第 40 卷 第 11 期 计 算 机 工 程 2014 年 11 月
Vol. 40 No. 11 Computer Engineering November 2014

·体系结构与软件技术· 文章编号：1000-3428(2014)11-0060-05 文献标识码：A 中图分类号：TP311

图引擎底层存储的设计与实现

马洪宾，陈贵海

（上海交通大学计算机科学与工程系，上海 200240）

摘 要：随着社交网络和语义 Web 等数据应用的兴起，催生了许多图数据处理产品，包括 Neo4j，HyperGraphDB 等，然而这些产品在设计时并未充分考虑图应用对数据可用性和可扩展性的更高要求。为此，提出一种基于分布式内存云的图引擎底层建模和存储解决方案。在内存云上搭建分布式键值引擎，进而在键值存储的基础上对图的数据进行建模和读写。在大规模数据集上的实验结果表明，该方案具有较好的图随机访问性能，并能够高效地支持海量规模的图数据应用。

关键词：图处理；云计算；分布式；数据建模；存储；数据结构

中文引用格式：马洪宾，陈贵海. 图引擎底层存储的设计与实现[J]. 计算机工程，2014，40(11)：60-64.

英文引用格式：Ma Hongbin，Chen Guihai. Design and Implementation of Underlying Storage for Graph Engine[J]. Computer Engineering，2014，40(11)：60-64.

Design and Implementation of Underlying Storage for Graph Engine

MA Hongbin，CHEN Guihai

（Department of Computer Science and Engineering，Shanghai Jiaotong University，Shanghai 200240，China）

【Abstract】 Graph applications rise with the emerging of social network and semantic Web，and generate many graph data processing products，including Neo4j，HyperGraphDB，etc. However，current solutions fail to take into consideration graph applications' higher requirements on data availability and scalability. This paper proposes a modeling and storage solution based on distributed memory cloud. It takes advantage of the prior work to build a key-value system over the memory cloud，then builds data modeling and read-write based on it. Experimental results on large scaled datasets show that this solution has a good figure random access performance，and it can support massive graph applications efficiently.

【Key words】 graph processing；cloud computing；distributed；data modeling；storage；data structure

DOI：10.3969/j.issn.1000-3428.2014.11.012

1 概述

图是最常见的数据结构之一，与线性表和树相比，它的结构更加复杂，对数据的表现能力也更丰富。传统的图应用包括道路分析[1]、论文引用分析[2]、网页链接分析[3]等。随着社交网络[4-5]、语义网络[6]等研究领域的兴起，由于图在表示实体间关系方面的显著优势，因此越来越多的研究尝试使用图来存储和挖掘数据。

学术界和工业界已有大量工作致力于图的应用和理论研究。截止目前，已有很多图数据库产品可供选择。但是随着数据规模的日益增长，图的规模也随之呈现爆发式增长的趋势。如何有效地存储和使用海量规模的图数据集，成为图数据库领域内的一大难题。

本文结合图数据应用对可用性和可扩展性的要求，提出一种基于分布式内存 Key-Value 引擎的图数据存储和建模方案，介绍底层使用的分布式内存云引擎，根据图应用对数据建模的需求分析建模技术，并将已知的语义 Web 数据集导入到系统中，对系统性能进行全面分析。

2 图数据应用及其挑战

2.1 使用图数据结构进行数据建模的优势

在已知的大数据问题中，有很大一部分的问题可以由图进行更为直观的建模。而且这种直观的建模方式可以带来高效的数据读写效率。以社交网络为例，网络中的两大元素：人物与人物之间的关系，可以分别对应到图里面的顶点和边。假设采用邻接表的形式存储图的边，那么所有的边都可以保存在

