·先进计算与数据处理· 文章编号:1000-3428(2015)07-0048-07 文献标识码:A 中图分类号:TP18

MapReduce 环境下支持大规模文本检索的概念索引

张 生,胡加靖

(上海理工大学光电信息与计算机工程学院,上海 200093)

摘 要:随着信息化技术飞速发展,爆炸性数据的增长以及数据的多样化给大数据检索带来了挑战。MapReduce 作为一种并行处理框架,在大数据处理上具有明显优势。为此,结合概念格的相关知识,采用形式概念分析发现文档之间的关系并用格进行表示,提出一种新型的支持大规模文本检索的形式概念索引结构,给出基于 MapReduce 框架建立概念索引的相关算法。通过与 Lucene 索引进行比较,验证了所提索引的有效性。实验结果表明,将文档之间关系采用概念格表示并建立概念索引,能够提高大规模文本检索的性能。

关键词:大数据;MapReduce 框架;数据检索;形式概念分析;概念格;概念索引

中文引用格式:张 生,胡加靖. MapReduce 环境下支持大规模文本检索的概念索引[J]. 计算机工程,2015,41(7):48-54.

英文引用格式:Zhang Sheng, Hu Jiajing. Concept Index Supporting Large Scale Text Retrieval Under MapReduce Enviroment[J]. Computer Engineering,2015,41(7):.48-54.

Concept Index Supporting Large Scale Text Retrieval Under MapReduce Enviroment

ZHANG Sheng, HU Jiajing

(School of Optical-Electrical and Computer Engineering,
University of Shanghai for Science and Technology, Shanghai 200093, China)

【Abstract】 With high speed developing of the informatization, the coming of big data era brings some revolution to the world, and it becomes a challenge for big data searching by its explosive growth and variety. MapReduce is commonly used in processing big data and shows its great advantages. Combined with the relative knowledge of lattice, this paper uses Form Concept Analysis (FCA) to discover the relationships among textual documents and expresses them with lattice, and proposes a novel conceptually index structure, which supports large scale data retrieval. In addition, it describes the related algorithms for building conceptual index. Compared with Lucene index, conceptual index supporting queries has better efficiency. Experimental results show that using lattice to express the relationship of documents and indexing it with conceptual can significally improve the performance of large scale documents retrieval.

【Key words】 big data; MapReduce framework; data retrieval; Formal Concept Analysis (FCA); concept lattice; conceptual index

DOI:10.3969/j.issn.1000-3428.2015.07.009

1 概述

近年来,随着 Web 数据爆炸性的增长,如何高效地从大规模文本数据集中快速检索到用户需要的信息,已成为进一步提高 Web 信息检索效率亟待解决的问题之一。为了使文本检索更加快速有效,一个良好的索引结构成为文本检索中的关键,也是研究者普遍应用的方法。

传统集中式索引已无法满足大数据集上检索性能的要求,因此,研究人员提出了分布式索引结构,这种索引分布在不同的机器上且支持并行查询[1],但是这些索引技术需要完全本地化数据,在数据量很大的情况下难以实现。2008 年 Google 提出了 MapReduce[2]分布式编程模型,它采用"分而治之"的思想,且隐藏了复杂的并行处理、容错和可用性等机制,使得大数据分布处理程序的执行变得简单。

基于 MapReduce 框架,研究者们已经建立分布式索引结构,最广泛使用的是倒排索引技术[3]。倒排索引是全文搜索引擎的核心部分,其倒排表通常由关键词和其对应的倒排列表组成,即 <*Key*, *Value*>

作者简介:张 生(1968-),男,高级工程师,主研方向:云计算,数据挖掘;胡加靖,硕士研究生。

收稿日期:2014-05-19 修回日期:2014-07-15 E-mail:hujiajing0216@126.com

键值对(*Key* 表示单词,*Value* 表示倒排列表)它包括 2 个元素:文档标识号 *docid* 和记录该关键词在该文档中出现的相关信息 *payload*。在 MapReduce 架构上建立倒排索引时,Map 阶段用于对文档进行分词和统计,Reduce 阶段则是对 Map 阶段产生的结果进行分类合并,最后形成倒排索引文件。但是倒排索引并未考虑文档对象之间的关系,而文档之间的关系对于文本检索的质量和效率有一定的影响。

为发现文档对象的关系,本文采用形式概念分析(Formal Concept Analysis,FCA)对文档集进行描述。FCA 是一种从形式背景进行数据分析和规则提取的强有力工具,其核心数据结构是概念格,它被用来揭示数据对象和属性之间的关系,表明概念之间的泛化和例化关系[4],目前,FCA 已在数据挖掘、信息检索等领域中得到广泛应用[5]。

