节点(map 任务)和 1 个接收节点(reduce 任务).为每个 map 任务分配 10 个 64 M 大小的输入文件.在混洗阶段,执行中间结果本地汇聚后从每个发送端传输到接收端的平均数据大小为 1 M.在实际的数据中心网络中,map 任务往往被调度到空闲服务器执行 incast 传输,从而发送节点呈随机分布,因此,为所有发送节

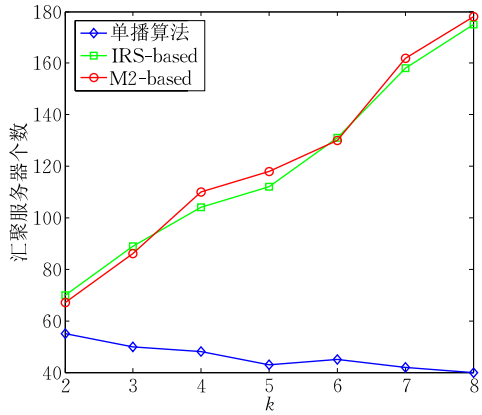
点和接收节点随机分配一个 BCube 编号，为了实现在 BCube(6,k)($2\leq k\leq 8$)网络上的 incast 传输，根据相应算法推算出中间节点，则所有节点共同构成一个 incast 连通子图，相当于一个部分的 BCube 网络. Incast 树中的节点与 31 台虚拟机的映射关系为：发送节点和接收节点分别映射到 30 台数据节点. 通过在每台虚拟机上设置代理软件模拟中间节点收发和存储数据包，将中间节点也映射到 30 台数据节点. 同时，为了确保执行 incast 传输时一台数据节点上的多个中间节点之间不发生本地通信，将任何一对在 incast 树的连续层次出现的邻居节点映射到不同的数据节点. 因此，incast 树中的每一条边映射为两台虚拟机之间的虚拟链路或服务器之间的物理链路.

5.2　网络规模的影响

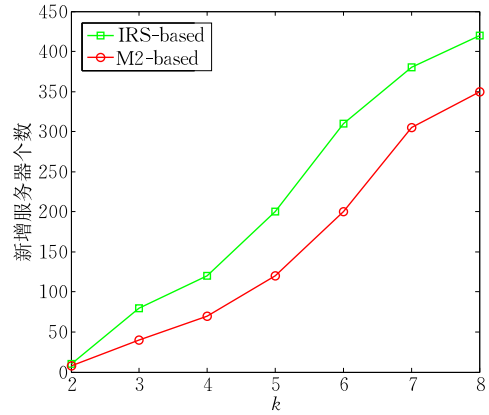
给定 incast 树，在 BCube(6,k)子网上部署具有 120 个发送节点和 1 个接收节点的 incast 传输. 实验分别采用 M2-based 算法、现存方法、单播算法和 IRS-based 算法生成相应的 incast 树，执行 100 次取平均结果，如图 5 所示.



