ТЕХНОСФЕРА

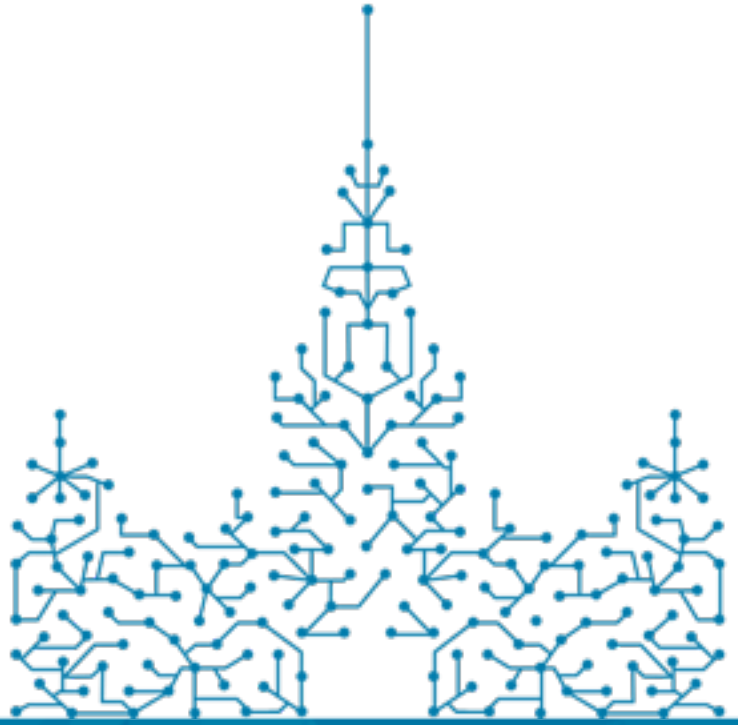
Методы распределенной обработки больших объемов данных в Hadoop

Вычислительная модель Pregel

Москва 2014

План занятия

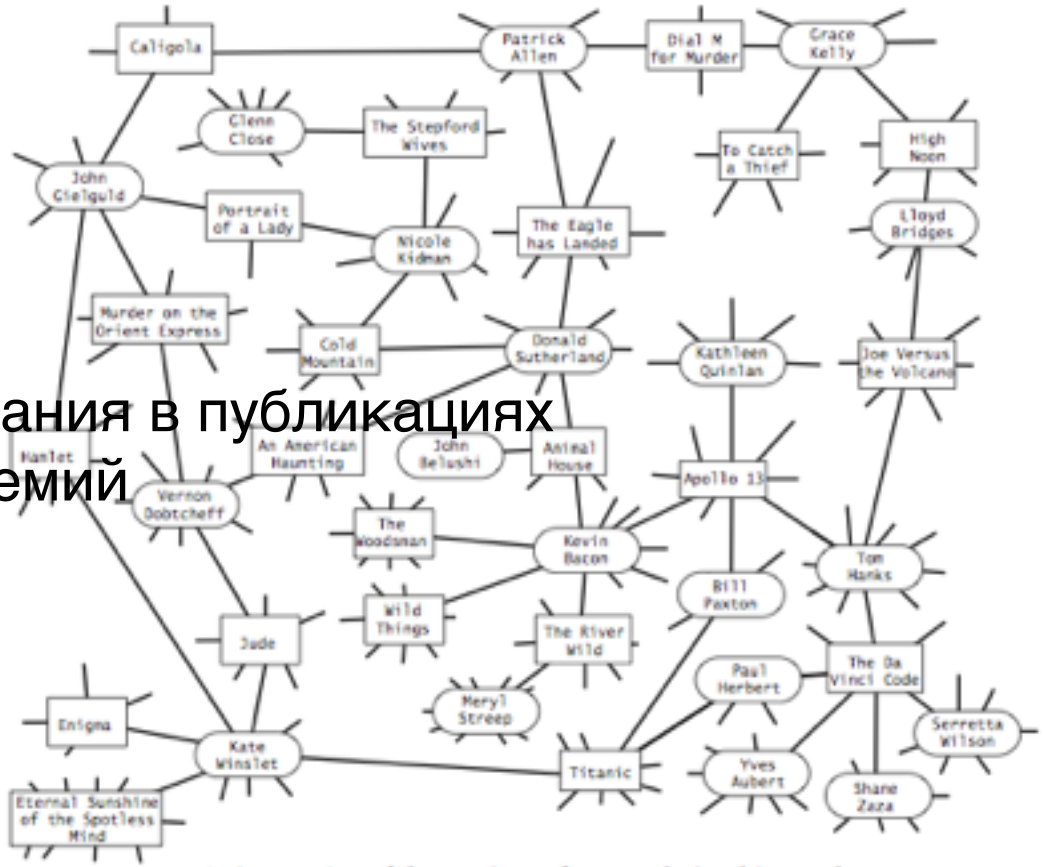
- Обработка больших графов и вычислительная модель Pregel
- Реализация Pregel: Apache Giraph
- Примеры алгоритмов



