

Distributed Optimization

Petuum Spark Hadoop

Distributed Optimization

- 一、MapReduce and Hadoop
- 二、GraphX and GraphLab and Pregel (Graph Computing Frameworks)
 - (1) 单机 GraphLab (2010)
 - (2) 多机 GraphLab (2012)
 - (3) BSP and Pregel (SIGMOD2010)
 - Bulk synchronous parallel(BSP)
 - Pregel
 - (4) GraphX (2012)
- 三、Petuum
 - (1) Background:
 - (2) Parameter Server
 - (3) Stale Synchronous Parallel (SSP)
 - (4) SSPTable
 - (5) Theoretical Analysis

Reference

一、MapReduce and Hadoop

Hadoop 是一个分布式的计算框架。Hadoop的框架最核心的设计就是：HDFS (Hadoop Distributed File System) 和MapReduce。HDFS为海量的数据提供了存储，则MapReduce为海量的数据提供了计算。这里就不讲HDFS了，主要focus on计算的内容。

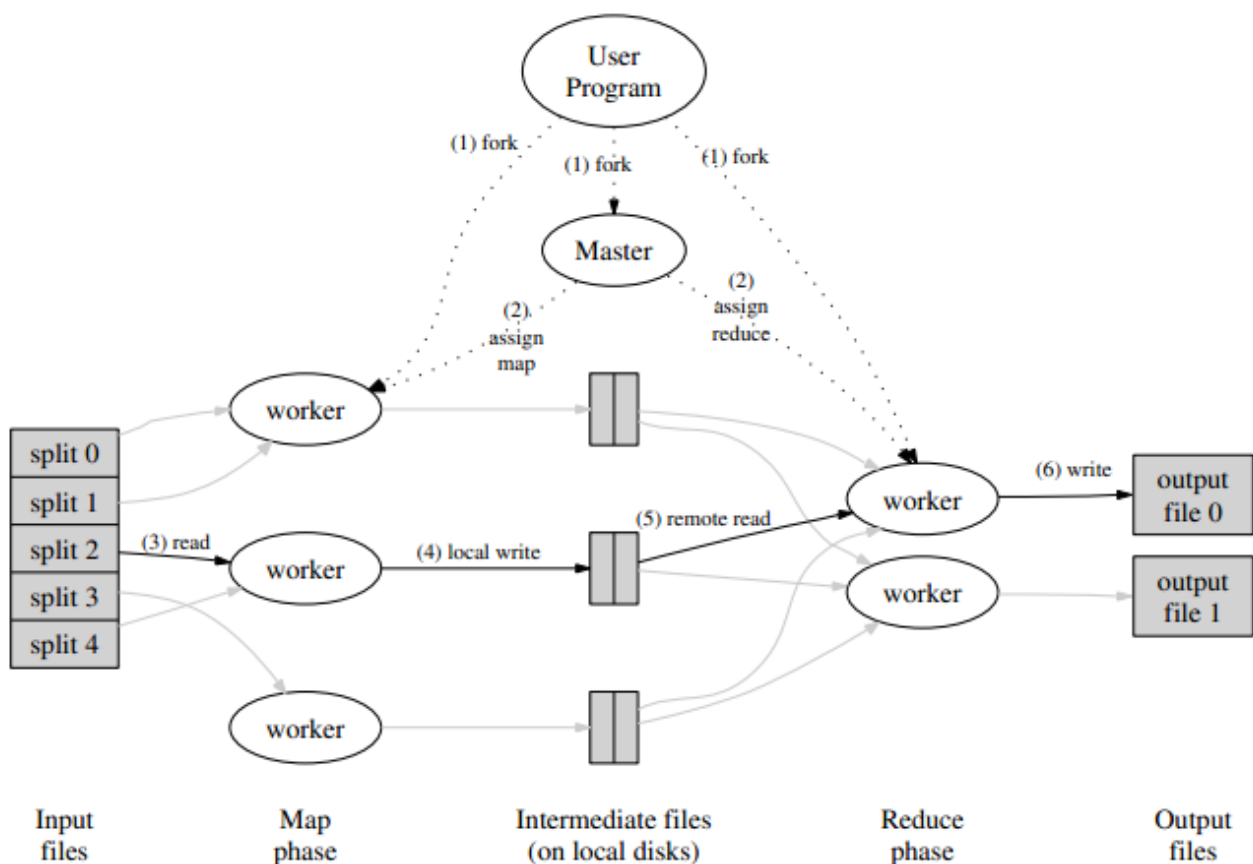


Figure 1: Execution overview

计算过程是 Input -> Split -> Map -> Shuffle -> Reduce -> Output 。传递的数据是 key/value对。其中对不同任务，要设计不同的Map 和 Reduce 函数。

举一个简单的例子，有不同的商品，商品有很多商店卖，价格不同。Key就是商品，Value就是价格。要找最便宜的。把数据分成几块（Split）传递给worker，worker在每一块分别找到每一个商品最便宜的（Map），收集所有Worker的结果再把相同的Key的合并起来（Shuffle），再从其中找出每个商品的最便宜的价格（Reduce）

在工程实现上很很多很好的技巧，比如容错性，节点分配控制系统。（Master）

二、GraphX and GraphLab and Pregel (Graph Computing Frameworks)

