《大数据技术原理与应用》

http://dblab.xmu.edu.cn/post/bigdata

温馨提示：编辑幻灯片母版，可以修改每页PPT的厦大校徽和底部文字

第九章 图计算

（PPT版本号：2016年1月29日版本）

林子雨

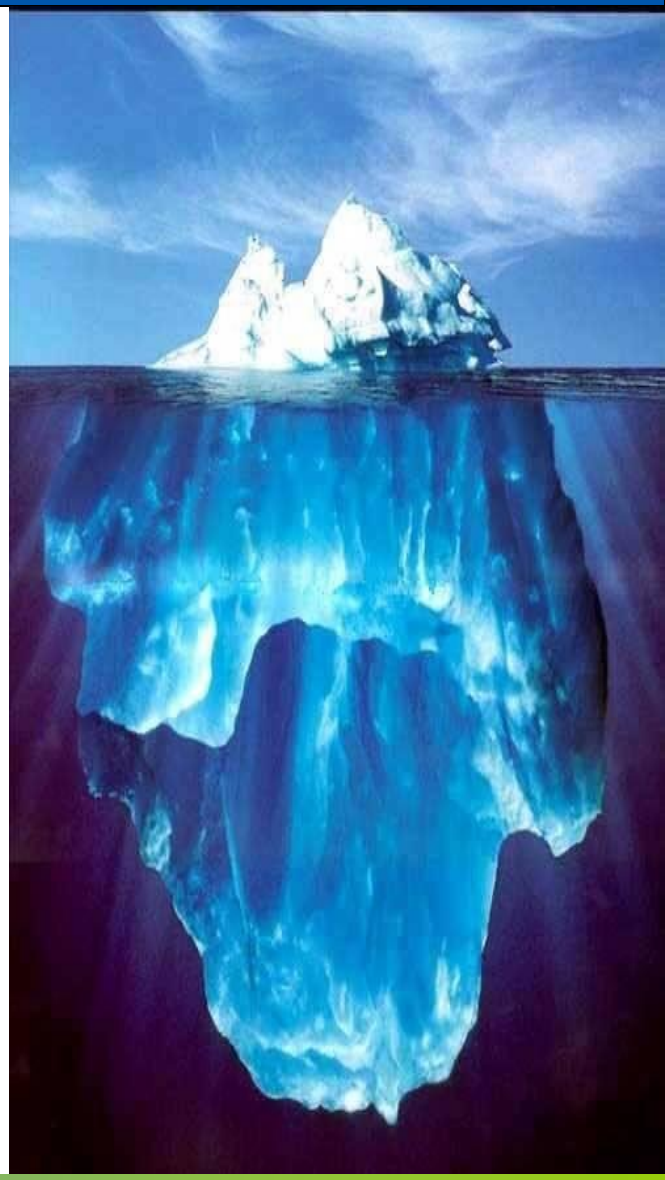
厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

主页：**http://www.cs.xmu.edu.cn/linziyu**

提纲

- **9.1 图计算简介**
- **9.2 Pregel简介**
- **9.3 Pregel图计算模型**
- **9.4 Pregel的C++ API**
- **9.5 Pregel的体系结构**
- **9.6 Pregel的应用实例**
- **9.7 Pregel和MapReduce实现PageRank算法的对比**



本PPT是如下教材的配套讲义：
21世纪高等教育计算机规划教材
《大数据技术原理与应用
——概念、存储、处理、分析与应用》
（2015年6月第1版）
厦门大学 林子雨 编著，人民邮电出版社
ISBN:978-7-115-39287-9

欢迎访问《大数据技术原理与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata

9.1 图计算简介

- 9.1.1 传统图计算解决方案的不足之处
- 9.1.2 图计算通用软件

9.1.1传统图计算解决方案的不足之处

很多传统的图计算算法都存在以下几个典型问题：
（1）常常表现出比较差的内存访问局部性；
（2）针对单个顶点的处理工作过少；
（3）计算过程中伴随着并行度的改变。

针对大型图（比如社交网络和网络图）的计算问题，可能的解决方案及其不足之处具体如下：

- 为特定的图应用定制相应的分布式实现：通用性不好
- 基于现有的分布式计算平台进行图计算：在性能和易用性方面往往无法达到最优
- 使用单机的图算法库：在可以解决的问题的规模方面具有很大的局限性
- 使用已有的并行图计算系统：对大规模分布式系统非常重要的一些方面（比如容错），无法提供较好的支持

9.1.2图计算通用软件

一次BSP计算过程包括一系列全局超步（所谓的超步就是计算中的一次迭代），每个超步主要包括三个组件：

•**局部计算**：每个参与的处理器都有自身的计算任务，它们只读取存储在本地内存中的值，不同处理器的计算任务都是异步并且独立的

•**通讯**：处理器群相互交换数据，交换的形式是，由一方发起推送(put)和获取(get)操作

•**栅栏同步**(Barrier Synchronization)：当一个处理器遇到“路障”（或栅栏），会等到其他所有处理器完成它们的计算步骤；每一次同步也是一个超步的完成和下一个超步的开始。图9-1是一个超步的垂直结构图

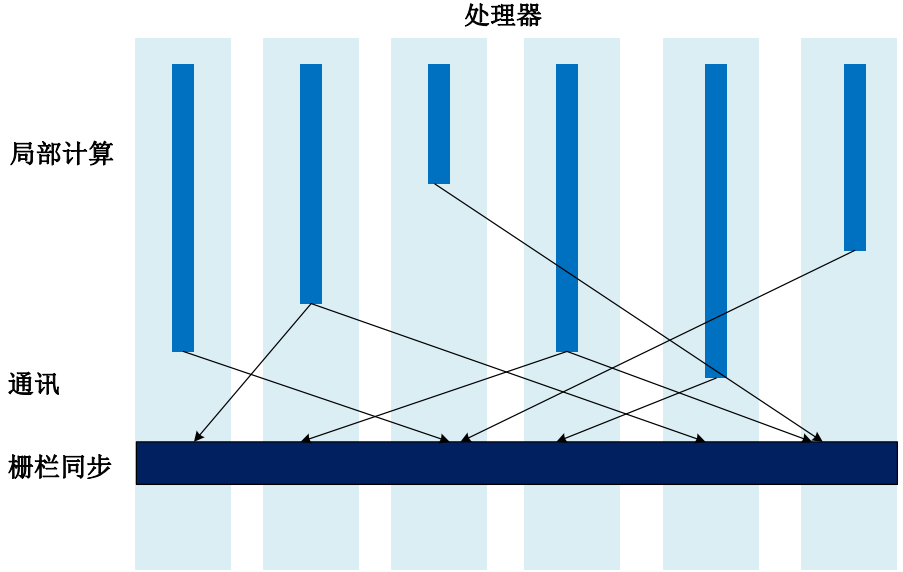


图9-1 一个超步的垂直结构图

9.2 Pregel简介

- Pregel是一种基于BSP模型实现的并行图处理系统
- 为了解决大型图的分布式计算问题，Pregel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
- Pregel作为分布式图计算的计算框架，主要用于图遍历、最短路径、PageRank计算等等

