Software Engineering and Applications 软件工程与应用, 2013, 2, 6-14
http://dx.doi.org/10.12677/sea.2013.21002 Published Online February 2013 (http://www.hanspub.org/journal/sea.html)

Building an Information Retrieval System: Global Indexing or Local Indexing?*

Haitao Wang[1], Yanqiong Zhao[2], Jiaxin Han[3], Pang Yue[1#]

[1]College of Computer Science & Software Engineering, Shenzhen University, Shenzhen
[2]China Mobile Limited (Anhui), Hefei
[3]Oracle Corporation, Redwood City, USA
Email: htwang@szu.edu.cn, zhaoyanqiong@ahmobile.com, jiaxinjasonhan@gmail.com, #ymyp1987@163.com

Received: Nov. 19th, 2012; revised: Nov. 29th, 2012; accepted: Dec. 11th, 2012

Abstract: Nowadays, there is more and more data generated in production and life. Information retrieval system (IRS, such as Baidu and Google) is an indispensable tool to search usable information from magnanimity information. For an IRS, especially based on large-scale data set, indexing is necessary. The index is good or bad directly determines the success or failure of the IRS. In the past decades, the research of indexing of IRS has been intensive. The research focus is comparison and discussion of global indexing, local indexing, hybrid indexing, etc. In this paper, these indexing are introduced, their advantages and disadvantages are discussed, and achievements of them are reviewed. Practical application system will be analyzed. Finally, our views and solutions will be given.

Keywords: Information Retrieval; Global Indexing; Local Indexing; Hybrid Indexing; Mass Data; Distributed System

构建信息检索系统：全局索引还是局部索引?*

王海涛[1]，赵艳琼[2]，韩家鑫[3]，岳　磅[1#]

[1]深圳大学计算机与软件学院，深圳
[2]安徽移动网络部，合肥
[3]甲骨文公司，红木城，美国
Email: htwang@szu.edu.cn, zhaoyanqiong@ahmobile.com, jiaxinjasonhan@gmail.com, #ymyp1987@163.com

收稿日期：2012 年 11 月 19 日；修回日期：2012 年 11 月 29 日；录用日期：2012 年 12 月 11 日

摘　要：当今社会在生产与生活中产生的数据越来越多，要在海量的数据中搜索有用的信息，信息检索系统(IRS：Information Retrieval System，比如百度、谷歌等)是必不可少的工具。一个信息检索系统，特别是基于大规模数据集的信息检索系统，只有建立索引才能满足用户的检索需求，索引的好坏直接决定了信息检索系统的成败。数十年以来，对于信息检索系统中索引如何构建的研究一直没有中断，研究主要集中在对全局索引(Global Indexing)与局部索引(Local Indexing)及其混合类型(Hybrid Indexing)等结构的比较与探讨。本文详细介绍了几种索引的架构及其优缺点，回顾了相关的研究成果，分析了实际应用系统。最后，给出我们的观点与解决方案。

关键词：信息检索；全局索引；局部索引；混合索引；大数据；分布式系统

1. 引言

当需要从大数据集中查找所需要的很小一部分数据时，全盘扫描整个大数据集显然不是一个好方法，这时就需要用到针对该大数据集而建立的索引，以便快速查找。例如，在图书馆里找一本书，如果图书馆一共只有几十本书，一本一本去找可以找到；但是当图书管里有几十万本书时，用这种方法就无法完

*资助信息：国家自然科学基金面上项目，编号 61170076；2010 年深圳市基础研究项目，编号 JC201005280408A。
#通讯作者。

成任务了。如果这些书被整理分类，按照一定的顺序摆放，那么找起来就会容易很多。对书籍的整理摆放，其实就是一个建立索引的过程，按照书籍摆放的顺序去找书，就是一个使用索引进行数据查找的过程。

随着数据集规模的不断增大，索引的规模也越来越大。大多数情况下，针对原始数据集建立的索引大小与原始数据集在同样的数量级，有些情况下甚至可能膨胀翻倍。当前，现实中很多应用的数据规模已经达到了 TB 级，甚至 PB 级，单机或低端服务器已经无法满足数据存储以和处理的需求，而大型商业服务器价格昂贵且不易扩展；因而基于 Shared-Nothing[1] 架构的分布式集群系统就成为了解决问题的首选方案。在分布式集群当中，如何高效地建立并组织索引，就是本文所要探讨的核心问题。

分布式索引的架构选择是拥有大数据的企事业单位都会面临的问题。很多信息系统都拥有海量数据，例如百度数以十亿计的网页，淘宝与京东等丰富的商品信息及交易信息，银行的大量账户信息，公安部门的案件信息等等。如何更好的构建并组织索引，关系到企业的核心竞争力，直接影响事业单位的办事效率。

