文章编号:1000-5641(2014)05-0055-17

基于内存计算的大规模图数据管理研究

袁培森[1], 舒 欣[1], 沙朝锋[2], 徐焕良[1]
(1. 南京农业大学 信息科技学院, 南京 210095;
2. 复旦大学 计算机科学技术学院,上海 200433)

摘要: 图是一种重要的数据模型,能够描述结构化的信息,在诸如交通网络、社交网络、Web 页面链接关系等领域应用广泛,因而获得了广泛的研究. 海量的图数据管理对传统的图分析处理技术提出了挑战,分布式集群计算为大规模图数据分析提供了基础平台. 随着计算机硬件性价比的大幅提升以及高性能应用需求,基于内存计算的海量数据处理技术获得了业界青睐. 图数据高效存储和计算与内存计算密切相关,在此背景下,文章综述了大规模图数据处理相关技术进展,研究了典型的基于内存计算的大规模图数据管理系统,最后总结了基于内存计算的图数据管理的关键点.

关键词: 内存计算; 图数据; 分布式计算
中图分类号: TP391 **文献标识码**: A **DOI**:10.3969/j.issn.1000-5641.2014.05.005

Research of large-scale graph data management with in-memory computing techniques

YUAN Pei-sen[1], SHU Xin[1], SHA Chao-feng[2], XU Huan-liang[1]
(1. *College of Information Science & Technology, Nanjing Agricultural University, Nanjing* 210095, *China*;
2. *School of Computer Science, Fudan University, Shanghai* 200433, *China*)

Abstract: Graph is an important data model, which can describe structural information including dependent relationship, such as transportation network, social network and webpage hyperlink. Management of big graph brings challenges for traditional techniques, however, distributed cluster provide platform and techniques for this problem. Nowadays, the ratio of performance and price of memory promote rapidly, while demand of applications of high-performance, in-memory computing for massive data management is becoming popular. The storage and evaluation of massive graph requires high-performance platform. In this context, it's significant for studying graph data management with in-memory techniques. This paper surveyes key techniques

收稿日期:2014-06
基金项目:江苏省农业三新工程项目(SXGC[2014]309)
第一作者:袁培森,男,博士,讲师,研究方向为大数据分析与处理. E-mail: peiseny@njau.edu.cn.
第二作者:舒欣,男,博士,讲师,研究方向为机器学习和计算机视觉. E-mail: xinshu@outlook.com.
第三作者:沙朝锋,男,博士,讲师,研究方向为数据控据和机器学习. E-mail: cfsha@fudan.edu.cn.
第四作者:徐焕良,男,博士,教授,研究方向为农业信息化学与大数据技术.
E-mail: huanliangxu@njau.edu.cn.

of management of massive graph data, and researched graph data management of in-memory computing techniques, and finally summarizes the paper.

Key words: in-memory computing; graph data; distributed computing

0 引 言

图数据是一类重要的数据,可以描述丰富的信息及信息之间的依赖关系,是一种经典的数据建模工具,在社交网络、Web 数据超链接、交通网络等方面被广泛应用. 这些应用中的图包含了百万和亿万级别的顶点和边,如截至 2014 年第一季度 Facebook 包含了 12.3 亿个活跃用户,每个用户平均好友 130 个;Web 链接图顶点数达到 T 级,边的个数达到 P 级. 同时,图数据分析与处理技术在机器学习与数据挖掘中具有重要应用,例如信念传播算法[1]、随机优化[2]等. 鉴于图建模的灵活性和应用广泛性,大规模图数据的存储和分析处理技术成为近年来数据库等领域的研究热点,尤其是分布式集群计算的广泛应用,研究者提出了在集群上处理大规模图数据存储和分析处理技术[3-7].

分布式集群为海量数据处理提供了技术和平台,也给大规模图数据管理带来机遇. 大规模图数据在分布式集群上处理涉及的关键技术:① 图数据划分(partition),需要将大规模的图分割为相互独立的部分,进而根据一定的数据分布算法存储到集群节点上;② 计算模型,图中的信息存储在顶点或者边上,然而由于图数据的结构性质,数据之间存在依赖关系,图的计算模型一般涉及多次迭代,数据状态更新需要通过消息通信或者数据流. 图的划分是图数据管理的关键步骤,不仅与数据存储与数据均衡、负载均衡有关,还与计算节点间通信与数据移动量有关;同时计算模型关系到计算表达能力和粒度.

现有集群技术在处理大规模图数据时,其性能、计算表达能力等方面存在不足:首先,MapReduce[8]计算模式适合批式处理无依赖关系的数据,然而图的数据元素之间的存在依赖关系,难于表达图之间计算依赖关系. 第二,图的大部分算法需要多次迭代才能收敛,此外迭代过程产生大量的中间结果,需要在计算节点之间消息结果和移动数据. 文献[9]指出图数据计算过程需要多次随机访问数据,是典型的 I/O 密集型计算. 第三,此外,图中的遍历计算需在整个图上进行,数据访问缺少局部性,对性能优化带来了限制. 最后,对实时性要求不能满足,而大量的应用需要实时的获得分析的结果. 例如社交网络中好友关系分析、推荐系统等. 以上几个方面对直接使用现有集群管理大规模图数据提出了挑战.

近年来,内存的容量根据摩尔定律在发展,同时价格在大幅地下降,可以使得单机内存容量高达 TB 级,为海量数据的放入内存带来了可能. 最近,研究者提出的内存计算(In-Memory Computing,IMC)作为提升数据分析效率的有效技术发展迅速. 基于内存计算避免了 I/O 瓶颈,成为海量数据分析的利器,主要应用在 BI、ERP 、数据仓库等方面[28]. 典型的基于内存计算技术的数据分析产品为 SAP 的 HANA[10]. 目前,内存计算的研究已成为数据库等领域关注的主题之一[7,11],对于大规模图数据,研究者提出了在多核单机环境下和在分布式内存集群下大规模图数据管理的技术.

本文在大规模图数据管理需求和内存计算大发展背景下,研究了大规模图数据并行计算的编程模式、计算策略、图划分策略及计算同步等问题,接着介绍了内存计算相关的概念、设计理念和产品. 主要介绍了基于内存计算机制的图数据管理进展和典型的系统,总结了

基于内存计算的大规模图处理的关键，最后对本文进行了总结.

1　图数据管理

1.1　图数据表示

一般把图建模为二元关系模型，$G=(V,E)$，$V=\{v_1,\ldots v_n\}$ 为图 G 的非空顶点集合，$E=\{\langle v_i,v_j\rangle\}\subseteq V\times V$ 为由顶点集合构成的边的集合，如果边集合为顶点构成的有序对，称为有向图，否则称为无向图. 通常在构造图的过程中，图的顶点和边可以附带一些属性，例如权重、用户信息等. 文献[12]把图数据附带的属性考虑进去，提出了带属性图(property graph)，定义为 $G_P=(V,E,P)$，其中 $P=(P_V,P_E)$，P_V 和 P_E 分别为图 G 的顶点和边所附带的属性.

1.2　计算框架

图数据的分布式并行计算一般分为三个步骤：① 图划分，② 图数据分布与计算，③ 最终结果产生. 大规模图数据研究框架见图 1 所示.

