基于哈希存储器的大图生成器

李亚韬[1] 邵 斌[2]
[1]香港科技大学
[2]微软亚洲研究院

关键词：大数据 大图 哈希存储器

绪论

现实世界中的大规模图（简称大图）数据，例如网页链接拓扑和社交网络，规模都在十亿节点级别或以上。然而，获取真实大规模数据的代价是高昂的。除了少数商业公司，大部分研究者无法获得真实的大图数据。一个具有能快速生成某种近似真实数据特性的大图生成器将对研究人员实现和验证大图算法提供巨大帮助。

以前的研究工作专注于提出特定的生成算法，使得生成图具有某种特性（例如拓扑结构与社交网络类似，度分布符合Power-Law模型）。目前图研究中常用的生成器模型有Erdös-Rényi、BA和RMAT等。在Erdös-Rényi模型[1]中，任意两个节点间存在边的概率为p。因为有$|E|=|V|2*p$，V表示图的节点数，所以可以通过控制总边数$|E|$来控制p。如果用邻接矩阵来表示一个图，那么实际上在该模型下的一次生成相当于在邻接矩阵中随机挑选$|E|$个元素。虽然该模型非常朴素，且不具有真实数据的特性，但由于思想简单，容易在其基础上推导公式求解问题，所以该模型被广泛采用。RMAT模型[2]是近来最为流行的图生成模型之一。它算法简单，并且生成图具有很多真实数据特性。RMAT模型生成的图符合Power-Law分布，少量节点拥有大量边而绝大多数节点的边数都很小。RMAT图具有若干可调参数，根据参数的不同可以生成类似社交网络或网络拓扑等类型的合成图。如果将图视为一个邻接矩阵，RMAT算法迭代地在该矩阵中选取两点确定一个矩形（初始为整个矩阵）。当这个矩形中包含多于一个元素的时候，RMAT算法继续迭代，在每一步迭代中将当前矩形等分成四个象限，并按照一个随机数与给定的四个参数（分别对应选取四个象限的概率）来缩小当前矩形到某个象限。由于每一次缩小矩形的长宽都会减半，每一次生成边的复杂度是$O(\log(|V|))$。设四个象限的概率参数分别为p_1、p_2、p_3、p_4，满足$p_3>p_2$，$p_4>p_1$便可得到符合Power-Law分布的图结构。

目前基于各种图生成模型，存在一些流行的图生成器系统，比如GTGraph[3]、NetworkX[4]、GraphStream[5]和Gephi[6]等。GTGraph支持流行的RMAT模型，NetworkX则可以支持多种图生成器模型。GraphStream侧重于小规模图的可视化，而Gephi则侧重于图数据的交互探索。这些图生成系统都是单机系统，无法胜任十亿和百亿节点规模大图的生成。

本文作者设计了一种在分布式环境下具有良好扩放性的生成器系统，实现了大图的高速生成。在传统的生成器方案中，需要对生成的边表做排序（通常是在磁盘上做外排序）。对海量边集进行排序的时间开销很大，用传统方法生成一张十亿节点、百亿边的大图通常需要几十个小时。作者设计的生成器系统的优点是回避了耗时的外排序过程，并且在分布式环境下实现了高效的多机并行生成。

基于哈希的分布式存储器

我们的目标是在给定参数和图模型后生成一张合成图。一个图由顶点集合V和边集$E \subseteq V \times V$构成。在初始状态下，设$V=\emptyset$且$E=\emptyset$，生成图的过程可以描述为根据图生成算法不断往V与E中添加元素的过程。在有N台机器构成的分布式环境下，目标图G被划分为N份。我们用一个哈希函数$h(v)$来执行图的划分。亦即，顶点v将被划分到机器$h(v)$上，其中$h(v) \in \{1,2,...,N\}$。当生成过程结束后，生成图将被导出到磁盘文件中。

