计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20140403
52(6):1452-1462,2015

基于统计方法的 Hive 数据仓库查询优化实现

王有为[1] 王伟平[2] 孟 丹[2]
[1](中国科学院计算技术研究所集成应用中心 北京 100190)
[2](中国科学院信息工程研究所 北京 100093)
(wangyouwei@ncic.ac.cn)

Query Optimization by Statistical Approach for Hive Data Warehouse

Wang Youwei[1], Wang Weiping[2], and Meng Dan[2]
[1](*Integration Application Center*, *Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)
[2](*Institute of Information Engineering*, *Chinese Academy of Sciences*, *Beijing* 100093)

Abstract Map/Reduce is an efficient parallel programming model, which is now widely utilized to analyze massive data. Hive is an open source data warehouse which utilizes Map/Reduce to implement its query processing engine. However, the issue of unbalanced workload distribution in the whole cluster arises when processing skewed data using Map/Reduce. The possible results range from low runtime efficiency to task failures. To solve such problem, we propose an approach named the computation balanced model (CBM), which optimizes to queries by using distribution statistics. The main contributions of this paper include two parts correspondingly: firstly, the runtime cost evaluation model is established for two widely-used types of queries, i.e., the GroupBy and Join queries, especially under different situations; secondly, the highly-efficient statistics approach for massive data is designed and implemented adapting to the data access mechanism of Hive. Experiment results show the processing time of GroupBy query optimized by CBM is reduced by about 8%-45%, while the processing time of Join query is reduced by over 12%-46%. And the balance distribution of cluster payload is improved by about 60%-80% for CPU and 60%-90% for I/O. We believe the optimized query plan generator by CBM significantly balances the payload distribution during the execution of Map/Reduce tasks, as well as improves the query efficiency greatly.

Key words offline processing of massive data; distributed data warehouse; payload balance; statistics information collection; query optimization

摘 要 Map/Reduce 是海量离线数据分析中广泛应用的并行编程模型. Hive 数据仓库基于 Map/Reduce 实现了查询处理引擎,然而 Map/Reduce 框架在处理偏斜数据时会出现工作负载分布不均的问题.均衡计算模型(computation balanced model, CBM),其核心思想是通过数据分布特征指导查询计划优化.相应研究贡献包括 2 部分,首先针对应用极广的 GroupBy 查询和 Join 查询建立了运行估价模型,确定了不同场景下查询计划的优化选择分支;其次基于 Hive ETL 机制设计了一种统计信息收集方法,解决了统计海量数据分布特征的问题.实验数据表明,通过 CBM 优化的 GroupBy 查询耗时节省了 8%~45%,Join 查询耗时节省了 12%~46%;集群 CPU 负载均衡指标优化了 60%~80%,I/O 负载均

收稿日期:2014-05-07;**修回日期**:2014-08-28
基金项目:国家"八六三"高技术研究发展计划基金项目(2013AA013204);"核高基"国家科技重大专项基金项目(2013ZX01039-002-001-001);中国科学院战略性先导科技专项项目(XDA06030200)

衡指标优化了 60%～90%.实验结果证实了基于 CBM 模型优化的查询计划生成器能显著均衡化 Hive 查询运行时的集群负载,并优化了查询处理效率.

关键词 海量数据离线处理;分布式数据仓库;负载均衡;统计信息收集;查询优化

中图法分类号 TP311.133.1

互联网应用的持续爆炸性增长推动海量数据存储和处理技术的稳步发展,传统数据仓库技术受扩展性和数据处理成本等因素所限,无法胜任该领域的任务.如何高效存储并处理分析海量离线数据,从中提取宝贵信息成为研究人员面临的重大挑战.大量研究机构以及互联网领军企业对大数据应用进行了深入探索和研究,创造性地提出了许多关键技术.Google 提出的分布式文件系统 Google File System[1]和分布式计算框架 Map/Reduce[2](M/R)为海量数据存储和处理分析开辟了全新思路.Apache 基于 Google 发表的研究成果实现了开源的 Hadoop 分布式文件系统[3](Hadoop distributed file system,HDFS)以及相应的 M/R 开源实现,旨在提供高效廉价的海量数据存储和处理平台.然而直接基于 M/R API 编制程序较为冗繁,开发人员需要针对不同业务逻辑书写不同程序,代码冗余度大,难以重用且维护困难.针对该问题,Facebook 公司开发了数据仓库 Hive[4],封装了 HDFS 存储逻辑和 M/R 数据处理逻辑,对用户提供了友好的类 SQL 查询接口,实现了高效存储和分析海量数据的能力.M/R 框架的关键原理之一是通过特定分区算法将 Map 任务输出结果划分至不同 Reducer 上进行归并操作.对于存在偏斜分布的输入数据而言,在执行某些特定查询时,M/R 框架会将大量数据汇集至少数几个 Reducer 集中处理,从而使这些 Reducer 成为计算瓶颈,出现运行负载不均的现象.

Blanas 等人[5]针对 M/R 框架中 Join 算法优化问题,列举了计算等值 Join 的多种算法,并通过实验对比分析了部分关键实现细节.Kwon 等人[6]研究了现实数据中的偏斜分布现象,探讨了利用 M/R 框架对其处理时可能造成的负面影响,并建议了若干种解决方法;然而他们仅简单枚举了若干已知方法,未提及不同方法的适用场合和可能获得的优化效果,因此对解决我们的问题缺乏足够指导意义.

Ibrahim 等人[7]调查了 M/R 系统中的分区偏斜问题,提出了名为 LEEN 的算法,通过在 shuffle 阶段均衡分区并异步缓存 M/R 中间的 Key,从而解决了由数据偏斜导致的 Reduce 阶段负载不均的问题.Okcan 等人[8]提出了一种随机化算法用于在单个 M/R 任务中实现任意 Theta-Join 操作,关键是利用数据分布统计信息和以 Reducer 为中心的估价模型优化输出结果.Hassan 等人[9]讨论了一种基于 Map-Reduce-Merge 模型的 Semi-Join 算法,通过分布式直方图处理 Semi-Join 操作,基于代价分析证实了该算法对数据偏斜不敏感,并有效降低了磁盘 I/O 开销.Zhang 等人[10]提出了 SJMR 算法,通过基于条带的平面翻转算法和基于瓦块的空间分区算法优化了地理空间数据的 Join 操作效率,虽然这些算法模型从不同角度优化了基于 M/R 的 Join 操作效率,然而它们未涉及对聚合操作的优化,即缺乏充足实验数据证实相关优化机制对 GroupBy 这样的单表操作可达到同样优化效果.

Gufler 等人[11]通过细粒度分区以及基于预估代价的任务分配方法尝试解决 Reduce 阶段数据负载不均的问题.Ramakrishnan 等人[12]通过一种渐进式群采样器预估了聚合不同 Reduce Key 的运行代价,然后优化分割不同 Key 以最小化 Reducer 的峰值负载.这些方法的有效性在不同程度上取决于用户提供的数据特征描述的准确度.然而在缺乏足够样本信息的前提下,基于主观猜测的方式描述海量数据的分布规律难以满足制定优化决策的准确性要求,因此该方法对解决我们面临的问题并无太大帮助.

