

可扩展的大图数据管理框架 和查询处理

高军

信息科学技术学院

提纲

♠ 背景和意义

♠ 单机大图数据管理框架

♠ 分布式大图数据管理框架

背景-图数据出现在不同的应用领域

社交网络



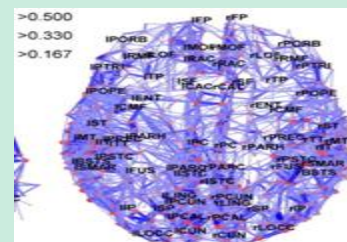
>1 billion vertices
~1 trillion edges

Web数据



>50 billion vertices
>1 trillion edges

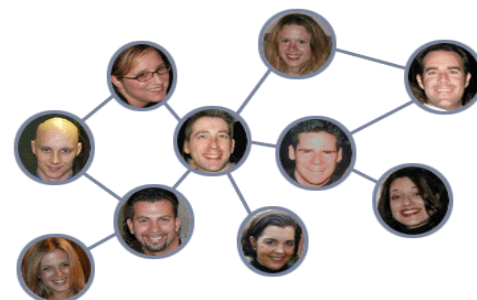
生物数据



>100 billion vertices
>100 trillion edges

♠ 大数据中不仅仅有数据项的信息，也有数据项之间的关联信息

♠ 我们称这些规模庞大、结构复杂的数据为
大图(Big Graph)数据

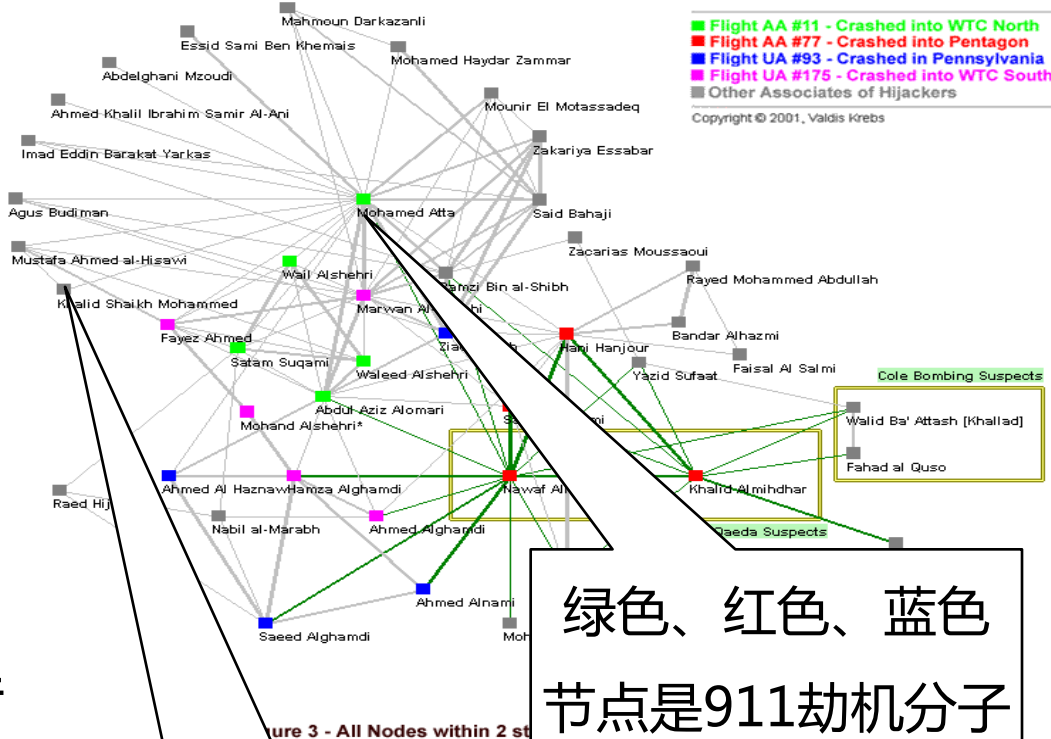


大图数据分析的应用示例

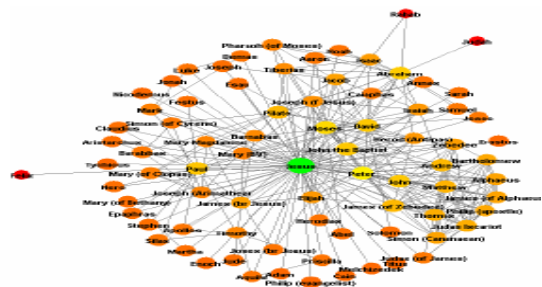
♠ 图数据分析有助于判定敏感人物或者社区

♣ 美国911事件后构造的关系网络

♣ 2011年在伦敦骚乱和挪威枪击事件中，Twitter, Facebook等社交网络起到的传递鼓动信息和散播谣言等作用



大图数据管理在不同领域中应用



Social Network

♠ 特定节点的发现

- ♣ 发现社交网络中有影响力节点，用于广告发送
- ♣ 社交网络中发现潜在的犯罪嫌疑人



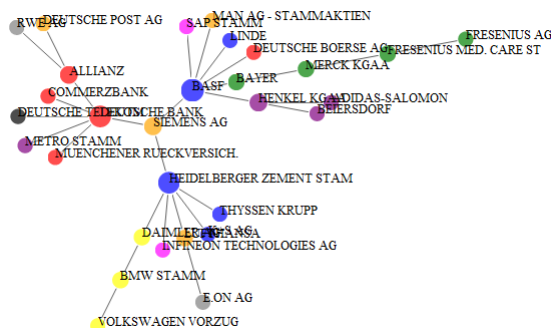
Program Flow

♠ 特定路径的发现

- ♣ 交通领域中的最短路
- ♣ 社交网络中异常路径的发现

♠ 特定子图的发现

- ♣ 生物领域中基因数据的频繁模式
- ♣ 社交网络的特定兴趣社区
- ♣ 财经网络中可疑交易集合
- ♣

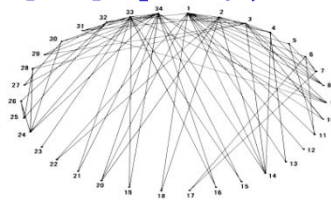


Financial Network

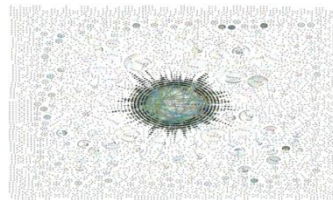
大图数据查询分析面临挑战

♠ 数据复杂性

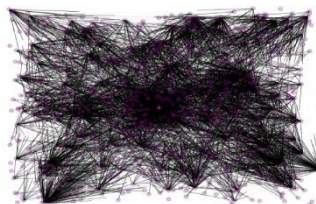
- ♣ 数据量大
- ♣ 结构+内容
- ♣ 数据之间的结构关联复杂



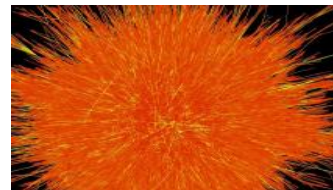
1970s~ 10^1
nodes



1990s~ 10^4
nodes



2000~ 10^8 +
nodes



2010~ 10^{10} +
nodes

♠ 查询复杂性

- ♣ 查询表达方式灵活
- ♣ 查询搜索空间大
- ♣ 图数据模型灵活

存不下：大图数据无法存储在单机内存
算不出：大图数据操作代价过高



北京大学

大图数据管理1：针对特定图查询编写方法

♠ 优点

- ♣ 面向大图数据处理的专有算法，处理效率高

♠ 缺点

- ♣ 系统实现代价高

- ◆ 大图数据分块、存储、索引、数据缓存策略、网络传输、副本、故障恢复.....

