一种 Spark 环境下的高效率大规模图数据处理机制 *

杨天晴，王　津，杨旭涛，张学杰
(云南大学 信息学院，昆明 650091)

摘　要：针对现有的图处理和图管理框架存在的效率低下以及数据存储结构等问题，提出了一种适合于大规模图数据处理机制。首先分析了目前的一些图处理模型以及图存储框架的优势与存在的不足。其次，通过对分布式计算的特性分析采取适合大规模图的分割算法、数据抽取的优化以及缓存、计算层与持久层结合机制三方面来设计图数据处理框架。最后通过 PageRank 和 SSSP 算法来设计实验与 MapReduce 框架和采用 HDFS 作持久层的 Spark 框架做性能对比。实验证明提出的框架要比 MapReduce 框架快 90 倍，比采用 HDFS 作持久层的 Spark 框架快 2 倍，能够满足高效率图数据处理的应用前景。

关键词：图计算；内存计算；图数据库；Hadoop；Spark；PageRank
中图分类号：391.41

High efficiency large-scale graph data processing mechanism in environment of Spark

Yang Tianqing, Wang Jin, Yang Xutao, Zhang Xuejie
(School of Information Science & Engineering, Yunnan University, Kunming 650091, China)

Abstract: Due to the inefficiency problems in processing, storage and management framework of graph data, a feasible processing mechanism of large-scale graph data was proposed in this paper. The pros and cons of existing graph processing models and graph data storage frameworks were first reviewed. By analyzing the characteristics of distributed computing, a new graph data framework was implemented including three main parts: segmentation algorithm of large-scale graph, caching and optimization for data extraction, and combination mechanism of calculation and persistence layer. By applying Pagerank and SSSP algorithm, experiments were conducted to compare the performance of the proposed framework, MapReduce and Spark with HDFS. Results showed that the proposed framework is more 90x faster than MapReduce, and 2x faster than Spark with HDFS, and the proposed framework can satisfy the needs of high performance graph data processing.

Key Words: graph computing; graph database; memory computing; Hadoop; Spark; PageRank

0　引言

相对于线性表结构和树型结构，无论是在结构还是语意方面，图数据结构在现实世界都具有更好的表达能力。甚至现实生活中的一些场景用图数据表示更具有优势，因此针对图计算的研究具有重大意义[1]。现实世界的实体扩增和大量数据的大规模应用，给大数据存储及处理带来了巨大的挑战[2,3]。在图数据爆炸式增长和图数据处理需求多样化发展的时代，涌现出了很多大规模图数据处理系统。这些系统大致可以分为两类：基于 MapReduce 模型的分布式并行处理系统[4,5,6,7]以及基于 BSP (Bulk Synchronous Parallel Computing Model)模型的分布式并行处理系统[8,9,10]。MapReduce 计算模型[1]在数据挖掘和数据分析领域取得了很大的成绩，并在业界被广泛的采用。该类系统是具有较好容错性的非循环的数据量模型。就目前状况来看，这些计算模型都能容易的访问分布式集群中的资源，但是对内存的利用率很低，以至于对那些重复利用中间结果的计算显得效率很低[11,12]。这些算法都是在多个并行计算之间重用数据，例如单源最短路径等迭代算法、PageRank[13]。UC Berkeley AMPLab 在 2009 年提出的 Apache Spark[14]计算模型刚好解决了这些问题。Spark 将分布式内存抽象成弹性分布式数据集 (Resilient Distributed Datasets，RDD)[14, 15]。RDD 支持大规模分布式数据集的操作，并且具有数据流模型的特点：自动容错和可伸缩性。为了方便多个操作时能够重用相应数据集，RDD 允许用户显式地将数据集缓存在内存中，从而极大地提高了查询效率。但是 Spark 的持久化机制需要将图结构数据转换为元数据存储，并不能在存储层保持图数据的结构，这将为整个框架

基金项目：国家自然科学基金资助项目（61170222）
作者简介：杨天晴（1991-），男，云南保山人，硕士研究生，主要研究方向为图数据分析和高性能计算；王津（1985-），男，博士研究生，主要研究方向为机器学习、文本挖掘和大数据资料分析；杨旭涛（1984-），男，博士研究生，讲师，主要研究方向为图数据分析和高性能计算；张学杰（1965-），男，教授，博导，主要研究方向为高性能计算和大数据资料分析（xjzhang@ynu.edu.cn）.

