从系统角度审视大图计算

● 吴城文　张广艳　郑纬民
清华大学计算机科学与技术系 北京 100084

摘要：
大图计算已经成为学术界和工业界的一种基本的计算模式，并且已经被应用到许多实际的大数据计算问题上，比如：社交网络分析，网页搜索，以及商品推荐等。对于这些问题，大图的规模约有10亿级的点，以及1000亿级的边，这样的规模给大图的高效处理带来了诸多挑战。在这篇文章中，我们将主要介绍大图计算的基本特征和挑战，典型的计算模型，以及具有代表性的分布式、单机处理系统，同时对图处理系统中关键的技术进行总结，最后从系统的角度给出大图计算可能的一些研究方向。

关键词：大数据计算，大图计算，计算模型，计算系统

1. 引言

图可以用来表征不同实体间复杂的依赖关系。因而，在许多实际的应用当中，如社交网络分析，网页搜索，商品推荐等，都可以使用图来进行问题的建模和分析。然而，在大数据时代，这类问题的规模通常十分庞大，以社交网络为例，Facebook在2014年7月的用户已经达到22亿[1]，而用户之间的关系数量则更多，以数据的方式进行存储通常会占用几百GB甚至TB级的存储量。因此，大图计算不仅是计算密集型，同时也是存储密集型问题，如何在可以接受的时间内对大图进行计算，是我们需要解决的难题。

```
vertex_scatter(vertex v)
  send updates over outgoing edges of v

vertex_gather(vertex v)
  apply updates from inbound edges of v

while not done
  for all vertices v that need to scatter updates
    vertex_scatter(v)
  for all vertices v that have updates
    vertex_gather(v)
```

图1 以点为中心的计算模型[15]

通常，为了快速地对大图进行处理，我们常常会使用分布式并行计算的思想，但是由于图计算本身特征使得我们在实现并行图计算时，不能使用传统的科学计算领域的并行模式（计算偏微分方程）[2]；且以往在处理大数据问题上的Map/Reduce[3]模式，在处理图问题时效率极低；另外，并行图算法库Parallel BGL[4]或CGMgraph[5]没有容错机制。基于以上几点，需要一套符合大图计算特点的高效分布式并行计算框架。现在一些常见的分布式处理系统有Pregel[6]及其对应的开源实现Giraph[7]，以及GraphLab[8]，PowerGraph[9]，GraphX[10]和Cyclops[11]。这些分布式系统大部分采用“think like a vertex”的思想，即以点为中心(vertex-centric)的计算模型，如图1所示，在这种模型当中，所有的点从其入边的邻点获取数据，执行用户自定义的函数对自己的状态进行更新，然后将自己的更新状态通过消息发给其出边的邻点。还有少数一些分布式系统采用了其它的计算模型，如PowerGraph的以边为中心(edge-centric)的计算模型，如图2所示，在这种计算模型当中，首先依次遍历所有的边，将边的源点 的更新值通过其出边传递给目的点，然后遍历所有的更新值，将更新到目的点（在PowerGraph中将gather操作移到了scatter操作前面）。另外，还有以块[12]、路径[13]为中心的计算模型，在这类计算模型中，针对图结

基金项目：国家重点基础研究发展计划（“973”计划）基金资助项目（No.2014CB340402），国家自然科学基金资助项目（No.61170008，No.61272055）

构来进行图划分增加了计算的局部性，但是，也存在图划分时间过长等一些问题。

```
edge_scatter(edge e)
  send update over e

update_gather(update u)
  apply update u to u.destination

while not done
  for all edges e
    edge_scatter(e)
  for all updates u
    update_gather(u)
```

图2 以边为中心的计算模型[15]

分布式图处理系统，随着问题规模的扩大能够具有很好的拓展性，但是在提高系统处理效率方面我们任然面临许多挑战。比如，图的划分，要提高系统性能，需要在保证集群各节点负载均衡的情况下，并且使得集群内各节点的通信量最少，是一个NP问题。此外，一个分布式系统需要解决集群内各节点协同工作，容错等一系列的问题，而这类问题系统的性能影响有着重要的影响。另一方面，对于使用分布式系统的程序员来说，环境的搭建、编写分布式程序比较复杂，而且程序的调试和优化又相对困难。基于此，最近一些大图计算的研究工作，在使用单台计算机进行大图计算处理上有了一些新的成果，如以点为中心的计算模型的GraphChi[14]，和以边为中心的计算模型的X-Stream[15]，另外还有VENUS[16]，GridGraph[17]等。这些成果极大地降低了大图计算成本开销，同时能够达到甚至是好于一些分布式图计算系统处理时延。

本文将介绍当前大图计算的主要特征及挑战，并且从系统角度给出当前大图处理系统的主要特征及其研究成果，并对图处理系统中的关键技术进行总结，最后将对全文进行总结并给出大图计算系统方面的相关可能的研究方向。