- ♣ 系统稳定性差

- ♣ 市场接受度差

大图数据管理2：图数据管理系统

- ♠ 实现以图模型为底层存储模型的数据管理系统
- ♠ 支持图数据的存储，索引等
- ♠ 提供图数据的基本操作，如遍历、最短路等
- ♠ 简化用户应用程序开发代价
- ♠ 但是
 - ♣ 图数据的操作过于灵活、复杂，有限的几种操作很难满足应用的需求

大图数据管理3：图数据管理框架

- ♠ 支持图数据的透明存储、任务调度等公有操作
- ♠ 提供合适的底层接口，允许用户表达各类查询
 - ♣ 接口尽可能简单
 - ♣ 通过接口实现尽可能多的查询操作
 - ◆ PageRank、最短路、社区发现、异常点发现、可达性发现

抽象：不同类型的图操作都是 ???

系统优化公共部分

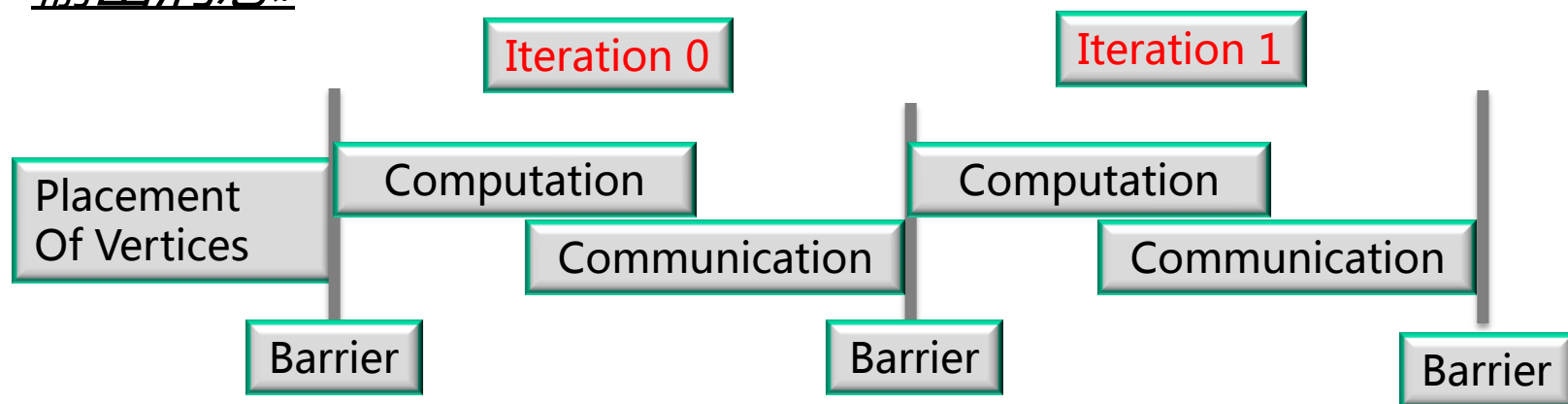
用户编写不同图操作特有的代码，提高通用性



北京大学

以点为中心计算模型的图数据管理框架

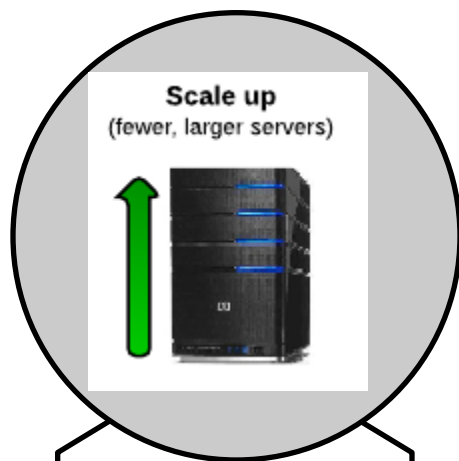
- ♠ 不同类型的图操作都是一系列迭代组成，在每次迭代中在每次迭代中，每个节点接受消息、按照用户输入的脚本处理消息、输出消息



- ♠ 主流的框架大多遵从以点为中心的计算模式

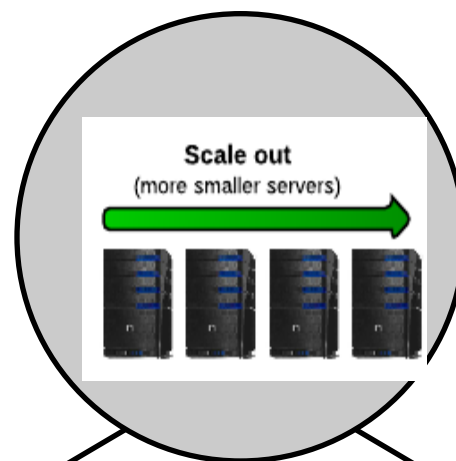
Google	Microsoft	Apache/Facebook	CMU	CMU
				