带来不必要的开销。同时在元数据上的查询操作也会涉及到复杂的计算，使得整个应用的性能大幅度降低。基于 BSP 模型的分布式并行处理系统中，最具有代表性的就是 Google 的 Pregel 系统[16]。Pregel 系统是目前大规模图处理系统中比较成熟的系统，它对于图的切分、计算、同步控制、容错管理都提出了比较稳定的解决方案。但是 Pregel 系统所处理的作业类型较为单一，只能支持大规模的图数据分析，对高效的图数据查询不能得到满足。作为 Apache 的另一个开源项目 Hama[9]也是一个分布式处理系统。Hama 可以与 HDFS[17]完美的结合，这样有利于数据的持久化，因此近几年 Hama 在图处理中有很强的应用性。然而，Hama 仍然是一个离线批处理系统，在大规模的图数据处理中响应性能达不到需求。

Spark Graph X[14]作为一种分布式内存共享图处理框架，在主从式集群上实现了图的内存计算。它对于图的加载、顶点处理、边处理和相邻节点计算等方面对通信的要求更低,产生的图结构更简单，从而在性能方面得到了很大提升。利用 Graph X 可以便捷的构建多种图操作算法。但是 Graph X 的持久化机制尚未实现集群节点的全局化统一管理。虽然 Graph X 在存储层支持分布式文件系统(HDFS)和分布式 NoSQL 数据库 HBase[18]，但是目前还不支持图形数据库，以至于不能将数据以图结构的形式持久化到存储层，严重的影响上层服务对数据的遍历和查询。存储层和计算层的协调一致对于一个大规模图数据处理框架是必要的。

大规模图数据存储基于云计算环境下的分布式存储系统，这些存储系统分为两种：一种是以 GFS[19]、HDFS 为代表的分布式文件系统，另一种是以 Neo4j [20]、MySQL 为代表的分布式数据库。如果图数据存储在分布式文件系统上，因为不支持数据更新和数据插入操作以及需要人为的组织数据存储模式，使其在复杂的图数据操作和管理上没有优势。虽然传统的关系型数据库已成为目前应用最广泛的数据库，但随着数据规模与复杂性的增加，关系模型已不能满足领域需求，以社交网络[21]为例，采用传统的关系型数据库将产生大量冗余数据，且不能满足社交数据的实时性，也不能有效的支持复杂的多层传递关系查询。针对数据之间内在的复杂关系以及动态变化的问题，人们将研究重心转向图数据库(Graph Database)[22]，它能够有效的对图数据间的内在关系进行存储、管理、和更新，并且能够高效的对图数据之间的复杂关系进行操作[23, 24]。在面临如此庞大的数据量时，图处理工作的难题主要表现为两个方面。其一是如何提高图数据的处理能力，得到一个良好的图处理机制。其二是如何解决大规模图数据管理问题，在扩展性、数据查询效率和大数据吞吐量等方面得到一个良好的性能。

本文的组织结构为：第 1 节首先针对目前主流的图处理系统 MapReduce[5]、Pregel[6]、Hama[7]等做了详细的分析和比较，分别说明了各个系统的优点与不足。其次分析了目前图数据管理的两种方式，并且分析了针对图数据存储非关系型的分布式数据库相比分布式文件系统的优势。第 2 节从图分割、数据抽取以及框架结构等方面介绍本文提出的大规模图数据处理框架。第 3 节本文选取 Web 图数据的典型算法 PageRank 和 SSP 作为比较对象，分别给出了 Hadoop 平台下基于 MapReduce 的设计思路、以 HDFS 为存储层的 Spark 框架和本文提出的框架下的设计思路，并且通过实验验证了本文提出的框架在大规模图数据处理中要比其余两个框架的性能更高。

1 基于 Spark 的大规模图数据处理框架

本文提出的分布式图计算框架将多种对大规模图数据操作封装为 API，让分布式存储、并行计算等复杂问题对上层透明，从而使用户更关注图的相关模型设计和使用。因此不需要考虑底层的实现。为了实现该目的，我们针对两个主要的问题进行分析：图的计算模型和图的存储模型。

表 2-1 带权重的深度优先搜索树算法

Init.	G 为包含 n 个节点$\{x_1, x_2, \ldots, x_n\}$的图;
Step 1	选择有最大下标的节点，即 x_n，标记为 1。将这个点和标记代入 **Step 2**;
Step 2	给定一个节点 x，标记为 k。如果该点存在邻接点且未标记则选择具有最大下标的邻接点，为其从集合$\{1, 2, \ldots, n\}$中分配最小未用标记并对该点和标记重复 **Step 2**。否则，意味着点 x_j 的所有邻接点均被标记过。 **if** 节点 x_j 的标记 k 满足 k>1，回溯到该节点被标记为 k 的位置，重复 **Step 2**; **if** 节点 x_j 的标记 k 满足 k=1，则算法终止。

