DOI:10.7544/issn1000-1239.2017.********
卷(期):起止页,年

分布式大图处理系统中的索引设计与实现

陈修司[1,2] 郑淇木[1,2]

[1]（北京大学信息科学技术学院 北京 100871）

[2]（高可信软件技术教育部重点实验室(北京大学) 北京 100871）

（xiusichen@pku.edu.cn）

The Design and Implementation of Index on Distributed Graph Processing System

Chen Xiusi[1,2], and Zheng Qimu[1,2]

[1]（*School of Electronics Engineering and Computer Science*, *Peking University*, *Beijing* 100871）

[2]（*Key Laboratory of High Confidence Software Technologies*（*Peking University*）, *Ministry of Education*, *Beijing* 100871）

Abstract With the rapid growth of technologies like web, social network, and location based service, graph data exist in a variety of applications. Systems like Google Pregel and Apache Giraph have been proposed for efficient large-scale graph data analysis. However, these graph processing systems often suffer from significant synchronization overhead when applied for iterative computational tasks. On the other hand, Pregel-like systems cannot efficiently process queries involving properties other than ID. Thus, it is our major concern to improve Pregel-like graph processing systems for a better performance on querying any properties on these systems. We investigate various index mechanisms that fit the graph data querying on distributed systems and apply one of them to the Apache Giraph Framework. Finally, we validate the superiority of our proposed mechanism by experimenting on massive synthesized graph data.

Key words distributed graph processing system, big graph, index, non-ID property, synchronization/ asynchronization

摘要 随着 Web 技术、社交网络、基于位置的服务等互联网技术和应用的发展，大规模图结构数据的存在越来越广泛。用于高效完成大规模图数据分析任务的 Google Pregel 及其开源实现 Apache Giraph 已经在实际应用中发挥了极大的作用。然而现有系统存在两个主要问题：首先，现有系统无法高效完成基于图的迭代式计算任务。其次，Giraph 等类 Pregel 图数据处理系统尚无法高效处理涉及节点 ID 以外的复杂条件的查询。针对上述问题，我们提出了一种分布式大图处理系统中的索引设计及其若干改进，并基于现有的分布式环境 Apache Giraph 实现上述索引机制，最后我们通过实验验证了我们提出的索引机制的有效性。

关键词 分布式；大图；索引；非关键码属性；同步异步

中图法分类号 TP311.13

随着 Web2.0 时代的到来，互联网上的信息海量增长。图结构是对这类数据的良好抽象。在图中，节点可以代表任何实体，节点之间的边代表实体之间的联系，节点的属性描述了该节点的特征，从而构成一个属性图网络。属性图作为一种灵活的数据模型，在日常生活、科学技术和商业等各个领域都广泛存在。

基于这样的现状，学术界和工业界产生了多种大规模图数据处理系统。其中技术成熟、应用广泛的系统有 Pregel[1], Giraph[2], GraphChi[3], GPS[4], GPI[5] , Snap[6]等。这些系统对于离线的数据分析任务有着优异的性能，却不能非常高效地完成在线的、涉及非 ID 属性的复杂查询请求。对于这类常见且极为重要的查询，上述系统需要遍历所有的节点，导致整个查询过程的代价庞大。为了在分布式大图处理系统中实现节点 ID 以外的复杂条件的查询以满足应用需求，迫切需要一套有效的机制以加速此类查询。

本文通过为系统建立索引的方式，来解决上述问题。本文主要贡献如下：

1. 设计了一种面向大图处理系统的索引设计以支持复杂查询的高效执行，并提出了负载均衡相关优化策略。

2. 通过扩展原有的系统，支持异步消息的发送接

收稿日期：2017.5.31；**修回日期**：2017.7.9
基金项目：科技部重点研发计划资助(2016YFB1000700); 深圳市科技计划项目(JCYJ20151014093505032); 国家自然科学基金项目(61572040).
通信作者：陈修司（xiusichen@pku.edu.cn）