GraphLab是一个面向大规模机器学习/图计算的分布式内存计算框架，由CMU在2009年开始的一个C++项目。

Graph计算的背景

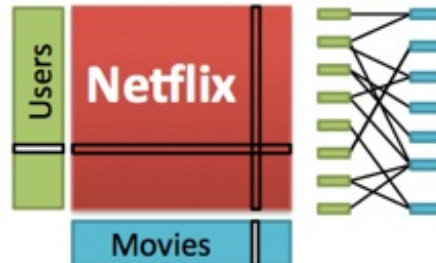
sense
learn
act

Graphs are Everywhere

Social Network



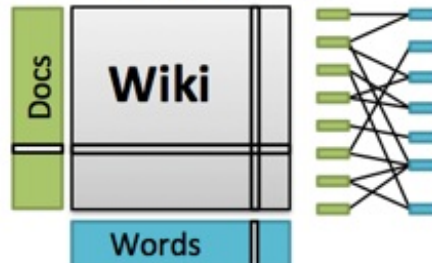
Collaborative Filtering



Probabilistic Analysis



Text Analysis



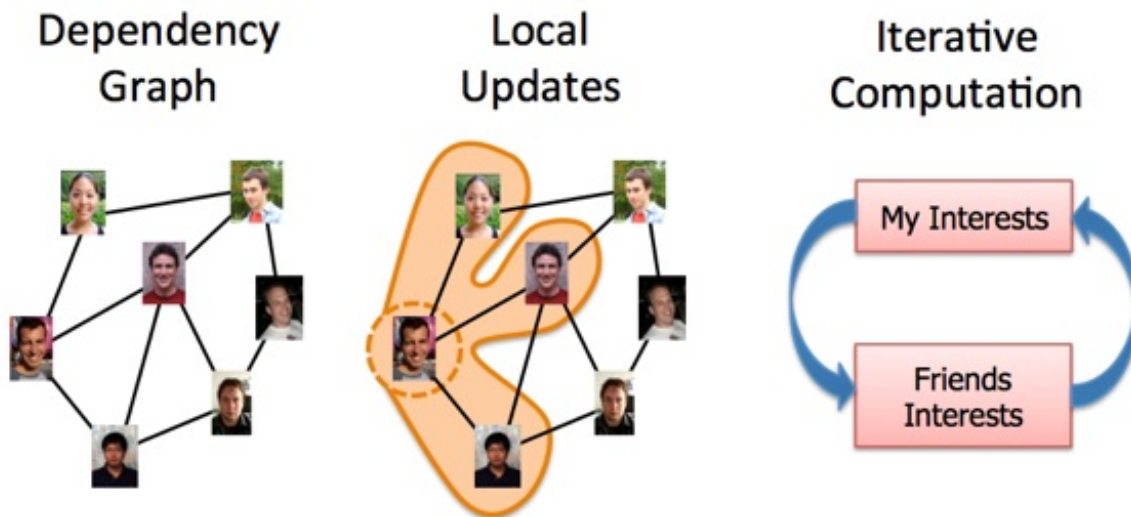
13

Graph可以刻画的范围是很广的，用户和商品之间的关系是一个典型的二部图，pagerank的random walk也是一张图

Graph database(Neo4j, Titan, flockdb)是用于图数据的存储检索，而涉及到复杂的Graph Processing，就适合用graphlab做。

Graph计算的特点

Properties of Computation on Graphs



1. **Dependency Graph** : MapReduce对于大的data并行任务 (Feature Extraction/Cross Validation) 是适用的, 但data并行系统很难刻画data之间的依赖关系, 而这一点在机器学习 (Gibbs Sampling , 变分法 , PageRank , CoEM , Collaborative Filtering等) 中非常重要。
2. **Local Updates** : 在Graph并行系统中, 一个结点的值只受相邻结点的影响, 因此可以根据局部值就可以做更新。而在data并行系统中是没有Local Updates的概念的, local信息可以加快计算, 不同local之间可以做并行。
3. **Iterative Computation** : 和普通Map-reduce任务不同, 图计算天然涉及到迭代计算。更新结点a的时候, 对其所有邻居(包括邻居结点b)map, 再reduce所有邻居的结果, 用得到的值来update结点a的值。然后就可以用结点a的最新值去更新他的结点b了。

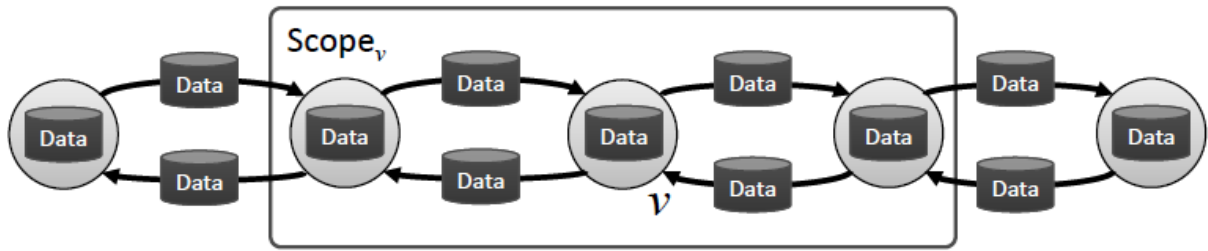
有两代产品：1.**GraphLab: A New Parallel Framework for Machine Learning** (2010) 2.**Distributed GraphLab: A Framework for Machine Learning in the Cloud** (2012)

(1) 单机 GraphLab (2010)

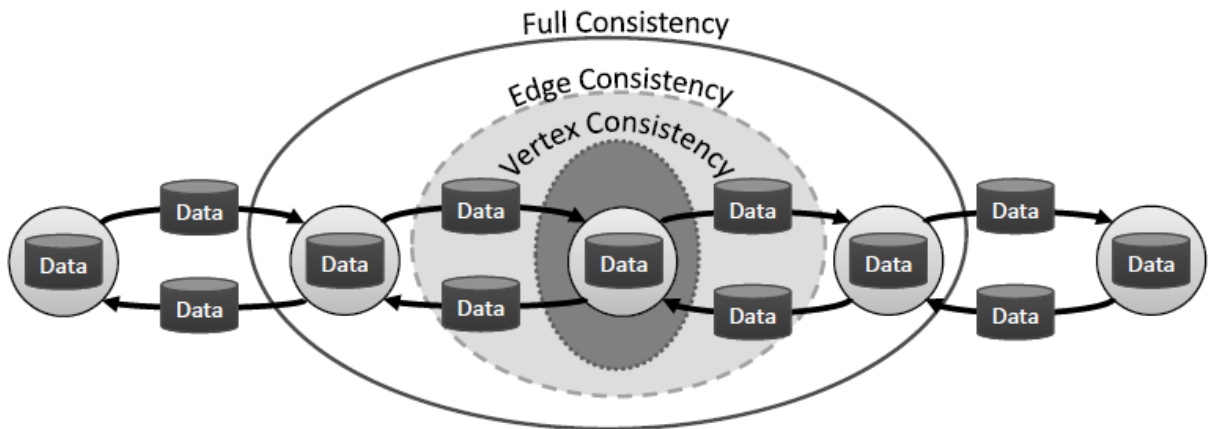
1. Data Model: **data graph** and **shared data table**.(T[Key] -> Value, between keys and arbitrary blocks of data.)

2.a Update Function

$$D_{S_v} \leftarrow f(D_{S_v}, \mathbf{T}).$$



(a) Scope



(b) Consistency Models

Figure 1: **(a)** The scope, S_v , of vertex v consists of all the data at the vertex v , its inbound and outbound edges, and its neighboring vertices. The update function f when applied to the vertex v can read and modify any data within S_v . **(b)**. We illustrate the 3 data consistency models by drawing their exclusion sets as a ring where no two update functions may be executed simultaneously if their exclusions sets (rings) overlap.

