一种面向图数据的预装载缓存策略

黄 硕[1,2]，左 遥[1,2]，梁 英[1]，许洪波[1]，熊锦华[1]，王千博[1,2]，程学旗[1]

（1.中国科学院计算技术研究所，中国科学院网络数据科学与技术重点实验室，北京 100190；

2.中国科学院大学，北京 100190）

摘　要：真实世界中存在很多数据规模大且关联性强的图数据，对其分析和查询能够帮助我们获取巨大价值，而图缓存技术可以有效提高图数据的访问效率和查询效率。本文提出了一种面向大规模数据的图数据预装载缓存策略，采用“基于结点访问日志”和“大度数优先”的两种装载方法，缓存图数据边表的热数据。在图存储系统 GolaxyGDB 中设计了一个分布式图数据缓存框架，实现了缓存装载、访问、替换和一致性维护策略。实验表明，图数据预装载缓存策略能有效提高图数据复杂查询的效率，满足实际应用的在线访问需求。

关键词：预装载缓存策略；图数据；大度数优先；访问日志；Apache HBase

中图分类号：TP391　　**文献标识码：** A　　**文章编号：** 0372-2112（xxxx）xx-xxxx-xx

电子学报 URL：http://www.ejournal.org.cn　　**DOI 码：**10.3969/j.issn.0372-2112.xxxx.xx.xxx

A Preloaded Cache For Graph Data

Huang Shuo[1,2], Zuo Yao[1,2], Liang Ying[1], Xu Hongbo[1], Xiong Jinhua[1], Wang Qianbo[1,2], Cheng Xueqi[1]

(1. CAS Key Laboratory of Network Data Science and Technology，Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190, China

2. University of Chinese Academy of Sciences, Beijing, 100190, China)

Abstract: There exist many large scale and highly connected graphs in real world. Analysis and query of these graph data help us reveal the great value in them Caching is an efficient way to accelerate the visiting and querying of graph data. We propose a graph data cache preloading strategy with 2 methods, namely ‘log-based’ and ‘big degree first’，to cache frequently accessed data. We design a distributed cache framework for GolaxyGDB graph storage system, and describe the implementation of cache strategies in it. Experiments demonstrate the effectiveness of our cache preloading strategy for improving the efficiency of complex graph querying.

Key words: cache preloading strategy; graph data; big degree first; visit log; Apache HBase

1　引言

大数据时代数据规模爆炸性增长，从 TB 级别升至 PB、EB 甚至 ZB 级别。在数据规模大的同时，数据间的高相关性让数据的分析与挖掘变得更有意义。真实世界中存在很多数据规模大且关联性强的图数据，如社交网络、通讯网络、知识库网络等。据新浪微博上市 F-1 文件统计，新浪微博 2013 年 12 月的月活跃用户超过 1.2 亿，用户间关注边超过十亿（依据实际采集数据）；中国移动 2013 年年报显示，其用户数已超过 7.67 亿，以每个号码平均与 10 个号码联系的保守估计，用户间关系边达到百亿规模；在知识库网络中，全球著名开放知识库 Freebase 中三元组（Triple，相当于图中的边）数量达到 19 亿（截止至 2014 年 4 月）。

图数据具有数据量大、查询复杂等特点，为多维度多约束的复杂查询的快速响应带来了挑战。由于单机内存和磁盘容量的限制，大规模的图数据需要使用多机分布式架构来存储，且数据无法全部装载进内存，数据访问中避免不了磁盘访问，磁盘较慢的读取速度和分布式环境下的网络传输开销为查询快速响应带来挑战。缓存技术是降低查询响应时间的有效手段。

基金项目：国家重点基础研究发展计划(973)项目 (2012CB316303,2013CB329602)；国家自然科学基金重点项目(61232010)；国家自然科学基金面上项目(61173064)；国家科技支撑计划项目(2012BAH39B04)。

图数据的缓存加载、替换和一致性维护是图数据复杂查询中的重要环节，高效的缓存策略可以加速查询的执行速度，减少用户延迟。本文在我们研发的分布式图存储系统 GolaxyGDB[1]基础上，提出并设计实现了一种面向图数据的预装载缓存技术，包括“基于结点访问日志”和“大度数优先”两种图数据缓存装载方法和策略，加速复杂关系查询的执行速度，支持在线模式的图数据应用。

2 相关研究

缓存技术在计算机系统中被广泛应用，在 CPU、数据库、Internet 服务等领域发挥了重要的作用。缓存技术能够充分利用数据访问的局部性，提高系统性能、降低查询用时、减少用户延迟[2]。分布式缓存将数据分布到多个缓存服务节点，获得更大的缓存容量；在内存中统一管理数据，对外提供统一的访问接口[3]，具有更好的扩展性和更高的可用性。

命中率高的缓存策略可以有效地减少响应时间和系统负载。以搜索引擎查询为例，缓存资源装载有静态（离线，基于查询日志）和动态（在线，实时变换）方式[4]，Fagni 等人提出的静态与动态结合的缓存策略有效地利用了两种方式的优点，在实际测试中取得了较高的命中率[5]。Baeze-Yate 等人采用缓存热点查询词的倒排表的方式提高缓存的命中率[6-7]。此人还通过阻止用户查询中不活跃的查询结果进入到缓存提高命中率[8]。

Memcached[9]是一个开源分布式内存对象缓存系统，使用 Key-Value 数据模型[10]，支持对于小块数据的非持久化存储和访问，在 Facebook、Twitter 等大规模网站中发挥了重要的作用[11]。如在 Facebook TAO 图存储系统中，使用 Memcached 作为基本元素构建了分布式缓存层，极大的提升了系统的读性能[12-13]。Ehcache 是 TerraCotta 公司的一款开源 Java 分布式缓存产品，提供了 JSR107-JCACHE 规范的完整实现，能够在包括 Java EE 在内的各种 Java 开发场景下发挥作用[14]。Coherence 是 Oracle 公司的一款内存数据网格产品。Coherence 的核心是一个高可用、可扩展的缓存引擎，具有无单独故障的架构和自动 failover 功能。