如何利用 FCA 描述文档对象之间的关系,并开发一个分布式高效索引结构以支持大规模文本查询,是本文研究的关键点。本文首先使用 FCA 表示文档背景以发现文档对象之间的关系,然后通过连接形式概念格(Formal Concept Lattice,FCL)的方式显示不同文档对象和其属性之间的关系,最后基于 MapReduce 建立分布式概念索引结构,即实现以概念的形式索引文档数据。本文设计相应的概念索引结构构建算法,并通过与流行的索引结构 Lucene 进行比较验证该算法的性能。

2　相关工作

随着人们对大数据关注度不断增加,如何提高大规模文本数据的检索效率成为学者们研究的热点,其中建立索引是最普遍的方法。常用的索引结构有倒排索引[3]和签名索引[6]。倒排索引是通过属性值查找记录,采用键值对的形式表示属性值在记录中的相关信息。但是,在倒排索引上进行检索(不用 Hash)时,需要与关键字逐一比对,导致检索低效,不过在文献[7]中已通过减少查询过程中的条目检查,提出优化策略。签名索引则是为每个关键字分配一个固定大小的向量,称为签名。但对于较大的文本文件,必须进行分块处理,检索时需要顺序扫描签名导致速度慢,且签名向量的大小选择也不定。

此外,传统的倒排索引大都运行在单机上,对于大规模文本索引的建立会受限于可用内存的大小。但自从 Jeffery 等人提出 MapReduce 架构后,将索引建立算法分布到集群的多台机器上执行是当今搜索引擎所运用的技术,文献[8]实现了在分布式集群中为大规模文档建立倒排索引,Map 函数解析文档集,发射形如 $<word, doc\_ID>$ 的序列,Reduce 函数对于给定的 *word* 接受所有的序列,并且对 *doc_ID* 进行排序,生成形如 $<word, list(doc\_ID)>$ 的键值对,最后将所有结果输出形成一个简单的倒排索引表。文献[9]提出了第一个基于 MapReduce 平台的概念格生成算法,它使用迭代若干个 Map 和 Reduce 过程实现,但未考虑形式背景在节点上的划分。文献[10]则横向划分形式背景,采用 merge 方法合并最终形成的概念。但是,横向划分时需要知道所有对象的属性,这在大数据背景下是难以实现的。

为提升文本检索效率,又考虑到目前已有的索引策略均未利用文档之间的关系构建 MapReduce 上的索引,本文利用 FCA 发现文本文档之间的关系,并用格进行表示,提出一种新颖的概念索引技术。为了使概念索引更适合管理大数据集,将在 MapReduce 框架上构建此索引,该索引不仅存在于整个集群上,对部分文档集也有文档之间的关系表示。该索引使得网络占用和查询时间都大大减少,有效提高了检索性能。

3　背景知识

形式概念分析的基础理论是概念格,概念格是一种概念分析数据的数学方法[11]。在形式概念上,域用内涵和外延表示,通常,外延由对象组成,内涵由对象属性组成。下面介绍如何使用 FCA 理论表达文档之间的关系。

用 FCA 分析数据的第一步是构建形式背景,形式背景由对象、对象的属性和它们之间的关系组成。此处将文档作为对象,词语当作属性,如果文档包含该词语,就看作一种关系。因此,本文将文档检索中的形式背景称为文档背景,其定义如下所示:

定义 1　文档背景是一个三元组 $C(D,T,R)$,其中,D 代表所有文档的集合;T 是文档 D 中出现的所有词的集合;R 代表 D 和 T 之间的关系;$R = D \times T$。$d \in D, t \in T, r \in R$,如果 t 在 d 中出现,则 $r = 1$;否则,$r = 0$。

例如,设 $D = \{doc1, doc2, doc3, doc4, doc5\}$,$D$ 是文档的集合;$T = \{printer, book, price, dictionary, car\}$,$T$ 是文档中所有词的集合。文档背景则表示词语是否被这篇文档包含,如表 1 所示。其中,行表示 T 中的词,列是 D 中的文档号,如果格的值为 1,表明这篇文档包含这个词,否则就表示不包含。但是,文档背景表达了文档和词之间的关系,却不能直观地描述文档之间的联系,由于在 FCA 中,概念格通常按照背景来组织对象,因此本文使用概念格来表达文档背景。图 1 是与文档背景表 1 对应的概念格。

表 1　文档背景

doc	printer	book	price	dictionary	car
doc1	1	1	1	0	0
doc2	1	0	1	0	1
doc3	1	0	0	1	0
doc4	0	0	0	1	1
doc5	1	0	0	1	1