9.3 Pregel图计算模型

- 9.3.1 有向图和顶点
- 9.3.2 顶点之间的消息传递
- 9.3.3 Pregel的计算过程
- 9.3.4 实例

9.3.1有向图和顶点

•Pregel计算模型以有向图作为输入，有向图的每个顶点都有一个String类型的顶点ID，每个顶点都有一个可修改的用户自定义值与之关联，每条有向边都和其源顶点关联，并记录了其目标顶点ID，边上有一个可修改的用户自定义值与之关联

•在每个超步*S*中，图中的所有顶点都会并行执行相同的用户自定义函数。每个顶点可以接收前一个超步(*S*-1)中发送给它的消息，修改其自身及其出射边的状态，并发送消息给其他顶点，甚至是修改整个图的拓扑结构。需要指出的是，在这种计算模式中，边并不是核心对象，在边上面不会运行相应的计算，只有顶点才会执行用户自定义函数进行相应计算

9.3.2顶点之间的消息传递

采用消息传递模型主要基于以下两个原因：

（1）消息传递具有足够的表达能力，没有必要使用远程读取或共享内存的方式

（2）有助于提升系统整体性能

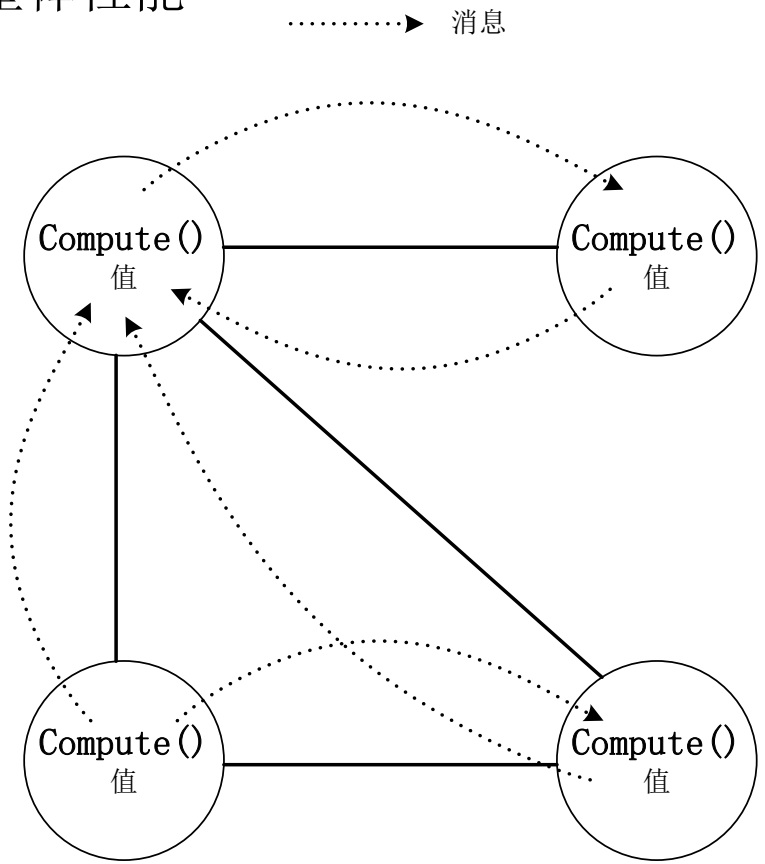


图9-2 纯消息传递模型图

9.3.3Pregel的计算过程

•Pregel的计算过程是由一系列被称为“超步”的迭代组成的。在每个超步中，每个顶点上面都会并行执行用户自定义的函数，该函数描述了一个顶点*V*在一个超步*S*中需要执行的操作。该函数可以读取前一个超步(*S*-1)中其他顶点发送给顶点*V*的消息，执行相应计算后，修改顶点*V*及其出射边的状态，然后沿着顶点*V*的出射边发送消息给其他顶点，而且，一个消息可能经过多条边的传递后被发送到任意已知ID的目标顶点上去。这些消息将会在下一个超步(*S*+1)中被目标顶点接收，然后像上述过程一样开始下一个超步(*S*+1)的迭代过程

•在Pregel计算过程中，一个算法什么时候可以结束，是由所有顶点的状态决定的，当图中所有的顶点都已经标识其自身达到“非活跃（inactive）”状态时，算法就可以停止运行