上述缓存策略与分布式缓存系统，面向的是较为通用的使用场景，缺少对图数据的存储形式和访问模式等方面特点的利用，可以作为图缓存的借鉴或者基础，但不适合直接应用到图存储系统中。因此，本文将从图数据的特点出发，设计适应复杂图查询需求的分布式图缓存，加速图存储系统的查询处理。

Apache Hadoop[15-16]是当前最成熟和最活跃的开源大数据处理平台，它包括了支持海量数据存储的分布式文件系统 HDFS（Hadoop File System）和分布式计算框架 MapReduce。Apache HBase 是基于 HDFS 的一个开源分布式数据库，提供对半结构化数据的存储和实时随机访问。HBase 具有高可靠、易扩展、自动 failover 等特点，在 Facebook、淘宝等多个互联网公司得到成功应用，是构建大数据在线应用的重要软件基础。

本文将基于分布式图缓存系统 GolaxyGDB，研究图缓存方法和策略，在 Apache Hadoop 分布式框架下，以 Apache HBase 作为底层存储，实现一种面向图数据的预装载缓存技术。

3 图数据预装载缓存技术

图缓存指的是在图存储系统中，利用支持高效随机访问的内存来存储部分图数据，减少访问时间，提高系统整体性能。本文提出的图数据预装载缓存技术利用对图数据访问的局部性，通过对访问情况和图数据特点进行分析，主动将部分图数据预先装入到缓存中，使得后续的访问能够获得更好的性能。

3.1 基本概念

为更加严谨的描述本文提出的图缓存技术，首先对其中用到的术语进行定义[1]。

定义 1 图（G）：在本文的图数据模型中，将一个图 G 形式化的定义为一个五元组：

$$G = < gid, V, L, T, \mu > \quad (1)$$

式（1）中：

（1）gid 是图实例的一个标记（Graph ID）；

（2）V 是图中所有结点（Node）的集合，即 $V = \{node\}$，其中每个 $node$ 定义为：

$$node = < nid, ntype, name, \{property\}, \{ref\} > \quad (2)$$

（3）L 是图中所有关系（Link 或者 Relation）的集合，即 $L = \{relation\}$，其中每个 $relation$ 定义为：

$$relation = < node_1, node_2, rtype, weight, timestamp, \{ref\} > \quad (3)$$

（4）T 是图中允许的所有结点类型的集合，是 V

中所有类型的集合T_N的一个超集，即$T \supseteq T_N$，其中：

$$T_N = \{type \mid node \in V \wedge type = node[ntype]\} \quad (4)$$

（5）μ是图中所有时间信息的基本单位（Time Unit）。

定义 2 边（Edge）：图G中从$node_A$到$node_B$的“边”$edge(node_A, node_B, \varphi_R)$是两点间该方向上所有满足给定关系约束函数$\varphi_R: \{relation\} \to \{true, false\}$的关系的集合，定义为式（5）：

$$\begin{aligned} & edge(node_A, node_B, \varphi_R) \\ & = \{rel \mid rel[node_1] \\ & = node_A \wedge rel[node_2] \\ & = node_B \wedge \varphi_R(rel) = true\} \end{aligned} \quad (5)$$

当$\forall rel, \varphi_R(rel) = true$成立时，将$edge(node_A, node_B, \varphi_R)$记为$edge(node_A, node_B)$，即从$node_A$到$node_B$的所有关系组成的边。

在不引起歧义的前提下，允许将$edge(node_A, node_B, \varphi_R)$简写为$edge$。

定义 3 边表（EdgeList）：图G中从$node$出发的“边表”$edgelist(node, \varphi_E, \varphi_R)$是满足给定边约束函数$\varphi_E: \{edge\} \to \{true, false\}$的所有“边”$edge(node, n_2, \varphi_R)$的集合，定义为式（6）：

$$\begin{aligned} & edgelist(node, \varphi_E, \varphi_R) \\ & = \{edge(n_1, n_2, \varphi_R) \mid n_1 \\ & = node \wedge \varphi_E(edge) \\ & = true\} \end{aligned} \quad (6)$$

在不引起歧义的前提下，允许将$edgelist(node, \varphi_E, \varphi_R)$简写为$edgelist$。

预装载缓存中的缓存项粒度是一个边表。进入预装载缓存中的关系数据被组织成边表集合的形式，预装载缓存中内容可以写为式（7）：

$$\{edgelist(node, \varphi_E = true, \varphi_R = true) \mid node \in SelectedNodeSet\} \quad (7)$$

$SelectedNodeSet$是所有被选中加载进缓存的边表的起始结点的集合。对于每个$node \in SelectedNodeSet$，从该点出发的所有关系形成的边表$edgelist(node, true, true)$（不含数据源引用和时间信息）进入缓存。

3.2 缓存数据的装载方法

3.2.1 基于结点访问日志的装载方法

对于每个结点，由于其对应的边表数据量大小不同，带来的缓存容量开销（本文使用结点对于边表载入缓存后的内存占用作为其开销，这一开销可以通过结点对应的边表数据直接计算出来）不同，使得不同结点装入缓存所可能带来的性能收益（本文使用结点的历史访问次数作为其收益）不同。通过上述定义的每个结点的“开销”和“收益”，可以将缓存的装载问题（即$SelectedNodeSet$的构造问题），建模为一个 0-1 背包问题（0-1 Knapsack）。

