X-Stream: Edge-centric Graph Processing using Streaming Partitions

1. 系统概述

a）不需要GraphChi的预处理和后处理的阶段（主要是分片和排序的开销）。

b）以边为核心的计算框架。

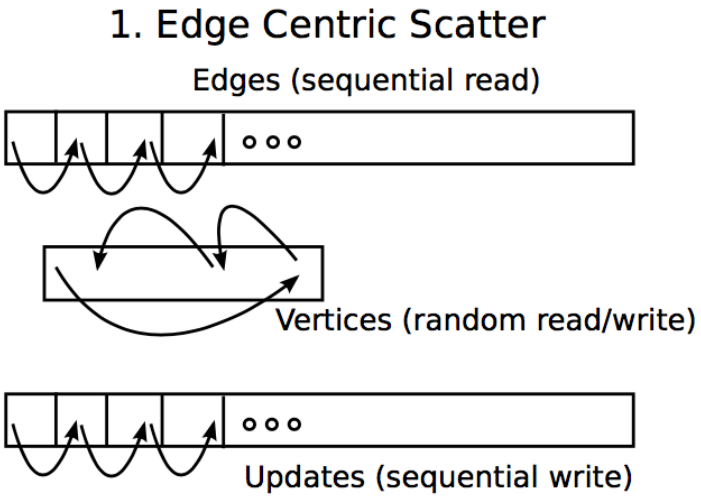
c）采用同步的计算方式来保证计算的正确性。

d）系统分为外存-内存Engine和内存-CPU Engine两个部分。

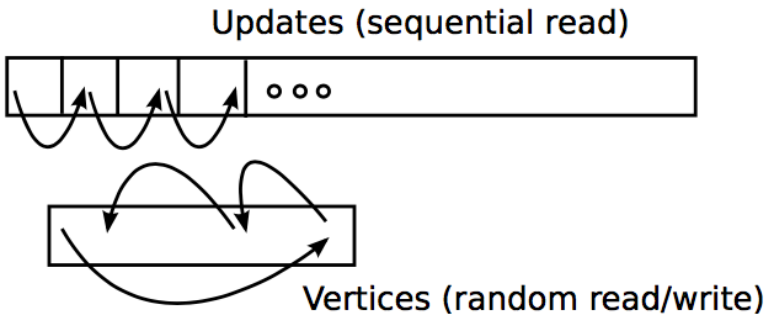
X-stream中，stream指的是边的数据流，系统不对边做任何聚合、排序等操作，将外存中的边的信息以流的形式读入内存进行处理，这样就省去了预处理的时间。

2. 单Stream

假设所有的节点信息都能存入内存中（而且内存还有足够的空间缓存其他的数据），整个系统的图计算模式一下如下图所示：



2. Edge Centric Gather



整个过程分为两个阶段——scatter和gather。scatter阶段主要计算边的函数以此来生成更新，并将更新写入buffer当中；gather阶段主要讲scatter阶段计算的更新写入节点数组当中。这两个阶段算是一个step，系统就是一个step一个step进行迭代操作，直到达到了一个预设的终止条件，每个step之内不分计算顺序，但是不同step之间计算顺序严格区分，所以下一个step进行之前，必须要等上一个step完全完成之后，这也就是X-Stream系统所采用的**同步的计算方式**。

3. Streaming Partitions

实际情况中一般节点信息是不能完全存入内存中的，所以才用了分片的机制。每一个partition中包含三个信息：Vertex Set（存储一部分节点信息）；Edge Set（存放Source在Vertex Set的边的集合）；Update List（存放destination在Vertex Set的更新的集合）。其中Vertex Set和Edge Set一旦初始化完成后就不会变化，所以X-Stream不适合做动态变化的图上的计算。整个流程如下：

```
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)
```

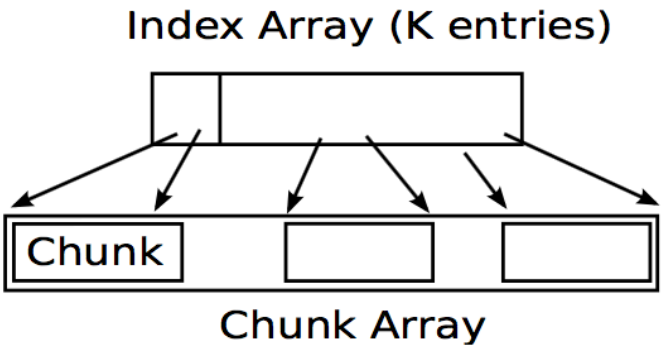
增加了shuffle阶段，用来将scatter阶段产生的更新分配给每个partition。整个流程变为：scatter（计算更新）→ shuffle（分配更新）→ gather（更新节点值）。其中scatter和gather阶段是天然的可以进行并行计算的。

4. Out-of-core Streaming Engine

我们可以看到上面流程的瓶颈在于shuffle阶段，所以将shuffle阶段与scatter进行合并，这样做的好处在于减少了对外存的随机读写：

```
merged scatter/shuffle phase:
  for each streaming partition s
    while edges left in s
      load next chunk of edges into input buffer
      for each edge e in memory
        edge_scatter(e) appending to output buffer
        if output buffer is full or no more edges
          in-memory shuffle output buffer
          for each streaming partition p
            append chunk p to update file for p
```

为了方便对scatter和gather进行多线程的操作，系统采用了新的数据结构——Stream Buffer。



其中K为分片的总数，每个partition单独进行计算的时候chunk用于存放一部分数据，处理完之后，再将剩下的部分读取进来处理，直到处理完成。整个系统提前在内存中申请空间，减少动态申请空间的开销。采用RAID的架构以此来增加外存到内存的带宽，加快处理过程。

5. In-memory Streaming Engine

总体架构与之前的差不多，不同的是内存引擎需要具体实现多线程的操作，而且为了保证CPU的cache的命中率，分片的大小要与CPU的cache相同，所以需要处理的partition的数量就会变得十分多。

因为缺少均匀分片的操作，所以每个线程分到的partition很容易出现负载不均衡的情况，为了解决这个问题，X-Stream采用了steal机制，允许一个进程在处理完任务之后，去 “steal” 其他未完成任务的线程的任务。

6. 总结

由于X-Stream是以边为核心的计算框架，所以对于以节点为核心的图算法来说其实效果并不是很好。因为边的数量一般远大于点的数量，所以以边为核心的计算模型的计算量会很大，虽然省去了预处理的时间，但是提升效果很有限。但也有可以借鉴的点，X-Stream尽量完全使用外存到内存的带宽，而且尽量扩展此带宽，因为这个是所有单机计算框架的瓶颈所在，其在这方面做的优化值得借鉴。