计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20190683
57(12):2683-2693,2020

一种分布式异构带宽环境下的高效数据分区方法

马卿云[1] 季航旭[1] 赵宇海[1] 毛克明[2] 王国仁[3]
[1](东北大学计算机科学与工程学院 沈阳 110169)
[2](东北大学软件学院 沈阳 110169)
[3](北京理工大学计算机学院 北京 100081)
(maqy1995@163.com)

An Efficient Data Partitioning Method in Distributed Heterogeneous Bandwidth Environment

Ma Qingyun[1], Ji Hangxu[1], Zhao Yuhai[1], Mao Keming[2], and Wang Guoren[3]
[1](*School of Computer Science and Engineering*, *Northeastern University*, *Shenyang* 110169)
[2](*Software College*, *Northeastern University*, *Shenyang* 110169)
[3](*School of Computer Science and Technology*, *Beijing Institute of Technology*, *Beijing* 100081)

Abstract A large quantity of data is transmitted through the network during the process in distributed big data processing framework, resulting in the time consumption for data transmission between each node becomes one of the main costs of the operation. However, in the case of heterogeneous bandwidth of nodes, traditional data partitioning methods such as Hash partitioning or range partitioning will be inefficient, due to the existence of bandwidth bottleneck nodes. Data partitioning is necessary for big data processing and inefficient data partitioning methods would significantly increase the running time of jobs. We therefore propose a data transmission model between nodes to reduce time consumption in distributed heterogeneous bandwidth networks. The model calculates each node's optimal data distribution ratio to minimize the data transfer time, according to its uplink and downlink bandwidth as well as the initial data size. Besides, a bandwidth-based data partitioning method is designed based on the proposed model, enabling each node to allocate data under the optimal data distribution ratio. We demonstrate the effectiveness of our bandwidth-based data partitioning method through the implementation in the Apache Flink framework and have significantly improved efficiency. Extensive experimental results show that the bandwidth-based data partitioning method can effectively reduce the time consumption of data partitioning in distributed heterogeneous bandwidth conditions.

Key words data partitioning; Apache Flink; load balancing; heterogeneous bandwidth; distributed system

摘 要 在分布式大数据处理框架的作业运行过程中,会有大量的数据通过网络传输,数据在各节点之间传输所需的时间已成为作业运行的主要开销之一.在节点异构带宽的情况下,因为带宽瓶颈节点的存在,传统的数据分区方法效率低下.针对这个问题,建立了节点间的数据传输模型,该模型以降低数据

收稿日期:2019-09-17;**修回日期**:2020-03-02
基金项目:国家重点研发计划项目(2018YFB1004402);国家自然科学基金项目(61772124)
This work was supported by the National Key Research and Development Program of China (2018YFB1004402) and the National Natural Science Foundation of China (61772124).
通信作者:赵宇海(zhaoyuhai@mail.neu.edu.cn)

传输时间为目标,根据各节点的上下行带宽和初始数据量大小,计算出各节点的最优数据分发比例.以该模型为基础,设计了基于带宽的数据分区方法,该数据分区方法使得各节点按最优数据分发比例来分配数据.最后在 Apache Flink 框架中将基于带宽的数据分区方法进行了实现,并通过实验进行了验证.实验结果表明:异构带宽条件下,基于带宽的数据分区方法可以有效减少数据分区所需的时间.

关键词 数据分区;Apache Flink;负载均衡;异构带宽;分布式系统

中图法分类号 TP311

随着物联网、移动互联网、产业互联网和社交媒体等技术的飞速发展,每天都会产生大量的数据,人们已经身处大数据时代[1].根据国际数据公司(International Data Corporation, IDC)的预测,到2025年,全球的数据量将是现在的10倍,达到175 ZB.

大数据中有着丰富的信息,并且蕴含着巨大的价值[2].谷歌通过用户搜索词频的变化成功对冬季流感进行了预测,沃尔玛通过分析消费者购物行为对纸尿裤和啤酒进行共同销售,这些耳熟能详的案例都印证了这一点.但随着数据产生速度的加快,数据量的急剧增长,如何对庞大的数据进行处理成为了新的难题.传统的单机处理已经无法满足大数据的需求,分布式的大数据处理框架应运而生.谷歌首先提出了用于大规模数据并行计算的编程模型MapReduce[3],引起了极大的反响,也因此促使了Hadoop[4]的诞生.之后为了改进传统 MapReduce 中运行效率低下的问题,基于内存计算的 Spark[5]被提出.时至今日,为了追求更快的处理速度、更低的时延,Flink[6]开始崭露头角,并得到了飞速的发展.

与此同时,随着云计算[7]的兴起,包括谷歌、微软、阿里巴巴等在内的互联网公司都提供了大数据存储与分析的相关服务,众多企业开始选择将自己的业务上"云".这些提供云服务的公司需要存储和处理的数据同样是海量的,为了更好地为客户提供服务,提供云服务的公司通常都会在各地建立数据中心[8],例如微软和谷歌在世界各地就分布着超过十个的数据中心.各数据中心之间经常需要联合进行数据分析,此时分布式大数据处理框架依然是不二之选.跨数据中心的大数据分析业务许多都是数据密集型作业,作业运行过程中,通常需要使用数据分区方法将相同键的数据发送到同一数据中心进行处理,而各个数据中心之间通常相隔较远,这样会产生大量的网络传输开销,导致数据在网络中的传输时间成为大数据分析作业的瓶颈.由于网络提供商硬件设备的不同,各数据中心之间的带宽通常差异较大,这样便会形成异构带宽的分布式环境[9].当然,即使在同构的集群中,也可能因为某些节点上的作业抢占了带宽而导致集群环境中各节点带宽异构.综上所述,在异构带宽环境下,如何高效地进行数据分区是一个急需解决的问题.

数据分区是大数据框架的一个基本功能,通过数据分区可以将各分区数据交给不同的节点进行处理.常用的数据分区方式有随机分区、Hash 分区和 Range 分区[10].其中 Hash 分区和 Range 分区都能保证具有相同键的数据分发到同一节点,这也为许多需要这种保证的算子提供了保障.现有的研究很少在数据分区时对节点的带宽进行考虑,在节点间异构带宽的情况下,传统的数据分区方法效率低下,完成数据分区的时间开销较大.针对该问题,本文提出了一种基于带宽的数据分区方法,在带宽异构的集群环境下可以有效减少数据分区完成的时间.