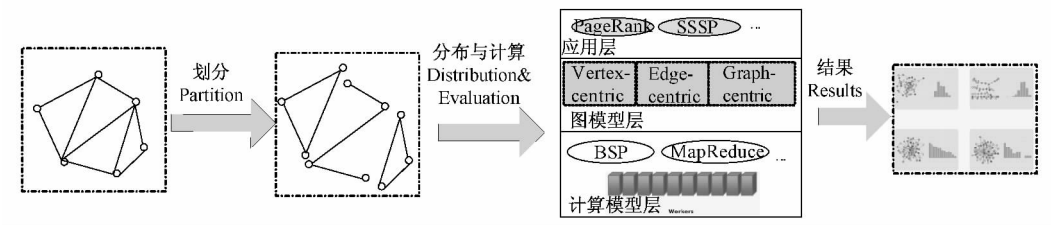

图 1　大规模图数据处理框架图

Fig. 1　Framework of processing on massive graph management

对于海量的图数据，首先是对图根据一定的规则进行划分，把图数据划分为若干个不相交的部分，进而把数据分布到集群上的计算节点. 典型的图分割算法包括：基于边的切割(p-way edge-cut)、基于顶点的切割(p-way vertex-cut)、基于随机的哈希方法和启发式的划分等. 图的划分质量对工作负载均衡、计算节点通信、存储和计算效率都有着极大影响.

第二步，将划分后的数据分布到计算节点上进行分布式存储和并行计算处理. 该步骤是图计算的核心，从逻辑上可以划分为三层：最上层的应用、中间的模型、底层物理层.

应用层的应用从实时性可以划分为在线查询和离线分析. 在线查询实时查询，一般无法预测查询与图结构的关系，而离线分析数据访问模式是可以预测. 从计算方法分三大类图计算：① 图的遍历，例如最短路径、连通分量计算等；② 随机行走，例如 PageRank[13]，HITS[14]等；③ 图聚集计算，例如图概要[15]、图粗化[16]等.

中间层是图计算模型层，该层次是图的计算策略，包括三种策略：① 以顶点为中心的策略(Vertex-centric)；② 以边为中心的策略(Edge-centric)；③ 以图为中心的策略(Graph-centric).

底层为物理层，包括计算同步策略，数据存储方式、编程框架和容错机制等. 同步策略包括 BSP(Bulk Synchronous Parallel)同步[4]、异步[17]和异步混合模式[3,17]等. 存储方法包括分布式文件、key-value 方式存储、数组、BigTable 等.

最后是计算最终分析结果.

2 图的计算

2.1 图的编程模式

根据图的结构性质,图的并行分布式计算基于以下两点:① 应用的数据及状态更新是在顶点或边上的迭代计算,计算直到计算状态收敛到一个固定点;② 计算的每一次迭代可以在图的顶点层面或边上并行独立进行. 计算中间结果扩展和聚集方法的不同可以把编程模式分为 SG 和 SAG 两种策略.

SG (scatter-gather)模式[22],该模式分为两个阶段:① scatter 阶段发送状态信息到顶点的邻接点与边;② gather 阶段应用更新信息到顶点上.

SAG 模式[3],该模式分为三个阶段:Gather、Apply 和 Scatter. Gather 阶段收集计算邻接点与边的信息,Apply 阶段将 Gather 阶段计算的结果应用于顶点,Scatter 阶段使用 Apply 阶段的更新值更新顶点的邻接边.

图 2 所示展示了两种编程模式数据更新的过程. 图 2(a)中的 SG 模式,首先顶点 u 通过 scatter 把更新数据传递给邻接点 v,接着顶点 u 获取以它为目标节点的更新. 图 2(b)的 GAS 模式中,顶 Gather 阶段获得点 u、边 $u \rightarrow v$ 和邻接点 v 的值的计算结果,Apply 阶段把该结果应用于顶点 u,scatter 阶段把顶点 u 的值更新到相应的边上.

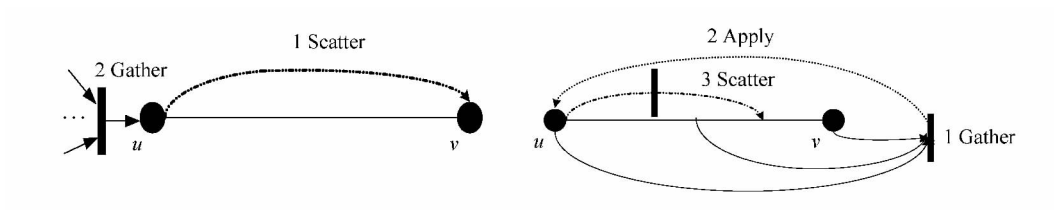

图 2 图的 SG 和 GAS 编程模式

Fig. 2 SG and GAS program model of graph

2.2 图的计算策略

图数据的计算包括两部分:图的顶点计算和图的边计算. 根据对图的处理视角的不同,可以把计算模型划分为三种:以顶点为中心、以边为中心的和以图为中心. 三种计算模型的表达能力和计算能力不同,各有优缺点.

为了形式描述图上的运算,把图数据的计算问题形式定义为 $P_X = \{G, Q\}$,其中 X 代表图 G 的元素,可为图的顶点、边或子图,$Q(X)$ 为计算在图 G 的元素 X 上的运算. 例如 X 为顶点时,$Q(X)$ 可以表示顶点上的 PageRank 计算.

(1) 顶点为中心. 以顶点为中心的图计算表示为 $P_V = \{G, Q\}$,其中 $Q(V)$ 在 G 中顶点集合 V 上运行的算法或者应用. 该模型把图数据的顶点作为处理对象,采用 SG 编程模型,利用图的边传递信息,P_V 的计算通过顶点与邻接点之间的边通信,进而在邻接点之间传递计算和状态信息. 该模型的特点是 scatter 和 gather 都在图的顶点上进行迭代.

顶点为中心实际上是把顶点作为独立的计算代理节点,计算代理节点相互独立地执行计算和通信任务. 不同的 P_V 相互独立,要求满足交换律和结合律,因此其执行顺序互不影响. 顶点为中心的计算模型可以解决诸如图挖掘,PageRank 计算等典型的图数据计算问题.

(2) **以边为中心**. 以边为中心的图计算问题表示为 $P_E = \{G, Q\}$，其中 $Q(E)$ 在图 G 的边集合 E 上的算法. 该方式同样在图的顶点中保存计算状态信息，并把计算分为 scatter 和 gather 阶段，每个阶段的顶点通过边进行更新状态. 以边为中心与以顶点为中心的计算过程如表 1 所示.

表 1　以顶点为中心和以边为中心的计算策略

Tab. 1　Evaluations of vertex-centric and edge-centric strategies

顶点为中心	边为中心
Scatter(vertex v) 　Send updates over outgoing edges of v Gather(vertex v) 　Apply updates from inbound edges of v While not done 　For all vertices v that need to scatter updates 　　Scatter(v) 　For all vertices v that have updates 　　Gather(v)	Scatter(edge e) 　Send updates over e Gather(update u) 　Apply updates u to u. destination While not done 　For all edges 　　Scatter(e) For all updates u 　　Gather(u)

(3) **以图为中心**. 以图为中心的图计算问题表示为 $P_{S_G} = \{G, Q\}$，其中 $Q(S_G)$ 在 G 中子图 S_G 上运行的算法或者应用.