2.b Sync Mechanism

Algorithm 1: Sync Algorithm on k

```
 $t \leftarrow r_k^{(0)}$   
foreach  $v \in V$  do  
   $t \leftarrow \text{Fold}_k(D_v, t)$   
 $\mathbf{T}[k] \leftarrow \text{Apply}_k(t)$ 
```

$$r_k^{(i+1)} \leftarrow \text{Fold}_k \left(D_v, r_k^{(i)} \right) \quad (3.1)$$

$$r_k^l \leftarrow \text{Merge}_k \left(r_k^i, r_k^j \right) \quad (3.2)$$

$$\mathbf{T}[k] \leftarrow \text{Apply}_k(r_k^{(|V|)}) \quad (3.3)$$

3 Data Consistency

比如在Update Function从中访问到同一数据，就要考虑数据的一致性的问题。有三种一致性考虑，GraphLab的调度保证不会在一致性范围内发生同时作用Update Function的情况。

4 Scheduling 调度

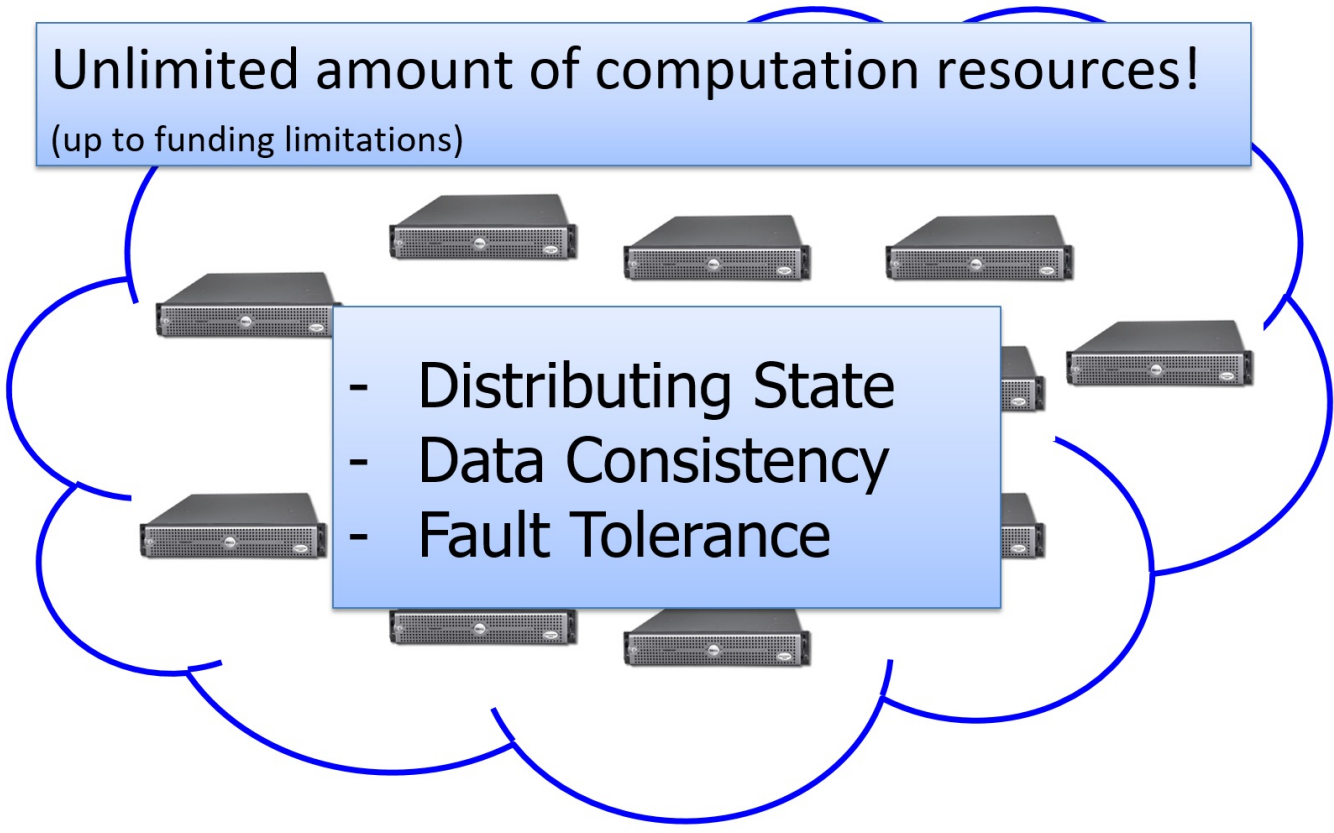
5 Termination Assessment

(2) 多机 GraphLab (2012)

Distributed Cloud

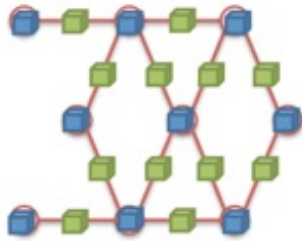
Unlimited amount of computation resources!

(up to funding limitations)

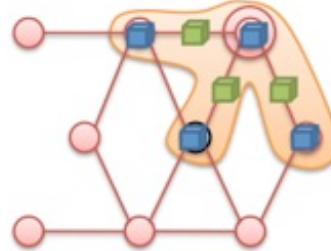
- 
- The diagram illustrates a distributed cloud system. At the top, a blue box contains the text 'Unlimited amount of computation resources!' followed by '(up to funding limitations)'. Below this, a cloud shape is formed by a blue line. Inside the cloud, there are ten server icons arranged in a circular pattern. In the center of the cloud is a light blue box containing a list of three items: '- Distributing State', '- Data Consistency', and '- Fault Tolerance'.
- Distributing State
 - Data Consistency
 - Fault Tolerance

The GraphLab Framework

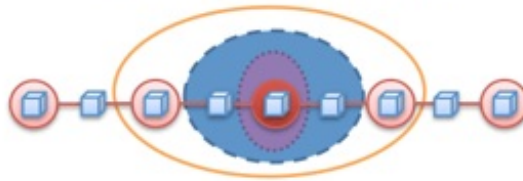
Graph Based
Data Representation



Update Functions
User Computation



Consistency Model



24

Graph Based Data Representation : GraphLab将图切成若干子图分布式存储，其中ghost vertex是子图之间的边界点，其上存储了邻接结构以及remote数据的副本，子图之间也是有通信的，因此disk数据共享做备份很困难。

Update Functions : 采用的是Asynchronously Dynamic Update，这种动态计算的主要思想是根据vertex的priority更新，每台机器上都有一个优先队列，每次迭代中如果当前vertex变化量不大的话就不再将该点的scope（一步可达的点）入队了，ghost顶点不需要入队。改进空间：可以用排队论优化。