目前对于分布式索引架构问题的研究主要集中在三种架构上：全局索引架构(Global Indexing)与局部索引架构(Local Indexing)，同时还有混合架构(Hybrid Indexing)。Tomasic[2]很早就进行了分布式索引架构问题的研究；Abusukhon[3]、Cambazoglu[4]通过实验与分析对三种分布式索引架构进行了比较；Zhang[5]对全局索引架构在文档信息检索系统中的检索效率进行了研究；Bhagwat[6]将相似性搜索的方法用于索引的划分；Abusukhon[7]研究了全局索引架构中的负载均衡问题；Tang[8]对混合型索引进行了优化，并实际构建了一个分布式索引系统 eSearch。

较为普遍的观点是全局索引架构有较好的吞吐率，而局部索引架构的并行化程度较高，Hybrid 结合了两者的优点，但付出了极高的数据冗余代价。总结分析前人的科研成果与实际中使用的索引架构，可以得出：混合索引架构由于较高的成本，一般在实际的应用中不会被使用；全局索引架构检索时的开销较低，局部索引架构有较好的响应时间；在面对大规模的数据时，全局索引架构存在无法克服的缺陷，因此系统总体的架构采用局部索引架构为宜。

从数据的类型上，我们可以把索引分为三类：一维索引、多维索引[9,10]以及度量空间索引[11,12]。我们在本文中所讨论的索引架构问题都是以一位索引为基础的，一维索引也被称为第一代索引技术，相应的多维索引就是第二代索引技术，但两种索引类型面临的架构问题是相同的，因此本文对索引架构问题的讨论也适用于多维索引。度量空间索引作为第三代索引技术更具用一般性，一维索引与多维索引可以看作是度量空间索引的特例，目前其在生物信息学等领域已经有了广泛的应用[13,14]。支撑点空间模型建立了度量空间到多维空间的联系[15,16]，基于该技术我们在本文中对索引架构问题的讨论也可以推广应用到度量空间中去。当然，具体的细节可能会存在不同，这也是我们下一步计划研究的问题。

本文首先概括了索引问题的应用领域及相关的科研成果与应用实例，接着简单的阐述本文对分布式索引架构的观点。第二部分详细讲述了分布式索引技术的具体应用，并举了两个具体实例进行相应说明。第三部分深入具体索引架构的细节，说明它们的构建与工作原理。除了几种常见的索引架构，结合两个实际应用实例进行更具体的说明。第四部分对各个索引架构的优缺点进行了具体分析，结合现有的软硬件基础条件与实际的应用需求，给出了相对合理的解决方案。最后，文章对分布式索引系统的科研与应用进行了总结与展望。

2. 索引的应用

很多信息系统都有建立使用分布式索引的需求，其中互联网企业在这方面的需求较为典型。本节通过两个实例来详解分布式索引的实际应用。第一个实例介绍 Google 的 Web 搜索服务的整体架构及其对分布式索引；第二个实例是作者参与的一个具体项目，数据集是数以十亿计的上网记录，要求能够对其中的 URL 字段中任意字符序列进行快速检索，由于索引数据很大，故采用分布式索引架构。

2.1. Google 的 Web 搜索服务(GWS, Google Web Service)

在介绍 Google 的 Web 搜索服务架构之前，首先

了解一个基本概念：倒排索引。所谓倒排索引，即在一个文档集之中，每一个文档有一个唯一的编号(doc_id)，然后每个文档里面的文本内容被切分为一个一个独立的关键字(Key)，针对关键字 Key 建立索引，每一个 Key 对应的是一个 doc_id 串，形成一个 Key-Value 结构。其格式如下：

Key value

Key1 doc_id3, doc_id100, doc_id103…doc_idL;

Key2 doc_id12, doc_id130, doc_id183…doc_idM;

Key3 doc_id125, doc_id320, doc_id1003…doc_idN;

……

这些 Key-Value 结构的记录会按照 Key 的字典序(也可能是其他的顺序)组织成索引(如 B+树索引)。上面只是一个简化的结构，实际应用中 Value 里还会包含其他辅助信息，如 Key 在相应的文档里面的位置，出现的频率，字体大小、颜色等等。

Google 先利用爬虫工具在互联网中抓取足够的网页，每个网页即是一个文档，然后针对所获得的文档集建立倒排索引。因为所建立的倒排索引很大，故采用分布式架构对索引进行组织，如图 1 即为 Google 的 Web 搜索服务架构图[17]。

当用户使用 Google 进行一次查询，由于 Google 在全球分布有多个物理集群，DNS 负载平衡系统通过计算用户与每一个物理集群地理上的距离来选择一个合适的物理集群。然后用户浏览器向选定集群发送一个 http 请求，这样，对于该集群来说，所有的处理都变成了本地的。收到一个请求之后，Google Web Server 负责协调这个查询的执行，并将结果格式化为 html 语言返回给用户。