本文的主要贡献有3个方面:

1) 提出了一种基于带宽的数据分区方法,该方法在异构带宽的集群下能有效减少数据分区所需的时间;

2) 在新一代大数据计算框架 Flink 中,对基于带宽的数据分区方法进行了实现;

3) 通过实验对基于带宽的数据分区方法进行了验证,实验结果显示该方法可以有效地减少完成数据分区所需的时间.

1 相关工作

针对异构集群环境下的大数据框架优化的研究已有不少,主要的研究方向是针对节点间计算能力的不同,为各节点分配不同的数据量或者不同的计算任务.如在异构 Hadoop 集群中,文献[11]针对集群中节点计算性能不同的特点,以数据本地性策略为基础,通过在计算能力更强的节点放置更多的数据块,使得计算能力强的节点处理更多的数据,从而提升系统的性能;文献[12-13]则针对异构的 Hadoop 集群,考虑提交至集群的作业运行时需要

的资源大小和集群中可用资源的数量,提出了一种新的调度系统 COSHH,该调度系统可以结合 Hadoop 中原始的调度策略进行使用,进一步减少异构 Hadoop 集群中作业的平均完成时间,使得 MapReduce 模型在异构集群中的运行效率更高.

在异构 Spark 集群中同样有着相应的研究,如文献[14]提出了一种在异构 Spark 集群下的自适应任务调度策略,其主要考虑的是集群中各节点的计算能力不同,通过对各节点的负载和资源利用率进行监测来动态地调整节点任务的分配;文献[15]则采用了一种主动式的数据放置策略,通过对任务所需的计算时间进行预测,在初始数据加载过程中将数据放置在适当的节点上,并在作业执行的过程中进一步对数据的放置进行调整,缩短作业的整体运行时间.

对于数据在节点之间的传输,主要的研究方向是针对同构集群中的数据倾斜问题,比如文献[16]提出了一种用于 MapReduce 的采样算法,在 Hadoop 集群中不需要对输入数据运行额外的预采样程序,就能比较精确地估计出中间结果的分布,从而均衡各节点的数据量;同样是针对 MapReduce 框架中出现的数据倾斜问题,文献[17]基于对 Map 端中间结果的采样,提出了一种基于动态划分的负载均衡方法,可以保证每个 Reduce 任务处理的数据量尽量均衡;文献[18]则针对 Spark 提出了一种基于键重分配和分区切分的算法,该算法作用于中间结果的产生和 shuffle 过程中,同样用于解决数据倾斜问题;针对数据传输过程的优化通常都需要使用采样算法来获取数据的信息,文献[19]针对大规模数据流,提出了一种改进的水塘抽样方法,Flink 中使用该抽样方法实现了 Range 分区.

以上研究都没有考虑在异构带宽情况下,如何对数据分区方法进行优化.对于数据密集型作业,网络传输往往是瓶颈所在,在异构带宽条件下,传统考虑负载均衡的数据分区方法运行效率反而低下.针对该问题,本文通过建立基于传输时间的数据分发模型,提供了一种基于带宽的数据分区方法,在异构带宽的集群环境下可以有效地减少数据的传输时间.

2 相关概念介绍

2.1 Flink 框架

Flink 与大多数大数据框架一样也可以分为 Master 和 Slave 节点,如图 1 所示,其中充当 Master 的称为 JobManager,充当 Slave 的称为 TaskManager,除此之外,提交作业的节点通常称为 Client.

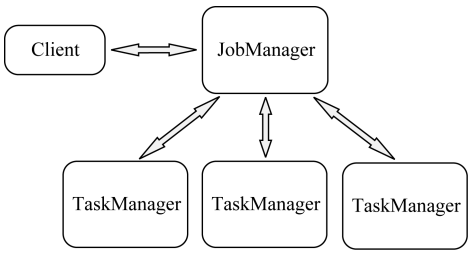

Fig. 1 The architecture of Flink

图 1 Flink 架构图

Flink 中 JobManager 将接收 Client 提交的作业,对作业进行调度并选定 TaskManager 进行任务的执行,收集作业运行的状态,并在作业运行失败时进行容错和恢复,TaskManager 上则真正运行着作业的各个子任务[20].通常 Flink 集群中会有一个 JobManager 和多个 TaskManager.

Flink 中作业会被抽象为数据流图,通常都是一个 DAG 结构[21].具体来讲,作业在 Client 端提交后,如果是批处理作业会通过优化器生成 Optimized-Plan,如果是流处理作业则会生成 StreamGraph,之后会继续在 Client 统一转化为 JobGraph,提交给 JobManager.在 JobManager 处接收到 JobGraph 之后,会将其转化为 ExecutionGraph,最后调度执行.

2.2 Flink 中的分区方法

大数据计算框架通常都会为用户提供数据分区的功能[22],Flink 在其批处理 API 中也提供了 3 种常用的数据分区方法,包括 Rebalance 分区、Hash 分区和 Range 分区.

2.2.1 Rebalance 分区

Rebalance 分区是 Flink 中最简单的数据分区方法,通过该分区方法可以很好地均衡每个节点上的数据,但其无法保证具有相同键的数据分发到同一节点上.Flink 中使用 Round-Robin 算法实现了 Rebalance 分区,具体算法如算法 1 所示:

算法 1. Rebalance 分区算法.

输入:待分区的记录 *record*、分区数量 *numPartitions*、当前分区编号 *partitionToSendTo*;

输出:待分区记录的分区编号 *partitionToSendTo*.

① *partitionToSendTo*++;

② IF *partitionToSendTo*≥*numPartitions*

③ *partitionToSendTo*=0;

④ END IF

⑤ RETURN *partitionToSendTo*.