表 2-2 将大图分割成若干个均衡的子图算法

Step 1	用 S 来表示 G 中点的子集，用 c_i 表示子图的点集，其中 i=1, 2, …, n，用 c_w 表示临时工作集。初始化设置 S=$\{x_1, x_2, \ldots, x_n\}$, C=Φ, 并且 c_w=Φ;
Step 2	选择最大长度的叶子节点作为分割算法的搜索起始节点，将这些叶子节点移植到 c_w 工作子集中同时从 S 中进行删除。此外假设叶子节点的最大长度为 k 且子图的 pointer i=1;
Step 3	$k = k - 1$;
Step 4	在 c_w 中寻找有不同父节点的节点，对于每个父节点 x_p 将其孩子节点$\{x, [x_p = \text{parent}(x_j)]\}$按照子树的权重 $wt(T[x_j])$ 以降序的方式进行排列，之后测试序列中的每一个孩子节点 x_j，如果 $T[x_j]$ 满足：$$\left| wt(T[x_p]) - wt(T[x_n]) / n_g \right| \geq \left| wt(T[x_n]) / n_g \right| \quad (1)$$ 之后分配子树 $T[x_j]$ 作为字数 c_i=$T[x_j]$并且 i=i+1，同时删除从工作子集 C_w 和 S 中删除 x_j 和 $T[x_j]$，如果 x_p 的所有孩子节点都不满足(1)，就从 C_w 中删除 x_p 的所有孩子节点，并将 x_p 加入到 C_w 中，重复 **Step 4** 直到所有不同的父节点都被测试过。
Step 5	找到所有长度等于 k 的叶子节点，并将这些叶子节点加入到工作集 c_w 中;
Step	如果 k=0，且 S=Φ，则意味着 G 的所有节点都被系统谈就

6	过了，因此算法终止，否则重复 **Step 3** 到 **Step 6**。

表 2-3 迭代处理算法

Step 1.	i=1；
Step 2.	对子图和 C_i 其边界节点集 F_i 建立相应的双边 H_i，同时初始化 $M=\Phi$；
Step 3.	对于中 F_i 的每张路径可以被找到，那么增加这条路径的匹配 M。重复执行 **Step 3** 知道 F_i 的所有节点都被测试过，转向 **Step 4**。否则，根据以 x 为根的交替树，通过 $Y=L_0 \cup L_2 \cup L_{2j}$ 返回需要的 Y，得到改良过的基于**命题 1** 的和 f_i，中断并从 **Step 1** 重启该算法；
Step 4.	令 $i=i+1$,如果 $i=ng$，则这个边界减少算法终止，否则跳到 Step 2。
命题 1	假设 F_i 是 C_i 和 U 的边界节点集，则 $\overline{F_i}=(F_i-Y)\cup Adj(Y,C_i)$ 是 $\overline{C_i}=C_i-Adj(Y,C_i)$ 和 $\overline{U}=U\cup Y$ 的边界节点集。

1.1 图分割策略与分布式计算

图划分的质量对整个集群的存储和计算效率、节点之间的通信及负载均衡都有较大影响。优化划分是通过减少顶点或边跨越划分的数目，来减少集群节点间的通信，从而加快计算收敛的速度。典型的图分割算法包括随机划分、基于边的平衡划分、基于顶点的平衡划分和启发式划分。如图 3-1 所示，基于边的平衡划分和基于定点的平衡划分方法为：

(1) 图的切边法是按照图的边进行划分，把顶点尽可能均匀的分布到每个计算节点，使边在各节点之间跨越的数量最小化。该方法要求每一个割边需要多个计算节点上保留复制和通信，以保持图之间的结构依赖关系。

(2) 图的切点法是按照中心顶点划分，使边尽可能均匀的分布到每个计算节点。该方法可以最小化存储可通信开销，图 2-1 显示了这两种切割策略。

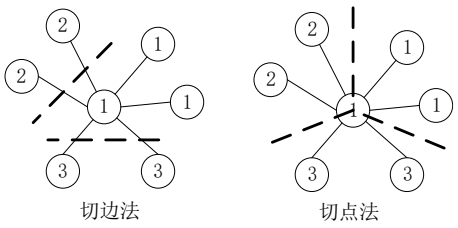

图 2-1 图的切边法与切点法(虚线为图的切割)

图分割的难点在于现实应用中的图一般符合幂律分布，这对分布式集群的工作平衡带来巨大挑战，使得图很难被均匀的分割。并且以 Hash 算法进行的图划分将使得数据分布局部性非常差，从而增加了节点之间的通信开销。另一方面，为了提高缓存系统的命中率和置换效率，图的分割要保持块内的子图联系紧密,块间联系松散。为了解决这些问题我们选择了一种高效的多路大规模图分割法[25]，该算法主要通过 3 个步来实现大规模图的分割：(1)建立带权重的深度优先搜索树；(2)将大图分割成若干个均衡的子图；(3)迭代处理,尽量减少子图之间的关联，表 2-1、2-2 和 2-3 为该算法详细步骤。