以图为中心模型把顶点集划分为不相交的子集，子图由顶点、顶点链出的顶点以及边构成，每个子图构成一个划分. 该模型计算和同步的粒度为子图. 以图为中心的模型把子图中的顶点分为两类：内部(internal)顶点和边界(boundary)顶点. 同时每个顶点数据维护两个拷贝，其中内部顶点的数据为主拷贝，边界顶点的数据为本地拷贝. 把顶点分为主拷贝和本地拷贝该模型是与顶点为中心的计算模型的重要区别[18]. 内部顶点是构成子图的核心，内部顶点之间交换信息代价较小，但是边界顶点交换信息或改变状态需要在在节点之间消息传递，代价较大. 表 2 对比了三种计算策略的优缺点.

表 2　三种计算策略的优缺点对比

Tab. 2　The comparison of three evaluation strategies

三种计算策略	优点	缺点
顶点为中心	简介、易编程，应用广泛	收敛慢，计算在顶点间逐跳进行
边为中心	避免边数据随机访问	需要随机访问顶点，计算模型复杂
图为中心	在图划分较好时，图的连通计算性能较好，适合图的聚集计算	编程模型复杂，需要精心设计划分，易导致数据分布不均衡

2.3　图的并行策略

集群平台的图计算的并行典型策略有两种：一种不考虑数据依赖关系，称为数据并行(Data-parallel)；另一种是考虑图元素之间的依赖关系，根据依赖关系迭代计算和通信，称为图并行(Graph-parallel).

数据并行是把图数据作为相互独立的部分并行处理，通过把图数据划分为独立的部分，分布到集群各节点，在不同的分布节点上独立计算. 典型的为 Spark[7]，该类系统的优点是在分布节点上对数据在计算节点间的移动不做限制，扩展性好. 但是由于图分析迭代计算的性质，对数据并行系统提出了挑战，例如图结构、数据 Join 导致数据移动代价较高. 典型

的操作为 map,reduce,filter,join 等.

图并行策略是把根据图分割算法把图划分为具有依赖关系的部分,在具有依赖关系的数据上进行迭代并行计算,依赖部分的计算是通过在邻居节点迭代以及节点间的通信完成的. 在该并行策略下,典型的计算策略是以顶点为中心的计算. 该方法大幅提升了图并行处理的性能. 典型的系统为 Power Graph[3-5]. 但是该类型系统的计算表达能力不强,例如图构造、图结构更新、多图合并计算等. 两种并行策略的对比如表 3 所示.

以上两种并行策略各有优缺点,通过两者的结合,在图数据加载阶段采用数据并行,而在图数据分析阶段采用图并行,利用二者的优点. 典型的系统为 GraphX[6].

表 3 数据并行和图并行两种策略

Tab. 3 Two strategies of data-parallel and graph-parallel

两种策略	优点	缺点	典型系统
数据并行	数据自由分布,扩展性好	迭代计算代价高,数据移动量大,不适合依赖计算	Spark[7]
图并行	以顶点为中心的模型,计算依赖关系	表达能力有限,不适合多图比较与计算,对图的聚集计算也有限	PowerGraph[3] Pregel[4] GraphLab[5]

2.4 图划分

图划分是大规模图数据计算的重要操作,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[6]、流划分、谱划分(Spectral Partitioning)[35]等. 文献[19]研究了经典的图划分方法.

(1) 基于边的切割是沿着图的边划分,把顶点均匀地分布到 p 个计算节点,最小化边在计算节点的跨越. 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) 基于顶点的切割沿着中心顶点划分,把边均匀地分布到 p 个计算节点,该方法可以最小化存储可通信开销. GraphX[6]系统采用该方式对图进行分割. 图 3 显示了这两种切割.

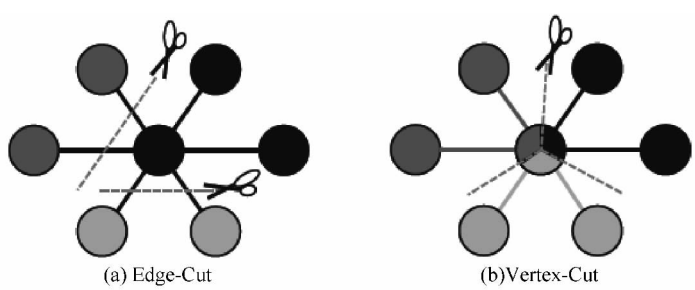

图 3 边和顶点切割[6]

Fig. 3 Edge-cut and vertex-cut[6]

(3) 基于随机的哈希方法对图的每个顶点指定一个 ID,通过使用 hash(ID)mod$p$ 把顶点均匀分布到 p 个结算节点. 随机划分方法能够快速、方便实现,且数据分布均衡[4-5],但由于没有考虑图的结构性质,导致计算节点间通信开销较高,收敛速度较慢.

(4) 启发式的划分以最小化数据复制为目标函数，找出划分的规则，根据规则确定每一条边的存储节点[3].

图划分的质量对在工作负载均衡、计算节点通信、存储和计算效率等都有极大影响. 优化划分的原则是：减少边跨越划分的个数，减少计算节点之间的通信，加快计算收敛速度. 图分割的难点在于真实世界的图一般符合幂率分布，这对分布式的工作平衡带来挑战，使得图难于均匀分割[3]；第二，基于 Hash 的图节点划分技术导致数据局部性非常差；第三，不平衡数据的分布导致计算节点间大量的通信开销；第四，度数高的节点对计算和存储的扩展性带来挑战. 此外，过于复杂的分割带来的计算开销也是必须要考虑的一个重要因素，基于哈希的分割简单易于实现，代价较小，应用也比较广泛[4-5].

2.5　图处理的同步策略

由于图结构依赖性导致其计算往往需要多次迭代，复杂性的结构使得达到稳定点计算步骤不同，需要在计算步之间进行控制. 常见的同步方式有：同步计算、异步计算和混合方式. 其中 BSP(Bulk Synchronous Parallel)[20] 是最常用的同步模型.

BSP 把计算划分为计算步(superstep)，模型采用异步计算同步迭代，每一个计算步的计算并行运行，在下一个计算步发起之前，需要等待前一个计算步的计算全部完成. 每个计算步可分为三个阶段：本地计算、通信和栅栏同步. 具体如图 4 所示.

本地计算时各计算节点的数据驻留内存，各结算节点相互独立；通信是在进程间通过 put 和 get 操作交换数据；栅栏同步等待所有进程计算通信完毕. 采用 BSP 同步计算结束的条件是消息处理完毕且所有计算投票表示停止. 使用 BSP 的系统典型有 Power Graph[3]，Pregel[4] 和 Graphx 系统[6] 等.

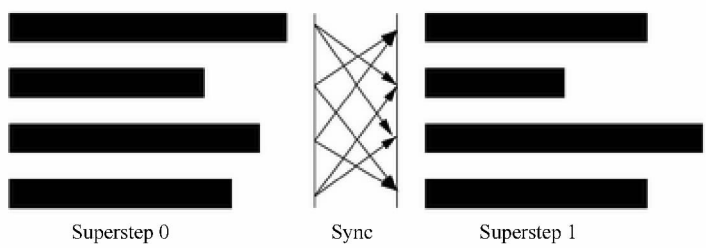

图 4　BSP 处理模型

Fig. 4　Processing model of BSP

BSP 同步处理确保计算的确定性和最大化并行性，用户易于设计，编程，测试和部署，并且具有良好的扩展性和加速比. 但是由于每个计算步的运算时间不同，最慢的计算将严重影响整体收敛速度，例如 PageRank、最短路径的计算[5].