2.2.2 Hash 分区

Hash 分区是使用最普遍的数据分区方法,该分区方法是基于 Hash 算法实现的[23].使用该分区方法首先会根据待分区记录的 *key* 值得到相应的 Hash 值,之后利用 Hash 值对分区数量取余,得到的结果作为该条记录所属的分区编号.因为相同的 *key* 值一定有相同的 Hash 值,因此 Hash 分区可以保证键相同的记录分发到同一节点上.具体算法如算法 2 所示:

算法 2. Hash 分区算法.

输入:待分区的记录 *record*、分区数量 *numPartitions*;

输出:待分区记录的分区编号 *partitionToSendTo*.

① *key*=*extractKey*(*record*); /* 提取记录的 *key* */

② IF *key*==null

③ *partitionToSendTo*=0;

④ ELSE

⑤ *hash*=*key*.*hashCode*;

⑥ *partitionToSendTo* =*Hash*%*numPartitions*;

⑦ END IF

⑧ RETURN *partitionToSendTo*.

2.2.3 Range 分区

Range 分区是一种根据所有待分区记录的键的范围进行数据分区的方法[24],也就是说每个分区结果都包含互不相交的键在一定范围内的记录,也因此使用 Range 分区方法时需要确定每个分区的边界.为了确定每个分区的边界,通常使用的方式是对输入数据进行抽样.Flink 中使用的抽样算法是改进后的蓄水池抽样算法,各分区边界的确定则是对抽样数据进行排序后按等比例获取各个分区的边界.

举例来说,假设抽样得到的数据按键排序后的结果为{(10,*value*1),(20,*value*2),(30,*value*3),(40,*value*4),(50,*value*5),(60,*value*6),(70,*value*7),(80,*value*8),(90,*value*9)},分区数量为 3,则计算得到的边界为{30,60},也就是说 *key*≤30 的记录将会被发往第 1 个分区,*key*∈(30,60]之间的数据会发往第 2 个分区,*key*>60 的数据则会发往第 3 个分区.Range 分区方法通过抽样并等比例划分各个分区的边界,可以在保证键相同的记录发往同一节点的同时,使得各分区拥有的数据大致相等.具体算法如算法 3 所示:

算法 3. Range 分区算法.

输入:输入源的分区数量 *numInputPartitions*、待分区的记录 *record*、分区数量 *numPartitions*;

输出:待分区记录的分区编号 *partitionToSendTo*.

① 使用改进后的蓄水池抽样算法在每个输入源分区上进行抽样;

② 将各输入源分区上的抽样结果进行汇总,得到 *sampleData*[],并排序;

③ 根据分区数量 *numPartitions* 和 *sampleData*[],计算出分区边界 *rangeBoundary*[];

④ 对于每条待分区的记录 *record*,在分区边界 *rangeBoundary*[]中查找出所属的分区编号 *partitionToSendTo*;

⑤ RETURN *partitionToSendTo*.

3 基于带宽的数据分区方法

本节我们先对最优数据分发比例的计算建立模型.之后举例说明异构带宽的集群中不同的数据分发比例对数据分区完成时间的影响,体现基于带宽的数据分区方法的重要性.最后介绍针对异构带宽的数据分区方法的算法流程以及在 Flink 中的实现.

3.1 最优数据分发比例建模

本节对异构带宽环境下各节点最优数据分发比例的计算建立模型,首先对所要用到的变量进行定义.

D_i:节点 i 上的初始数据量大小;

D_i^{u}:节点 i 需要传出的数据量大小;

D_i^{d}:节点 i 需要传入的数据量大小;

D:各节点的数据量之和,$D=\sum_i D_i$;

x_i:节点 i 上数据分发的比例,显然 $\sum_i x_i=1$;

u_i:节点 i 的上行带宽;

d_i:节点 i 的下行带宽;

t_i^{u}:节点 i 上传数据所需时间;

t_i^{d}:节点 i 下载数据所需时间;

cost:数据分发所要花费的总时间.

其中节点 i 需要传出的数据量 D_i^{u} 和需要传入的数据量 D_i^{d} 分别为

$$D_i^{\mathrm{u}}=(1-x_i)D_i, \tag{1}$$

$$D_i^{\mathrm{d}}=x_i(D-D_i). \tag{2}$$

节点 i 上传完所有数据所需时间 t_i^{u} 和下载完所有数据所需时间 t_i^{d} 分别为

$$t_i^{\mathrm{u}}=D_i^{\mathrm{u}}/u_i, \tag{3}$$

$$t_i^{\mathrm{d}}=D_i^{\mathrm{d}}/d_i. \tag{4}$$

数据分发所要花费的总时间 $cost$ 则是各节点传输数据所需时间的最大值.我们的目标是最小化数据分发所需的时间,则形式化地针对数据传输时间的优化模型表示为

$$\begin{aligned}&\min \quad cost\\ &\text{s.t.} \quad \forall i: x_i \geqslant 0,\\ &\qquad \sum_i x_i = 1;\\ &\forall i: t_i^{\mathrm{u}} - cost \leqslant 0;\\ &\forall i: t_i^{\mathrm{d}} - cost \leqslant 0.\end{aligned}$$

该模型是一个典型的线性规划问题,使用计算机可以比较方便地求解.

3.2 示 例

考虑集群中参与数据分区的 2 个节点 *Slave*1 和 *Slave*2,它们初始的节点信息如表 1 所示:

Table 1 Information of Nodes

表 1 节点信息表

Nodes	D_i/MB	u_i/Mbps	d_i/Mbps
*Slave*1	320	2	10
*Slave*2	160	10	10

其中 *Slave*1 节点上的初始数据量 $D_1=320$ MB,上行带宽 $u_1=2$ Mbps,下行带宽 $d_1=10$ Mbps. *Slave*2 节点上的初始数据量 $D_2=160$ MB,上行带宽 $u_2=10$ Mbps,下行带宽 $d_2=10$ Mbps.