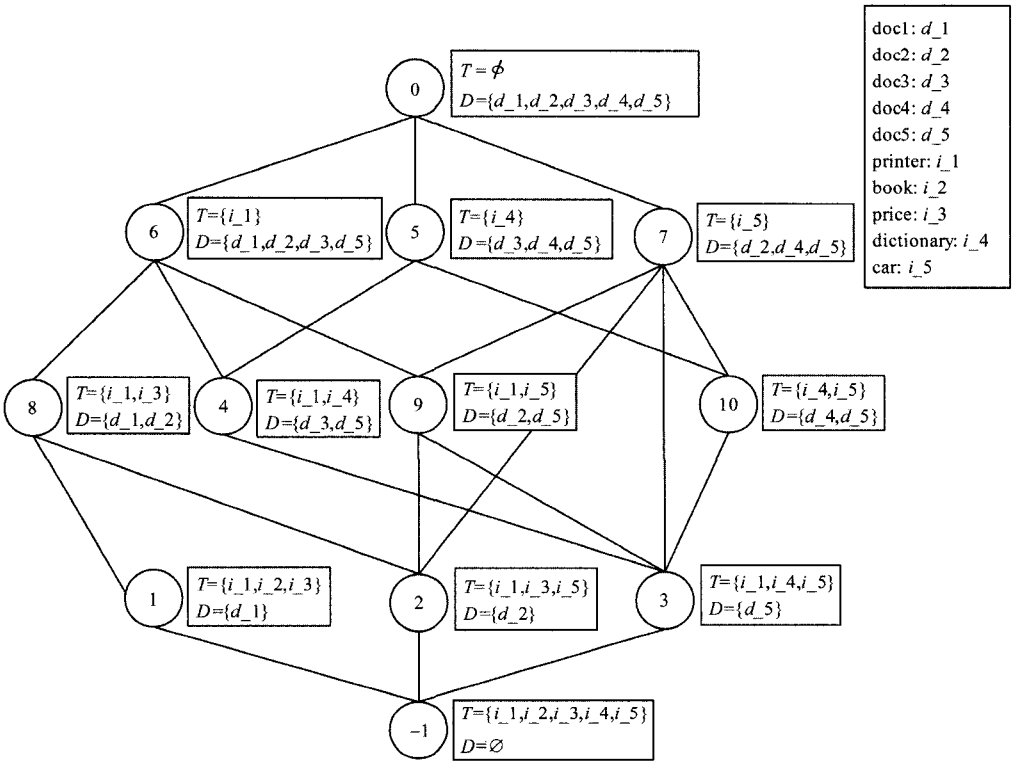

图 1 表 1 对应的概念格

定义 2 给定一个文档背景 $C(D,T,R)$，L 是文档背景 $C(D,T,R)$ 上的索引，称为概念索引。它们有以下 2 个性质：

性质 1 假设 $N(T,D)$ 是 L 上的一个节点，T,D 分别是 N 的内涵和外延，T 和 D 满足 $f(T)=D$，$g(D)=T$，其中，$f(T)=\{t\in T\mid \forall d\in D,(d,t)\in R\}$，$g(D)=\{d\in D\mid \forall t\in T,(d,t)\in R\}$。$N(T,D)$ 是 $C(D,T,R)$ 中的形式概念或者简称概念。

性质 2 给定 L 中的 2 个邻接概念 $N1(T1,D1)$ 和 $N2(T2,D2)$，如果存在 $T_1\subset T_2$ 且 $D_2\subset D_1$，则 $N1$ 称为 $N2$ 的子索引即孩子，或者 $N2$ 为 $N1$ 的父索引即父亲。记为 $N_1<N_2$。

在概念索引中，一个节点 $N(T,D)$ 即为对应 FCA 中的一个形式概念，为了使节点便于在 MapReduce 平台上执行相关操作，本文将节点 $N(T,D)$ 中的 T 作为 *key*，D 作为 *value*。因此，在后文中，如果没有特别的说明，$N(key,value)$ 仅代表一个形式概念。例如，对于图 1 中的 2 个邻接点 *Node*2 <(printer, price, car), (doc_2) > 和 *Node*9 <(printer, car), (doc_2, doc_5) >，其中词的列表是内涵，对应的文档列表是外延。*Node*2 中，(printer, price, car) 在文档 2 中出现，*Node*9 中(-printer, car) 在文档 2 和文档 5 中出现。由于 *Node*2 的 *key*(printer, price, car) 包含 *Node*9 的 *key*(printer, car)，且 *Node*2 的 *value*(doc_2) 被 *Node*9 的 *value* (doc_2, doc_5) 所包含，因此，*Node*2 < *Node*9。