Web 2.0 и социальные графы

Примеры графов

- Страницы в Интернете
- Компьютерные сети
- Статьи в блогах
- Транспортные системы
- Отношения ко-цитирования в публикациях
- Распространение эпидемий



A tiny portion of the movie-performer relationship graph

Задачи обработки графов

- Работа с путями в графе
  - Остовные деревья (Kruskal, Prim)
  - Кратчайшие пути (Bellman-Ford, Dijkstra)
- Кластеризация (Affinity Propagation, Girvan–Newman)
- Ранжирование
  - Centralities: degree, betweenness, closeness, …
  - PageRank, HITS

Средства обработки больших графов

- Собственное распределенное решение
- Распределенные фреймворки общего назначения (Hadoop)
- Нераспределенные библиотеки обработки графов (NetworkX, Gephi, LEDA, GraphBase)
- Распределенные фреймворки обработки графов (**Giraph**, Bagel, Parallel BGL)

Pregel — мотивация

- Распределенные вычисления
  (вычисления легко распараллеливаются на кластер)
- Локальность вычислений
  (вычисления, соответстующие каждой вершине независимы)
- Отказоустойчивость
  (успешность выполнения задачи при выходе из строя нескольких машин)
- Универсальность
  (различные форматы входных и выходных данных)
- Удобная вычислительная модель
  (поддержка большого количества алгоритмов)

Pregel — концепция

- Вычисления состоят из некоторого количества итераций (supersteps)
- Каждая вершина имеет значение (value), набор исходящих ребер (edges) и состояние: active или vote to halt
- Вершины общаются с помощью сообщений (messages), посылаемых через ребра
- Единицей вычисления является метод compute, который выполняется независимо для каждой вершины на каждой итерации.

Vertex

```java
/**
 * Basic abstract class for writing a BSP application for computation.
 * Giraph will store Vertex value and edges, hence all user data should
 * be stored as part of the vertex value.
 *
 * @param <I> Vertex id
 * @param <V> Vertex data
 * @param <E> Edge data
 * @param <M> Message data
 */
public abstract class Vertex<I extends WritableComparable, V extends Writable, E extends Writable, M extends Writable>
    extends DefaultImmutableClassesGiraphConfigurable<I, V, E, M> implements WorkerAggregatorUsage {
  /** Vertex id. */
  private I id;
  /** Vertex value. */
  private V value;
  /** Outgoing edges. */
  private OutEdges<I, E> edges;
  /** If true, do not do anymore computation on this vertex. */
  private boolean halt;

  public void setEdges(Iterable<Edge<I, E>> edges) {}

  public abstract void compute(Iterable<M> messages) throws IOException;

  public long getSuperstep() {}

  public I getId() {}

  public V getValue() {}

  public void setValue(V value) {}

  public Iterable<Edge<I, E>> getEdges() {}

  public void sendMessage(I id, M message) {}

  public void voteToHalt() {}
```