收，在查询阶段利用异步接口加速查询过程。

3. 通过大量在随机人工图上的实验，验证我们的索引机制在正确性及实行效率上的有效性。

1 相关工作

本章以 Giraph 为例介绍现有大图处理系统的计算模型及编程框架，并在此基础上阐明现有工作的一些不足之处及我们工作的主要关注点。

Apache Giraph 是一套迭代式大规模图数据处理系统，其计算任务的输入是一个由节点和有向边组成的图（图 1），该输入确定了图的拓扑结构以及图中节点和边的初始值。

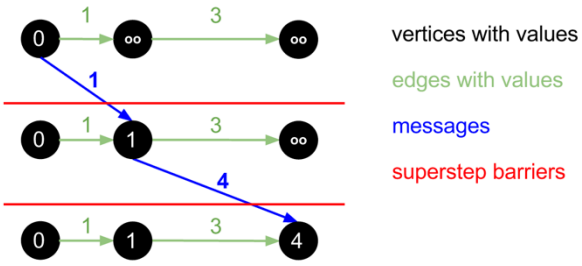

图 1 Giraph 执行任务框架

Giraph 采用了 BSP[7]计算模型，它的计算由一系列的迭代序列组成。这里的迭代称为“超步”。初始时，每个节点都是“活跃的”。用户面向图中的节点描述其计算任务。每个超步中，每个活跃的节点调用由用户定义的 Compute()方法。Compute()方法应明确三个任务的具体执行指令：（1）接受上一个超步发到当前节点的消息；（2）利用收集到的消息、节点当前的值、出边的值进行计算并产生新值；（3）若有需要，向其它节点发送消息。

相邻的超步之间设有同步屏障，它保证了任意当前超步发送的消息只在下一超步到达目标节点，以及所有节点完成当前超步的计算后才开始下一超步。

节点可以在完成自己的计算任务之后变为“不活跃”状态。当所有节点都变为不活跃，并且图中没有消息在传递的时候，整个计算任务结束。

Giraph 这些类 Pregel 的系统对于大规模离线数据分析任务有良好的处理性能[8]，但是仍然存在以下问题：首先，由于 Giraph 只支持在节点 ID 上的索引功能，对于现实中经常出现的涉及非 ID 属性的查询，则需要遍历图中所有的结点，并逐一作出选择。其次，BSP 计算模型只支持同步计算，而同步计算则使得一部分并行度损失。

2 分布式大图系统的索引设计和实现

本章主要介绍我们提出的分布式大图索引设计，我们的设计包括索引数据结构的选取、B+树索引的增量式和批量式构建，以及通过异步消息对于查询阶段效率的加速。本文提出的索引设计适用于现有的类 Pregel 大图处理系统，但考虑到 Giraph 平台应用的广泛性，我们选择在该平台上对索引进行实现。尽管采用了 B+树这一数据结构，本文设计的关注点在于如何在分布式环境下的大图处理系统中，高效建立针对复杂图数据查询的有效索引，这和针对关系型数据进行单机优化的传统数据库优化工作存在显著不同。

2.1 数据结构的选取

顺序索引结构及 B+树索引结构是两种常见的索引数据结构。顺序索引结构最大的缺点在于，随着文件的增大，索引查找显著下降。B+树索引结构是使用最广泛的索引结构之一，它能够在数据频繁增删的情况下保持对数级别的执行效率。由于 B+树在结构上也属于图结构的一种，能够非常自然地与被索引的图数据深度融合，嵌入到我们的分布式大图处理系统中，所以我们选择 B+树这种索引效率高，并具有图特征的数据结构来实现我们分布式大图系统中的索引。

2.2 分布式 B+树索引概述

在我们的 B+树索引中，用一个标记位来区分索引节点与数据节点。索引树根节点的 ID 固定分配，在由根节点构建的 B+树分支中，索引节点的 ID 自增式往上加。由于 Giraph 将节点分配到机器的策略采用根据节点 ID 哈希的机制，我们不需要额外考虑索引的物理存储优化，而可以充分利用 Giraph 系统自身对于图数据高度优化的数据结构设计和底层存储。