4 文本概念索引的建立

4.1 文档背景的存储与 Map 函数

由于每个文档中的单词只是词典中的少部分，当文档集较大时，会造成文档背景很稀疏，如果采用常用的矩阵存储，将会造成低效的空间存储率。因此，本文采用一系列的二元组存储文档背景，即 $\{(r_1,c_1),(r_1,c_2),\cdots,(r_n,c_m)\}$，其中，$r_i$ 表示行 *id*；c_j 表示列 *id*，(r_i,c_j) 则表示文档 doc_i 包含单词 t_j（文档背景中的元素为 1 的值），如表 1 所示的存储形式即为{(1,1),(1,2),(1,3),(2,1),(2,3),(2,5),(3,1),(3,4),(4,4),(4,5),(5,1),(5,4),(5,5)}。根据形式背景（表 1）和概念索引（图 1）的定义，可知计算概念归结为从文档子集中提取公共词语。然而，不是所有词语和文档都有关系，因此，产生形式概念时考虑形式概念特点可提高效率。

首先定义一个所有文档子集上的字典序，文档集合被简化为 $D=\{1,2,\cdots,n\}$，其中，D 中的文档用 *doc_ID* 来表示。而且，D 的每一个子集也是字典序的。下面定义文档集合中任意 2 个子集排序方法：

定义 3 假设 $A,B\subseteq D$，A 和 B 中的元素都是字典序的。$x\in A,y\in B,x,y$ 是 A,B 中从左边开始第一个不等的元素。如果存在 $x>y$，则 A 字典的序小于 B，记为：

$$A<B\Leftrightarrow C(\exists i\in B-A,A\cap\{1,2,\cdots,i-1\}=B\cap\{1,2,\cdots,i-1\})$$

例如,假设 $A,B\in D,D=\{1,2,3,4,5,6,7\}$ 和 $A=\{1,3,6,7\}$,$B=\{1,5,7\}$。A,B 中从左边开始第一个不相等元素是 3 和 5,则 $B<A(i=3,A\cap\{1,2\}=B\cap\{1,2\}=\{1\})$。

显然,定义 3 定义了一个 D 的幂集 $2^{|D|}$ 的全序,即 $A,B\subseteq D$ 且 $A\neq B$,那么总会有 $A<B$ 或者 $A>B$。为有效地产生形式概念,首先要产生所有的外延,下面先介绍 2 个表达式:$A<_iB$ 和 $A\oplus i$。

定义 4 给定 $A,B\subseteq D,A<_iB$ 被定义为 $i\in B-A$ 和 $A\cap\{1,2,\cdots,i-1\}=B\cap\{1,2,\cdots,i-1\}$,$A\oplus i$ 被定义为 $g(f((A\cap\{1,2,\cdots,i-1\})\cup\{i\}))$。

按照定义 4,下列推论成立:

(1)对于任意的 $i\in D,A<B\Leftrightarrow A<_iB$;

(2)如果 $A<_iB$ 且 $A<_jC$,那么 $i<j\Rightarrow C<_iB$;

(3)$i\notin A\Rightarrow A<A\oplus i$;

(4)$A<_iB$ 且 B 是一个外延$\Rightarrow A\oplus i\subseteq B$,那么 $A\oplus i\leqslant B$;

(5)$A<_iB$ 且 B 是一个外延$\Rightarrow A<_iA\oplus i$。

上述推论证明参考文献[11-12]。下面,本文将采用以上定理和推论证明算法中的定理。

定理 大于一个给定集合 $A\subset D$ 的最小概念外延(关于字典序)是 $A\oplus i$。其中,i 是满足 $A<_iA\oplus i$ 的 D 的最大元素。

证明:设 $A+$ 是(字典序)大于 A 的最小索引键。由 $A<A^+$ 知,必存在某个 $i\in U$ 使 $A<_iA^+$。由于 A^+ 是索引键,由推论(5)知 $A<_iA\oplus i$,再由推论(4)知 $A\oplus i\leqslant A^+$。然后由 $A\oplus i$ 的定义知 $A\oplus i$ 一定是索引键,这样由 $A<A\oplus i\leqslant A^+$ 以及 A^+ 是大于 A 的最小索引键,知必有 $A\oplus i=A^+$。下面再证 i 是最大的,设还有 $j\neq i$ 使得 $A<_jA\oplus j$,由于 $A\oplus j$ 也是大于 A 的索引键,因此,由 A^+ 的最小性知 $A^+<A\oplus j$,于是 $A\oplus i<A\oplus j$。注意到 $A<_iA\oplus i$ 和 $A<_jA\oplus j$,由理论(2)知,$j<i$。