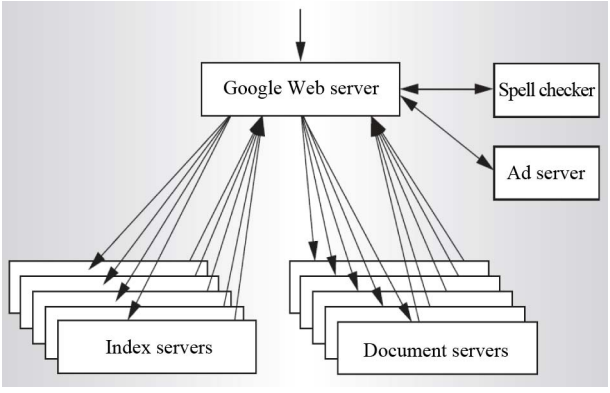

Figure 1. Google query service architecture
图 1. Google 的 Web 搜索服务架构图

查询执行由两个主要阶段组成：第一个阶段，索引服务器(Index Servers)查阅倒排索引，每个检索关键字得到一个相应的文档列表，通过对所有检索关键字得到的文档列表求交集，得到满足查询条件的结果集；然后索引服务器为每个文档计算出一个相关性的分值，这个分值决定了该文档在输出结果中的排序。搜索的过程是整个系统的关键之所在，因为需要处理海量数据，就需要用到分布式索引架构，详细的情况在 3.4 节介绍。第一阶段的查询最终输出一个排过序的文档编号列表。第二阶段，针对所获取的文档编号列表，使用文档服务器(Document Servers)计算出所有文档的标题和 url 以及面向查询内容的文档摘要，最后将结果格式化为 html 语言返回给用户。

通过图 1 以及上文的分析，可知索引服务在 Google 的 Web 搜索服务系统中处于核心地位，是解决问题的关键，其索引服务的架构正是分布式索引架构。

2.2. URL 序列检索系统

该系统针对的数据集是数以十亿计的上网记录，其记录格式大致如下：

Id datetime out_ip url mac

具体的记录有约二十个字段，比较复杂，这里做了相应的简化。要求能够对其中的 url 字段中任意字符序列进行快速检索。例如，用户设置检索条件为字符串“feng.com/mil/4”，数据集中所有记录的 url 字段中只要含有字符串序列“feng.com/mil/4”，那么该记录就应返回给用户，对于 url：

news.ifeng.com/mil/4/detail_2012_11/01/18741908_0.shtml

其所在的记录的相关信息就应该包含在结果集之中。

该系统用于分析特定用户的上网行为，如学校、公司、科研单位、安全部门等。有些情况下用户可能只知道一个 url 的片段(一个子序列)，就只能利用该片段来进行检索，从而找的相关的上网记录。

该系统的难点在于构建索引，而索引构建的关键在于 url 的切分。然而不论 url 如何切分，所建立索引的大小都远远大于原始数据。数十亿的上网记录接近 1 TB 大小，所建立的索引的大小约为原始数据的十倍，即 10 TB 左右。数十亿的上网记录也只是一所大

学三四个月所产生的数据量。为了更好的可扩展性，必须考虑到存储更长时间的数据，满足更大的集体的使用需求。所以索引的数据量会更大，因此采用分布式的索引架构。

在介绍系统的搜索服务之前，先说明所采用的 url 切分策略。对于一个 url，采用定长、逐个字符后退的切分方法，如对于 url：

news.ifeng.com/mil/4/detail_2012_11/01/18741908_0.shtml

使用切分长度为十五，得到结果如下：

news.ifeng.com/
ews.ifeng.com/m
ws.ifeng.com/mi
……
8741908_0.shtml
741908_0.shtml
41908_0.shtml
1908_0.shtml
……
ml

使用切分出来的 url 片段作为 Key 建立索引，使用包含相应的 url 片段的 id 组成 id 串作为 Value，形成 Key-Value 结构，如下：

Key Value
ws.ifeng.com/mi id3,id803,id6320……idN;

其中作为 Value 的 id 串“id3,id803,id6320……idN”中的每一个 id 所指向的上网记录中的 url 字段，都包含字符串序列“ws.ifeng.com/mi”。

该系统的搜索服务的架构与 Google 的 Web 搜索服务系统的架构类似，主要由一个索引服务器和一个记录服务器构成，下面举例说明其检索过程。用户提供所要检索的 url 片段：

news.ifeng.com/mil/4/detail

片段长度大于十五，则以十五为长度对片段进行切分，得到查询关键字：

news.ifeng.com/
mil/4/detail

使用索引服务器对两个关键字分别进行检索，得到两个 id 串，求它们的交集，得到目标 id 串，将目标 id 串提交给记录服务器，记录服务器根据每一个 id 得到其相应的记录，同时检测并剔除不满足条件的记录(通常没有)，最后形成记录列表，返回给用户。

3. 分布式索引架构

所谓索引，即是将文档、记录、书籍等对象中具有检索意义的事项(可以是人名、地名、词语、概念、或其他等等)按照一定方式有序地编排起来以供检索时使用的工具。举一个简单的例子，将词语按照字典序排列起来，这样在查找所需要的词语时就会很方便。

基于前文的分析，对于大数据集，所建立的索引通常也很大，单个机器无法存储处理，因此需要使用分布式索引架构。本节对主要的分布式索引架构(全局索引架构、局部索引架构以及混合索引架构)进行详细介绍，并以 Google 的 Web 搜索系统与 url 字符序列检索系统所使用的分布式索引系统架构为例，深入剖析实际应用中分布式索引系统架构的解决方案。