大数据处理的通常思路



更多CPU
更大内存
更多存储

单机大图数据管理



更多的
计算节点
分布并行
计算

分布式大图数据管理

提纲

♠ 背景和意义

♠ 单机大图数据管理框架

♠ 分布式大图数据管理框架

单机大图数据管理框架

♠ 单机程序开发效率相对高

♠ 单机程序不需要考虑网络传输代价，相对效率高

♠ 单机安装、管理、部署代价低

♠ 单机运行程序能耗低

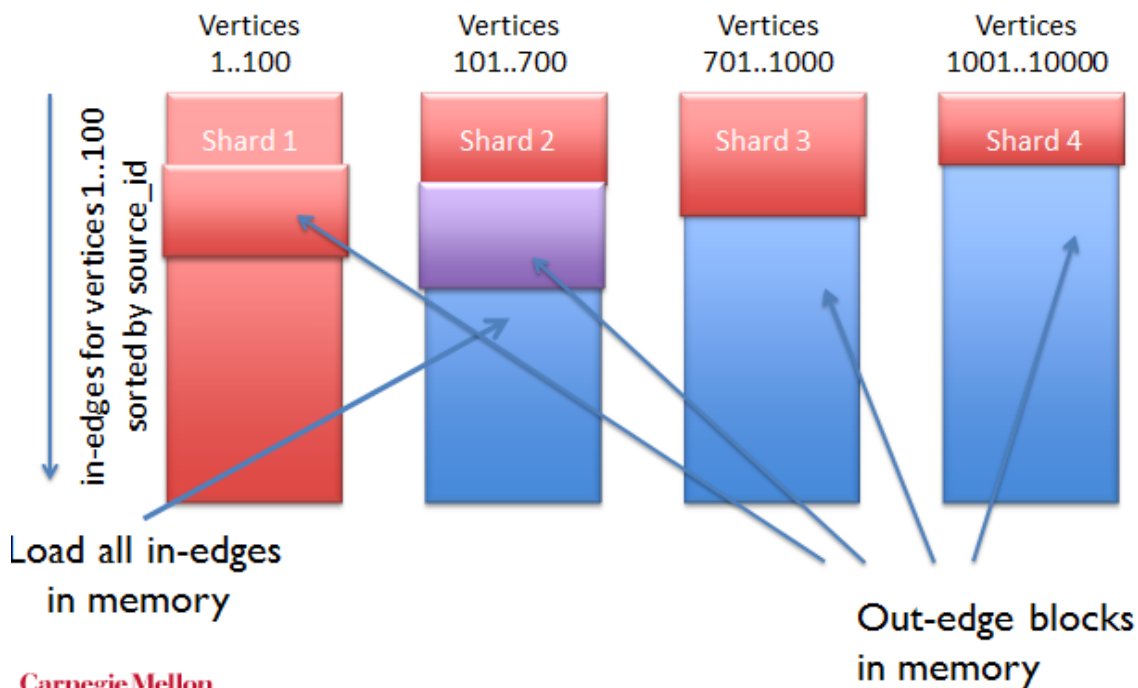
典型框架介绍：GraphChi

♠ Graphchi是CMU开发的单机图数据管理框架，发表于OSDI 2012

♠ 支持超过内存的图数据有效处理

♠ 图的操作中涉及到大量的数据随机访问操作，GraphChi利用顺序读写代替随机读写降低外存影响

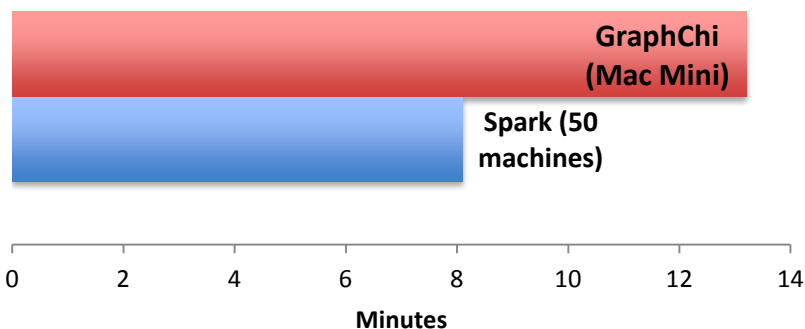
♠ 单机上获得和分布式集群可比较的效率



典型框架介绍：Mac Mini上Graphchi的测试结果

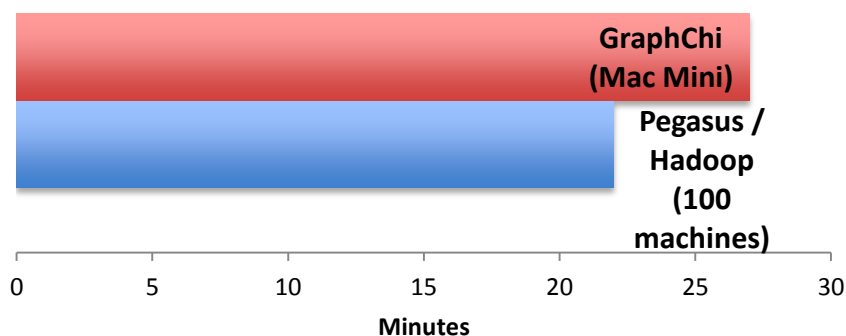
PageRank

Twitter-2010 (1.5B edges)



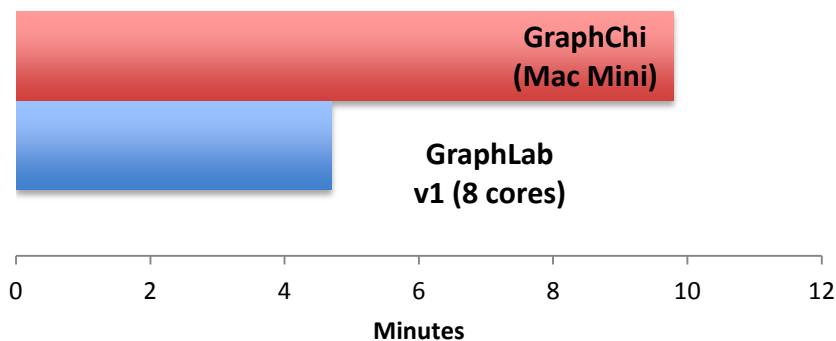
WebGraph Belief Propagation (U Kang et al.)

Yahoo-web (6.7B edges)



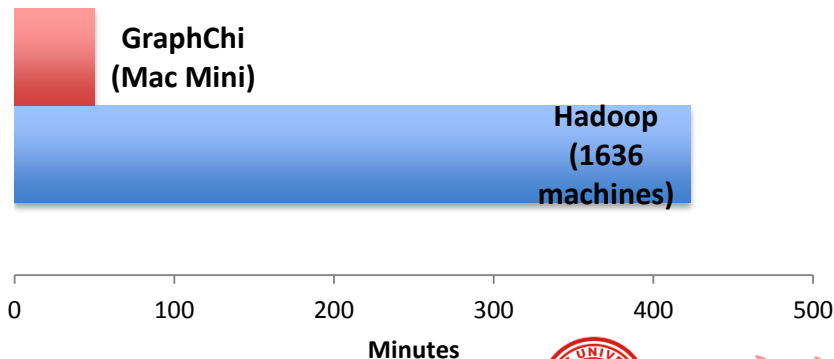
Matrix Factorization (Alt. Least Sqr.)

Netflix (99B edges)



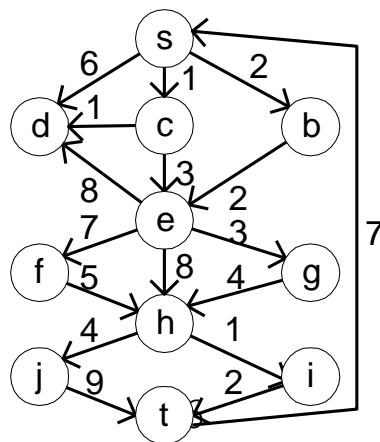
Triangle Counting

twitter-2010 (1.5B edges)



基于关系数据库管理图数据

- ♠ 关系数据库是一种成熟的系统软件，能够高效的管理超过内存的**结构化表格**数据
- ♠ 将图数据以表格的形式存储起来，利用关系数据库的存储能力和查询能力来管理图数据
- ♠ 扩展关系数据库支持图数据，也是关系数据库发展的重要技术思路



TNodes

<u>nid</u>
s
b
.....

TEEdges

<u>fid</u>	<u>tid</u>	<u>cost</u>
s	d	6
d	c	1
s	b	2
...