按照以上定义,一个外延是最小外延,它在字典序上要比 A 大,算法从所有外延中最小外延 $g(f(\phi))$ 开始,发现离它最近的外延子集(大于 $g(f(\phi))$),且该子集是另一个外延,然后将其保存起来,并作为最小外延,重复这个发现过程,直到最大外延为 D 本身。故对于给定集合 $A\subset D$,其外延发现算法描述如下:

算法 1 Map 函数产生部分概念

输入 形式背景 C,且 $C=(T,R,D)$

输出 全部概念$\{ngid,KVs\}$,其中,$ngid$ 表示集群中节点,KVs 表示 $N(key,value)$ 集合

```
begin
Initialize:value ←g(f(ϕ)); /* 索引键 */
   KVs ←{D,ϕ};  /* ϕ 的索引键为 D */
i←|D| - 1;
repeat
if value < ivalue ⊕ i then
value←value ⊕ i; /* 把 CIK ⊕ i 作为一个索引键 */
key←SearchKey(value); /* 由索引键查询内涵 */
i←|D|;
endif
i←i - 1;
until value ⊕ i = =D; /* 重复 D 中所有的对象 */
return {nid,KVs};
end
```

算法 2 SearchKey 函数根据索引键找到对应的索引值

输入 $N(key,value)$ 中的 $value$,即形式背景 C 的外延

输出 $N(key,value)$ 中满足 $f(key)=value$ 的 key,即内涵

```
begin
key ←ϕ; /* 设最初属性集为空 */
k ← 0; /* 记录外延匹配的个数 */
foreach d ∈ value do
      k ← k + 1;
      temp ← ϕ;
      foreach p in T do /* T 表示存储文档背景的二元组 */
        if p.r = =d then /* 如果找到外延 */
      temp ← temp∩p.c ; /* temp 集合加入内涵 */
       endif
     endfor
      if k > 1 then
        key ← key∩temp ;
      else
    key ← temp;
      endif
  endfor
key←replace(key);/* 替换稀疏矩阵属性值为相应列号 */
return key;
end
```

算法 1 中 Map 函数求出每个概念的外延后,传递给算法 2 中 SearchKey 函数找出每个外延对应的内涵,为使返回值更便于在 MapReduce 平台上处理,key 表示词的 id 而不是单词,SearchKey 算法采用 replace 函数把词替换为相应的 id,以提高 Reduce 阶段的计算效率。概念在集群中被分为不同的组,每组都有一个 ID,每个组节点中用 $ngid$ 表示。在用 Map 函数产生了所有节点上形式概念后,计算的结果将传输到 Reduce 函数计算阶段,下面讨论 Reduce 函数。

4.2 Reduce 函数

Map 函数计算部分形式背景下的形式概念,在 Reduce 阶段需将若干个形式概念合并,形成较大形式概念集合。Map 阶段的返回值是$\{ngid,KVs\}$的形式,$ngid$ 是某个集群节点的 id 号,KVs 是概念($\{key,value\}$)的集合。Reduce 阶段即合并有相同 $ngid$ 的全部概念的集合。

考虑给定 2 个概念 $N_1(key_1, value_1)$ 和 $N_2(key_2, value_2)$，它们来自不同的 Map 节点，但有相同的 *ngid*。明显，将 N_1 和 N_2 合并后新产生的概念只有下面 2 种形式：$N_{merge(1,2)}(key_1 \cap key_2, value_1 \cup value_2)$ 或 $N_{merge(1,2)}(key_1 \cup key_2, value_1 \cap value_2)$。为减少整个集群的工作量，需确保一篇文档只在唯一一个节点上产生概念，即 $value_1 \cap value_2 = \phi$。因此，上述概念的合并只产生形如 $N_{merge(1,2)}(key_1 \cap key_2, value_1 \cup value_2)$ 的新概念，即 2 个概念 $N_1(key_1, value_1) \in KVs_1$ 和 $N_2(key_2, value_2) \in KVs_2$，可得出以下合并规则：

(1)如果 $key_1 = key_2$，新概念为 $N_{merge(1,2)}\{key_1, value_1 \cup value_2\}$；

(2)如果 $key_1 \neq key_2$ 且 $key_1 \cap key_2 \neq \phi$，新概念为 $N_{merge(1,2)}(key_1 \cap key_2, value_1 \cup value_2)$；