2.3 BSP 模型单点增量式构建 B+树

由于 Giraph 采用了 BSP 模型，我们首先尝试在 BSP 同步模型下使用一种单点增量式的，每次将一个数据项加入索引的算法构建 B+树。它的基本步骤是：

1.每读取一个数据项，找到其应该插入的叶节点。

2.若叶节点所包含的关键码没有达到上限，则直接插入；否则，递归向上执行分裂

这种算法适合少量数据插入的情况如动态图索引构建，而不适合运行在初始情况下对于大量离线图数据进行索引建立的场景。这是由于 Giraph 在每个超步之间存在同步屏障(barrier)，这一机制导致所有对图结构做出的改变和某一节点在当前超步发出的消息，在下一个超步才会被处理和被目标节点接收。对于一个新的索引项的插入，最坏情况需要经过h个超步搜索应该插入的叶节点，$h+1$个超步向上分裂直到根节点分裂，共$2h+1$个超步。其中h是树高。在我们的小型实验中，当 B+树度数为 3 时，插入前 10 个节点经过了 24 个超步；前 12 个节点插入需要 32 个超步。这样的同步代价对于我们在大图系统下的索引构建工作而言，这无疑是低效的。

2.4 改进的 BSP 模型多点批量式构建 B+树

为解决上文提出的对海量数据进行索引建立时

的效率问题，我们进一步提出了一种在 Giraph 平台上运行高效的多点批量式 B+树加载算法，主要步骤如下：

1) 新建一个节点负责整棵 B+树的创建；

2) 该节点将数据节点按索引码排序，将索引码按序写入叶节点中，并保证叶节点尽可能被填满。

3) 根据叶节点构造中间节点直至创建根节点。

4) 当图的结构发生少量变化时，使用前文的增量式排序；如果新增的节点数量达到足够多（比如原有节点数量的 20%时），把该图上原有的所有非 ID 属性索引全部删除，重新运行批量加载构建 B+树算法。

```
Function BatchBPTreeConstruction(messages):
Sort(messages);
upperMessages = new ArrayList();
while(messages.size() > idxDegree)
  while(message.size() > idxDegree + idxDegree/2)
    newPartition = partitionValue.fetch(idxDegree);
    addVertexRequest(numNodes++, newPartition)
    upperMessages.add(newPartition.max());
  while(message.size() in
        [idxDegree,idxDegree+idxDegree/2])
    newPartition1 = partitionValue.fetch(idxDegree);
    addVertexRequest(numNodes++, newPartition1)
    upperMessages.add(newPartition1.max());
    newPartition2 = partitionValue.fetch(idxDegree);
    addVertexRequest(numNodes++, newPartition2)
    upperMessages.add(newPartition2.max());
  exchange(messages, upperMessages);
return rootNode of the B+ tree
```

算法 1 改进的 BSP 模型批量式构建 B+树

算法 1 解决了增量式算法中超步过多问题，但却引入了一个新的问题：对索引项的排序和整个索引的构建都由单一节点在一个超步之内完成，在这个超步中该节点的计算任务过大，而其它节点没有任何计算任务。负载过度集中于单一节点会导致构建时间太长。

为了解决这个问题，我们扩展 B+树索引为 B+森林索引：我们建立多个根节点，每个根节点负责对固定数量的数据节点创建一棵 B+树。并通过以下计算保证每一棵 B+树索引的数据节点互不重叠：

假设N为数据节点总数，D为索引 B+树任意节点的最大子节点数，索引森林中T棵树。

每一棵树的叶节点个数为$n = {N}/{TD}$;

$\Rightarrow$树高为$\log_D n$

$\Rightarrow$每一棵树的节点总数不超过$D^0 + D^1 + D^2 + \cdots + D^{\log_D n}$

$$= {1 - D^{\log_D n+1}}/{1 - D}$$

$$= {N - T}/{TD - T}$$

也就是说，在保证索引节点 ID 最小值大于数据节点 ID 最大值的基础上，我们只需要使森林中每棵树根节点的 ID 间隔大于 ${N - T}/{TD - T}$ 即可。