当 *Slave*1 和 *Slave*2 以 50%和 50%的比例进行数据分发时,可以分别计算出 *Slave*1 和 *Slave*2 传输的数据量大小和所需时间,具体如表 2 所示:

Table 2 Transmission Information on Proportional 50%:50%

表 2 50%:50%比例分配数据传输信息表

Nodes	D_i^{u}/MB	D_i^{d}/MB	t_i^{u}/s	t_i^{d}/s
*Slave*1	160	80	640	64
*Slave*2	80	160	64	128

其中 *Slave*1 需要传出 50%的数据,即 160 MB,接收来自 *Slave*2 的 80 MB 数据.*Slave*2 则需传出 80 MB 数据,接收来自 *Slave*1 的 160 MB 数据.根据 *Slave*1 和 *Slave*2 的上下行带宽可以计算得出相应的传输时间,而最终数据分区完成需要取决于传输最慢的节点,也就是说以 50%和 50%的比例进行数据分区,最终需要花费 640 s 来完成.

考虑以 90%和 10%的比例进行数据分发,也就是说数据分发结束后 *Slave*1 保留 90%的数据,*Slave*2 保留 10%的数据.同样可以计算出各节点所需传输数据量大小和相应的时间,如表 3 所示:

Table 3 Transmission Information on Proportional 90%:10%

表 3 90%:10%比例分配数据传输信息表

Nodes	D_i^{u}/MB	D_i^{d}/MB	t_i^{u}/s	t_i^{d}/s
*Slave*1	32	144	128	115.2
*Slave*2	144	32	115.2	25.6

其中 *Slave*1 需要传出 10%的数据,即 32 MB,接收来自 *Slave*2 的 144 MB 的数据.*Slave*2 则需传出 144 MB 数据,接收来自 *Slave*1 的 32 MB 数据.同理计算出传输时间后,可以得到最终传输结束所需时间为 128 s,与分配比例为 50%时相比速度提高了 4 倍.

3.3 算法设计

以建立的最优数据分发比例模型为基础,可以设计出基于带宽的数据分区算法,如算法 4 所示:

算法 4. 基于带宽的数据分区算法.

输入:输入源的分区数量 *numInputPartitions*、待分区的记录 *record*、分区数量 *numPartitions*、参与分区的节点信息 *instanceInfo*;

输出:待分区记录的分区编号 *partitionToSendTo*.

① 使用改进后的蓄水池抽样算法在每个输入源分区上进行抽样;

② 将各输入源分区上的抽样结果进行汇总,得到 *sampleData*[],并排序;

③ 根据参与分区的节点信息 *instanceInfo* 计算出最优数据分发比例 *ratio*[];

④ 根据最优数据分发比例 *ratio*[]和得到的抽样结果 *sampleData*[],计算出分区边界 *rangeBoundary*[];

⑤ 对于每条待分区的记录 *record*,在分区边界 *rangeBoundary*[]中查找出所属的分区编号 *partitionToSendTo*;

⑥ RETURN *partitionToSendTo*.

3.4 算法实现

鉴于新一代大数据计算框架 Flink 的出色性能,选用 Flink 对基于带宽的数据分区算法进行了实现.

实现基于带宽的数据分区方法,需要完成最优数据分发比例的计算和作业图逻辑的修改.3.4 节已经提到过,计算节点的最优数据分发比例需要节点的带宽信息和数据量.原始的 Flink 无法获取集群

中各节点的带宽信息,考虑实现简便性,我们在Flink的配置文件中添加了各节点上下行带宽的配置项,在Flink集群启动时,各TaskManager会将自身的带宽信息汇总到JobManager处.各节点的数据量则根据作业JobGraph中的Source算子,获取相应的数据源分布情况后推算得出.作业图逻辑的修改包括采样算法的加入和分区方法的重写,这部分将在生成OptimizedPlan时完成,这样可以减少JobManager处的负载.

图2对一个基于带宽的数据分区作业的整体流程进行了详细描述.如图2中Step1～3所示,作业在Client端提交后,首先会通过优化器优化生成OptimizedPlan,之后将以生成的OptimizedPlan为基础,生成作业图JobGraph.在作业图生成的过程中我们添加了采样的逻辑和用于计算分区边界的算子,并重写了数据分区的方法.其中计算分区边界的算子会根据最优数据分发比例得到数据分区的界,该结果将会通过广播的方式发送到每个分区算子.需要注意的是,此时还在Client端,计算分区边界的算子还没有实际获取到最优数据分发比例,最优数据分发比例的获取需要在JobManager处完成.完成作业逻辑的修改后,通过Step3生成的JobGraph将被提交到JobManager.

如图2中Step4～7所示,JobManager收到作业的JobGraph后,首先会遍历JobGraph中的算子并找到Source算子,通过Source算子中存储的数据源信息去获取待处理数据在集群中的分布情况,并结合作业的并行度选择运行该作业的节点.考虑网络传输是作业运行的瓶颈,节点的选择策略是尽可能选择拥有数据的节点来运行作业,这样根据数据本地性策略,可以减少Source算子读取数据源时的网络传输.确定作业运行的节点后,通过节点的带宽信息和初始数据量大小,使用数据分发比例计算模块就可以计算出各节点的最优数据比例,该比例将会被写回JobGraph中用于计算分界的算子.至此,包含最终作业执行逻辑的作业图JobGraph才真正构建完成.最后根据JobGraph中各算子的并行度,会生成对应的执行图ExecutionGraph,执行图中的每个任务通过Step7将部署至对应的节点,进行调度执行.