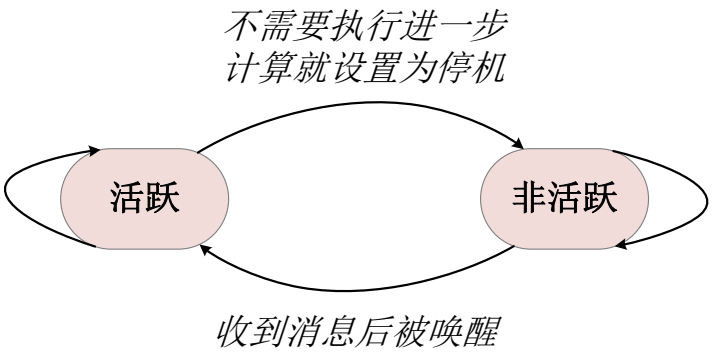


图9-3 一个简单的状态机图

9.3.4实例

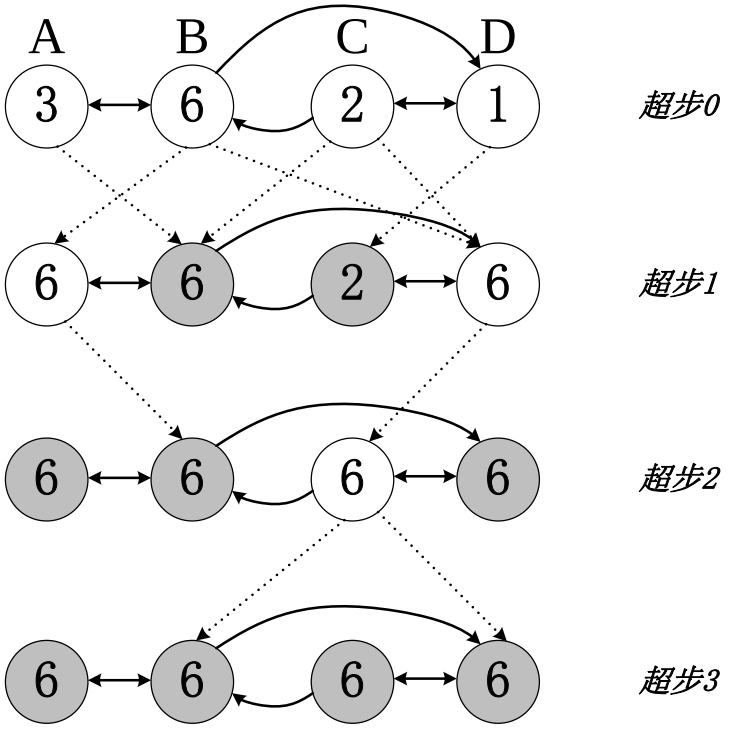


图9-4 一个求最大值的Pregel计算过程图

9.4 Pregel的C++ API

Pregel已经预先定义好一个基类——Vertex类：

```
template <typename VertexValue, typename EdgeValue, typename MessageValue>
class Vertex {
 public:
        virtual void Compute(MessageIterator* msgs) = 0;
        const string& vertex_id() const;
        int64 superstep() const;
        const VertexValue& GetValue();
        VertexValue* MutableValue();
        OutEdgeIterator GetOutEdgeIterator();
        void SendMessageTo(const string& dest_vertex,        const MessageValue& message);
        void VoteToHalt();
   };
```

•在Vetex类中，定义了三个值类型参数，分别表示顶点、边和消息。每一个顶点都有一个给定类型的值与之对应

•编写Pregel程序时，需要继承Vertex类，并且覆写Vertex类的虚函数Compute()

9.4 Pregel的C++ API

- 9.4.1 消息传递机制
- 9.4.2 Combiner
- 9.4.3 Aggregator
- 9.4.4 拓扑改变
- 9.4.5 输入和输出

9.4.1消息传递机制

- 顶点之间的通讯是借助于消息传递机制来实现的，每条消息都包含了消息值和需要到达的目标顶点ID。用户可以通过Vertex类的模板参数来设定消息值的数据类型
- 在一个超步S中，一个顶点可以发送任意数量的消息，这些消息将在下一个超步（S+1）中被其他顶点接收
- 一个顶点V通过与之关联的出射边向外发送消息，并且，消息要到达的目标顶点并不一定是与顶点V相邻的顶点，一个消息可以连续经过多条连通的边到达某个与顶点V不相邻的顶点U，U可以从接收的消息中获取到与其不相邻的顶点V的ID

9.4.2Combiner

- Pregel计算框架在消息发出去之前，Combiner可以将发往同一个顶点的多个整型值进行求和得到一个值，只需向外发送这个“求和结果”，从而实现了由多个消息合并成一个消息，大大减少了传输和缓存的开销
- 在默认情况下，Pregel计算框架并不会开启Combiner功能，因为，通常很难找到一种对所有顶点的Compute()函数都合适的Combiner
- 当用户打算开启Combiner功能时，可以继承Combiner类并覆写虚函数Combine()
- 此外，通常只对那些满足交换律和结合律的操作才可以去开启Combiner功能，因为，Pregel计算框架无法保证哪些消息会被合并，也无法保证消息传递给 Combine()的顺序和合并操作执行的顺序

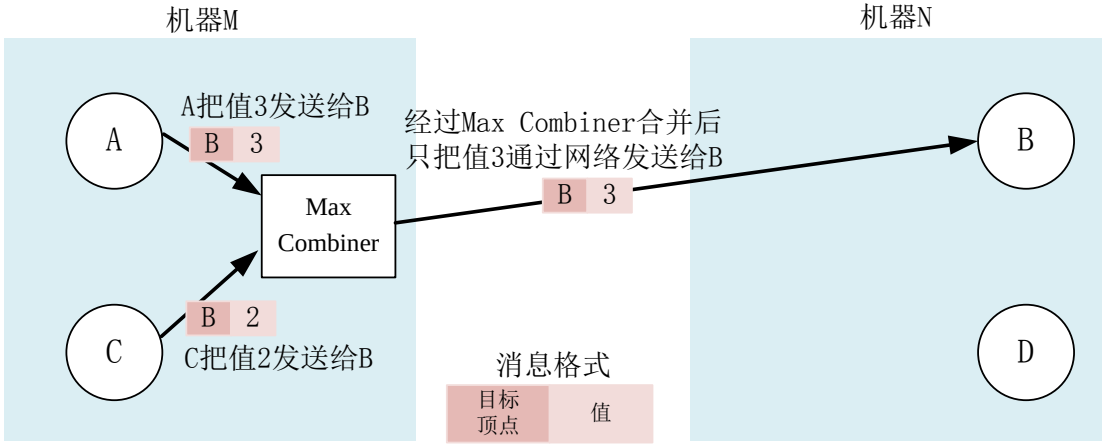


图9-5 Combiner应用的例子

9.4.3Aggregator

- Aggregator提供了一种全局通信、监控和数据查看的机制
- 在一个超步S中，每一个顶点都可以向一个Aggregator提供一个数据，Pregel计算框架会对这些值进行聚合操作产生一个值，在下一个超步（S+1）中，图中的所有顶点都可以看见这个值
- Aggregator的聚合功能，允许在整型和字符串类型上执行最大值、最小值、求和操作
- Pregel计算框架预定义了一个Aggregator类，编写程序时需要继承这个类，并定义在第一次接收到输入值后如何初始化，以及如何将接收到的多个值最后聚合成一个值
- 为了保证得到正确的结果，Aggregator操作也应该满足交换律和结合律