为了达到高速生成图的目的，需要对生成器算法实行并行化。为此，做出以下假设：

1 只生成边，不删除边；

2 不查询已生成的边；

3 生成器算法仅在系统请求时返回一条边；

4 在有多个生成器算法实例的情况下，这些生成器实例相互不依赖。

假设1是不删除已生成边的一个图节点的边集只线性增长，不需要对已生成边集进行维护。假设2不查询已生成边，意味着不需要对已生成边建立用于查找的索引。假设3意味着边集生成器是被动的，边生成的节奏和速率由上层控制算法控制。在分布式多生成器实例存在的情况下，被动式生成器算法使得我们可以在发生网络拥塞的情况下停止生成新边，避免在本地生成器中缓存过多边，造成不必要的存储开销。假设4是生成器算法可以并行化的根本保证，不同生成器实例互不依赖，因而不需要互斥和全局同步，从而所有的生成器实例可以零等待地全速运行。该假设保证了不同的生成器运行实例之间不会出现阻挡对方进度的情况。这使得生成器的性能可以随着处理器数量的增加而线性提高。这些假设虽然在一定程度上约束了生成器算法，但是对很多现有的生成器算法都是适用的。无论是将现有图数据导入（此时生成器只需要每次从输入文件中读入一条边即可）还是分阶段多模型生成，均符合我们对图生成器算法的假设。

在图生成过程中，新产生的边将被不断地添加到生成图中。在此过程中，我们只需要存储有边的节点。因而主要的问题是如何维护边数据，存储点的问题将随着边存储的实现而自动解决。此处“维护”意味着将产生的边序列转换为中间结果存储在中间结果存储器中，并在生成结束后以一定的格式导出到磁盘文件中。常用的描述图的数据结构有两种：邻接矩阵和邻接边表。无论采用何种形式存储，本质上都要求将所有的边按照源节点进行归类，亦即将具有相同源节点的边归为一组。这表明无论选择何种输出格式，维护边数据的主要问题在于如何对生成的边序列实施归类存储。

在我们的系统中，会实时地进行边的分组归类，而不是顺序追加新边。籍此我们可以用维护归类的时间代价换取额外的存储空间。如果使用顺序追加，对于每一条边都需要记录其源节点和目标节点编号；而如果实时地实施了归类，则同源的边只需存储目标节点编号。更重要的是，如果没有实时地实施边归类，当生成过程结束后，需要对所有的边进行排序。对一个具有十亿节点、百亿边的边表文件做一次外排序通常需要花几十个小时。即使排序过程完全在内存中执行，$O(|E|\log(|E|))$的时间复杂度下限仍是无法避免的。当目标图足够大的时候，$\log(|E|)$系数将使得性能与$O(|E|)$的算法产生非常明显的区别。对大规模边表的排序不仅会降低性能，而且会降低系统可生成图的大小。

实时的边归类可以避免之后费时的排序过程，但需要频繁执行细粒度的插入操作。为此我们需要设计一个高效的中间结果存储器。在本系统中，我们设计了一种基于哈希的分布式存储器以高效支持大量细粒度插入操作。

我们将整个内存地址空间视为一个大的哈希空间并向下对齐到不超过上限值的最大质数。在存储图节点时，将一个64位的整形节点编号映射到这个空间中作为该节点的存储地址。由于在存储器中仅存储节点编号（没有字符串等变长结构），所以将最小存储粒度设定为8字节。由于内存空间有限，并且每一个节点占用的空间是不确定的，不同图节点的存储地址可能发生冲突，为此我们采用多重哈希的方法来解决地址冲突。在为一个节点寻址的时候，将哈希冲突的次数记为i，并定义一组哈希函