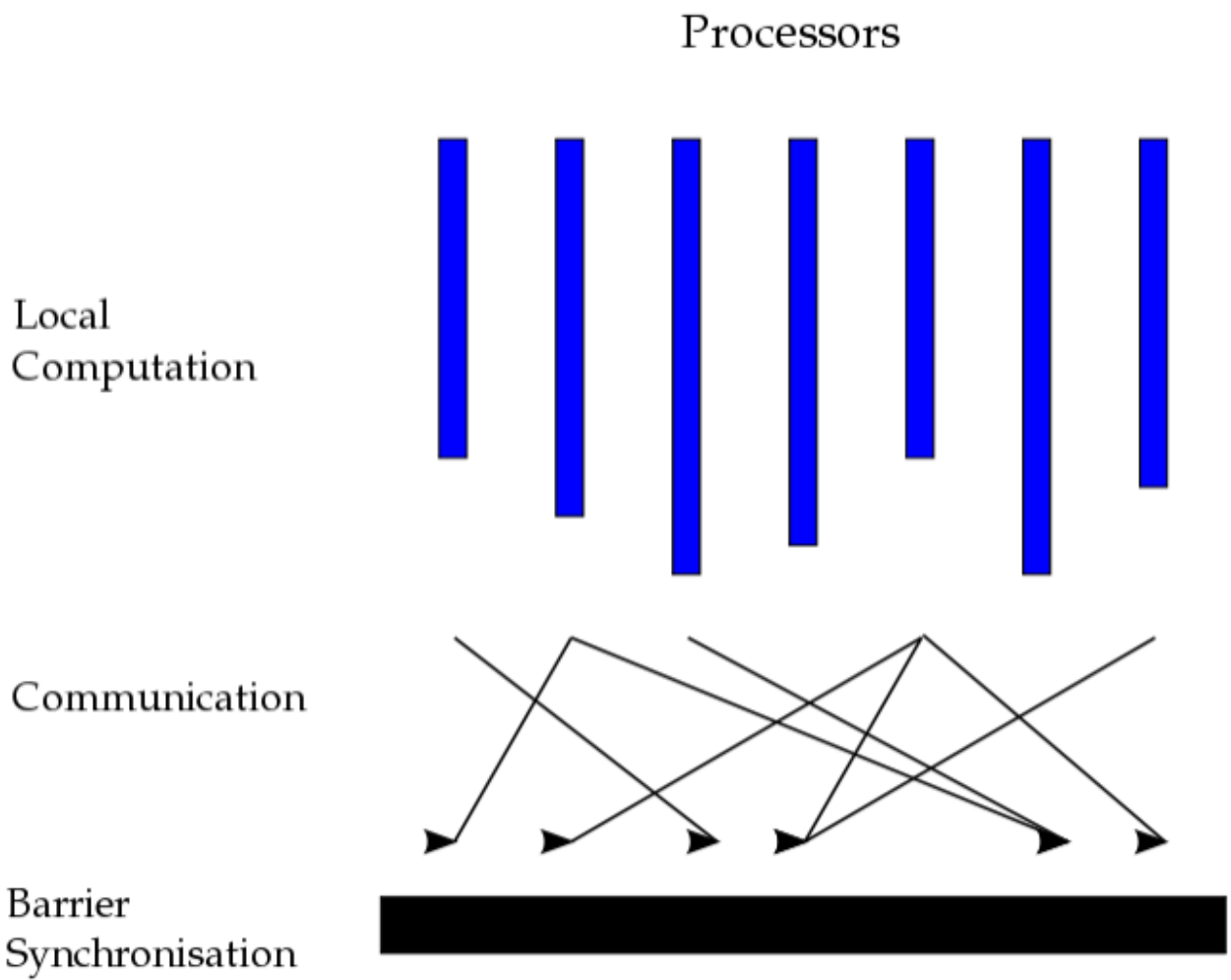
Data consistency : 需要新考虑的是怎么在机器之间保证Distributed Consistency 有两种解决办法

- 1) 图着色(算法复杂，并且可能有些颜色的partition比较小影响效率)
- 2) Distributed Locking with pipelining(高效，Latency Hiding)

Fault tolerance : GraphLab在这方面做的还不是很好，主要是Chandy-Lamport的asynchronous snapshotting algorithm。

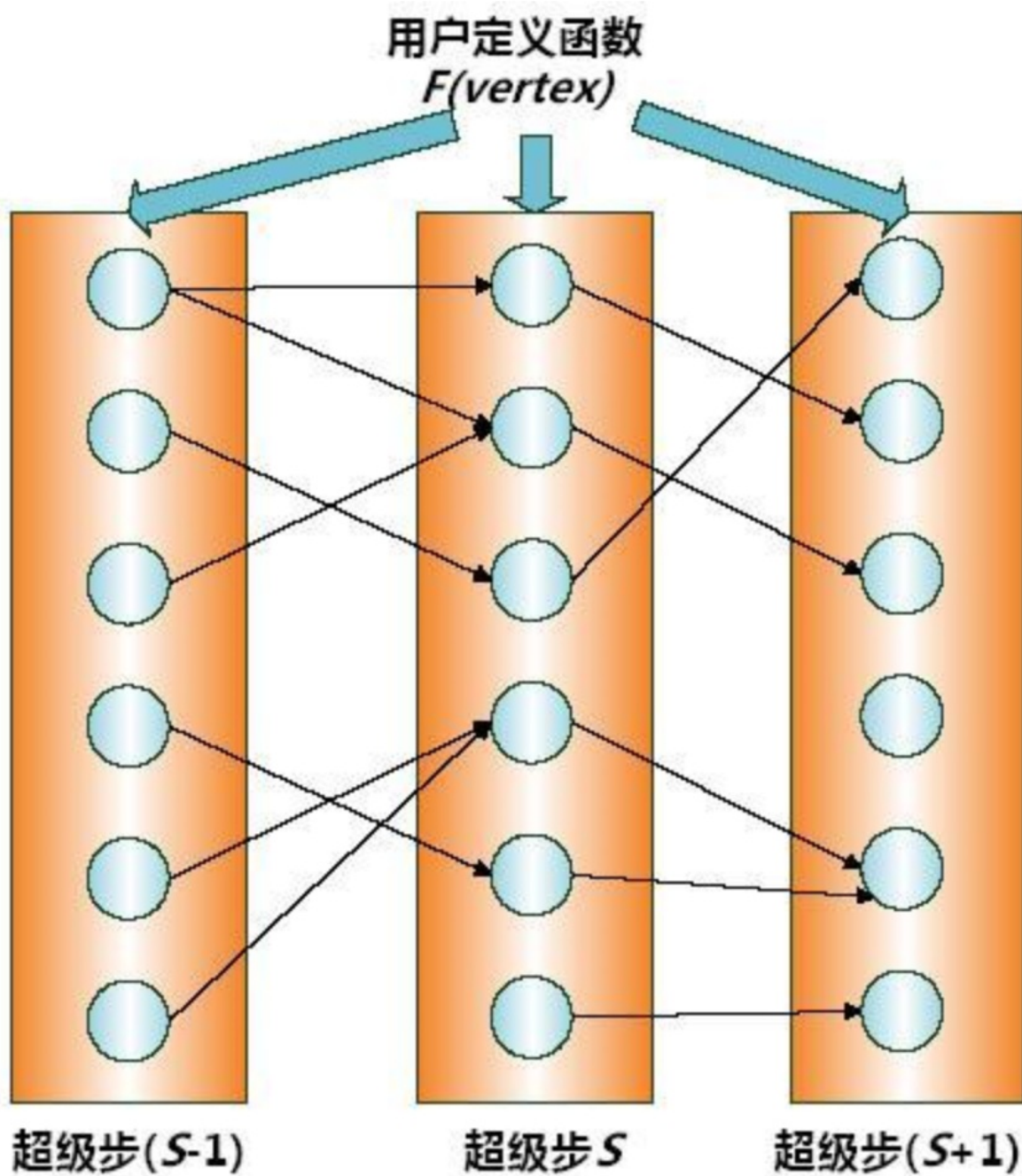
(3) BSP and Pregel (SIGMOD2010)