Yang 等人[13]对原 M/R 模型加入了 Merge 过程,在此基础上增加了若干种新的关系代数操作,拓展了 M/R 框架的处理能力.Ananthanarayanan 等人[14]针对日志分析输入数据源分布不均衡的问题,提出了 Scarlett 系统,关键在于根据数据使用频率自动完成数据块的均衡复制.Vernica 等人[15]提出了一种运行时机制,基于分布式元数据存储驱动 Mapper 感知当前任务状态,从而自适应地优化 M/R 任务性能.赵彦荣等人[16]提出了一种并行 Join 查询处理算法 CHMJ,通过多副本一致性 Hash 算法将 Join 操作目标数据集根据 Join 属性 Hash 值在集群中进行分布,并在此基础上提出了 HashMapJoin 并行 Join 算法以优化 Join 查询处理效率.Hammoud 等人[17]提出了一种名为 LARTS 的策略,关键在于识别输入数据的位置和大小,然后协同分布 Reduce

任务从而实现了静态优化 M/R 任务的目的. Kolb 等人[18]提出了一种优化负载均衡的方法,通过预处理确定每个数据输入分区的块分布矩阵,在 M/R 启动阶段细粒度地重组数据分布,从而实现对不同 M/R 节点的负载调节. 这些方法主要从动态运行时优化以及静态存储布局调整等方面着手解决 M/R 任务运行时负载不均的问题,并不涉及数据分布规律本身. 由于缺乏对输入数据统计本质的认识和了解,这些基于补偿原则的优化方法在处理偏斜数据时需要一定时延完成状态反馈,收效不及基于先验知识预先制定优化策略的方法.

不同于上述方法,本文提出的均衡计算模型(computation balanced model, CBM)关键思路阐述如下:首先通过统计采样方法识别入库数据分布规律;然后根据统计信息指导查询计划优化,重构计算过程从而解决负载不均的问题. 与之相呼应,本文组织如下:1)基于实际示例揭示了目标问题的实质,即 Hive 数据分析应用中常见的 GroupBy 以及 Join 查询在处理偏斜数据时的困难;2)提出了基于数据统计特征指导查询优化的 CBM 模型,包括核心思想和实现方法;3)通过 TPC-H 基准测试严格评估了 CBM 模型的实际应用性能,综合分析并详细讨论了实验结果,证实了 CBM 模型能高效处理原 Hive 系统无法处理的偏斜数据,缩短了查询耗时的同时有效均衡化集群的运行负载;4)总结全文并展望了未来工作.

1 问题阐述

1.1 GroupBy 查询优化问题

为求直观阐述问题,我们通过示例进行说明. 现有查询语句 Select Count(Distinct UserID) From Users Group By Gender,语义为按照性别对 UserID 进行分组统计,对应数据表 Users 结构如表 1 所示:

Table 1 The Structure of the Table Users

表 1 Users 表结构

Field	Description
UserID	String, Unique Identifications of Users
Gender	Enumeration, Genders of Users
Information	Rich Text, Detailed Description of Users

该查询语句经过翻译后,形成 M/R 任务如图 1 所示. 在 Map 阶段,原始数据表分块在不同 Mapper 上并行处理,Map 阶段输出结果按照字段 Gender 完成 Hash 计算后,属性值相同的记录被推送至相同 Reducer. 实线和虚线表示根据不同 Key 进行数据分区. Reducer 完成去重操作后输出最终统计结果(如果此时查询语句中带有 Distinct 子句). 考虑字段 Gender 的实际取值只有 2 种可能,因此该查询中间过程将至多产生 2 个 Reduce 分区,即所有数据在 Reduce 阶段将最多聚合至 2 个 Reducer 节点完成规约计算. 据图 1,在 Reduce 阶段集群已经处于负载不均的状态,进而引发的问题包括查询执行效率极为低下、大量节点闲置导致的处理资源严重浪费以及 Reducer 端的 JAVA 虚拟机进程由于内存不足而崩溃. 上述现象轻则导致查询性能低下,重则导致查询任务失败. 鉴于实际生产环境中超过一半查询需要使用分组和去重操作,且入库数据中预估带有偏斜分布的字段比例可达 70%以上,因此解决该问题具备极其重大的现实意义.

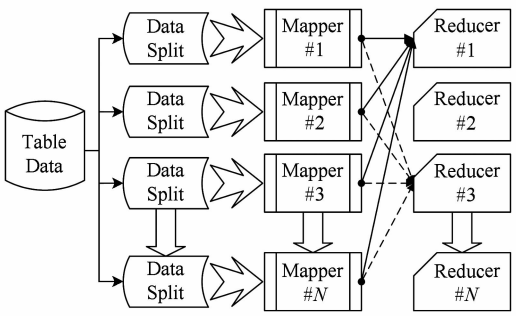


Fig. 1 Illustration of M/R task for the access statistic task by the Gender field.

图 1 根据 Gender 字段统计访问量的 M/R 任务

实际上,当 GroupBy 子句操作字段分别为 UserID 和 Gender 时,查询运行效率区别十分显著. 进一步观察可发现,数据偏斜分布主要包括 2 种情况:1)Key 值域宽广,多见于连续数值区间,但实际数据大量聚集在少数几个 Key 上;2)Key 值域狭小,多见于枚举型字段,如字段 Gender 仅有 2 种可行值. 对上述查询而言,仅当 GroupBy 操作字段为 Gender 才会出现这种计算负载不均衡现象. 即对单阶段 M/R 任务而言,计算负载不均衡主要是由数据分布偏斜所导致;反之亦然,即数据偏斜分布的不利影响仅在执行特定查询时才会表现出来.

一种可行的解决方法是调整查询引擎翻译的 M/R 任务,引入额外 M/R 任务将原有去重和统计操作分离:1)第 1 阶段 M/R 任务将原始数据以 UserID 作为 Key,通过 Hash 映射分配至不同 Reducer,然后执行计数操作;2)第 1 阶段 Reduce 按照 Gender 计算局部分组统计值;3 第 2 阶段 M/R 任务将前一

阶段输出的局部统计值再次按照字段 Gender 进行汇总,最终完成分组统计操作.该方案改进之处在于第 1 阶段 M/R 任务处理的是根据 UserID 离散化后的数据,均衡化了计算负载;第 2 个 M/R 任务处理的是第 1 阶段中预聚合的数据,由于预聚合过程极大压缩了初始数据,此时即使存在分布偏斜也不会严重影响查询性能.该事实启示我们多阶段 M/R 任务可在该场景下优化查询性能,然而随之而来的额外开销也必须加以权衡.总之我们可以在制定查询计划时根据输入数据的统计分布分析预估不同查询计划的可能开销,从而判定多阶段 M/R 任务是否可达到优化效果.