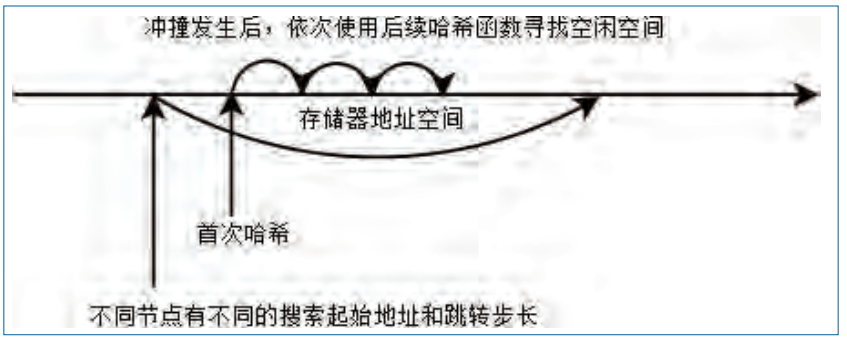

图1　节点寻址示意图

数，令$H_i(x)=[G(x)+i*(1+(((G(x)>>5)+1)\bmod(S-1)))]\bmod S$，其中$S$为哈希地址空间的总大小，$G(x)$为主哈希函数。由于$S$是一个质数，对任意的$i,j\in Z$，若想有$i*C\equiv j*C(\bmod S)$，则必须有$(i-j)*C=k*S$，而$(C,S)=1$，故$i-j=q*S$。因此，若有两次尝试的哈希函数值冲撞，则意味着已经尝试过S的整数倍次——这也就意味着我们可以使用$[0..S)$不重复地遍历整个地址空间，并且对于不同的x，每次重试的步长是不同的。图1展示了两个不同节点的寻址过程。

当我们通过多重哈希函数得到一个图节点的存储地址之后，需要判断当前地址指向的内存空间是否属于当前节点。为此我们规定了一些特殊的节点编号：

- -0代表空闲单元；
- -1代表当前节点存储空间结束；
- -2代表“节点头”；
- -3代表“带锁节点头”；
- -4代表“保留单元”。

如果当前地址指向的值既不是“节点头”也不是“带锁节点头”，那么当前单元节点不是从来没有被分配过（否则会被当前线程找到），就是已经被另外的节点占用。当找到空闲单元的时候，我们会占用这个单元并开始分配空间。此外，我们在每个节点头上设置一个自旋锁，使用原子操作让其在“节点头”和“带锁节点头”两种状态中切换。保证同一个节点不会被两个线程重入。虽然这个锁设置在非常细的粒度上，但是由于两个线程重入一个图节点的概率非常低，故阻塞在锁上所产生的性能开销可以忽略不计。为了避免频繁扩展同一个节点的存储空间，我们在分配内存空间的时候会为节点预留一些空间（以-4填充）。当一个节点需要扩展的时候，首先检查其后是否有预留空间。若无预留空间，则使用原子操作尝试将其后紧随的空闲单元申请为保留单元。若申请失败，就意味着当前节点在扩展过程中遇到了另外一个节点分配的内存空间。此时需要再次进入哈希过程，重新寻找空闲的地址，并在当前节点末尾的存储单元上标记跳转地址，形成链式存储结构。图2展示了节点的结构图，而图3则展示了内存空间申请的过程。