定义 4 缓存装载问题：在图G中，已知结点集合$G[V]$上分别定义有收益函数$nGain: G[V] \to N$和开销函数$nCost: G[V] \to N$；对于给定的缓存容量CM（**C**apacity **M**ax），求结点子集$SelectedNodeSet \subseteq G[V]$，使得满足容量限制$\sum_{n \in SelectedNodeSet} nCost(n) \le MC$且总收益$totalGain = \sum_{n \in SelectedNodeSet} nGain(n)$最大。

在实际数据规模下，上述 0-1 背包问题不能够使用动态规划来求最优解。依据定义 4 背包总数为$n = card(G[V])$，动态规划的时间复杂度最好为$O(n \cdot CM)$，空间复杂度最好为$O(CM)$。由于实际图数据中，$card(G[V])$规模可达十亿级别，CM按字节计算可到若干GB（即百亿级别）。显然，在实际系统规模下，利用动态规划求得最优解的时间复杂度无法接受。

因此，使用一个贪心算法来近似的求解上述 0-1 背包问题。理论表明，0-1 背包问题存在一个贪心算法来求近似解，且可证明近似比为 1/2（即求得的结果的总收益大于最优解总收益的 1/2）。该算法描述如算法 1 所示。

算法 1 缓存装载问题贪心算法

输入： $G[V]$ //图 G 的结点集合

$nGain: G[V] \to N$ //图 G 结点的收益函数

$nCost: G[V] \to N$ //图 G 结点的开销函数

CM //缓存容量限制

输出： $SelectedNodeSet$ //缓存装载的结点集合

1. $SortedNodeList = SortByRatio(G[V])$ // 按照 $nGain(n_i)/nCost(n_i)$ 的降序对 $n_i \in G[V]$ 各个结点进行排序
2. $curCapacity \leftarrow 0$, $SelectedNodeSet \leftarrow \varnothing$, $index \leftarrow 0$, $curGain \leftarrow 0$
3. **WHILE** $index \le SortedNodeList.Size$ **DO**
4. $node = SortedNodeList[index]$;
5. **IF** $curCapacity + nCost(node) \le CM$ **THEN**
6. $AddToSet(node, SelectedNodeSet)$; // $node$ 加入 $SelectedNodeSet$ 集合
7. $curCapacity += nCost(node)$;
8. $curGain += nGain(node)$
9. $index$ ++;
10. **ELSE** break;
11. **END IF**
12. **END WHILE**
13. **IF** $index \le SortedNodeList.Size$ **THEN**
14. $node = SortedNodeList[index]$
15. **IF** $nGain(node) > curGain$ **THEN**
16. $SelectedNodeSet \leftarrow \{node\}$;
17. **END IF**
18. **END IF**
19. **RETURN** $SelectedNodeSet$;

上述算法的时间复杂度主要为对结点排序的复杂度，为 $O(n\log n)$（其中 $n = card(G[V])$），空间复杂度为 $O(n)$。实现中通过利用 MapReduce[17]等并行处理技术，能够在可接受时间范围内求解。利用实现中缓存的数据布局方法，可以计算每个结点的存储开销，即 $nCost: G[V] \to N$。

3.2.2 大度数结点优先的装载方法

当结点访问日志记录较少时，大规模图中的大部分点的访问次数均为 0 或非常小的值，这使得访问次数仅能够反映用户访问兴趣，而不能够包含充分的拓扑信息。为此，采用“大度数结点优先”方法来进行弥补，即预装载一部分度数较大的结点所对应的边表。直观上看，度数较大的结点对应的边表，更有可能在多个不同的路径子图查询（GolaxyGDB 提供的一种查询类型，即两点间一定度数范围内所有路径形成的子图）结果中出现，反映了图的拓扑信息。

为了与基于结点访问日志的装载方法进行融合，实际中将总的缓存容量限制 $CMax$ 分为两部分 CM_1 和 CM_2，分别作为“大度数结点优先”方法的容量限制和算法 1 的内存容量限制。当访问日志充足时，可适量减少 CM_1 的占比。

3.3 图缓存策略

3.3.1 缓存数据装载策略

在图数据预装载缓存中，装载策略包括基于节点访问日志的装载策略和大度数节点优先的装载策略。“基于结点访问日志”的装载策略通过基于节点访问日志的方法实现，其缓存命中效果主要依赖于用户的实际访问模式，如果用户频繁访问某些结点对应的边表，这部分点在访问日志中会频繁出现，从而对应的边表被装载进入缓存，后续对它们的访问将命中。“大度数结点优先”装载策略通过大度数节点优先的装载方法实现，能够反映图的拓扑信息，倾向于选择在图中具有更“重要”地位的度数大的结点所对应的边表。

在实现中，缓存数据的装载不是在查询未命中时触发，而是定时计算和选取需要缓存的数据，并由缓存层主动将数据装载到缓存中。

3.3.2 缓存访问策略

预装载缓存中包含了大量从不同结点出发的边表数据，在关系图查询（GolaxyGDB 提供的一种查询类型，即与给定结点直接关联且满足给定约束的所有结点和关系形成的关系图）、路径子图查询等各种复杂图查询的执行过程中，查询处理模块会产生一个或者多个对边表数据的访问。这些访问首先被发送到预装载缓存中，如果访问涉及的数据已经在缓存中，则命中并返回数据；否则，查询处理模块将去底层存储中获取所需数据。在整个缓存访问过程中，不会触发缓存数据的装载，也不会导致缓存项的替换。在访

问预装载缓存的过程中，被访问的结点会被记录到结点访问日志中。

3.3.3 缓存替换策略

预装载缓存中的内容通过定时更新来实现替换，每隔一段时间，重新求解缓存装载问题。将得到的装载点集合与前一次的结果进行比对，从而知道哪些结点需要从缓存中删除，哪些需要新加载进缓存，将相应的消息发送给预装载缓存。