1.2 Join 查询优化问题

Join 操作是数据分析中另一种使用频率极高的操作.传统关系型数据库已对此深入研究多年,并实现了多种高效 Join 算法.然而对基于 M/R 模型的 Join 操作而言,相关性能优化研究工作尚待继续研讨.基于 M/R 框架的 Join 操作大致可以分为 2 种方式:

1) Map 端 Join.关键步骤包括将大表数据均等分配至参与任务的所有节点,同时复制小表数据至所有 Mapper 端;然后在 Map 阶段预聚合部分数据,即逐行比对大表数据和小表数据,如发现等值 Join Key 则将中间结果写入磁盘临时文件,该措施主要目的是为了增强系统可靠性,然而在一定程度上降低了系统性能;最后在 Reduce 端归并数据从而获得最终结果.执行过程如图 2 所示:

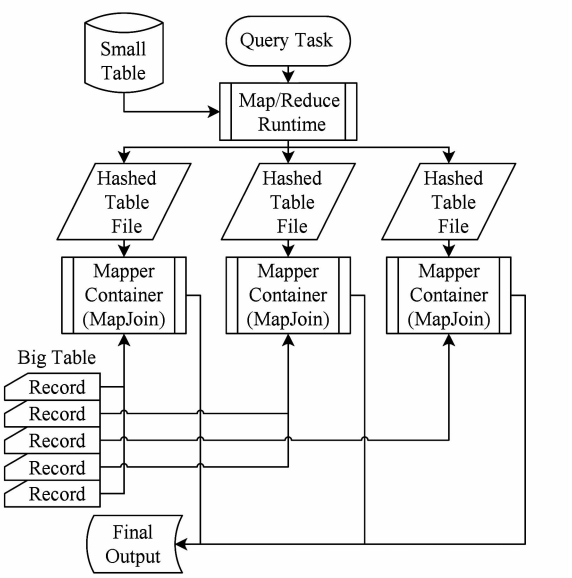


Fig. 2 Join implementation on the Map side.

图 2 Map 端 Join 实现

2) Reduce 端 Join.该方式由单个 M/R 任务完成.首先将大表和小表分别在 Map 端进行 Join 操作,在 Map shuffle 阶段将每个 Map 输出键值通过附加前缀的方式进行变换,分区操作时只使用实际列值计算 Hash 值;然后在 Reduce 的 shuffle 阶段,每个 Reducer 接收来自所有 Mapper 的数据分块,还原 Key 值并完成后续计算,如图 3 所示.这种方式的问题在于执行效率较低且处理带偏斜分布的数据时可能带来严重的计算负载不均问题.

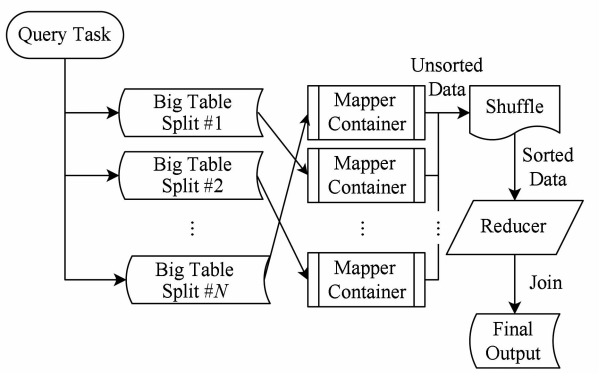


Fig. 3 Join implementation on the Reduce side.

图 3 Reduce 端 Join 实现

并非所有聚合操作都必须在 Reduce 端完成,然而一味使用 Map 端 Join 也无法保证必然获得更优性能,因此需要建立相应指导准则在不同 Join 方式之间作出最优选择.

2 查询优化分析和实现

本节主要讨论数据统计分布信息和查询优化之间的联系.从系统设计动机和目标出发,我们希望能最大限度均衡化查询过程中的系统运行负载;然而 M/R 框架的运行负载状况受到许多硬软件因素左右,包括 M/R 框架调度算法、存储布局以及数据统计特征等.这些因素在运行过程中高度可变且难以预测,因此精确地计量和分析这些因素对查询全过程完全无误差地建模并不现实.

为了评估数据分布规律对查询运行性能的影响,我们采取先验方法,即预先生成统计规律已知的数据;然后运行不同查询语句,观测系统运行性能.基于运行时性能数据,我们即可获得系统性能曲线和统计特征之间联系的初步认识.为了方便讨论,我们将已知与系统运行性能相关性较强的若干统计量列于表 2 中:

Table 2 The Variable List Used in the Sampling Process

表 2 采样过程使用的变量列表

Variables	Description
M	The Mapper number in the Map phase.
R	The Reducer number in the Reduce phase.
N	The Record number of the input data.
n	The number of sampled records.
f_i	The item counter of the ith frequency in n-sampled records.
$\overline{L_i}$	The average length of the ith field.
φ_i	The selectivity of the ith field.
d	The number of all possible values in n-sampled records.
D	The Distinct value.
$b_i=\theta(a_i)$	The frequency list of common values. b_i is the frequency of value a_i. Also the list length is restricted to be K.
$c_i=\xi(a_i)$	The histogram array. c_i is the statistical proportion of the interval (a_{i-1},a_i).

2.1 GroupBy 优化

本节主要讨论 GroupBy 查询(附带或者不带 Distinct 子句)的优化方案. 首先我们基于 Haas 和 Stokes[19] 提出的算法估算字段区分度 D. 参照表 2 中的变量定义,估算公式为 $D=\dfrac{nd}{n-f_1+\dfrac{f_1 n}{N}}$.

$D\geqslant 0$ 意味着该字段可行取值较少,数据呈偏斜分布的可能性非常高;$D<0$ 则说明该字段存在较多可能取值,需要进一步鉴定分布特征. 此时我们检查常用值列表,通过取值比例判断是否存在大量记录汇集分布的现象. 如果判定通过,即意味该字段已出现偏斜分布,需要优化查询生成过程. 根据上述讨论,我们制定出查询计划优化方案如表 3. 现将对应优化策略的理由说明如下:

1) 分支 A,Map 端的局部聚合操作. 可将大量数据聚合成少量局部统计值,因此无需开启多阶段 M/R 任务优化.

2) 分支 B,禁用打散优化选项. 可统计 GroupBy Key 的分布情况,根据统计信息得到合适的 Reduce 分区算法,将数据较均匀地分散到不同节点完成计数操作.

3) 分支 C,禁用打散优化选项. Map 端的局部聚合即可有效压缩数据量,因此没必要开启均衡性优化.

4) 分支 D,启用打散优化选项. 经过 Map 端局部聚合后,Distinct 操作需要处理的数据量依然很大,因此需要开启多阶段 M/R 任务优化.

5) 分支 E,禁用打散优化选项. 理由同分支 B.

