Pregel：一个大规模图计算系统

李妍妍 [1]

（1 广州大学 计算机科学与网络工程学院，广东 广州 510000）

摘 要：在许多实际计算问题中，大数据都是以大规模图或网络的形式呈现。而传统的图计算存在着诸如内存访问局部性等问题。Google 在 2010 年提出了一种适用于此任务的计算模型，即 Pregel。Pregel 是一种基于 BSP 模型实现的并行图处理系统。在该系统中，程序是一系列的迭代，每个顶点都可以接收在前一个迭代中发送的消息，然后发送消息给其他顶点，并修改自己的状态和其输出边的状态，或改变图拓扑结构。这种以顶点为中心的方法足够灵活，可以表达一组广泛的算法。Pregel 就搭建了一套可扩展的、有容错机制的平台，该平台提供了一套非常灵活的 API，可以描述各种各样的图计算。本文详细的介绍了并行计算模型 BSP 和 Pregel 的基本原理及其基础架构。

关键词：大数据，分布式计算，图算法，Pregel

1 引 言

互联网使网格化图形成为分析和研究的热门对象。例如运输路线、报纸文章的相似性、疾病爆发的路径、或发表的科学作品之间的引用关系。常用的算法包括最短路径计算、不同类型的聚类算法，还有许多具有实用价值的图计算问题，如最小割和连通分量等。高效处理这些大型图形常常是具有挑战性。传统的图算法通常表现出较差的内存访问局域性，每个顶点的工作量很少，并且在执行过程中并行度不断变化，分布在多台机器上加剧了局部性问题，并增加了机器在计算过程中失败的概率。Pregel 是 Google 自 2009 年开始对外公开的图计算算法和系统，主要用于解决无法在单机环境下计算的大规模图论计算问题。与其说 Pregel 是图计算的算法，不如说它是一系列算法、模型和系统设计组合在一起形成的一套图模型处理方案。图计算的实际应用非常广泛，因此自 Pregel 公开之后，一些开源的方案也被实现出来，比如说来自 Spark 的 GraphX 就实现了 PregelAPI。值得注意的是，Pregel 作为一个近十年前起就为人所知的算法，虽然新近也已经提出了不少增强和改进的技术，但在现实场景下仍然是很有生命力的。

目前的图计算模型基本上都遵循 BSP 计算模式。在详细介绍 Pregel 模型之前，我们先简单了解一下 BSP 模式的相关概念。

2 BSP 模式

BSP(Bulk Synchronous Parallel，整体同步并行) 是一种并行计算模式，由英国计算机科学家 Viliant 在上世纪 80 年代提出。BSP 计算模式如图 1 所示：

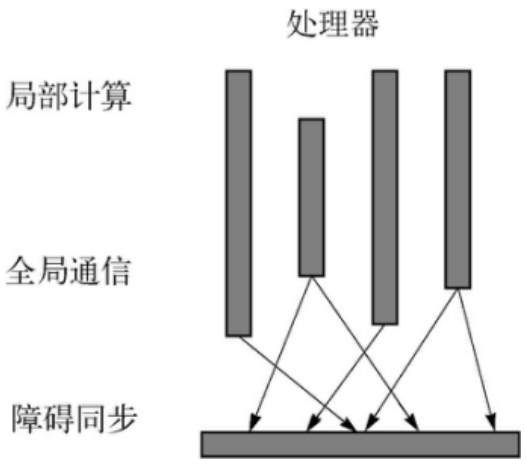

图 1: BSP 计算模式图

作为计算机语言和体系结构之间的桥梁，BSP 使用下面三个参数（或属性）来描述的分布存储的多处理器模型：处理器/储器模块；执行以时间间隔 L 为周期的所谓路障同步器和储器模块对之间点到点传递消息的选路器。

所以 BSP 模型将并行机的特性抽象为三个定量参数 p、g、L, 分别对应于处理器数、选路器吞吐率 (亦称带宽因子)、全局同步之间的时间间隔。

2.1 BSP 模型中的计算行为

在 BSP 模型中，计算过程是由一系列用全局同步分开的周期为 L 的超级步（supersteps) 所组成的。一个抽象的程序由分布在 n 个处理器上的 p 个进程或线程组成，并且被分为 superstep。在各 superstep 中，每个处理器均执行局部计算，并通过选路器接收和发送消息；然后做一全局检查，以确定该超级步是否已由所有的处理器完成：若是，则前进到下一超级步，否则下一个 L 周期被分配给未曾完成的超级步。BSP 计算模式中以下几个概念需要了解一下：