基于关系数据库的图查询方法

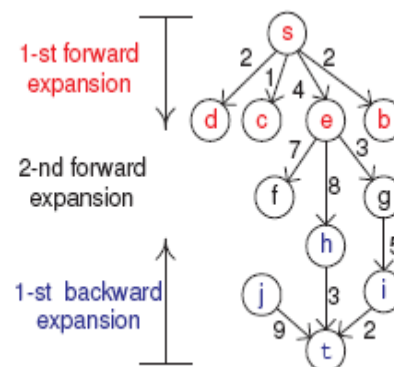
Gao, et al VLDB 2012

♠ 性能挑战：

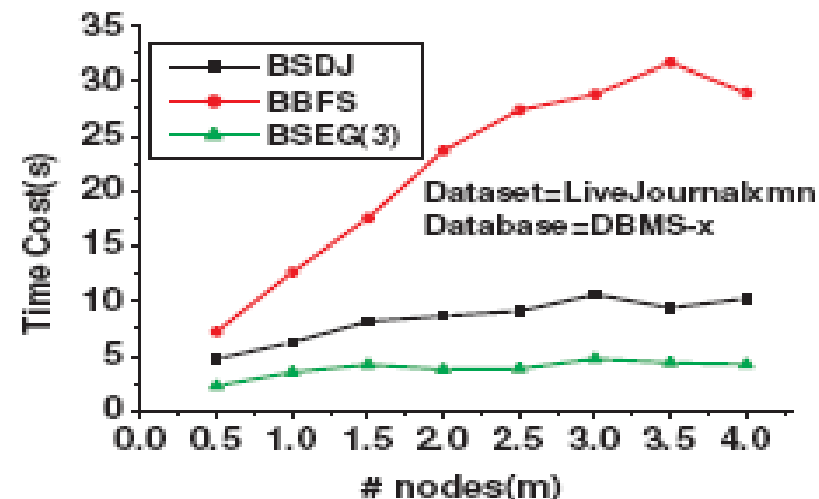
- ♣ 关系数据库的操作一次一个集合，图操作一次一个记录，这种差异使得直接使用关系数据库查询图数据效率低

♠ 贡献：

- ♣ 提出关系数据库管理图数据FEM (Frontier-Expand-Merge) 框架
- ♣ 设计利用SQL语言的新标准，包括Window function和merge语句提高FEM框架的效率



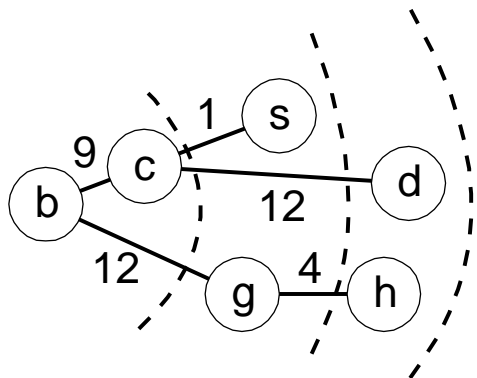
p2s	d2s	f	nid	b	d2t	p2t
s	0	1	s			
s	2	1	b			
s	1	1	c			
s	2	1	d			
s	4	1	e			
e	11	0	f			
e	7	0	g			
e	12	0	h	0	3	t
			i	0	2	t
			j	0	9	t
			t	1	0	t



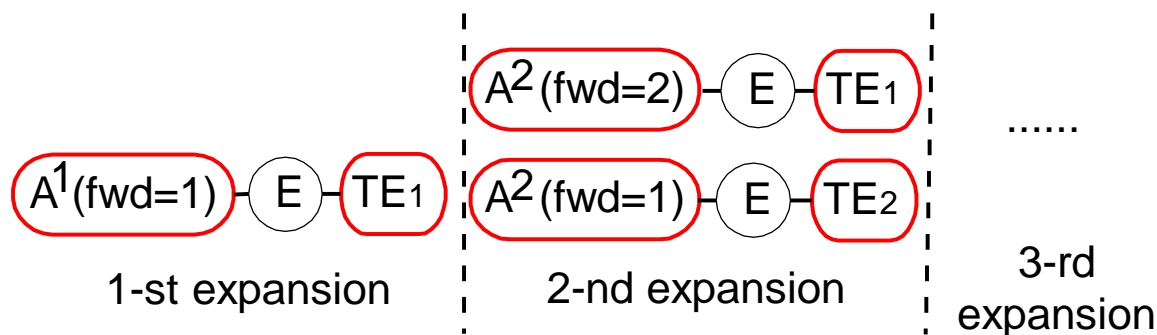
基于权重分表的图查询方法

Gao, et al. TKDE 2014

- ♠ 关系数据库环境中，图的遍历通过边表的连接操作完成
- ♠ 随着图规模的增长，边表规模变得很大，连接操作代价高
- ♠ 将表按照权重进行划分，同时调整搜索算法