(3)如果 $key_1 \cap key_2 = \phi$，没有新概念产生。

根据以上合并规则，下面给出 Merge 合并算法和 Reduce 算法。

算法 3 Reduce 函数合并部分概念

输入 组 ID 中的某个结点和该组节点中的概念集合 *Listn*

输出 合并后新的集合 *NListn*

```
begin
NListn ← Listn[0]; /*新建一个集合存储全集概念*/
for i ←1 to |Listn| -1 do /*部分概念集合中每个概念*/
    NListn ← LMerge(NListn,Listn[i]); /*合并*/
endfor
return NListn;
end
```

算法 4 Merge 函数合并概念集合

输入 从每组集群不同节点抽取 2 个概念集合 *KVs*1 和 *KVs*2

输出 合并产生的新概念 *KVsnew*

```
begin
KVsnew←KVs1;/*将其中一个集合看成新概念集合*/
foreach N(key,value) ∈ KVs2 do /*对另一个概念集合*/
sign = 0;
foreach Nnew(keynew,valuenew) ∈ KVsnew do
if key = =keynew then  /*判断 key 是否相等*/
  KVsnew←KVsnew-Nnew(keynew,valuenew) ∪(keynew,
value∪valuenew);
    sign ← 1;
      else if keynew ≠ key and keynew ∩ key≠ϕ then
/*键不等也无交集*/
          KVsnew ← KVsnew ∪ (key ∩ keynew,
value∪ valuenew);
            else
              Continue;
        endif
endfor
    if sign = =0 then
      KVsnew ←KVsnew ∪ (key,value);
    endif
endfor
return NListn; /*返回合并后的集合*/
end
```

4.3 概念索引的建立

用上述 Map 函数和 Reduce 函数得到文档集上所有概念，下面将介绍如何建立概念索引。表 2 给出了概念索引的存储结构，每个概念分配一个 id 进行唯一标识，*key* 和 *value* 分别表示形式概念中词和文档的集合，*P* 和 *C* 分别代表给定概念的父节点集合和子节点集合，*Level* 代表当前概念的层级。特别地，概念 $\{T, \phi\}$ 的 *id* 为 0，$\{\phi, T\}$ 的 *id* 为 −1。

表 2 概念索引存储结构

字段	数据类型	描述
id	integer	索引概念的唯一标识
key	array	索引键(词)
value	array	索引值(文档)
P	array	父节点集合
C	array	子节点集合
Level	integer	索引概念的层次

算法 5 说明了如何为文档背景建立概念索引，为了更高地进行计算，算法按照概念中 *value* 的个数排序且在处理过程中，概念将按照 *Level* 进行排序，最后生成概念索引。概念索引将采用分布式存储。

算法 5 Establish 函数建立概念索引

输入 全集概念 *KVs*

输出 全集概念的存储结构 *FKV*

```
begin
SortByNumberOfValues(KVs);/*按照 value 数降序排列*/
Head ← {ϕ,D}; /*Head 是索引格的头*/
Head.level ← 0;
FKV ← FKV ∪ Head;
foreach N ∈ KVs do
    if N.C.size ==1 then/*判断是否只有一个子节点*/
     Head.C←N.id;
     N.P ← Head.id;
   else
SortByLevel(FKV); /*否则按照 level 降序排列*/
     Temp ←FKV;
foreach Nf ∈ FKV do
if Nf.C ⊂N.C then  /* Nf 是 N 的父索引节点*/
       N.P.add(Nf,C.id);
       Temp.removeAll(Nf.P);
endif
endfor
FKV←Temp;
  endif
  FKV← FKV∪N;
endfor
```

```
return FKV;
end
```

5　实验评估

5.1　实验设置

实验数据集是系统生成的文档集合,包含 50 000 篇文档 20 个关键字,记为 $5 \times 10^4 \times 20$,文档背景矩阵占用 2.28 MB。实验环境配置如下:共 12 台虚拟机安装在 6 台物理机上。物理机配置如下:Inter(R) Xeon(R) CPU E3-1230 V2 @ 3.30 GHz, 4 GB内存,1 TB 硬盘,Windows 7 Ultimate in 64-bit 操作系统。虚拟机操作系统是 Ubuntu 12.04,软件环境:Hadoop-1.0.4 和 HBase-0.90.4。本文实验所有代码都是用 Java 编写,执行时间取 5 次实验平均时间。

5.2　实验结果分析

本文从以下方面考察所提索引的有效性和性能:

实验 1　检测文档规模的大小对计算概念时间的影响。索引概念是分布式的索引,在集群中的单个节点上,为其分配合适数目的文档集,能够提高全局索引的构建速度,即在文档分配上要尽量做到负载均衡,而索引构建速度与概念计算时间成正比,因此,本实验探索节点文档规模大小与概念计算时间的关系。