Table 3 The Optimization Guidance Table for GroupBy Query

表 3 GroupBy 查询优化制导表

Query Type	Distribution Pattern	Optimization Plan
GroupBy (Without Distinct)	GroupBy Key: narrow	Branch A, disable multi-phase M/R optimization.
	GroupBy Key: wide	Branch B, disable multi-phase M/R optimization.
GroupBy (With Distinct)	GroupBy Key: narrow Distinct Key: narrow	Branch C, disable multi-phase M/R optimization.
	GroupBy Key: narrow Distinct Key: wide	Branch D, enable multi-phase M/R optimization.
	GroupBy Key: wide	Branch E, disable multi-phase M/R optimization.

2.2 Join 操作优化

在现实应用中,Join 操作对象通常为一张小表和一张大表,现针对该场景讨论 Join 查询的计算均衡优化方法. 类似上述 GroupBy 优化方案,我们先预估不同查询方案的运行代价. 考虑到 Reducer 需要完成排序操作,然而排序代价和数据具体特征关系较大,分配到不同 Reducer 的数据特征不尽相同,精确估计比较困难. 进一步考虑其他因素,例如处理器计算性能和磁盘 I/O 带宽等硬件因素、操作系统内核进程调度算法以及磁盘调度算法等软件因素以及当前系统负载状态和 Reducer 中执行的用户程序操作的不确定性,即使多次重复执行相同程序也难以重现特定的运行耗时规律. 因此,我们基于对比不同查询方案的网络传输数据量估计执行代价. 现假设参加操作的数据表分别为表 A 和表 B,大小分别为 $\|A\|$ 和 $\|B\|$,满足 $\|A\|\gg\|B\|$,即表 B 为小表.

2.2.1 Map 端 Join 代价估计

1) 在 Map 启动阶段,M/R 框架对每个持有表 A 数据分片的 Mapper 分发表 B 的数据副本,相应

数据传输量预估为表 A 分片数和表 B 数据量之间的乘积,即 $\|A\|+M\|B\|$. 接着执行 Map 端 Join 操作,对应代价记为 $\sum_{i=1}^{n} HashJoin(A_i,B)$.

2) 我们估算 Map 操作完成后推送至 Reduce 端的记录数,其中 Where 子句过滤出符合筛选条件的记录推送至 Reduce 端,包含 2 种情况:①求解等值过滤,我们结合常用值频度列表和直方图数据,估算选择率为目标值在频度统计表中的占用比例,即 $\varphi_i=\frac{b_i}{\sum_{i=1}^{n}\theta(a_i)}$;②求解区间过滤,我们通过归并等高直方图相应区间估算选择率. 设 (α_x,α_y) 为需要预估选择率的区间且满足 $a_m\leqslant a_x, a_y\geqslant a_n$,即 a_m 是最接近 α_x 的值,a_n 是最接近 α_y 的值,故有 $\varphi_i=\sum_{j=m}^{n} c_j$.

3) 将字段平均长度与预估筛选记录数相乘,获得 Mapper 向 Reducer 发送数据量的估计值,即该阶段数据传输量为 $N\varphi_i\overline{L_i}$,则 Map 端 Join 操作的整体代价估计为

$$TRANS(\|A\|+M\|B\|)+TRANS(N\varphi_i\overline{L_i})+\sum_{i=1}^{n} HashJoin(A_i,B).$$

2.2.2 Reduce 端 Join 代价估计

Reduce 端 Join 的 Map 阶段相对较简单,即该阶段仅完成数据分发操作,相应传输代价为 $TRANS(\|A\|+\|B\|)$. 数据分发完成后即触发全部 Reducer 开始工作,排序代价为 $\frac{SORT(\|A\|+\|B\|)}{R}$,则 Reduce 端 Join 的整体代价估计为

$$TRANS(\|A\|+\|B\|)+\frac{SORT(\|A\|+\|B\|)}{R}.$$

至此,我们基于网络传输代价对不同查询计划建立了估价模型. 显然只需对比上述 2 个算式的具体取值,指导查询计划生成器生成代价较小的方案即可.

3 系统设计与实现

总结以上讨论可知,这 2 类查询优化的关键是根据数据分布情况合理选择查询计划,从而达到优化查询性能的目的. 我们设计了 2 种实现手段:1)手动方式. 用户通过配置选项调整查询翻译器行为,强制生成多阶段 M/R 任务,使用场景仅限于显式数据倾斜以及数据统计规律已知的场合. 2)自动方式. 手动优化数据仓库中所有数据表的代价显然太高,而且许多数据表的统计分布情况在后续更新过程中可能发生剧烈改变,再考虑到各种人为误差,自动优化机制势在必行;然而实现自动查询优化机制需要解决 2 个关键问题,即判定目标字段是否具备偏斜分布特征以及分析查询语义,条件满足时触发查询优化. 本节着重讨论自动优化方式的实现.

3.1 基于 M/R 框架的统计过程实现

根据第 1 节讨论可知,获得目标字段的统计信息是实现计算均衡的先决条件. 我们关注的统计信息包括 2 类:1)表/分区级统计信息,主要包括记录数、磁盘空间占用、文件数、文件块数等;2)字段级统计信息,主要包括字段平均长度、字段区分度以及 Key 值分布情况等.

在现有 Hive 工作流内,我们确定了统计信息收集的 2 个关键时机:在原始数据入库阶段收集统计信息或者利用系统闲置时间增量更新已入库数据的统计信息. 考虑到我们面临的数据规模,设计和实现统计流程需要解决若干问题,即时间复杂度要求,为保证系统以可接受的时空代价处理规模极大的数据流,统计过程必须具备次线性处理速度,该要求可以通过采样手段满足;增量更新要求,极为庞大的数据规模要求采样过程能在单遍扫描过程中完成增量更新,这使得多遍扫描的重采样方式不再适用;结合考虑 M/R 框架的 SIMD 特征,每节点处理彼此独立的并行数据流,计算流程中信息严格单向流动,不存在反馈回环. 针对这些特点,我们设计的采样方法将完整统计过程分割为若干独立操作算子,通过这些算子相互协作以完成统计信息收集.

ETL 阶段实现的采样器由 SELECT 谓词驱动,即在 SELECT 谓词编译生成的行扫描器中我们实现了基于滑动采样窗口的 Haas 统计算法,该算法按照 M/R 框架的组织分为 Map 和 Reduce 阶段,如图 4 所示. 为方便工程实现以及清晰组织,我们将统计信息收集任务进一步拆分为 3 个子过程:1)全局统计算子,负责统计部分简单常量如记录数、平均字段长度等;2)采样算子,实现对原始数据的过滤和采样;3)直方图算子,完成对关键特征值的直方图统计. 采样流程大致可分解为 3 个阶段:1)数据输入阶段,M/R 任务启动后,Mapper 从输入文件中解析获得一行文本数据,作为下一阶段的输入. 2)数据处理阶段,全局统计算子负责统计记录数和空值字段,计

算字段平均长度;采样算子对输入记录进行过滤,将采样记录移交至直方图统计算子处理,输出结果包括常用值频度图和等高直方图. 3)数据输出阶段,当所有 Mapper 阶段完成后可获得的特征值包括该 Mapper 处理的总记录数、空值比例、字段平均长度以及各字段键值分布直方图.

