基于Cassandra的可扩展分布式反向索引的构建

唐李洋 倪志伟 李 应
(合肥工业大学管理学院智能管理研究所 合肥 230009)

摘 要 随着云计算时代的到来,大型Web应用的不断发展,海量数据不断增加,集中式的数据检索已不再满足需求。如何在分布式的环境中高效地处理数据检索成为亟待解决的问题。传统的关系型数据存储也无法完全适应云环境,NoSQL(Not only SQL)作为一种云存储形式应运而生,其中Cassandra的应用较为广泛。以分布式的多节点架构的索引构建为背景,提出了建立在分布可扩展的数据存储Cassandra之上的分布式反向索引(DII,Distributed Inverted Index),并给出了数据模型和查询处理流程的分析,最后给出了Cassandra的性能测试。

关键词 云存储,分布式索引,反向索引,Cassandra

Scalable Distributed Inverted Index Built on Cassandra

TANG Li-yang NI Zhi-wei LI Ying
(Institute of Business Intelligent Management, School of Management, Hefei University of Technology, Hefei 230009, China)

Abstract As the age of cloud computing, giant Web-scale application and massive data exert big challenges on centralized data retrieval. It becomes a tricky issue to process efficiently data retrieval in distributed environment. Traditional relational database is not fully suitable to the cloud any more. As a novel cloud storage, NoSQL (Not only SQL) emerges, among which Cassandra is widely used. With the application of index building under the distributed multi-node architecture, a scalable distributed inverted index built on Cassandra was put forward to solve the problem. Besides, analysis on data model and processing flow, as well as a performance test about Cassandra, were proposed.

Keywords Cloud storage, Distributed index, Inverted index, Cassandra

1 引言

随着计算机网络的高速发展,加之并行计算、分布式计算和虚拟化技术的不断进步,为云计算[1]这一新兴的计算模式带来了机遇,其中云存储[18]是云计算研究中的重要部分。近几年NoSQL[2,3]成为新型数据存储的代名词,它最早由Carlo Strozzi于1998年用来命名一个基于文件的数据库[4]。这是一个没有SQL接口的关系型数据库,但却与NoSQL存储不尽相同。NoSQL是传统关系型数据库(RDBMS)的下一代,是Not Only SQL的简称,它是随着Web 2.0应用的不断发展及数据和请求的不断增长而产生的。RDBMS具有"关系"和"依赖"等本质特征,要求强一致性。而CAP理论[5]告诉我们,一致性(Consistent)、可用性(Available)和网络可分割性(Partition-tolerant)3者无法同时满足,RDBMS选择了CA,却不能很好地支持分布式。RDBMS在小规模应用和小数据量时运作得很好,但随着Web-scale的数据(TB甚至PB级)和应用日益增多,关系数据库越来越无法支撑,此时NoSQL应运而生。NoSQL在CAP中选择了AP,采用最终一致性[6](eventually consistent),从而很好地适应需求。

NoSQL作为下一代数据存储,着重解决非关系的、分布式的、可扩展的问题。Google使用Bigtable[7]作为其分布式文件系统、计算模型和底层存储结构,用于分布式地存储结构化的大规模数据,采用列(column)和列族(column family)进行列式存储。Amazon的高可用、可扩展的存储体系Dynamo[8]支撑了Amazon不少核心服务,它是一个完全去中心化的系统,人工管理很少,且保证总是可写(always writable)。Facebook使用Cassandra来处理用户的inbox search。Cassandra[9,11,12]结合了Bigtable的列族和Dynamo的分布式系统技术,越来越多的应用开始转向Cassandra,如twitter,Digg,Racksapce,Cisco等。Cassandra逐渐成为RDBMS的替代选择。

索引的构建是NoSQL的一个典型应用。本文首先描述了一个典型的应用场景,进而分析并设计了Cassandra上的数据模型,提出DII(Distributed Inverted Index)模型实现高效的分布式反向索引,并保证可扩展性;最后给出了Cassandra的性能测试。

2 场景描述及术语定义

现有一个社会网络(Social Network Service,SNS)站点,考虑其搜索[17]功能的实现。起初,站点访问量不大,数据不

到稿日期:2010-07-01 返修日期:2010-10-05 本文受国家自然科学基金(70871033),国家高技术研究发展计划(863)(2007AA04Z116),国家社会科学基金项目(10CGL024)资助。