3.3.4 数据一致性维护策略

本文通过更改通知的方式来维护预装载缓存与底层存储中的数据一致性。当发生数据更改时，向预装载缓存中发送一个相应结点的失效消息（cache invalidation message），缓存收到消息后将该结点对应的边表从缓存中删除。当进行大批量数据更改时，失效消息允许批量发送。

4 图数据缓存的设计与实现

4.1 图缓存的总体架构

本文实现了分布式的图数据预装载缓存，预装载缓存与上层操作以及下层 HBase 关系如图 1 所示。

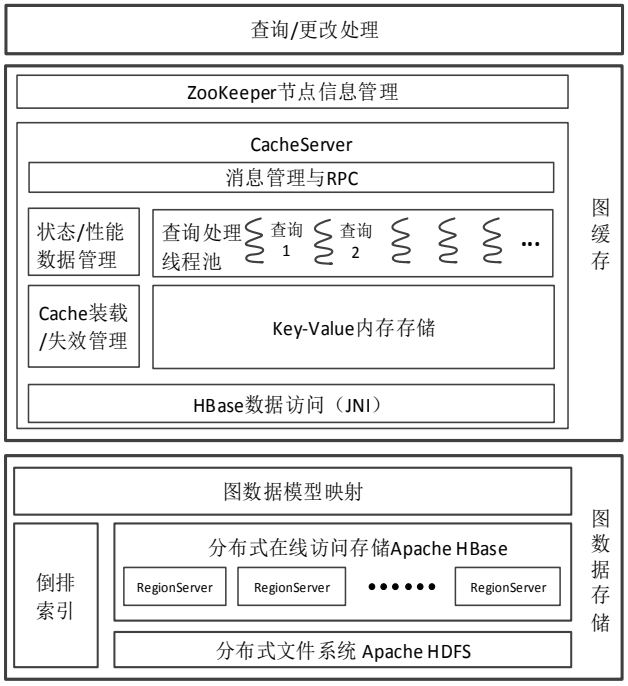

图 1 图缓存总体框架

分布式预装载缓存可以支持上层的查询、更改操作以及下层的 HBase 图数据存储访问。它包括了处于多个机器上的多个独立 CacheServer 进程（图 1 中以一个完整的 CacheServer 为代表），负责节点信息和状态管理的 Zookeeper。为了与图数据中的“结点”进行区分，本节使用“节点”指代分布式缓存中的 CacheServer。

CacheServer 的核心是一个 Key-Value 内部存储，用于存储被装载的图数据（见 4.2 节）； Key-Value 存储接受查询处理线程池中线程发起的访问，这些线程包含了利用缓存数据处理查询的逻辑；CacheServer 还包含了进行数据统计（如命中率、缓存结点数等）和处理 Cache 装载、失效消息的模块，并通过一个到 Zookeeper 的连接报告自身健康状态。

值得指出的是，在底层的 HBase 中同样存在着数据缓存，CacheServer 相比于 HBase 中的缓存有以下的优点：首先，CacheServer 按照图数据特点和查询要求设计内存布局，相比 HBase 中 HFile 块缓存能更充分的利用内存和适应查询；其次，CacheServer 中能够使用从图数据特点出发的装载策略，实现对缓存更精确的控制。

分布式预装载缓存中，对图数据进行随机划分。CacheServer 的总数量 CSN （**C**ache **S**erver **N**umber）在缓存启用时静态指定，对于被加载到预装载缓存中的所有结点，按照结点的 ID 通过一个全局唯一且固定的哈希函数 $nhash:\{nodeid\}\rightarrow\{k\,|\,k\in N\wedge k<CSN\}$，将其映射到一个 CacheServer 之上。其他模块访问预装载缓存时，通过 Zookeeper 获得 CSN 值和每个节点的网络地址、可用性状态等信息，进而可以在模块本地利用 $nhash$ 函数将对某个图结点数据的请求，路由到其所在的 CacheServer，实现缓存数据访问。

CacheServer 的数量和物理位置独立于底层 HBase 存储，使得缓存和持久存储能够独立扩展，可以依据实际需求部署合适数量的 HBase 节点和缓存节点，使用不同特性的机器（大内存、大磁盘容量）。

4.2 边表数据的存储

每个缓存项存储的是边表数据，被组织成内存中的 Key-Value 形式。在 Key-Value 内存存储中，为基本的关系数据构建了索引，弥补了底层 HBase 较弱的索引能力，从而加快图数据访问速度。每个结点出发的边表信息被组织成多个 Key-Value，包括了基本关系数据和之上的索引数据两部分。图 2 是一个实例，展示了与一个从 A 出发的边表相关的所有 Key-Value。关系数据的 Key 格式为“起始结点-0-结点类型-关系类型”，Value 包含了与 A 存在相应关系的相应类型的结点，以及相应的关系上权重。索引部分数据包含

三种 Key 格式“起始结点-3”、“起始结点-2-结点类型”、“起始结点-1-关系类型”，Value 为满足相应条件的所有 Key。索引的存在，使得能够在 Key-Value 存储中快速的定位满足给定结点类型、关系类型约束的数据。

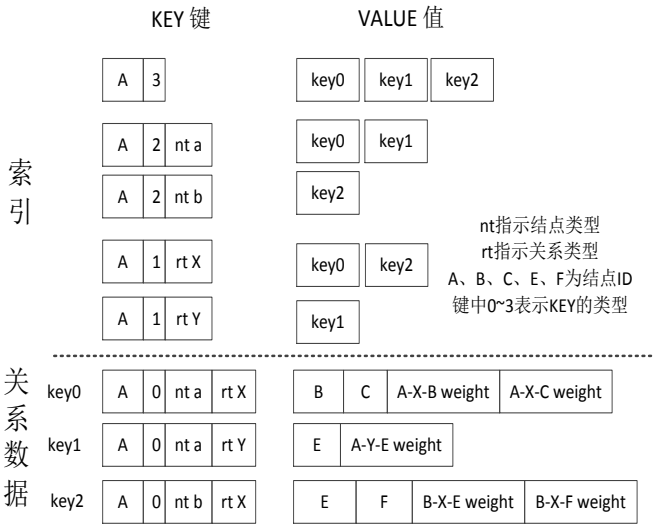

图 2 边表数据的内存 Key-Value 形式

依据边表的 Key-Value 格式和数据序列化方式，对图数据进行离线统计，能够较为准确的计算给定的边表的内存占用量，从而给出本文实现方式下定义 4 中的结点内存开销函数 $nCost:G[V]\to N$。

