Gemini：以计算为中心的分布式图计算系统

朱晓伟　陈文光
清华大学

关键词：图状结构数据　图计算分布式系统

图状结构数据

图 (graph) 是用于表示对象之间联结关系的抽象数据结构，通常使用顶点 (vertex) 集和边 (edge) 集进行描述：顶点表示对象，边表示对象之间的关系。根据边是否有方向和权值，图又可细分为有向图 / 无向图以及有权图 / 无权图等。当边有方向时，我们将一条边连接的两个顶点按方向分为源点 (source vertex) 和终点 (destination vertex)。图 1 展示的是一个简单的包含了 10 个顶点和 16 条边的有向无权图，其中 0 和 1 分别是 0 → 1 这条边的源点和终点。

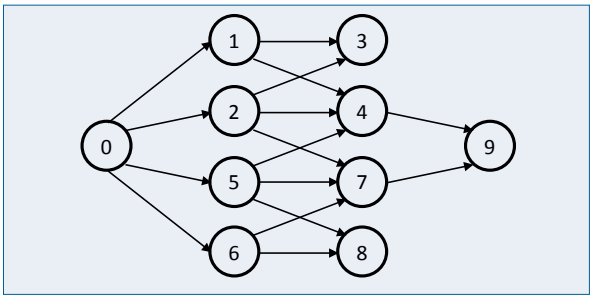


图1　有向无权图示例

对于可抽象成用图表示的数据，我们通常称之为图状结构数据 (graph-structured data)。随着互联网的飞速发展，这类数据正在受到越来越多的关注：社交网络中用户间的关注和互动、互联网上网页之间的链接、电子商务平台上用户对物品的评分记录等都是典型的图状结构数据。在天体物理学、计算化学、生物信息学等自然科学领域，图状结构数据也是无处不在。对这些数据的分析都离不开基于图的算法设计，以及面向图的高性能计算系统。

尽管图在数学上可以对应成矩阵，然而图状结构数据的一些特点让我们很难将实现传统科学计算应用所得到的经验直接移植过来：现实世界的图通常平均度数（即边数与顶点数的比值）只有几到几百，与上千万甚至上亿个顶点的规模相比显得极为稀疏，且度数呈幂律分布；而科学计算中我们经常面对的是稠密矩阵，或是较为规则的稀疏矩阵，数据的划分和并行较为容易。

另一方面，基于图的算法通常采用迭代式计算，同时，并非每一轮迭代都需要所有顶点参与计算，且活跃的顶点集不断变化的特点也使得图状结构数据的处理更加复杂，导致通用的大数据处理系统难以有效应对图计算问题。例如，MapReduce[1] 在每轮迭代计算中都需要反复读写磁盘，产生大量不必要的开销；而 Spark[2] 的数据模型 RDD 由于其不变性 (Immutable)，产生的大量中间结果会导致不必要的内存占用从而影响能够处理的数据集大小和数据处理的效率。

图计算系统的现状

为了更有效地解决大规模图上的计算问题，学术界与工业界提出了大量面向图优化的计算系统。Pregel [3] 是来自谷歌的大规模图计算系统的开山之作，很多后续的图计算系统均借鉴了其中的核心思想，例如“以顶点为中心”(vertex-centric) 的编程模型，让用户将计算过程抽象为基于顶点的计算和基于边的消息传递 (message passing)；整体同步并行计算模型 [4](Bulk Synchronous Parallel, BSP)，顶点之间并行处理（计算和通信），通过超步 (super-step) 之间的栅栏 (Barrier) 来同步计算过程。Giraph[5] 是 Pregel 的一个开源实现，脸书 (Facebook) 内部已进行了大规模的部署与应用。

GraphChi[6] 和 PowerGraph[7] 是来自卡内基梅隆大学 (CMU) 团队的工作。前者是基于单机外存的图计算系统，通过容量更大的外存来扩展能够处理的图的规模；后者则是面向分布式内存的解决方案，通过使用更多的机器来实现类似的目的。PowerGraph 的一个重要贡献是提出了基于“顶点切割”(vertex-cut) 的图划分思想，通过在不同机器上创建顶点的多个副本 (replica)，以主 - 从 (master-mirror) 副本间的同步来替代传统的沿着边传递消息的通信模式，有效地减少了通信量以及由度数较高顶点导致的负载不均衡。后续的很多分布式图计算系统如 GraphX[8]、PowerLyra[9] 等均沿用了 PowerGraph 的处理模型。