由于在 Hadoop 架构中,所有文档信息都保存在 HDFS 上,因此执行 Map 时,可利用代码控制一个概念索引的规模,使得每个概念索引包含文档数目相同。图 2 展示了节点中不同数目文档集时计算概念花费的时间。横轴表示每读入一定数目的文档就调用一次 Map 函数,纵坐标表示索引概念计算的时间。可以发现,随着节点中文档数目增多,概念产生时间增长但是增长趋势减缓,且随着集群节点的增多,计算概念时间迅速减少。因此,在 MapReduce 平台,为每个节点分配合适数目的文档集将会优化构建概念索引的时间。

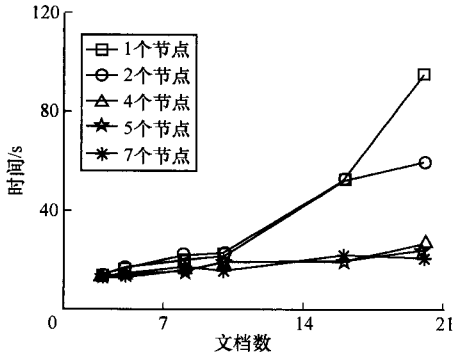

图 2　节点文档数目与概念计算时间的关系

实验 2　考察一定文档集下,子概念索引在集群中的数目与其建立时间的关系。随着文档集的增大,构建索引的时间将会越长,因此,可利用多线程技术处理多个子概念索引以降低构建时间,在产生概念索引之前,将文档集合分为不同的组,每组用一个 ID 标识,在 Map 阶段把此 ID 作为 *key* 输出。在 Reduce 阶段,根据 *key* 进行 shuffle,这样,可保证一个 Reduce 任务计算同一组文档集的概念构建概念索引。

因此,子概念索引的数目会影响 Reduce 阶段的性能。图 3 显示了在一定的文档集下,子概念索引的数目和索引构建时间之间的关系,可发现随着子概念索引数目的增多,产生概念索引所需时间不断减少,当子概念索引数目少于 20 时,所需时间较长。这是因为在 Reduce 阶段,当子概念索引数目越少,越多的索引将被合并,所需时间也较长。且集群中节点越多,计算性能越优秀,但当子概念索引数目(40)增多时,集群中节点数目多少对其性能影响也随之降低。

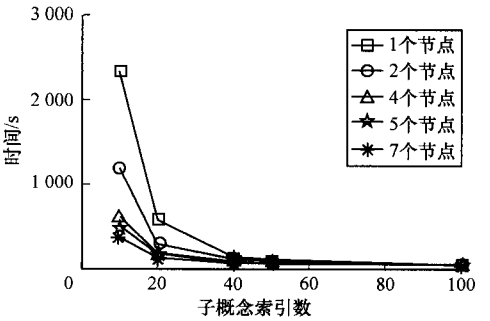

图 3　子概念索引数与索引构建时间的关系

以上 2 个实验说明了索引的构建时间随着节点文档数目增加而增长,随着子概念索引数目的增加而降低。因此,为了提升整体的索引构建性能,需要在上述 2 个影响因素之间进行折中。

实验 3　概念索引和 Lucene 索引[13]查询性能比较,检测在概念索引上查询特定词的性能。实验采用不同的词集分别在概念索引和 Lucene 上进行查询,测定响应时间,响应时间取 1 000 次随机词查询时间的平均响应时间。

图 4 展示了在概念索引和 Lucene 上查询 1 个词和 2 个词的平均响应时间。

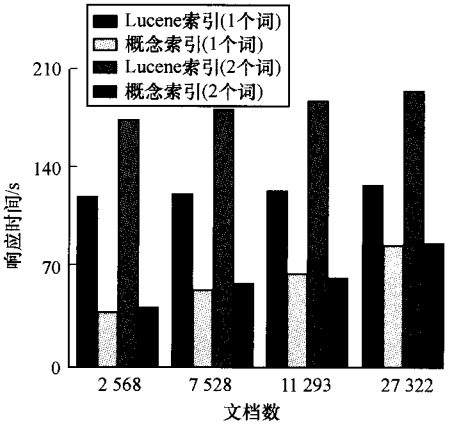

图 4　概念索引和 Lucene 索引查询性能比较

可以看出,尽管概念索引的查询需合并不同节点的结果,和 Lucene 相比,仍然可以保持较快的响应。这是因为,概念索引在 MapReduce 平台上可以进行多线程并行查询,且概念索引拥有父子关系的特点,可以很好地进行剪枝。

