大规模分布式图计算综述

管仁阳

摘要

本文是对分布式图计算框架的综述，总结了分布式图计算的难点，分析了分布式图计算框架的发展趋势，并归纳出主要的技术瓶颈与优化方向。

关键词：分布式，图计算

1 图计算任务及难点

图数据结构可以较好地表征事物之间的联系与依赖关系，在社交网络、网络搜索、自然语言处理和推荐系统等领域得到了广泛应用。尽管大规模分布式图计算十分重要，但是传统的使用 MapReduce[7] 编程模型的分布式框架 (如 Hadoop) 并不适合执行图算法。主要原因有两个：**1. 局部性差**：图算法需要访问图的任意子集使得子任务难以划分 **2. 迭代性强**：需要对同一个节点及其相邻节点不断迭代计算，而 MapReduce 将每次迭代中间文件保存到磁盘导致较大磁盘访问开销。此外，真实图的节点度通常服从**幂律分布**，使得均匀划分的子图的负载非常失衡。

此外，随着深度学习的兴起，图神经网络也对图计算系统提出的新的要求。图神经网络 (GNN) 所需要的大量张量计算使得运算不得不从 CPU 转向 GPU，GNN 也面临着图的特殊性质和传统深度学习框架的冲突。

2 图计算框架发展趋势

图计算框架的的发展趋势可以概括为**专用化**。在专门的图计算框架提出以前，大规模图计算的任务主要使用 MapReduce 模型，但是其性能相比同规模非图任务而言相当差。

2.1 “通用”到“图专门”

图计算框架的提出主要是为了应对 MapReduce 模型无法处理图低局部性和强迭代性的问题。强迭代性在图计算框架中通过**内存计算**的方式得到较好的解决。低局部性带来的两个难点是**任务划分**和**多机交互**，它们是不同图计算框架的主要发力点。

任务划分类似于 Map，指的是如何将在整张图的上的任务切分为子任务（子图）进而分配给各个主机执行。多机交互指的是分配给不同主机的子图之间如何交互信息。由于不同子图之间仍然有边相连，这些跨主机的边就是分布式图计算的主要性能瓶颈。每个子任务在独自计算的同时还需要和其他子任务交互，不能像 MapReduce 模型一样不加交互地处理 Map 任务最后再由 Reduce 来汇总结果。

Pregel [1] 是第一个完整的专用图计算框架，它针对图算法进行了特别的优化并提供了容错机制，但是在**任务划分**和**多机交互**方面只进行了简单的设计。此后的图计算框架 [2], [4], [5], [6], [8] 都是针对任务划分和多机交互进行的再优化。

2.2 “图专门”到“图针对”

通用的分布式框架无法针对图进行专门优化，同样，针对普适性图结构的分布式框架依然难以应对不同结构的图、针对所有“图算法”的分布式框架难以对每种图算法都进行优化。

针对结构：幂律分布 不同图的结构需要使用不同的划分方式，当每个节点的度的大小相近时，对图进行均匀切分无疑是最好的选择；但是真实世界的图的节点度大多符合**幂律分布**，一些图计算框架可以根据这一特性使用不同的图划分方式。PowerGraph[4] 和 GraphLab[2] 都使用基于边的切分方式来做到负载均衡。PowerLyra[6] 则使用了更加智能化的方法，对不同节点使用不同的划分、计算策略。

针对算法：机器学习 图的机器学习算法 (如 PageRank) 与非机器学习算法 (如 SSSP) 的一个重要区别是机器学习算法没有确定解，而仅仅凭借一定的收敛原则来作为终止条件。**异步 (Asynchronous)** 的图计算框架可以显著加速机器学习算法的收敛速度、提升收敛效果，甚至有的机器学习算法只有在异步的条件下才可以正确运行。PowerGraph[4] 使用了异步计算模型来专门执行机器学习算法，大大优化了速度和效果。TUX2[8] 提出了四种异步计算机制，对异步模型内部进行了再优化。Dorylus[10] 和 GNNAdvisor[11] 针对深度图的层间交互造成的大量内存占用，使用采样的方式进行调优。

2.3 “图针对”到“自适应”

