摘要

1、绪论

图作为一种基本的数据类型，可以作为现实中实体或实体间关系的抽象，同样在社会或科学领域中的许多问题都可以通过转换成图计算模型从而得到解决。例如: Google公司每天都需要对数亿级别的网页进行排序[1]; 生物医学上需要对蛋白质进行子图匹配，来分析蛋白质间的相互作用从而开发出更有效的临床药物[2 分布式图处理系统技术综述]; 社会安全方面更需要对整个社交网络进行精准的模式匹配，从而增加寻找贩毒、犯罪集群的可能性。因此，对图数据的分析和计算具有很重要的现实意义，目前在实际应用中，存在着很多图计算的问题，例如图的连通性[3]、单源最短路径[4]、网页排序[5]。

早期的图计算数据规模较小，因此大部分使用BGL等单机图处理算法[6]，极大的限制了图计算问题的规模，后来随着图规模的增长，出现了Parallel BGL[7]等并行图计算系统，他们实现了很多并行图计算算法，但是系统本身不具备良好的稳定性和容错性，同样对于部分算法也没有提供很好的正确性保证。为了解决稳定性等问题，部分工作专门为特定的图计算应用开发分布式架构来实现并行图计算，虽然针对性的架构可对代码进行深度优化以达到更好的处理性能，但是对于一个新的图算法，实现或移植操作需要重写大量的代码，不仅导致可拓展性和可维护性差，代码的复用率也极低，因此对编程人员的挑战极高。

近年来，随着大数据时代的来临，大图数据的分析与计算已经远远超出了单台计算机的存储和计算能力，给图分析计算带来了极大的挑战。于此同时，随着计算机硬件的飞速发展，云计算技术[8]为处理大规模复杂计算带来了挑战。MapReduce[9]作为一个并行的计算框架，及其开源实现Hadoop[10]为海量的数据处理提供了便利，为此，用户只需要定义Map和Reduce函数就可完成相应的逻辑计算，不需要关心并发、容错、一致性等复杂问题。但是由于MapReduce框架的自身特点，任务的执行划分为多次map和reduce阶段，每个阶段都需要多次的读写分布式文件系统，并且，任务的执行期间无法共享更新数据，再加上图计算的过程通常涉及多轮的迭代，各迭代涉及顶点间的复杂联系，且迭代完成时还要依赖对点的消息传递以满足图计算正确性，因此若将每轮迭代抽象成MapReduce作业不能高效的进行并行图计算，虽然Haloop[11]等处理框架对MapReduce过程进行许多改进，但是并没有从根本上解决基于分布式文件系统所带来的I/O开销问题。

针对上述图计算的特点，为解决对大规模图数据计算的迫切需要和传统的计算平台无法满足当今的图处理需求的问题，面向大规模图计算处理的研究吸引了越来越多的研究者投身其中。

2、国内外研究现状

2.1 同步计算模型

1990年，图领奖获得者Valiant提出了BSP(bulk synchronous parallel)计算模型[12]，这种模型区分于现有的MapReduce计算框架在于其适合做数据的迭代计算。其核心思想是将每一次迭代称为一个超步(super-step)，整个图计算划分为多个超步，如图2.1所示，并行迭代直到全部顶点收敛(即每个顶点相邻两次迭代的变化量小于给定的常量)或达到预设的最大迭代次数。在每一次的超步计算中，又会分为本地计算、消息传递、全局同步三个阶段，如图2.2所示。

（1）本地计算: 各个计算节点相互独立，各自负责处理存储在本地的计算数据。

（2）消息传递: 本地计算阶段完成后，各个计算节点将需要更新的消息通过网络通信进行交换，所交换的信息供下次本地计算使用。

（3）全局同步: 各个计算节点在完成本地任务后进入全局同步阶段，先完成本地任务的节点需要等待没完成任务的节点，直到最后一个节点完成本地任务，所有节点统一进入下一轮的本地计算。

从上图中我们可以清楚的看出，每轮计算中所有顶点只能使用上一轮迭代计算的更新消息，而通过计算得到的最新消息也只能通过同步在下一轮的计算中使用。因为全局同步的存在，使得整个图计算的调度十分简单，同时，每一轮的消息会在全局的同步下批量的传送，极大限度的利用了网络的流量。