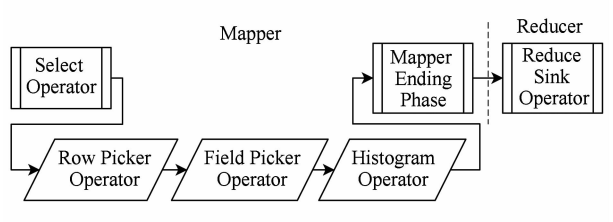


Fig. 4 Sampler implementation based on the M/R framework in Hive.

图 4 在 Hive 中实现的基于 M/R 框架的采样器

由于 ETL 流程处理的数据通常是文本格式,缺少了数据分块的整体信息(例如单个数据分片中包含的记录数),因此不能直接套用经典采样算法. 目前我们采取了 2 种应对策略:1)阈值截断读取,采样算子接收来自于上个操作算子转发的数据记录,如果达到了预定采样阈值则直接丢弃,否则转发给直方图统计算子;2)间隔采样,观察前若干条记录的平均数据长度,根据该数据长度和对应数据分块总长度估计该数据分块中包含的记录数 N',之后由采样算子根据该参数完成数据采样,由直方图统计算子完成直方图统计.

Reduce 阶段汇总 Map 阶段输出的数据,通过如下操作最终确定目标字段统计参数:累加所有 Mapper 处理的记录数得到总记录数;累加所有 Mapper 空值计数,与总记录数相除即获得字段占空比;将不同 Mapper 输出的字段平均长度按比例变换后获得平均字段长度.

直方图算子对采样记录完成如下统计操作:常用值统计,即累加常用值频率;等高直方图统计,借助 M/R 内建的排序机制将每个 Mapper 输出采样记录进行排序,然后指定直方图区间数目,将所有记录值划分至不同采样区间,最后输出等高直方图统计结果.

在 M/R 结束阶段,调用 HDFS API 将统计获得的总数据量、文件量、总块数等信息更新至元数据库中. 在翻译查询语句时,可直接读出这些信息并根据预定义策略执行查询优化.

3.2 针对已入库数据的静态采样

对于已获得统计信息的数据集而言,数据集通常会随着业务增长而扩容并更新,这意味原有统计信息也需要完成相应更新;同时顾及不断扩增的数据规模,也必须对统计算法的运行时间加以限制. 同时考虑这 2 方面需求,我们认为优化统计过程的性能十分必要. 分析上述统计流程可知,关键步骤包含了分割、计算和汇总阶段. 从底层文件系统的存储视图来看,已入库数据是存放于不同数据节点的文件块. 这暗示着我们可通过 HDFS API 获取新增数据块的文件信息,针对性发起 I/O 访问,而后执行增量统计. 依据此思路,我们设计的离线采样方法解决了统计信息的增量更新问题,具体步骤为:1)通过 HDFS API 获得新入库数据的存储位置;2)对这些数据块进行采样操作;3)将采样结果更新至对应元数据文件中. 我们使用 C 语言实现了增量数据的独立统计程序,在计算性能方面获得了较显著的提升,稍后将在实验部分给出相应实现的性能评价.

现考虑统计信息时效性的问题. 如果当前入库数据集发生变化但相应统计信息尚未及时更新,则优化查询时不得不沿用陈旧统计信息;反之,如果统计信息更新完毕,那么将依据最新统计信息优化查询. 为保证统计信息时效性,我们通过后台线程定期监测并追踪数据集的状态变动. 如果发现目标数据表已更新了统计信息,将依据目标数据集的当前统计信息制定优化决策. 总之,该方法要点在于直接访问文件数据实现采样逻辑,节省 M/R 框架的运行开销;优化程序实现,提升数据计算吞吐率;在系统相对空闲时启动采样任务,从而有效提升了系统的整体利用率,避免对前台任务造成性能影响.

4 性能评价

4.1 实验环境与数据

为验证第 2 节和第 3 节论述的正确性以及 CBM 模型的实际应用性能,我们全面测试了 CBM 模型的优化效果,对比目标是原始版本的 Hive 系统. 实验环境为由 20 台物理节点组成的集群,每节点硬件配置如下:1)CPU 型号为 AMD Opteron™ Processor 6132 HE,主频 2.2 GHz;2)内存容量为 64 GB;3)硬盘控制器连接 2 个 Seagate ST31000524NS SATA2 硬盘,容量均为 1TB;4)以太网卡最大工作速率为 1 000 MBps. 软件堆栈包括 CentOS 5.5 操作系统,对应 Linux 内核版本为 3.7.1,JDK 版本为 1.6,Hadoop 版本为 1.1.2,Hive 版本为 0.13.0.

Hadoop 集群配置方式如下:1 个物理节点配置

为 Namenode;1 个物理节点配置为 JobTracker;剩余 18 个物理节点配置为 Datanode 和 TaskTracker. 实验数据集使用 TPC-H 基准测试生成,选用 Lineitem 表作为大表,包含 16 个字段,约 8.65 亿条记录,累计数据量为 100 GB;将 Nation 表进行扩容后选作小表,包含 4 个字段,约 160 万条记录,累计数据量约为 100 MB. 我们修改了 Lineitem 表的原始范式,新增了一个用于和 Nation 表关联的外键;最后我们使用统计分布生成程序对表数据进行些许改动,使之匹配特定分布规律,以用于测试 CBM 模型的有效性.

4.2 统计信息收集性能测试

该实验目的是验证数据采样过程的性能,主要评估指标为运行耗时. ETL 过程的性能测试结果如图 5~6 所示,其中横坐标对应输入数据规模,纵坐标对应运行时间. 从图 5 可以看到,附加的采样程序对 ETL 造成的性能影响有限,额外开销基本控制在 30%以下. 我们认为主要原因是 ETL 属于 I/O 密集型操作,而信息统计过程主要消耗 CPU 时间,由于两者不存在硬件资源竞争,可彼此独立运行,且适当设置采样区间可有效降低数据依赖导致的延迟;同时我们发现适当扩大采样规模并不会显著增加运行开销. 从图 6 可以看到,基于 C 语言的增量采样实现的性能在一定程度上优于基于 M/R 框架实现的采样机制,然而优化效果比较有限,性能瓶颈主要来自于磁盘 I/O 能力限制. 基于该实验结论,我们可通过先入库少量数据,然后以增量方式批量入库剩余数据的方式显著改善系统的 ETL 性能.