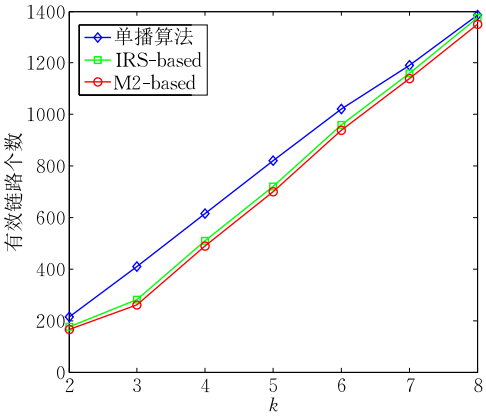
(a) 网络中传输的数据量



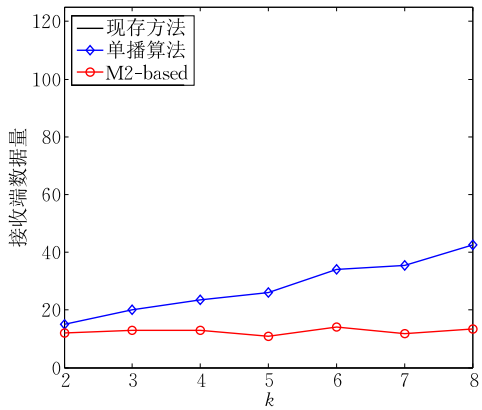
(b) 汇聚服务器的数量



(c) 新增服务器的数量



(d) 有效链路个数



(e) 接收端的数据量

图 5　incast 传输中，各评价指标随 BCube(6,k)网络的 k 值的变化趋势

图 5(a)表示不同网络规模下 incast 树中传输的流量的变化趋势. 与现存方法相比,M2-based 算法、单播算法和 IRS-based 算法均显著节省了网络中传输的流量,证明了关联性流量网内汇聚策略的有效性. M2-based 算法和 IRS-based 算法明显优于单播算法,一个重要原因是前两种算法中的汇聚点个数随 k 值的增大而增加,而单播算法中的汇聚点个数随着 k 值的增大而减少,如图 5(b)所示. M2-based 算法的效果更好,因为 M2-based 通过增加最少的节点完成 incast 树的构建,减少了输出的流的数量,从而减少了网络中传输的数据流的总量,如图 5(c)所示. 此外,M2-based 算法占用更少的链路,从而占用更少的服务器和网络设备,节约更多的网络资源,如图 5(d)所示. 同时,关联性流量的网内汇聚策略均极大地减少了接收端的数据量,如图 5(e)所示. 表 1 中的数据说明与现存方法相比,M2-based 明显减少了整个作业在混洗和 reduce 阶段的等待延迟.

表 1 网络规模对混洗和 reduce 阶段完成时间的影响

(单位:s)

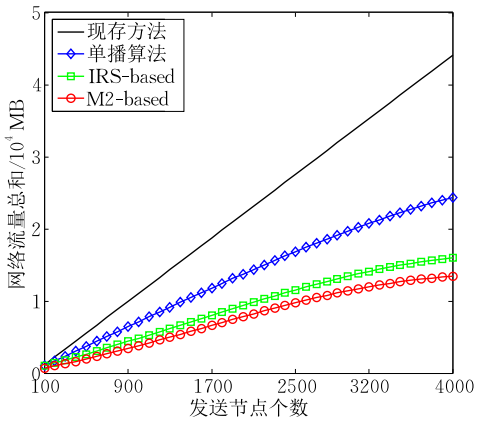
BCube(6,k)	现存方法	M2-based
$k=2$	15.3072	4.0374
$k=3$	15.3110	5.1632
$k=4$	15.3182	5.2543
$k=5$	15.3210	5.0372
$k=6$	15.3291	5.6022

综上所述,M2-based 算法支持发送节点为 120 的小规模 incast 传输,且生成更少的网络流量并占用更少的数据中心资源.

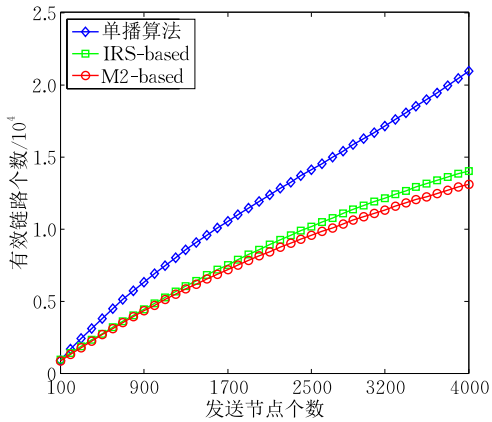
5.3 Incast 传输节点规模的影响

在现实的数据中心网络中,类 MapReduce 作业通常包括数百甚至上千个 map 任务,由于资源有限,现有测试平台无法运行大规模的词频统计程序,因此,我们通过模拟程序证明 M2-based 算法的扩展性. 令发送节点 $m\in\{100,200,\cdots,3900,4000\}$,通过 Hadoop 的样例程序 *RandomTextWriter* 为每个发送节点提供输入数据. 控制从每个发送节点传输到接收节点的平均数据量为 1 G. 图 6 为 BCube(4,8)网络中,incast 传输的数据量和有效链路数随发送节点数量的变化趋势. BCube(4,8)网络的规模为 262144,能够满足实际数据中心网络的规模要求.

结果表明,当发送节点的数量从 100 增长到 4000 时,与现存方法相比,M2-based 算法和单播算法均显著节省了网络中传输的数据量,证明了关联性流量网内汇聚策略在大规模 incast 传输中的有效性. 且在拥有 4000 个节点的 incast 传输中 M2-based 算法仍然优于其他两种方法. 图 6(b)表明 M2-based 算法占用更少的链路,从而占用更少的服务器和网络设备,节约更多的网络资源. 表 2 显示在大规模 incast 中,与现存方法相比,M2-based 仍然能够减少整个作业在混洗和 reduce 阶段的等待延迟.