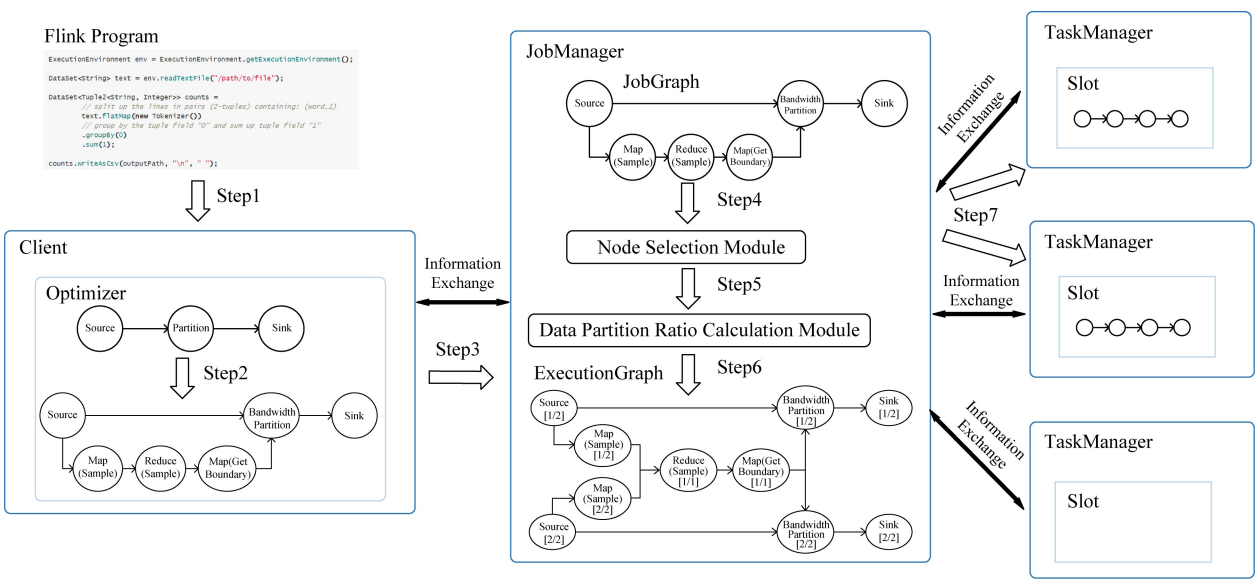

Fig. 2 The process of bandwidth partitioning job

图2 基于带宽的数据分区方法作业运行过程

4 实验与分析

4.1 实验环境

实验所用环境为4个节点的分布式集群,每个节点的处理器为Intel Xeon E5-2603 V4(6核6线程),内存为64 GB,节点间通过千兆以太网连接,安装的操作系统为CentOS7.集群上通过Standalone模式搭建了修改后的Flink集群,其中1台master节点作为JobManager,另外3台Slave节点作为TaskManager,使用的版本为Flink1.7.2.除此之外集群中还基于Hadoop2.7.5搭建了Hadoop集群,使用其中的HDFS作为分布式文件存储系统.集群中各节点带宽的控制则通过工具Wondershaper[25]来实现.

4.2 数据集

实验使用的数据是通过 TPC-H[26] 基准测试工具生成的数据集,该工具可以生成 8 种表,选取了其中较大的 Lineitem 表和 Orders 表作为数据源.其中 Lineitem 有 16 个字段,前 3 个字段 Orderkey,Partkey,Suppkey,其中 Suppkey 是主键.Orders 表有 9 个字段,前 2 个字段 Orderkey 和 Custkey,其中 Custkey 是主键.

4.3 实验结果与分析

本节从算法开销和算法效果 2 方面进行实验结果的说明与分析.

4.3.1 算法开销

基于带宽的数据分区方法的算法开销主要包括作业图逻辑修改、最优数据分发比例的计算、数据采样 3 部分,本文针对这 3 部分所需的开销分别进行了实验.

作业图逻辑的修改发生在 Flink 作业图生成的过程中,主要包括采样算子的添加、计算分界算子的添加以及分区方法的重写等步骤.通过与未修改作业图逻辑进行对比,可以得到作业图逻辑修改所需的时间.经过 5 次实验并取平均值,得到从作业提交到作业图生成完毕所需的平均时间为 185 ms,如果进行作业图逻辑的修改,所需的平均时间则为 232 ms,即作业图逻辑修改平均所需时间为 47 ms.

最优数据分发比例的计算是利用数学规划优化器 Gurobi Optimizer[27] 实现的数据分发比例计算模块完成的,实验对不同节点数量下最优比例计算所需的时间进行了测试,每个节点的带宽和数据量大小则随机生成.实验结果如图 3 所示,当节点数量为 5 时所需的计算时间为 32 ms,节点数量扩大至 1 000 时计算时间仍在 100 ms 以内,当节点数量达到 6 000 时所需的计算时间也仅为 293 ms.

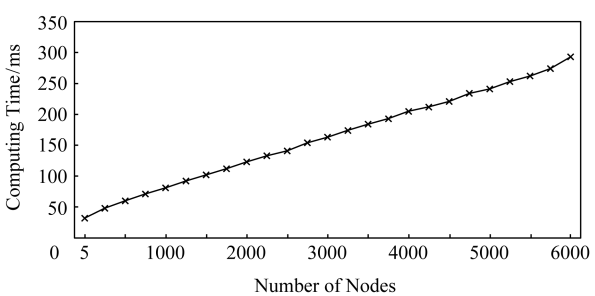

Fig. 3 The time of optimal ratio calculation

图 3 最优比例计算时间

为了测试数据采样所需的开销,我们分别运行了添加了采样过程的作业和未添加采样过程的作业,使用作业运行的时间差作为采样所需的开销.实验中使用了不同数据量大小的 Lineitem 表作为输入,具体实验结果如图 4 所示,在数据量大小分别为 3.6 GB,7.2 GB,14.6 GB,29.4 GB 时,所需的采样时间分别为 21 s,44 s,86 s,167 s,平均每 GB 数据所需的采样时间约为 5.8 s.

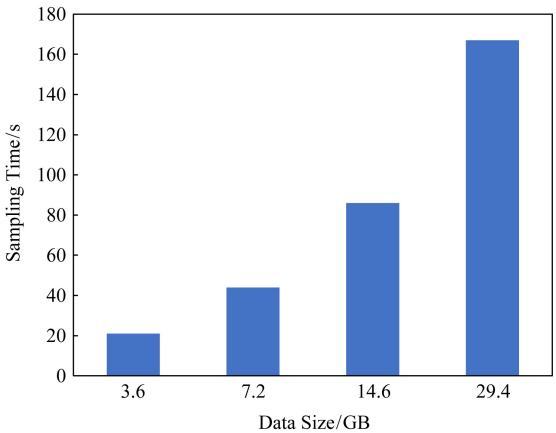

Fig. 4 The time of sampling

图 4 采样时间

总体来说,基于带宽的数据分区方法在作业图逻辑修改和最优数据分发比例的计算过程中所需的时间开销都较小,为毫秒级别.数据采样则相对时间开销较大,且与数据量大小相关,但在计算资源更为充足的情况下,采样所需时间可以进一步减少.