9.4.4拓扑改变

- Pregel计算框架允许用户在自定义函数Compute()中定义操作，修改图的拓扑结构，比如在图中增加（或删除）边或顶点
- Pregel采用两种机制来解决这类冲突：局部有序和Handler
- （1）局部有序：拓扑改变的请求是通过消息发送的，在执行一个超步时，所有的拓扑改变会在调用Compute()函数之前完成
- （2）Handler：对于“局部无序”机制无法解决的那些操作冲突，就需要借助于用户自定义的Handler来解决，包括解决由于多个顶点删除请求或多个边增加请求（或删除请求）而造成的冲突

9.4.5输入和输出

- 在Pregel计算框架中，图的保存格式多种多样，包括文本文件、关系数据库或键值数据库等
- 在Pregel中，“从输入文件生成得到图结构”和“执行图计算”这两个过程是分离的，从而不会限制输入文件的格式
- 对于输出，Pregel也采用了灵活的方式，可以以多种方式进行输出

9.5 Pregel的体系结构

- 9.5.1 Pregel的执行过程
- 9.5.2 容错性
- 9.5.3 Worker
- 9.5.4 Master
- 9.5.5 Aggregator

9.5.1Pregel的执行过程

- 在Pregel计算框架中，一个大型图会被划分成许多个分区，每个分区都包含了一部分顶点以及以其为起点的边
- 一个顶点应该被分配到哪个分区上，是由一个函数决定的，系统默认函数为hash(ID) mod *N*，其中，*N*为所有分区总数，ID是这个顶点的标识符；当然，用户也可以自己定义这个函数
- 这样，无论在哪台机器上，都可以简单根据顶点ID判断出该顶点属于哪个分区，即使该顶点可能已经不存在了

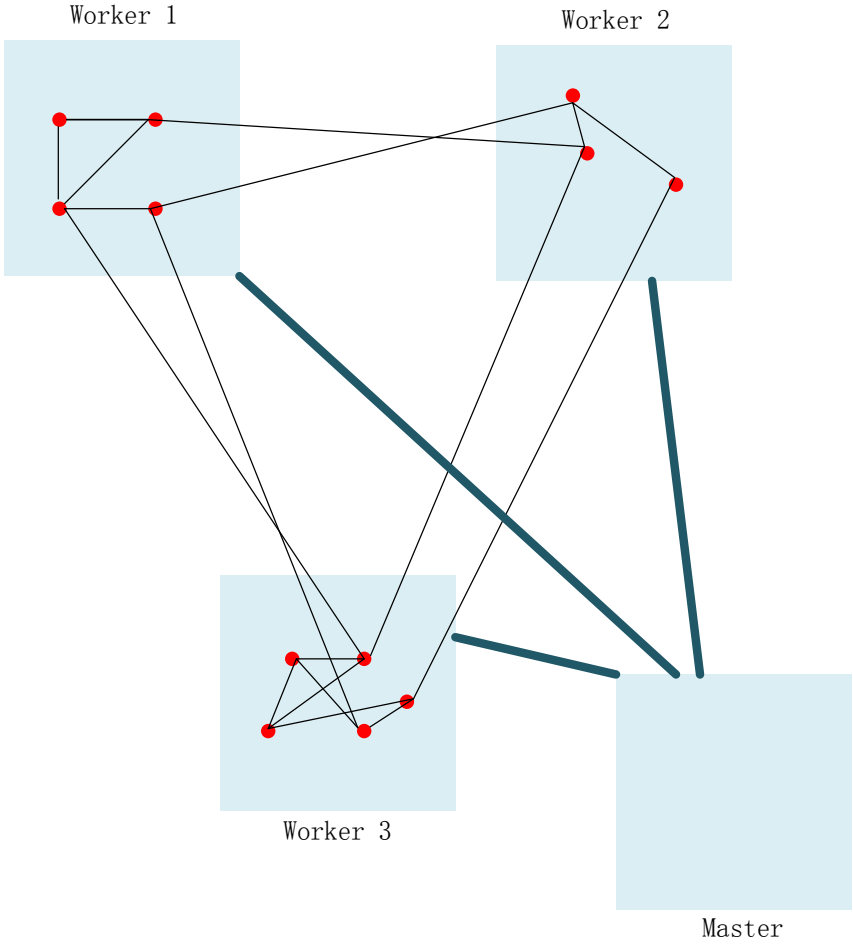


图9-6图的划分图

9.5.1Pregel的执行过程

在理想的情况下（不发生任何错误），一个Pregel用户程序的执行过程如下：
（1）选择集群中的多台机器执行图计算任务，每台机器上运行用户程序的一个副本，其中，有一台机器会被选为Master，其他机器作为Worker
（2）Master把一个图分成多个分区，并把分区分配到多个Worker
（3）Master会把用户输入划分成多个部分，通常是基于文件边界进行划分

（4）Master向每个Worker发送指令，Worker收到指令后，开始运行一个超步 。当完成以后，Worker会通知Master，并把自己在下一个超步还处于“活跃”状态的顶点的数量报告给Master。上述步骤会被不断重复，直到所有顶点都不再活跃并且系统中不会有任何消息在传输，这时，执行过程才会结束
（5）计算过程结束后，Master会给所有的Worker发送指令，通知每个Worker对自己的计算结果进行持久化存储