3.1. 局部索引(Local Indexing)架构

除了一些众所周知的英文缩写，如 IP、CPU、FDA，所有的英文缩写在文中第一次出现时都应该给出其全称。文章标题中尽量避免使用生僻的英文缩写。

局部索引架构是将大数据集划分成多个数据子集(数据集的划分方法没有严格的要求)，针对每个数据子集单独建立索引(我们称为一个索引块)。一般而言，每个索引块会单独存储(存储在一台单独的机器上或一个规模更小的分布式系统中)。以对文档集建立倒排索引为例，假设有六篇文档：d1(a，b，c)、d2(b，d)、d3(b，c，d)、d4(c，d)、d5(a，c)、d6(b，c)，其中的 a、b、c、d 为在文档中所出现的关键字，在一个由三台机器(X、Y、Z)组成的集群上面建立分布式局部索引系统。如图 2 所示，该索引系统的架构图为：

整个文档集被分成三个文档子集，分别存放于不同的机器之上(X(d1，d2)，Y(d3，d4)，Z(d5，d6))，然后在每台机器上分别建立了索引。

用户在使用该系统进行检索时，检索条件被提交给 Client Server(在实际的应用之中，Client Server 可能有多个，以达到更大的吞吐量)，Client Server 简单地将检索条件分发个每一个索引节点(X,Y,Z)，每个节点

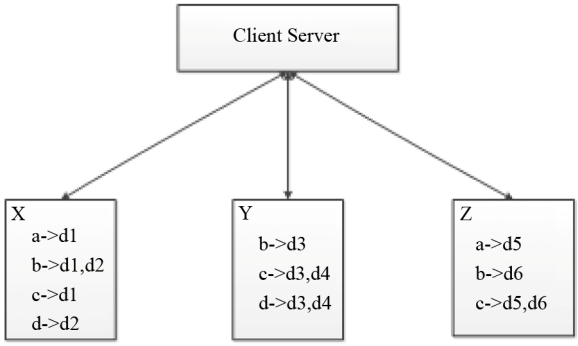

Figure 2. Local indexing architecture
图 2. 局部索引架构示意图

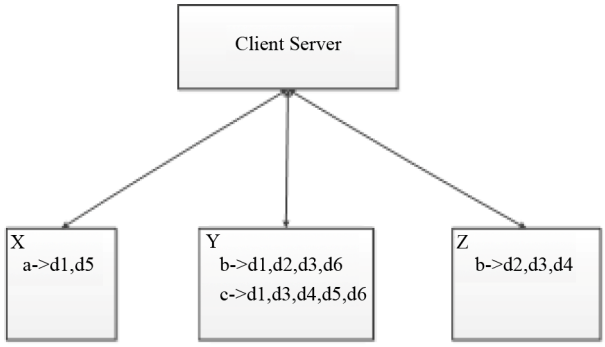

Figure 3. Global indexing architecture
图 3. 全局索引架构示意图

独立执行检索，然后将结果分别返回给 Client Server，Client Server 进行归并整理即得到用户所需的所有文档的文档编号列表。

例如，用户的检索条件为 Q(a,c)，即检索同时包含关键字 a 与 c 的文档，Q(a,c)被同时发送到 X、Y、Z 上面，X 上包含关键字 a 的为 d1，包含关键字 c 的为 d1，所以文档 d1 是满足检索条件的；Y 上没有包含关键字 a 的文档；Z 中包含关键字 a 的有 d5，包含关键字 c 的为(d5，d6)，所以满足条件的文档只有 d5。X、Y、Z 上面检索式并发执行的，Client Server 得到所有检索节点的返回之后，整理得到最终的满足条件的文档编号列表(d1，d5)。

3.2. 全局索引(Global Indexing)架构

全局索引架构在建立索引时的操作对象是整个数据集，所建立的索引是对完整数据集的索引。由于最终索引规模很大，故将所形成的完整索引按照一定的规则(如按照索引关键字的字典序等)切分成多个小索引片段。与索引块类似，每个索引片段会单独存储。同样使用 3.1 中的实例，整个数据集所建立的索引如下：

a->d1,d5
b->d1,d2,d3,d6
c->d1,d3,d4,d5,d6
d->d2,d3,d4

对索引进行切分，将不同的索引片段分配到不同的机器上，得到如图 3 的所示的全局索引架构示意图。整个索引以字典序分为三个索引片段，关键字 a 存储在机器 X 上面，关键字 b、c 被存储在机器 Y 上，关键字 d 在机器 Z 上。

Client Server 完成索引片段与机器的对应工作。对给定的关键字，Client Server 判断该关键字在哪一个索引片段及其相应索引节点，该功能可以通过一个对照表或者一个散列函数实现，对于不同的索引切分方法实现也不同。用户的检索条件被提交给 Client Server，Client Server 抽取检索条件中的关键字，然后将不同的关键字发到对应的索引节点，索引节点返回对应该关键字的文档编号列表给 Client Server，Client Server 将所得到的对应不同关键字的不同文档编号列表求交集，即得到用户所需求的最终的文档编号列表。