然而，目前的分布式图计算系统在获得了扩展性的同时却显得较为低效，使用了上百核集群资源的分布式程序甚至不如精心优化的单线程程序 [10]。另一方面，扩展性却又至关重要：在处理规模较大的数据时，我们不得不通过多台机器的内存来容纳需要处理的图 [1]。

Gemini

那么，能否实现一个兼具扩展性和高性能的分布式图计算系统呢？经过对几大主流图计算系统的性能分析，我们发现现有分布式系统的瓶颈并不在于其着重优化的通信，而是在于计算部分，具体表现为远超单机系统的 CPU 指令数、访存次数、缓存未命中 (cache miss)，以及较为明显的多核间负载不均衡。而计算部分的低效，主要来源于两方面：过大的分布式开销和非最优的本地计算实现。

为此，我们设计并实现了 Gemini，一个以计算为核心优化目标的分布式图计算系统。考虑到目前 Infiniband 等高速网络的发展趋势，以及图计算中消息传递与本地处理的可重叠性，我们将计算而非通信作为主要关注点，一方面尽可能避免分布式带来的开销，另一方面吸纳现有单机系统的技术来实现高效的计算。

双模式计算引擎

图计算根据信息的流动方向有两种典型的处理模式：推动 (push) 模式和拉动 (pull) 模式。使用推动模式时，每个参与计算的顶点沿着出边传递消息；而拉动模式则相反，让所有顶点沿着入边从邻接顶

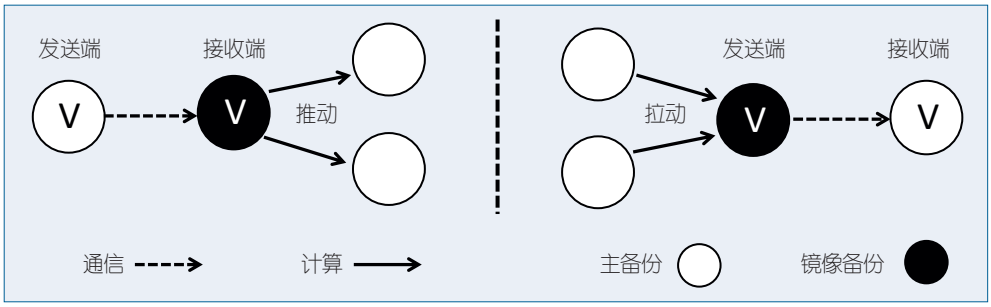


图2　双模式计算引擎

[1] 尽管面向单机外存的图计算系统也可以处理规模超过单机内存的图状结构数据，但是硬盘与内存之间巨大的性能差距使得外存系统在处理超大规模数据时依然捉襟见肘。

点获取消息。两种模式各有利弊：推动模式可以实现选择性调度 (selective scheduling)，从而在从活跃顶点出发的边较少时跳过那些不需要参与计算的边，不利之处则是需要用锁或原子操作来保证并发环境下数据修改的正确性，引入了额外的开销；拉动模式的优势在于数据的修改没有竞争，而对应的劣势则是必须查看所有边，即使很多时候大多数边并不参与计算。显然，将两种模式有机地结合起来才是最优选择。Ligra[11]、Galois[12]、Polymer[13] 等单机图计算系统均采用了这种混合策略，根据需要参与计算的边数自适应地在两种模式间切换。

Gemini 将这种双模式计算引擎从单机的共享内存扩展到了分布式环境中。我们进一步将两种模式下的计算过程都细分成发送端和接收端两个部分，从而将分布式系统的通信从计算中剥离出来。图 2 展示了 Gemini 中推动模式和拉动模式的示意图。为了便于说明，我们引用了 PowerGraph 的术语，将顶点分为主备份和镜像备份，即图中的白色和黑色顶点。其中，主备份存储了顶点的数据，每个顶点有且仅有一个主备份；镜像备份则不包含任何实际数据，仅仅作为消息传递的媒介。图中的实线表示计算所需要访问的边，虚线则表示主 - 从备份之间发生的通信（箭头代表了通信的方向）。