在通常的概念当中，可扩展性 (Scalability) 与性能 (Efficiency) 是难以兼顾的两个属性，能够适应多种不同任务的通用型框架势必难以为每种算法每种结构进行极致优化。然而这个观点并不是绝对的。为了性能而牺牲了可扩展性的本质原因是最初的通用系统的设计本身过于简单。

笔者认为今后分布式图计算系统的趋势为**自适应**，即博采众“图针对”系统之长，为框架提供统一的顶层抽象，但能够根据图的性质不同、图算法的不同，选用不同的运行逻辑。自适应指的是一种细粒度的动态的适应，而不仅是各个分布式系统的简单组合。PowerLyra[6] 对不同的节点动态地使用不同划分和计算策略，而不是让用户指定图的性质或指定策略，就是对“自适应”系统的一种尝试。

3 图计算系统设计

3.1 CAP 模型

图计算框架认为图中的节点都是同质的。算法的实现者只需要对节点编程，指明一个顶点的行为和算法的终止条件，系统就会对图中的所有顶点执行同一个函数直到满足终止条件。

现有的图计算框架都是基于 GAS 模型，即认为一个节点的动作分为三个阶段：收集 (Gather)、应用 (Apply)、分发 (Scatter)。Gather 指从当前节点接受每一个相邻节点发来的信息，Apply 指运用 Gather 阶段收集到的信息和节点本身的属性来执行特定的计算，Scatter 指根据 Apply 阶段计算的结果计算要发送给每个相邻节点的信息并发送。节点执行运算的最小周期就是一次 GAS。

同步 (Synchronize) 与异步 (Asynchronous)

同步 同步的图计算框架指的是所有节点处在同一 GAS 步中，先完成 Scatter 的节点必须等待其他所有节点都完成本轮 GAS 才能进入下一轮。同步的图计算能够保证分布式与单机的运算结果一致，因此每种图计算框架都必须提供同步机制。但是同步计算的性能受制于每轮中计算最慢的节点，也就是常说的“水桶效应”。

异步 异步的图计算框架指所有节点不出在同一 GAS 步中，只使用全局标准进行终止判别，不能保证算法一致性。异步计算模型的优势在于可以帮助机器学习算法更快地收敛。GraphLab[2] 指出异步计算模型的收敛精度也要优于同步计算模型。

3.2 任务划分

“点”中心： 点中心的划分让每个主机上节点的个数均匀。这使得分配高“度”节点的主机需要与其他主机频繁交互，造成了负载不平衡和高阶节点高争用，如 Pregel[1] 和 GraphLab[2]。

“边”中心： 边中心的划分让每个主机上边的个数均匀，这解决了幂律分布图的负载失衡问题，但是导致高阶顶点的高通信成本和内存消耗，如 PowerGraph[4], GraphX[3]。

“点”“边”混合 PowerLyra[6] 使用启发式函数：对高阶节点倾向于使用启发式“边”中心划分而对低阶倾向于使用“点”中心划分，在高争用与高通信之间取得折中。

3.3 内存计算

图计算的强迭代性使其不适合使用 MapReduce 模型。因为 MapReduce 要求所有中间结果都保存在磁盘中，大量磁盘交互为迭代式计算造成了极大开销，但是这一问题却在 Spark 之后得到了解决。Spark 在内存中计算，解决了图计算与 MapReduce 之间最大的冲突。GraphX[3] 就是直接使用 Spark API 所搭建的图计算模型，它将非结构化的图编码为结构化表格，取得了不逊于许多专用模型的性能。这种“通用型”框架的优势在于能够更好地兼容上下游计算。

3.4 多机交互

多机交互是分布式图计算系统的主要性能瓶颈，也是绝大多数框架主要优化的方向。多机交互方式总体上可以分为**消息传递 (Message Passing)** 和**共享内存 (Shared Memory)** 两种。

3.4.1 消息传递

Pregel[1] 的多机交互过程是简单的消息传递机制，每个节点保存在一个主机上，通过消息队列进行跨设备点值读取。在每一轮计算结束的时候 (Gather)，执行计算的节点将待分发的值发送到相邻节点的消息队列中，下一轮迭代中节点就从队列中取出上一轮迭代的结果以进行后续计算。