作者简介：马洪宾(1989－)，男，硕士研究生，主研方向：数据查询处理，分布式系统，云计算；陈贵海，教授、博士生导师。

收稿日期：2013-11-18 修回日期：2013-12-17 E-mail：790123072@qq.com

顶点上。人物的一些属性,例如姓名、年龄等,可以作为顶点的属性存放在顶点内。与此相对地,也可以使用关系数据库将数据建模成为一个人物表和一个朋友关系表。当客户端程序需要获取某个特定人物的朋友列表时,假设采用图建模的方式,程序可以首先找到代表该人物的图顶点,然后通过一次顶点内的访问获得其所有邻居。而在关系数据库中,需要将朋友关系表与人物表进行一次内联结才能达到相同目的。当然,用户可以选择对朋友关系表中的外键来加速这一过程,但这又势必会带来了额外开销。

无论是在单机的多核计算处理器上,还是在分布式环境中的多机多处理器环境下,对并行计算是否友好都已经成为衡量一个模型好坏的重要标准。在图模型的领域,以顶点为中心的计算模型因为它的简单性、可扩展性和灵活性而被广泛地采用。在以顶点为中心的计算模型中,每个顶点都可以成为最细粒度的计算单元,一个或者多个顶点的计算工作可以由同一个操作系统线程或进程负责,以顶点为中心的计算模型可以将工作量自然地切割,并且适用于同步和异步、集中式和分布式的计算模型。

2.2　图数据库面临的挑战

传统的图数据库面临可用性与可扩展性[7]之间的博弈。传统图数据库可以粗略地分为以下 3 类:

(1)基于磁盘存储的单机实现

已有的借助磁盘存储空间的单机图数据库,虽然能够在假设磁盘空间没有限制的前提下,克服数据的规模问题。根据程序访问局部性原理,这类图数据库也可以在内存中缓存适量的缓存,以期减少数据访问的开销。然而,图应用的随机访问性质决定了程序在图中对顶点的访问呈现随机的特征,因此难以找到一种简单通用的缓存算法来加速数据访问。而大量的随机读写是对磁盘访问速度的灾难。因此,基于磁盘的解决方案会带来不能接受的性能问题。

(2)基于 MapReduce/Hadoop 的实现

这类实现的典型代表为 PEGASUS[8],一个基于 Hadoop 的分布式图信息挖掘系统。Hadoop 对应 Google 的分布式计算框架 MapReduce[9],它的文件系统 HFS(Hadoop File System)对应于 MapReduce 的 GFS(Google File System)[10]。图数据以文件的形式存放在 HFS 中。虽然这类实现在可扩展性和容错性方面得到了保证,但是 Hadoop 的计算模型与图数据上的计算需求并不完全匹配。Hadoop 的作业调度与任务分配要求对数据的反复读写,这造成了同样的图信息被反复地在磁盘中读写,增加了系统的 I/O 开销。另外,为离线数据分析而设计的 Hadoop 也不能满足对反应时间具有更高要求的在线图数据查询需求。

(3)基于分布式和列式存储的实现

在 MapReduce 后,Google 相继推出了基于 GFS 的列存储引擎 BigTable[11]以及 Dremel[12]。这类列式储引擎将数据的不同维度(列)单独存储,并且可以灵活地根据数据的使用频率,将某些常用维度的数据配置在内存中,以减少磁盘 I/O,提高读写性能。列式存储需要对数据模型具有严格定义。在图数据中,使用邻接表来表示顶点间的关系。邻接表的一个基本性质是其长度可变,而像 BigTable 这样的列式存储引擎对这一性质并无很好的支持。可变长、可嵌套、可重复的列成员在 Dremel 中被引入,但是由于 Dremel 是一个只读的交互式在线分析系统,图数据的顶点增加/删除、关系的增加/删除就无法被支持。

3　基于分布式内存云的图引擎存储解决方案

3.1　分布式内存云引擎——Trinity

Trinity 是微软亚洲研究院设计并实现的服务于云计算的一款轻量的高性能分布式内存 Key-Value 存储引擎[13]。在 Trinity 系统中,所有数据都被保存在内存云中,因此能够支撑每秒数百万次的随机读写。另外,Trinity 支持通过配置服务器集群的方式,灵活地调整系统的服务能力[14]。Trinity 向客户端提供以下统一接口:

```
byte[ ] getBinary(long id)
void setBinary(long id,byte[ ] blob)
```