使用推动模式时，每个主备份先向所有备份广播传递需要的消息；各个备份（包括主备份）在接收到消息后，沿着出边更新邻接顶点的状态和数据；使用拉动模式时，所有备份先沿着入边从邻接顶点获取信息并进行本地的局部计算，然后将消息发送至主备份，主备份根据收到的消息更新状态和数据。

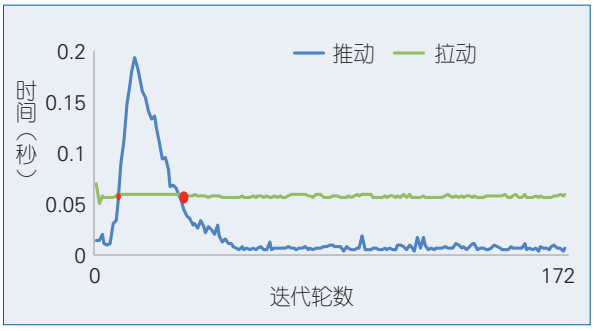


图3　每轮迭代计算使用推动/拉动模式的运行时间对比

双模式计算引擎为 Gemini 带来了颇为可观的性能提升。如图 3 所示，两种模式在不同情况下互有胜负，而 Gemini 通过自适应地在两种模式间切换，在绝大多数情况下能够使用较优的选择（图 3 中的红色部分为 Gemini 做出“误判”所损失的时间，与总运行时间相比几乎可以忽略）。

块式划分

图数据的划分是所有分布式图计算系统的核心。然而，已有的划分方法主要关注负载均衡和通信代价，往往忽视了因此产生的系统复杂度和对计算效率的影响。

Gemini 采用了一种十分简单的划分方法：将顶点集进行块式划分，将这些块分配给各个节点，然后让每个顶点的拥有者（即相应节点）维护相应的出边 / 入边。图 4 展示了将图 1 所示的有向图划分给两个节点的结果：10 个顶点中的 0~4 分配给左边的节点，5~9 分配给右边的节点。图中的白色顶点表示拥有的顶点（即主备份），黑色顶点则为镜像备份。

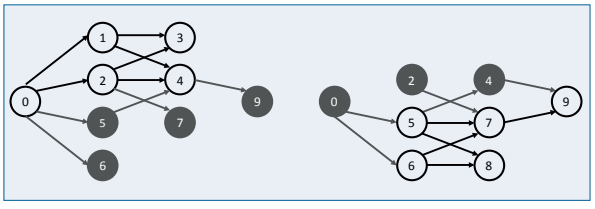
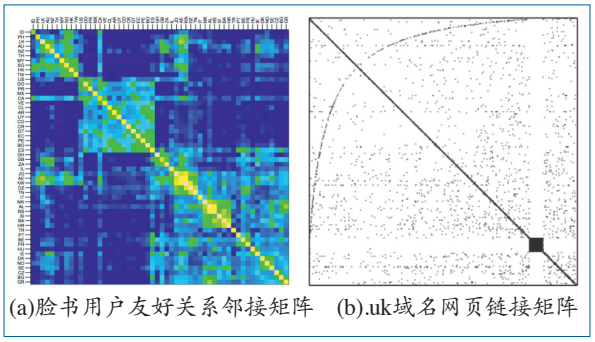


图4　块式划分示例

这种划分方式看似简单，但是应用在很多现实世界的图上却格外有效。这是由于很多图数据本身蕴含了局部性，而块式划分很好地保留了这些特征。图 5 展示了其中的两个例子。图 5(a) [14] 将脸书的用户按照所在国家聚集起来，矩阵中各个格子的亮度反映了两个国家间用户的好友关系数量，而其中绝大多数（超过 84%）的边都在对角线上（即双方在同一国家）。图 5(b)[15] 展示了将抓取的 .uk 域名的网页按照统一资源定位符 (URL) 排序后的链接矩阵，可以看到灰度值较高的区域集中在对角线，说明绝