为高效的处理大规模图数据的计算问题，在上述BSP计算模型的启发下，Google首次提出了以点为中心的计算框架Pregel[13]。在点中心的计算模型中，用户只需要从点的角度出发，重写一个与具体应用紧密相关的计算函数Compute()，而不需要考虑数据的处理过程。在Compute()函数内，涉及(1)处理上轮迭代的更新消息; (2)本地计算逻辑，并更新本节点的值或节点状态信息; (3) 消息传递逻辑; Pregel的具体执行过程如下: 首先，选定一个图分区策略[14-15 edge-cut vertec-cut]进行图分区操作，划分后的不同分区分配到不同的计算节点上，然后在每一个图节点上执行Compute()函数，即用户自定义的超步处理逻辑，每一次的迭代后，通过全局同步操作保证所有计算节点统一进入下一轮的迭代计算，整个迭代过程直到没有任何消息需要同步或所有的图节点达到收敛状态时停止，输出结果。Pregel框架从容错性、一致性、同步控制等方面给出了可行的解决方案，整个计算模型简单、易用、可拓展性强。

同样采用上述BSP模型的分布式图计算框架还有GPS[16], Giraph++[17], GRAPE[18], 其中GPS是点中心(vertex-centric)模型; Giraph++是块中心(blocks)模型，但是其继承了点中心模型的思想，即将块的处理同点的处理方式相同; GRAPE是以子图(subgraph)为中心模型。

虽然BSP模型极大的简化的图计算任务，充分利用了网络流量，但是该模型仍有以下几方面限制：

（1）BSP模型具有木桶效应，因为全局同步的存在，每一轮迭代的计算时间取决于最慢的计算节点。因此，节点的计算能力不同或图数据的分区不合理都会导致计算节点间的负载不均衡，严重影响图计算效率。

（2）BSP模型每轮迭代只能使用上一轮迭代的更新信息，导致迭代次数较多，收敛速度较慢。

（3）消息量大，导致通信代价昂贵。在分布式的图处理框架中，为了满足大规模图数据处理，通常需要将原始的图数据划分为多个子图，并把每个子图分配到不同的计算节点上，因为处于不同计算节点之间的顶点具有一定的关联关系，这就导致不同计算节点间需要进行消息传递以满足图计算的正确性。然而在实际的图数据中，尤其是社交网络等图应用，图中顶点的度数服从幂律分布，从而使得通信的代价成为影响系统性能的瓶颈之一。

（4）某些需要协调图中邻接点的算法无法在BSP模型下实现，例如图着色问题[19],该问题意在使用最少的颜色为图着色，保证相邻的顶点颜色不同。这类问题对于贪心的解决方案在BSP模型下不会收敛，因为图节点试图获得非邻居节点的颜色值，只能通过两者的共同邻居节点来获得，因此同步操作的过程中可能会出现这两个节点颜色不断翻转的情况，为本轮迭代具有相同颜色的邻接顶点在下轮的迭代仍会选取相同的颜色，从而导致无法收敛。

2.2 异步计算模型

为解决上述BSP同步计算模型的诸多问题，科研工作者提出了异步计算模型AP(Asynchronous Parallel)，如图2.3所示。异步模型中，相邻两次迭代没有明显的界限，任何计算节点随时接收来自其它节点的消息，并直接接受调度器的调度执行; 同样，本地计算生成的最新消息也随时传递给其它需要的节点。因为异步模型摒弃了全局同步的操作，使得每个计算节点不用关心其它节点的进度、状态，只要消息队列存在消息就可以进行下一轮的本地计算，因此消除了同步计算模型下木桶效应问题。经大量的理论分析和实验表明，异步模型在执行效率以及资源的利用率方面相对与于同步BSP计算模型具有较大的优势，尤其是在图处理系统负载不均衡时，这种优势体现的更加明显[20-21]，但是异步模型也同样存在着以下方面的不足之处：