同样以 Q(a,c)为例，Client Server 抽取关键字得到 a 与 c，即将 a 发到其对应的索引节点 X 上，c 发到 Y 上，X 返回包含关键字 a 的文档编号列表(d1，d5)给 Client Server，Y 返回包含关键字 c 的文档编号列表(d1，d3，d4，d5，d6)，求返回的文档编号列表的交集，得到最终满足检索条件的文档编号列表(d1，d5)。

3.3. 混合索引(Hybrid Indexing)架构

混合索引先按照全局索引架构的方式建立索引并对索引进行切分，同样每个索引关键字会指向一个检索对象编号列表(例如以文档编号列表，每个文档编号对应一个文档，文档中会包含该索引关键字)，不同之处在于检索对象编号列表中每个编号所对应的文档都会冗余的存储在该索引关键字所在的节点上(实际当中存储的并不是检索对象本身，而是检索对象所包含的所有的关键之及相关的信息)，这样就造成了极大的数据冗余，以文档为例，若一个文档里包含一千个不同检索关键字，则在极端的情况下该文档的所有关键字及相关信息可能被冗余的存储了一千次，这里

的极端情况是指每个关键字都在不同的节点上。这种索引方式结合了全局索引与局部索引的优点，在检索时只需要使用包含相应检索关键字的索引节点，不像局部索引需要搜索所有的索引节点；返回的结果集也较小，而全局索引所返回的文档列表往往包含很多不满足条件的文档，以致需要在中间节点求交集。但是，混合索引付出了数据冗余的代价，而且其冗余度非常高，通常无法被接受。

同样使用 3.1 节中对文档集合建立倒排索引的实例，如图 4 所示为所建立的混合索引架构示意图：

可以看出，由于每个索引节点保存有相关文档详细的关键字信息，这样对于每一个检索只需要在一个相关的索引节点就可以完成。

考虑 Q(a,c)，与全局索引类似，Client Server 首先抽取关键字得到 a 与 c，接下来 Client Server 将检索条件发到关键字 a 对应的索引节点 X 或 c 对应的 Y 都可以完成检索。假设检索条件被发到索引节点 Y 上，接下来所有的处理都将变成在 Y 上的本地处理。其先，根据关键字 c 得到文档编号列表(d1,d3,d4,d5,d6)，由于 Y 存有相应文档详细的关键字信息，即可在文档编号列表(d1,d3,d4,d5,d6)所列出的文档中直接找出包含关键字 a 的所有文档，得到最终满足检索条件的文档编号列表(d1，d5)。

3.4. Google 的分布式索引架构

从总体上来看，Google 的分布式索引架构实际上是一种局部索引架构，如图 5 所示为 Google 的 Web 检索服务架构图[18]，图的左下部分是其分布式索引系统的架构：

上图中，Google 将整个的索引划分为多个 Index Shards，每个 Index Shards 负责一个文档子集索引，

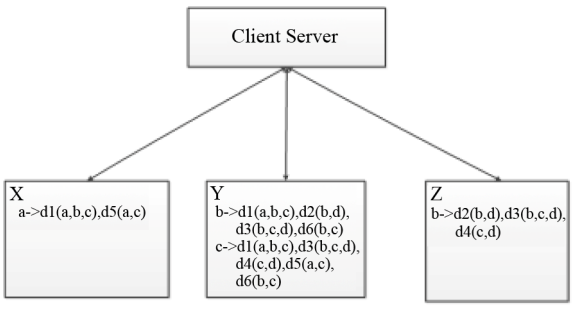

Figure 4. Hybrid indexing architecture
图 4. 混合索引架构示意图

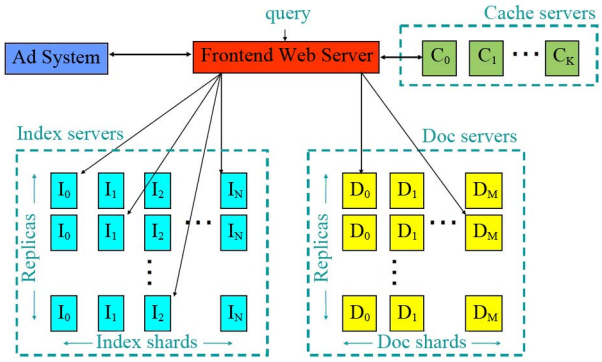

Figure 5. Google web retrieval service architecture
图 5. Google 的 Web 检索服务架构图

文档子集是随机选择的，因此各个 Index Shard 之间相互独立。每个 Index Shard 可能放在一个机器或一组机器中，当使用一组机器时，每个 Index shard 内部可以看成是全局索引架构。由于总体上是局部索引架构，所以系统存在吞吐量瓶颈的问题，Google 通过 Index Shard 的多副本机制来扩大吞吐量，同时这种机制也提供了较好的容灾容错性能(这里主要指单机的容灾容错)。