最终，我们根据节点 ID 将该节点哈希到某个根节点，这一优化有效避免了单点任务过大的问题。

2.5 异步查询阶段处理框架

现有大图处理系统在 BSP 模型的约束下，图中任意节点必须等待图中其他节点当前超步结束后才能开始下一个超步。然而，由于查询任务对于数据仅进行只读操作，我们可以通过异步执行允许节点执行完当前超步的任务之后，立即进入下一超步，以增加系统的并行性。我们在原有的系统下扩展了发送和处理异步消息的接口，支持异步消息查询的直接反馈。在查询阶段，我们的查询由一个查询节点发送到索引森林的每个根节点，利用异步消息接口并行地完成查询过程。

3 实验与结果

为了考察我们提出的索引机制的构建效率以及与原有 Giraph 系统对比下对于不同类型的查询带来的查询效率提升，我们分别考察了 B+树索引结构对于索引构建时间的影响和 B+树索引对于不同类型的查询带来的查询效率提升。

3.1 实验数据

在所有实验中，我们通过 GTgraph[9]来人工生成图数据。我们控制我们需要生成的图的节点数量、节点平均度数等参数随机生成了节点个数分别为 100、1000、10 万、100 万的随机图，平均度数为 10。

3.2 实验设计与结果分析

3.2.1 构建时间

为了考虑索引中 B+树的数量对构建时间的影响，我们分别在 3.1 中的 4 个图数据集上构建了由 B+树数量为 100、1000、10000、100000、1000000 的索引森林。实验结果如下：

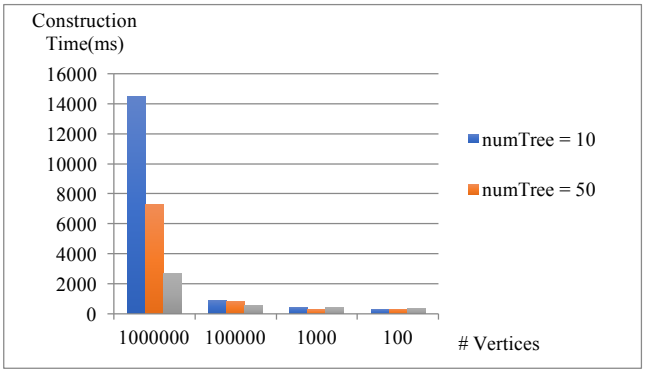

图 2 B+树数量对构建时间的影响

从图 2 中可以看出，索引构建的时间受索引中 B+树的数量的影响比较大。当 B+树数量比较少时，构建时间长；当 B+树数量比较多时，构建比较快。这是由于增加 B+森林中 B+树的数量提高了索引构建阶段的并行度，削弱了单点瓶颈现象。当同时参与构建 B+树的节点越多时，每个节点的计算任务较小，单点瓶颈效应越弱。这说明我们在 2.4 中提出的 B+森林索引方案能够有效加速索引的构建效率。

3.2.2 查询

我们进一步对比了我们的通过索引的查询和原始 Giraph 同步查询，在单点查询、范围查询的时间代价：

实验数据：节点数为 100 到 1000000 的随机图；随机图中的每个节点有 5 个不同类别的属性，包括字符串、布尔值、整数、浮点数，用以模拟社交网络中用户的姓名、性别、年龄、爱好、标签等属性。

单点查询：从图 3 中可以明显看出，当查询条件为选择出某一个属性为某一确定值的时候，索引异步查询的效率明显高于原始 Giraph 同步查询。

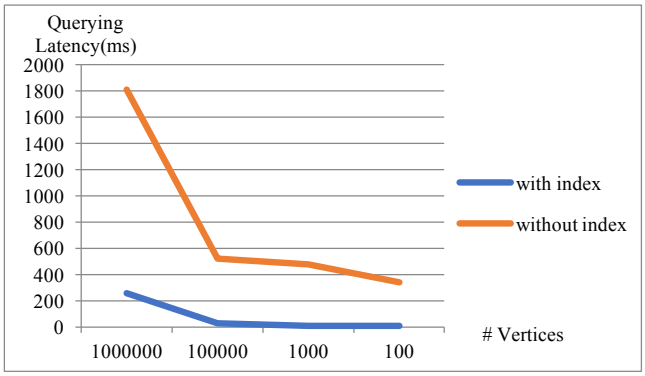

图 3 单点查询同步与异步查询时间对比

范围查询：我们以查询非 ID 属性$Value \in [18, 28]$为例，比较了异步方式与同步遍历方式的查询时间。一下实验在节点总数为 100~1000000 的随机图上完成。实验结果如图 4。