Bulk synchronous parallel(BSP)



Pregel

一个典型的Pregel计算过程如下：读取输入初始化该图，当图被初始化好后，运行一系列的超级步直到整个计算结束，这些超级步之间通过一些全局的同步点分隔，输出结果结束计算。



在每个超级步中，顶点的计算都是并行的，每个顶点执行相同的用于表达给定算法逻辑的用户自定义函数。每个顶点可以修改其自身及其出边的状态，接收前一个超级步(S-1)中发送给它的消息，并发送消息给其他顶点(这些消息将会在下一个超级步中被接收)，甚至是修改整个图的拓扑结构。边，在这种计算模式中并不是核心对象，没有相应的计算运行在其上。

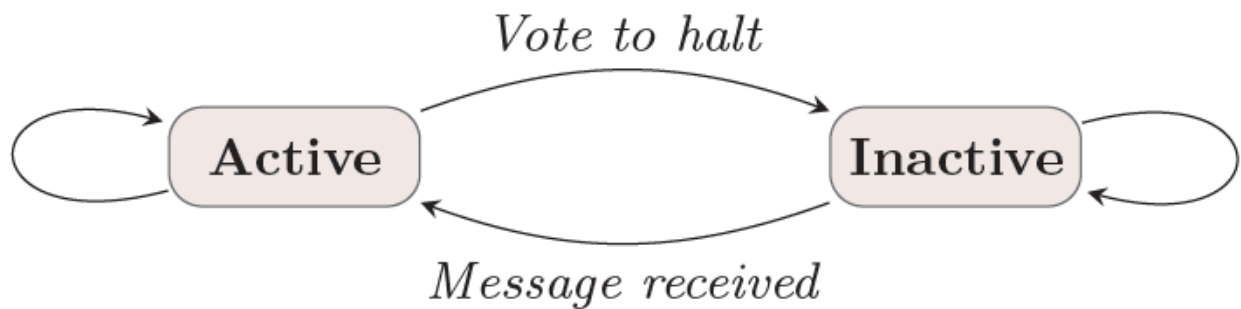


Figure 1: Vertex State Machine

每个节点有两种状态：活跃与不活跃，刚开始计算的时候，每个节点都处于活跃状态，随着计算的进行，某些节点完成计算任务转为不活跃状态，如果处于不活跃状态的节点接收到新的消息，则再次转为活跃，如果图中所有的节点都处于不活跃状态，则计算任务完成，Pregel输出计算结果。

1. 用户程序的多个copy开始在集群中的机器上执行。其中有一个copy将会作为master，其他的作为worker，master负责协调worker间的工作。
2. Master决定对这个图需要多少个partition，并分配一个或多个partitions到worker所在的机器上。每一个worker负责维护在其之上的图的那一部分的状态(顶点及边的增删)，对该部分中的顶点执行Compute()函数，并管理通讯。每一个worker都知道该图的计算在所有worker中的分配情况。
3. Master进程为每个worker分配用户输入中的一部分初始化。当所有的输入都被load完成后，所有的顶点将被标记为active状态，
4. Master给每个worker发指令，让其运行一个超级步，worker轮询在其之上的顶点，会为每个partition启动一个线程。调用每个active顶点的Compute()函数，传递给它从上一次超级步发送来的消息。当一个worker完成了其所有的工作后，会通知master，并告知当前该worker上在下一个超级步中将还有多少active节点。

不断重复该步骤，只要有顶点还处在active状态，或者还有消息在传输。

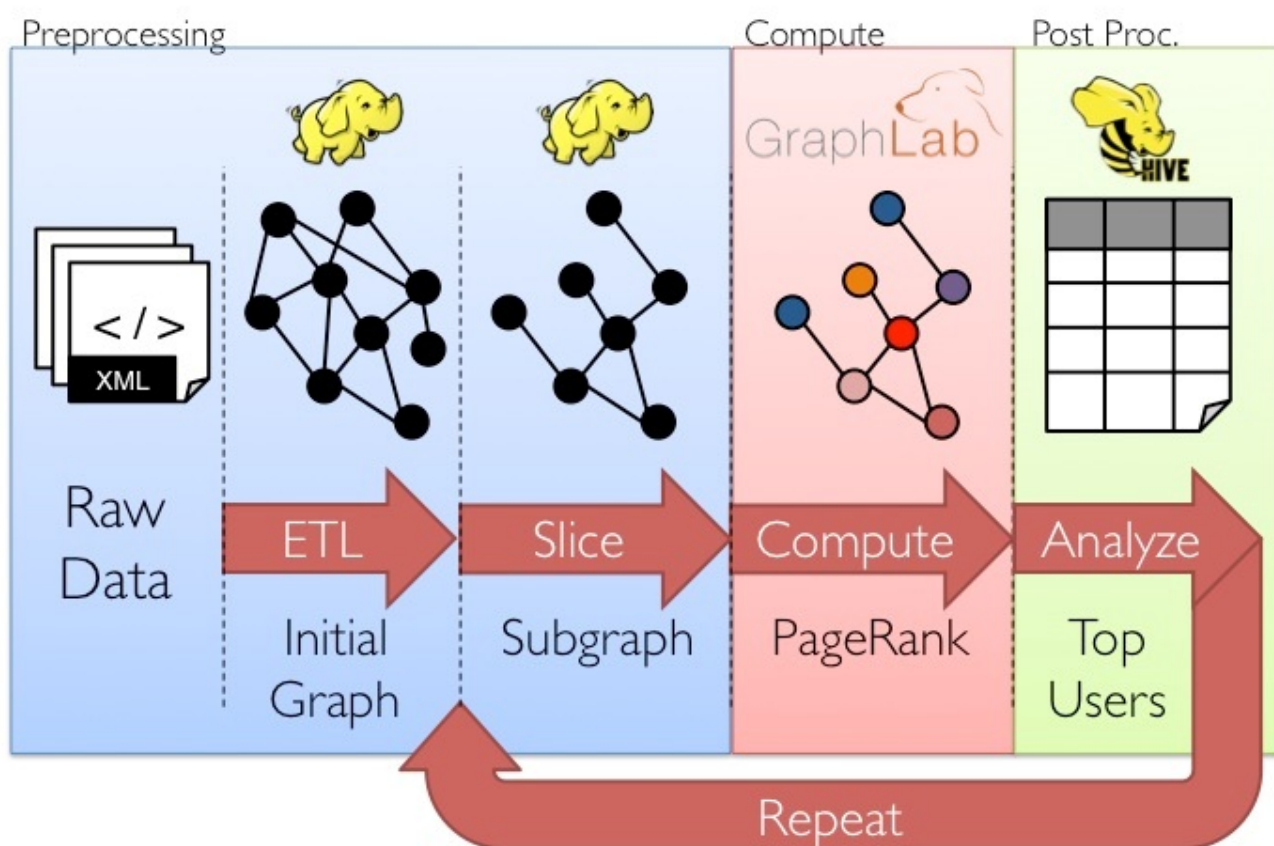
5. 计算结束后，master会给所有的worker发指令，让它保存它那一部分的计算结果。