2. 大图计算的特征及挑战

大图计算是大数据计算中的一个子问题，除了满足大数据的基本特性之外，大图计算还有着自身计算特性，相应地面临着新的挑战。

（1）局部性差

图表示着不同实体之间的关系，而在实际的问题当中，这些关系经常是不规则和无结构的，因此图的计算和访存模式都没有好的局部性，而在现有的计算机体系架构上，程序的性能获得往往需要利用好局部性。所以，如何对图数据进行布局和划分，并且提出相应的计算模型来提升局部性是我们提高图计算吞吐量，是我面临的关键挑战。

（2）数据及图结构驱动的计算

图计算基本上完全是由图中的数据所驱动的。当我们执行图算法时，算法是依据图中的点和边来进行指导，而不是直接通过程序中的代码展现出来。所以，不同的图结构在相同的算法实现上，将会有着不同的计算性能。因此，如何使得不同图结构在同一个系统上都有较优的处理结果，也是我们面临的一大难题。

（3）图数据的非结构化特性

图计算中图数据往往是非结构化和不规则的，在利用分布式框架进行图计算时，首先需要对图进行划分，将负载分配到各个节点上，而图的这种非结构化特性，使得很难实现对图的有效划分，从而达到存储，通信和计算的负载均衡。而一旦，划分不合理，节点间不均衡的负载将会使系统的拓展性受到严重的限制，处理能力也将无法符合系统的计算规模。

（4）高访存/计算比

绝大部分的大图计算规模使得内存中无法存储下所有的数据，计算中磁盘的I/O必不可少，而且大部分图算法呈现出迭代的特征，即整个算法的进行多次迭代，每次迭代需要遍历一遍整个图结构，而且每次迭代时，所进行的计算又相对较少。因此，呈现出高的访存/计算比。另外，图计算的局部性差，使得计算在等待I/O上花费了巨大的开销。

3. 分布式大图计算系统

在本章，我们将介绍几个典型的大图处理的分布式系统，重点突出每个系统的特点。

3.1 Pregel

Pregel是由Google开发的分布式处理图系统，其主要的设计思想是基于BSP（bulk synchronous parallel）[18]。在此思想上，Pregel使用了以点为中心的计算模型，对整个图根据点进行划分，将不同的点以及相关的邻边存储到不同的计算机器上。在Pregel中，用户可以自定义点的compute()函数，每个点多次迭代执行这个函数，并最终得出整个图的计算结果。具体地，在每一次迭代(superstep)中，每个活跃的点(active vertex)会执行compute()函数，在这个函数中，该点读取在前一次迭代中其邻点发送的消息，通过这些消息计算自己新的状态，再自己最新的状态通过出边发送给其邻点（邻点将会在下一次迭代中收到这些消息），然后该点会进入不活跃状态(inactive)（如图3所示）。当不活跃的点(inactive vertex)在下一轮收到消息时，就会重新处于活跃状态。当所有活跃的点执行完compute()函数之后，当前迭代结束，并且进入到下一次迭代。如果系统当中

所有的点都处于不活跃的状态并且没有任何新的消息，那么算法就结束。

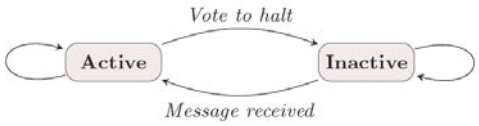


图3 Pregel点的状态机[6]

Pregel使用了消息传递(message passing)的方式进行计算节点之间的通信，在一次迭代中每个点可以向其它点发送任意量的消息，而这些消息将会在下一次迭代中被对应的点读取。在分布式的环境中，为了减少机器间的通信量，提升计算的性能，当点的compute()函数的操作符合交换律和结合律时，Pregel可以支持用户实现combiner()函数，把从机器Mi到另一台机器Mj上点v的所有消息合并成一条消息。

3.2 Giraph

Giraph构建在Hadoop[19]之上，是对Google的Pregel的开源实现。Facebook使用Giraph来进行社交关系图的分析。为了提升系统的性能，在原有Giraph基础上，增加了一些优化的措施。Facebook在Giraph的加载图数据，写回图数据以及计算阶段引入了多进程，提升了系统的整体性能，尤其对计算密集型的应用，引入多线程可以使性能随着处理器的增加获得接近线性的加速比。

3.3 GraphLab和PowerGraph

与Pregel的同步数据推送的BSP模型不同，GraphLab使用异步的GAS(Gather, Apply, Scatter)模型来实现大图分布式并行计算。GraphLab使用共享内存(shared memory)的方式来实现以点为中心的计算模式，在这种方式下，每个点可以直接读取和修改其邻点和邻边的值。在GraphLab上实现算法时，用户需要实现符合算法要求的GAS函数，在算法执行时，图的每个点都会执行该函数。