消息传递的优点在于设计简单和内存占用少，每个节点只在一台主机上保存一个副本。但是这种交互方式的缺点却是很显然的，主机之间的消息传递受制于网络带宽，主机内部计算的时延与跨主机互联的时延相比甚至不在一个数量级，消息的更新极大地制约着图计算的总体性能。因此，在 Pregel[1] 之后的图计算框架纷纷摒弃了消息传递而转向设计更为复杂的共享内存机制。

3.4.2 共享内存 (Shared Memory)

共享内存指的是跨设备的边所连的节点在多个主机上保存副本，例如边 e_{ij} 连接点 v_i 和 v_j 且跨过主机 m_i 和 m_j，那就分别在两台主机 m_1, m_2 上都保存 v_1, v_2 两个节点。对于每台主机来说，只有一个节点实际执行计算，另一个节点是为减少跨设备交互而做的缓存。

PowerGraph[4] 使用了一种称为“Delta-缓存”的机制，其跨机器节点设定为主从模式。从所有备份中随机选择主结点，主节点实际执行计算而镜像节点只读。每次执行完 Gather 后镜像节点先把值发回主结点，然后主结点进行 Apply 操作后然再进行 Scatter。由于 PowerGraph[4] 中有主从节点之间的交互，因此需要对主节点使用开销极大的分布式锁。Cyclops[5] 使用了相似的机制但是取消了分布式锁，并不把 GAS 过程分发给主从节点，而是主节点执行完整 GAS，故而只有主节点向从节点的单向更新。

GraphLab[2] 和 TUX2[8] 是为异步模型设计的，每一个节点计算完成以后可以直接更新其所有备份，却不能保证一致性。它通过对每一个副本上分布式锁来实现串行性，但是造成了同步的高开销。

共享内存虽然导致了更多内存占用，但是成功避免了跨设备网络开销，也成为了绝大多数分布式框架的选择。共享内存机制内生的数据一致性问题也是每个框架的重点创新方向。

4 总结

本文调研了十年内分布式图计算领域的多篇论文，对分布式图计算的难点和技术类型做出归纳总结，并分析了大规模分布式图计算的发展趋势。云计算技术这门课将我领进了分布式计算的殿堂，在选择进修方向的关键时期培养了我对系统架构的兴趣，是我的一盏引路灯。

感谢李超老师一学期的辛勤付出！

参考文献

[1] Malewicz G, Austern M H, Bik A J C, 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.

[2] Low Y, Gonzalez J, Kyrola A, et al. Distributed graphlab: A framework for machine learning in the cloud[J]. arXiv preprint arXiv:1204.6078, 2012.

[3] Gonzalez J E, Xin R S, Dave A, et al. Graphx: Graph processing in a distributed dataflow framework[C]//11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014: 599-613.

[4] Gonzalez J E, Low Y, Gu H, et al. Powergraph: Distributed graph-parallel computation on natural graphs[C]//10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12). 2012: 17-30.

[5] Chen R, Ding X, Wang P, et al. Computation and communication efficient graph processing with distributed immutable view[C]//Proceedings of the 23rd international symposium on High-performance parallel and distributed computing. 2014: 215-226.

[6] Chen R, Shi J, Chen Y, et al. Powerlyra: Differentiated graph computation and partitioning on skewed graphs[J]. ACM Transactions on Parallel Computing (TOPC), 2019, 5(3): 1-39.

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

[8] Xiao, Wencong, et al. "Tux²: Distributed graph computation for machine learning." 14th USENIX symposium on networked systems design and implementation (NSDI 17). 2017.

[9] Xie X, Wei X, Chen R, et al. Pragh: Locality-preserving graph traversal with split live migration[C]//2019 USENIX Annual Technical Conference (USENIXATC 19). 2019: 723-738.

[10] Thorpe J, Qiao Y, Eyolfson J, et al. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads[C]//15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21). 2021: 495-514.

[11] Wang Y, Feng B, Li G, et al. GNNAdvisor: An Adaptive and Efficient Runtime System for GNN Acceleration on GPUs[C]//15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21). 2021: 515-531.