4.3.2 算法效果

为了探究本文提出的算法在不同的异构带宽条件下的效果,我们设置了带宽异构程度不同的 4 个实验.同时为了实验的方便,主要针对 $Slave1$ 节点设置了不同的下行带宽,这样已经可以涵盖不同的数据分发比例,其他情形的异构带宽集群则与此类似.实验中各节点的具体带宽如表 4 所示,表 4 中上行带宽在前、下行带宽在后.

Table 4 Bandwidth Information

表 4 带宽信息表 Mbps

Experiments	u_{Slave1}/d_{Slave1}	u_{Slave2}/d_{Slave2}	u_{Slave3}/d_{Slave3}
experiment1	100/10	100/100	100/100
experiment2	100/25	100/100	100/100
experiment3	100/50	100/100	100/100
experiment4	100/75	100/100	100/100

实验中使用的数据为 3.6 GB 的 Lineitem 表和 1.63 GB 的 Orders 表.实验程序会先对数据集进行分区操作,数据分区结束之后将在每个分区中进行

一次聚合,统计各分区最终的记录数量,以便计算出最终各分区数据的比例.程序的执行模式设置为 Batch 模式,分区方法使用 Hash 分区、Range 分区与基于带宽的 Bandwidth 分区进行比较,验证基于带宽的 Bandwidth 分区效果.

因为实验主要针对的是节点带宽的影响,而实验所使用的集群 TaskManager 的数量是 3,因此作业运行的并行度同样设置为 3.使用的数据源则被上传到 HDFS 中,3 个节点上的数据量几乎是相等的,因此可以认为 Source 算子的每个并行度读入的数据量都是相同的.对数据集进行介绍时有过说明,Lineitem 表中有 3 个字段是主键,Orders 表中有 2 个字段是主键,在实验过程中我们发现这 2 个表中并没有明显的数据倾斜,通过主键中的任一字段做数据分区,实验结果都是相似的.后续的实验结果都是以各个表的第 1 个字段作为键来进行数据分区,也就是说数据源 Lineitem 和 Orders 都使用 Orderkey 作为键进行数据分区.

如图 5 所示,在实验 1 条件下,使用 Bandwidth 分区的作业时间在 Lineitem 上所需时间为 198 s,在 Orders 上所需时间为 92 s,明显小于 Hash 分区和 Range 分区所需时间,作业运行完成整体速度提升了为 2.5～3 倍.

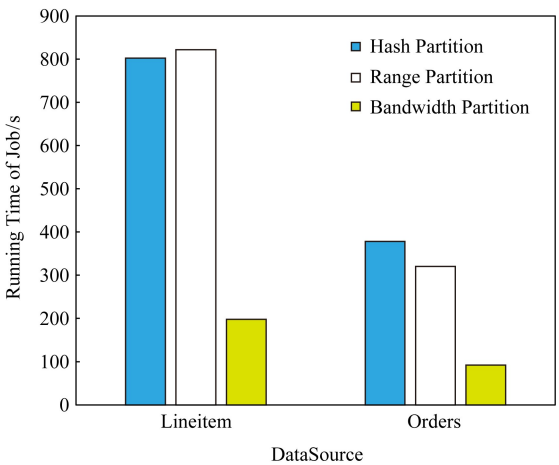

Fig. 5 Running time in different partition modes in experiment 1

图 5 实验 1 中不同分区模式下的执行时间

在实验 1 的条件下,可以计算出 3 个节点数据的最优分配比例为 4∶48∶48,通过表 5 和表 6 可以看出,使用 Bandwidth 分区很好地契合了最优数据分配比例,特别是在 Lineitem 上实际数据分区比例与最优比例几乎完全相同.

Table 5 Proportion of Lineitem After Partition in Experiment 1

表 5 实验 1 中 Lineitem 分区后各节点数据比例 %

Partition Method	*Slave*1	*Slave*2	*Slave*3
Hash Partition	33	33	33
Range Partition	31	39	30
Bandwidth Partition	3	48	49

Table 6 Proportion of Orders After Partition in Experiment 1

表 6 实验 1 中 Orders 分区后各节点数据比例 %

Partition Method	*Slave*1	*Slave*2	*Slave*3
Hash Partition	33	33	33
Range Partition	29	35	36
Bandwidth Partition	3	43	54

如图 6 所示,在实验 2 条件下,使用 Bandwidth 分区的作业时间在 Lineitem 上所需时间为 209 s,在 Orders 上所需时间为 99 s,相较于 Hash 分区和 Range 分区,效果同样不错,提升为 0.6～0.7 倍.

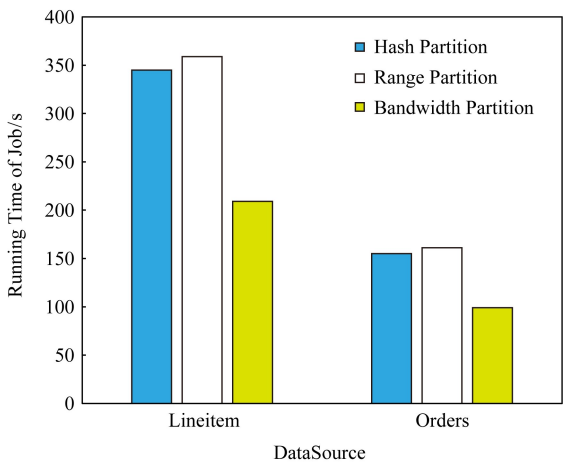

Fig. 6 Running time in different partition modes in experiment 2

图 6 实验 2 中不同分区模式下的执行时间

在实验 2 的条件下,计算出的各节点数据的最优分配比例为 11∶44∶44,通过表 7 和表 8 可以看出,使用 Bandwidth 分区后的数据分布也与最优数据分配比例比较契合.

如图 7 所示,在实验 3 条件下,使用 Bandwidth 分区的作业时间在 Lineitem 上所需时间为 184 s,在 Orders 上 Bandwidth 分区所需时间为 68 s,相较 Hash 分区所需时间 194 s 和 88 s 已经没有太大的优势,但仍然能节省一些时间.