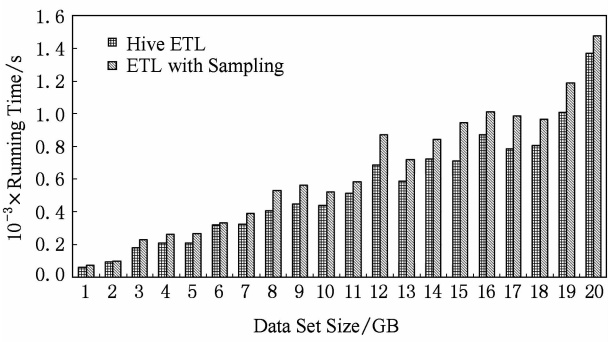


Fig. 5 Performance comparison of ETL with and without sampling.

图 5 附加采样过程的 ETL 过程和原始过程性能对比

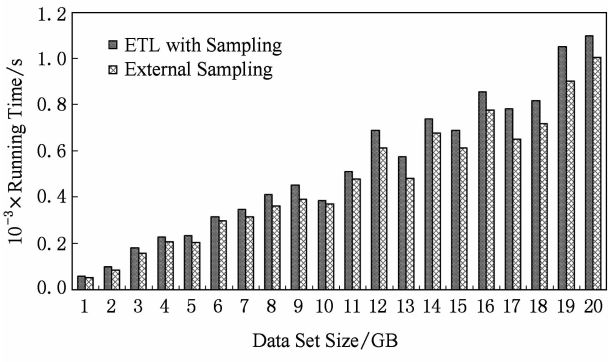


Fig. 6 Performance comparison of the sampling procedure implemented in different languages.

图 6 基于不同语言实现的采样过程性能对比

4.3 统计信息误差测量

该实验主要评估统计结果的误差率. 由于统计量中的平均字段长度、频度值列表以及 Distinct 值与最终查询计划优化过程关系最大,因此我们侧重评估这些统计量的误差. 我们对单个本地文件运行采样算法获得精确统计值,考虑到完全采样生成精确统计值极为耗时,因此我们控制实验数据规模为 1~10 GB. 频度值列表误差估算包含 2 部分:

1) 和精准统计相比,频度值列表项的差异数. 定义精确频度值 Top-K 集合 $\omega=\{a_i \mid 1\leqslant i\leqslant k\}$,采样结果集为 $\partial=\{b_i \mid 1\leqslant i\leqslant k\}$,则差异数为 $\Delta=|\omega-\partial|$.

2) 和精准统计相比,相同频度值列表项对应的频度差值 λ 计算为

$$\lambda_i=\frac{(\beta_i-\alpha_i)}{\alpha_i}\times 100\%,$$

$$\lambda=\sum_{i=1}^{k}|\lambda_i|.$$

其中,α_i 是第 i 个统计项的精确频度,β_i 为对应的采样统计值频度.

实验结果如图 7~8 所示,图 7 表明了字段平均长度和区分度值的误差基本控制在 10%以内,因此我们认为该采样算法的精度是可接受的. 图 8(a)表示和精确频度列表项不一致的统计项数目,本实验中频度列表差异项少于 5 个;图 8(b)表示统计项和精确频度值差异比例的累加值,本实验中对应项差异比例均控制在 10%以下,从而反映了字段频度统

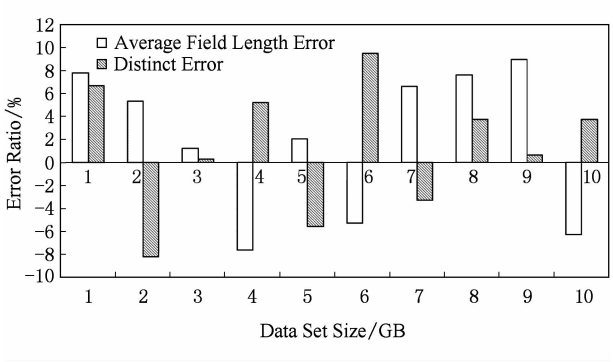


Fig. 7 Error estimate of the average field length and the distinct value for the sampling result.

图 7 采样结果平均字段长度和区分度值的误差评估

计结果是可信的.然而考虑到实验数据规模,获得完全精确的统计特征值既不现实也非必要,主要包括如下原因:1)后续追加数据可能影响原有数据分布,甚至完全颠覆原先分布特征;2)采样过程最终是为CBM模型服务,只要能保证最终生成最优查询计划,即使统计结果存在少许误差,我们仍承认该统计结果的有效性.

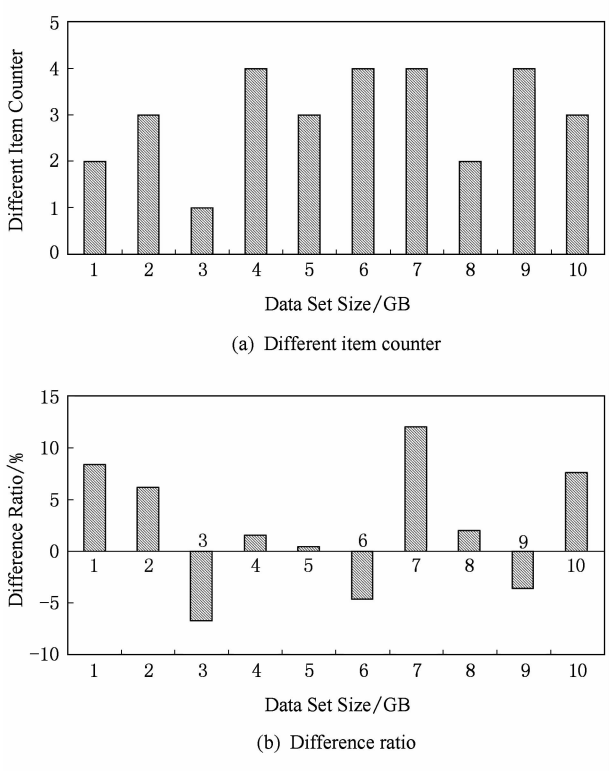


Fig. 8 Difference between the statistic result and the precise values.

图8 统计结果与精确值之间的差异

4.4 查询优化实验

对于查询优化实验,我们采取的验证思路为:首先对查询操作字段执行统计,将统计信息反馈至CBM模型;然后将系统设置为自动优化制导,对不同数据集合分别运行优化查询和非优化查询;最后对比评估不同查询计划的运行效果.评估标准分为2部分,包括查询运行耗时对比以及查询过程中系统负载分布情况.特别指出,优化开关只有2种可取状态,再考虑到2个额外因素:1)优化估价模型本身并不完全精确;2)统计值也存在一定误差.因此,我们提出优化成功的判定条件为在CBM查询优化器的指导下,执行目标查询时能获得更优的性能和更好的负载均衡度.