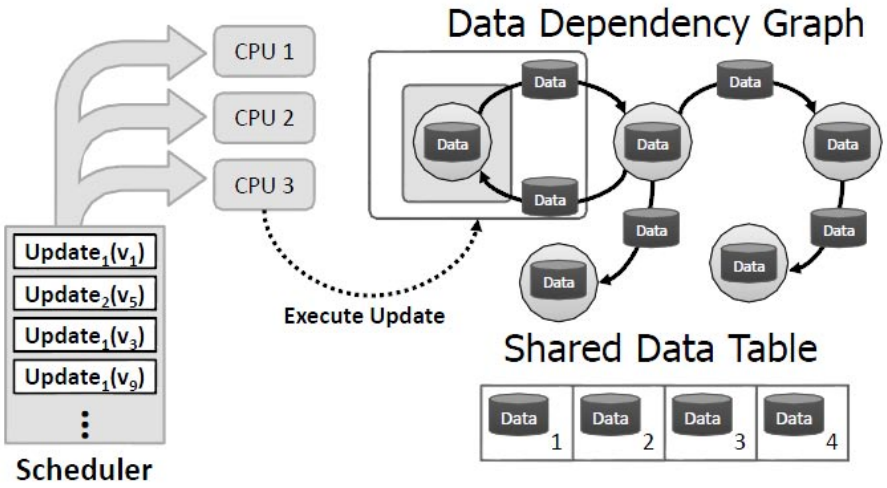


图4 GraphLab计算框架[20]

在Gather阶段，每个执行GAS函数的活跃点从其邻点和邻边获取数据，然后使用这些值来计算自己的更新值，这里计算操作必须满足交换律和结合律。在Apply阶段，活跃点把计算得到的新值更新原来的旧值。在Scatter阶段，活跃的点会通过邻边激活对应的邻点。在GraphLab中，使用一个全局的调度器，如图4所示，各个工作节点从该调度器获取活跃的点来进行计算，这些正在被计算的点也可能会将其邻点调入到调度器中。最后当调度器中没有任何可调度的点时，算法终止。这种调度器的使用使得GraphLab同时支持算法的异步调度执行和同步调度执行。

同步执行（synchronous execution），在这种计算模式下，每个点或者边的更新不能马上被接下的计算所感知到，直到当前迭代结束时，在下一次迭代当中才能读取到更新的值。异步执行(asynchronous execution)，与同步计算不同，点或者边的更新能够马上被接下的计算所感知，并使用到，这种计算模式可以使得如PageRank的一些算法收敛速度更快，但也同时会导致数据竞争，从而产生额外的计算开销，另外，在分布式系统中，这种模式会产生随机的信息传递，因而也会产生较大的通信开销，因此，一般来说，对于计算密集型的算法如BP，更适合使用异步计算的模式。

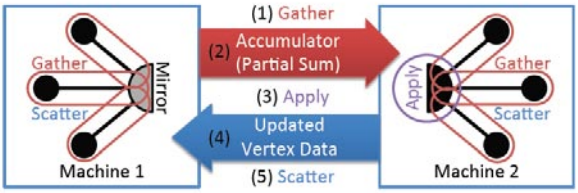


图5 PowerGraph切割点集划分及通信模式[9]

PowerGraph，包含在GraphLab2.2中，是在GraphLab的基础上，对符合幂律分布(power-low)的自然图计算性能的改进，其主要的改进是在图的划分上。PowerGraph使用了Vertex-cut的图划分策略，如图5所示，将待处理的图以切割点集的方式进行划分，将那些度极大的点的边分割给不同的计算节点，同时，将对应的点也复制给这些计算节点作为镜像(mirror)点，具体计算时，每个主点及其对应镜像点在本地执行Gather操作，随后镜像点将自己的计算结果发送给主点，收到全部计算结果后，主点执行Apply操作，并且将更新值发送给所有镜像点，最后主点和镜像点进行Scatter操作。

3.4 GraphX

GraphX，如图6所示，是构建在分布数据流框架Spark[21]上的分布式图处理系统。GraphX支持Pregel和GraphLab的计算模型，并且拓展了Spark中的RDD(Resilient Distributed Dataset)，引入了RDG(Resilient Distributed Graph)，这种结构可以支持许多图操作，因此现有的大多数图算法都可以使用系统中提供的基本操作算子（如：join，map，和group-by）来实现，并且实现十分简单。为了利用到Spark中这种算子操作，GraphX重构了新的vertex-cut图划分方法，将图划分成水平分区的顶点和边的集合。GraphX的性能比直接使用分布式数据流框架好一个数量级，稍差于GraphLab。另外，由于GraphX是构建在Spark之上的，所以GraphX能够得到低开销的容错和透明的错误恢复支持。

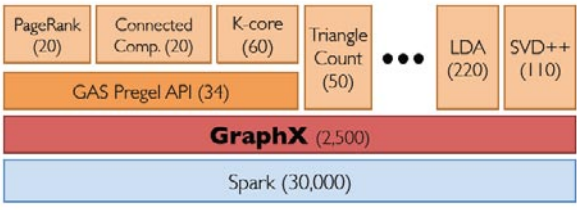


图6 GraphX的层次结构（括号中为代码行数）[10]