Table 7 Proportion of Lineitem After Partition in Experiment 2

表 7 实验 2 中 Lineitem 分区后各节点数据比例 %

Partition Method	*Slave*1	*Slave*2	*Slave*3
Hash Partition	33	33	33
Range Partition	34	30	36
Bandwidth Partition	10	44	46

Table 8 Proportion of Orders After Partition in Experiment 2

表 8 实验 2 中 Orders 分区后各节点数据比例 %

Partition Method	*Slave*1	*Slave*2	*Slave*3
Hash Partition	33	33	33
Range Partition	35	30	35
Bandwidth Partition	15	47	38

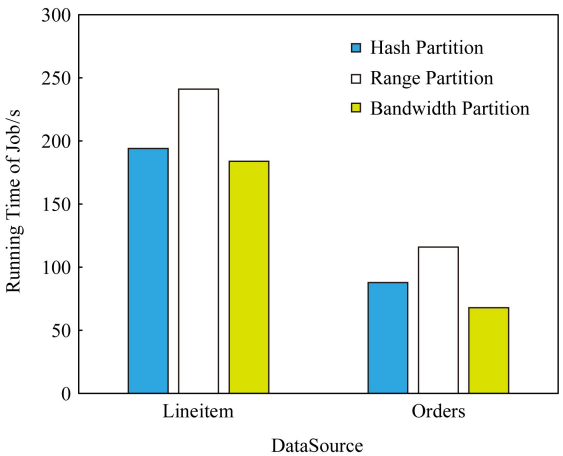

Fig. 7 Running time in different partition modes in experiment 3

图 7 实验 3 中不同分区模式下的执行时间

此时节点之间数据传输的最优比例已经是 20∶40∶40,与平均分配差距已经没有那么大,同时还可以发现,此次实验条件下 Range 分区表现较差,分区完成所用时间与其他实验相比明显变长.分析表 9 和表 10 可以发现,Range 分区在实验 4 中,对需要数据比例少的 *Slave*1 节点反而分配了更多的数据,导致 Range 分区所需时间远超了其他 2 种分区方法.

如图 8 所示,在实验 4 条件下,与 Hash 分区相比,Bandwidth 分区所需时间反而变得更长.此时节点间的最优比例是 27∶36∶36,与平均分配比例已经十分接近,而 Range 分区和 Bandwidth 分区需要额外的采样时间.除此之外,结合表 11 和表 12 可以发现,采样得到的结果并不是特别准确,导致并不能完全按计算得到的最优比例进行数据分发.

Table 9 Proportion of Lineitem After Partition in Experiment 3

表 9 实验 3 中 Lineitem 分区后各节点数据比例 %

Partition Method	*Slave*1	*Slave*2	*Slave*3
Hash Partition	33	33	33
Range Partition	39	26	35
Bandwidth Partition	15	45	40

Table 10 Proportion of Orders After Partition in Experiment 3

表 10 实验 3 中 Orders 分区后各节点数据比例 %

Partition Method	*Slave*1	*Slave*2	*Slave*3
Hash Partition	33	33	33
Range Partition	37	33	30
Bandwidth Partition	14	51	35

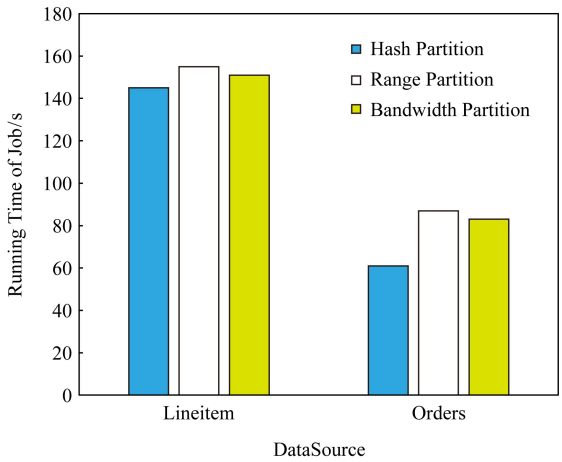

Fig. 8 Running time in different partition modes in experiment 4

图 8 实验 4 中不同分区模式下的执行时间

Table 11 Proportion of Lineitem After Partition in Experiment 4

表 11 实验 4 中 Lineitem 分区后各节点数据比例 %

Partition Method	*Slave*1	*Slave*2	*Slave*3
Hash Partition	33	33	33
Range Partition	30	36	34
Bandwidth Partition	31	31	38

结合 4 个实验,可以发现 Hash 分区十分稳定,每次实验分区结果都十分均衡,说明数据源中并没有明显的数据倾斜.Range 分区和 Bandwidth 分区

则并不能每次都保证数据按预设的比例分配,主要是因为它们都需要使用采样算法来估计数据分布,有时候采样的结果并不是十分精确.同样由于采样算法的存在,Range 分区和 Bandwidth 分区都需要额外的开销,这也导致大多数时候 Range 分区都比 Hash 分区花费更多的时间.唯一例外的是在实验 1 中对 Orders 表进行分区,原因是 Range 分区恰好给瓶颈节点 *Slave*1 分配了更小的比例,而 *Slave*1 下行带宽很小,较小的数据量就会对传输时间产生较大的影响.

Table 12 Proportion of Orders After Partition in Experiment 4

表 12 实验 4 中 Orders 分区后各节点数据比例 %

Partition Method	*Slave*1	*Slave*2	*Slave*3
Hash Partition	33	33	33
Range Partition	26	45	29
Bandwidth Partition	26	40	34

综合来看,当带宽异构性强,各节点之间最优数据分发比例比较不均衡时,基于带宽的数据分区方法可以取得较好的效果,甚至带来数倍的速度提升.当带宽异构性较弱时,由于采样算法需要额外的开销,基于带宽的数据分区方法所需时间可能会长于 Hash 分区方法,这种情况下可以通过更充足的计算资源来降低采样过程所需的开销.在实际应用过程中,则可以综合考虑最优比例的计算结果和采样所需的时间,在速度提升较为明显时选择使用基于带宽的数据分区方法.