Trinity 作为一个 Key-Value 存储引擎,其中的 Key 仅限长整数类型,而 Value 则是一段不定长的字节数组,称为 BLOB。因此,Trinity 可以看作是一个 long->BLOB 的 Key-Value 存储引擎。

从单机角度观察,Trinity 在启动时向操作系统申请大块的内存,用以对 BLOB 进行动态储存。Trinity 封装了高效的内存管理模块,可以进行高效的垃圾收集和内存清理。另外,Trinity 提供细粒度的锁机制,保证对于单个键值对的修改是原子的。从集群角度观察,每个 Trinity 的服务器实例负责维护一批 BLOB,Trinity 系统通过对 Key 的哈希对不同 BLOB 进行分割。各个实例之间由高速以太网互联。

3.2　在 BLOB 上的图顶点建模

Trinity 中的每个 BOLB 都是独立的个体,彼此之间互相独立,且每个 BLOB 拥有全局唯一的长整数标识符。在 BLOB 的不同区间段内存储不同的信息,可以自然地将 BLOB 隐射成图数据中的顶点,如图 1 所示。每个顶点中存储的信息可以分为两部

分:属性和边。然而,BLOB 只是简单的字节数组,无法为上层应用提供更多数据格式的信息。因此,需要在 BLOB 上搭建数据访问层,以便上层图处理程序能够有效获得感兴趣的图信息。

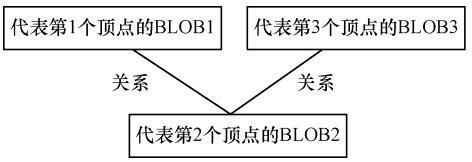

图 1 每个 BLOB 视作一个图顶点的情况

图的应用千变万化,对图顶点和边的定义也会存在巨大差异。例如,一个典型社交网络可能需要定义人物这样的图顶点,也需要把人物和人物之间的关系定义为图中的边。而在一个语义 Web 的图应用中,用户可能更希望将任意的主语和宾语定义为图顶点,而将谓词定义为图中的关系。总而言之,一个通用的图引擎无法提前预知图应用所需的数据结构,因此允许用户提前对图的结构进行定义。例如,社交网络中的人物顶点可以由以下语句定义:

```
Node Person
{
    int age;
    String name;
    List < long > friends;
}
```

根据定义,所有人物顶点的实例都由一个整数记录其年龄,一个字符串记录其姓名,还有一个可变长的长整数容器记录其所有朋友的标识符。对顶点的定义模仿了面向对象语言(例如 Java)中对类的定义。然而,在实际存储方式上却完全不同。在面向对象语言中,对象只保存各个非基本类型成员的引用,成员具体的内容保存在堆上,其物理地址并不一定相邻。而在实现中,由于需要将图顶点所有的信息保存到一段 BLOB 中,成员会按序依次排列,所有成员在存储逻辑地址上是保证相邻的,称为线性排列。

采用基于 BLOB 的线性排列,而不采用类似于面向对象语言基于堆的储存方式的主要原因有以下 2 个方面:

(1)前者相对于后者更具空间优势。为了克服传统图数据库在可用性上的瓶颈,达到最高效的访问速度来适应图的随机访问性质,考虑将大部分的图数据放入内存中。尽管采用分布式的内存云引擎作为底层存储,但是考虑到内存云的代价,更经济有效地利用内存空间仍然具有重要的意义。以对象为单元存储图顶点,涉及到引用类型的开销,以及对象本身的开销(例如对象上的锁)等。事实上,在 64 bit. NET 平台上,一个空对象也需要占据 12 Byte 的空间。因此,以 BLOB 的形式存储图顶点更具空间优势。