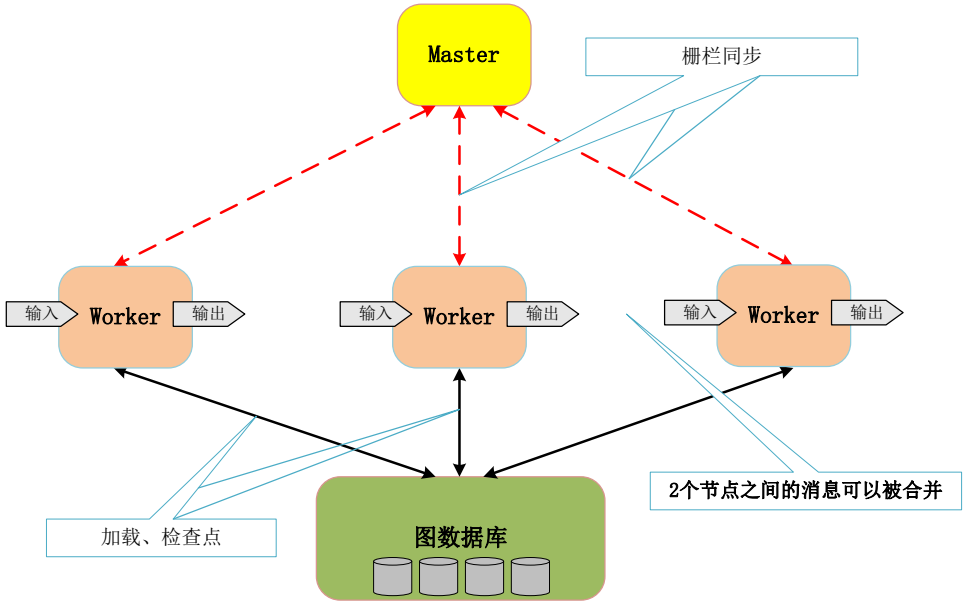


图9-7 Pregel的执行过程图

9.5.2容错性

- Pregel采用检查点机制来实现容错。在每个超步的开始，Master会通知所有的Worker把自己管辖的分区的状态（包括顶点值、边值以及接收到的消息），写入到持久化存储设备
- Master会周期性地向每个Worker发送ping消息，Worker收到ping消息后会给Master发送反馈消息。如果Master在指定时间间隔内没有收到某个Worker的反馈消息，就会把该Worker标记为“失效”。同样地，如果一个Worker在指定的时间间隔内没有收到来自Master的ping消息，该Worker也会停止工作
- 每个Worker上都保存了一个或多个分区的状态信息，当一个Worker发生故障时，它所负责维护的分区的当前状态信息就会丢失。Master监测到一个Worker发生故障“失效”后，会把失效Worker所分配到的分区，重新分配到其他处于正常工作状态的Worker集合上，然后，所有这些分区会从最近的某超步S开始时写出的检查点中，重新加载状态信息。很显然，这个超步S可能会比失效Worker上最后运行的超步S1要早好几个阶段，因此，为了恢复到最新的正确状态，需要重新执行从超步S到超步S1的所有操作

9.5.3Worker

在一个Worker中，它所管辖的分区的状态信息是保存在内存中的。
分区中的顶点的状态信息包括：
•顶点的当前值
•以该顶点为起点的出射边列表，每条出射边包含了目标顶点ID和边的值
•消息队列，包含了所有接收到的、发送给该顶点的消息
•标志位，用来标记顶点是否处于活跃状态

在每个超步中，Worker会对自己所管辖的分区中的每个顶点进行遍历，并调用顶点上的Compute()函数，在调用时，会把以下三个参数传递进去：
•该顶点的当前值
•一个接收到的消息的迭代器
•一个出射边的迭代器

9.5.4Master

•Master主要负责协调各个Worker执行任务，每个Worker会借助于名称服务系统定位到Master的位置，并向Master发送自己的注册信息，Master会为每个Worker分配一个唯一的ID
•Master维护着关于当前处于“有效”状态的所有Worker的各种信息，包括每个Worker的ID和地址信息，以及每个Worker被分配到的分区信息
•一个大规模图计算任务会被Master分解到多个Worker去执行，如果参与任务执行的多个Worker中的任意一个发生了故障失效，Master就会进入恢复模式
•Master在内部运行了一个HTTP服务器来显示图计算过程的各种信息，用户可以通过网页随时监控图计算执行过程各个细节

9.5.5Aggregator

- 每个用户自定义的Aggregator都会采用聚合函数对一个值集合进行聚合计算得到一个全局值
- 每个Worker都保存了一个Aggregator的实例集，其中的每个实例都是由类型名称和实例名称来标识的
- 在执行图计算过程的某个超步S中，每个Worker会利用一个Aggregator对当前本地分区中包含的所有顶点的值进行归约，得到一个本地的局部归约值
- 在超步S结束时，所有Worker会将所有包含局部归约值的Aggregator的值进行最后的汇总，得到全局值，然后提交给Master
- 在下一个超步S+1开始时，Master就会将Aggregator的全局值发送给每个Worker

9.6 Pregel的应用实例

- 9.6.1 单源最短路径
- 9.6.2 二分匹配

9.6.1单源最短路径

Pregel非常适合用来解决单源最短路径问题，实现代码如下：

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
      int mindist = IsSource(vertex_id()) ? 0 : INF;
      for (; !msgs->Done(); msgs->Next())
        mindist = min(mindist, msgs->Value());
      if (mindist < GetValue()) {
        *MutableValue() = mindist;
        OutEdgeIterator iter = GetOutEdgeIterator();
        for (; !iter.Done(); iter.Next())
            SendMessageTo(iter.Target(),
                    mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

9.6.2二分匹配

程序的执行过程是由四个阶段组成的多个循环组成的，当程序执行到超步*S*时，*S* mod 4就可以得到当前超步处于循环的哪个阶段。每个循环的四个阶段如下：

（1）阶段0：对于左集合中的任意顶点*V*，如果*V*还没有被匹配，就发送消息给它的每个邻居顶点请求匹配，然后，顶点*V*会调用VoteToHalt()进入“非活跃”状态。如果顶点*V*已经找到了匹配，或者*V*没有找到匹配但是没有出射边，那么，顶点*V*就不会发送消息。当顶点*V*没有发送消息，或者顶点*V*发送了消息但是所有的消息接收者都已经被匹配，那么，该顶点就不会再变为“活跃（active）”状态

（2）阶段1：对于右集合中的任意顶点*U*，如果它还没有被匹配，则会随机选择它接收到的消息中的其中一个，并向左集合中的消息发送者发送消息表示接受该匹配请求，然后给左集合中的其他请求者发送拒绝消息；然后，顶点*U*会调用VoteToHalt()进入“非活跃”状态

（3）阶段2：左集合中那些还未被匹配的顶点，会从它所收到的、右集合发送过来的接受请求中，选择其中一个给予确认，并发送一个确认消息。对于左集合中已经匹配的顶点而言，因为它们在阶段0不会向右集合发送任何匹配请求消息，因而也不会接收到任何来自右集合的匹配接受消息，因此，是不会执行阶段2的