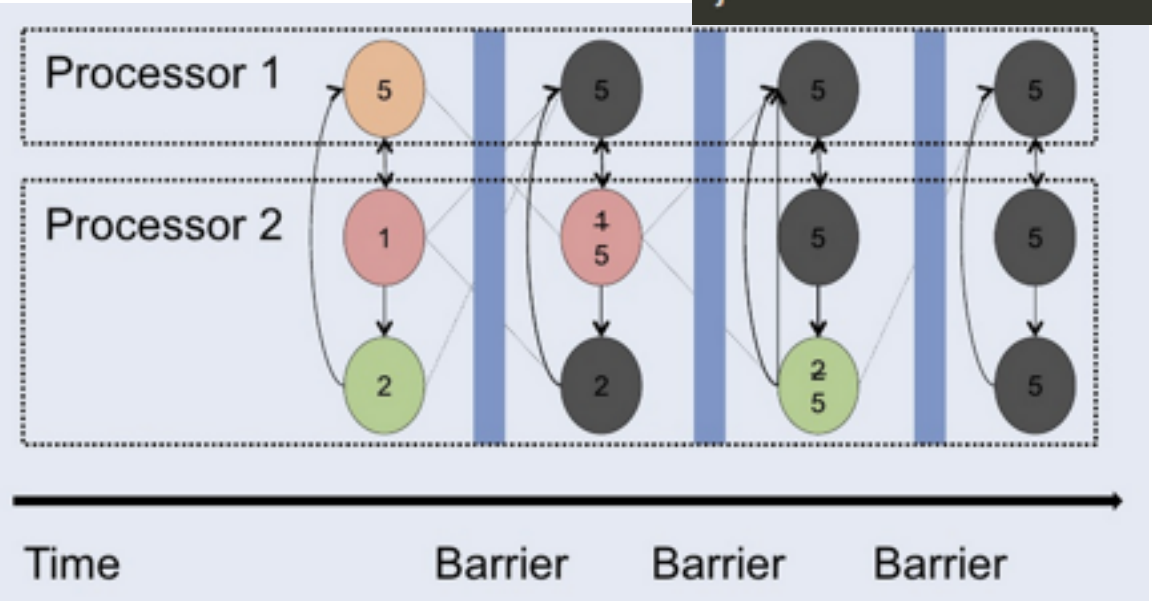
Метод compute

Разработчик программы наследует класс `Vertex`, определяя метод compute, в котором:

1. обрабатываются сообщения, посланные текущей вершине на предыдущей итерации
2. с учетом полученных сообщений, весов исходящих ребер и номера итерации изменяется value текущей вершины
3. вершинам, доступным по исходящим ребрам могут отсылаться сообщения

Метод compute

```java
public class MaxValueVertex
  extends Vertex<IntWritable, IntWritable, NullWritable, IntWritable> {

  @Override
  public void compute(Iterable<IntWritable> messages) throws IOException {
    boolean changed = false;
    for (IntWritable message : messages) {
      if (getValue().get() < message.get()) {
        setValue(message);
        changed = true;
      }
    }
    if (getSuperstep() == 0 || changed) {
      sendMessageToAllEdges(getValue());
    }
    voteToHalt();
  }
}
```



Combiner

Проблема: отправка сообщений – дорогая операция
Решение: по возможности скомбинировать сообщения перед отправкой (для коммутативных и ассоциативных операций)

```
/**
 * Abstract class to extend for combining messages sent to the same vertex.
 * Combiner for applications where each two messages for one vertex can be
 * combined into one.
 *
 * @param <I> Vertex id
 * @param <M> Message data
 */
public abstract class Combiner<I extends WritableComparable, M extends Writable> {

  public abstract void combine(I vertexIndex, M originalMessage, M messageToCombine);

  public abstract M createInitialMessage();
}
```

Aggregator

Механизм для глобальной коммуникации:

- Вершины отправляют значения на S итерации
- Pregel комбинирует эти значения
- Итоговое значение доступно на S+1 итерации

Применение:

- Расчет статистики
- Отслеживание выполнения программы
- Выбор «особой» вершины

```
 * Interface for Aggregator.  Allows aggregate operations for all vertices
 * in a given superstep.
 *
 * @param <A> Aggregated value
 */
public interface Aggregator<A extends Writable> {

  void aggregate(A value);

  A createInitialValue();

  A getAggregatedValue();

  void setAggregatedValue(A value);

  void reset();
}
```

Изменение графа

Некоторые алгоритмы могут требовать изменения графа

- Minimum Spanning Tree
- Divisive Hierarchical Clustering

Изменения происходят упорядоченно:

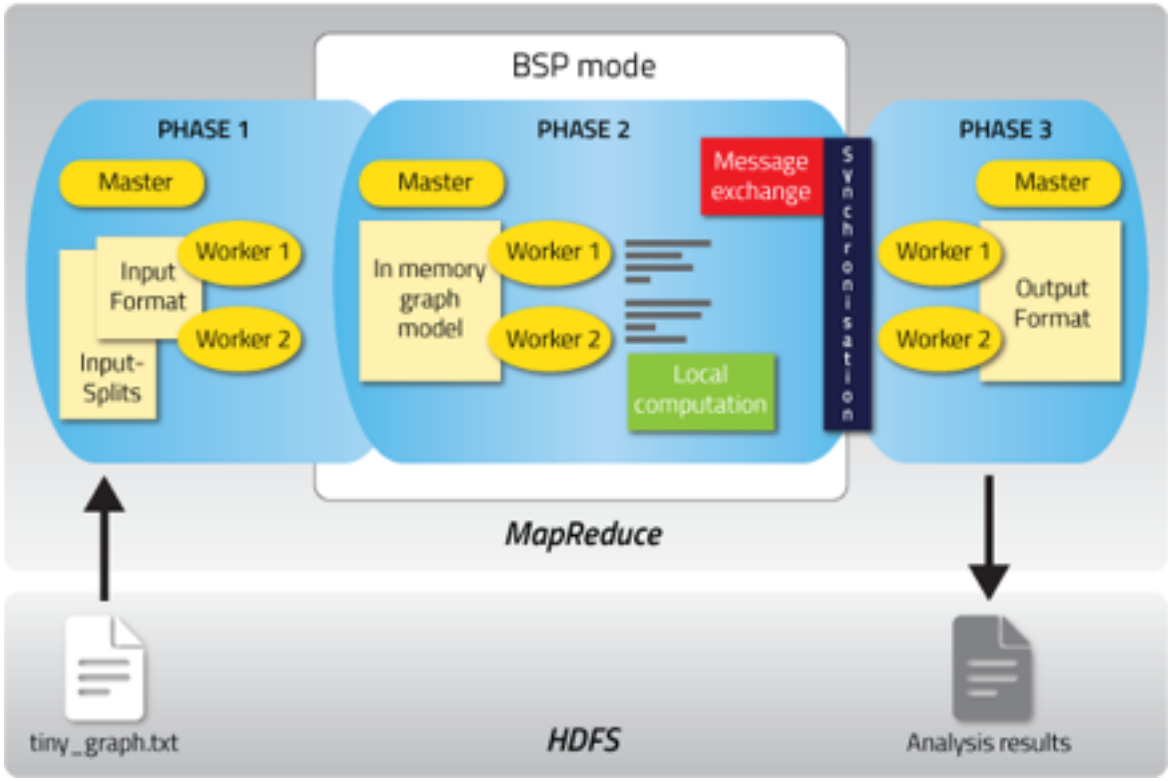
1) удаление ребер
2) удаление вершин
3) добавление вершин
4) добавление ребер

Giraph — архитектура

Реализация: map-only Hadoop job

- master
  - один активный master в каждый момент времени
  - разделение графа и распределение кусков по worker
  - контроль supersteps
- worker
  - загрузка графа из InputSplit
  - вычисления, относящиеся к “своему” куску графа
  - передача сообщений
- zookeeper
  - global application state

Giraph — выполнение программы

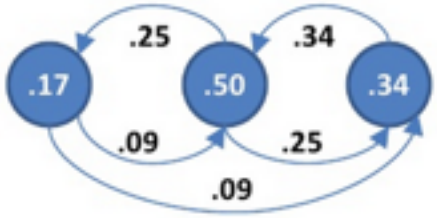


Giraph — отказоустойчивость

- master
  если один master “умирает”, автоматически включается другой
  (состояние хранится в zookeeper)
- worker
  если worker “умирает”, происходит “откат” к последней сохраненной итерации
- zookeeper
  сам по себе fault-tolerant