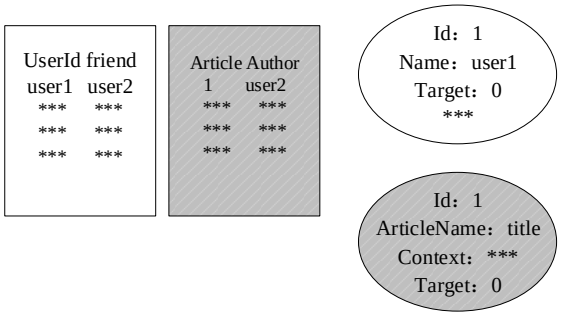

图 2-2 抽取出的结构数据及持久层的元数据

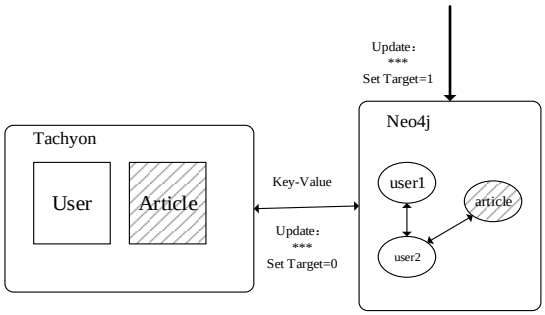

图 2-3 数据的更新与抽取操作

目前被业界广泛应用的图计算框架有 Google 的 Pregel、Apache 的 Giraph[26]，以及 GraphLab[27]，其中 Pregel 和 Giraph 都是基于 BSP 模型的。BSP 模型实现了超步，首先进行了本地计算，然后进行全局的通信，最后进行全局的 Barrier。虽然 BSP 编程简单，但是在一些情况下其运算性能较低。因为存在一个全局障碍，所以整个系统的运算时间取决于耗时最多的计算，即木桶的短板原理。另一方面，现实世界中的很多的关系都服从幂律分布，也就是顶点和边的分布很不均匀，因此,BSP 的木桶短板问题会很突出。Spark Graph X 采用了一种异步机制避免了全局的 Barrier。

本文采用 Spark Graph X 作为图计算引擎。Spark Graph X 是基于内存的分布式图计算框架，Spark Graph X 建立在 Spark Core 之上为图计算和图挖掘等应用提供了简洁的接口，极大地方便了用户对大规模图数据处理的需求。图分布式计算是将图拆分为多个子图，对其进行并行计算。

1.2 元数据到结构数据的提取与分割

在元数据到结构数据的提取中我们需要尽量减少不必要的数据流，以减少通信开销。在此之前我们不妨先了解 Spark Graph X 是如何构建图的。对于图坐标，坐标中每个点用 x 表示横坐标和 y 表示纵坐标，即：(x_1, y_1)和(x_2, y_2)，一个坐标点可以确定一个点的唯一位置。Graph X 与上面的概念类似。不同的是，Graph X 中的顶点可以通过标示唯一的对象，不一定要

采用二维坐标表示。在 Graph X 中用户通过 Graph 对象对其进行操作，它包含了边(edge)和顶点(vertices)两部分。所有边集包含的点构成点的全集，其中包含重复的点。去重后就是 VertexRDD。然后从持久层中加载边的数据，按行进行处理生成 tuple，即(srcId, dstId)。

也就是说，我们要构建基本的图只需要按照行处理来提交节点的数据，这样只需要将元数据的关键属性提取出来以矩阵结构放在缓存系统中。不同的关系以不同的文件区分。下面我们以一个简单的实例来说明：

一个社交平台中需要保存的用户属性是非常巨大的，而且还有用户之间的好友关系。同样用户发表的文章也是如此。但是我们在处理图数据的时候不需要如此多的属性来计算。我们在缓存系统中构建用户节点和好友关系的时候只需要用户的 id 属性以及计算需要的其他属性，而不同的关系以不同的文件区分，如图 2-2。

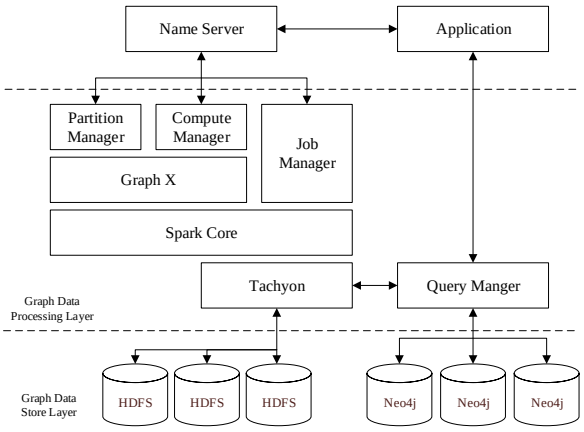

图 2-4 框架结构

在每一次大规模的图计算中，考虑到某些节点可能一直没被应用层写入过新的数据。这些节点是不需要更新到缓存文件中去的，所以我们在节点属性中增加了一个 target 属性，当应用程序对持久层写入完成的时候将此属性置为 1。在缓存系统读入完成的时候将其置为 0。这样就可以以此属性过滤掉不必要读入的节点，减少了系统的通信开销，如图 2-3 所示。