唐李洋 女,博士生,主要研究方向为云计算、海量数据处理,E-mail:tangliyang921@gmail.com;倪志伟 男,教授,主要研究方向为商务智能、机器学习、云计算等。

多，仅使用一个 Oracle server 存储所有的数据。后来，随着业务增加，数据量增大，单数据库显然不能满足需求。现在有两种选择：scale up 和 scale out[10]。scale up 是纵向扩展，强调单个节点的性能提升，成本较高，难以实现；而 scale out 是横向扩展，通过添加节点来达到更高的处理能力和性能。所以考虑采用分布式的多节点架构。

将数据库水平分割(horizontal partition)，采用 Database Sharding(Shared-nothing)[15]结构，将数据无共享地分割到不同的数据存储服务器上，底层数据存储使用 Cassandra，如图 1 所示。通过 RESTful[13]架构风格，所有数据访问都是以对象(object)为单位，object 具有唯一标识 ObjectID 和多个属性(property)——包括属性名(以 xpath[16]的路径元素标识)和属性值(value)。当浏览器的查询请求传来一个 URL 时，中间框架层可以将需要访问数据的部分剥离开来，识别出要查询的目标对象类型(object type)及该 URL 中包含的 xpath 和 value 对，然后传给 Cassandra 节点，通过调用 Cassandra 的数据访问 API，返回符合条件的所有 ObjectID 列表。ObjectID 按照指定的规则产生：为了使得对象本身天然地反映它所在的存储节点而不用额外的机制进行寻址[14]操作，我们将 ObjectID 的前 16 个字节作为 ShardID。中间框架层根据这些返回的 ObjectID，识别出各自的 ShardID，到各个节点上去装载数据，然后返回给 Web 层进行其他渲染等处理。最后，由 Web server 将处理后的结果返回给客户端用户。

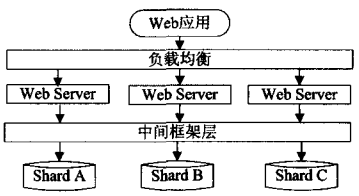

图 1　系统总体结构图

Cassandra 存储数据以 keyspace，column family，super column，column name 和 value 来表示。本场景需要两个数据结构：一个用于构建索引，以便在运行时回答查询请求；还有一个在检索到 ObjectID 以后，由中间框架层装载对象。存储的数据为 ObjectID，xpath，value。

1)检索阶段：客户端传来一个 URL，由中间框架层解析出其中的 xpath 和 value，然后到存储中由 xpath 和 value 检索到所有符合条件的 ObjectID，记 xpath+value—>ObjectID。

2)装载阶段：根据 ObjectID 查询到所有要显示在页面的 Object 简要信息(记 ObjectBrief)，记 ObjectID—>ObjectBrief。

DII(Distributed Inverted Index)用于处理检索阶段的问题，本文对于对象装载阶段不做详细阐述。

假定 DII 支持：

1)单条件检索：EQUAL(等值查询)，NOT，IN，LIKE(模糊匹配)，GT/LT(大于小于)等；

2)多个检索条件之间的 AND 和 OR；

3)二者的混合查询。

为了便于描述问题，现将本文中使用的一些术语做简略介绍。

定义 1(属性匹配)　若某对象 o 的某个属性 p 刚好等于给定的值 v，则称该对象 o 属性匹配 v，记 $op=v$。

定义 2(属性向量)　给定一个值 v 和一个对象集合 $O=\{o_1,o_2,\cdots,o_n\}$，$\exists V=(v_1,v_2,\cdots,v_n)$，满足 $\forall o_i\in O$，若 $o_{ip}=v$，则 $v_i=1$，否则 $v_i=0$。称 V 为属性 v 的属性向量，记 $V(v)$。

定义 3(属性数组)　存在一个属性向量 $V(v)$，$\forall v_i\in V$，v_i 在 V 中的位置为自右向左从零开始递增 1，记 pos_i，则 v_i 等价于一个长整型 power(2，pos_i)。因此，属性向量 $V(v)$ 可转化为一个属性数组，记 $B(v)=\{\text{power}(2,pos_i)\}$。

定义 4(序列，sequence)　将对象集合 $O=\{o_1,o_2,\cdots,o_n\}$ 表示为一个从零开始自增 1 的长整型数组，这个数组称为一个序列(sequence)。其中的每个元素唯一地表示 $o_i\in O$，称这个长整型数字为一个序列号，记 seqID。