6 结束语

为大规模文档集构建一个高效的索引结构以提高信息检索速度是目前的研究热点。MapReduce 为构建分布式文档索引提供了较高的性能支持,不同于已存在的分布式索引结构,本文通过分析文档和词之间的关系,提出了大规模文本结构上的概念格索引。采用 FCA 表示文档背景,用“格”表达文档与文档之间、文档与关键字之间的关系,并给出了在 MapReduce 上的相关构建算法。实验结果表明基于概念格索引进行文档检索效率较高。下一步将研究针对概念格索引的维护,如增量索引,使其具有更灵活的应用。

参考文献

[1] Melink S, Raghavan S, Yang B, et al. Building a Distributed Full-text Index for the Web[J]. ACM Transactions on Information Systems, 2001, 19(3): 217-241.

[2] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[3] Witten I H, Moffat A, Bell T C. Managing Gigabytes: Compressing and Indexing Documents and Images[M]. [S. l.]: Morgan Kaufmann, 1999.

[4] Kumar C. Designing Role-based Access Control Using Formal Concept Analysis[J]. Security and Communication Networks, 2013, 6(3): 373-383.

[5] 陈 湘,吴 跃. 基于基集与概念格的关联规则挖掘算法[J]. 计算机工程,2010,36(19):34-36.

[6] Abusukhon A, Talib M, Oakes M P. An Investigation into Improving the Load Balance for Term-based Partitioning[M]// Kaschek R, Kop C, Steinberger C, et al. Information Systems and E-Business Technologies. Berlin, Germany: Springer-Verlag, 2008: 380-392.

[7] Anh V N, Moffat A. Inverted Index Compression Using Word-aligned Binary Codes[J]. Information Retrieval, 2005, 8(1): 151-166.

[8] Lin J, Dyer C. Data-intensive Text Processing with MapReduce[M]. [S. l.]: Morgan and Claypool Publishers, 2010.

[9] Krajca P, Vychodil V. Distributed Algorithm for Computing Formal Concepts Using Map-Reduce Framework[M]// Adams N M, Robardet C, Siebes A, et al. Intelligent Data Analysis. Berlin, Germany: Springer-Verlag, 2009: 334-344.

[10] Xu Biao, de Fréin R, Robson E, et al. Distributed Formal Concept Analysis Algorithms Based on an Iterative MapReduce Framework[M]//Domenach F, Lgnatov I D, Poelmans J. Formal Concept Analysis. Berlin, Germany: Springer-Verlag, 2012: 292-308.

[11] Ganter B, Wille R, Franzke C. Formal Concept Analysis: Mathematical Foundations[M]. Berlin, Germany: Springer-Verlag, 1997.

[12] Ganter B. Two Basic Algorithms in Concept Analysis[M]. Berlin, Germany: Springer-Verlag, 2010.

[13] Apache Software Foundation, Lucene[EB/OL]. [2012-10-12]. http://en.wikipedia.org/wiki/Lucene.

编辑 金胡考

(上接第 47 页)

[5] 陈 军,卢涵宇,姚丹丹. 基于处理时间的网络地图云服务调度算法[J]. 计算机测量与控制,2013,21(7):1987-1989.

[6] Nie G, She Q, Chen D. Evaluation Index System of Cloud Service and the Purchase Decision Making Process Based on AHP[J]. Advances in Fuzzy Systems, Applications and Theory, 2011, 36(7): 115-126.

[7] 刘建勋,唐明董,曹步清. 考虑 QoS 属性相关性的 Web 服务选择[J]. 小型微型计算机系统,2014,35(4):786-790.

[8] 陈子侠. 基于模糊组合权的模糊 TOPSIS 方法在供应商选择中的应用[J]. 上海管理科学,2010,32(6):58-61.

[9] Josang A, Ismail R, Boyd C. A Survey of Trust and Reputation Systems for Online Service Provision[J]. Decision Support Systems, 2007, 43(2): 618-644.

[10] 赵 卓. 基于 PROMETHEE 的云服务推荐一种多目标决策方法[J]. 计算机应用研究,2014,31(7):2027-2034.

[11] 程功勋,刘丽兰,林智奇,等. 面向用户偏好的智能云服务平台研究[J]. 中国机械工程,2012,23(11):1318-1323.

[12] 冯建周,孔令富. 基于模糊 QoS 和偏好权重的 Web 服务组合方法研究[J]. 小型微型计算机系统,2012,33(7):1516-1521.

[13] 代 钰,杨 雷,张 斌. 支持组合服务选取的 QoS 模型及优化求解[J]. 计算机学报,2006,29(7):1167-1178.

编辑 陆燕菲