为了最大化并行计算的效率，研究者提出了异步处理. 异步处理模式的优势在于可以通过计算的顺序智能排序，加速计算的收敛速度，但是异步处理模式编程复杂，不便于调试和测试，不能保证更新一致性，结果是不确定的，并发和隔离需要用户控制. 典型的有 GraphLab[5] 系统.

为了兼具 BSP 同步简便性和异步的高效性，GRACE[17] 把计算策略和应用逻辑区分开来，提出了图编程模型的同步迭代，在 BSP 运行时中兼容用户内建的异步计算运行. 此外，

文献[3]提出了异步串行化模式.

2.6 图处理的容错机制

集群上的图数据计算需要稳定可靠的处理环境,系统容错至关重要,尤其是内存计算环境下内存数据的易失性. 典型的容错机制包括:分布式检测点机制[3-5];分布式文件备份[27]等.

图的分布式检测点机制通过存储快照到磁盘或者 SSD,包括顶点、边的值、消息等. 在故障时通过使用最近的快照快速恢复,采用 BSP 同步机制的系统一般采用检测点机制容错[3-4,27]. 快照分为同步快照和异步快照. 同步快照在建立的时候需要暂停所有的计算,清空消息并把所有的修改写到持久存储上. 异步快照通过增量式构建而不需要暂停计算. GraphLab[5]支持同步快照和异步快照.

对于核心的数据,可以采用分布式文件备份,例如文献[27]把共享地址表在集群的文件系统中保留备份,地址表的更新提交之前首先要写入文件系统的备份中.

3 内存计算

3.1 内存计算的兴起

内存计算以其快速响应成为目前海量数据快速分析计算的利器. 内存容量的增加与价格的下降,以及硬件技术的成熟使得内存计算成为现实. 目前 T、P 级别的内存已经应用在数据分析领域. Gartner 预计 2015 年将有 35%的大中型企业使用内存计算,而在 2012 年还不足 10%.

内存计算不是最新提出的概念,但是近年来成为业界和研究领域的一个热点,它组合了硬件和软件最新技术. 技术发展和应用需求是两大推动力,一方面多核计算和 64 位计算系统普及和价格的下降,另一方面是大数据和 Web 等应用的兴起. 通过把数据装载到内存中,避免了 I/O 瓶颈,以前在数小时、数天时间内计算的结果在内存计算环境中,可以在数秒内完成. 在此高性能的计算背景下,内存计算再次成为业界和学界研究关注的热点.

在多核 CPU 演进、内存价格的不断下降,以及系统架构的不断演进下,内存计算技术将成为未来高性能计算的主流.

3.2 内存计算的核心及产品

在 IMC 方式下,所有的数据在初始化阶段全部加载到内存中,数据及查询的执行都在高速内存内执行,CPU 直接从内存读取数据,进行实时的计算和分析,避免了应用程序、服务器、网络硬件、储存设备、磁盘之间的数据交换,减少了许多网络与 I/O 的影响,大幅提升了计算处理的数据吞吐量与处理的速度,通常这部分开销占 90%的计算资源. 例如,对于内存数据库(In-memory database)来说,可以避免 I/O 密集的索引创建等开销[23].

内存计算目前尚未有统一的定义. Gartner 对内存计算的定义为[21]:一种应用平台中间件,实现分布式、可靠及可扩展性、强一致或最终一致性的内存 NoSQL 数据存储,可供多个应用共享.

内存计算不仅仅是把数据驻留内存,还需要对软件体系、计算模型等进行专门的设计[29]. 内存计算主要的设计理念是:① 将数据存放在内存中以加快处理的速度. ② 使用压缩技术减少数据量. 内存计算可以同一块存储空间连续存储,为数据进行压缩和顺序访问提供便利进行. 文献[22]通过实验,在磁盘、SSD 和内存三种存储介质上对比了顺序访问和

随机访问性能，顺序读效率分别大约提升 500 倍、30 倍和 4.5 倍. ③ 减少数据的移动，仅移动运算后的结果，而非搬移数据到运算. ④ 利用多核处理器，提高处理效率.

一般认为内存计算的内存是 DRAM，数据具有易失性，因此 IMC 环境下数据恢复管理与传统的系统差别很大，灾难恢复、数据备份、监控和管理都较难，尤其是分布式集群环境中.

目前，内存计算的业界推动者为 SAP，典型的产品为 IMC 数据库 HANA[25]，主要应用于 ERP 和 CRM 等. 支持高速事务处理的内存数据库 VoltDB[30]，同时 IBM 的 solidDB[31]、Oracle 的 Exadata X3、微软的 SQLServer 2012 已经引入了内存计算. 文献[24]详细研究了基于内存的高性能集群，指出硬件和操作系统技术的进步使应用全部内存中成为可能.

本文把内存计算分为三类：第一类是在多核单机上的多线程内存计算；第二类是集群环境中各计算节点的内存全局统一管理和访问的内存集群架构；第三类是集群的各计算节点独立管理本地内存，但是计算时全部数据装载到本地内存中.

4　基于内存计算的图数据处理方案

图计算分析是一种 I/O 密集型计算[9]，大部分的应用计算需要多次迭代，计算的状态信息需要在计算节点间消息传递和频繁更新，尤其是大规模的图数据，需要在集群的节点间频繁的消息传递和中间结果存储. 如果把数据全部在内存中计算，将极大地提高效率. 同时，文献[4,37]指出，现有的 MapReduce 模式不适合大规模图数据处理，原因一是 MapReduce 的优势在于并行处理无依赖关系的计算，对在有依赖关系的图数据很难大规模并行；二是 MapReduce 共享数据的唯一方式是把数据写到分布式文件中，增加了数据复制带来的 I/O，文献[32]指出该操作带来超过 90%的计算开销，图数据处理将产生大量的中间结果.

传统的在单机运行的图数据计算算法库，例如 LEDA[33]，扩展性不好，面对大规模的图数据计算能力不足；MapReduce 计算框架容错性、扩展性等方面较好，但是对于图计算效率不高[32]；现有的图并行处理系统，存在容错性等问题[34]. 因此，需要研究适合大规模图数据计算的内存集群管理计算技术.

为了提升大规模图数据计算的效率，研究者提出了基于内存的图处理系统[7,17,27]. 图的内存计算系统大致可以分为三种：第一种是基于内存分布式集群系统，例如 Trinity[27] 系统；第二种是基于内存共享的分布式系统，例如文献[5-7]所介绍的；第三种是在多核单机上多线程共享大内存系统，例如 GRACE[17,35-37]. 下面介绍几个典型的基于内存计算的图处理系统.

4.1　典型的系统

下面根据现有的基于内存计算的图数据计算系统分别进行介绍.

4.1.1　内存分布式集群

Trinity[27] 是一种数据存储和计算框架，采用内存分布式框架和内存 key-value 模式存储，集群中的节点的内存全局共享，提供在线计算和离线分析功能. Trinity 系统包含三类组件：Slave、Proxy 和 client. Slave 主要负责存储图数据和数据上的计算，Proxy 负责 Salve 与 Client 间的消息通信，Client 是用户与系统通过 API 交互的接口. 系统支持在线查询和离线分析.