本文选择 Neo4j 作为元数据的存储层是为了保证应用层高效的图操作响应。Neo4j 是自适应规模的，而且其图遍历执行速度是常数，与图大小无关，所以其读性能相对较高[28,29]。但是若要将这些元数据直接放入 Spark Graph X 中计算将带来两个巨大的开销。一是会给整个集群带来巨大的通信代价，另一方面用这样的元数据来构建图需要大量的 OGM (object graph mapping) 过程在内存中进行，会导致集群消耗大量的计算性能和不必要的内存开支。

云计算环境下的索引机制，并没有考虑图结构的特点，因此在图查询方面，效果不理想。而分布式图数据库，在图数据存储和索引结构上都进行了优化。Neo4j 的索引机制分为两类：Neo4j 采用树形结构索引，从而提高了查询效率。同时 Neo4j 可使用独立的 Lucene 索引，在全文的基础上建立索引，并对索引命中率进行排序，从而提高了查找速度。

某些情况下，从持久层（Neo4j）到缓存储（分布式内存文件系统 Tachyon）的数据抽取效率将可能会影响整个系统的计算性能，因此我们在该过程中要尽量减少不必要的通信开销以及对图数据的操作。为了达到该目的，我们将缓存层中的结构数据按照实体间的关系进行分割，以便满足分布式计算特性。在缓存层方面，虽然引入缓存体系会加大内存的开销，但是分布式内存文件系统（Tachyon）的大小会随着计算节点的增加而增加，这样整个缓存体系就不会因为数据量的大幅度剧增而遇到瓶颈。同时 HDFS 和 Tachyon 的协同工作保证了缓存层的稳定性。其次，我们在元数据到结构数据的提取过程中加入了属性过滤机制，以避免不必要的节点和属性被载入内存。即使是在数据量很大的时候也不会频繁和大量的更新缓存层的结构数据，再加上持久层优秀的索引机制，从而避免了数据量增大而导致整个系统系能降低的问题。同时保证缓存层中的数据结构与计算层中需要构建的图数据结构一致，这样才能解决通信开销带来的瓶颈以提高整个框架的效率。

1.3 框架结构

该框架分为三层，如图 2-4 所示。顶层接收用户的查询请求，返回计算结果给用户及存储被分割的元数据。顶层由一个 Name Server 和一个 Application Server 构成，Name Server 面向应用提供接口和管理服务。Application Server 接收到用户提交的查询请求会直接交给 Query Manager 处理并返回处理结果。其他操作会交给 Name Server 协同处理。

图数据处理层将整个架构建立在 Spark core 上，包括了四个管理器用来处理相应的事务。第三层为图数据存储层，为数据提供存储。

(1) Query Manager：为了使整个框架得到一个高效的访问性能，持久层需要能够直接被应用层访问。Query Manager 除了要满足应用层提交的事务处理请求，还要将元数据结构化，提交到缓存系统 Tachyon。当一个分布式计算结束后，会将计算结果以 Key-Value 键值对的形式将结果返回到 Neo4j 集群中。

(2) Job Manager：Job Manager 负责整个分布式图计算框架的任务调度并对整个 Spark 的运行进行管理。为了保证结构数据缓存的安全性，需要负责缓存系统 Tachyon 到 HDFS 的数据备份工作。

(3) Computing Manager：Computing Manager 是负责图分布式处理的管理器。通过抽象出一系列 API 来简化基于图的编程。整个管理器搭建在 Spark Graph X 之上，而 Spark Graph X 本身是一个分布式图处理框架，基于 Spark 平台提供了对图的计算和数据挖掘的简洁易用的接口，极大的方便了用户对大规模图处理的需求。

Computing Manager 需要将用户提交的算法整合到框架中，调度这些算法来运行任务。在收到 Job Manager 的操作请求时，

会从缓存系统(Tachyon)中取出数据，提交到 Spark Graph X 计算。为了减少集群间的通信以及整个系统的 I/O 开销，计算的中间结果和最终结果将会被放在到缓存系统(Tachyon)中，最后由 Query Manager 处理数据的流向。