大多数链接指向了 URL 相近的网页（主要为站内链接）。



(a)脸书用户友好关系邻接矩阵　(b).uk域名网页链接矩阵

图5　图状结构数据中蕴含的局部性

不仅如此，块式划分的另一大优点是极小的分布式开销。其他划分方法依据策略的不同极有可能将连续的顶点划分到不同节点上，通常需要在各个节点上维护映射表，将每个节点负责的顶点编号从较为分散的大区间映射到紧凑的小区间内。块式划分则由于所有节点负责的均是连续的一块顶点，从而免去了顶点编号转换的开销。

块式划分由于块内顶点的连续性，还可以自然地“递归”应用到更细粒度上。例如，现有的多处理器通常呈现非一致内存访问效应 (Non-Uniform Memory Access)，尽管共享所有的内存，但是处理器访问不同区域的内存会有不同的延迟和带宽，对于各个处理器而言，有本地内存和远程内存的区别。因此，Gemini 在每个节点的多路处理器间继续进行块式划分，可以有效减少远程内存的访问比例。

传统的图划分方法根据划分对象为顶点或边进行负载均衡，让每个节点分配到的顶点数或边数尽可能相近。然而，我们发现这两种策略应用在块式划分上均有不同的缺陷。按顶点数均衡可能导致边数失衡，而让边数均衡时顶点数又会有潜在的较大差异。两者都会影响计算效率：边数决定了计算量，而顶点数决定了单次计算（即随机内存访问）的复杂度。因此，Gemini 采用了一种混合的策略，让每个节点分配的顶点数和边数的加权和尽可能相近。

除了节点间的负载均衡，Gemini 在节点内也通过细粒度的块式划分结合工作偷取 (work stealing) 的方式，使得多核间的负载尽可能均衡。

性能比较

首先我们通过实验比较了 Gemini 与面向单机共享内存的图计算系统 Ligra 和 Galois 的性能，如表 1 所示。实验平台是 1 台双路 Intel Xeon E5-2670 v3 的服务器，配有 128GB DDR4 内存；选择的测试数据为 twitter-2010，包含 4165 万个顶点和近 15 亿条边；选择的测试程序为 5 个常用的图计算应用，PageRank(PR)、连通分量 (CC)、单源最短路径 (SSSP)、广度优先搜索 (BFS) 和介数中心度 (BC)。

表1　Gemini与单机系统的性能比较

应用	Ligra	Galois	Gemini
PR	21.2	19.3	**12.7**
CC	6.51	**3.59**	4.93
SSSP	**2.81**	3.33	3.29
BFS	**0.347**	0.528	0.468
BC	2.45	3.94	**1.88**

三个系统在不同应用上的表现各有千秋。Gemini 在 PR、BC 上表现最好，其余应用略慢于最快者。Gemini 的劣势主要来源于一些不可避免的分布式实现所带来的开销，例如额外的用于消息收发的指令和访存，以及分布式内存环境下慢于共享内存（顶点数据可随时更新）的收敛速度。尽管如此，Gemini 尽可能地减少了分布式的开销，使得运行于单机时的效率能够接近甚至超过现有最佳性能的单机系统。

另一方面，作为一个分布式系统，Gemini 可以通过集群扩展能够处理的图状结构数据规模，从而完成很多单机系统无法实现的任务。表 2 展示了

表2　Gemini在大规模图上的性能

应用	PR	CC	SSSP	BFS	BC
时间（秒）	31.1	25.7	56.9	10.2	45.3

Gemini 使用 8 节点的基于 Infiniband EDR 网络的集群，处理的规模包含近 10 亿个顶点、420 亿条边的 clueweb-12 时的性能。该图的原始二进制格式边表文件需要占据 318 GB 的存储空间，超出了实验平台单机内存的大小。而使用 Gemini 运行的这 5 个应用，计算过程均可以在一分钟内完成。