Trinity 采用的内存集群采用内存全局共享模式，系统把内存划分为 2^p 个内存块，其中

$2^p > m$，m 为集群系统中的机器个数. 该内存管理模式可以在内存块级别增加并行性，并减少因单个哈希表冲突造成的性能下降. 为提高系统的容错性，Trinity 支持数据后台类似 HDFS 分布式文件备份.

Trinity 采用内存 key-value 存储，其中 key 为 64 位的全局唯一 ID，value 为任意长度的数据块，通过哈希方式访问 key-value 对. 具体访问方法如图 5 所示.

如图 5 所示，在集群全局内存空间数据通过 key 访问数据经三步：首先确定存储该 key-value 对的机器，可以通过 $i=\text{hash}(64\text{bitKey}) \bmod 2^p$ 获得所在的内存块 i；再通过查询全局地址表(Addressing Table)获得该内存块所在的机器；再次使用 key 在该机器上的内存块中访问 value.

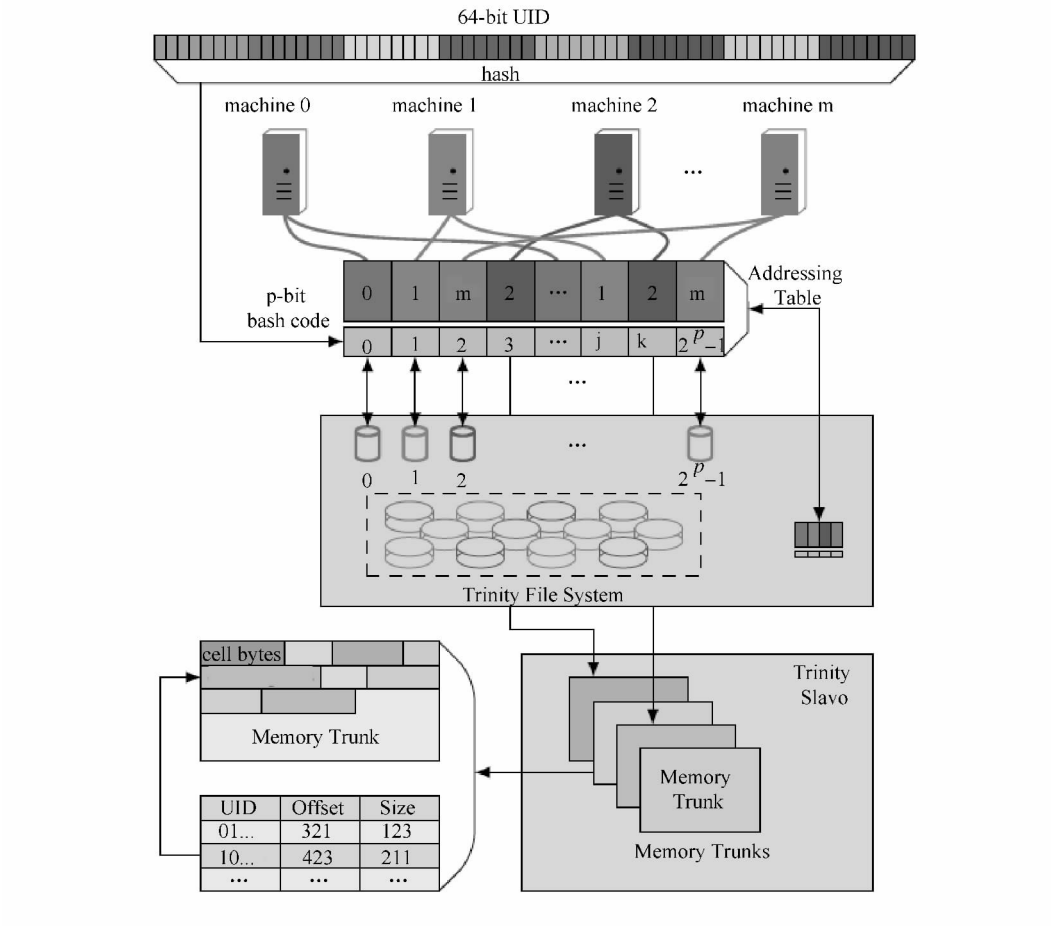

图 5 Trinity 系统中数据划分和访问[27]

Fig. 5 Data accessing and partitioning of Trinity[27]

在 Trinity 系统中，图的顶点和边元素建模为 cell，提供一种称为 TSL 的基于面向对象的语言. 在数据一致性方面，在 key-value 对上采用自旋锁，确保操作的原子性，但不保证并发序列化操作. 在计算策略选择方面，Trinity 同时支持 BSP 同步和异步模式，且其计算迭代的消息接收和发送是有选择性的. Trinity 对图的划分采用了二分图划分方式，极大地减少了通信开销. Trinity 在容错机制方面，采用分布式文件系统保存持久备份，对在线更新，

采用日志机制确保数据一致性.

Trinity 利用了内存支持快速随机访问和并行计算，支持高效的在线分析和基于顶点为中心计算模式的离线分析，其中离线分析采用有限制的顶点为中心计算模式，优化了消息传递和计算性能.

4.1.2　分布式内存共享的图处理

1. Spark[7]

Spark 是基于内存优化的内存计算引擎，用于多遍的迭代和交互计算. Spark 的核心概念是弹性分布数据集(Resident Distributed Datasets，RDDs)，系统把数据表示为弹性分布数据集. RDD 是只读的、具有容错性记录划分集合，它可以从稳定数据存储或者其他 RDD 上构建，允许用户在内存中保留中间结果和控制划分并提供操控操作原语.

系统在主从式集群上实现图的内存计算，提供了粗粒度数据并行操作的编程 API，操作原语包括两类：数据转换类(transformation)和行为类(action). 数据转换类用于定义 RDD，包括 map、filter、collect、Join、partitionByKey 等，行为类用于发起运算或保存结果，包括 count、collect、reduce、save 等. 待处理的数据存储可以存在 HDFS 或 Hbase 上，本地数据完全加载到本地节点的内存中，其运行和数据加载如图 6 所示. 任务的调度通过建立 RDD 世系图 DAG 分阶段执行.

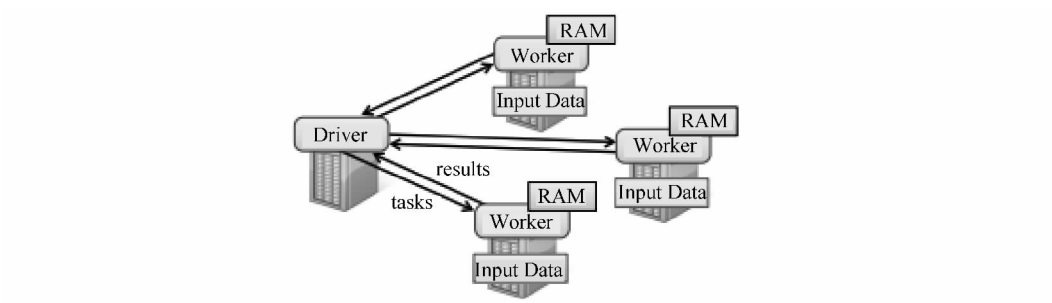

图 6　Spark 数据加载和运行时[7]

Fig. 6　Data load and runtime[7]

内存提供三种对象持久存储方式：内存反序列化 Java 对象，序列化数据和磁盘存储，但尚未实现集群节点内存的全局化统一管理. 三种方式在性能方面依次降低. 在数据一致性方面，Spark 支持检测点机制和 RDD 世系恢复. 数据划分和分布采用基于哈希的数据划分.

