# Hadoop 下的分布式搜索引擎①

胡　彧[1]　封　俊[2] (1.太原理工大学 测控技术研究所 山西 太原 030024；2. 太原理工大学 计算机与软件学院 山西 太原 030024)

**摘　要：** 分析了 Hadoop 系统结构，提出一种改进的 PageRank 算法，使用 Map/Reduce 模式设计系统模块。实验证明，使用 Hadoop 框架能够设计出具有高性能、高可靠性和易扩展性的分布式搜索引擎。

**关键词：** Hadoop; PageRank; Map/Reduce; 分布式搜索引擎

## Distributed Search Engine Using Hadoop

HU Yu[1], FENG Jun[2]

(1.The Institute of Measuring and Controlling Technology, Taiyuan University of Technology, Taiyuan 030024, China; 2.College of Computer and Software, Taiyuan University of Technology, Taiyuan 030024, China)

**Abstract:** Initially, the architecture of Hadoop is analyzed, and an improved PageRank algorithm is proposed. Then, we design system modules using Map/Reduce. The implementation presents that the distributed search engine using Hadoop is good in its performance, reliability and scalability.

**Keywords:** Hadoop; PageRank; Map/Reduce; distributed search engine

## 1　引言

互联网的高速发展使 Internet 上信息数量呈几何式增长。现有的集中式搜索引擎从如此海量的信息中快速检索出真正需要的信息正变得越来越困难，所以搜索引擎系统应具有分布式处理能力，能根据需要处理信息的增长，不断地扩展系统规模以增强系统处理信息的能力。因此，构建分布式搜索引擎就变得非常有意义了。本文首先分析了 Map/Reduce 编程模型运行原理及其优点，其次介绍了 Map/Reduce 模型的开源实现版本——Hadoop 分布式处理平台，在此基础上将搜索引擎的爬行器、索引器和查询器三个功能模块按照 Map/Reduce 模型进行设计，充分利用 Hadoop 的集群拓扑特性，实现了搜索引擎的分布式处理、高可靠性和易扩展性。同时分析了 PageRank 算法的优缺点，引入时间平衡因子对算法进行改进。

## 2　Map/Reduce编程模型与实现平台

### 2.1 Map/Reduce 简介

Map/Reduce[1]是 Google 实验室提出的一种新的分布式程序设计模型，用于在集群上对海量数据进行并行处理。Map/Reduce 的名字源于这个模型中的两项核心操作：Map 和 Reduce。

执行流程如图 1 所示。首先对输入的数据进行分割，将分割后的数据分配给 Map，而 Map 把分配到的数据(一般为一组<key, vaule>对)映射为另外的一组<key2, vaule2>型中间数据，其映射的规则由一个函数指定；Reduce 是对 Map 输出的<key2, vaule2>型中间数据进行归约并输出最终结果，这个归约的规则也是由一个函数指定。

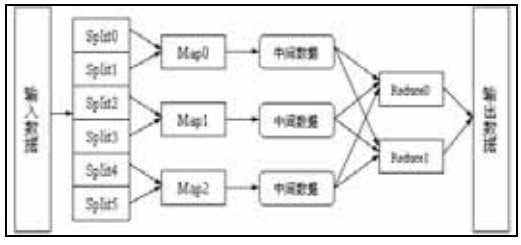


图 1　Map/Reduce 模型执行流程图

① 基金项目:山西省自然科学基金(2009011019-2)
收稿时间:2009-10-16;收到修改稿时间:2009-11-20

实际应用中这两项操作可以自由指定，正是这一点带给 Map/Reduce 模型巨大的灵活性，且效率很高，因此非常适合于分布式搜索引擎这种数据量大但数据类型简单的应用。

### 2.2 Hadoop 开源分布式处理平台

Hadoop 开源分布式处理平台是 Map/Reduce 模型的开源实现版本，由两大核心元素构成：最底部的 Hadoop Distributed File System(HDFS 分布式文件系统)，以及进行分布式计算的 Map/Reduce 引擎。