4. 单机大图计算系统

随机单台计算机处理能力和存储能力的提升，再加上人们对于图计算模式研究的深入，一些在单机上处理大图计算的系统被提出，这些系统有着很好的图计算性能，同时相比分布式系统，其低硬件成本和低功耗的优势明显。本章，我们将介绍几个代表性的单机大图计算系统。

4.1 GraphChi

GraphChi，是一个基于磁盘的单机大图处理系统。在大图计算中，计算的访存局部性非常差，严重影响到计算的性能。特别地，在单机情况下，系统的计算能力十分有限，因此，为了提升计算性能，GraphChi使用了：一、具有创新性的磁盘数据布局和对应的计算模型，来减少磁盘的随机访问；二、选择性的调度来加速算法的收敛。

磁盘数据的布局和计算模型。GraphChi在计算前首先会对图数据进行预处理，将输入的图划分成多个shard，每个shard当中存储对应点集的所有入边，并且将入边按照其源节点的ID编号进行排序，划分时需要保证每个shard中边的数量大致相同，每个shard都能够加载进内存。GraphChi使用以点为中心的计算模型，使用并行滑动窗口(parallel sliding window)来加载数据进行计算，如图7所示，每次(interval)计算一个子图，即一个shard所对应点集中所有点的值，需要顺序读取某个点集对应的入边（深色部分），以及该点集在其它shard中所对应的出边（黑色矩形框部分），这种数据布局和计算模型可以保证每次interval计算的I/O是顺序的。这样，一次迭代，计算整个图中所有点的值，多次迭代，直到算法收敛。

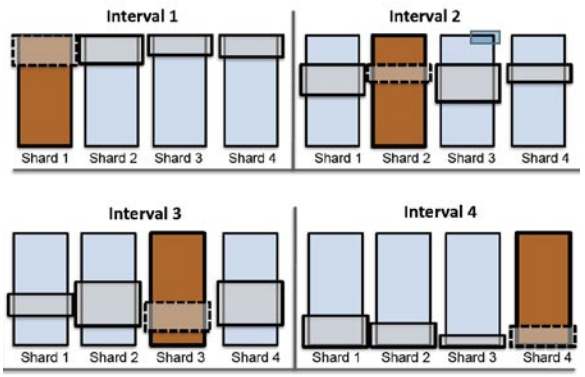


图7 并行滑动窗口计算模型[14]

选择性的调度。在GraphChi当中，可以使用选择调度性调度来加快图中某些点的收敛，尤其是对这些在两次相邻的迭代当中变化很显著的点。在点执行update()函数时，类似GraphLab中的Apply()，可以将其邻点加入到调度器当中，进行选择性地调度。

4.2 X-Stream

与GraphChi所使用的以点为中心的计算模型不同，X-Stream使用以边为中心的计算模型，并且所有

的状态都保存在点当中。X-Stream的计算过程主要分为两步，Scatter和Gather，如图所示。在Scatter阶段，X-Stream依次遍历每一条边，判断边的源节点是否产生更新，如果有更新产生，将边通过出边发送给目的节点。在Gather阶段，X-Stream依次遍历在Scatter阶段产生的所有更新，并更新对应点的状态值。X-Stream以边为中心的计算模型对边进行顺序访问，可以充分发挥磁盘的等二级存储介质的顺序访问高带宽加速图计算，但是在X-Stream中对点的访问还是随机的，为了对此进行优化，进一步提高计算性能，X-Stream对图的点集合均等划分成小的子点集合，每个子点集合其每个点所有的出边也对应地组成一个边的划分集合。对点的划分主要满足每个子集合中的点都能够存储进内存当中，这样当计算每个划分块的时候，对点的随机访问开销能够极大地降低，如图8所示，为X-Stream进行划分后的计算模型。

```
scatter phase:
  for each streaming_partition p
    read in vertex set of p
    for each edge e in edge list of p
      edge_scatter(e): append update to Uout

shuffle phase:
  for each update u in Uout
    let p = partition containing target of u
    append u to Uin(p)
  destroy Uout

gather phase:
  for each streaming_partition p
    read in vertex set of p
    for each update u in Uin(p)
      edge_gather(u)
    destroy Uin(p)
```

图8 X-Stream以边为中心的计算模型（Uin/Uout为输入/输出缓存）[15]

在对图进行划分之后，每个划分块在Scatter阶段，首先将所有的更新值写在本地的一个输出缓存当中，当所有的块都完成Scatter之后，进入一个Shuffle阶段，这个阶段的主要工作是将所有划分块的更新进行分配，将更新分配到对应的划分块的输入缓存中，作为Gather阶段的输入，来点的状态进行更新处理。相比于GraphChi，X-Stream对所有边进行顺序访问，能够充分发挥磁盘等二级存储介质的顺序带宽的速度，同时预处理阶段无需开销巨大的排序，因此能够获得较好的图处理性能。

4.3 VENUS

尽管GraphChi在大图处理上能够取得较好的计算效果，但是也存在如下的缺陷：一、预处理需要对边的源节点进行排序，开销大；二、图数据的加载和计算是分开的，没有充分利用到磁盘和I/O的并行，来提高计算性能；三、对shard内的边排序后，每个点所对应的边不在相邻的位置，Cache局部性不高。