(4) Partition Manager：结构化数据会被 Partition Manager 进行图的分割，每个子图会被传递到相应的节点中去进行计算。当缓存系统(Tachyon)的空间被占满时，Partition Manager 会采用 LRU 算法与 HDFS 替换结构数据文件。

在本文设计的框架中将结构数据和元数据都采用分布式的方式进行管理。无论是在持久层（Neo4j）还是缓存层（Tachyon）中，我们都采用了分布式的图数据库和分布式的内存文件系统。每个 Manager 的协同工作保证了整个分布式计算框架的计算性能较高。同时还能解决传统分布式计算框架中通信开销和结构数据转换的瓶颈问题，这样能有效减少算法执行的时间。

2　实验

2.1　实验环境

本文在一个小规模集群中进行实验对比分析，该集群由 3 个 Dell R-210 节点组成。三个节点之间用交换机相互连接，并且配置相互之间的 SSH 无密钥连接。每个节点的硬件配置如下：Intel(R) Xeon(R) E3-1220 CPU，4 核，主频 3.10GHz；内存为 4G；硬盘大小为 1T。软件配置如下：操作系统是 64 位 Ubuntu 12.04.5 LTS；JDK 版本是 jdk1.7.0_71(x64) ；Hadoop 版本是 Hadoop-2.2.0(x64) ；Neo4j 的版本是 Community-2.1.5；Scala 为 Scala-2.10.4 版本；Spark 的版本是 Spark-1.0.0。

2.2　实验数据及实验叙述

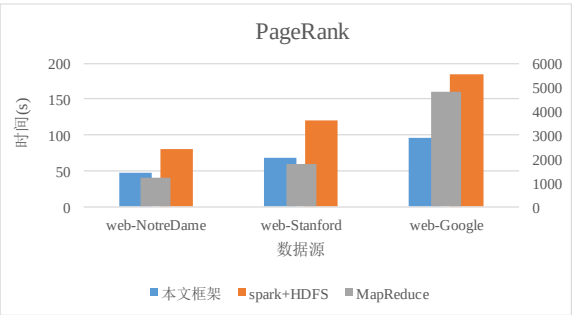

图 3-1 PageRank 算法实验结果（MapReduce 结果为次坐标轴）

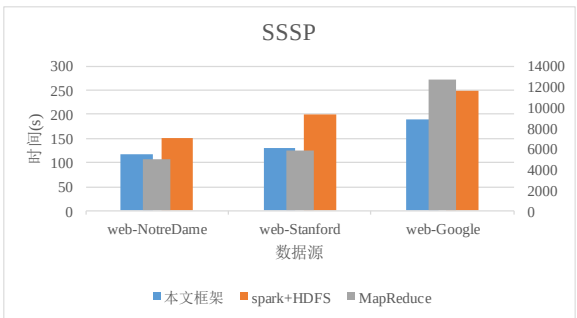

图 3-2 SSSP 算法实验结果（MapReduce 结果为次坐标轴）

表 3-1 实验数据集信息

	Google Web	Web NotreDame	Web Stanford
节点数	875713	325729	281903
边数	5105039	1497134	2312497
平均聚类系数	0.5143	0.2346	0.5976
三角形数目	13391903	8910005	11329473
闭三角分数	0.01911	0.03104	0.002889
直径	21	46	674
90%有效直径	8.1	9.4	9.7

实验数据均来自 SNAP(Stanford Network Analysis Platform)[30]。数据表示为：(IDA, IDB)，表示顶点 IDA 指向顶点 IDB。本文选取了 Google-Web、web-NotreDame 和 web-Stanford 三组实验数据集。表 3-1 给出了每个实验数据集的具体信息。其中平均聚类系数是衡量图中节点聚齐程度的参数。平均聚类系数越高的图，图中的节点就越密集，节点之间的关系也就越复杂。三角形数目表示无向网络中节点之间连接成三角形的数目。闭三角分数为三元组中的节点数除以度为 2 的路劲数。三角形数目和闭三角分数图能够反应图数据的紧密程度。直径表示最大的最短路劲长度。90%有效直径表示 90%的最短路劲分布程度。这三个图数据集当中 web-NotreDame 数据集和 web-Stanford 数据集的节点数较为接近。但是 web-Stanford 数据集中数据之间的关系复杂程度要比 web-NotreDame 数据集高出很多。Google-Web 数据集为较大并且较为复杂的数据集。

为了比较本文框架在计算层和存储层的性能，我们将本文框架与 MapReduce 框架和以 HDFS 为存储层的 Spark 框架分别进行 PageRank 算法和 SSSP 算法实验对比。PageRank 算法的迭代次数以及产生的中间结果都相对较多，所以我们选择 PageRank 算法来验证各个框架的图数据处理能力。而 SSSP 算法设计到复杂的查询操作，所以我们采用 SSSP 算法来验证存储层对图数据的操作性能。