HDFS 是分布式计算的存储基石，它采用 Master/Slave 结构，由一个 NameNode 和多个 DataNode 组成。NameNode 协调用户对文件的访问，DataNode 负责数据存储。其原理是将一个完整的文件拆分为多个块(Block)，每个块被分别存储到数据节点的磁盘中。HDFS 提供了统一的命名空间，使得用户可以像访问单个文件系统一样访问 HDFS。

Map/Reduce 引擎也采用 Master/Slave 结构，由一个 JobTracker 和多个 TaskTracker 组成。JobTracker 负责作业调度，TaskTracker 执行计算任务，所有的 TaskTracker 都需要运行在 DataNode 上，这是因为 Hadoop 的 Map/Reduce 引擎遵循了“移动计算比移动数据更经济”的原则[2]：数据存储在哪一个节点上，就由此节点进行这部分数据的计算。这样可以减少数据在网络上的传输，降低对网络带宽的需求。

综合 Map/Reduce 和 HDFS 的 Hadoop 结构如图 2 所示。

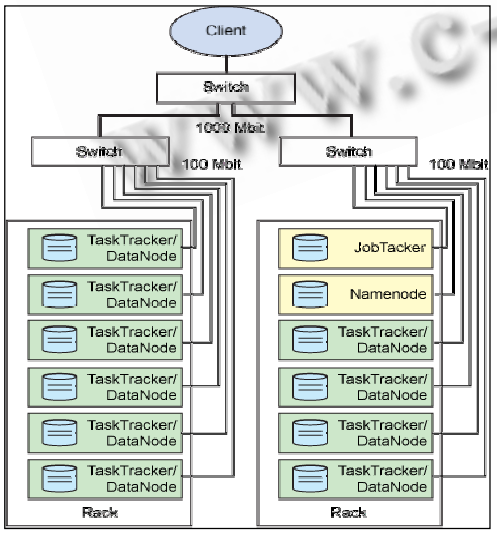


图 2 Hadoop 结构

在 Hadoop 分布式系统中，有一个 Master 节点，具有 NameNode 和 JobTracker 功能，也可以将 NameNode 和 JobTracker 分别布署到两台计算机中共同组成 Master 节点；此外，还会有多个 Slave 节点，每个 Slave 节点都具有 DataNode 的功能并负责 TaskTracker 的工作。

## 3 分布式搜索引擎的设计与实现

分布式搜索引擎由三个子系统构成：爬行子系统、索引子系统和查询子系统。在系统的设计过程中充分利用了 Map/Reduce 编程模型分布式处理的优点，将以上三个子系统设计成分布式系统。下面对这三个子系统进行详细分析。

### 3.1 分布式爬行子系统

爬行器的主要功能是对网页数据进行抓取，并进行分析提取链接，生成链接列表以供爬行器下一次爬行使用。爬行器循环执行这一流程从网络上获取网页数据。分布式爬行子系统正是由这些部署在 Slave 节点上的爬行器和调度器组成的。爬行过程中涉及两个数据库的操作：网页链接库和网页数据库，分别存放着网页链接列表和下载回来的网页数据。

爬行子系统设计的核心是任务调度，其所有爬行器统一由 JobTracker 负责调度。执行流程如下：

(1) JobTracker 首先通过“心跳”获得所有 TaskTracker 的信息，形成空闲 Slave 节点列表。访问网页链接库获得爬行链接列表的规模 u，此后 JobTracker 会根据空闲 Slave 节点的数量 s 与状态，以及网页链接实际存储的物理位置，将链接列表分割为 s 个爬行任务(job)，分发给 s 个 TaskTracker。

(2) TaskTracker 接到任务后启动 m 个 Map，每个 Map 又会启动 t 个下载线程，这样整个系统就拥有 s×m×t 个下载线程协同工作，因此整体性能会获得极大提高。下载线程与网站建立连接使用 Http 协议下载网页数据，当下载完成后 Map 将数据映射为“<网页链接，网页数据>”这种类型的<key，vaule>对，作为中间数据输出。

(3) TaskTracker 监测到 Map 的输出时会启动 n(n≤m)个 Reduce。Reduce 汇总中间数据后，根据网页链接(key)从中间数据中取出网页数据(value)，对网页数据进行分析，提取出对其他网页的引用链接，同时记录这些链接，用来构建网页引用关系图。最后