鉴于 Spark 粗粒度编程 API，使得它的编程能力有限. Spark 适用于在数据集上操作相同的批处理应用，而对更新类型应用效率不高.

GraphX[6] 系统是在 Spark 系统的基础上，除了提供数据并行操作，例如 map、reduce、filter 等之外，还提供了以边为中心图并行操作 API，例如 subgaph 等. 由于 GraphX 使用了索引、执行策略以及 Join 优化，使得 GraphX 的性能比 Spark 快一个量级以上.

2. GraphLab[5]

GraphLab 是采用分布式共享内存的图处理系统，即把整个图和程序状态存储在内存中，但是集群中的内存由本地计算机管理，每个本地计算节点采用多线程并发. 首先储存在分布式文件中的图被划分为 k 个部分，分别存储于各计算节点，每一个部分存储一个包括顶点和邻接边信息的文件，图的连通结构和 k 部分的数据位置信息存储在索引中. 索引用于

数据加载以确保数据划分均衡. GraphLab 系统视图如图 7 所示. 图中 GraphLab 的图处理分为两个阶段:初始化阶段和执行阶段. 初始化阶段主要完成数据解析、划分和创建索引. 执行阶段,数据文件从分布式文件系统加载到内存在 GraphLab 引擎上运行.

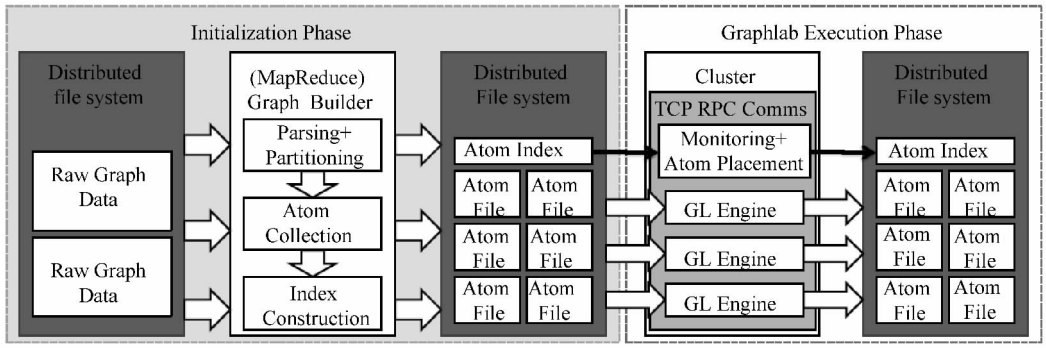

图 7 GraphLab 系统视图[5]

Fig. 7 System overview of GraphLab[5]

GraphLab 中图的结构是静态不可改变的. GraphLab 把图的计算抽象为三部分:数据图、更新函数和同步操作. 数据图是在顶点或边上关联用户数据的有向图. 更新函数可以表示为 $f(v,S_v)\rightarrow(S_v,T)$,函数的输入为顶点 v 以及由顶点 v 和 v 邻接的顶点和邻接边集合 S_v,函数更新 S_v 中元素的状态并返回下一次迭代将要更新的元素集合 T. GraphLab 同步和一致性分别通过着色引擎和分布式锁引擎来实现,支持三种一致性模型:完整一致性、边一致性和顶点一致性. 三者的并行性依次增强. 数据的容错则采用了分布式检测点机制.

3. Pregel[4]

Pregel 系统是工作在 Google 集群架构之上,采用分布式集群和 BSP 消息同步机制处理有向图. Pregel 把所有的计算状态驻留在集群中工作节点的内存,也是分布式内存计算的一种形式. 根据图计算的特点,把计算表达为迭代序列,计算序列之间通过图的顶点接收和发送消息或改变状态. Pergel 计算模型如图 8 所示,图中两个顶点 v 和 n,其计算包括接收其他顶点发送的消息,更新自身状态,更新边的状态,向邻接点发送消息.

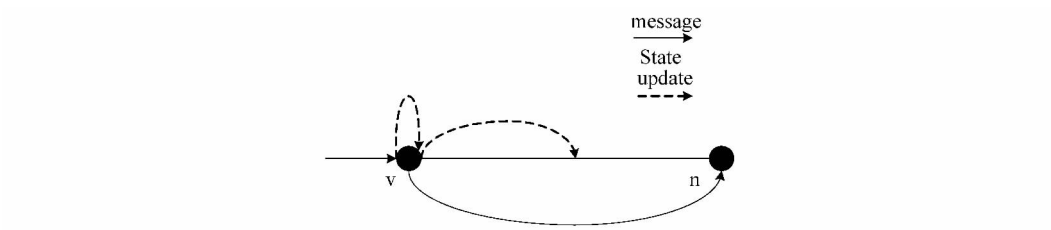

图 8 Pregel 计算模型[4]

Fig. 8 Computation of Pregel[4]

Pregel 的核心包括节点间消息传送、中间结果合并(Combiner)、全局结果聚合(Aggregator). 聚合函数要求满足交换律和结合律. 数据的划分采用哈希函数随机划分并支持用户定制的划分. 在容错方面 Pregel 采用检测点机制.

此外,Giraph[39]是一种在采用 Hadoop 分布式平台上处理大规模图数据的平台,系统与

Pregel 采用相似的设计，实质是 Pregel 的开源实现. Giraph 所有的计算在内存中进行，采用 ZooKeeper 同步.

4. PowerGraph[3]

PowerGraph 综合了 GraphLab 与 Pregel 的优点，采用共享内存结构，引入了 GAS 模型来表示图处理的过程，G 阶段在顶点 u 与邻居节点 v 和边 $e(u,v)$ 上进行计算，形式化记为：$\Sigma = \oplus_{v\in Nbr(u)} g(D_u, D_{(u,v)}, D_v)$，其中运算 ⊕ 要求满足交换律和结合律；A 阶段的任务是更新顶点 u 的值，记为：$D_u^{new} \leftarrow a(D_u, \Sigma)$；S 阶段使用新的值来更新顶点 u 的邻居节点，记为：$\forall v \in Nbr(u): (D_{(u,v)}) \leftarrow (D_u^{new}, D_{(u,v)}, D_v)$.

PowerGraph 把基于顶点的计算分解为边并行(edge-parallel)和顶点并行(vertex-parallel)两个阶段. 在多数情况下，节点上运行的计算并不涉及全部的邻居节点，PowerGraph 通过缓存 G 阶段的计算值而避免大量的计算. PowerGraph 同时支持同步和异步运行模式，试验表明异步模式较适合数据挖掘类的应用. 在异步处理时，PowerGraph 采用与 GraphLab 类似的序列化技术：为每一个顶点运行的程序定义相应的执行序列，通过锁技术控制邻居节点的运行顺序. PowerGraph 提供了同步、异步和异步＋串行模式，采用快照机制实现容错.

4.1.3　单机多核图处理

单机上的图计算多利用多核 CPU，采用大内存和多线程并行，一般为了充分发挥单机的计算效能，采取充分利用内存和 CPU 的 cache、优化磁盘读取等措施. 单机环境的图内存计算一般可以支持图的动态更新，但一般扩展性有限.

