# Beyond Hadoop: Pig and Giraph

Franco Maria Nardini

HPCLab@ISTI

ISTI – CNR, Pisa, Italy

```
/*
 * Licensed to the Apache Software Fo
 * or more contributor license agreem
 * distributed with this work for add
 * regarding copyright ownership.  Th
 * to you under the Apache License, V
 * "License"); you may not use this f
 * with the License.  You may obtain
 *
 *     http://www.apache.org/licenses
 *
 * Unless required by applicable law
 * distributed under the License is d
 * WITHOUT WARRANTIES OR CONDITIONS O
 * See the License for the specific l
 * limitations under the License.
 */
package com.hadoop.examples.anagrams;

import java.io.IOException;
import java.util.Arrays;

import org.apache.hadoop.io.LongWrita
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapRe
import org.apache.hadoop.mapred.Mappe
import org.apache.hadoop.mapred.Outpu
import org.apache.hadoop.mapred.Repor
/**
 * The Anagram mapper class gets a wo
 * letters in the word and writes its
 * Key : sorted word (letters in the
 * Value: the word itself as the valu
 * When the reducer runs then we can
 *
 * @author subbu iyer
 *
 */
public class AnagramMapper extends Ma
                Mapper<LongWritable,

        private Text sortedText = new
        private Text orginalText = ne


        public void map(LongWritable
                        OutputCollect
                        throws IOExce

                String word = value.t
                char[] wordChars = wo
                Arrays.sort(wordChars
                String sortedWord = n
                sortedText.set(sorted
                orginalText.set(word)
                outputCollector.colle
        }

}
```

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.hadoop.examples.anagrams;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;

public class AnagramJob {

        /**
         * @param args
         */
        public static void main(String[] args) throws Exception{
                JobConf conf = new JobConf(AnagramJob.class);
                conf.setJobName("anagramcount");

                conf.setOutputKeyClass(Text.class);
                conf.setOutputValueClass(Text.class);

                conf.setMapperClass(AnagramMapper.class);
                // conf.setCombinerClass(AnagramReducer.class);
                conf.setReducerClass(AnagramReducer.class);

                conf.setInputFormat(TextInputFormat.class);
                conf.setOutputFormat(TextOutputFormat.class);

                FileInputFormat.setInputPaths(conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(conf, new Path(args[1]));

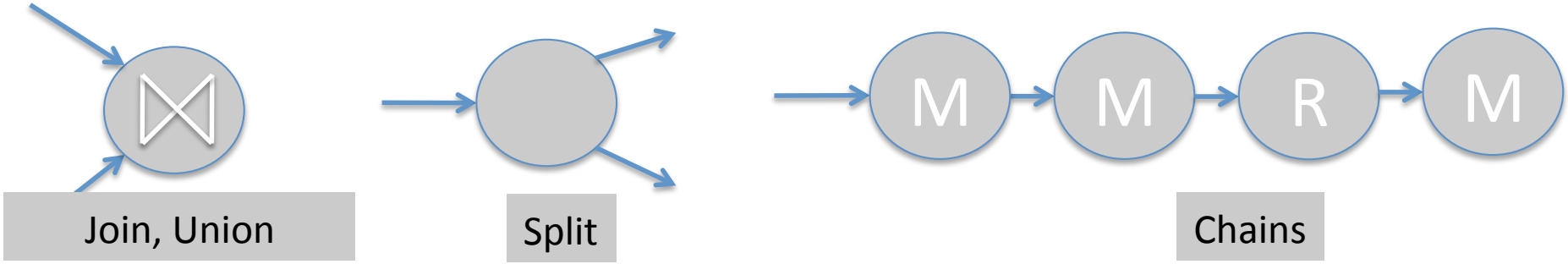
                JobClient.runJob(conf);

        }

}
```

# Map-Reduce Limitations

- One-input, Two-stage dataflow
  - That's it!!!
- Other flows constantly hacked in




- Need to program over and over (or to build library functions):
  - Projections
  - Filtering
  - Aggregates
  - Order By
  - Distinct

# Solution: *Pig Latin*

- Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. 2008. **Pig latin: a not-so-foreign language for data processing**. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08)*. ACM, New York, NY, USA, 1099-1110.

- It is a high-level declarative language (à la SQL) and a low level procedural programming (à la Map-Reduce)
- Example
  - ```
    urls = LOAD 'urls.txt' AS (url,category,pagerank);
    ok_urls = FILTER urls BY pagerank > 0.2;
    groups = GROUP good_urls BY category;
    big_groups = FILTER groups BY COUNT(ok_urls)>10^6
    o = FOREACH big_groups GENERATE category,
                                    AVG(good_urls.pagerank)
    ```
    ```

# Step-by-Step Procedure Control

Target users are entrenched procedural programmers

> The step-by-step method of creating a program in Pig is much cleaner and simpler to use than the single block method of SQL. It is easier to keep track of what your variables are, and where you are in the process of analyzing your data.