定义 5(段(segment)和局部序列)　将对象集合 $O=\{o_1,o_2,\cdots,o_n\}$ 分成大小固定的段，每个段内包含 64 个 object，以 8 个字节表示，即一个长整型(long)，每个段具有唯一标识 segID。段内的 object 具有 0～63 的序列号，这个序列称为局部序列，记为 local_seqID。

3　数据模型

3.1　模型分析

采用属性向量 $V(value)$ 表示符合条件的对象集合(也可通过属性数组来实现)，大大减少了存储一个长串 ObjectID (32B)的空间消耗。还可以使用其他压缩方法，如只存储 1，而 0 直接丢弃。另外，通过与 ObjectID 的直接映射，降低了复杂度。那么，如何将属性向量 $V(value)$ 中的 0，1 值与具体的 ObjectID 对应起来？

首先，生成序列号。

Object 是不断增长的，且 ObjectID 没有直接反映对象之间的关系，这就使得直接从 ObjectID 转化得到属性向量变得愈加困难。我们采用序列(sequence)的方法，将 ObjectID 映射到一个长整型(long)数字。序列的生成方法很简单，序列初始值为 0，在新创建一个 object 时，序列号加 1。这样，一个序列号就代表一个 ObjectID(注：产生的序列号和 ObjectID 的对应关系也存在 Cassandra 中，只是结构十分简单，仅为 key/value 存储，只在创建对象和最终返回 ObjectID 列表时被访问)。

为了避免序列不断增长和处理序列的方便，将序列进一步划分为 64 bit 的段(segment)。整体上看，seqID 是唯一的。每个段内部的 local_seqID 取值范围为 0～63，且遵循从零开始自增 1 的规则。local_seqID 和分段信息 segID 共同决定 seqID，它们之间的关系如下：

$$seqID=segID<<8+local\_seqID;$$

$$segID=seqID/64;$$

$$local\_seqID=seqID\%64。$$

其次，属性匹配的表示。

段 segID 内部，若 local_seqID 对应的 ObjectID 属性匹配 value，则属性向量 $V(value)$ 中自右至左 $posi$ 处置 1，其中 $i=$ local_seqID，$i\in[0,63]$。属性向量 $V(value)$ 以 Byte 数组的形式存储在 Cassandra 节点中，表示属性匹配 value 的所有 ObjectID 对应的 sequence。

最后，建立索引的过程如下：

```
For each path+value,do
  For segID 0:MAX do
```

```
    For local_seqID 0:63 do
      If Objectp=value then
        Set power (2,local_seqID)=1
        Else Set power (2,local_seqID)=0
      End if
    End for
  End for
End for
```

3.2 数据结构

具体模型设计有以下几点，如图 2 所示。

1)定义两个 ColumnFamily:一个用于文本型属性值的查询，名为 StringType，按 UTF8Type 排序列；另一个用于数值型属性的查询，名为 LongType，按 LongType 排序列。

2)行关键字(key)为 xpath，保证一个 xpath 的查询限定在一个节点上，即每个单条件的子查询都在本地节点执行，而不用跨节点，降低了复杂度。

3)将 value 作为超列(super column)，根据 ColumnFamily 的排序类型 CompareWith 对 value 进行排序，从而 value 是有序排列的，便于执行 LIKE,GT/LT 等操作。

4)将 segID 作为列(Column)，按照 LongType 排序，由 CompareSubcolumnsWith 定义。

5)值为 value 对应的属性向量 $V(value)$，存储为 Byte 数组，表示所有属性匹配 value 的对象集合。

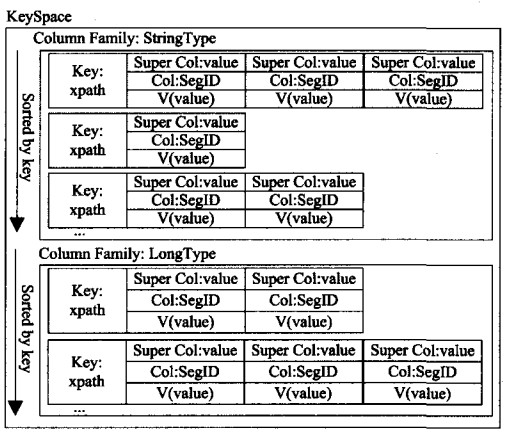

图 2 数据结构

具体定义如下：

```
〈ColumnFamily Name="StringType"
        ColumnType="Super"
        CompareWith="UTF8Type"
        CompareSubcolumnsWith="LongType"
        RowsCached="100"
        KeysCached="10%"/〉
〈ColumnFamily Name="LongType"
        ColumnType="Super"
        CompareWith="LongType"
        CompareSubcolumnsWith="LongType"
        RowsCached="100"
        KeysCached="10%"/〉
```

