GraphChi: Large-Scale Graph Computation on Just a PC

主要成果

1. A novel method -- parallel sliding windows （PSW）

2. a disk-based system for Graph computation-- GraphChi

Introduction

图计算中，分布式系统的图划分问题、通信问题和一致性问题。当计算的图的大小变的巨大的时候，人们自然而然就想到使用分布式的解决方案，但是不同于其他的大规模计算问题，图计算的分布式解决方案中需要使用图划分、works 之间通信、计算结果的容错等问题，事实证明这些问题是很难去完美解决的，所以作者就提出了一个观点，**是否可以在单机上进行图计算。**

大部分分布式系统都是实现的同步操作，异步实现的比较少。由于图上存在着计算顺序的问题，大部分的分布式系统都是采用 step→step 的方式进行操作。step 之内可以进行并行异步的操作，step 之间必须按照严格的先后顺序执行，这就会造成资源空闲浪费的情况。

虽然单机上运行不存在或者是减轻了上述的一系列的问题，但是单机图运算也有着自身的技术问题：

随机访问问题。由于图的规模过大，必须要一部分一部分从外存读到内存中进行处理，而且节点和边的修改必须要及时写入外存中去，这其中就会充斥着大量的外存随机读写问题，这将大大降低单机的效率。

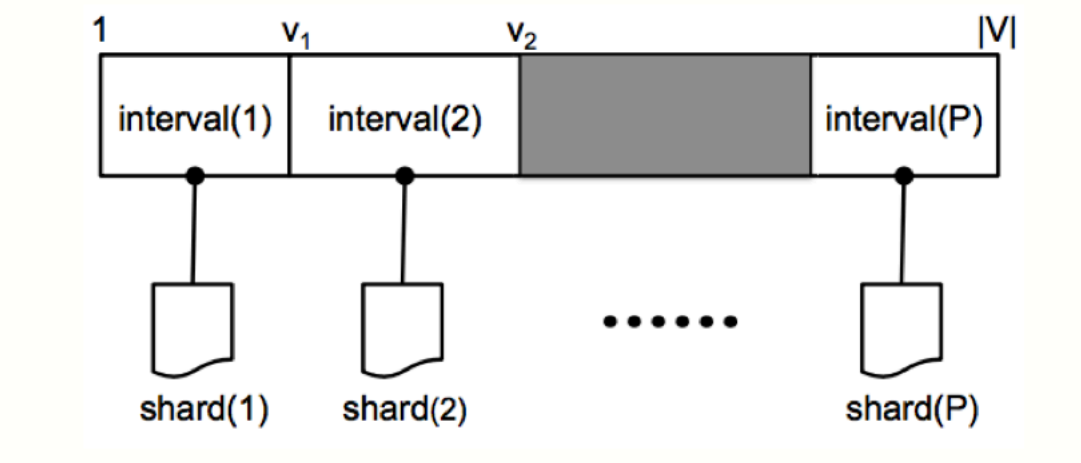
有人提出了一些解决方案：

SSD：的确会加快硬盘的读写速度，但是因为随机读和随机写的数量基本一致，SSD 的随机写的性能并不强，所以提高的能力很有限。

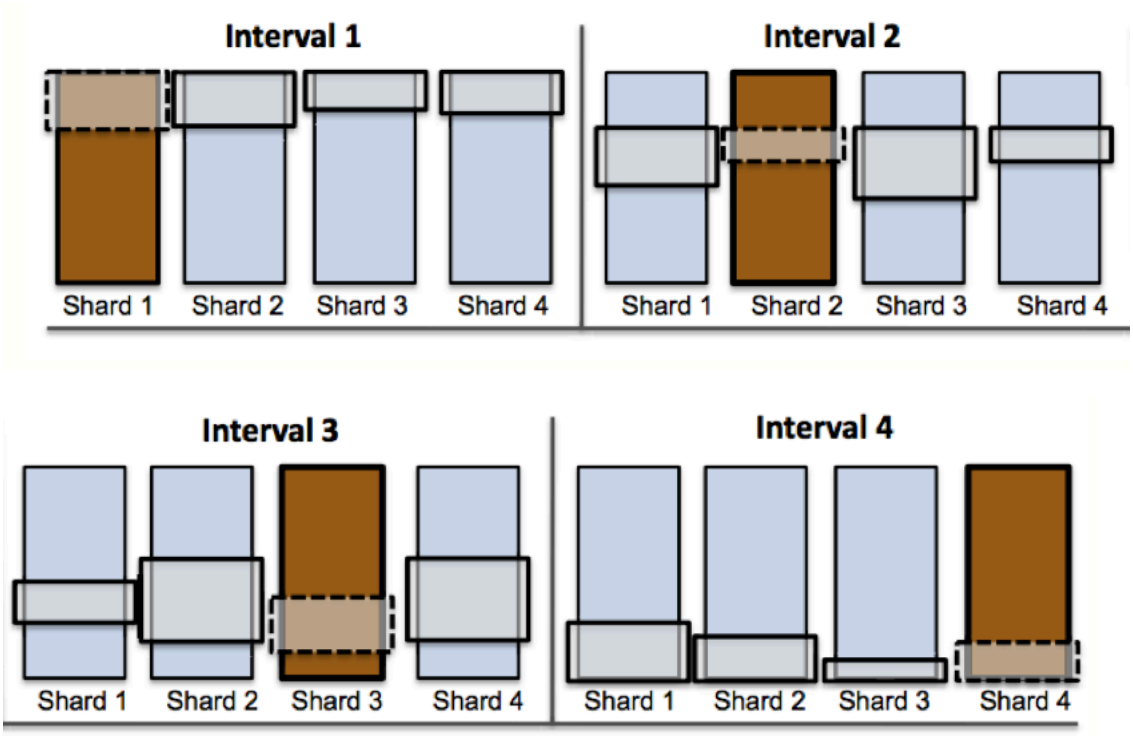
利用图的局部特性：现实世界里的图存在着聚簇的现象，存在一些比较重要的节点，尝试使用内存存重要节点，硬盘存次要节点方式。但是这个方法的效果并不好，原因有二：结果的不可预测性（取决于图的结构）；优化图的存储的消耗很大，而且有时是不可做到的。