(a) 网络中传输的数据量



(b) 有效链路个数

图 6 BCube(4,8)网络中,incast 传输的数据量和有效链路数随发送节点数量的变化趋势

表 2 网络规模对混洗和 reduce 阶段完成时间的影响

(单位:min)

m	现存方法	M2-based
500	15.61	6.03
1000	22.73	6.51
1500	29.55	6.82
2000	36.35	7.24
3000	55.15	7.46

综上所述,M2-based 算法能够支持较大规模的 incast 传输,且生成更少的网络流量并占用更少的数据中心资源.

6 结 论

在大规模分布式计算应用中,混洗传输阶段产

生的巨大网络流量给数据中心网络带来巨大压力，严重影响应用性能.在数据传输过程中执行网内关联性流量的聚合能够大大减少网络流量，提高通信性能.数据中心网络的数据汇聚可形式化描述为最小代价 incast 树的建立问题，然而，最小代价传输汇聚树的构建是一个 NP 难问题.针对这一问题，本文提出了一种构建最小代价传输汇聚树的算法 M2-based，仅根据已知数据中心网络的拓扑结构以及发送节点和接收节点的编号即可快速构建出一棵近似的最小代价 incast 树，实验结果表明，与现有方法相比，M2-based 能够占用更少的数据中心资源、在较短时间内进一步减小网络中传输的数据量.

参 考 文 献

[1] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters//Proceedings of the OSDI'04. Berkeley, USA, 2004: 27-39

[2] Isard M, Budiu M, Yu Y, et al. Dryad: distributed data-parallel programs from sequential building blocks//Proceedings of the EuroSys'07. New York, USA, 2007: 59-72

[3] Murray D G, Schwarzkopf M, Smowton C, et al. CIEL: A universal execution engine for distributed data-flow computing //Proceedings of the NSDI'11. Boston, USA, 2011: 227-236

[4] Malewicz G, Austern M H, Bik A J C, et al. Pregel: A system for large-scale graph processing//Proceedings of the SIGMOD'10, Indiana, USA, 2010: 135-146

[5] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster computing with working sets//Proceedings of the HotCloud'10. Boston, USA, 2010: 27-39

[6] Chowdhury M, Zaharia M, Ma J, et al. Managing data transfers in computer clusters with orchestra//Proceedings of the SIGCOMM'11. Toronto, Canada, 2011: 98-109

[7] Greenberg A, Hamilton J R, Jain N, et al. VL2: A scalable and flexible data center network. Communications of the ACM, 2011, 54(3): 95-104

[8] Guo C, Wu H, Tan K, et al. DCell: A scalable and fault-tolerant network structure for data centers//Proceedings of the SIGCOMM'08, New York, USA, 2008: 75-86

[9] Al-Fares M, Radhakrishnan S, Raghavan B, et al. Hedera: Dynamic flow scheduling for data center networks//Proceedings of the NSDI'10. Berkeley, USA, 2010: 19

[10] Chen Y, Griffith R, Liu J, et al. Understanding TCP incast throughput collapse in datacenter networks//Proceedings of the WREN'09. New York, USA, 2009: 73-82

[11] Mysore R N, Pamboris A, Farrington N, et al. PortLand: A scalable fault-tolerant layer 2 data center network fabric. ACM SIGCOMM Computer Communication Review, 2009, 39(4): 39-50

[12] Guo C, Lu G, Li D, et al. BCube: A high performance, server-centric network architecture for modular data centers. ACM SIGCOMM Computer Communication Review, 2009, 39(4): 63-74

[13] Deke G, Junjie X, Xiaolei Z, et al. Exploiting efficient and scalable shuffle transfers in future data center networks. IEEE Transactions on Paralled and Distributed Systems, 2015, 26(4): 997-1009

[14] Bhuyan L N, Agrawal D P. Generalized hypercube and hyperbus structures for a computer network. IEEE Transactions on Computers, 1984, C-33(4): 323-333

[15] Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture//Proceedings of the SIGCOMM'08. New York, USA, 2008: 63-74

[16] Kim J, Dally W J, Abts D. Flattened butterfly: A cost-efficient topology for high-radix networks//Proceedings of the ISCA'07. New York, USA, 2007: 126-137