（1）虽然异步模型可以加速图计算的收敛，但由于没有全局同步的限制，导致迭代轮数的增多, 造成了大量的冗余计算，具体内容会在第4节关键性技术分析中指出。

（2）异步模型在分布式图计算系统的设计方面相对于同步模型来说更加复杂，在设计过程中，不仅要设计正确的调度器，还需要考虑数据的一致性问题，因为异步模型下的任何一个计算节点随时接收来自其它计算节点的消息，因此必须设计一套额外的机制来保证相同数据对象可以被不同的进程和线程互斥访问，例如GraphLab[22 分布式图处理系统技术综述]采用分布式锁的形式来保证数据的一致性，而GRAPE+[23 AAP]采用了聚合函数Aggr()来保证数据的一致性。

（3）异步模型的计算过程与计算结果具有不确定性，如果算法设计的不合理，迭代计算可能无法达到预期的效果，甚至不能收敛，即使能够收敛，也同样会在迭代计算的最终收敛前，造成大量的冗余计算。不仅如此，对于采用异步模型的分布式图计算系统来说，编程难度有所增加，调试十分困难[21 GRACE]。

2.3 其它计算模型

最近研究表明不同计算模型下，图算法的执行效率受到图应用算法、图分区方式、数据图的特性与规模、分布式集群的规模等多方面因素的影响，不仅如此，在整个迭代周期的不同阶段，可能也需要不同的计算模型以达到全局最优[24 powerswitch]，因此为了解决上述问题，科研工作这提出了一些新的计算模型。

2.3.1 Stale Synchronous Parallel

Stale Synchronous Parallel计算模型[25]，简称SSP模型。其通过引入一个固定的限定值bound来保证最快与最慢的计算节点间相对的迭代轮数不会超过给定的bound值。因此，其允许计算能力强计算节点采用异步的方式执行下去，来减缓同步模型下的木桶效应; 同时通过bound值的限定强制要求计算能力强的节点需等待计算能力差的节点以减少异步模型下的冗余计算。

2.3.2 Hybrid Model

Hybrid Model计算模型[24]，简称Hsync模型。其通过一组启发式的算法建立代价收益模型来动态预测同步模型和异步模型两种调度方式的性能，并实现了在计算过程中对两种计算模型自由的切换，如图2.4所示，[24]的实验结果表明，PowerSwitch可以准确预测两种模型的性能，并完成调度模式的切换，相对于使用单一的调度模式，在其上执行了大量的图算法，如PageRank算法、单源最短路径算法、图着色算法等，都在执行效率上得到了不同程度的提高。

2.3.3 Adaptive Asynchronous Parallel

Adaptive Asynchronous Parallel计算模型，简称AAP模型。这种模型摒弃了全局同步的操作，因此从本质上来看其属于异步模型。但与AP计算模型不同的是，每个计算节点在AAP模型下均保留一个参数来（1）测量与其它计算节点的相对执行进度; （2）累积接收到的消息，并决定是否进入下一轮迭代。上述整个过程只在本计算节点下完成，不用关心其它任何节点的执行状态。AAP模型不同与其它计算模型主要体现在以下几个方面:

（1）BSP模型：AAP模型通过异步的消息传递机制来消除BSP模型下木桶效应问题，摒弃了全局同步机制从而消除了这一操作所带了的额外开销。

（2）AP模型：AAP模型通过引入限定值Delay Stretch使得各个计算节点自己决定每一轮迭代需等待的时间，聚集更多的消息传递量从而达到执行更有效的计算，因此很大程度上减少了AP模型下的冗余计算问题。

（3）SSP模型：SSP模型下通过预设的限定值bound，使得在bound值允许的范围内，各计算节点采用要异步AP模型进行计算；一旦到到达bound限定的范围，计算快的节点就需等待，直到计算慢的节点完成当前轮数的计算，才可进行下一轮的计算。但AAP模型不同于SSP模型在于其通过限定值Delay Stretch动态调整各计算节点间的相对进程，当出现计算落后的节点时，其允许该节点等待一段时间以便接受更多的消息，这样可以最大化的利用其它计算节点快的优势，使得慢节点的每一轮计算更有效率。