（4）阶段3：右集合中还未被匹配的任意顶点*U*，会收到来自左集合的匹配确认消息，但是，每个未匹配的顶点*U*，最多会收到一个确认消息。然后，顶点*U*会调用VoteToHalt()进入“非活跃”状态，完成它自身的匹配工作

9.7 Pregel和MapReduce实现PageRank算法的对比

- 9.7.1 PageRank算法
- 9.7.2 PageRank算法在Pregel中的实现
- 9.7.3 PageRank算法在MapReduce中的实现
- 9.7.4 PageRank算法在Pregel和MapReduce中实现的比较

9.7.1 PageRank算法

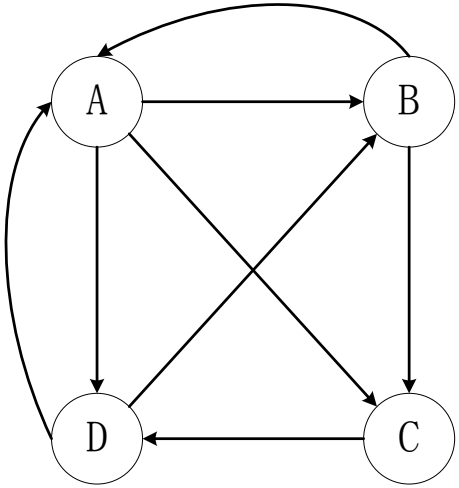
- PageRank是一个函数，它为网络中每个网页赋一个权值。通过该权值来判断该网页的重要性
- 该权值分配的方法并不是固定的，对PageRank算法的一些简单变形都会改变网页的相对PageRank值（PR值）
- PageRank作为谷歌的网页链接排名算法，基本公式如下：

$$PR = \beta \sum_{i=1}^{n} \frac{PR_i}{N_i} + (1-\beta)\frac{1}{N}$$

- 对于任意一个网页链接，其PR值为链入到该链接的源链接的PR值对该链接的贡献和，其中，N表示该网络中所有网页的数量，N_i为第i个源链接的链出度，PR_i表示第i个源链接的PR值

9.7.1 PageRank算法

- 网络链接之间的关系可以用一个连通图来表示，下图就是四个网页（*A*,*B*,*C*,*D*）互相链入链出组成的连通图，从中可以看出，网页*A*中包含指向网页*B*、*C*和*D*的外链，网页*B*和*D*是网页*A*的源链接



9.7.2 PageRank算法在Pregel中的实现

- 在Pregel计算模型中，图中的每个顶点会对应一个计算单元，每个计算单元包含三个成员变量：
  - 顶点值（Vertex value）：顶点对应的PR值
  - 出射边（Out edge）：只需要表示一条边，可以不取值
  - 消息（Message）：传递的消息，因为需要将本顶点对其它顶点的PR贡献值，传递给目标顶点
- 每个计算单元包含一个成员函数Compute()，该函数定义了顶点上的运算，包括该顶点的PR值计算，以及从该顶点发送消息到其链出顶点

9.7.2 PageRank算法在Pregel中的实现

```
class PageRankVertex: public Vertex<double, void, double> {
public:
        virtual void Compute(MessageIterator* msgs) {
                if (superstep() >= 1) {
                        double sum = 0;
                        for (;!msgs->Done(); msgs->Next())
                        sum += msgs->Value();
                        *MutableValue() =
                        0.15 / NumVertices() + 0.85 * sum;
            }
            if (superstep() < 30) {
                const int64 n = GetOutEdgeIterator().size();
                SendMessageToAllNeighbors(GetValue()/ n);
                } else {
                        VoteToHalt();
                }
            }
};
```

9.7.2 PageRank算法在Pregel中的实现

- PageRankVertex继承自Vertex类，顶点值类型是double，用来保存PageRank中间值，消息类型也是double，用来传输PageRank值，边的value类型是void，因为不需要存储任何信息
- 这里假设在第0个超步时，图中各顶点值被初始化为1/NumVertices()，其中，NumVertices()表示顶点数目
- 在前30个超步中，每个顶点都会沿着它的出射边，发送它的PageRank值除以出射边数目以后的结果值。从第1个超步开始，每个顶点会将到达的消息中的值加到sum值中，同时将它的PageRank值设为0.15/NumVertices()+0.85*sum
- 到了第30个超步后，就没有需要发送的消息了，同时所有的顶点停止计算，得到最终结果

9.7.3 PageRank算法在MapReduce中的实现

- MapReduce也是谷歌公司提出的一种计算模型，它是为全量计算而设计
- 采用MapReduce实现PageRank的计算过程包括三个阶段：
  - **第一阶段：解析网页**
  - **第二阶段：PageRank分配**
  - **第三阶段：收敛阶段**

9.7.3 PageRank算法在MapReduce中的实现

1. 阶段1：解析网页

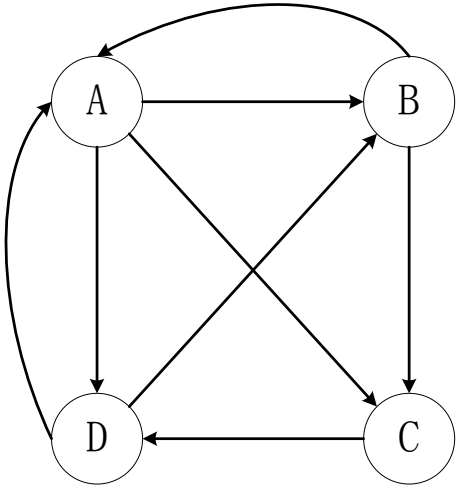
- 该阶段的任务就是分析一个页面的链接数并赋初值。
- 一个网页可以表示为由网址和内容构成的键值对< URL，page content>，作为Map任务的输入。阶段1的Map任务把<URL，page content>映射为<URL，<PR_{init}，url_list>>后进行输出，其中，PR_{init}是该URL页面对应的PageRank初始值，url_list包含了该URL页面中的外链所指向的所有URL。Reduce任务只是恒等函数，输入和输出相同。
- 对右图，每个网页的初始PageRank值为1/4。它在该阶段中:

Map任务的输入为：

<A_{URL}，$A_{content}$>
<B_{URL}，$B_{content}$>
<C_{URL}，$C_{content}$>
<D_{URL}，$D_{content}$>