(2)以 BLOB 形式存储的节点更有利于在分布式环境中分发传送。在不同机器间传送对象,需要首先在发送端将对象序列化成字节序列,然后通过网络传送至接收端,并由接收端负责将字节序列反序列化成对象,单个对象的序列化和反序列化的开销可能难以察觉,但是在大型的图应用中,通常需要在不同机器间传输成千上万的图顶点。在这种情况下,序列化和反序列化的代价就变得十分显著。然而,基于 BLOB 的储存形式无需经历序列化和反序列化的过程,BLOB 本身就是字节数组,可以直接在网络上传输。

3.3 具体实现

为了能够用方便的接口读写 BLOB 中的数据,需要将用户定义的图顶点类型编译并生成相应的访问器。

3.3.1 访问器类的生成

对于每种类型的图顶点,为它的每一个成员分配一个访问器。根据成员的长度是否固定,访问器又可以细分为定长访问器和变长访问器。定长访问器适用于对基本类型成员的访问,而可变长的成员如字符串、线性容器等需要由变长访问器来访问。例如人物顶点中,由于 age 成员占据固定 4 Byte 的空间,因此只需分配一个定长访问器 IntegerAccessor 来访问它,而对于 name 成员来说,由于无法提前确定该字符串的长度,使用变长访问器 StringAccessor 访问它。同理,成员 friends 的类型是长整数线性表,也需要生成一个变长访问器 LongListAccessor 才能访问它。

可以看到,对于每种成员类型,需要为其生成对应的访问器类型。已知的最大规模图应用需要定义数千种顶点类型,但是由于不同顶点类型的成员类型存在大量重复,例如同样的 StringAccessor 适用于任何拥有字符串成员的顶点类型,因此访问器类型的种类数目反而不是很多。访问器访问 BLOB 内数据的方式如图 2 所示。

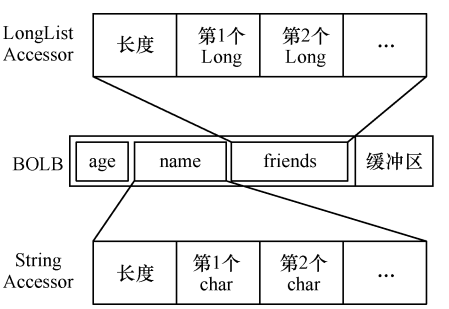

图 2 使用访问器的 BLOB 数据访问

在初始化访问器实例时,传入成员的指针,以便访问器知道成员从何开始。对与定长访问器而言,无需额外的信息,即可了解数据存放的格式。例如,如果在给定开始指针之后,一个 IntegerAccessor 就明白在开始指针之后的 4 Byte 就是需要访问的整数数据。而对于变长访问器而言,需要借助一些辅助信息才能够确定数据的格式。例如,对于所有简单数据类型的线性表容器,如 List < long > 类型,在成员的开始用一个整数的空间存放该容器的大小,当一个 LongListAccessor 访问这个成员时,首先读取首部 4 Byte 的容器大小信息,然后将指针向后偏移 4 Byte,才开始真正访问数据。

3.3.2　动态容器扩容的支持

图应用的数据处于经常性的变化之中。社交网络中的人物随时会增加新的好友关系,同时也有可能解除原有的好友关系。因此,可支持动态增减的容器类型不可或缺。由于本文设计一个图顶点所有的数据顺序存放在一个固定大小的字节数组中,因此任何一个容器的扩容都有可能导致空间不足。可以观察到,在图 2 中,BLOB 的尾端有一部分不属于任何成员的空间,称为缓冲区。当任何一个容器试图扩容,请求更多的空间时,如果缓冲区的大小足够,那么该扩容请求可以通过简单的向后生长来实现。如图 3 所示。如果缓冲区的大小不足以支持当前的扩容请求,那么需要向 Trinity 系统申请一块更大的内存区间来存放扩容后的新数据,同时把原有的内存区间标记为废弃,以便 Trinity 内置的内存垃圾收集器能够回收利用。