1. Graphchi[38]

Graphchi 是在多核单机系统上采用多线程和内存并行滑动窗口(Parallel Sliding Windows，PSW)技术. Graphchi 通过三个阶段：① 从磁盘加载图数据到内存块；② 更新顶点和边的值；③ 把更新写入磁盘. 首先把图的顶点划分为 P 个区间，对每一个顶点区间，关联一个用于存储以该区间内的顶点为终点的边的内存块(Shard)，区间的大小需确保所有的边都能加载到内存. 根基 PSW 的设计，区间 p 存储了该区间顶点的所有入边，该区间顶点的出边可以通过滑动($P-1$)个滑动窗口获得，如图 11 所示.

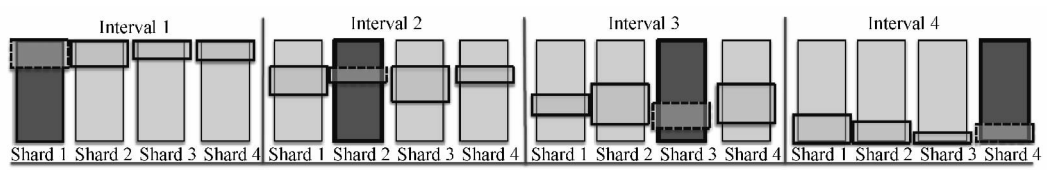

图 9　Graphchi 内存并行滑动窗口示意图[38]

Fig. 9　Parallel sliding windows illustration of Graphchi[38]

Graphchi 系统计算的每一次迭代需要顺序访问磁盘 P 次，因此对于一次计算需要 $O(P^2)$ 磁盘 I/O. 通过内存计算和磁盘操作并行，最大化单机上计算效率. Graphchi 采用异步计算模型，支持图结构更新，但对于图遍历访问效率不高，因为顶点邻接点的访问需要扫描所有的内存块.

2. Grace[35]

Grace 在多核单机上实现基于内存图数据管理，提供了从底层 cache、内存分配到高层

图查询和更新的 API 接口. Grace 采用以顶点为中心的数据更新模型,对图进行哈希和基于启发式的划分策略,每个物理核处理一个划分,各个计算内核采用 BSP 同步计算,其事务采用快照隔离技术支持事务级的图结构和数据更新.

Grace 内存的数据结构如图 12 所示. 内存数据结构主要有:① 顶点数组(Vertex Log),用于存储该划分内的所有顶点;② 边边指针数组,用于存储顶点的边集的位置;③ 边数组(Edge Log)用于存储边的度数和边集,每条边包括所在划分的 ID 和目标顶点的位置确定;④ 顶点的索引,用于在顶点单数组中查询;⑤ 顶点分配位图,用于指示顶点数组中的顶点是否有效.

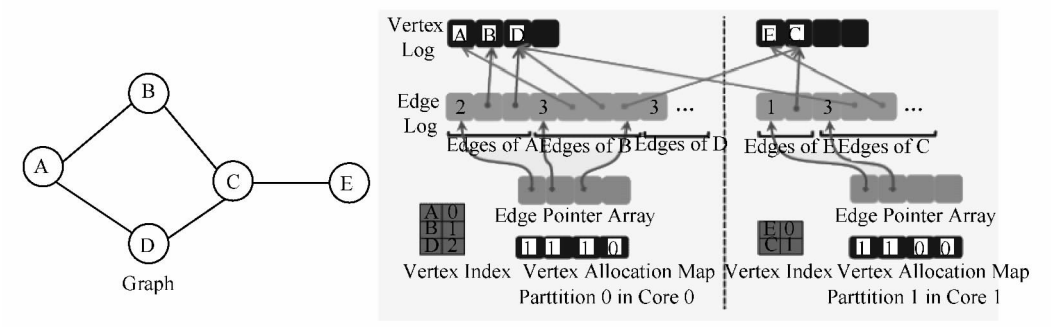

图 10 Grace 内存的数据结构[35]

Fig. 10 In-memory data structure of Grace[35]

Grace 中图的计算在多核之间迭代进行,实验表明 Grace 在多核系统上具有良好的扩展性能够和加速比.

综上所述,目前基于内存的图处理系统从计算策略、并行策略、计算同步以及容错处理等方面进行了研究和实验验证. 表 4 总结了以上介绍的代表性图处理系统.

表 4 代表性系统图处理系统对比

Tab. 4 The comparison of representative systems

内存计算方式	系统	并行策略	同步方式	划分	存储	容错性
内存分布式集群	Trinity	graph-Parallel	BSP 同步、异步	bipartite	key-value	分布式文件备份
	Spark	Data-Parallel	BSP 同步	随机哈希	HDFS	检测点机制
分布式内存共享的图处理	GraphLab	graph-Parallel	同步	随机哈希	分布式文件	分布式检测点机制
	Pregel	graph-Parallel	BSP 同步	随机哈希	关系数据库、文件或 Bigtable	检测点机制
	PowerGraph	graph-Parallel	BSP 同步和异步	随机哈希、edge-cut,vertex-cut、启发式 vertex-cut	HDFS	快照
单机多核系统	Graphchi	graph-Parallel	异步	Interval		
	Grace	graph-Parallel	BSP	哈希、启发式	数组	快照

4.2 图内存计算关键技术

综合基于内存计算的图数据管理技术进展,文章分析总结了基于内存的图处理系统的

研究关键，主要包括以下几个方面.

（1）内存分配与管理. 内存是内存计算的核心资源，内存分配与管理是内存计算的关键，如何在内存中存储和访问，是首先要解决的问题. Trinity[27]系统研究了集群中全局内存统一的访问与管理，采用 key-value 和自旋锁 key-value 数据固定在物理内存中. 文献[35]采用多种数据结构管理图的边和顶点，文献[38]通过并行滑动窗口机制，达到图的内存计算和 I/O 并行.

（2）图计算模式. 为了充分的利用内存计算数据随机访问的特点，需要研究新的计算模式. 以顶点为中心和以图为中心的计算两种策略对于图并行处理的效率差别很大，计算策略极大地影响计算效率和计算表达能力，SG 计算策略和 GAS 计算策略都是基于内存共享的集群，文献[27]指出在全局内存共享的环境上有进一步的优化空间.

（3）操作原语与优化机制. 图数据处理的性能优化及人性化操作原语，同时图的各种应用计算可通过一列的 Join 和聚集操作实现，研究适合内存计算的图操作原语，从物理层、计算层优化性能，进一步提升计算模型的计算表达能力. 现有系统缺乏统一的模型、优化机制和操作原语.

（4）同步机制. 目前大多系统采用 BSP 同步，部分系统采用异步机制. 内存环境下计算的效率远高于磁盘环境，消息的传递的网络开销就显得影响很大，要研究适合内存环境下图计算的同步机制. 此外，图并行处理的数据一致性和序列化研究较少.

（5）图的划分. 大规模图的划分是图并行处理的关键，众多研究和实验表明，图的划分的质量对计算效率、通信开销、负载均衡等都有极大的影响. 为了简化划分和易于实现，部分系统采用随机哈希技术，对于内存计算环境下的图划分优化，对性能的影响更为明显.