5 总 结

在异构带宽的条件下,传统的数据分区方法会因为瓶颈节点的存在,导致数据分发效率低下.通过对各节点之间数据传输模型进行分析,本文提出了一种针对异构带宽集群的数据分区方法,并在 Flink 中进行了实现.实验证明:在节点间带宽异构的情况下,基于带宽的数据分区方法可以极大地提升数据分区完成的速度.

参 考 文 献

[1] Brown B, Chui M, Manyika J. Are you ready for the era of 'big data'[J]. McKinsey Quarterly, 2011, 4(1): 24-35

[2] LaValle S, Lesser E, Shockley R, et al. Big data, analytics and the path from insights to value [J]. MIT Sloan Management Review, 2011, 52(2): 21-32

[3] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113

[4] Apache. Hadoop [EB/OL]. [2019-05-31]. http://hadoop.apache.org

[5] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster computing with working sets [C] //Proc of HotCloud 2010. Berkeley, CA: USENIX Association, 2010

[6] Carbone P, Katsifodimos A, Ewen S, et al. Apache Flink: Stream and batch processing in a single engine [J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015, 36(4): 28-38

[7] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing [J]. Communications of the ACM, 2010, 53(4): 50-58

[8] Patterson D A. Technical perspective: The data center is the computer [J]. Communications of the ACM, 2008, 51(1): 105-105

[9] Nygren E, Sitaraman R K, Sun J. The Akamai Network: A platform for high-performance Internet applications [J]. ACM SIGOPS: Operating System Review, 2010, 44(3): 2-19

[10] Microsoft Research. Sampling based range partition methods for big data analytics [EB/OL]. [2019-05-31]. https://www.researchgate.net/profile/Jin-gren_Zhou4/publication

[11] Xie Jiong, Yin Shu, Ruan Xiaojun, et al. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters [C] //Proc of Workshop on Heterogeneity in Computing (IPDPS 2010). Piscataway, NJ: IEEE, 2010: 1-9

[12] Rasooli A, Down D G. A hybrid scheduling approach for scalable heterogeneous Hadoop systems [C] //Proc of 2012 SC Companion: High Performance Computing, Networking Storage and Analysis. Piscataway, NJ: IEEE, 2012: 1284-1291

[13] Rasooli A, Down D G. COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems [J]. Future Generation Computer Systems, 2014, 36: 1-15

[14] Yang Zhiwei, Zheng Quan, Wang Song, et al. Adaptive task scheduling strategy for heterogeneous Spark cluster [J]. Computer Engineering, 2016, 42(1): 31-35 (in Chinese)
(杨志伟, 郑烇, 王嵩, 等. 异构 Spark 集群下自适应任务调度策略[J]. 计算机工程, 2016, 42(1): 31-35)

[15] Zhang Haitao, Xu Bin, Yan Jin, et al. Proactive data placement for surveillance video processing in heterogeneous cluster [C] //Proc of 2016 IEEE Int Conf on Cloud Computing Technology and Science. Piscataway, NJ: IEEE, 2016: 206-213

[16] Chen Qi, Yao Jinyu, Xiao Zhen. LIBRA: Lightweight data skew mitigation in MapReduce [J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 26(9): 2520-2533

[17] Zhou Jiashuai, Wang Qi, Gao Jun. An approach for load balancing in MapReduce via dynamic partitioning [J]. Journal of Computer Research and Development, 2013, 50(Suppl): 369-377 (in Chinese)
(周家帅, 王琦, 高军. 一种基于动态划分的 MapReduce 负载均衡方法[J]. 计算机研究与发展, 2013, 50(增刊): 369-377)

[18] Lü Wei, Tang Zhuo, Li Kenli, et al. An adaptive partition method for handling skew in Spark applications [C] //Proc of the 4th IEEE Smart World Congress (SmartWorld 2018). Piscataway, NJ: IEEE, 2018: 1063-1070

[19] Tirthapura S, Woodruff D P. Optimal random sampling from distributed streams revisited [C] //Proc of the 25th Int Symp on Distributed Computing. Berlin: Springer, 2011: 283-297

[20] Bergamaschi S, Gagliardelli L, Simonini G, et al. BigBench workload executed by using Apache Flink [J]. Procedia Manufacturing, 2017, 11: 695-702

[21] Marcu O C, Costan A, Antoniu G, et al. Spark versus Flink: Understanding performance in big data analytics frameworks [C] //Proc of 2016 IEEE Int Conf on Cluster Computing. Piscataway, NJ: IEEE, 2016: 433-442

[22] Gounaris A, Kougka G, Tous R, et al. Dynamic configuration of partitioning in Spark applications [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(7): 1891-1904

[23] Bertolucci M, Carlini E, Dazzi P, et al. Static and dynamic big data partitioning on Apache Spark [C] //Proc of 2015 Int Conf on Parallel Computing. Amsterdam: IOS Press, 2015: 489-498

[24] Paul A K, Zhuang Wenjie, Xu Luna, et al. Chopper: Optimizing data partitioning for in-memory data analytics frameworks [C] //Proc of 2016 IEEE Int Conf on Cluster Computing. Piscataway, NJ: IEEE, 2016: 110-119

[25] Wondershaper [CP/OL]. [2019-05-31]. https://github.com/magnific0/won-dershaper

[26] Transaction Processing Performance Council. TPC-H [EB/OL]. [2019-05-31]. http://www.tpc.org/tpch/

[27] Gurobi. Gurobi Optimizer [EB/OL]. [2019-05-31]. https://www.gurobi.com

Ma Qingyun, born in 1995. Master. His main research interests include big data mining and parallel computing.

Ji Hangxu, born in 1990. PhD candidate. His main research interests include graph embedding and distributed computing.

Zhao Yuhai, born in 1975. PhD, professor. Senior member of CCF. His main research interests include data mining and machine learning.

Mao Keming, born in 1981. PhD, associate professor. His main research interests include machine learning and pattern recognition.

Wang Guoren, born in 1966. PhD, professor. Senior member of CCF. His main research interests include XML data management, query processing, optimization, high-dimensional indexing, parallel database systems, P2P data management and uncertain data management.