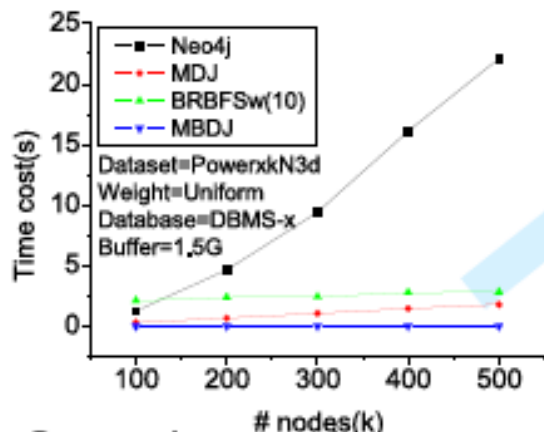
(a) Example of Restrictive BFS



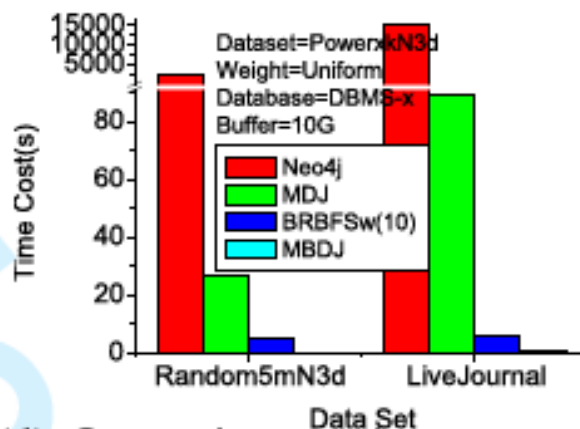
(b) Illustration of Extended E-operator

和著名开源系统对比

- ♠ Neo4j号称是世界领先的图数据库系统，支持图的最短路发现等基本操作



(c) Query time vs. in-memory methods



(d) Query time vs. in-memory methods

基于分表之后的关系数据库操作的方法再处理有权图最短路方面超过Neo4j几个数量级

提纲

♠ 研究背景和意义

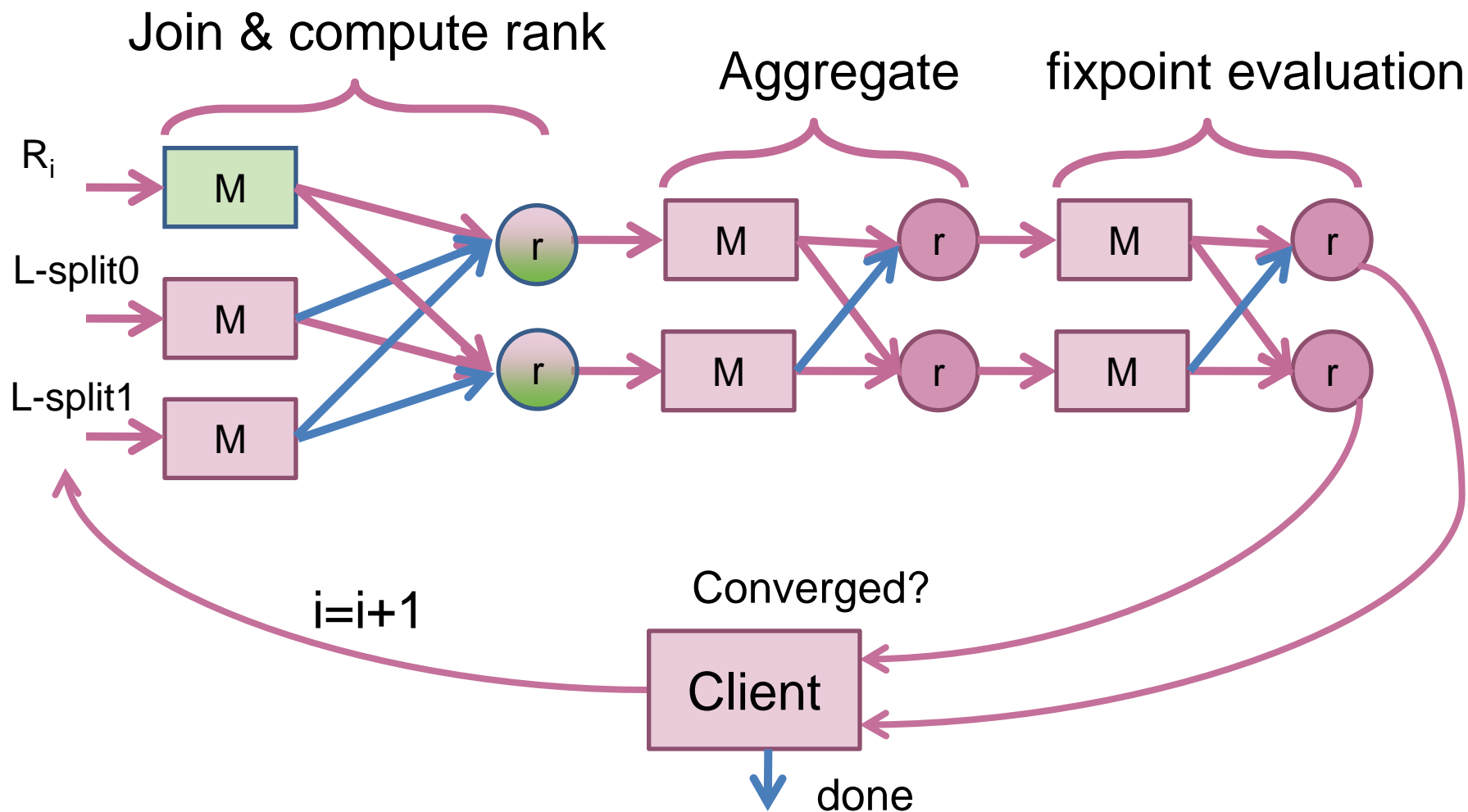
♠ 单机大图数据管理框架

♠ 分布式大图数据管理框架

MapReduce分布式框架

- ♠ MapReduce是Google提出的分布式计算框架，极大简化最终用户分布式编程的代价
- ♠ Hadoop是MapReduce的开源实现。
- ♠ Hadoop系统扩展性极强，集群支持的计算节点数超过4000个
- ♠ Hadoop系统广泛用于非结构化数据处理，如文本处理、日志处理、数据分析等，被称为大数据处理事实标准。

基于MapReduce图操作



基于MapReduce的图数据管理面临的问题

WordCount.java

```
1. package org.myorg;
2.
3. import java.io.IOException;
4. import java.util.*;
5.
6. import org.apache.hadoop.fs.Path;
7. import org.apache.hadoop.conf.*;
8. import org.apache.hadoop.io.*;
9. import org.apache.hadoop.mapred.*;
10. import org.apache.hadoop.util.*;
11.
12. public class WordCount {
13.
14.     public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
15.         private final static IntWritable one = new IntWritable(1);
16.         private Text word = new Text();
17.
18.         public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter)
19.             throws IOException {
20.             String line = value.toString();
21.             StringTokenizer tokenizer = new StringTokenizer(line);
22.             while (tokenizer.hasMoreTokens()) {
23.                 word.set(tokenizer.nextToken());
24.                 output.collect(word, one);
25.             }
26.         }
27.
28.         public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
29.             public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output,
30.                 Reporter reporter) throws IOException {
31.                 int sum = 0;
32.                 while (values.hasNext()) {
33.                     sum += values.next().get();
34.                 }
35.                 output.collect(key, new IntWritable(sum));
36.             }
37.
38.             public static void main(String[] args) throws Exception {
39.                 JobConf conf = new JobConf(WordCount.class);
40.                 conf.setJobName("wordcount");
41.
42.                 conf.setOutputKeyClass(Text.class);
43.                 conf.setOutputValueClass(IntWritable.class);
44.
45.                 conf.setMapperClass(Map.class);
46.                 conf.setCombinerClass(Reduce.class);
47.                 conf.setReducerClass(Reduce.class);
48.
49.                 conf.setInputFormat(TextInputFormat.class);
50.                 conf.setOutputFormat(TextOutputFormat.class);
51.
52.                 FileInputFormat.setInputPaths(conf, new Path(args[0]));
53.                 FileOutputFormat.setOutputPath(conf, new Path(args[1]));
54.
55.                 JobClient.runJob(conf);
56.             }
57.         }
58.     }
59. }
```