(4) GraphX (2012)

GraphX是Spark中用于图(e.g., Web-Graphs and Social Networks)和图并行计算(e.g.,

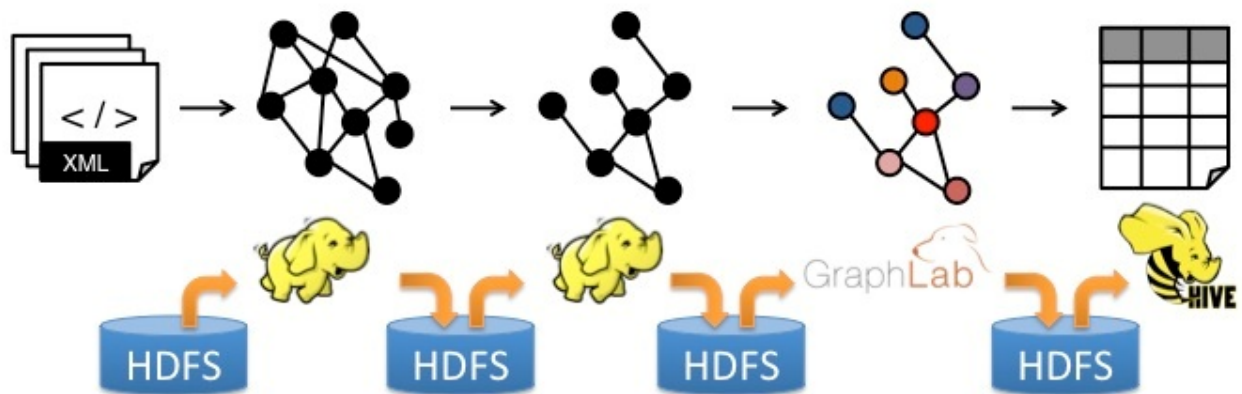
PageRank and Collaborative Filtering)的API,可以认为是GraphLab(C++)和Pregel(C++)在Spark(Scala)上的重写及优化,跟其他分布式图计算框架相比, GraphX最大的贡献是,在Spark之上提供一栈式数据解决方案,可以方便且高效地完成图计算的一整套流水作业。GraphX最先是伯克利AMPLAB的一个分布式图计算框架项目,后来整合到Spark中成为一个核心组件。

Example Graph Analytics Pipeline



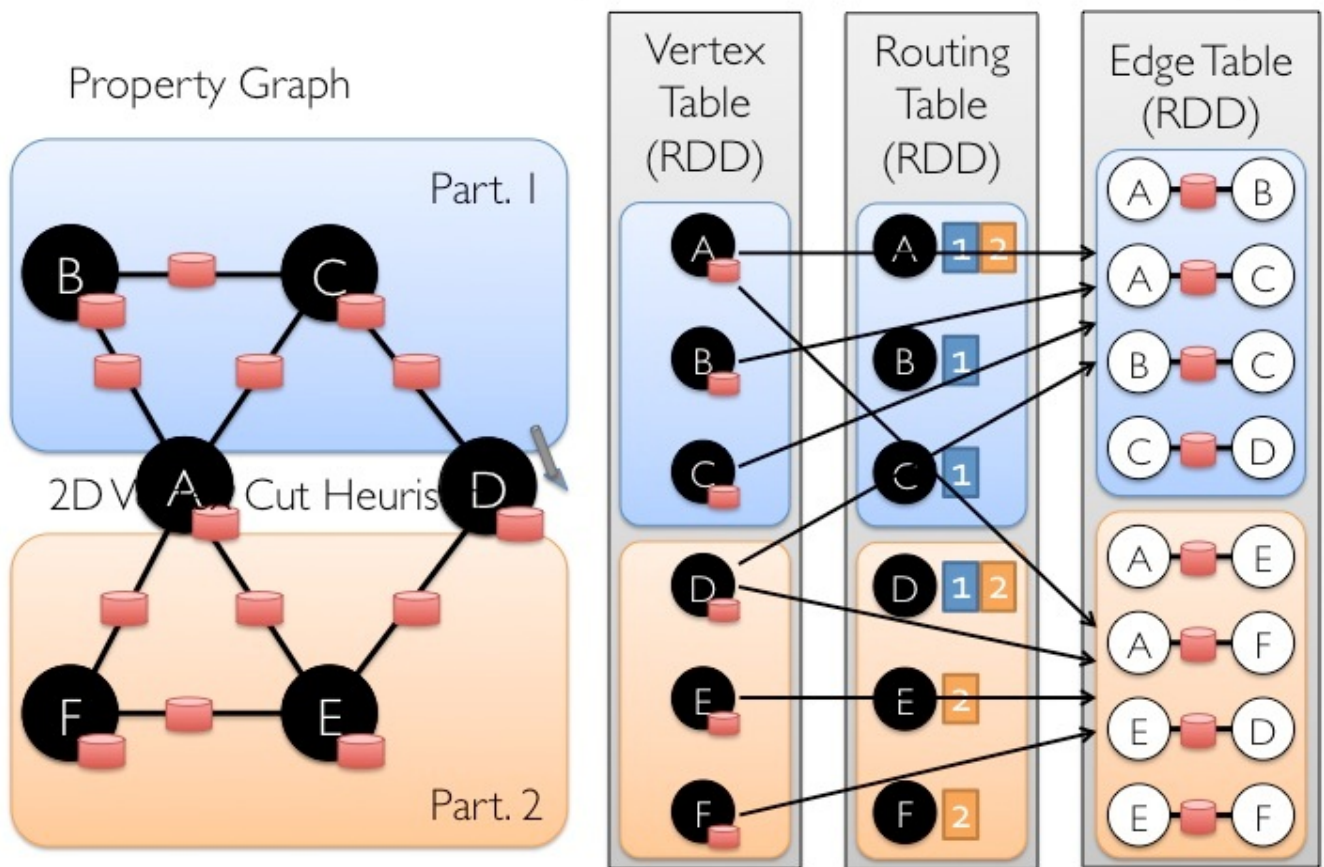
Inefficient

Extensive **data movement** and **duplication** across the network and file system