图压缩：优化图的存储结构，尽量减少图的存储空间，从而减少分块的块数，是一个很好的 idea，但是很快这个就走不下去了，因为压缩总是有极限的。

Parallel Sliding Windows（PSW）



如上图所示，GraphChi 将节点集分为 P 个 interval 中，每个 interval 有一个 shard 集，其中主要存储 destination 在相应的 interval 的边的集合，而且边按照 source 的编号进行排序，以便实现之后的 PSW 操作。其中 interval 的分片原则就是其 shard 能够比较均衡，而且足够全部放入内存当中。



上图就是 PSW 的一个运行示例图。首先，读入棕色框的 shard 和其对应的 interval（如果节点集不大的话，可以直接全部放在内存当中），运行边上的图算法，对节点进行更新，然后将节点的更新接着传到以其为起点的边上，由于之前有

排序操作，所以每次更新边的时候，并不需要把所有的 shard 都读入内存，只需要读取图中方框所框住的部分，这样就大大减少了 I / O 的时间。

下图就是更加详细的一个例子：

Shard 1

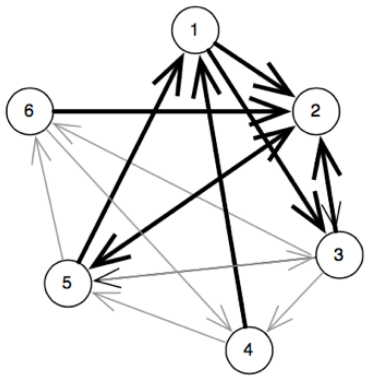
src	dst	value
1		
	2	0.3
3		
	2	0.2
4		
	1	1.4
5		
	1	0.5
	2	0.6
6		
	2	0.8

Shard 2

src	dst	value
1		
	3	0.4
2		
	3	0.3
3		
	4	0.8
5		
	3	0.2
6		
	4	1.9

Shard 3

src	dst	value
2		
	5	0.6
3		
	5	0.9
	6	1.2
4		
	5	0.3
5		
	6	1.1



Shard 1

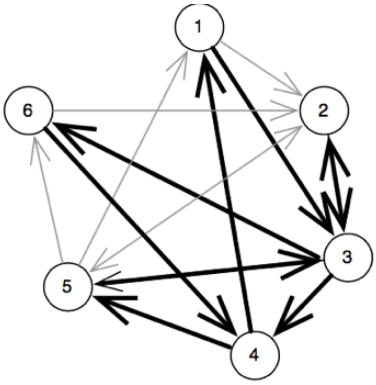
src	dst	value
1		
	2	0.273
3		
	2	0.22
4		
	1	1.54
5		
	1	0.55
	2	0.66
6		
	2	0.88