4.3 缓存装载算法的实现

在算法 1 中，首先要确定结点的收益和开销函数的计算方式。上文已经介绍了内存开销函数 $nCost:G[V]\to N$ 的计算方式。结点访问次数函数 $nGain:G[V]\to N$ 通过在查询处理模块中收集结点访问日志来计算。对于任意结点的访问，会产生一条“结点 id，访问时间”的日志记录，通过分析日志即可得到每个结点在给定时间范围内的访问次数。由于存在访问次数为 0 的图结点，对 $nGain:G[V]\to N$ 进行 Add-Delta 平滑，即把访问次数加上一个常数 $\lambda,\lambda\in(0,1]$ 作为 $nGain$ 的取值。

实现算法 1 的关键是步骤 1 中对结点按照 $nGain(n_i)/nCost(n_i)$ 降序进行排序。本文采用 MapReduce 任务结合 Apache Hive 来实现相关计算和排序。首先，使用一个 MapReduce 任务根据 HBase 中图结点的数据计算每个结点的缓存内存开销，结果对应 Hive 数据表 id_memcost(node_id, memcost)。然后，将访问日志数据对应到 Hive 数据表 access_log(node_id, timestamp)。设每个结点的 $nGain(n_i)/nCost(n_i)$ 值存放于数据表 node_ratio (node_id, ratio, memcost, count)，则算法 1 步骤 1 用下面的 Hive QL 语句实现，得到 node_ratio 表数据（平滑参数 $\lambda=0.5$，访问时间戳在 1397997232 之后）：

```
INSERT OVERWRITE TABLE node_ratio
SELECT A.id,(case when B.count is NULL then 0.5 else 1.0*B.count+0.5 end)/A.memcost AS ratio, A.memcost, (case when B.count is NULL then 0.5 else 1.0*B.count+0.5 end) AS count
FROM id_memcost A
LEFT OUTER JOIN(
  SELECT node_id,count(*) AS count FROM access_log
  WHERE ts>1380394342 GROUP BY node_id
) B
ON A.id=B.node_id
ORDER BY ratio DESC;
```

得到 node_ratio 表数据后，读取其内容执行算法 1 的后续步骤，即可最终得到需要加载进缓存的结点 ID 的集合。通过与缓存上一次加载的结点集合进行比对，即可为 CacheServer 生成相应的结点加载和删除消息。

5 实验与效果评估

本章对上述存储系统以及其中的关键技术点进行实验评价和分析，验证其有效性。

5.1 实验概述

实验硬件环境包含一个基于 10 个结点的集群，集群中 8 台机器作为 Hadoop 和 HBase 的数据节点（DataNode 和 RegionServer），另外两台放置 Hadoop NameNode、MapReduce JobTracker、HBase Master 等。10 个结点的配置均为 2 颗 AMD Opteron 6128（8 Core, 2.0 GHz）、48GB 内存、1TB 硬盘、1Gbps Ethernet 网卡。集群使用的操作系统为 Linux Centos 5.6，文件系统为 Linux Ext3，开发环境为 JDK1.6.0_31，Hadoop 1.2.0，HBase 0.94.6，Hive 0.12。

本文用于分析或实验的图数据实例，包括从实际应用中获取的两组实际图数据 Weibo-ER 和 Web-Log。

Weibo-ER 是新浪微博用户数据得到的关系图，包含了微博用户、微博用户的地点、机构信息三类结点，以及用户间相互关注关系等关系类型。其中结点数量为 122,614,009，有向边数量为 1,818,311,689，有向关系数量为 1,818,311,689，结点出边数量的分布是一个长尾分布。Web-Log 是通过互联网搜索引擎收集的信息，包括时间、人、机构、地点、email、电话六类信息，保存在 GolaxyGDB 的 web 通道中，结点数量为 181,403,969，有向边数量为 2,203,384,350，有向关系数量为 2,203,384,350。

通过命中率和命中时的加速比的实验来评价图缓存的性能。在预装载缓存中，使用了“基于结点访问日志”和“大度数结点优先”两种缓存装载策略，将分别对这两种缓存装载策略进行实验评估；访问速度是用关系图查询进行实验评估。目前系统并未大规模投入使用，因此访问日志数据积累较少,我们使用搜狗公开的搜索日志[18]来模拟对图数据库的访问。

5.2 大度数优先策略性能分析

在 Weibo-ER 中随机选取 5000 个结点，对于每个结点 n_1，随机获取它的一个相关结点 n_2，发起 n_1 和 n_2 间的最大路径长度为 3 的路径子图查询。因此，一共发起了 5000 个最大路径长度为 3 的路径子图查询。通过分析产生的结点访问日志可知，这些查询共访问了 Weibo-ER 中 1334928 个边表，其中每个边表的访问次数从 1 到 11。

5.2.1 结点访问频率分析

由图 3 和图 4 可知：(1) 访问次数越多的边表越少（图 3），呈指数下降。这表明少部分“重要”的边表会在各个路径子图查询中频繁被访问，因此我们能通过缓存一部分结点对应的边表取得更好的命中率。(2) 访问次数越多的边表对应的结点，平均度数越高（图 4）。这表明，度数较大的点对应的边表有可能是被访问次数多的边表。进一步计算数据中各个边表的访问次数和对应结点的度数之间的 Pearson 相关系数为 0.477，表明边表对应结点的度数与边表被访问次数存在一定的正相关。