Limited reuse internal data-structures across stages

Distributed Graphs as Tables (RDDs)



三、Petuum

Petuum是一个机器学习专用分布式计算框架，本文介绍其架构，并基于文章 More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server, NIPS 2013 重点探讨其核心内容SSP协议。

(1) Background:

1. 数据多
2. 模型复杂
3. 对上面的数据处理的支持不够

BSP协议和Asynchronous协议的弱点

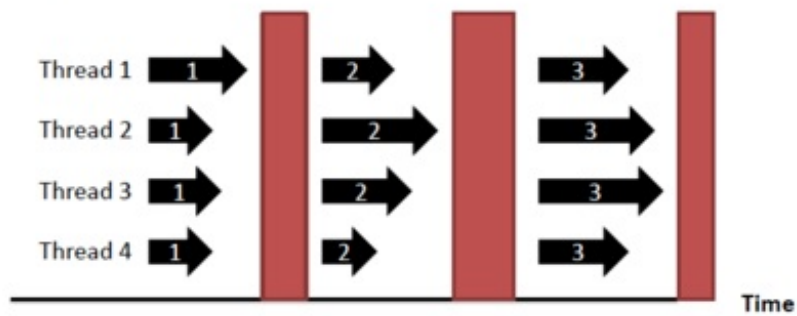


Bulk Synchronous Parallel(BSP)

Synchronization Barrier (Parameters read/updated here)

(a) Machines perform unequally

(b) Algorithmic workload imbalanced



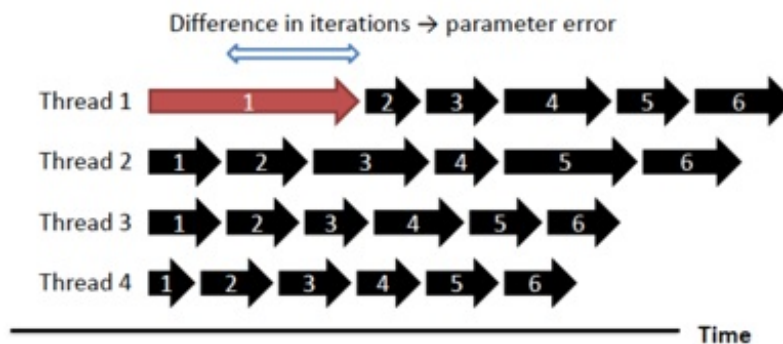
4



Asynchronous

Parameters read/updated at any time

Asynchronous is fast but has weak convergence guarantees



5

(2) Parameter Server

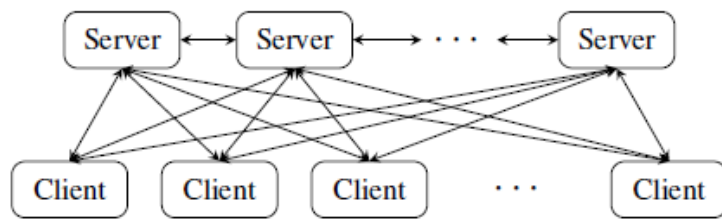


Figure 1: Communication pattern between clients and servers. Clients process data while servers synchronize parameters and perform global updates. Note that most code is shared between clients and servers, the main difference being the manner in which they update parameters.

(3) Stale Synchronous Parallel (SSP)

一个分布式系统有 P 个workers，需要训练的参数是 \mathbf{x} ，更新公式是 $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{u}$ 。每一个worker有一个时钟 c 也就是代表了循环计算了几次。SSP协议的关键是每个worker更新的参数不是马上对其它worker可见的，相反的他可见的是有一个延迟参数 s 控制的过去一个时刻版本的参数。通过这种延迟，有一个好处是，每一个worker可以从本地的缓存读取 \mathbf{x} 不用通过Server去读取 \mathbf{x} 节省了计算机通讯开销。

bounded staleness conditions :

1. 最慢的时钟和最快的时钟不能相差超过 s 个时刻
2. 当一个worker在 c 时刻提交一个update \mathbf{u} ， \mathbf{u} 被标记为 c 时刻
3. 每一个worker至少可以看到 $c - s - 1$ 时刻以前所有worker的update
4. 每一个worker一直可以看到自己的更新作用

$$\tilde{\mathbf{x}}_{p,c} = \mathbf{x}_0 + \underbrace{\left[\sum_{c'=1}^{c-s-1} \sum_{p'=1}^P \mathbf{u}_{p',c'} \right]}_{\text{guaranteed pre-window updates}} + \underbrace{\left[\sum_{c'=c-s}^{c-1} \mathbf{u}_{p,c'} \right]}_{\text{guaranteed read-my-writes updates}} + \underbrace{\left[\sum_{(p',c') \in \mathcal{S}_{p,c}} \mathbf{u}_{p',c'} \right]}_{\text{best-effort in-window updates}},$$