现在举例说明 Google 的分布式索引系统执行过程。由于总体上是局部索引架构，因此检索条件会被发送到每一个 Index Shard 上，每一个 Index Shard 都有一组副本与之对应，负载均衡器会动态选择执行检索的副本。对于每个 Index Shard 来说，所有的操作都是本地的，首先抽取检索条件的关键字，不同关键字检索得到不同的文档编号列表，然后将各个文档编号列表进行归并求交集，得到满足检索条件的文档编号列表，再将满足条件的文档编号列表中的文档编号按照信息的相关度(通常是一个数值)进行排序，得到最终用以返回的文档编号列表。每个 Index Shard 都会返回这样一个文档编号列表，索引服务器将所有的文档编号列表中的文档编号以其相关度为标准进行归并，即得到用户所需的最终结果，用户浏览到的结果是经过文档服务器进一步处理的。

3.5. URL 任意字符序列检索系统的分布式索引解决方案

该系统基于开源的 Hadoop[19]项目与 HBase[20]项目构建，其底层数据的组织是一个分布式文件系统。在索引较小不超过 100 GB 级别时，只有一个索引表，

相当于分布式数据库 HBase 中的一个数据表。由于 Hbase 本身即是分布式的，所以这时的架构实际上是一个全局索引架构。随着数据量的增大，检索时过长的记录编号列表归并过程是影响系统性能的关键，这时采用分而治之的策略，建立多个索引表，每个索引表负责一个时间段范围内记录的索引。检索时，各个表同时执行检索，具有很高的并发度。这样，从总体上来看系统又呈现了一种局部索引的架构。

可以看出，该系统的总体架构与 Google 的分布式索引架构类似，不过在底层使用的是分布式文件系统而不是多副本机制。该系统的搜索执行过程与 Google 系统也存在一些不同，在每个索引表里检索时产生的记录编号列表没有相关度之类的附加信息，所以不像 Google 的系统那样归并之后得到的结果还要依据相关度再次排序。

4. 索引架构的分析与选择

结合前人的研究成果与实际中架构的使用，本节对三种分布式索引的优缺点进行总结与分析，结合实际应用的讨论，最终在如何选择索引架构这个核心问题上给出本文的观点。

4.1. 局部索引分析

局部索引架构将整个数据集分成多个子集，这种划分往往是随机的，或者根据应用的具体情况来进行划分，然后针对各个数据子集来分别建立索引，因此建立的各个索引也相互独立。

4.1.1. 局部索引架构的优点

1) 局部索引架构的每个索引块可以独立完成检索。因为一个索引块包含了一个数据子集的全部检索关键字信息，故针对一个给定的检索条件，不需要其他索引块的任何数据，即可判定它对应的数据子集中哪些数据满足条件。全局索引架构则不具备这个特点，其每个索引块只是完整的大索引的一段，只包含针对整个数据集的部分关键字的索引信息，当一个检索条件有多个检索关键字时，这些关键字被一个索引块包含的概率较小，因此一个检索往往需要搜索多个索引块得到多个结果集列表(每个针对一个检索关键字)，最终的结果通过它们求交集得到。

2) 局部索引架构在数据信息更新时操作更为方便。由于数据集被分为多个数据子集，每个子集对应的索引块相互独立，故当数据子集里面的数据项(记录、文档等)发生更新时，相应的索引更新也只发生在一个索引块中。

3) 局部索引架构在数据传递时产生更少的网络流量。因为每个索引块返回的结果集列表中，每个列表项(记录编号、文档编号等)都满足检索条件。而全局索引架构每个检索关键字都返回一个结果集列表给中间节点，然后中间节点对多个检索关键字对应的不同结果集列表求交集，得到满足检索条件的结果集，因此搜索单个检索关键字所得到结果集列表一般包含不满足所有检索条件的列表项。

4.1.2. 局部索引架构的缺点

1) 由于局部索引架构对每个数据子集建立独立索引，所以一个检索会被发送到所有索引块上独立执行。而全局索引架构在抽取检索关键字之后，只将相应的关键字发送到包含该关键字的索引片段，若有 K 个关键字，最多只会有 K 个索引片段执行搜索。

2) 对于每一个检索，局部索引架构需要执行 K*N 次磁盘寻址及数据读取操作，其中 K 为检索的关键字的个数，N 为总的索引块个数。局部索引架构总是将一个检索发送到 N 个索引块上去并发的执行。单个索引块在执行一个检索时，首先会抽取检索条件里的关键字，然后对每个关键字执行一次搜索，得到 K 个结果集列表，这就需要 K 次的磁盘寻址及数据读取操作。因此，对于局部索引架构，一个检索需要执行 K*N 次磁盘寻址及数据读取操作。

4.2. 全局索引分析

全局索引架构针对整个数据集建立一个全局索引，然后将其划分为多个索引片段，每个索引片段负责整个关键字集合中的分关键字。索引的划分有一定的策略，不是随机划分的，划分的标准就是要能够通过中间节点快速判定一个关键字所在的索引片段。

4.2.1. 全局索引架构的优点

1) 全局索引架构在执行一个检索时需要访问的索引片段较少。假设一个检索有 K 个关键字，最多只有 K 个索引片段执行搜索，具体分析参见上文 4.1.2 节。与局部索引架构相比，全局索引架构具有更高的