- 用户需要较强的编程能力

- 调试MapReduce框架程序困难

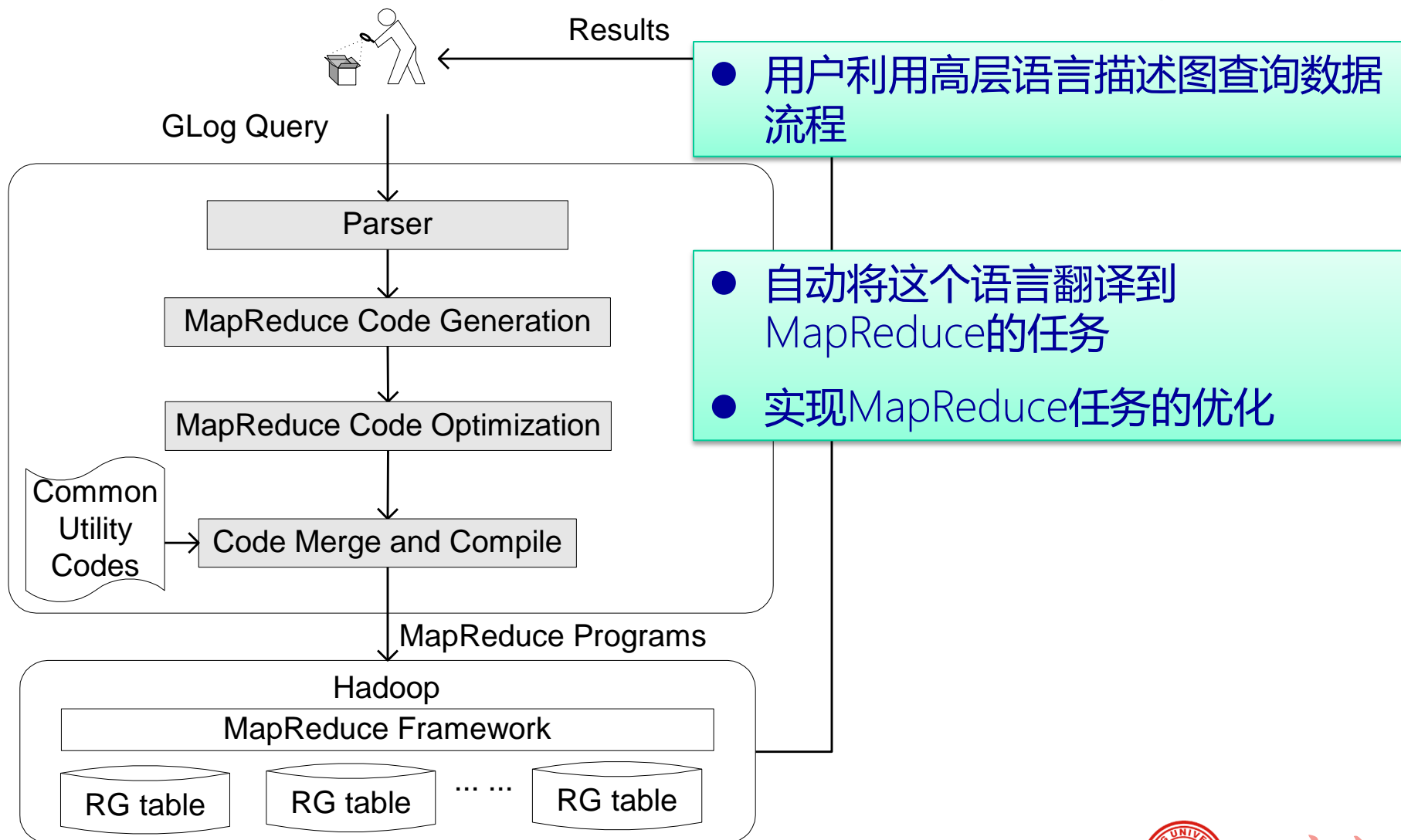
- 用户可能编写执行效率不高的程序

- Mapduce框架对循环的支持较弱



基于MapReduce的描述性图查询方法

Gao, et. ICDE 2014

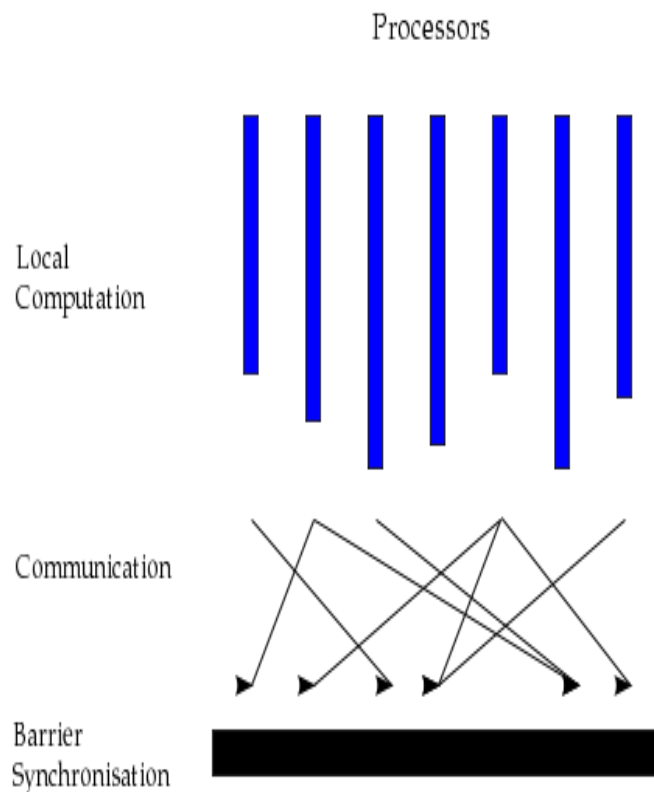


以点为中心的大图数据框架

- ♠ 为了克服MapReduce框架管理大图数据的问题
 - ♣ 大图数据节点间消息传递、状态修改全部通过文件系统来实现，导致大量的磁盘读写代价
- ♠ Google提出了基于BSP模型和以点为中心计算模式的Pregel系统，希望
 - ♣ 高可扩展性
 - ♣ 支持容错
 - ♣ 支持多种图操作
- ♠ Apache社区中出现了两个类似的子项目，Hama和Giraph

BSP模型和以点为中心计算

- ♠ 计算任务由超步组成
- ♠ 在每个超步中，执行
 - ♣ 用户给定的节点之上的操作序列
 - ♣ 在节点的操作序列中，输入参数包括其他节点发送的消息，输出结果包括向其他节点发出的消息
 - ♣ 节点可以投票终止超步，系统汇总决定循环是否终止



以点为中心计算-用户编写代码示例

```
Class MaxFindVertex
```

```
    : public Vertex<double, void, double> {
```

```
public:
```

```
    virtual void Compute(MessageIterator* msgs) {
```

```
        int currMax = GetValue();
```

```
        for ( ; !msgs->Done(); msgs->Next()) {
```

```
            if (msgs->Value() > currMax)
```

```
                currMax = msgs->Value();
```

```
        }
```

```
        if (currMax > GetValue())
```

```
            *MutableValue() = currMax;
```

```
            SendMessageToAllNeighbors(currMax);
```

```
        else VoteToHalt();
```

```
    }
```