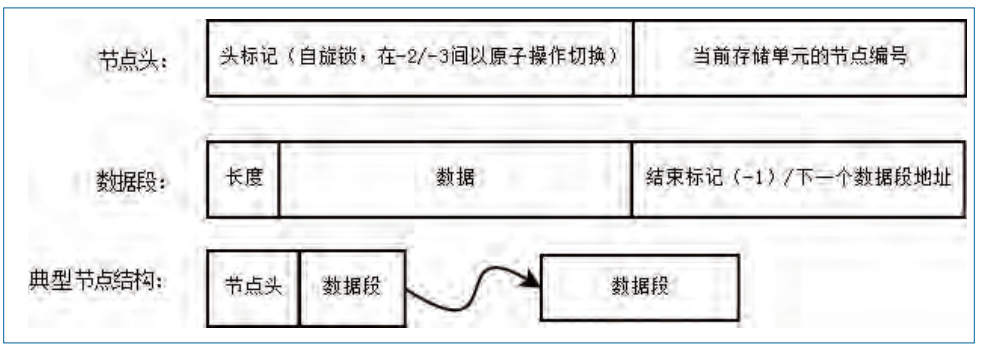

图2　节点结构示意图

系统实现

在一个有N台机器组成的分布式环境中，一条边(v_1,v_2)将被发送至目标机器$h(v_1)$。由于生成器算法与中间结果存储器都对机器的编号不敏感，并且

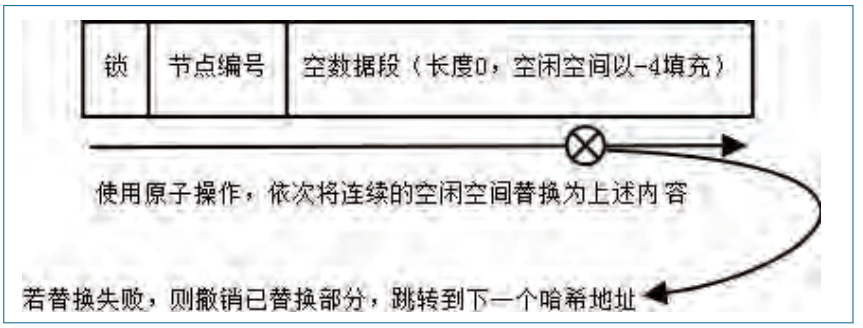

图3　节点空间分配示意图

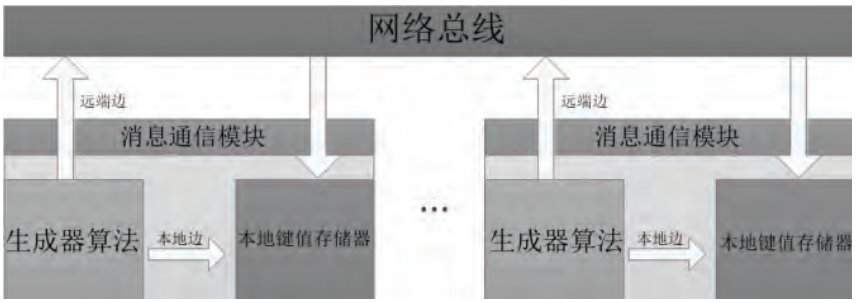

图4　系统架构示意图

生成器算法是被动的，我们有足够的自由度来构建一个多机通信子系统以决定生成边的目的地、发送方式与时机。为了及时探测网络拥塞，我们采用同步通信的方式进行发送。在当前发送完成前，下一次发送任务将被阻塞，同时也阻塞了新边的生成。如此，通信模块可在不受缓存压力影响的情况下选择合适的发送方法使吞吐率最大化。

本系统的消息通讯采用Trinity[7]作为底层支持。图4展示了本系统的架构。与采用消息传递接口（message passing interface，MPI）进行多机通信相比，Trinity自带的通信模块更适合完成生成边的传送任务，因为它能更好地整合利用操作系统提供的资源。为了实现对CPU的饱和利用并以最快速度生成边，集群中的每台机器应运行与CPU核心数量相当的生成器算法实例。在MPI编程模型中，这意味着需要在同一台机器上运行多个MPI实例，也就是多个独立进程。由于这些MPI实例只能通过套接字（socket）传递消息，同一台机器上不同实例间的通信将浪费计算和存储资源。此外，由于操作系统可以同时提供的socket数量是有限的，随着生成器实例数量的增加，保持所有MPI实例间的全连接将变得越来越困难。而在Trinity提供的通信模块中，同一台机器上的不同生成器实例处于同一进程，它们之间可以共享网络连接。在我们的应用场景中，Trinity通信模块比MPI拥有更高的系统资源利用率和更好的性能。

当所有追加边的操作结束后，需要把中间结果存储器中的数据导出到磁盘文件中。由于在边生成的过程中，没有使用存储索引记录节点的位置。需要扫描整个存储器以找到所有的节点。为此，我们把整个哈希空间划分成若干份，并分配多个线程并行扫描整个内存空间。扫描过程非常简单，只需搜索节点头，命中即找到一个节点。需要注意的是，由于生成器算法并不查询当前已生成的数据，在存储器中有可能出现重边。因此，在扫描一个节点的时候，需要做一次去重边操作。经过不断地实验和调整后，我们发现对于大量小规模去重操作，采用排序比采用哈希集合的效率要高。得到去重的边集后，我们会将最终的生成图导出到Trinity磁盘文件中。

实验结果

我们在一个由8台机器组成的小型集群中进行实验。

我们从运行时间和哈希冲撞次数两个方面测试了系统的性能。为了测试系统的各项特性，我们在不同机器数量和不同数据规模下进行了多项实验。在实验中，我们采用每次倍增数据量的方法尽可能