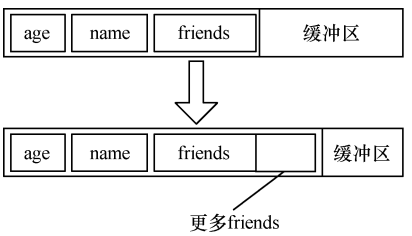

图 3　容器扩容

3.3.3　嵌套支持

为向图应用提供更丰富的建模工具,允许用户自行定义除了图顶点之外的结构体,并允许在图顶点的定义中直接嵌套使用它。以社交网络中的人物为例,如果对于每个好友,不仅希望保存他的标识符,还希望保存对各个不同好友的备注签名,那么可以将好友关系定义为一个特定结构体,在这个结构体中分别记录好友的标识符和对好友的备注签名,这种情况下人物的朋友成员可以按以下方式定义:

```
struct Friendship
{
    String alias;
    long friendID;
}
Node Person
{
    int age;
    String name;
    List < Friendship > friends;
}
```

在存储上,结构体会以和基本类型相似的方式,顺序保存在 BLOB 中。不同的是,系统会为每个结构体生成特别的访问器,保证结构体能被正确地读写。另外,结构体的引入增加了成员内存管理的复杂度,为了支持嵌套成员的扩容和缩减,每个成员都需要保存其上层成员的 Resize 函数指针,在需要扩容或缩减时,各个成员递归地调用 Resize 方法,直到传递到最终能够处理该事件的图顶点自身。对于好友关系中的备注签名成员 alias,如果希望将其修改成更长的字符串,就需要向上递归地调用结构体 Friendship、容器 FriendshipList 以及图顶点 Person 的 Resize 函数。该过程如图 4 所示。

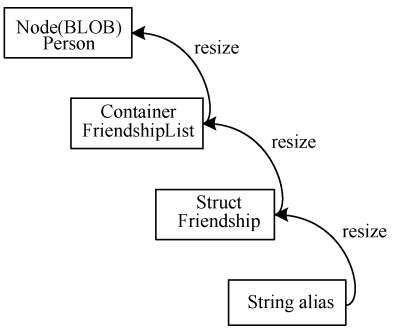

图 4　递归调用 Resize 方法

4　实验结果与分析

基于分布式内存云的图引擎存储解决方案,首先借助内存云的优良扩展性,解决了可扩展性的问题,能够对海量图数据的存储和计算提供较好的支持。在实验中,将 RDF(Resource Description Framework)数据集导入到图引擎中。原数据集采用文本记录,大小超过 1 TB,拥有约 110 亿条三元组,导入到图引擎中共生成了 30 亿个顶点、50 亿条边。即使采用基于 BLOB 的紧凑内存存储结构,整个数据集仍然占用了 480 GB 左右的内存。

基于内存的底层存储对图的随机访问具有较好的支撑,因此图引擎在可用性上体现出较大优势。

图 5 是对 17 个不同类型的图顶点成员进行随机访问所花费的时间。横坐标从 P01 至 P17 分别代表不同的 17 种查询,纵坐标表示所花费时间。

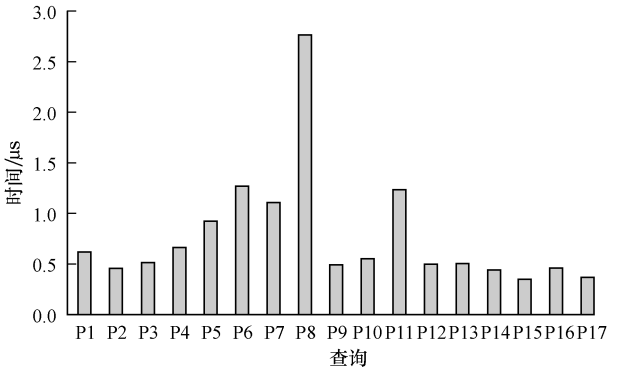

图 5 图顶点成员随机访问速度

借助于性能优异的底层存储引擎,上层的图引擎应用能够表现出更优异的性能。分别使用 8 台96 GB DDR3 内存的服务器存储上文提到的 RDF 数据集,并使用高速路由 InfiniteBand 对服务器进行互联,并使用子图匹配算法对图数据进行子图搜索。图 6 刻画了若干查询的响应速度,横坐标从 Q1 至 Q7 分别代表 7 个不同的插叙,纵坐标即完成查询的时间。