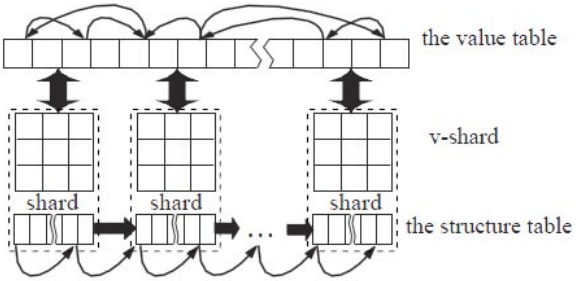
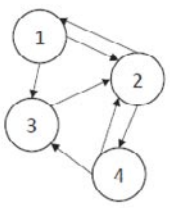


图9 以点为中心的流线型计算模型[16]

基于以上的这几点观察，作者提出了如图9所示的以点为中心的流线型(Vertex-Centric Streamlined)计算模型。在这种计算模型当中，作者分别构建了g-shard和v-shard，其中g-shard与GraphCHi中shard的概念类似，存储了一个子点集对应的所有入边，但是不用对边进行排序，而是将目的顶点相同的边存储在相邻的位置，v-shard存储对应一个g-shard中所有目的顶点和源顶点的值。另外，使用了一个全局的点值表，v-shard从其中读取和写回对应的点值。系统计算点的更新值时，无需将所有的入边和出边加载进内存，只需加入边加载进内存，同时节点更新后，不用再加更新值写入出边，这样可以极大地减少I/O。此外，在当加载完g-shard中一个点的所有入边时，即可对该点的值进行计算，重叠了I/O和CPU的时间开销，极大地提高了系统的性能。实验结果表明，VENUS的性能要显著地好于GraphChi和X-Stream。

4.4 GridGraph

在X-Stream中，在Scatter和Gather阶段之间，还需要一个Shuffle阶段将每个划分在Scatter阶段产生的更新值，分配到对应的划分的输入缓存中，供Gather阶段进行计算。在Scatter阶段，更新值会有O(|E|)这样的规模，其中|E|代表图中边的数量。所以，当内存不足时，需要将一部分缓存先写入磁盘，并且在Gather阶段需要将写入磁盘的更新值重新读入到内存，因此，在此过程中可能会触发较多的I/O，严重影响系统的性能。



(a) An example graph

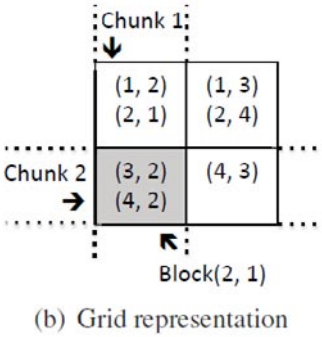
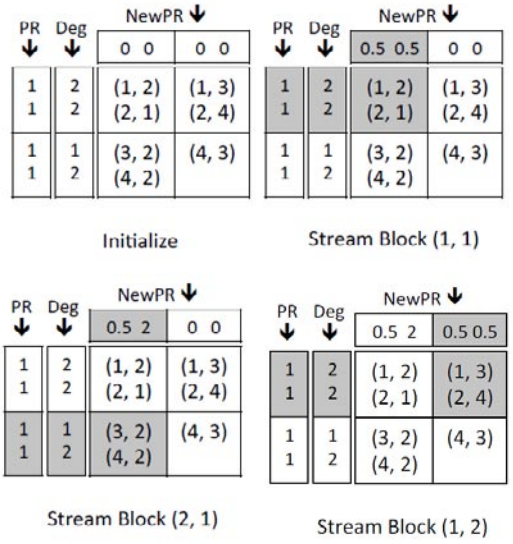


图10 GridGraph的图划分例子[17]

为此，GridGraph提出了如图10所示的格子划分方式，首先，将整个点集划分成相同大小的P份子点集，然后对边以行和列划分成格子，每一行对应着在某个子点集内的点所对应的所有出边，每一列对应在某个子点集内的点所对应的所有入边。对应着这种图的划分方法，作者提出了双重滑动窗口的计算模型，如图11所示，是图10(a)中图结构的PageRank第一次迭代过程，计算点的更新值需要读取其入边源节点的值，为此从上到下，依次读取该列每个一个格子内的边进行计算，然后当一列计算完毕后，即完成一个子点集中点的值的计算，窗口滑动到下一列，继续进行计算，直至所有的格子都遍历完毕。在这种计算模型当中，值的更新计算的操作必须符合交换律，另外，这种方式点的更新是就地更新，不会产生中间的更新结果，极大地减少了I/O，同时，点的数据访问的局部性也有了提升。在进行图划分时，使用二级的图划分策略，即先将图划分成Q大份，使得每个格子的边都能够存储进内存当中，然后再对刚才的每个格子进行划分，使得每个小格子能够存储进最后一级Cache当中。另外，GridGraph还支持选择性的调度，在BFS和WCC这样的算法中，可以极大地减少I/O，提高计算性能。