扩大数据规模，直到占满当前配置的所有内存。在实验图中，每一条曲线都是按这种方式测绘而成。为了简便起见，设定图中节点的平均度为10，并设定每台机器上生成器算法线程的数量等于CPU核心数量，以充分利用硬件资源。

从图5中可以看出，随着数据规模的增长，相同配置上的运行时间基本是线性增长的，而增加机器会线性降低系统的运行时间。这表明本系统具有良好的性能扩放性。在128M节点这组测试中，1台机器的运行时间低于两台，而4台的运行时间低于7台，这是因为测试数据太少，网络通信的开销没有被系统速度的优势掩盖住。

反映哈希冲撞次数的图6可以排除由运行时的不确定因素与通信开销等带来的偏差，更清楚地展示了系统良好的扩放性。从实验数据可以看出，随着数据规模增加，单台机器上的冲撞次数并不是随之线性增加的。这是因为机器的总容量是有上限的，随着哈希空间里元素的增多，剩余空间会越来越少，导致冲突的增加。在实验中，每台机器拥有48GB内存。对于单机200M点2B（byte）边的情况，由于存储的是双向边，实际上需要存储4B边，每个8字节，共计32GB，系统内存利用率大约为66%，此时的哈希冲撞次数在10^8级别，均摊到每条边上的冲撞期望值不足1。

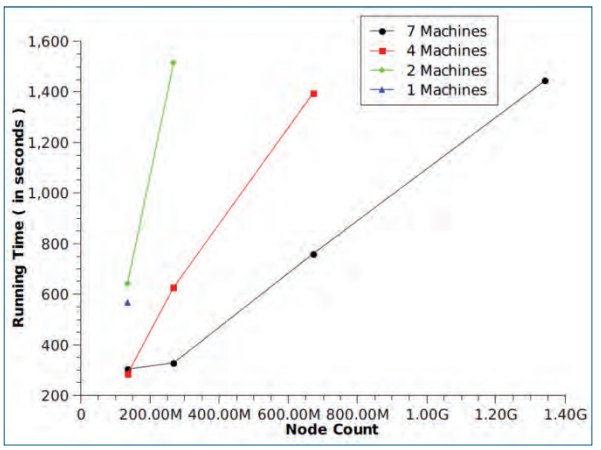

图5　运行时间

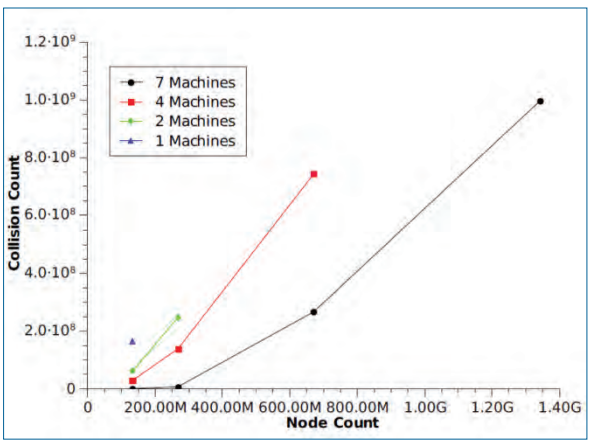

图6　冲撞次数

本系统目前尚未实现对BA等图生成模型的支持，该类图生成模型需要密集查询当前已生成的图节点数据。BA等图生成器对已生成边的访问有一些固定模式，例如总是访问度最大的点等。未来我们将利用这些数据访问的特点，改进当前的设计，使之支持带查询的图生成模型。■

李亚韬

香港科技大学博士生。主要研究方向为分布式图数据库。
glocklee@gmail.com

邵　斌

CCF会员。微软亚洲研究院副研究员。主要研究方向为分布式图数据库、数据并行处理和分布式内存系统的设计与优化等。

参考文献

[1] D. Chakrabarti and C. Faloutsos. Graph min- ing: Laws, generators, and algorithms. ACM Comput. Surv., 38(1), June 2006

[2] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R- MAT: A Recursive Model for Graph Mining. In SIAM International Conference on Data Mining, 2004

更多参考文献：www.ccf.org.cn/cccf