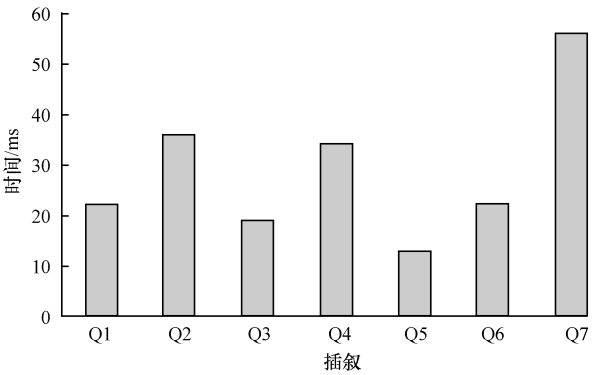

图 6 图查询响应速度

由图 5、图 6 可以看出,分布式内存云引擎可以为图引擎提供稳定高效的底层数据访问。在此之上,图引擎可以向客户端提供几十毫秒内的查询响应时间,有助于客户端高效快速地完成查询任务。

5 结束语

在大数据环境背景下,图数据处理面对上亿规模顶点的海量数据处理问题。传统的基于磁盘或者分布式文件系统的解决方案,在应对图应用的大量随机访问请求时存在性能瓶颈。本文基于分布式内存云提出一种新颖的图建模和存储方案,可对上层图应用提供灵活高效的数据访问接口。实验结果表明,本文方案能够实现海量规模的图数据处理。

参考文献

[1] Porta S, Crucitti P, Latora V. The Network Analysis of Urban Streets: A Dual Approach [J]. Physica A: Statistical Mechanics and Its Applications, 2006, 369(2): 853-866.

[2] Narin F. Evaluative Bibliometrics: The Use of Publication and Citation Analysis in the Evaluation of Scientific Activity[M]. Cherry Hill, USA: Computer Horizons, 1976.

[3] Lawrence P. The PageRank Citation Ranking: Bringing Order to the Web[R]. Stanford University, Technical Report: SIDL-WP-1999-0120, 1999.

[4] Wasserman S. Social Network Analysis: Methods and Applications[M]. Cambridge, UK: Cambridge University Press, 1994.

[5] Mislove A. Measurement and Analysis of Online Social Networks[C]//Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement. New York, USA: ACM Press, 2007: 29-42.

[6] Berners-Lee T, Hendler J, Lassila O. The Semantic Web [J]. Scientific American, 2001, 284(5): 28-37.

[7] Ramakrishnan R, Gehrke J. Database Management Systems[M]. [S. l.]: McGraw-Hill, 2000.

[8] Deelman E. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems [J]. Scientific Programming Journal, 2005, 13(3): 219-237.

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

[10] Ghemawat S, Howard G, Leung Shun-Tak. The Google File System [J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 29-43.

[11] Chang F. Bigtable: A Distributed Storage System for Structured Data [J]. ACM Transactions on Computer Systems, 2008, 26(2): 4-9.

[12] Melnik S. Dremel: Interactive Analysis of Web-scale Datasets[J]. Proceedings of the VLDB Endowment, 2010, 3(1/2): 330-339.

[13] Shao Bin, Wang Haixun, Li Yatao. Trinity: A Distributed Graph Engine on a Memory Cloud[C]//Proceedings of 2013 ACM SIGMOD International Conference on Management of Data. New York, USA: ACM Press, 2013: 505-516.

[14] 于 戈,谷 峪,鲍玉斌,等. 云计算环境下的大规模图数据处理技术[J]. 计算机学报, 2011, 34(10): 1753-1768.

编辑 陆燕菲