（4）Hybrid Model模型：相对于Hsync模型，AAP模型不需要全局的预测开关在BSP和AP模型进行切换，相反，AAP模型允许每个计算节点决定各自的计算模式，因此在图算法的执行期间，某些计算节点采用BSP模式，与此同时，其它节点采用AP模式。除此之外，犹豫AAP模型动态调整计算节点间的相对进程，因此其不产生模型间切换所带来的而外开销。

3、关键性技术分析

3.1 影响图计算效率的因素

为了提高并行图计算的效率，现有的工作主要从以下三个方面进行研究，计算粒度、计算模型和通信机制，如图3.1所示:

3.1.1 计算粒度

目前，针对分布式图计算系统在处理图计算遍历的过程主要使用了3中不同的计算粒度：以顶点为中心的计算粒度、以子图为中心的计算粒度和以路径为中心的计算粒度。其中以顶点为中心的计算粒度最为常见，例如开源框架Pregel、Giraph[26]等都采用这种计算粒度，这些系统结构简单、易于实现且算法表达能力强，但是不利于算法的收敛且网络负载较重；以子图为中心的计算粒度将分区后的子图看成一个整体，虽然这类系统的结构和实现较为复杂，但是其算法的表达能力更强，很多图上的单机算法可直接迁移，并且在减少通信量的同时有效的减少了计算的迭代次数；以路径为计算粒度可以减少对边的随机访问，从而提高图遍历算法的性能，但是在分布式的环境下缺少有效的实本质上继承了以顶点为计算粒度的特点，只是将块当作点来思考。表3.1展示了常见图处理系统的计算粒度。

**顶点为中心的计算粒度**：以顶点为计算力度的图处理系统中，其迭代处理过过程是通过对图中所有顶点的遍历完成的，首先用户需要以顶点为基本单位编写图处理逻辑，例如Pregel系统中的Compute()函数，该逻辑的功能主要包括消息的接收与发送，顶点计算以及状态更新。然后，系统会自动的按照函数的逻辑针对图中的每一个顶点进行同步或异步的运算。

我们以图的连通性问题为例，给定一个无向图，若中的子图满足（1）中的任何两个顶点间均存在连通路径（2）向中增加任何一个属于G的顶点都会使不具有连通性，我们就认为子图是一个连通分量(connected component)，下图3.1所表示图只有一个连通分量，但包含两个子图，分配到不同的计算节点，图中每个顶点含有全局唯一的整形值，我们以最小的顶点号表示该顶点所在连通分量，图3.2中展示了在以定点为中心的计算粒度模式下，求解图连通分量问题的最大迭代过程。

从上图中可以清楚的看到，在第一次迭代（超步1）中，每个顶点向其邻居节点发送自己的value值，如图中的箭头所示，其中虚线表示属于同一子图内顶点间的内部通信，实线则表示跨越计算节点的子图间通信；接下来的每次迭代，每个顶点选取其所收消息的最小值于自己的value值进行比较，如果最小值小于自己的value值，则对自身的value值进行更新，并向邻居借点广播更新后的新值；否则，将自己的状态设置为非活跃状态(即图中的灰色节点)，跌倒过程一直持续到图中所有的顶点都是非活跃状态且没有任何需要传递的消息，在本例中，共进行5从迭代，8次子图间的消息传递。

**子图为中心的计算粒度**：为加快算法的收敛速度，减轻网络负载，大量以子图为计算粒度的分布式图计算系统被提出，如GRAPE[]等系统，这类系统将子图看成整体，执行图应用算法，在计算完成时，通过子图的边界节点进行消息传递，如图3.3展示了在子图上直接进行图连通性计算的迭代过程，在第1次迭代中，每个子图根据遍历或者其它方式求解出子图内的各连通分量，子图的每个内部点将所在连通分支的最小值设置成自身的value值，处于边界的节点向邻居发送自己的value值，接下来的每次迭代，每个子图选取消息中的最小值与消息对应的连通分量进行比较：如果该最小值小于对应的value值，则更新连通分量内所有的内部点的value值，并通过边界结点传递消息；否则，将自己设置成非活跃状态。迭代持续进行，直到子图内所有节点为非活跃状态且没有任何需要传递的消息，在该例中，共进行了3次迭代域3次跨计算节点间的消息传递。