（1）Processors: 并行计算进程，它对应到集群中的多个结点，每个结点可以有多个 Processor。

（2）LocalComputation: 单个 Processor 的计算，每个 Processor 都会切分一些结点作计算。

（3）Communication: Processor 之间的通信。接触的图计算往往需要做些递归或是使用全局变量，在 BSP 模型中，对图结点的访问分布到了不同的 Processor 中，并且哪怕是具有局部聚类特点的结点也未必会分布到同一 Processor 上，所有需要用到的数据都需要通过 Processor 之间的消息传递来实现同步。

（4）BarrierSynchronization: 栅栏同步。每一次同步也是一个超步的完成和下一个超步的开始。

（5）Superstep: 超步，这是 BSP 的一次计算迭代，拿图的广度优先遍历来举例，从起始结点每往前步进一层对应一个超步。

任务结束，一个作业可以选出一个 Proceessor 作为 Master，每个 Processor 每完成一个 Superstep 都向 Master 反馈完成情况，Master 在 N 个 Superstep 之后发现所有 Processor 都没有计算可做了，便通知所有 Processor 结束并退出任务。

2.2 BSP 成本分析（Computational analysis）

考虑一个由 S 超步骤组成的 BSP 程序。那么，超步骤 i 的执行时间为：

$$T_{super} = \max_{processes} + \max gh_i + L \tag{1}$$

其中， wi 是进程 i（process i）的局部计算函数， hi 是进程 i 发送或接收的最大数据包，g 是带宽的倒数（时间步/数据包），L 是路障同步时间（注意我们不考虑 I/O 的传送时间）。所以，在 BSP 计算中，如果使用 S 个超级步，则总运行时间为：

$$T_{BSP} = \sum_{i=0}^{S-1} w_i + g\sum_{i=0}^{S-1} h_i + SL \tag{2}$$

w_i 和 h_i 分别被称为超步骤的深度和程序的深度

3 Pregel 的计算模型

在 BSP 模式的基础上，我们详细解释一下 Pregel 模型的原理。Pregel 在概念模型上遵循 BSP 模式。整个计算过程由若干顺序运行的超级步（Super Step）组成，系统从一个“超级步”迈向下一个“超级步”，直到达到算法的终止条件。一个典型的 Pregel 计算过程如下：其计算过程遵循下面的步骤：

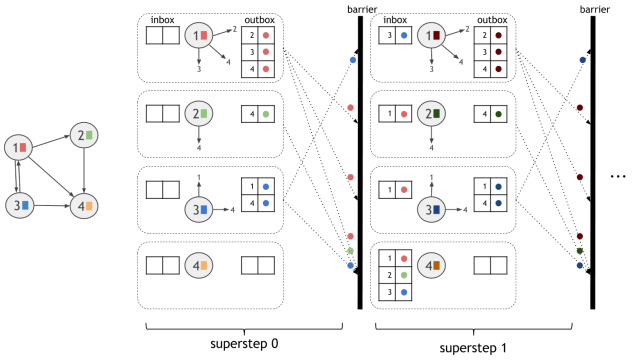

图 2: Pregel 计算过程

(1) 读取图数据并对图初始化；

(2) 当图被初始化完毕，执行一系列的超步（SuperStep）直到整个计算结束，这些 SuperStep 之间通过一些全局的同步点分隔;

(3) 输出计算结果。

在每个 SuperStep 中，顶点上的计算都是并行的，每个顶点执行相同的用于表达指定逻辑的用户自定义函数。每个顶点都可以需修改自身以及出边的状态，接收前一个 SuperStep 发送给它的消息，并将计算的结果或信息发送给其他顶点，这些信息会在下一个 SuperStep 中被目标顶点接收。边在这种计算模式中并不是核心对象，只用于表明消息传递的方向，没有相应的计算运行在其上。

3.1 Pregel 模型状态机

算法结束的时机取决于所有的顶点是否均已经达到 halt 状态。整个状态转换如下图所示:

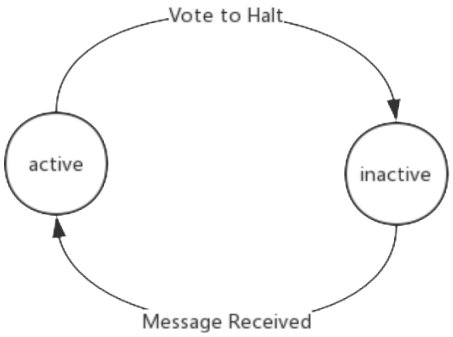

图 3: Pregel 状态转换图

图 3 中，首先在刚开始的时候，所有的顶点都处于 active 状态，所有的 active 顶点都会参与到 SuperStep 中的相应计算。顶点通过将其自身的 status 设置成 inactive 来表示它已不再 active，以此表明在下一次的 SuperStep 中，该顶点不再需要执行相应的计算。除非该顶点接收到其他顶点传送的消息，否则 Pregel 框架不会再接下来的 SuperStep 中执行该顶点的计算。如果顶点在接收到消息后进入 active 状态，那么在随后的计算中该顶点必须显式的 deactive。整个计算在所有顶点都达到 inactive 状态，并且没有消息传送时结束。

3.2 Pregel 模型示例

接下来展示一个以 Pregel 计算最大值的例子。假设图中存在 A/B/C/D 四个顶点，其中每个顶点的数据表示当前顶点的值。在计算的每个 SuperStep 中），每个顶点将接收其他顶点传过来的值（初始 SuperStep 除外），并判断当前顶点的值是否小于传递过来的值。若小于，更新当前顶点的值，并将该顶点状态设置为 active 状态，同时将当前顶点的最新值传递出去；若不小于，则什么都不做，并将当前顶点的状态设置为 inactive。直到所有的顶点状态均变成 inactive，计算结束。整个过程入下图所示：

以图 4 为例，我们详细介绍一下 Pregel 模型的执行过程。

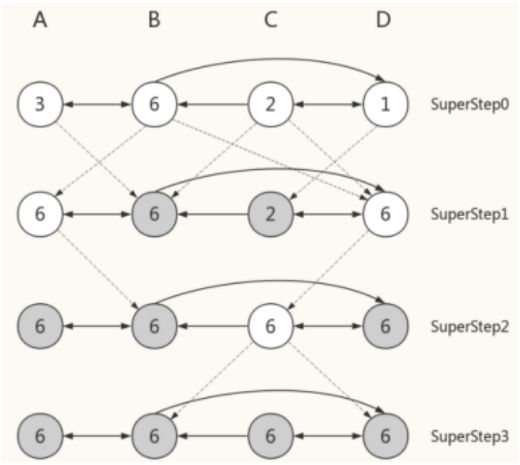

图 4: Pregel 模型示例图

- SuperStep0：初始 SuperStep，不接收信息，只负责将当前顶点的值传递出去，并设置所有顶点状态为 active。

- SuperStep1: 顶点 A 接收的信息为 6，将当前顶点的值设置为 6，状态设置为 active，并将 6 作为消息发送给顶点 B（顶点 B 下一轮迭代的时候会收到，当前轮次并不会收到）；顶点 B 接收的信息为 3 和 2，均小于当前顶点值 6，不更新当前顶点值，并将顶点的状态设置为 inactive；顶点 C 接收的信息为 1，小于当前顶点值 2，不更新当前顶点值，并将顶点的状态设置为 inactive；顶点 D 接收信息为 2 和 6，将当前顶点的值更新为 6，状态设置为 active，并将 6 作为消息发送给顶点 C；

- SuperStep2: 顶点 A 未接收到信息，且状态为 inactive，什么都不做；顶点 B 接收到顶点 A 上一个轮次发送的消息，由于当前顶点值为 6，不小于接收到的消息 6，因此不更新当前顶点值，并将顶点的状态设置为 inactive；顶点 C 接收到的消息为 6，当前顶点值为 2，将当前顶点的值更新为 6，状态设置为 active，并将 6 作为消息发送给顶点 B 和 D；顶点 D 未接收到信息，且状态为 inactive，什么都不做；

- SuperStep3: 顶点 A 未接收到信息，且状态为 inactive，什么都不做；顶点 B 接收到顶点 C 上一个轮次发送的消息，由于当前顶点值为 6，不小于接收到的消息 6，因此不更新当前顶点值，并将顶点的状态设置为 inactive；顶点 C 未接收到任何信息，因此将状态设置为 inactive；顶点 D 接收到顶点 C 上一个轮次发送的消息，由于当前顶点值为 6，不小于接收到的消息 6，因此不更新当前顶点值，并将顶点的状态设置为 inactive；