4.4.1 查询耗时对比测试

本实验分别测试了启用CBM模型后,GroupBy和Join查询耗时对比情况.实验中使用的查询语句的一般形式为:1) Select Count(Distinct *FIELD_X*) From *TABLE* Group By *FIELD_Y*,目的是验证CBM模型能否正确处理GroupBy子句和Distinct子句混合使用的情况;2) Select *TABLE_X*.* From *TABLE_X* Join *TABLE_Y* On (*TABLE_X*.*id*=*TABLE_Y*.*id*),目的是验证CBM模型能否正确优化Join查询.我们在实验中测试了20组查询语句,实验结果如图9~10所示,横坐标表示查询语句序号,纵坐标对应该查询语句的运行耗时.实验数据表明,对GroupBy查询而言,CBM最高加速比可达1.9;对Join查询而言,CBM最高加速比可达1.8.

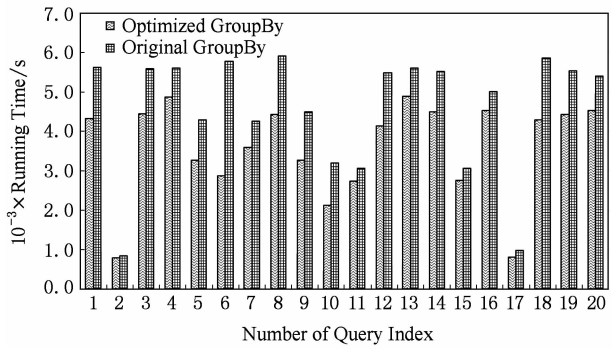


Fig. 9 Performance comparison of the GroupBy queries based on CBM optimization.

图9 基于CBM优化前后的GroupBy查询性能对比

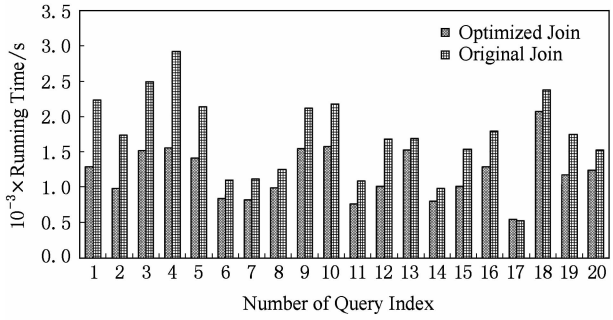


Fig. 10 Performance comparison of the Join queries based on CBM optimization.

图10 基于CBM优化前后的Join查询性能对比

4.4.2 集群负载均衡化测试

本实验目的是评估集群负载情况.我们基于均方差量化集群负载的均衡程度.不妨设集群内节点负载标准方差为σ;主机i性能度量值为γ_i,代表CPU利用率或I/O负载,使用Linux的uptime以及iostat工具测量,则

$$\bar{\gamma}=\frac{1}{N}\sum_{i=1}^{N}\gamma_i,$$

$$\sigma=\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\gamma_i-\bar{\gamma})^2}.$$

为确保实验结果可靠性,我们以独占方式运行查询任务,尽可能排除不确定因素的干扰.我们从图 11 看到,对 GroupBy 查询而言,CPU 负载 σ 值平均改善了约 72.9%,I/O 负载 σ 值平均改善了约 63.9%;从图 12 看到,对 Join 查询而言,CPU 负载 σ 值平均改善了约 62.3%,I/O 负载 σ 值平均改善了 82.2%.本实验证实了 CBM 确实能显著改善集群负载的均衡程度.

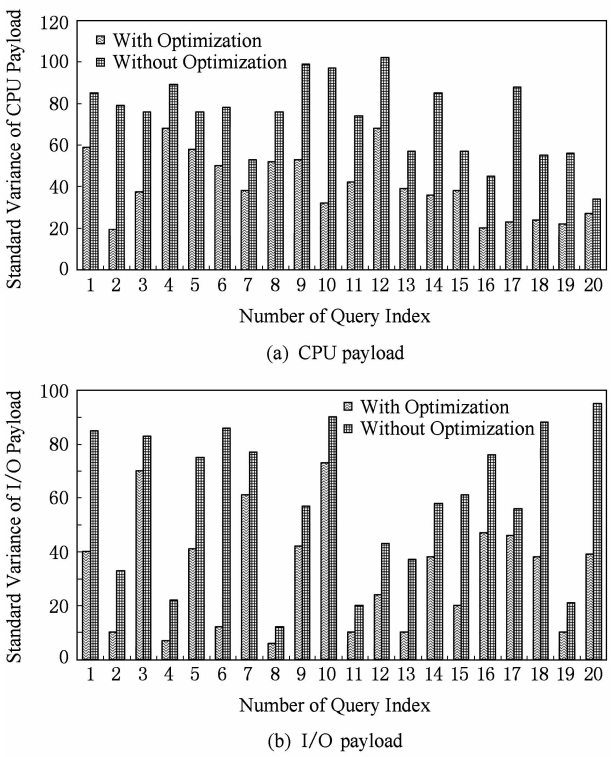


Fig. 11 Payload distribution comparison of CPU and I/O payload for queries optimized by CBM (GroupBy query).

图 11 基于 CBM 优化前后的集群负载情况(GroupBy 查询)

4.5 实验总结

采样统计实验证明了我们提出的统计信息收集算法可有效处理规模极大的数据集,获得的统计信息精度就优化查询的需求而言是可接受的;查询优化实验证明了基于 CBM 模型指导优化的 GroupBy 和 Join 查询能很好地处理偏斜分布的数据集,且较好地均衡化集群工作负载.此外,查询耗时也相应地获得改善,在较好情况下可达到 1.8 倍左右的加速提升.

综上所述,我们认为 CBM 模型具备如下优势:1)有效解决了原始 Hive 系统在处理偏斜数据时可能出现负载不均的问题,显著优化了查询执行效率;

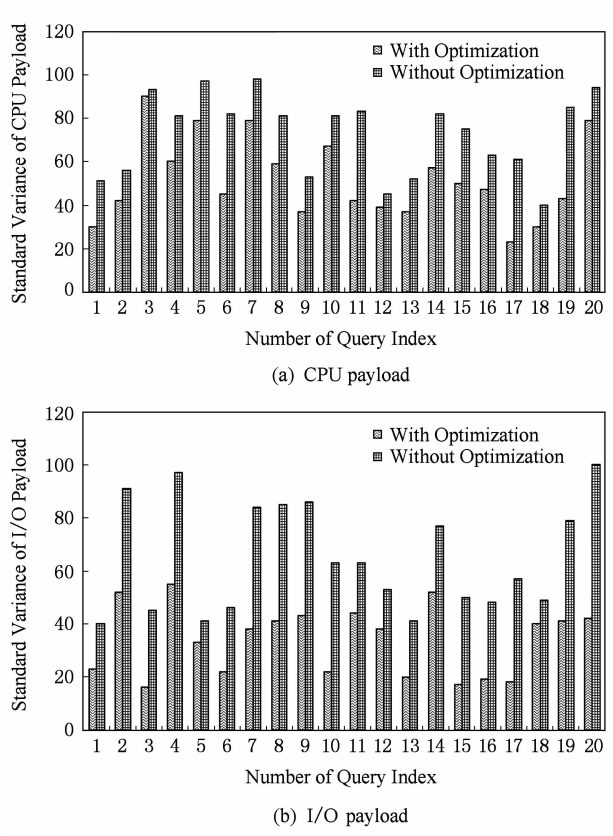


Fig. 12 Payload distribution comparison of CPU and I/O payload for queries optimized by CBM (Join query).