3.3 查询过程

当按照上述数据模型和规则构建好索引后，处理来自客户端的查询请求。整个查询处理过程如图 3 所示。

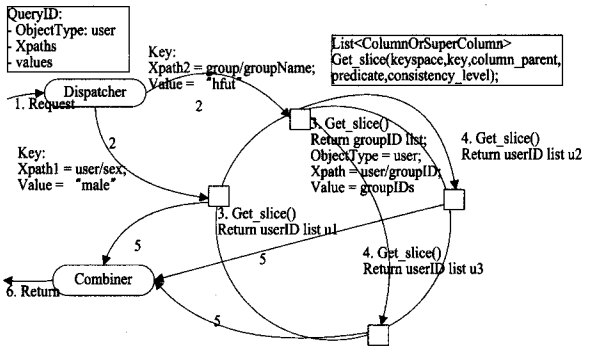

图 3 查询处理流程

假定有一个查询请求，检索出所有在 hfut 组的男性用户，则处理过程如下：

1)中间框架层剥离出 URL 中的 ObjectType，xpath 和 value 等，交给 Dispatcher。

2)Dispatcher 首先赋给该查询请求一个 QueryID，将 xpath 和 value 对拆成不同的子查询，并根据关键字 xpath 将请求发送给相应的 Cassandra 节点。

3)Cassandra 节点调用 API 访问数据。

a)若查询发送到该节点的子查询只有一个，则直接将结果集合传给 Combiner；

b)若查询发送到该节点的子查询有多个，则 Cassandra 节点还要负责这些子查询之间的 AND 或 OR；

c)若查询包含对象间的关系，如 groupName，groupID，userID 表达了 user 对象(记 Object)和 group 对象(记 ObjectReference)的关联关系，则查询需要分两步进行。第一步得到 ObjectReferenceID，第二步根据关联路径(如 user/groupID)和 ObjectReferenceID 得到 ObjectID。这时有个潜在的限制，即对象的 ObjectReferenceID 不能太多。事实上，ObjectReferenceID 应该不多。如果 ObjectReferenceID 相当多，则应当作为不同的客户端请求以免查询结果不清晰。

4)Cassandra 节点将本地结果返回给 Combiner，由 Combiner 汇总各个节点的数据，执行 AND 或 OR，然后再将序列号转换成 ObjectID，返回给客户端。

3.4 优化分析

3.4.1 查询范围

以 xpath 作为关键字，检索范围较大，且将如此多的数据都放在一个节点上，数据分布不够均匀，可能带来单个节点过热。所以，考虑在 key 中加入范围限制，以进一步打散数据分布，如容器 ContainerID 等。

3.4.2 Query Cache

对于某些常用的查询和经常访问的对象，使用 Cache 缓存可以大大加快响应速度。如新加一个 QueryCache 的 ColumnFamily，如图 4 所示。

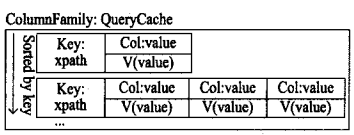

图 4 QueryCache 结构图

3.4.3 执行计划

不同子查询之间的执行顺序可能会对查询速度造成较大

的影响。初期随机执行子查询，并记录统计数据，然后根据统计数据的信息生成最佳的执行计划。

4 性能测试

本小节对 Cassandra 的多线程和批写入操作进行简单的测试，充分说明基于 Cassandra 的分布式反向索引具有较高的性能。测试环境为 2 台 PC，处理器为 Intel Core2 Dure 7400，4G 物理内存，JVM 内存为 1G，SATA 磁盘大小 250G，网络速度为 100M。