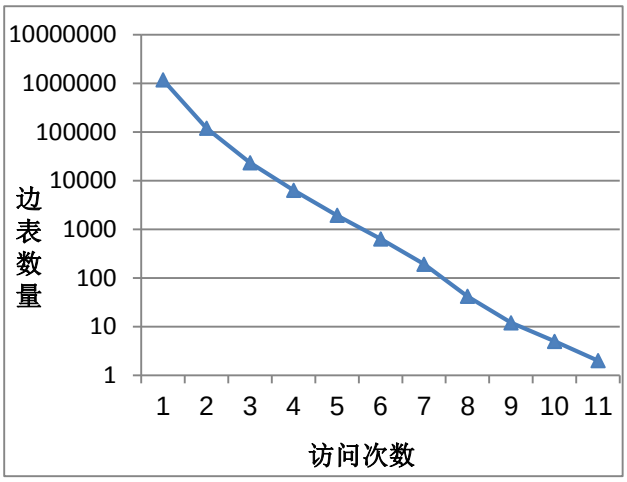

图 3 不同访问次数的边表数量变化

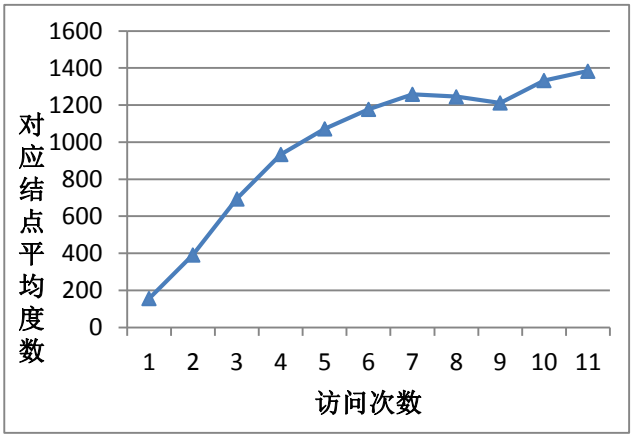

图 4 不同访问次数的边表对应结点的平均度数

表 1 中是不同度数范围内结点对应的边表被访问的情况。可见，相比于度数较小(<100)的结点，Weibo-ER 图中度数较大(>800)的结点对应的边表有更大的比例被访问到，且其中有更大的比例被访问超过一次。这进一步的表明，度数较大的结点对应的边表有更大的可能性在多个路径子图查询中被频繁访问。

表 1 不同度数范围内结点对应的边表的被访问情况

数据项	度数小于 100 的结点对应的边表	度数大于 800 的结点对应的边表
边表总数	118,828,199	79,407
被访问到的边表数量	601,964	54,542
被访问到的边表比例	0.51%	68.69%
被访问超过 1 次的边表数量	17,618	31,510
被访问超过 1 次的边表比例	0.015%	39.68%

综合上面的分析可知，“大度数结点优先”的缓存中边表对应结点的选择策略，对于加速路径子图查询具有意义。

5.2.2 缓存命中分析

下面，将“大度数结点优先”的策略与“随机选择”的边表装载策略进行对比，比较不同策略下路径子图查询中边表访问的缓存命中次数。

在 Weibo-ER 中随机选择了 10000 个对应结点度数在 800 到 5000 之间的边表，作为“大度数结点优先”策略选择的边表，其对应的结点集合记为 *SetA*，这些边表的内存总开销为 219801258。保持内存总开销不变，随机选取了 309987 个边表（对应结点的度数不限），作为“随机选取”策略的结果，其对应的结点集合记为 *SetB*。两个边表集合对应的结点集合的度数分布情况如表 2。

表 2 大度数优先装载与随机装载策略选取的边表所对应的结点集合分析（固定总内存开销）

边表对应的结点的集合	结点数量	占 Weibo-ER 点总数的比例	结点度数			结点度数百分位值			
			AVG	MIN	MAX	50%	75%	90%	99.9%
SetA（大度数）	10000	0.008%	1129.2	801	4942	1047	1259	1492	4459
SetB（随机）	309987	0.253%	14.0	1	29639	2	6	32	640

可见 *SetB* 结点数量较多，平均度数更小，但是从度数分位值可知也包含了部分度数较大的结点；*SetA* 点数量较少，仅占总结点数的 0.008%，平均度数较大。

在不同的装载情况下，分别再次执行上文中的 5000 个随机路径查询，得到缓存中边表的命中次数如图 5 所示。

可见，大度数优先装载在同样的内存开销下，能够取得三倍于随机装载的命中次数。因此，“大度数结点优先”装载策略对于路径子图查询具有有效性。

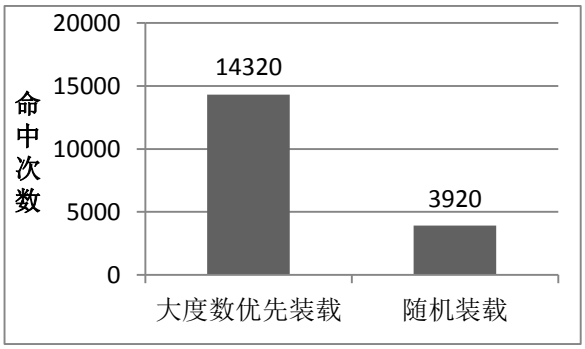

图 5 大度数优先装载与随机装载策略下的命中次数比较

5.2.3 缓存访问性能分析

预装载缓存能够加速获取一个点的满足给定约束的相关关系。在 Weibo-ER 中随机选取 10000 个结点，分别执行关系约束为“所有关系”和“仅相互关注关系”的两个关系图查询，在预装载缓存命中和未命中时，其用时的平均值如图 6 所示。