Map任务的输出为：

<A_{URL}，<1/4，<B_{URL}，C_{URL}，D_{URL}>>>
<B_{URL}，<1/4，<A_{URL}，C_{URL}>>>
<C_{URL}，<1/4，D_{URL}>>
<D_{URL}，<1/4，<A_{URL}，B_{URL}>>>



9.7.3 PageRank算法在MapReduce中的实现

2. 阶段2：PageRank分配

- 该阶段的任务就是多次迭代计算页面的PageRank值。
- 在该阶段中，Map任务的输入是<URL，<cur_rank，url_list>>，其中，cur_rank是该URL页面对应的PageRank当前值，url_list包含了该URL页面中的外链所指向的所有URL。
- 对于url_list中的每个元素u，Map任务输出<u，<URL, cur_rank/|url_list|>>（其中，|url_list|表示外链的个数），并输出链接关系<URL，url_list>。
- 每个页面的PageRank当前值被平均分配给了它们的每个外链。Map任务的输出会作为下面Reduce任务的输入。对下图第一次迭代Map任务的输入输出如下：

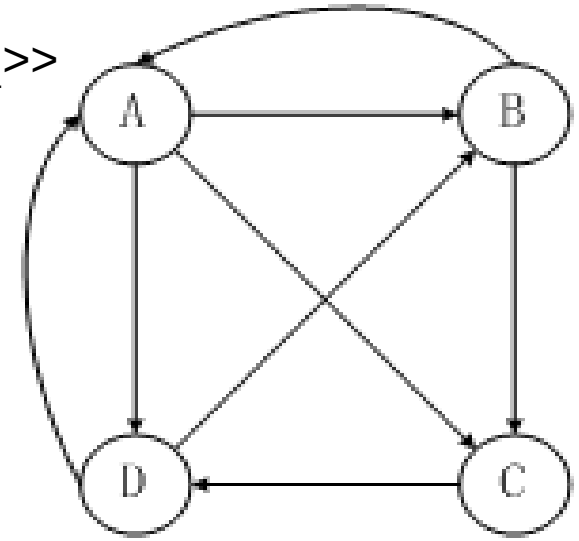
输入为：

<A_{URL}，$A_{content}$>
<B_{URL}，$B_{content}$>
<C_{URL}，$C_{content}$>
<D_{URL}，$D_{content}$>

输出为：

<B_{URL}，<A_{URL}，1/12>>
<C_{URL}，<A_{URL}，1/12>>
<D_{URL}，<A_{URL}，1/12>>

<A_{URL}，<B_{URL}，C_{URL}，D_{URL}>>
<A_{URL}，<B_{URL}，1/8>>
<C_{URL}，<B_{URL}，1/8>>
<B_{URL}，<A_{URL}，C_{URL}>>
<D_{URL}，<C_{URL}，1/4>>
<C_{URL}，D_{URL}>
<A_{URL}，<D_{URL}，1/8>>
<B_{URL}，<D_{URL}，1/8>>
<D_{URL}，<A_{URL}，B_{URL}>>



9.7.3 PageRank算法在MapReduce中的实现

2. 阶段2：PageRank分配（Reduce阶段）

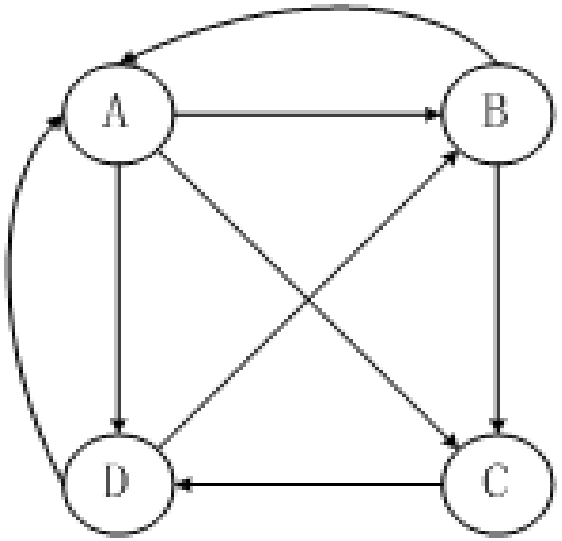
- 然后，在该阶段的Reduce阶段，Reduce任务会获得<URL，url_list>和<u，<URL，cur_rank/|url_list|>>，Reduce任务对于具有相同key值的value进行汇总，并把汇总结果乘以d，得到每个网页的新的PageRank值new_rank，然后输出<URL，<new_rank，url_list>>，作为下一次迭代过程的输入。

Reduce任务把第一次迭代后Map任务的输出作为自己的输入，经过处理后，阶段2的Reduce输出为：

$<A_{URL}，<0.2500，<B_{URL}，C_{URL}，D_{URL}>>>$
$<B_{URL}，<0.2147，<A_{URL}，C_{URL}>>>$
$<C_{URL}，<0.2147，D_{URL}>>$
$<D_{URL}，<0.3206，<A_{URL}，B_{URL}>>>$

经过本轮迭代，每个网页都计算得到了新的PageRank值。下次迭代阶段2的Reduce输出为：

$<A_{URL}，<0.2200，<B_{URL}，C_{URL}，D_{URL}>>>$
$<B_{URL}，<0.1996，<A_{URL}，C_{URL}>>>$
$<C_{URL}，<0.1996，D_{URL}>>$
$<D_{URL}，<0.3808，<A_{URL}，B_{URL}>>>$



9.7.3 PageRank算法在MapReduce中的实现

Mapper函数的伪码：

```
input <PageN, RankN> -> PageA,PageB,PageC
    ... // PageN外链指向PageA,PageB,PageC ...
begin
    Nn := the number of outlinks for PageN;
    for each outlink PageK
        output PageK -> <PageN, RankN/Nn>
    output PageN -> PageA, PageB, PageC ... //
     同时输出链接关系，用于迭代
end
/*************************
Mapper输出如下（已经排序，所以PageK的数据
    排在一起，最后一行则是链接关系对）：
PageK -> <PageN1, RankN1/Nn1>
PageK -> <PageN2, RankN2/Nn2>
...
PageK -> <PageAk, PageBk, PageCk>
```