PageRank

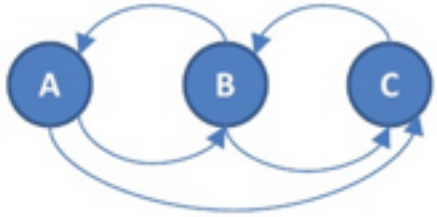
Superstep 0



Superstep 1

Superstep 2

Input graph



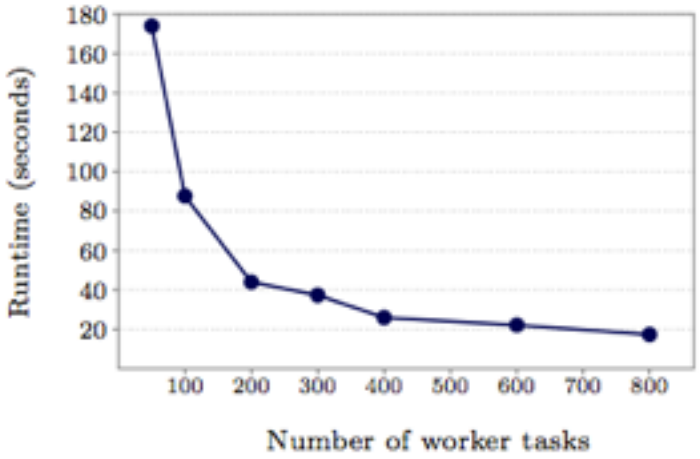
PageRank

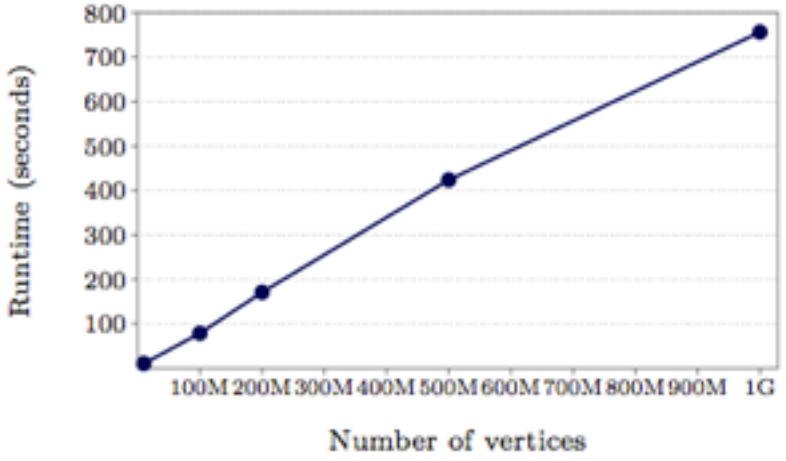
```
public class PageRankVertex extends Vertex<IntWritable, FloatWritable,
    NullWritable, FloatWritable> {
  /** Number of supersteps */
  public static final String SUPERSTEP_COUNT =
      "giraph.pageRank.superstepCount";

  @Override
  public void compute(Iterable<FloatWritable> messages) throws IOException {
    if (getSuperstep() >= 1) {
      float sum = 0;
      for (FloatWritable message : messages) {
        sum += message.get();
      }
      getValue().set((0.15f / getTotalNumVertices()) + 0.85f * sum);
    }

    if (getSuperstep() < getConf().getInt(SUPERSTEP_COUNT, 0)) {
      sendMessageToAllEdges(
          new FloatWritable(getValue().get() / getNumEdges()));
    } else {
      voteToHalt();
    }
  }
}
```

Shortest paths

```java
public class ShortestPathsVertex extends Vertex<LongWritable, DoubleWritable,
    DoubleWritable, DoubleWritable> {
  /** Source id. */
  public static final String SOURCE_ID = "giraph.shortestPathsBenchmark.sourceId";
  /** Default source id. */
  public static final long SOURCE_ID_DEFAULT = 1;

  private boolean isSource() {
    return getId().get() == getConf().getLong(SOURCE_ID, SOURCE_ID_DEFAULT);
  }

  @Override
  public void compute(Iterable<DoubleWritable> messages) throws IOException {
    if (getSuperstep() == 0) {
      setValue(new DoubleWritable(Double.MAX_VALUE));
    }

    double minDist = isSource() ? 0d : Double.MAX_VALUE;
    for (DoubleWritable message : messages) {
      minDist = Math.min(minDist, message.get());
    }

    if (minDist < getValue().get()) {
      setValue(new DoubleWritable(minDist));
      for (Edge<LongWritable, DoubleWritable> edge : getEdges()) {
        double distance = minDist + edge.getValue().get();
        sendMessage(edge.getTargetVertexId(),
            new DoubleWritable(distance));
      }
    }

    voteToHalt();
  }
}
```

Производительность





Кластер из 300 машин

Задача: дорожная сеть Бельгии

pregel.py — простая реализация модели Pregel
roadnet.py — реализация Vertex для нашей задачи и вспомогательные функции

Реализованные примеры:
- degree
- pagerank

Требуется реализовать:
- h-index (количество соседей, degree которых превышает их ранг)
- кратчайшие пути из Брюсселя в остальные города

Вопросы