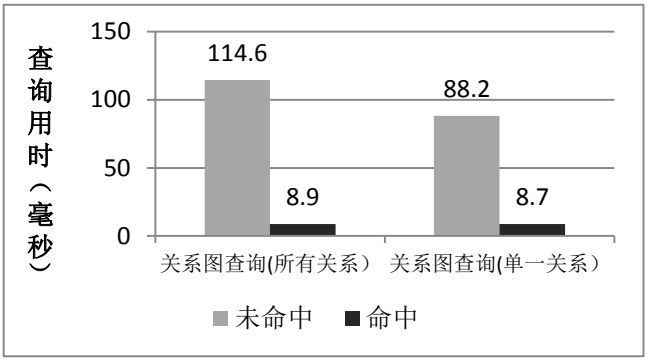

图 6 预装载缓存命中/未命中用时比较

比较图 6 中每组实验内命中/未命中的用时可知，预装载缓存在命中时能够明显加速图查询的执行，用时不到未命中时用时的 10%；比较两组实验可知，无论获取的关系数量多少，命中时的用时都保持在很短时间内。因此，预装载缓存命中时能带来明显的性能提升。

5.3 基于节点访问日志策略性能分析

实验数据来源于搜狗公开的搜索日志。原始数据共包含 5172943 条搜索日志。首先我们对每一条搜索字符串进行了命名实体识别，并剔除没有被 GolaxyGDB 中 web 通道数据所包含的命名实体。预处理后得到的数据包含 1423152 次访问，每一次访问我们都认为是对 web 图中相应命名实体结点的访问，其中不重复的命名实体共有 144521 个，单个命名实体最多被访问 47643 次。

我们将 1423152 次访问分成两部分，按照访问时间划分，前三分之二共 948768 次作为已知数据，用来决定要加载进缓存的节点，后三分之一共 474384 条作为测试数据，用来检验缓存加载策略的有效性。

对比随机装载策略和基于结点访问日志的装载策略。首先我们随机从已知访问中选择了 10000 个结点，其对应的结点集合记为 *SetA*，这些结点内存总开销为 80865308，保持内存总开销不变，按照基于节点访问日志的装载算法，我们选择了 17821 个结点，其对应的结点集合记为 *SetB*。两个结点集合的度数分布情况如表 3。

表 3 基于结点访问日志装载与随机装载策略选取的边表所对应的结点集合分析（固定总内存开销）

边表对应的结点的集合	结点数量	结点访问次数			结点访问次数百分位值			
		AVG	MIN	MAX	50%	75%	90%	99.9%
SetA（随机）	10000	30.2	1	12125	4	12	30	5186
SetB（访问）	17821	63.3	1	25403	3	7	25	8972

根据结点访问日志装载的结点平均访问次数更多，并且和大度数优先策略不同的是，在保持内存开销不变的情况下，根据访问日志选出的结点数量反而比随机选择更多。这说明，访问频繁的结点对应的边表未必占用更多的内存，一个被频繁访问的结点可能在图数据库中只有很少的结点与之相连，因此能够与大度数优先策略互补。

在不同的装载情况下，分别对上文测试数据中得 474384 次访问对应的结点进行“所有关系”查询，得到缓存中边表的命中次数如图 7 所示。

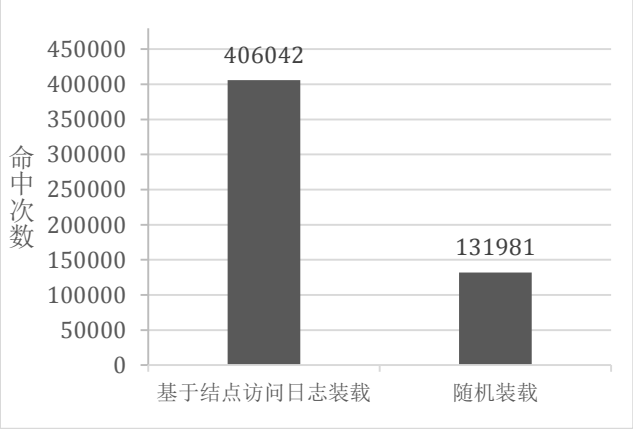

图 7 基于结点访问日志装载与随机装载策略下的命中次数比较

由图 7 观察到，基于结点访问日志装载，命中 406042 次，随机装载，命中 131981 次。可见，基于结点访问日志装载在同样的内存开销下，也能够取得三倍于随机装载的命中次数。因此，“基于结点访问日志”装载策略对于路径子图查询具有有效性。

我们以当前的内存开销作为 100%，测试了缓存占用不同内存开销百分比时的命中率和平均查询用时，查询同样针对“所有关系”。对于不同的内存开销，我们都重新根据缓存加载策略计算需要加载的结点，并在测试数据上进行测试。结果如图 8，图 9 所示。