图 12 基于 CBM 优化前后的集群负载情况(Join 查询)

2)获得优化效果所需的额外开销相对较小.事实上我们还可通过多种方式隐藏统计信息收集的开销,如后台线程、ETL 过程内联运行或嵌入其他查询任务中等方式实现统计开销最小化.

5 结束语

原始 Hive 使用 M/R 框架处理偏斜数据时可能出现计算负载不均的问题.为解决该问题,本文提出了一种优化查询的计算均衡模型 CBM,关键内容包括统计收集海量数据的分布信息,据此指导优化查询计划生成,确保了计算任务在集群中的均衡分布,最终提升了实际查询语句运行效率.针对 CBM 模型的实验结果表明,该模型的统计分析开销处于可接受的范围内,使得 GroupBy 和 Map 端 Join 查询在处理偏斜数据时获得了显著性能提升.基于这些事实,我们认为 CBM 模型已经达到可投入实际生产应用的水平,达到了设计和实现系统的预期目的.

本研究的未来工作包括从算法设计和程序实现层面优化统计算法的运行效率,并提升统计输出的

精准度;探索查询耗时和数据分布规律之间的更深层联系,提升 Hive 查询耗时估价模型的精确度;进一步细化查询优化策略的选择分支等.

参 考 文 献

[1] Ghemawat S, Gobioff H, Leung S T. The Google file system [C] //Proc of the 19th ACM Symp on Operating Systems Principles. New York: ACM, 2003: 29-43

[2] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [C] //Proc of the Operating Systems Design & Implementation'04. Berkeley, CA: USENIX Association, 2004: 137-150

[3] Shvachko K, Kuang H, Radia S, et al. The Hadoop distributed file system [C] //Proc of the 26th IEEE Symp on Mass Storage Systems and Technologies (MSST 2010). Piscataway, NJ: IEEE, 2010: 1-10

[4] Thusoo A, Sarma J S, Jain N, et al. Hive: A warehousing solution over a Map-Reduce framework [J]. Proceedings of the VLDB Endowment, 2009, 2(2): 1626-1629

[5] Blanas S, Patel J M, Ercegovac V, et al. A comparison of join algorithms for log processing in MapReduce [C] //Proc of 2010 ACM SIGMOD Int Conf on Management of Data (SIGMOD'10). New York: ACM, 2010: 975-986

[6] Kwon Y, Balazinska M, Howe B, et al. A study of skew in MapReduce applications [C] //Proc of the Open Cirrus Summit 2011. Piscataway, NJ: IEEE, 2011

[7] Ibrahim S, Jin H, Lu L, et al. LEEN: Locality/fairness-aware key partitioning for MapReduce in the cloud [C] //Proc of the 2nd IEEE Int Conf on Cloud Computing Technology and Science (CloudCom 2010). Piscataway, NJ: IEEE, 2010: 17-24

[8] Okcan A, Riedewald M. Processing Theta-Joins using MapReduce [C] //Proc of 2011 ACM SIGMOD Int Conf on Management of Data (SIGMOD'11). New York: ACM, 2011: 949-960

[9] Hassan M H, Bamha M. Semi-join computation on distributed file systems using Map-Reduce-Merge model [C] //Proc of 2010 ACM Symp on Applied Computing (SAC'10). New York: ACM, 2010: 406-413

[10] Zhang S, Han J, Liu Z, et al. SJMR: Parallelizing spatial join with MapReduce on clusters [C] //Proc of the IEEE Int Conf on Cluster Computing and Workshops. Piscataway, NJ: IEEE, 2009: 1-8

[11] Gufler B, Augsten N, Reiser A, et al. Handing data skew in MapReduce [C] //Proc of the 1st Int Conf on Cloud Computing and Services Science. Piscataway, NJ: IEEE, 2011: 574-583

[12] Ramakrishnan S R, Swart G, Urmanov A. Balancing reducer skew in MapReduce workloads using progressive sampling [C] //Proc of the 3rd ACM Symp on Cloud Computing (SoCC'12). New York: ACM, 2012: 16:1-16:14

[13] Yang H C, Dasdan A, Hsiao R L, et al. Map-Reduce-Merge: Simplified relational data processing on large cluster [C] //Proc of 2007 ACM SIGMOD Int Conf on Management of Data(SIGMOD'07). New York: ACM, 2007: 1029-1040

[14] Ananthanarayanan G, Agarwal S, Kandula S, et al. Scarlett: Coping with skewed content popularity in MapReduce clusters [C] //Proc of the 6th Conf on Computer Systems (EuroSys'11). New York: ACM, 2011: 287-300

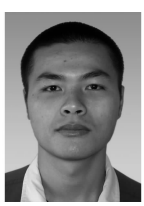
[15] Vernica R, Balmin A, Beyer K S, et al. Adaptive MapReduce using situation-aware mappers [C] //Proc of the 15th Int Conf on Extending Database Technology (EDBT'12). New York: ACM, 2012: 420-431

[16] Zhao Yanrong, Wang Weiping, Meng Dan, et al. Efficient join query processing algorithm CHMJ based on Hadoop [J]. Journal of Software, 2012, 23(8): 2032-2041 (in Chinese) (赵彦荣, 王伟平, 孟丹, 等. 基于 Hadoop 的高效连接查询处理算法 CHMJ [J]. 软件学报, 2012, 23(8): 2032-2041)

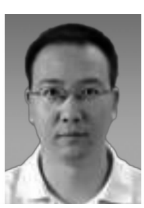
[17] Hammoud M, Sakr M F. Locality-aware reduce task scheduling for MapReduce [C] //Proc of the 3rd IEEE Int Conf on Cloud Computing Technology and Science (CloudCom 2011). Piscataway, NJ: IEEE, 2011: 570-576

[18] Kolb L, Thor A, Rahm E. Load balancing for MapReduce-based entity resolution [C] //Proc of the 28th IEEE Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2012: 618-629

[19] Haas P J, Stokes L. Estimating the number of classes in a finite population [J]. Journal of the American Statistical Association, 1998, 93(444): 1475-1487



Wang Youwei, born in 1986. PhD candidate of the Institute of Computing Technology, Chinese Academy of Sciences. His current research interests include distributed file systems, storage and process of big data.



Wang Weiping, born in 1975. PhD, professor of the Institute of Information Engineering, Chinese Academy of Sciences. Member of China Computer Federation. His current research interests include database technologies, storage and process of big data, and cloud computing(wangweiping@iie.ac.cn).

Meng Dan, born in 1965. PhD, professor of the Institute of Information Engineering, Chinese Academy of Sciences. Fellow member of China Computer Federation. His current research interests include cloud computing and system security (mengdan@iie.ac.cn).