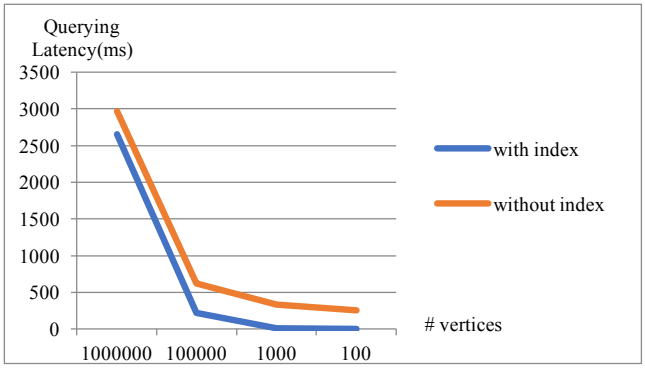

图 4 范围查询同步与异步查询时间对比

结果分析：在范围查询中，异步查询方式效果仍然好于同步方式，但是当图的原始节点数增大，导致处在$[18, 28]$范围内的数据节点增多时，返回给 BatchBPTreeIndexMasterCompute 节点的消息量成正比增加。由于在异步方式中，节点每收到一条消息就立马开始计算任务，故同时到达的其它消息只能盲目等待，所以异步查询时间产生了无谓损失，与同步遍历方式相比，优势相对减小。

4 总结

本文针对在 Giraph 分布式大图处理系统下，如何高效地处理对非 ID 属性的查询执行进行了分析和研究。首先，我们回顾了主流的大图数据处理系统，选取 Giraph 平台作为我们的实验环境。我们在 Giraph 中支持和引入非 ID 属性的索引，并在数据查询中同时支持同步、异步。在建立索引阶段，我们优化了构建算法；在查询阶段，我们针对复杂的查询扩展了现有的系统接口使其消息传递摆脱同步屏障的限制，增大了并行度。最后，我们通过在大数据集上的实验证实了我们方法的正确性和高效性。

参 考 文 献

[1] Malewicz, Grzegorz, et al. "Pregel: a system for large-scale graph processing." Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010.

[2] Welcome to Apache Giraph, [OL]. [2017-05-10] http://giraph.apache.org/

[3] Kyrola, A., Blelloch, G.E. and Guestrin, C., 2012, October. GraphChi: Large-Scale Graph Computation on Just a PC. In OSDI (Vol. 12, pp. 31-46).

[4] Salihoglu, S. and Widom, J., 2013, July. Gps: A graph processing system. In Proceedings of the 25th International Conference on Scientific and Statistical Database Management (p. 22). ACM.

[5] Ekanadham, Kattamuri, et al. "Graph programming interface (GPI): a linear algebra programming model for large scale graph computations." Proceedings of the ACM International Conference on Computing Frontiers. ACM, 2016.

[6] Leskovec J, Sosič R. Snap: A general-purpose network analysis and graph-mining library. ACM Transactions on Intelligent Systems and Technology (TIST). 2016 Jul 15;8(1):1.

[7] Valiant, L.G., 1990. A bridging model for parallel computation. Communications of the ACM, 33(8), pp.103-111.

[10] Gerbessiotis, A.V. and Valiant, L.G., 1994. Direct bulk-synchronous parallel algorithms. Journal of parallel and distributed computing, 22(2), pp.251-267.

[8] Facebook, Inc. Facebook Reports First Quarter 2015 Results[R]. California: MENLO PARK, 2014.

[9] GTgraph: A suite of synthetic random graph generators, [OL]. [2017-05-10] http://www.cse.psu.edu/~kxm85/software/GTgraph/