吞吐率。

2) 全局索引架构在执行一个检索时只需要 K(K为检索关键字的个数)次磁盘寻址及数据读取操作，具体分析参见上文 4.1.2 小节。因此全局索引架构相比局部索引架构在执行一个检索时具有更低的开销(数据的顺序读取比随机读取具有更高的效率)。

4.2.2. 全局索引架构的缺点

1) 全局索引架构在数据传递时产生更多的网络流量，同时必须将所有的归并操作集中执行，而局部索引架构由于每个数据子集相互独立，在执行归并操作时拥有很好的并发度，详细分析参见上文 4.1.1。

2) 在全局索引架构中，每个数据项(记录、文档等)的关键字信息分布在整个分布式系统之中，与局部索引架构相比，全局索引架构在数据项发生更新时，同步完成对索引的更新难度更大。

4.3. 混合索引分析

混合索引架构在检索时兼具全局索引架构和局部索引架构的优点：较小的数据流量、较高的吞吐率、所有的检索都在索引块本地执行、较少的系统开销等等。

但混合索引架构的缺点也十分的明显：高数据冗余需要更多的存储空间，副本数量过多导致对数据项的更新难度更大、开销更高。特别是对一些数据项包含较多检索关键字的应用(如文档检索)，数据的冗余度可能高达数百甚至更高，显然，这在实际应用之中不易被接受。

4.4. 索引架构的选择

因为混合索引架构系统的成本过高，应用范围较窄，一般仅应用在一些需要快速响应且关键字数量较少的特定领域，这里不对其使用情况进行讨论。所以接下来我们主要在全局索引架构与局部索引架构之间展开比较与讨论。

在选择索引架构时，主要考虑如下两个因素：

1) 系统的响应时间。系统的响应时间必须使用户满意，没有良好的用户体验系统注定会失败。

2) 系统的吞吐量。就是单位实际内索引系统完成检索的次数，吞吐量越大越好，但相应的代价也随之快速增加，实际应用中要进行平衡考虑。

现在我们详细的分析影响分布式索引系统响应时间的关键因素：通过上文分析可知，局部索引架构在针对单个的检索时对磁盘的读取时并行执行的，相比全局索引架构具有些许的优势；考虑网络数据传输，全局索引架构传输的数据量较大，局部索引架构建立的网络连接数较多，不过这些连接的建立与使用都是并发的，所以在数据传输方面局部索引架构也具有一定的优势；大多数的检索条件都会包含多个检索关键字，每个关键字会得到其对应的结果集列表，整个归并过程的开销还是巨大的，若将其时间复杂度设为 O(n)，而局部索引架构整个的归并过程是并行执行的，其时间复杂度可表达为 O(n/N)，其中 N 为索引块的个数，显而易见，局部索引架构在结果集归并过程上相比全局索引架构有着较大的优势。综上所述，局部索引架构在响应时间上要优于全局索引架构。

在吞吐率方面，全局索引架构所有的归并操作都集中在中间节点上完成，造成中间节点的负载过大，所以实际的应用中要采用多个中间节点进行负载均衡的方法。局部索引架构由于每一个检索都会被发送到所有的索引块上，所以其所有的索引节点的负载都是很大的，实际应用当中每个索引块都必须有多个冗余副本，以提高吞吐率，虽然局部索引架构的中间节点不需要做大量的归并操作，但实际应用当中，出于吞吐率与高可用性的考虑，还是需要采用多个中间节点的策略。综上，实际应用中为了达到高的吞吐率，局部索引架构相比全局索引架构要付出更高的代价，所以在吞吐率方面全局索引架构是具有优势的。

在构建 URL 任意字符序列检索系统时，开始采用的是全局索引的架构，随着数据量的增大，经常会遇到响应时间过慢的情况，提交一个检索往往要数秒甚至数十秒之后才返回结果，经分析发现是由于归并时结果集列表过长所导致，所以对数据进行了划分，在总体上形成了一个局部索引架构。局部索引架构的整个检索过程在所有索引块上并行执行，因此在响应时将上要优于全局索引架构。

全局索引架构所有的归并操作都集中在中间节点上，实际应用中一般采用多个中间节点进行负载均衡。局部索引架构每一个检索都会被发送到所有的索引块上，所以各个节点的负载都很大，实际应用当中一般每个索引块有多个冗余副本，以提高吞吐率。因

此为了提高吞吐率，局部索引架构相比全局索引架构要付出更高的代价，所以在这方面全局索引架构是具有优势。

在以往的索引系统当中，索引的更新大多以批量的方式进行，当新增数据到达一定规模时，将原有的索引废除不用，对整个数据集重新构建索引。但随着数据更新的频率越来越快、规模越来越大，重新构建索引的成本也越来越高，所以增量构建索引的方式逐渐成为首选的解决方案[21]。局部索引架构在数据更新方面与其他分布式索引架构相比具有优势。