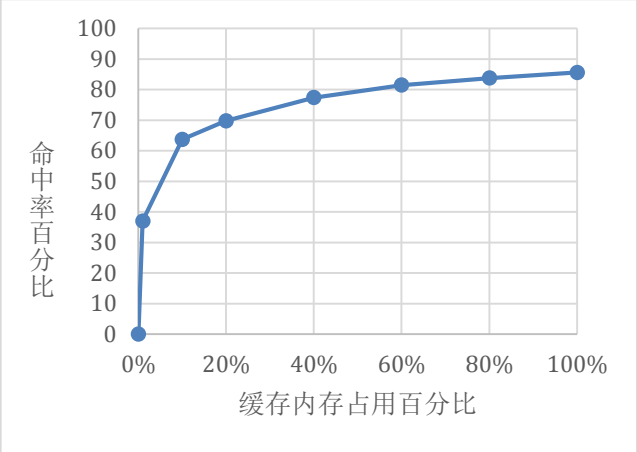

图 8 缓存内存占用变化时的缓存命中率曲线

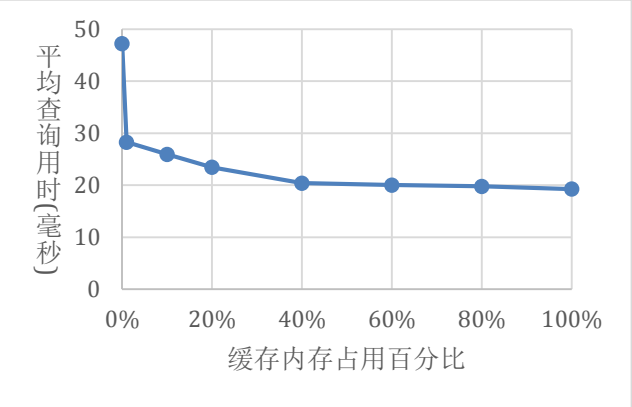

图 9 缓存内存占用变化时平均查询用时曲线

观察缓存命中率曲线和平均查询用时曲线可以得到以下结论，基于结点访问日志的缓存加载策略在占用内存较小时就能取得较好地效果，随着内存占用的增大，命中率的提升逐渐放缓，平均查询用时的减小速度也逐渐放缓。

6 结束语

大规模图数据的缓存机制是复杂关系查询的支撑，本文提出了一种分布式图数据预装载缓存机制，采用“基于结点访问日志”和“大度数优先”的两种互为补充的装载方法，缓存图数据边表的热数据。在图存储系统 GolaxyGDB 中设计了一个分布式图数据缓存框架，实现了缓存装载、访问、替换和一致性维护策略。有效提高图数据复杂查询的效率，满足实际应用的在线访问需求。

参考文献

[1] 黄硕，梁英，许洪波，等.一种支持复杂关系查询的图存储系统[C]. 第 20 届全国信息存储技术学术会议，2014.9. Huang S, Liang Y, Xu H B, et al. A graph data store with complex query support[C]. Proceedings of The 20th National Conference of Information Storage, 2014.9. (in Chinese)

[2] 王必尧, 王劲林, 吴刚, 等. 一种应用于分布式缓存系统中的缓存部署算法[J]. 小型微型计算机系统，2012, 33(008): 1645-1649. Wang B R, Wang J L, et al. Cache deployment algorithm in distributed caching system[J]. Journal of Chinese Computer Systems, 2012, 33(008): 1645-1649. (in Chinese)

[3] 秦秀磊, 张文博, 魏峻, 等. 云计算环境下分布式缓存技术的现状与挑战[J]. 软件学报, 2013, 24(1): 50-66. Qin X L, Zhang W B, Wei J, et al. Progress and challenges of distributed caching techniques in cloud computing[J]. Journal of Software, 2013, 24(1): 50-66. (in Chinese)

[4] 马宏远, 王斌. 基于日志分析的搜索引擎查询结果缓存研究[J]. 计算机研究与发展, 2012, 1. Ma H Y, Wang B. Search engine query results caching based on log analysis[J]. Journal of Computer Research and Development, 2012, 1. (in Chinese)

[5] Fagni T, Perego R, Silvestri F, et al. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data[J]. ACM Transactions on Information Systems (TOIS), 2006, 24(1): 51-78.

[6] Baeza-Yates R, Gionis A, Junqueira F, et al. The impact of caching on search engines[C]. Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2007: 183-190.

[7] Baeza-Yates R, Gionis A, Junqueira F P, et al. Design trade-offs for search engine caching[J]. ACM Transactions on the Web (TWEB), 2008, 2(4): 20.

[8] Baeza-Yate R, Junqueira F, Plachouras V, et al. Admission policies for caches of search engine results[C]. String Processing and Information Retrieval. Springer Berlin Heidelberg, 2007: 74-85.

[9] Fitzpatrick B. Distributed caching with memcached[J]. Linux journal, 2004, 2004(124): 5.

[10] Atikoglu B, Xu Y, Frachtenberg E, et al. Workload analysis of a large-scale key-value store[C]. ACM SIGMETRICS Performance Evaluation Review. ACM, 2012, 40(1): 53-64.

[11] Nishtala R, Fugal H, Grimm S, et al. Scaling Memcache at Facebook[C]//nsdi. 2013: 385-398.

[12] Bronson N, Amsden Z, Cabrera G, et al. TAO: Facebook's Distributed Data Store for the Social Graph[C]. USENIX Annual Technical Conference. 2013: 49-60.

[13] Venkataramani V, Amsden Z, Bronson N, et al. Tao: how facebook serves the social graph[C]. Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, 2012: 791-792.

[14] Jing W, Fan R. The research of Hibernate cache technique and application of EhCache component[C]. Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on. IEEE, 2011: 160-162.

[15] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system[C]. Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE, 2010: 1-10.

[16] Taylor R C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics[J]. BMC bioinformatics, 2010, 11(Suppl 12): S1.

[17] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[18] Sougou. 2012. 用户查询日志. Retrieved August 15, 2014 (http://www.sogou.com/labs/dl/q.html).

作者简介

黄　硕，1990 年出生. 2014 年在中国科学院计算技术研究所取得硕士学位. 主要研究兴趣包括大数据，中间件，服务计算等.
Email: hshuocn@gmail.com.

左　遥　男，1991 年 1 月出生. 2013 年毕业于清华大学，现为中国科学院计算技术研究所硕士研究生，从事大数据方向的研究.
Email: laike9m@gmail.com

梁　英　女，1962 年出生. 中国科学院计算技术研究所副教授，主要研究兴趣包括大数据，中间件，服务计算等.
Email: liangy@ict.ac.cn