实验开始前我们先将三个数据集存放到每个框架的存储层(本文框架为 Neo4j，其他两个框架为 HDFS)。为了得到更详细的实验结果分析，我们依次采用不同的数据集在每个框架下执行 SSSP 算法和 PageRank 算法。在不考虑集群故障因素下，每个实验重复执行 3 次，分别记录程序执行时间，最后取平均值。

2.3　实验结果及分析

实验结果如图 3-1、3-2 所示，图 3-1 显示了 3 个框架在执行 PageRank 算法时执行时间随着数据集规模变化的情况。从图 3-1 可以发现三个框架的执行时间都随着图数据规模的增加呈近线性增长。

web-NotreDame 数据集和 web-Stanford 数据集的节点数较为接近，但是 web-Stanford 数据集中数据之间的关系复杂程度

要比 web-NotreDame 高出很多。从这两个数据集的实验结果上来看，随着数据之间关系的复杂程度增高，本文框架在执行 web-Stanford 数据集计算时的时间花费只比执行 web-NotreDame 数据集的时间花费高出将近 10 秒，而以 HDFS 为存储层的 Spark 框架却高出了将近 40 秒。主要因为本文的图分割算法能够将复杂的图分割成相对均衡的子图。在计算期间减少了节点之间的通信开销，以及使得节点之间的计算任务相对平衡，这样就不会出现木桶短板现象。MapReduce 框架在这两个数据集上的时间耗费差到达了将近 500 秒，这主要你是由于 MapReduce 框架要将每次计算过程的中间结果放回磁盘，导致该框架在迭代计算上的性能太低。而本文框架的计算层是基于内存计算的框架，所以性能上要比 MapReduce 框架高出很多。

web-Stanford 数据集和 Google-Web 数据集都是数据之间关系比较复杂的数据集，但是 Google-Web 数据集的节点数量要比 web-Stanford 数据集大很多。从这两个数据集上来看，随着数据量的增加本文框架在执行 Google-Web 数据集时的时间耗费要比执行 web-Stanford 数据集时的时间耗费高出将近 30 秒。而以 HDFS 为存储层的 Spark 框架在执行这两个数据集时的时间耗费差达到了将近 60 秒。主要原因在于本文框架采用了基于内存的文件系统来对数据做缓存，以及能够将大图的分割成相对平衡的子图。在计算时节点之间的通信开销会很小。而 MapReduce 框架在这两个数据集上的时间耗费差达到了将近 3000 秒。随着迭代次数的增加 MapReduce 框架在图数据上的计算性能相对基于内存计算框架的性能要低很多。

图 3-2 显示在执行 SSSP 算法时，三个框架的执行时间都会随着数据集大小的增加而变长。

同样与 PageRank 算法实验对比类似，当数据集为 web-NotreDame 数据集和 web-Stanford 数据集时，随着数据集中的节点关系复杂度的提高，本文框架在两个数据集上的执行时间耗费差几乎相同，而以 HDFS 为存储层的 Spark 框架在执行 web-Stanford 数据集时的时间消耗要比在执行 web-NotreDame 数据集时的时间消耗要高将近 40 秒。因为本文框架中的存储层为图形数据库(Neo4j)，图形数据库具有良好的数据组织，存储结构以及索引机制，所以对于数据之间关系较为复杂的图数据具有良好的数据操作性能。采用 HDFS 为存储层的 Spark 框架要将图结构数据转化为元数据到图计算层操作，所以执行效率相对较低。MapReduce 框架在这两个数据集的执行耗时差将近 1000 秒，主要原因在于无论是从存储层到计算层的数据转换还是中间结果回放到磁盘上都增加了巨大的通信量，使得整个框架对图数据的操作性能很低。

对于 web-Stanford 数据集和 Google-Web 数据集，随着数据量的大幅度提升，本文框架在两个数据集上的执行时间消耗差将近 60 秒，而以 HDFS 为存储层的 Spark 框架在两个数据集上的时耗差将近 100 秒。主要原因在于本文框架在存储层采用的图形数据库是自适应规模的，而且它的图遍历执行速度是常数，与图数据大小无关。而 MapReduce 框架随着图数据的增加在图数据的操作上显得越来越慢。

3 结束语

本文针对图数据处理在计算层和存储层结合的效率低下问题，在 Spark 环境下提出了一种大规模图数据处理机制。该系统由一个图计算框架和一个图存储框架以及中间的一个结构数据缓存层组成。我们采取了相应的机制来解决数据抽取带来的瓶颈以及优化了对数据的查询操作。并且在分布式集群下对 MapReduce 框架、采用 HDFS 做存储层的 Spark 框架以及本文框架进行了 PageRank 和 SSSP 算法，实验结果表明本文框架在数据量增加时具有良好的稳定性，有效的减少了算法执行的时间。该框架可以完成针对较大数据的图计算和存储功能，且能够支持开发者自己的算法。此框架能够应用在社交网络计算、大数据挖掘以及大数据应用中。

未来工作主要关注在框架的容错功能，以及如何提升对内存的使用效率上。

参考文献

[1] Salihoglu S, Widom J. GPS: a graph processing system [C]//Proc of the 25th International Conference on Scientific and Statistical Database Management. New York: ACM Press, 2013: 1-22.