（6）容错处理. 内存数据的易失性，使得内存计算环境下数据恢复和容错至关重要. 文献[7]指出内存计算和存储的容错至关重要，目前容错处理主要有：① 数据复制. 内存计算的数据复制到分布式文件，将带来巨大的存储和通信开销；② 日志机制. ③ 分布式锁. ④ 快照.

总之，基于内存计算环境的大规模图数据计算获得了大量的研究成果，但是技术分散，模型尚不统一，系统各有优缺点，需要结合内存计算的特征，从物理层、通信层、模型层以及应用层设计整体的框架，整合各种资源，提供自动优化和便于操作的原语，支持图的结构更新和演化，提供在线查询和离线分析的统一计算平台.

4　结　　语

本文从大规模图计算的编程模式、计算和并行策略、图划分、计算同步等方面分析了大规模图数据并行处理的计算核心技术，研究了主流的基于内存计算的图处理系统进展，对比分析了典型系统核心功能和技术，总结了基于内存图处理系统关键技术，可作为大规模图数据管理研究的参考. 基于内存的图计算管理系统发展迅速，本文就典型的系统进行了介绍，没有包含所有的系统和技术，后期会继续跟踪相关技术的发展.

[参　考　文　献]

[1] GONZALEZ J, LOW Y, GUESTRIN C. Residual splash for optimally parallelizing belief propagation[C]//International Conference on Artificial Intelligence and Statistics. 2009: 177-184.

[2] SMOLA A, NARAYANAMURTHY S. An architecture for parallel topic models[J]. Proceedings of the VLDB Endowment, 2010, 3(1-2): 703-710.

[3] GONZALEZ J E, LOW Y, GU H, et al. PowerGraph: distributed graph-parallel computation on natural graphs [C]//Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation. 2012: 17-30.

[4] MALEWICZ G, AUSTERN M H, BIK A J C, et al. Pregel: a system for large-scale graph processing[C]// SIGMOD. 2010: 135-146.

[5] 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.

[6] XIN R S, GONZALEZ J E, FRANKLIN M J, et al. Graphx: A resilient distributed graph system on spark[C]// First International Workshop on Graph Data Management Experiences and Systems. 2013.

[7] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. 2012: 2-2.

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

[9] LUMSDAINE A, GREGOR D, HENDRICKSON B, et al. Challenges in parallel graph processing[J]. Parallel Processing Letters, 2007, 17(01): 5-20.

[10] FÄRBER F, CHA S K, PRIMSCH J, et al. SAP HANA database: data management for modern business applications[J]. ACM Sigmod Record, 2012, 40(4): 45-51.

[11] PLATTNER H. A common database approach for OLTP and OLAP using an in-memory column database[C]// SIGMOD. 2009: 1-2.

[12] ROBINSON I, WEBBER J, EIFREM E. Graph databases[M]. O'Reilly Media, Inc., 2013.

[13] BRIN S, PAGE L. The anatomy of a large-scale hypertextual web search engine[C]// WWW, 1998: 107-117.

[14] KLEINBERG J M. Authoritative sources in a hyperlinked environment[J]. Journal of the ACM (JACM), 1999, 46(5): 604-632.

[15] TIAN Y, HANKINS R, PATEL J M. Efficient aggregation for graph summarization[C]// SIGMOD, 2008: 419-432.

[16] KARYPIS G, KUMAR V. A Coarse-Grain Parallel Formulation of Multilevel k-way Graph Partitioning Algorithm [C]//PARALLEL PROCESSING FOR SCIENTIFIC COMPUTING. SIAM. 1997.

[17] WANG G, XIE W, DEMERS A J, et al. Asynchronous Large-Scale Graph Processing Made Easy[C]//CIDR. 2013.

[18] TIAN Y, BALMIN A, CORSTEN S A, et al. From "think like a vertex" to "think like a graph"[J]. Proceedings of the VLDB Endowment, 2013, 7(3):193-204

[19] KARYPIS G, KUMAR V. Multilevel *k*-way Partitioning Scheme for Irregular Graphs[J]. Journal of Parallel and Distributed Computing, 1998, 48(1): 96-129.

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

[21] GARTNER Says In-Memory Computing Is Racing Towards Mainstream Adoption[EB/OL]. 2013[2014-07-01]. http://www.gartner.com/newsroom/id/2405315.

[22] ROY A, MIHAILOVIC I, ZWAENEPOEL W. X-Stream: edge-centric graph processing using streaming partitions[C]//Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013: 472-488.

[23] DEFINITION in-memory database [EB/OL]. 2013[2014-07-01]. http://whatis.techtarget.com/ definition/in-memory-database.

[24] OUSTERHOUT J, AGRAWAL P, ERICKSON D, et al. The case for RAMClouds: scalable high-performance storage entirely in DRAM[J]. ACM SIGOPS Operating Systems Review, 2010, 43(4): 92-105.

[25] FÄRBER F, CHA S K, PRIMSCH J, et al. SAP HANA database: data management for modern business applications[J]. ACM Sigmod Record, 2012, 40(4): 45-51.

[26] CDH 100% Open Source Distribution including Apache Hadoop. [EB/OL]. 2014[2014-07-01]. http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html.

[27] Shao B, Wang H, Li Y. Trinity: A distributed graph engine on a memory cloud[C]//Proceedings of the 2013 international conference on Management of data. 2013: 505-516.

[28] KANE F. Why in-memory computing is going mainstream. [EB/OL]. 2013[2014-07-01]. http://www.information-age.com/technology/information-management/123457007/why-in-memory-computing-is-going-mainstream.

[29] In Memory Databases: HANA, Exadata X3 and Flash Memory. [EB/OL]. 2012[2014-07-01]. http://flashdba.com/2012/10/10/in-memory-databases-part2/.

[30] STONEBRAKER M, WEISBERG A. The VoltDB Main Memory DBMS[J]. IEEE Data Eng Bull, 2013, 36(2): 21-27.

[31] LINDSTRÖM J, RAATIKKA V, RUUTH J, et al. IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability[J]. IEEE Data Eng Bull, 2013, 36(2): 14-20.

[32] ZAHARIA M, CHOWDHURY M, DAS T, et al. Fast and interactive analytics over Hadoop data with Spark [C]// USENIX, 2012,37(4):45-51

[33] LEDA algorithmic. [EB/OL]. 2014[2014-07-01]. http://www.algorithmic-solutions.com/leda/.

[34] GREGOR D, LUMSDAINE A. The parallel BGL: A generic library for distributed graph computations[J]. Parallel Object-Oriented Scientific Computing (POOSC), 2005,2:1-18

[35] PRABHAKARAN V, WU M, WENG X, et al. Managing Large Graphs on Multi-Cores with Graph Awareness [C]//USENIX Annual Technical Conference. 2012: 41-52.

[36] JOUILI S, REYNAGA A. imGraph: A distributed in-memory graph database[C]//Social Computing (SocialCom), 2013: 732-737.

[37] LOW Y, GONZALEZ J, KYROLA A, et al. Graphlab: A new framework for parallel machine learning[J]. arXiv preprint arXiv:1006.4990, 2010.

[38] KYROLA A, BLELLOCH G, GUESTRIN C. Graphchi: Large-scale graph computation on just a pc[C]// OSDI, 2012, 8: 31-46.

[39] Welcome to Apache Giraph! [EB/OL]. 2014[2014-01-28]. https://giraph.apache.org/.

(责任编辑　李　艺)