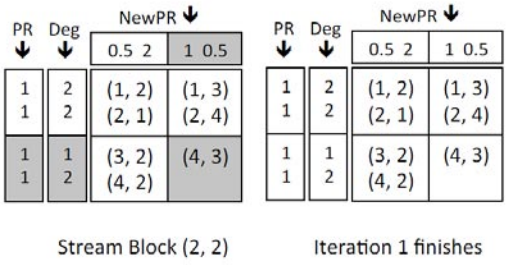


图11 双重滑动窗口计算模型示例[17]

5. 大图计算中的关键技术

本章，我们将介绍在分布式和单机图处理系统中常用的技术。

5.1 异构计算平台

在异构计算系统中，存在着计算能力和计算特点不同的计算单元。比如，GPU具有比CPU更强的多线程并行计算能力，因此在异构系统中，CPU会把一些或者全部的计算交给GPU来执行。在图计算领域，相关的异构计算系统已经被开发出来。TOTEM[22]，会将度高的点交给CPU计算执行，而将度低的点交给GPU来执行。而另外一些系统，如MapGraph[23]和CuSha[24]等，会将整个图都交给GPU来知行。除了，GPU和CPU的异构图计算平台之外，一些研究人员发现，Solid-State Drive (SSD)有着与传统Hard Disk Drive(HDD)不同的访存特性，一些图计算系统，如TurboGraph[25]，和FlashGraph[26]针对SSD对计算系统进行了优化，使得系统在SDD上有着很高的计算性能。现在，使用异构计算的平台的图处理系统主要是单机图处理系统。

5.2 通信模型

消息传递的通信模型，在这种通信模型当中，算法中点的状态保存在本地，通过消息传递的方式更新在其它机器上点的状态。在Pregel和Giraph中，使用了消息传递的通信模型，为了使得确保所有更新的数据可用，需要在前后两次迭代计算之间加入一个同步操作。

共享内存的通信模型，在这种通信模型当中，各个处理单元允许并发访问和修改相同地址的数据。在一些分布式的计算系统当中，如GraphLab和PowerGraph，使用了虚拟共享内存来实现各计算节点之间的透明的同步。在这些图处理系统中，使用了假点（ghost vertex）的方式来实现了虚拟共享内存。在假点的这种实现策略中，图中的每个点有一个归属的工作节点，另外有一些工作节点拥有该点的副本。因此在这种通信模型当中，当多个工作节点并

发访问同一内存地址时，需要考虑数据一致性的问题。

5.3 执行模型

同步执行，许多图算法由一系列迭代计算组成，在前后两次迭代之间，有一个全局的同步过程。这种执行模式将计算节点之间的通信控制在每次迭代的结束，因此适合于那些计算量小而通信量大的算法。

异步执行，在图中某个点的值有了更新值之后，立即将这个最新的更新值更新到该点上。在这种执行模式当中，节点之间的通信是不规则的，因此这种模式对于计算量不均衡，并且节点之间通信量小的算法非常适用。

5.4 图的划分

图的划分是进行高效图计算的一个关键问题，通常，一个理想的图划分情况是各工作节点的任务量基本相同，同时各工作节点之间的通信两最小，但是，这是一个NP难的问题。现在，常用的图划分算法分为三类。

第一类，首先对输入的图数据进行一个预处理步骤，将初始的图数据转化为某个特定的存储格式，使得图计算的访存局部性更好或者使图数据的数据量占用更少。比如GraphChi使用shard以及shard内存源点的排序来增强磁盘访存的局部性。另外，X-Stream使用简单的流划分来降低预处理的开销。

第二类，在算法执行过程中使用动态的重划分，由于算法在执行之前行为是无法预测的，所以这种动态划分的策略可以根据现有算法的执行状态，进行相应地划分，提高系统的性能。这种动态划分策略需要对图进行多次划分，引入了图划分开销。

第三类，使用Edge-cut和Vertex-cut划分。Edge-cut将图中的点均匀地划分，并且保证跨不同划分块之间的边最少。Vertex-cut将边均匀地划分，同时保证跨不同块之间的点最少。现实生活中的许多大图符合幂律分布[27](Power-law)，因此，相比于Edge-cut，使用Vertex-cut有助于系统的负载均衡，但是图计算系统需要使用以边为中心的计算模型，如PowerGraph。

5.5 负载均衡

负载均衡的算法分为静态负载均衡和动态负载均衡，静态负载均衡在算法执行之前进行任务的分配，但是由于算法在执行之前，我们无法预测其具体的行为，因而在算法的执行过程中可能出现负载不均衡的情况。动态的负载均衡策略针对静态负载策略进行了改进，即在算法的运行过程中，系统中任务少的工作节点可以从任务量大的工作节点“偷取”任务，来实现负载均衡，提高系统的整体性能。