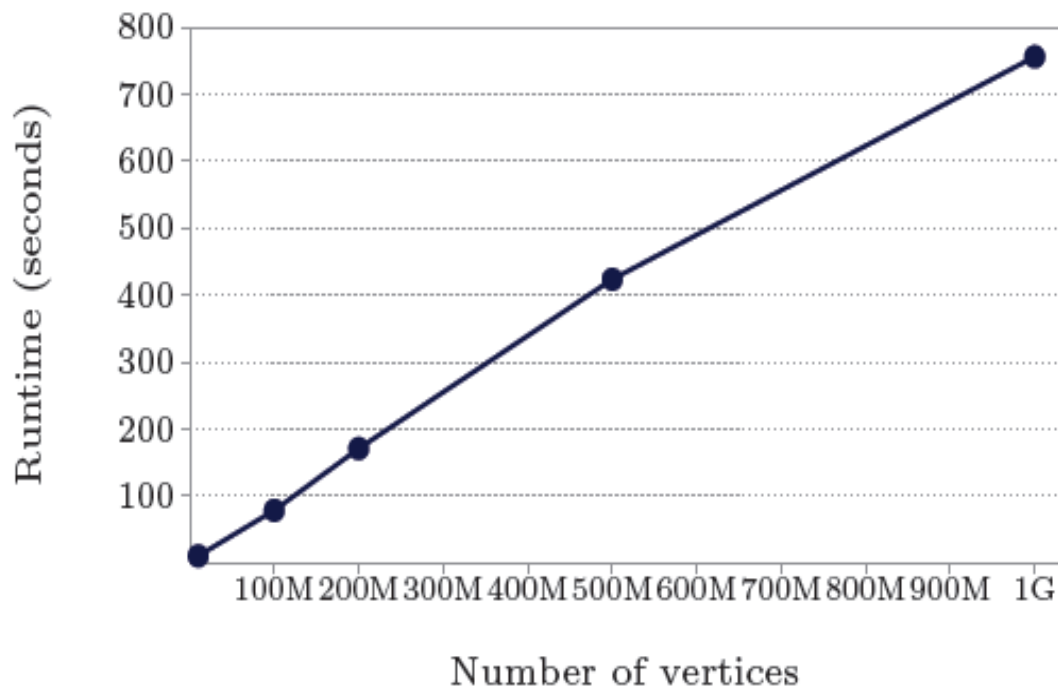
```
};
```

处理输入消息

输出消息

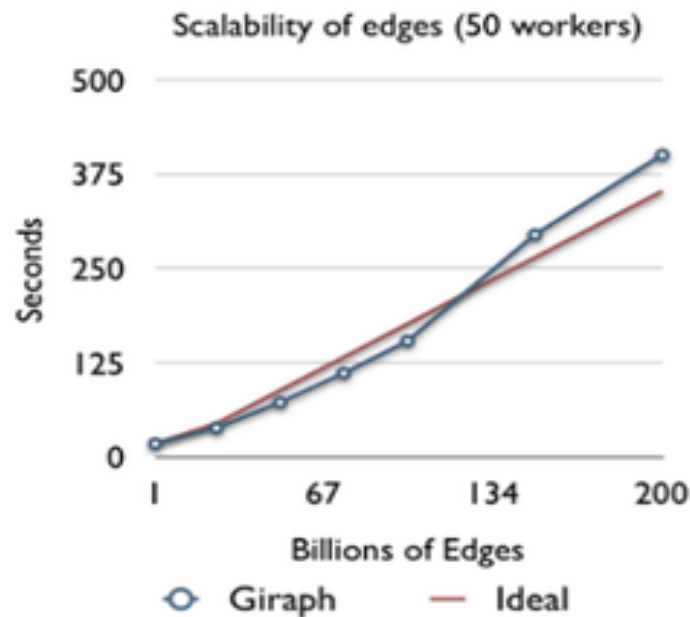
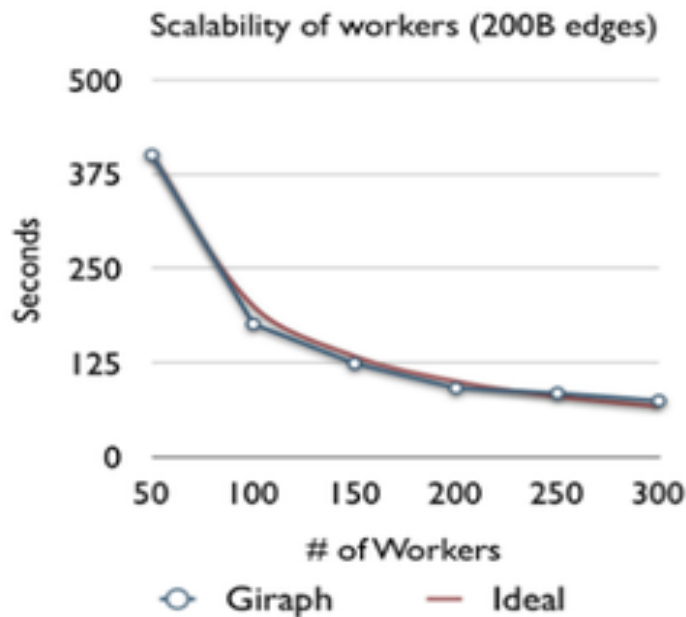


Google报告Prege1运行时间



- ♠ 在300台多核PC服务器组成的机群上，运行随机图的最短路查询，随机图的平均度数为127，那么在最大图上大约是1270亿条边的规模。在系统运行中，启动了800个worker。

Facebook报告Giraph运行时间



♠ Giraph在Facebook上的应用情况。在Facebook产品化已经1年半，每周超过100个任务与运行，支持内部的30个应用，单个应用处理超过7千亿条边。经过优化后，200台机器运行1万亿边的pagerank方法，每轮小于4分钟。

研究组的实验环境

♠ Hadoop集群

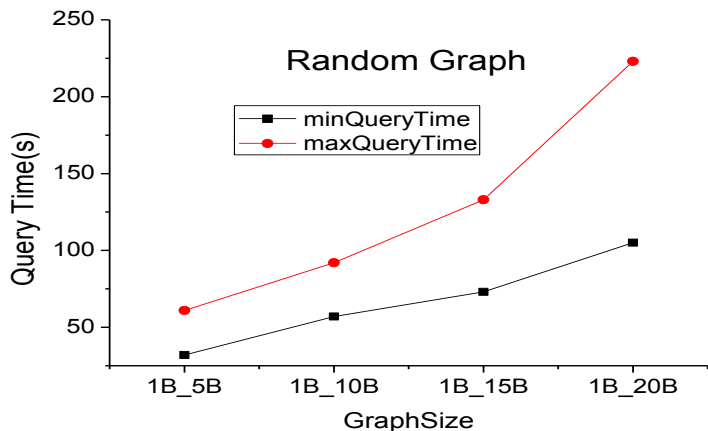
- ♣ 28个节点，每个节点2颗2.60GHz AMD Opteron 4180 处理器，48G内存，10T硬盘
- ♣ 我们安装了SUSE Linux Enterprise Server 11和 Java 1.7 64-bit
- ♣ 计算节点通过1G的网络连接

♠ 图数据规模

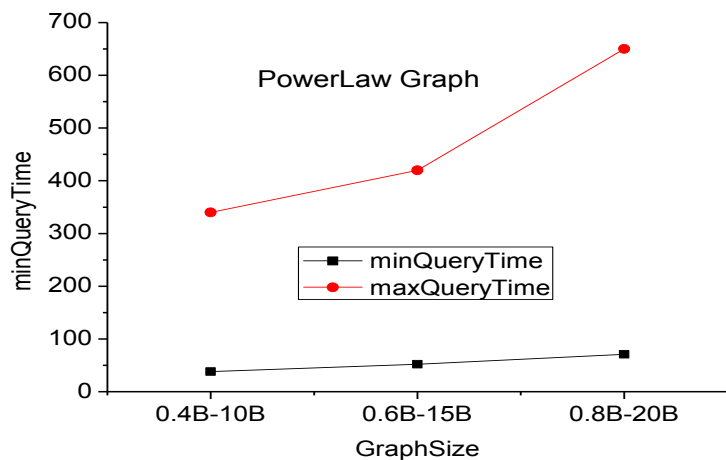
DataSet	# Nodes	# Edges	Storage(GB)
<i>livejournal</i>	4,847,571	68,993,773	0.73
<i>twitter-2010</i>	41,652,230	1,468,365,182	12.1
<i>uk-2007-05</i>	105,896,555	3,738,733,648	30.5
<i>random</i>	1.0×10^9	6.0×10^9	71.5