[17] Ahn J H, Binkert N, Davis A, et al. HyperX: Topology, routing, and packaging of efficient large-scale networks// Proceedings of the SC'09. New York, USA, 2009: 41

[18] Li D, Guo C, Wu H, et al. Scalable and cost-effective interconnection of data-center servers using dual server ports. IEEE/ACM Transactions on Networking, 2011, 19(1): 102-114

[19] Guo D, Chen T, Li D, et al. BCN: Expansible network structures for data centers using hierarchical compound graphs//Proceedings of the INFOCOM. Picataway, USA, 2011: 61-65

[20] Abu-Libdeh H, Costa P, Rowstron A, et al. Symbiotic routing in future data centers. ACM SIGCOMM Computer Communication Review, 2010, 41(4): 51-62

[21] Yanpei C, Ganapathi A, Griffith R, et al. The case for evaluating MapReduce performance using workload suites// Proceedings of the MASCOTS'11. Singapore, 2011: 390-399

[22] Lu G, Guo C, Li Y, et al. ServerSwitch: A programmable and high performance platform for data center networks// Proceedings of the NSDI'11. Berkeley, USA, 2011: 2-17

[23] Costa P, Donnelly A, Rowstron A, et al. Camdoop: Exploiting in-network aggregation for big data applications//Proceedings of the NSDI'12. Berkeley, USA, 2012: 3

[24] Kwon Y, Balazinska M, Howe B, et al. Skew-resistant parallel processing of feature-extracting scientific user-defined functions//Proceedings of the SoCC'10. New York, USA, 2010: 75-86

[25] Ramakrishnan S R, Swart G, Urmanov A. Balancing reducer skew in MapReduce workloads using progressive sampling// Proceedings of the SoCC'12. New York, USA, 2012: 11-16

[26] Kwon Y, Balazinska M, Howe B, et al. SkewTune: Mitigating skew in MapReduce applications//Proceedings of the SIGMOD'12. New York, USA, 2012: 25-36

[27] Kwon Y, Ren K, Balazinska M, et al. Managing skew in hadoop. IEEE Data Engineering Bulletin, 2013, 36(1): 24-33

[28] Ousterhout K, Wendell P, Zaharia M, et al. Sparrow: Distributed, low latency scheduling//Proceedings of the SOSP'13. New York, USA, 2013: 69-84

[29] Zaharia M, Borthakur D, Sen Sarma J, et al. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling//Proceedings of the EuroSys'10. New York, USA, 2010: 265-278

LU Fei-Fei, born in 1981, Ph. D. candidate, engineer. Her current research interests include distributed computing and data center network.

GUO De-Ke, born in 1980, Ph. D., associate professor. His current research interests include distributed systems, data center network and wireless communication systems.

FANG Xing, born in 1980, Ph. D., assistant professor. His current research interests include computer architecture and high-speed analog circuit.

XIE Xiang-Hui, born in 1958, Ph. D., professor, Ph. D. supervisor. His current research interests include computer architecture, network and distributed computing.

LUO Xing-Guo, born in 1951, professor, Ph. D. supervisor. His current research interests include wireless communication, software defined radio and high performance computing.

Background

This work is supported by the National Natural Science Foundation of China under Grant No. 91430214 and the National High Technology Research and Development Program (863 Program) of China (No. 2013AA01A213). This project aims to make advances to the system of cloud computing. This paper focus on managing the network activity at the level of transfers so as to significantly lower down the network traffic and efficiently use the available network bandwidth in future data centers.

In-network aggregation is a very important aspect in wireless sensor networks and has been widely studied. However, there has been little work on it in the field of data centers. Costa et al. are the first to introduce the basic idea of such an in-network aggregation to the field of data centers by pushing aggregation computation from the edge into the network. They proposed a unicast-based tree approach. But the resultant incast tree could not achieve the desirable benefit of inter-flow data aggregation since the number of aggregating vertices in the tree is often very small. As a result, such kind of solutions cannot considerably save the network traffic of an incast transfer. Guo et al. modeled shuffle transfer as an incast minimal tree problem and proposed an approximate method named IRS-based, but this algorithm produces many redundant aggregating nodes which increase the traffic and the link cost. As a result, this algorithm is not applicable for all cases.

The main objective of this paper is to find the incast minimal cost tree. Compared with the existing methods, our method can decrease the amount of network traffic and significantly save the data center resources.