5.6 容错

容错在分布式图处理系统中是需要解决的一个问题。在分布式处理系统当中，每台机器都会有一定的概率出错失效，如果不加以处理，将对系统产生严重的影响。常见的分布式图处理系统使用主从节点的方式，在这种构建方式中，主节点负责整个系统的管理和调度，从节点负责具体的计算。主要的容错方式有多副本策略，日志重做策略等。在多副本策略当中，在主工作节点执行其任务时，另外有一个工作节点作为副本工作节点会执行相同的任务，当主节点失效时，副本会接管主节点的工作任务，这种容错方式基本没有错误恢复时间，但是会消耗掉很多计算和内存资源。在日志重做的策略当中， 使用checkpoint或者log的方式，记录工作节点的计算操作，当机器出现失效时，可以将记录的操作重做来进行恢复，这种恢复方式以会消耗一定的恢复时间，但是对计算和内存资源的消耗相对较少。

6. 结论及未来研究方向

本文我们分别介绍了几个典型的分布式大图处理系统和单机大图处理系统，这两种类型的系统有着各自的优点和缺点。对于分布式系统，其特点是计算能力强，能够应对不同的计算需求，但是编程模型和系统的构建（计算的协调和容错机制）比较复杂；对于单机系统，其特点是编程和计算模型简单，硬件开销很低，但是计算能力有限，无法满足某些计算需求。从计算模型来看，现在大图计算的计算模型主要分为两种，以点为中心的计算模型和以边为中心的计算模型。在分布式的处理系统Pregel，GraphLab等，以及单机系统GraphChi主要使用了以点为中心的计算模型，这种计算模型更易于编程和理解，以边为中心的计算模型主要用于单机的系统，如X-Stream。除这两种主要的计算模型之外，还有一些系统如，从数据的局部性出发，提出以一些新的计算模型来提升系统的性能，但从本质上来说，这些计算模型是基于以点为中心的计算模型，只是针对数据的布局，做出了相应的修改。

尽管现在有许多针对大图计算系统的研究工作被提出，但是，从系统角度来看，在大图处理系统上还有许多值得深入研究的领域。在分布式图计算系统方面，设计一套高效合理的图划分策略，不仅

可以减少集群中各节点的通信开销，而且可以保证机器间的负载均衡，在这方面已经有一些相关的研究，但仍然值得更深入的研究。另外，容错也是分布式系统改善性能的一个重要方面，现在主要的容错方法有主副本备份容错，checking point容错等，在这方面的研究我们要解决在减少容错开销的同时尽可能地提高错误恢复的速度。在单机图计算系统方面，由于计算能力的限制，有效的图划分策略，并且使用与划分策略相匹配的计算模型来增强计算的局部性，是研究的热点。另一方面，应该充分发挥机器的多核特点，使得I/O和计算、以及计算并行，也是值得深入研究的方向。

参考文献：

[1] 腾讯科技 http://tech.qq.com/a/20140725/000288.htm.

[2] Lumsdaine Andrew, Gregor Douglas, Hendrickson Bruce, et al. Challenges in Parallel Graph Processing. Parallel Processing Letters 17, 2007, 5 - 20.

[3] Dean Jeffrey, Ghemawat Sanjay. MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 51(1), 2008, 107 - 113.

[4] Gregor Douglas, Lumsdaine Andrew. The Parallel BGL: A Generic Library for Distributed Graph Computations. Proc. of Parallel Object - Oriented Scientic Computing (POOSC), 2005.

[5] Chan Albert, Dehne Frank, Taylor Ryan. CGMGRAPH/CGMLIB: Implementing and Testing CGM Graph Algorithms on PC Clusters and Shared Memory Machines. International Journal of High Performance Computing Applications, 19(1), 2005, 81 - 97.

[6] Malewicz Grzegorz, Austern Matthew, Bik Aart J.C, et al. Pregel: a system for large - scale graph processing. SIGMOD, New York, NY, USA, 2010, pages 135 - 146.

[7] Apache Giraph. https://giraph.apache.org/.

[8] Low Yucheng, Bickson Danny, Gonzalez Joseph, et al. Distributed GraphLab: A framework for machine learning in the Cloud. PVLDB, 5(8), 2012, 716 - 727.

[9] Gonzalez Joseph E, Low Yucheng, Gu Haijie, et al. PowerGraph: Distributed graph - parallel computation on natural graphs. OSDI, Berkeley, CA, USA, 2012, pages 17 - 30.

[10] Gonzalez Joseph E, Xin Reynold S., Dave Ankur, et al. Graphx: Graph processing in a distributed dataflow framework. OSDI, Broomfield, CO, 2014, pages 599 - - 613.

[11] Chen Rong, Ding Xin, Wang Peng, et al. Computation and communication efficient graph processing with distributed immutable view. HPDC, New York, USA, 2014, pages 215 - 226.

[12] Yan Da, Cheng James, Lu Yi, et al. Blogel: A block - centric framework for distributed computation on real - world graphs. PVLDB, 7(14), 2014, 1981 - 1992.