把网页数据保存到分布式文件系统，将提取出的链接添加到网页链接列表，供下一轮爬行使用。

经过以上步骤就完成了一轮完整的爬行任务，子系统循环执行此爬行任务，直到满足终止条件。

### 3.2 分布式索引子系统

分布式索引子系统负责计算网页的 PageRank 值，倒排文档的构建，以及索引文件的分布式存储，它是整个搜索引擎的核心。在子系统的设计过程中，经过对网页发布规率和现有索引系统的分析，提出一种采用时间平衡因子的 PageRank 算法，应用 Map/Reduce 实现索引(倒排文档)模块，应用 HDFS 系统构建了分布式文件系统模块。

#### 3.2.1 时间平衡 (Time Balanced) PageRank 算法

PageRank 算法[3]的基本思想是：一个网页被多次引用，那么这个网页可能很重要；一个网页虽然没有被多次引用，但是被重要的网页引用，那么这个网页也可能是很重要的；一个网页的重要性被平均的传递到它所引用的网页。PageRank 算法的出现提高了搜索结果的相关性和质量，但是算法也忽略了这样一个事实，那就是新发布的网页即使很重要，但被引用的次数却较少。这就是本算法引入“时间平衡因子”——Time Balancer 的依据，其基本思想是：如果一个网页是最近发布且被多次引用，那么它可能比很早以前发布拥有同样引用数量的网页更重要，这个页面理应具有更高 PageRank 的得分。

PageRank 计算公式如下：

$$PR(p_i)=\frac{d}{N}+(1-d)\sum_{p_j}\frac{PR(p_j)}{Out(p_j)} \tag{1}$$

其中 $p_i$ 是被研究的网页，$PR(p_i)$ 是网页 $p_i$ 的 PageRank 值，$p_j$ 是引用 $p_i$ 的网页，$Out(p_j)$ 是 $p_j$ 链出网页的数量，而 N 是所有网页的数量，d 是阻尼系数，一般取值 0.15。在实际应用中采用一种迭代的方法计算网页的级别，也就是先给每个网页一个初始值(默认为 1)，然后利用上面的公式，循环进行有限次(100 次)运算得到近似的网页级别。

改进的 Time Balanced PageRank 算法在标准 PageRank 算法的基础上增加了时间平衡因子 Time Balancer 计算公式如下：

$$TBPR(p_i)=\left(\lambda\times e^{-\left(\frac{CD-PD(p_i)}{12}\right)}+1\right)\times PR(p_i) \tag{2}$$

其中 TB 代表时间平衡因子；$\lambda$ (0< $\lambda$ <1)是控制参数；CD 是开始计算 PageRank 值时的系统时间(单位为月)，是系统时间与标准基准时间(称为“历元(epoch)”，即格林威治时间 1970 年 1 月 1 日 00:00:00)的差值；PD(pi)是网页发布时的时间，单位为月，定义为发布时间与标准基准时间的差值。

当 PD(p1)<PD(p2)时(即网页 p1 的发布时间早于网页 p2)，则 TB(p1)>TB(p2)，且

$$\lim_{(CD-PD)\to\infty}\left\{\lambda\times e^{-\left(\frac{CD-PD}{12}\right)}+1\right\}=1 \tag{3}$$

可以看出，发布时间越早的网页时间平衡因子越小，且趋于 1，既随着发布时间的延长，平衡因子对网页 PageRank 值的贡献在逐渐减小。实际应用中根据计算的网页类型可以改变的值。例如，在计算时效性比较强的新闻类网页的 PageRank 值时可以取 0.95，使得较新网页的 PageRank 值较高；计算时效性比较弱的历史类网页的 PageRank 值时可以取 0.05，使得较早发布的网页不会被埋没。

在迭代计算 PageRank 值过程中，时间平衡因子并不参与计算，只有在迭代计算完成后，才利用时间平衡因子对结果进行修正，得到最终的 TB-PageRank 值。