图 6 展示了 Gemini 与 PowerLyra 使用不同数量的节点在 uk-2007-05 图上计算 PageRank 和连通分量时（以 Gemini 的 8 节点运行时间作为基准）的相对性能。两者间的性能有接近 40 倍的差距，且 PowerLyra 由于更大的分布式开销需要占用更多的内存，因此需要 2 台以上的机器才能处理该数据。

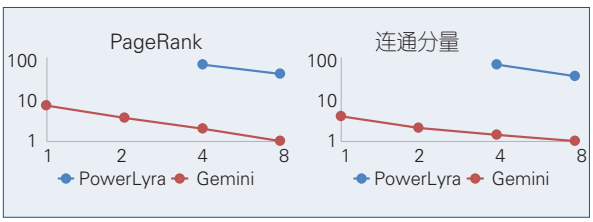


图6　Gemini与分布式系统的扩展性对比

结语与展望

图状结构数据的处理在各个领域均有广泛的应用，而面向大规模图数据的高性能计算系统则是大数据时代不可或缺的重要工具。单机图计算系统欠缺扩展性，而现有的分布式系统往往在具备扩展能力的同时缺乏令人满意的性能。通过在设计上避免过大的分布式带来的开销并尽可能地优化本地计算部分的实现，Gemini 将两类系统间的缝隙弥合起来，让分布式系统能够在保留单机系统高效性的基础上具备扩展性，从而节省部署多套系统的成本。

在 Gemini 的实践中，我们得到了一个重要的启示：分布式系统中计算部分的重要性会随着高速互联网络的不断发展与普及愈发凸显出来，很多现有的分布式系统设计经验需要被重新审视。计算的优化与通信同样重要，因此两者间的取舍将在分布式系统中扮演越来越重要的角色。 ■

朱晓伟

清华大学计算机科学与技术系高性能计算所博士生。主要研究方向为并行计算和分布式计算。
coolerzxw@gmail.com

陈文光

CCF副秘书长、理事、杰出演讲者，曾任*CCCF*编委、YOCSEF主席（2011~2012年度）。清华大学教授，兼任青海大学计算机系主任。主要研究方向为并行计算的编程模型、并行化编译和应用分析。
cwg@tsinghua.edu.cn

参考文献

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

[2] Zaharia M, Franklin M J, Ghodsi A, et al. Apache Spark: a unified engine for big data processing[J]. *Communications of the ACM*, 2016, 59(11): 56-65.

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

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

[5] Giraph[OL]. http://giraph.apache.org.

[6] Kyrola A, Blelloch G, Guestrin C. GraphChi: large-scale graph computation on just a PC disk-based graph computation[C]//*Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implemenation*. 2012:31-46.

[7] Gonzalez J, Low Y, Gu H. Powergraph: distributed graph-parallel computation on natural graphs[C]//*Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implemenation*. 2012: 17-30.

[8] Gonzalez J E, Xin R S, Dave A, et al.GraphX :graph processing in a distributed dataflow framework[C]//*Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implemenation*. 2014: 599-613.

[9] Chen R, Shi J, Chen Y, et al. PowerLyra: differentiated graph computation and partitioning on skewed graphs[C]//*Proceedings of the Tenth European Conference on Computer Systems*. New York: ACM Press, 2015:1.

[10]Mcsherry F, Isard M, Murray D G. Scalability! But at what COST ?[C]//*Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems* (HotOS XV), 2015:14.

[11]Shun J,Blelloch G. Ligra: a lightweight graph processing framework for shared memory[C]//*Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. New York: ACM Press, 2013: 135-146.

[12]Nguyen D, Lenharth A, Pingali K. A lightweight infrastructure for graph analytics[C]//*Proceedings of the 24th ACM Symposium on Operating Systems Principles*. New York: ACM Press,2013:456-471.

[13]Zhang K, Chen R, Chen H. NUMA-aware graph-structured analytics[C]//*Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. New York: ACM Press,2015:183-193.

[14]Ugander J, Karrer B, Backstrom L, et al. The anatomy of the Facebook social graph.arXiv preprint arXiv:1111.4503, 2011.

[15]WebGraph datasets[OL]. http://law.di.unimi.it/datasets.php.