[13] Yuan Pingpeng, Zhang Wenya, Xie Changfeng, et al. Fast Iterative Graph Computation: A Path Centric Approach. SC, NJ, USA , 2014, pages 401 - 412.

[14] Kyrola Aapo, Blelloch Guy, Guestrin Carlos, et al. GraphChi: Large - scale graph computation on just a PC. OSDI, Berkeley, CA, USA, 2012 pages 31 - 46.

[15] Roy Amitabha, Mihailovic Ivo, Zwaenepoel Willy. X - Stream: Edge - centric Graph Processing using Streaming Partitions. SOSP, New York, NY, USA, 2013, pages 472 - 488.

[16] Cheng Jiefeng, Liu Qin, Li Zhenguo, et al. VENUS: Vertex - Centric Streamlined Graph Computation on a Single PC. ICDE, Seoul, Korea, 2015, pages 1131 - 1142.

[17] Zhu Xiaowei, Han Wentao, Chen Wenguang. GridGraph: Large - Scale Graph Processing on a Single Machine Using 2 - Level Hierarchical Partitioning. ATC, Santa Clara, CA, 2015, pages 375—386.

[18] Valiant Leslie G. A Bridging Model for Parallel Computation. Communications of the ACM, 33(8), 1990, 103 - 111.

[19] Apache Hadoop. http://hadoop.apache.org/.

[20] Low Yucheng, Gonzalez Joseph, Kyrola Aapo, et al. GraphLab: A New Framework for Parallel Machine Learning. UAI, Catalina Island, California, USA, 2010.

[21] Apache Spark. http://spark.apache.org/.

[22] Gharaibeh Abdullah, Costa Lauro Beltrão, Santos-Neto, Elizeu, et al. On Graphs, GPUs, and Blind Dating: A Workload to Processor Matchmaking Quest. IEEE 27th International Symposium on Parallel and Distributed Processing, Washington, DC, USA, 2013, pages 851-862.
[23] Fu Zhisong, Personick Michael, Thompson Bryan. MapGraph: A High Level API for Fast Development of High Performance Graph Analytics on GPUs. GRADES, New York, NY, USA, 2014, pages 1-6.
[24] Khorasani Farzad, Vora Keval, Gupta Rajiv, et al. CuSha: Vertex-Centric Graph Processing on GPUs. HPDC, New York, NY, USA, 2014, pages 239-252.
[25] Han Wook-Shin, Lee Sangyeon, Park Kyungyeol, et al. TurboGraph: A Fast Parallel Graph Engine Handling Billion-scale Graphs in a Single PC. KDD, New York, NY, USA, 2013, pages 77-85.
[26] Zheng Da, Mhembere Disa, Burns Randal, et al. FlashGraph: processing billion-node graphs on an array of commodity SSDs. FAST, Berkeley, CA, USA, 2015, pages 45-58.
[27] Baraba´si Albert-La´szlo, Albert Re´ka. Emergence of Scaling in Random Networks. Science 286, 5439 (Oct.), 1999, 509-512.

要闻集锦

橡树岭国家实验室开始安装“山峰”超级计算机

据www.hpcwire.com网站2017年8月7日消息报道，美国橡树岭国家实验室（ORNL）近日已开始安装由IBM和NVIDIA公司联合制造的“山峰（Summit）”超级计算机。该系统安装完毕后有望成为当今世界上最快的超级计算机，也将是ORNL实验室迈向E级计算的最后一站。

第一批“山峰”系统的机柜已经于7月31日运抵ORNL，工作人员随即开始组装内部的计算和互连部件，并将它们接入其先进计算中心的电源和冷却设施。整机系统的安装将持续6个多月，安装完成后，开发人员可在初始级“山峰”机器上移植并优化大挑战（Grand Challenge）代码，以适应其新的CPU-GPU架构。ORNL计划于2019年1月将“山峰”系统正式提供给科学用户使用。

ORNL期望“山峰”整机系统能在2018年6月份的全球超级计算机TOP500排行榜中出现，以取代中国的“太湖之光”而占据TOP500榜首。2017年底，中国有望部署“天河-2a”超级计算机，报道性能为100Pflops左右，但美国业界认为其性能肯定会提高，因为中国仍有雄心要夺取TOP500桂冠。假设“山峰”系统由大约4600个节点构成，每个节点含有6个7.5Tflops的NVIDIA V100 GPU和2个IBM Power 9 CPU，那么仅依靠GPU，整机峰值性能就会超过200Pflops。ORNL打算今年10月或11月在部分“山峰”机器上运行Linpack，其性能将足以超过美国目前最快的超级计算机。但问题是，IBM公司到目前为止尚未正式发布Power 9处理器，可能要到2018年早期才能推出，当然在此之前肯定会有部分芯片提供使用。

除了为传统HPC应用提供无与伦比的计算能力外，“山峰”系统还可提供世界上最强大的深度学习应用平台。据宣传材料称，该系统将能提供高达3.3Eflops的深度学习性能（混合16/32位精度数学），这要归因于V100 GPU中专门为加速矩阵操作而设计的Tensor核心。

（柯 庆）