Reducer函数的伪码：

```
input mapper's output
begin
    RankK :=(1-beta)/N;  //N为整个网络的网
     页总数
    for each inlink PageNi
        RankK += RankNi/Nni * beta
    //输出PageK及其新的PageRank值用于
     下次迭代
    output <PageK, RankK> -> <PageAk,
     PageBk, PageCk...>
end
```

该阶段是一个多次迭代过程，迭代多次后，当PageRank值趋于稳定时，就得出了较为精确的PageRank值。

9.7.3 PageRank算法在MapReduce中的实现

3. 阶段3：收敛阶段

- 该阶段的任务就是由一个非并行组件决定是否达到收敛，如果达到收敛，就写出PageRank生成的列表。否则，回退到PageRank分配阶段的输出，作为新一轮迭代的输入，开始新一轮PageRank分配阶段的迭代
- 一般判断是否收敛的条件是所有网页的PageRank值不再变化，或者运行30次以后我们就认为已经收敛了

9.7.4 PageRank算法在Pregel和MapReduce中实现的比较

- PageRank算法在Pregel和MapReduce中实现方式的区别主要表现在以下几个方面：

➢ （1）Pregel将PageRank处理对象看成是连通图，而MapReduce则将其看成是键值对

➢ （2）Pregel将计算细化到顶点，同时在顶点内控制循环迭代次数，而MapReduce则将计算批量化处理，按任务进行循环迭代控制

➢ （3）图算法如果用MapReduce实现，需要一系列的MapReduce的调用。从一个阶段到下一个阶段，它需要传递整个图的状态，会产生大量不必要的序列化和反序列化开销。而Pregel使用超步简化了这个过程

本章小结

- 本章内容介绍了图计算框架Pregel的相关知识。传统的图计算解决方案无法解决大型的图计算问题，包括Pregel在内的各种图计算框架脱颖而出。
- Pregel并没有采用远程数据读取或者共享内存的方式，而是采用了纯消息传递模型，来实现不同顶点之间的信息交换。Pregel的计算过程是由一系列被称为“超步”的迭代组成的，每次迭代对应了BSP模型中的一个超步。
- Pregel已经预先定义好一个基类——Vertex类，编写Pregel程序时，需要继承Vertex类，并且覆写Vertex类的虚函数Compute()。在Pregel执行计算过程时，在每个超步中都会并行调用每个顶点上定义的Compute()函数。
- Pregel是为执行大规模图计算而设计的，通常运行在由多台廉价服务器构成的集群上。一个图计算任务会被分解到多台机器上同时执行，Pregel采用检查点机制来实现容错。
- Pregel作为分布式图计算的计算框架，主要用于图遍历、最短路径、PageRank计算等等。
- 本章最后通过对PageRank算法在MapReduce和Pregel上执行方式的不同进行比较，说明了Pregel解决图计算问题的优势。

附录：主讲教师

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），现为厦门大学计算机科学系助理教授（讲师），曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度厦门大学奖教金获得者。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，编著出版中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》并成为畅销书籍，编著并免费网络发布40余万字中国高校第一本闪存数据库研究专著《闪存数据库概念与技术》；主讲厦门大学计算机系本科生课程《数据库系统原理》和研究生课程《分布式数据库》《大数据技术基础》。具有丰富的政府和企业信息化培训经验，曾先后给中国移动通信集团公司、福州马尾区政府、福建省物联网科学研究院、石狮市物流协会、厦门市物流协会、福建龙岩卷烟厂等多家单位和企业开展信息化培训，累计培训人数达2000人以上。

附录：大数据学习教材推荐

扫一扫访问教材官网

《大数据技术原理与应用——概念、存储、处理、分析与应用》，由厦门大学计算机科学系林子雨博士编著，是中国高校第一本系统介绍大数据知识的专业教材。

全书共有13章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据 库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、流计算、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase和MapReduce等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata

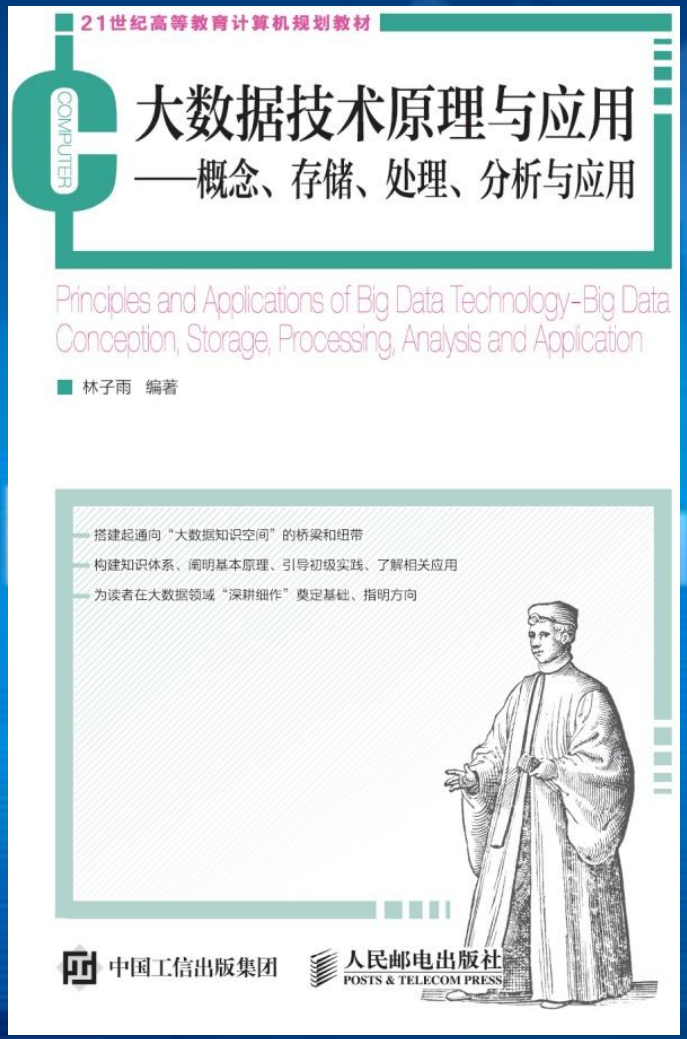
附录：中国高校大数据课程公共服务平台

中国高校大数据课程
公 共 服 务 平 台

http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

扫一扫访问平台主页

扫一扫观看3分钟FLASH动画宣传片



Department of Computer Science, Xiamen University, 2016