[2] 伍勇，钟志农，景宁，等. 海量图数据可视化研究[J]. 计算机应用研究，2012, 29(9): 3216-3220.

[3] 涂新莉，刘波，林伟伟. 大数据研究综述[J]. 计算机应用研究，2014, 31(6): 1612-1616.

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

[5] Lin J, Schatz M. Design patterns for efficient graph algorithms in MapReduce[C]//Proc of the 8h Workshop on Mining and Learning with Graphs. 2010: 78-85.

[6] Bu Y, Howe B, Balazinska M, *et al*. HaLoop: efficient iterative data processing on large clusters [J]. **Proceedings of the VLDB Endowment**, 2010, **3**(1-2): 285-296.

[7] HaLoop[EB/OL]. http: //code. google. com/p/haloop/.

[8] Malewicz G, Austern M H, Bik A J C, *et al*. Pregel: a system for large-scale graph processing[C]//Proc of ACM SIGMOD International Conference on Management of Data. 2010: 135-146.

[9] Seo S, Yoon E J, Kim J, *et al*. Hama: An efficient matrix computation with the MapReduce framework[C]//Proc of the 2nd International Conference on Cloud Computing Technology and Science. 2010: 721-726.

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

[11] 金伟健，王春枝. 适于进化算法的迭代式 MapReduce 框架[J]. 计算机应用, 2013, 33(12): 3591-3595.

[12] 梁秋实，吴一雷，封磊．基于 MapReduce 的微博用户搜索排名算法[J]. 计算机应用, 2012, 32(11): 2989-2993.

[13] Chebolu P, and Melsted P. PageRank and the random surfer model[C]//Procs of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2008: 1010-1018.

[14] Spark[EB/OL]. http: //spark. apache. org.

[15] Zaharia M, Chowdhury M, and Franklin M J, *et al*. Spark: cluster computing with working sets[C]//Proc of the 2nd USENIX Conference on Hot Topics in Cloud Computing. 2010: 10.

[16] Malewicz G, Austern M H, Bik A J C, *et al*. Pregel: a system for large-scale graph processing[C]//Proc of ACM SIGMOD International Conference on Management of Data. 2010: 135-146.

[17] Hadoop file system[EB/OL]. http: //hadoop. apache. org/.

[18] HBase[EB/OL]. http: //hbase. apache. org.

[19] Ghemawat S, Gobioff H, Leung S T. The Google file system[J]. **ACM SIGOPS Operating Systems Review**, 2003, **37**(5): 29-43.

[20] Neo4j[EB/OL]. http: //www. neo4j. org/.

[21] Seidman S B. Network structure and minimum degree [J]. **Social Networks**, 1983, **5**(3): 269-287.

[22] Angles R, Gutierrez C. Survey of graph database models [J]. **ACM Computing Surveys**, 2008, **40**(1): 1.

[23] Angles R. A comparison of current graph database models[C]/Proc of the 28th International Conference on Data Engineering. 2012: 171-177.

[24] Vicknair C, Macias M, Zhao Z, *et al*. A comparison of a graph database and a relational database: a data provenance perspective[C]//Proc of the 48th Annual Southeast Regional Conference. 2010: 42.

[25] Bi T, Ni Y, Shen C M, *et al*. An efficient graph partition method for fault section estimation in large-scale power network[C]//Proc of Power Engineering Society Winter Meeting. 2001: 1335-1340.

[26] Ching Avery. Giraph: Large-scale graph processing infrastructure on Hadoop[C]//Proc of the Hadoop Summit. 2011.

[27] Low Y, Bickson D, Gonzalez J, *et al*. Distributed GraphLab: a framework for machine learning and data mining in the cloud [J]. **Proceedings of the VLDB Endowment**, 2012, **5** (8): 716-727.

[28] R. Angles, C. Gutierrez. Survey of graph database models [J]. **ACM Computing Surveys**, 2008, **40**(1): 1-6.

[29] Renzo A R, Domingo G C. Survey of graph database models [J]. 2009.

[30] SNAP[EB/OL]. http: //snap. stanford. edu/.