#### 3.2.2 索引模块

索引的主要任务是生成倒排文档，之前由爬行子系统生成的时正排文档，数据形式为<DocumentID, Document>，而与此相对的是形如<Word, List(DocumentID)>的倒排文档。其实，正排文档与倒排文档不光数据形式不一样，使用方式也完全不一样。正排文档是通过文档 ID 得到文档的所有内容，而倒排文档是通过关键词来查找所有相关的文档 ID。

模块基于 Hadoop 的 Map/Reduce 引擎进行设计，综合使用了中科院的 ICTCLAS 中文分词系统和 Lucene 索引系统[4]。执行流程如下：

(1)JobTracker 将网页数据库分成 m 块，形成 m 个子任务，通过与 TaskTracker 之间的 RPC 通信，将这些任务分配给 m 个空闲的 TaskTracker。

(2)每个 TaskTracker 启动 2 个 Map，Map 对输入的每一个<网页链接,网页数据>数据对进行处理，执行 ICTCLAS 分词，网页去噪(去除没有实际意义的词语，如：中文的“是、的”和英文的“is，a”等)，

记录关键词所属的文档号和所在文档的位置等信息，生成<key,value>型(<关键词,文档号和位置信息>)中间数据并输出给 Reduce。

(3)每个 TaskTracker 启动 1 个 Reduce，它汇总中间数据，根据 Key(关键词)进行排序，合并相同 Key 的 value(文档号和位置信息)，生成<key, List(value)>数据，然后就可以利用 Lucene 将<key,List(value)>数据构建成索引(倒排文档)了。每个 Slave 完成的只是最终索引的一部分，都暂存在本地，经过合并后再保存到分布式文件系统中。

3.2.3 分布式文件系统模块

分布式文件系统负责 Lucene 索引文档的存储与读取。本文在参考 Google 的 GFS 分布式文件系统[5]的基础上应用 HDFS 实现了一种高可靠性的分布式文件系统。

文件系统使用副本存储策略来实现可靠性。系统的复制因子为 3，意味着每个块都会有 3 个副本，分别位于 3 个 DataNode 上，其中一个位于不同机架(Rack)的 DataNode 上。

如图 3 所示，所有块的元数据都被注册在 NameNode。当一个 DataNode 出现故障后，其保存的块仍然可以通过 NameNode 上注册的冗余块进行读取。NameNode 周期性的收到来自集群内 DataNode 的心跳报告，能收到心跳证明 DataNode 工作是正常的，如果 NameNode 没有收到一个 DataNode 的心跳报告，则说明此 DataNode 已经出现故障，这时 NameNode 把应由此 DataNode 保存的块的副本复制后存储到其他 DataNode 上，时刻保持系统中每个块都会有 3 个副本，以此来保证系统的高可靠性。

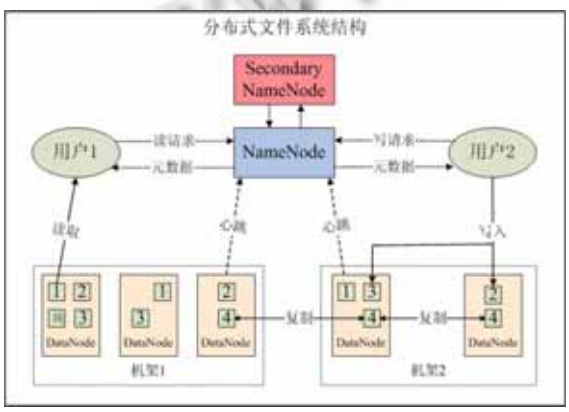


图 3 分布式文件系统结构

此分布式文件系统数据的读取和存储机制与一般文件系统有一些区别。当用户读取文件时，首先必须向 NameNode 提交读取请求，NameNode 查询元数据表后将文件的元数据(此文件分为几个块，每个块属于哪个 DataNode 等)返回给用户。接下来用户直接访问相关 DataNode 获得所需块，得到完整的文件。而当用户保存文件时，同样首先向 NameNode 提交保存请求，NameNode 在文件命名空间中写入文件名，根据文件大小将文件分割为许多 64M 的片段，并查询元数据表为文件分配空闲块，最后将相关元数据返回给用户，然后用户直接与相关 DataNode 建立连接，将文件写入到块中。