顶点的状态均为 inactive，迭代停止，输出结果。

3.3 Pregel 的一些应用

3.3.1 PageRank

在实现 PageRank 时，首先，我们实现 PageRankVertex 类，继承 Vertex. 顶点用一个 double 来存储当前的 PageRank 值。因为边不存储数据，边的类型是 void。我们设置 Graph 在超级步 0 初始化，每个顶点的值是 1/NumVertices()。

```
class PageRankVertex : public Vertex<Double, Void, Double>
    public  void Compute(MessageIterator msgs) {
        if (superstep() >= 1) {
            double sum = 0;
            for (Message msg : msgs) {
                sum += msg.getValue();
            }
            this.value = 0.15 / getNumOfVertices() + 0.85 * sum;
        }

     if (superstep() < 30) {
         int outSize = getOutEdgerIterator().getSize();
         sendMessageToAllNeighbors(this.value / n);
     } else {
         voteToHalt();
     }
}
```

图 5: PageRank 实现代码图

从超级步 1 开始，每个顶点把发送过来的值都加到 sum 里。并且设置临时的 PageRank 值为 0.15 / getNumOfVertices() + 0.85 * sum。到达超级步 30 后，不再发送信息，并且投票结束。实际运行中，PageRank 算法可能会一直计算到收敛。

3.3.2 最短路径

为了简单明了，我们主要关注单源最短路径，即从一个顶点出发，到所有其他顶点的最短路径。

```
class ShortestPathVertex extends Vertex<Integer, Integer, Integer> {
    public  void Compute(MessageIterator msgs) {
        int mindist = isSource(this.getVertexID()) ? 0 : Integer.MAX_VALUE;
        for (Message msg : msgs) {
            mindist = Math.min(mindist, msg.getValue());
        }
        if (mindist < this.getValue()) {
            this.setValue(mindist);
        }
        OutEdgeIterator iter = this.getOutEdgeIterator();
        for (OutEdge outEdge: iter) {
            sendMessageTo(outEdge.getTarget(), mindist + outEdge.getValue());
        }
        voteToHalt();
    }
}

class MinIntCombiner extends Combiner<Integer> {
    public void Combine(MessageIterator msgs) {
        int mindist = Integer.MAX_VALUE;
        for (Message msg : msgs) {
            mindist = Math.min(mindist, msg.getValue());
        }
        output("combined_source", mindist);
    }
}
```

图 6: 最短路径实现代码图

在这个程序里，我们把每个顶点的初始距离都是 $Integer.MAX\_VALUE$ 在每一超级步，每个顶点先接收从上游邻接顶点发过来的信息。用最小的值更新当前的值，然后更新它的下游邻接顶点。依次类推，此算法会在没有更新时结束。

本算法有一个 Combiner 来减少 Woker 之间传输的数据量。

4 结束语

本文详细的介绍了 Pregel 这个大数据的一个分布式的编程框架，其关注于为用户提供一个自然的图计算 API，并且隐藏分布式的一些细节，如信息传输和故障恢复。在概念上，它和 MapReduce 很相似，但是提供了更加自然的图 API, 对图的迭代计算更高效。Pregel 和 Sawzall、Pig Latin、Dryad 不同，因为 Pregel 隐藏了数据的分布细节。Pregel 还有一点不一样，因为它实现了一个有状态的模型，它用一个长驻进程来实现计算、交换信息、修改本地状态，而不是用一个数据流模型。Pregel 是为稀疏图设计的，主要是沿着边进行通信。尽管已经非常谨慎的支持高扇出、高扇入的信息流，但是当绝大部分顶点都和其他的大部分顶点进行通信时，性能会下降。一些类似的算法可以用 Combiners、Aggregators、或者修改图来编写 Pregel 友好的算法。当然类似的计算对于任何的高度分布式的计算都困难。

参考文献

[1] Thomas Anderson, Susan Owicki, James Saxe, and Charles Thacker, High-Speed Switch Scheduling for Local-Area Networks. ACM Trans. Comp. Syst. 11(4), 1993, 319-352.

[2] Luiz Barroso, Jerey Dean, and Urs Hoelzle, Web search for a planet: The Google Cluster Architecture. IEEE Micro 23(2), 2003, 22-28.

[3] 林子雨. 大数据技术原理与应用 [M]. 北京: 人民邮电出版社

[4] 蔡娇. 基于 Pregel 编程模型的图模式匹配方法 [D]. 云南大学,2018.

[5] 张骏雪. 面向大规模图数据处理的虚拟机管理系统研究与实现 [D]. 东南大学,2016.

[6] 李健, 张晓琳, 刘娇, 高鹭, 张换香. 基于 Pregel 的分布式保护节点影响力匿名算法 [J]. 计算机应用研究,2020,37(11):3428-3432.