SSP: Bounded Staleness and Clocks

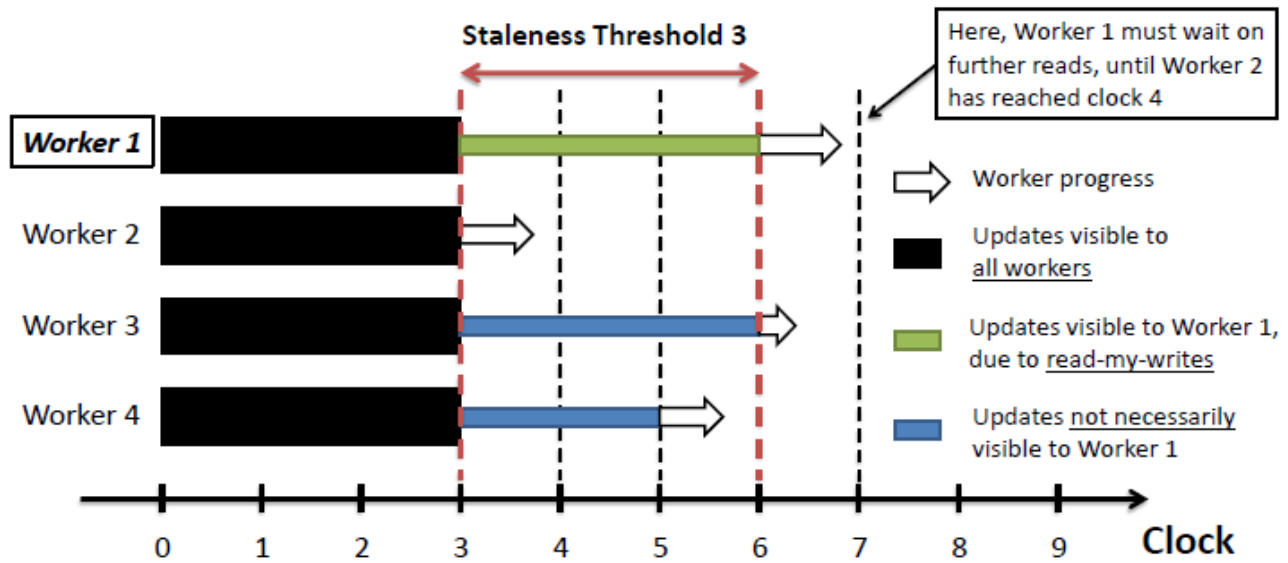


Figure 1: Bounded Staleness under the SSP Model

(4) SSPTable

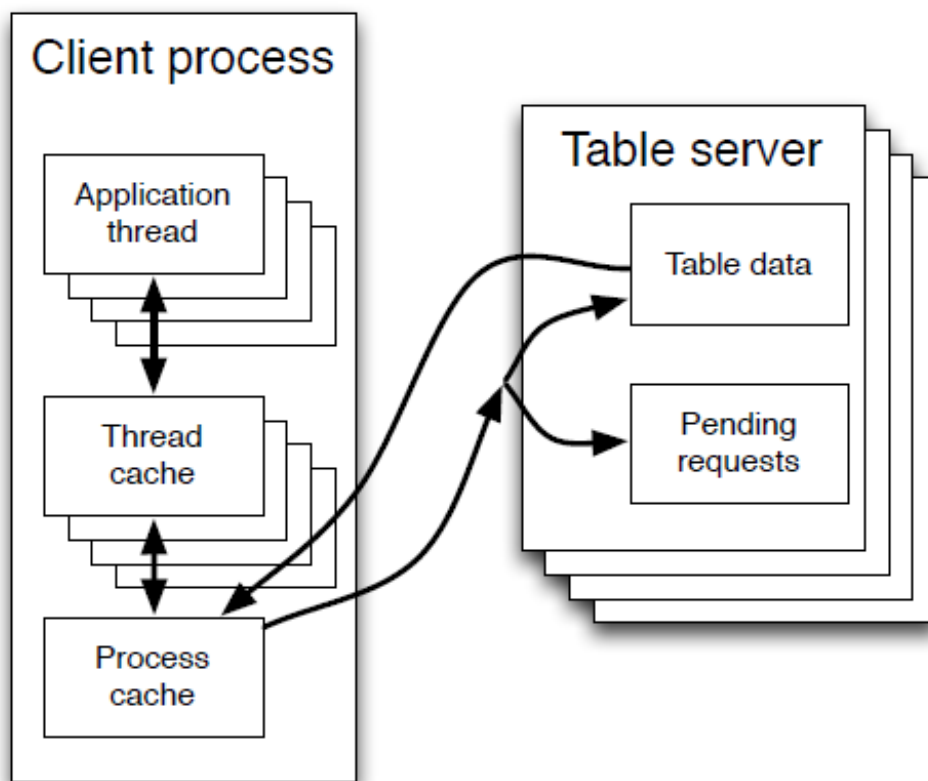


Figure 2: Cache structure of SSPtable, with multiple server shards

- **Table Organization:** SSPtable supports an unlimited number of *tables*, which are divided into *rows*, which are further subdivided into *elements*. These tables are used to store x .
- `read_row(table, row, s)`: Retrieve a table-row with staleness threshold s . The user can then query individual row elements.
- `inc(table, row, el, val)`: Increase a table-row-element by val , which can be negative. These changes are not propagated to the servers until the next call to `clock()`.
- `clock()`: Inform all servers that the current thread/processor has completed one clock, and commit all outstanding `inc()`s to the servers.

Server的时钟是最慢的时钟。快的worker会在`read_row`里面等慢的worker。在读数据的时候先从Thread Cache里面读，如果没有版本在 $\geq c - s$ 的数据就从Process Cache读，有就拿Process Cache里的数据更新Thread的数据，再没有就从Parameter Server里面读，这一步的网络通讯是十分耗时的。这样慢的worker只需要 s 个循环通过网络更新一次就行了，快的worker就要不断进行网络通讯，自然慢的worker就可以赶上快的worker了。

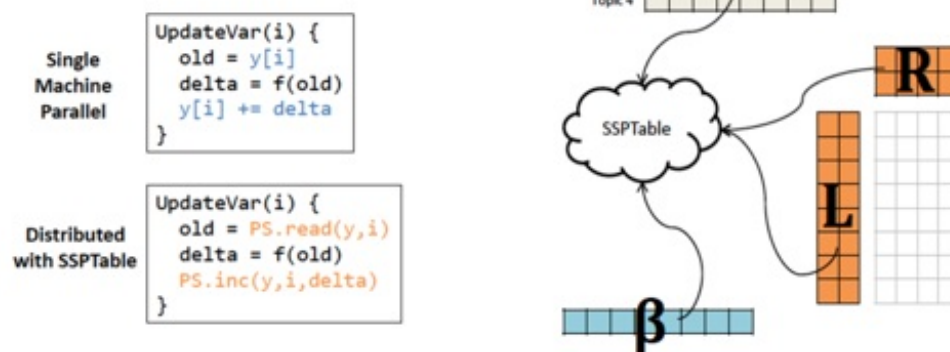
Petuum提供了分布式共享global模型参数的接口，使得很容易可以将多线程版本算法修改为Petuum版本。



SSP + Parameter Server (SSPTable)

Put global parameters in SSPTable!

Local memory access => PS access



(5) Theoretical Analysis

Convergence Guarantee

Details in **More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server** , NIPS 2013 Section IV

Reference

<http://ijcai-15.org/downloads/tutorials/T29-ML.pdf>

J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

<http://www.cnblogs.com/wei-li/archive/2012/04/01/2429448.html>

Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "GraphLab: A New Parallel Framework for Machine Learning " (UAI 2010)

Low, Yucheng, et al. "Distributed GraphLab: A Framework for Machine Learning in the Cloud" Proceedings of the VLDB Endowment 5.8 (2012)

https://en.wikipedia.org/wiki/Bulk_synchronous_parallel

Pregel: A System for Large-Scale Graph Processing

Xin, Reynold S., et al. "**GraphX: Unifying Data-Parallel and Graph-Parallel Analytics.**" arXiv preprint arXiv:1402.2394 (2014).

More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server , NIPS 2013

Mu Li. **Scaling Distributed Machine Learning with the Parameter Server.**