Giraph集群压力测试结果



# Nodes	# Edges	minQueryTime	maxQueryTime
1B	5B	32	61
1B	10B	57	92
1B	15B	73	133
1B	20B	105	223

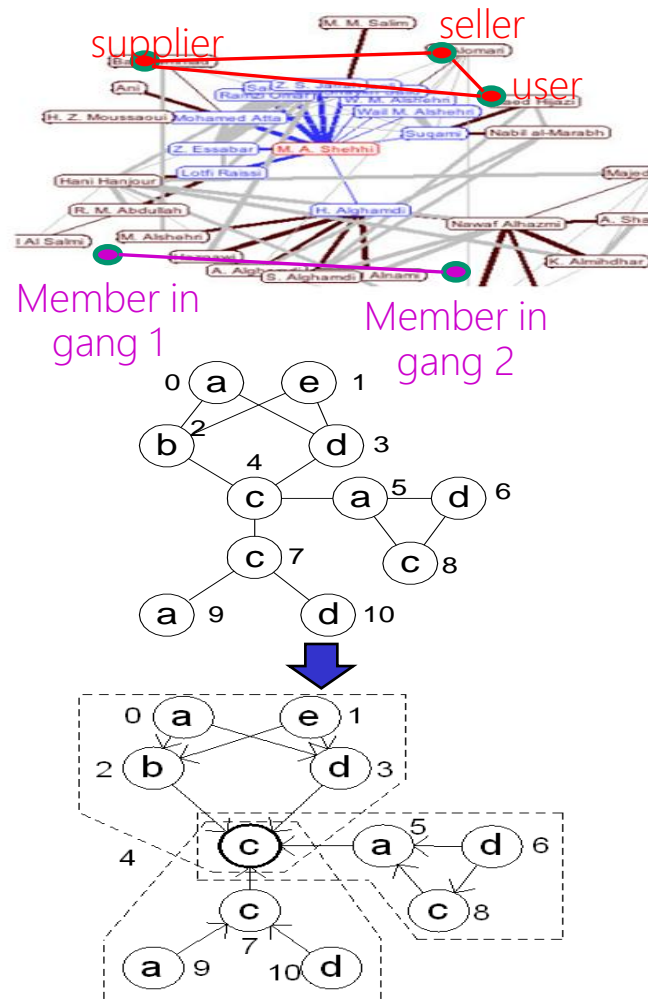


# nodes	# edges	minQueryTime	maxQueryTime
0.4B	10B	38	340
0.6B	15B	52	420
0.8B	20B	71	650

基于类Pregel框架的动态图上模式的检测

Gao, et al. ICDE 2014

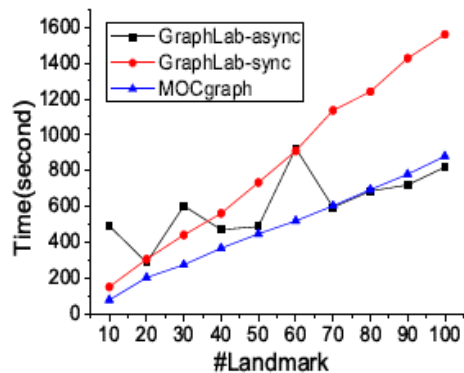
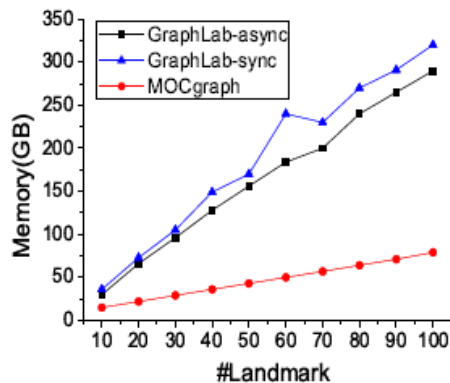
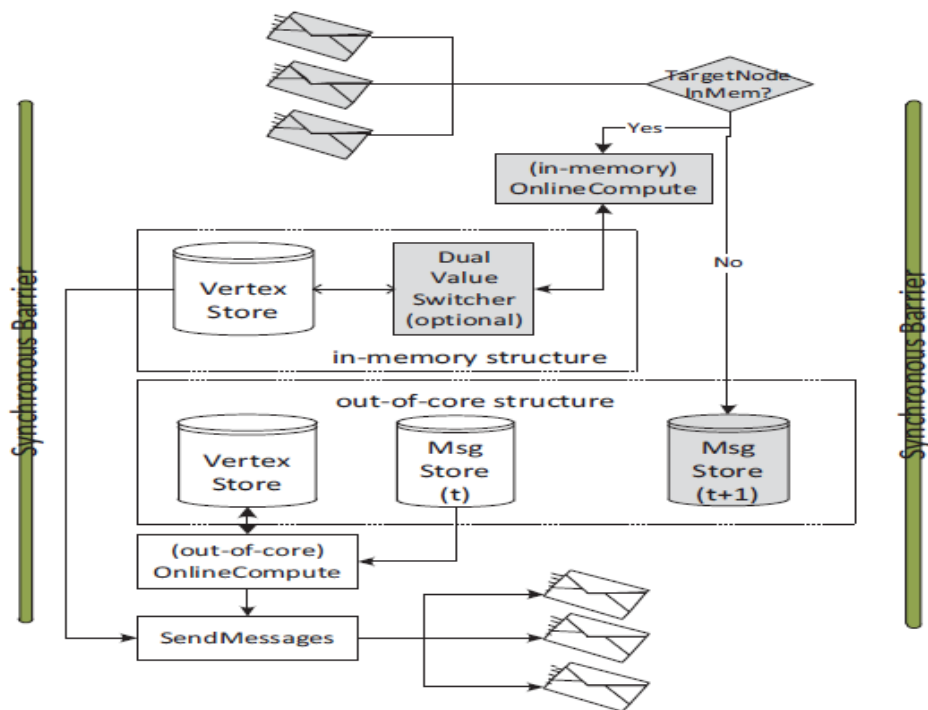
- ♠ 图模式查询本身是NP-C问题，大图、动态图带来更多的挑战。
- ♠ 基本框架
 - ♣ 图数据分布存储于不同的计算节点中
 - ♣ 模式查询的执行基于数据节点之间的消息驱动
 - ♣ 查询执行类似于自动机的运行
- ♠ 利用分布式计算框架支持超过10亿边的动态大图模式匹配
- ♠ 对比现有方法，我们的方法在消息量和相应时间方面优势明显



基于消息流式处理的分布式计算框架

Zhou, et al. VLDB2015

- ♠ 类Pregel框架将图数据保存在内存中
- ♠ 图数据自身和计算临时结果规模庞大
- ♠ 一旦超出内存，现有框架奔溃或者性能严重下降
- ♠ 提出利用流式处理机制减少内存消耗的方法，在Giraph框架中实现，提高系统的可扩展性



总结

- ♠ 图数据广泛出现在不同的应用领域.
- ♠ 图数据的处理需要考虑结构信息，处理复杂性高
- ♠ 利用图数据管理框架，能够简化用户编写图数据处理方法的代价，同时提高图数据查询的灵活度
- ♠ 目前出现多种单机和分布式的图数据管理框架，大多遵循以点为中心的计算模式

研究组进展



敬请指正！

gaojun@pku.edu.cn