4.1 多线程测试

测试结果如图 5 和图 6 所示。其中 X 轴表示线程数，Y 轴表示访问 5000 条记录所需的总时间。

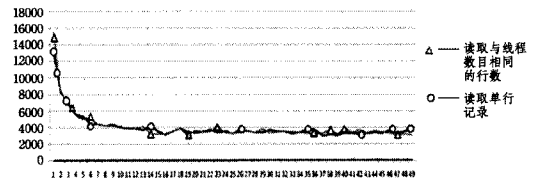

图 5 50 个线程测试

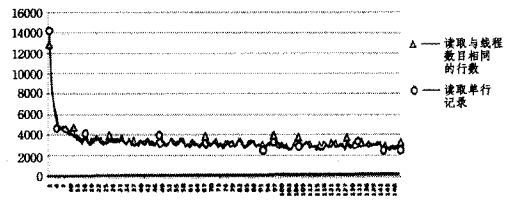

图 6 150 个线程测试

由上可知，Cassandra 的并行能力是可接受的。可能的原因是，Cassandra 只向内存中的 Commit log 和 SSTable 写数据，然后 flush 到磁盘，lock 的时间较少，并发性较好。然而，受限于测试环境，当线程数目达到 15 时，整体的访问时间几乎稳定在一定水平，所以 15 个线程是最好的并发点。

4.2 批写入测试

测试结果如图 7 所示。其中 X 轴表示写入的记录数(1～5000)，Y 轴表示所需的总时间。

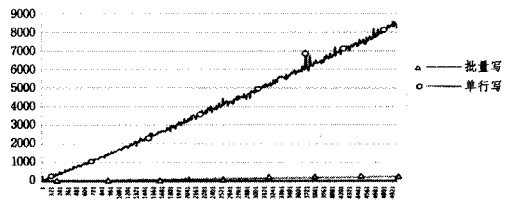

图 7 批写入测试

可见，Cassandra 对于写入数据性能很好，所以它可以高效地解决 DII 的索引更新问题。

结束语 提出了一个基于 Cassandra 的分布式反向索引(DII)，以解决传统关系型数据存储无法解决的可扩展性问题。借助于 Cassandra 良好的分布式处理性能，在此基础上分析与设计了数据模型和查询处理过程，并给出了可能的优化方案。最后通过几个简单的测试说明了 DII 建立在 Cassandra 上其性能的可行性。需要说明的是，本文没有重点分析查询优化部分，这将是下一步研究的核心。

参 考 文 献

[1] Miller M. Cloud Computing: Web-based Applications That Change the Way You Work and Collaborate Online[M]. Que Publishing Company, 2008

[2] Wikipedia NoSQL [EB/OL]. http://en.wikipedia.org/wiki/NoSQL, 2010-6-20

[3] NoSQL database [EB/OL]. http://nosql-database.org/, 2010-6-20

[4] NoSQL: a non-SQL RDBMS [EB/OL]. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page, 2010-6-20

[5] Gilbert S, Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services [J]. SIGACT News, 2002, 33(2): 51-59

[6] Vogels W. Eventually Consistent[J]. Queue, 2008, 6(6): 14-19

[7] Chang F, Dean J, et al. Bigtable: A Distributed Storage System for Structured Data. ACM Trans. Comput[J]. Syst, 2008, 26(2): 1-26

[8] DeCandia G, Hastorun D, et al. Dynamo: Amazon's highly available key-value store[C]// Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles. Stevenson, Washington, USA, ACM, 2007: 205-220

[9] Apache Cassandra [EB/OL]. http://cassandra.apache.org/, 2010-06-20

[10] Wikipedia, Scale out [EB/OL]. http://en.wikipedia.org/wiki/Scale_out#Scale_vertically_.28scale_up.29, 2010-06-20

[11] Lakshman A, Malik P. Cassandra: structured storage system on a P2P network[C]// Proceedings of the 28th ACM Symposium on Principles of Distributed Computing. Calgary, AB, Canada, ACM, 2009

[12] Lakshman A, Malik P. Cassandra : a decentralized structured storage system[J]. SIGOPS Oper. Syst. Rev, 2010, 44(2): 35-40

[13] Richardson L, Ruby S. RESTful Web Services: Web services for the real world[M]. O'Reilly, 2007

[14] Hildrum K, Kubiatowicz J D, Rao S, et al. Distributed Object Location in a Dynamic Network[J]. ACM, 2004

[15] Wikipedia, Sharding [EB/OL]. http://en.wikipedia.org/wiki/Sharding, 2010-06-20

[16] Bonifati A, Cuzzocrea A. Storing and retrieving XPath fragments in structured P2P networks[J]. Data Knowl. Eng, 2006, 59(2): 247-269

[17] Hatcher E, Gospodnetic O. Lucene in Action (In Action series) [M]. Manning Publications Co, 2004

[18] Zeng W, Zhao Y, et al. Research on cloud storage architecture and key technologies[C]// Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human. Seoul, Korea, ACM, 2009: 1044-1048