通过对比图3.2与图3.3可以发现，以子图为计算粒度的分布式图计算系统不仅可以有效的减少运行过程中的迭代次数，还可以显著的减轻网络负载，但是相对于以顶点为中心的计算粒度，用户需要针对不同的图算法确定出子图内的边界节点，以便正确的进行消息传递。

**路径为中心的计算粒度**：目前，以路径为中心的计算粒度主要是集中式的，采用该计算粒度的主要目的是为了减少对边数据的随机访问率，提升系统性能。X-Stream[27]就是此类系统的代表，为减少对磁盘的随机访问率，该系统将边数据和点数据进行分区，并对边采用顺序访问的方式进行遍历计算。但是到目前为止，在分布式的环境下，以路径为计算力度还缺少有效的实践。

3.1.2 计算模型

3.1.3 通信机制

分布式系统中，计算节点间的相互通信是不可避免的。在分布式图计算系统中，计算节点间的通信通常代表划分到节点中的子图间通信。当图数据规模增加或执行的图应用算法通信密集时，计算节点间的通信量会明显上升，如果没有高效的通信机制，势必会成为影响图计算效率的主要因素，最终成为整个分布式图计算系统的瓶颈。目前，常见的分布式图计算系统的通信方式主要分为两类：基于共享内存的方式和基于消息传递的方式，表3.2展示了部分系统及其采用的通信机制。

**基于消息传递的通信机制**：在基于消息传递的通信机制当中，计算节点间的信息传递是通过利用网络通信平台在节点间发送消息来实现。传递的消息中包含需要发送的信息数据和目标顶点的全局ID。通常使用的网络传输协议主要包括3大类：基于MPI(Message Passing Interface)通信、基于RPC(Remote Procedure Call)通信和基于Netty通信。在具体的分布式图系统中，通常会使用上述的一种或多种通信协议，如：GRAPE[]、GPS[]、Pregel等使用MPI通信协议；Giraph[]同时支持RPC和Netty通信协议，其协议可以通过配置文件进行选择。

基于消息传递的通信机制不需要其它额外的机制就可以保证数据的一致性，且具有良好的可拓展性，因此被大多数的同步图计算系统采用。在此类系统中，图节点在计算过程中产生的消息会随着通信过程发送到目的节点。如果目的节点和源节点划分到同一台机器上，则直接将该消息放到该节点的接收队列中；否则，将该消息发送到放松消息缓冲队列池中等待发送。当所有节点的本地计算完成后，调用相应的发送接口将缓冲池中的消息批量发送出去。目的节点接收消息后，会根据消息中封装的顶点ID将数据送入对应顶点的接收队列中。整个通信过程中，系统会使用批量发送的方式来优化网络通信。

**基于共享内存的方式**：在基于共享内存进行通信的分布式图计算系统中，每个顶点的数据通过共享变量的方式存储在计算节点上，当某活跃顶点在计算过程中需要其它其它顶点的数据时，可以直接按照相应的内存地址进行读取。在分布式的环境中，由于每个顶点均有自己的独立内存地址且需要保证所有共享顶点的数据一致性，使得共享内存的方式实现起来较为困难。

为有效地管理集群中各计算节点的内存地址，Trinity[27]系统设计一套集群内存集中管理方案。该方案将集群内每个计算节点的内存组织成一个巨大的虚拟内存空间，并按照规则给予每个存储单元一个64位的存储空间地址，存储在集群内的任意节点可以通过地址访问虚拟内存空间的任意单元，从而将分布式集群间通信在形式上于单机环境类似。

4、关键性技术分析（算法的结构，各自的优点，缺点，能不能混合）

5、总结与展望

6、参考文献

[1] Google. How search works. <http://www.google.com/insidesearch/howsearchworks/thetory>

[6] Jeremy G. The boost graph library: user guide and reference manual[M]. Addison-Wes-key,2002