### 3.3 分布式查询子系统

分布式查询子系统负责响应用户查询请求并向用户返回结果。查询模块与上文分析的爬行子系统和索引子系统类似，也采用了 Map/Reduce 模型设计。

子系统采用 Tomcat 作为 Web 服务器，使用 Jsp/Servlet 技术与用户交互。用户在网页中输入的关键词，Web 服务器获得数据后提交给查询模块，提取出关键词，生成查询任务，通过 JobTacker 把查询任务分别发送到 TaskTracker，Map 根据关键词检索分布式文件系统得到索引数据，Reduce 汇总索引数据并根据 PageRank 得分进行排序后输出到 Web 服务器，Web 服务器对数据缓冲、分页，最后把查询结果以网页形式呈现给用户。

## 4 实验结果与分析

实验系统由 4 台 PC(CPU:酷睿 1.86G;内存:1G;硬盘:160G)构成，由 100M 网互联。其中一台作为 Master 节点，启动 NameNode 和 JobTracker 进程；其余三台都启动 DataNode 和 TaskTracker 进程，作为 Slave 节点。在实验过程中分别启动 1、2 和 3 台 Slave 节点，构成 2、3、4 节点的分布式环境。在性能测试阶段选取了运行时间长误差相对较小的爬行和索引的总时间，运行三次取平均值。

从图 4 可以看出，在网页量较小时，2 节点比 4 节点性能更高。因为在 2 节点情况下，JobTracker 无需做任务分割运算，任务集中在单 TaskTracker，无需与其他 TaskTracker 传递数据，且与 JobTracker 通信量也少。而随着网页量的增加，4 节点系统处理效能才体现出来，3 节点系统性能基本处于两者之间。

同时也发现 JobTracker 没有好的任务动态分割算法，在一些节点的 Map 结束时，另一些节点还未完成，导致 Reduce 处于等待状态，拖慢了整个系统性能。

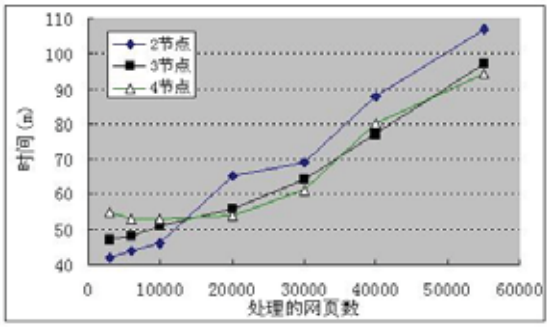


图 4　性能测试结果

在可靠性测试中，分别在 4 节点和任一 Slave 节点故障状态下进行测试。对相同关键词进行查询两种状态下，查询结果完全相同，任一 Slave 节点故障不会影响系统查询结果的完整性，证明系统稳定可靠。

在扩展性测试中，只需将新 Slave 节点地址添加到 Master 的 slaves 列表中，格式化 NameNode 并重启群集，新节点即可加入集群，系统规模很容易就可以得到扩展。

## 5　结语

本文探讨了分布式搜索引擎设计过程中的各种技术，通过实现一个基于 Hadoop 的分布式搜索引擎系统，证明 Map/Reduce 模型应用于搜索引擎可以提高系统性能、可靠性和扩展性。下一步通过改进 JobTracker 任务动态分割算法，充分利用节点计算能力，可以构建出更完善的分布式搜索引擎。

## 参考文献

1 Dean J, Ghemawat S. Map/Reduce:Simplied Data Proc. on Large Clusters. OSD I 2004, San Francisco, 200 4,137－1501.

2 曹羽中.用 Hadoop 进行分布式并行编程.[2008-05-22].http://www.ibm.com/developerworks/cn/open-source/os-cn-hadoop1/index.html

3 Sergey Brin, Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks and ISDN System, 1998,30(1-7):107－17.

4 高文举,李晓伟,孙春燕,李哲.基于全文检索 Apache Lucene 引擎的原理与流程研究.长春工业大学学报, 2008, 29(4):424－427.

5 Ghemawat S, Gobioff H, Shun-Tak Leung. The Google File System.19th ACM Symposium on Operating Systems Principles. Lake George, NY, October, 2003.