Shard 2

src	dst	value
1		
	3	0.364
2		
	3	0.273
3		
	4	0.8
5		
	3	0.2
6		
	4	1.9

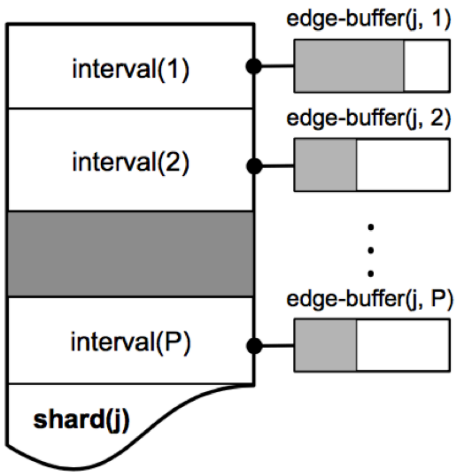
Shard 3

src	dst	value
2		
	5	0.545
3		
	5	0.9
	6	1.2
4		
	5	0.3
5		
	6	1.1



黄色的就是需要读取到内存的内容，因为 interval 中的节点是按顺序划分的，每次更新 shard 中的边的信息的时候，读取的部分像是一个窗口逐步往下滑动，所以论文中称其为 Parallel Sliding Windows。

为了适应图结构的实时更新，PSW 做了一些改进，如下图所示：



将每个 shard 逻辑上再按照 interval 的数量进行分片，每个分片建立一个 buffer，用来存放新加的节点，当 buffer 中的数量达到一个阈值（用户自定义）之

后，就会在下次写入外存的时候一同将 buffer 的内容写入外存中，相当于融入到 shard 中，当每个 shard 的大小增大到不能全部放进内存里的时候，就会将一个 shard 块分为两个。这样就实现了**图的动态改变**。

GraphChi

利用上述 PSW 的核心架构，本文实现了一个系统——GraphChi。

存储结构：如果图使用矩阵来表示的话，该矩阵大部分情况下都是一个稀疏矩阵，所以在 GraphChi 中对于边的存储使用链表的结构，同 source 的边存储在一个链表当中，既节约了存储空间，又方便了程序读取使用。

预处理：预处理主要是计算内存的空间和具体能使用的空间的大小，然后对整个图进行分片，然后将分片结果存储到外存当中。

多线程：对于每一个 shard 的滑动窗口的更新可以使用多线程并行的方式来处理，但是如果出现了一条边的起点和终点都在一个 interval 中时，则需要按照顺序串行来执行。

有选择的更新：建立一个个关于节点的 bit / array，根绝索引来判断是否要更新这个节点的边的值。

编程限制：GraphChi 对于每个节点来说只能读取相邻边上的值，不能读取相邻节点的值，对于算法编程来说增加了一些难度，但是保证了算法的高度并行性。

Experimental Evaluation

预处理阶段

Graph name	Vertices	Edges	P	Preproc.
live-journal [3]	4.8M	69M	3	0.5 min
netflix [6]	0.5M	99M	20	1 min
domain [44]	26M	0.37B	20	2 min
twitter-2010 [26]	42M	1.5B	20	10 min
uk-2007-05 [11]	106M	3.7B	40	31 min
uk-union [11]	133M	5.4B	50	33 min
yahoo-web [44]	1.4B	6.6B	50	37 min

运行阶段

Application & Graph	Iter.	Comparative result	GraphChi (Mac Mini)	Ref
Pagerank & domain	3	GraphLab[30] on AMD server (8 CPUs) **87 s**	**132 s**	-
Pagerank & twitter-2010	5	Spark [45] with 50 nodes (100 CPUs): **486.6 s**	**790 s**	[38]
Pagerank & V=105M, E=3.7B	100	Stanford GPS, 30 EC2 nodes (60 virt. cores), **144 min**	approx. **581 min**	[37]
Pagerank & V=1.0B, E=18.5B	1	Piccolo, 100 EC2 instances (200 cores) **70 s**	approx. **26 min**	[36]
Webgraph-BP & yahoo-web	1	Pegasus (Hadoop) on 100 machines: **22 min**	**27 min**	[22]
ALS & netflix-mm, D=20	10	GraphLab on AMD server: **4.7 min**	**9.8 min** (in-mem) **40 min** (edge-repl.)	[30]
Triangle-count & twitter-2010	-	Hadoop, 1636 nodes: **423 min**	**60 min**	[39]
Pagerank & twitter-2010	1	PowerGraph, 64 x 8 cores: **3.6 s**	**158 s**	[20]
Triange-count & twitter- 2010	-	PowerGraph, 64 x 8 cores: **1.5 min**	**60 min**	[20]

从测试结果上来看，GraphChi 在大部分的情况下不会差于分布式图计算系统所耗时的两倍，所以单机图计算系统的能力是可以期待的，因为其更轻便，更灵活。

从红框里面的内容可以看出来 Map-Reduce 架构的确很不适合做图计算，其分布式的计算架构性能远远比不上单机版 GraphChi。