采用局部索引架构，索引块成千上万时，对中间节点也是极大的负担[22]。因为很多的应用要求根据相关度等信息来归并成千上万的结果集列表，这样的任务难以在几秒内完成。因此在数据规模过大时，单单的局部索引架构还不足以解决问题，有必要将两层的索引架构变为三层。总体采用局部索引架构，每个索引块又是一个单独的分布式索引系统。单个索引块所对应的分布式索引系统可以是局部索引架构，也可以是全局索引架构，如能满足应用需求，因全局索引架构开销更小，应优先采用。在成本允许的情况下，也可以使用一个商用服务器来存储处理一个索引块[23]。

5. 总结

通过前人的研究成果和对实际应用系统的分析，可以看出，当面临大数据规模时，局部索引架构是较好的解决方案。而未来数据规模只会越来越大，所以局部索引架构是首选的分布式索引系统架构解决方案。当数据达到局部索引架构也无法应对的规模时，则应该将两层的索引架构变为三层以满足应用需求。

参考文献 (References)

[1] M. Stonebraker. The case for shared nothing. Database Engineering Bulletin, 1986, 9(1): 4-9.

[2] A. Tomasic, H. Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. Proceedings of the Second International Conference on Parallel and Distributed Information Systems, San Diego, 1993.

[3] A. Abusukhon, M. P. Oakes, M. Talib and A. M. Abdalla. Comparison between document-based, term-based and hybrid partitioning. IEEE Conference on Applications of Digital Information and Web Technologies, 4-6 August 2008: 90-95.

[4] B. B. Cambazoglu, A. Catal and C. Aykanat. Effect of inverted index partitioning schemes on performance of query processing in parallel text retrieval systems. Unpublished manuscript.

[5] J. Zhang, T. Suel. Optimized inverted list assignment in distributed search engine architectures. Proceedings of the 21st Parallel and Distributed Processing Symposium, 36-30 March 2007: 1-10.

[6] D. Bhagwat, K. Eshghi and P. Mehra. Content-based document routing and index partitioning for scalable similarity-based searches in a large corpus, in KDD'07. Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2007: 105-112.

[7] A. Abusukhon, M. Talib and M. Oakes. An investigation into improving the load balance for term-based partitioning. In: R. Kaschek, et al., Eds., Proceedings of UNISCON 2008, The 2nd International United Information Systems Conference, Klagenfurt, 22-25 April 2008. Berlin, Heidelberg: Springer-Verlag, 2008: 380-392.

[8] C. Tang, S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. Proceedings of Networked Systems Design and Implementation, Grand Hyatt, 29-31 Match 2004.

[9] A. Guttman. R-trees: A dynamic index structure for spatial searching. Proceedings of SIGMOD, Boston, 18-21 June 1984.

[10] J. L. Bentley. Multidimensional binary search trees used for associative searching. Communications of the ACM, 1975, 18(9): 509-517.

[11] R. Mao, W. J. Xu, N. Singh and D. P. Miranker. An assessment of a metric space database index to support sequence homology. International Journal on Artificial Intelligence Tools, 2005, 14(5): 867-885.

[12] E. Chavez, G. Navarro, R. Baeza-Yates and J. L. Marroquin. Searching in metric spaces (survey). ACM Computing Surveys, 2001.

[13] R. Mao, S. R. Ramakrishnan, G. Nuckolls and D. P. Miranker. Case study: An inverted index for mass spectra similarity query and comparison with a metric-space method. Proceedings of the 3rd International Conference on Similarity Search and Applications, Istanbul, 18-19 September 2010: 93-99.

[14] R. Mao, W. J. Xu, S. Ramakrishnan, G. Nuckolls and D. P. Miranker. On optimizing distance-based similarity search for biological databases. Proceedings of the 2005 IEEE Computational Systems Bioinformatics Conference, Stanford University, 8-11 August 2005: 351-361.

[15] R. Mao, W. L. Miranker and D. P. Miranker. Pivot selection: Dimension reduction for distance-based indexing. Journal of Discrete Algorithms, Elsevier, 2012, 13(1): 32-46.

[16] R. Mao, W. L. Miranker and D. P. Miranker. Dimension reducetion for distance-based indexing. Proceedings of the 3rd International Conference on Similarity Search and Applications, Istanbul, 18-19 September 2010: 25-32.

[17] L. A. Barroso, J. Dean and U. Holzle. Web search for a planet: The Google cluster architecture. IEEE Micro, 2003, 23(2): 22-28.

[18] J. Dean. Challenges in building large-scale information retrieval systems: Invited talk. Web Search and Data Mining, 2009.

[19] Hadoop Apache Project. http://hadoop.apache.org

[20] Apache HBase Home. http://hbase.apache.org

[21] D. Peng, F. Dabek. Large-scale incremental processing using distributed transactions and notifications. Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, Vancouver, 4-6 October 2010.

[22] S. Melnik, et al. Dremel: Interactive analysis of web-scale datasets. Proceedings of the VLDB Endowment, 2010, 3(1): 330-339.

[23] U. Hoelzle, L. A. Barroso. The datacenter as a computer: An introduction to the design of warehouse-scale machines. San Rafael: Morgan and Claypool Publishers, 2009.