Jasmine Novak
*Engineer, Yahoo!*

> With the various interleaved clauses in SQL, it is difficult to know what is actually happening sequentially. With Pig, the data nesting and the temporary tables get abstracted away. Pig has fewer primitives than SQL does, but it's more powerful.

David Ciemiewicz
*Search Excellence, Yahoo!*

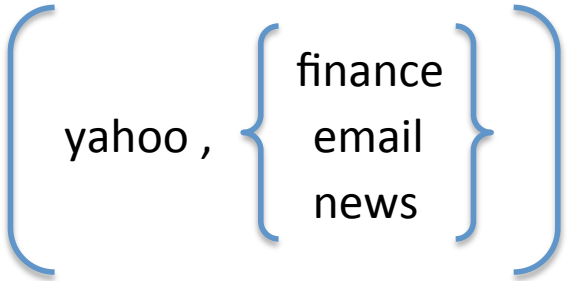
# Data Format

- None!
- The good news is that Pig is able to read text files.
- No need to import data into a DB-like application.
  Pig applications requirements are, often, the following:
  - Read-only data analysis workload (no transactional consistency guarantees)
  - Scan-centric (no point lookups)
  - Datasets are temporary (no curation needed)
- Even a fixed schema is not necessary as in Pig you can refer to dimensions in tuples via $i notation, e.g.,

```
ok_urls = FILTER urls BY $2 > 0.2;
```

# Nested Data Model

- Pig Latin has a fully-nestable data model with:
  - Atomic values, tuples, bags (lists), and maps




- More natural to programmers than flat tuples
- Avoids expensive joins

# Nested vs. Relational Data Models

- We have an inverted file. For each term we have a list of documentIDs with which positional information is associated. E.g.,

  *term1 -> doc1(1,4,5);doc2(2,3,5)*

- In Pig you represent it as a

  ```
  Map<termID,Set<Map<documentID,Set<positions>>>>
  ```

- In Relational Data Models the same dataset should be represented in Normal Form as:

  ```
  term_info(termID, termString, …)
  document_info(termID, documentID, …)
  position_info(termID, documentID, position)
  ```

# Data Types

- *Atom*: An atom contains a simple atomic value such as a string or a number, e.g., `'alice'`
- *Tuple*: A tuple is a sequence of fields, each of which can be any of the data types, e.g., `('alice', 'lakers')`
- *Bag*: A bag is a collection of tuples with possible duplicates. The schema of the constituent tuples is flexible, i.e., not all tuples in a bag need to have the same number and type of fields, e.g., `{('alice', 'lakers'), ('alice', ('ipod', 'apple'))}`
- *Map*: A map is a collection of data items, where each item has an associated key through which it can be looked up. As with bags, the schema of the constituent data items is flexible, i.e., all the data items in the map need not be of the same type. However, the keys are requested to be data atoms, mainly for efficiency of lookups. Example `['fan of' -> {('alice'), ('lakers')}, 'age' -> 20]`

# Expressions

$$\texttt{t} = \left(\text{`alice'}, \left\{ \begin{array}{c} (\text{`lakers'}, 1) \\ (\text{`iPod'}, 2) \end{array} \right\}, [\text{`age'} \rightarrow 20]\right)$$

Let fields of tuple `t` be called `f1, f2, f3`

| Expression Type | Example | Value for t |
|---|---|---|
| Constant | `'bob'` | |
| Field by position | `$0` | |
| Field by name | `f3` | |
| Projection | `f2.$0` | |
| Map Lookup | `f3#'age'` | |
| Function Evaluation | `SUM(f2.$1)` | |
| Conditional Expression | `f3#'age'>18? 'adult':'minor'` | |
| Flattening | `FLATTEN(f2)` | |

# User Defined Functions (UDFs)

- To accomodate specialized data processing tasks, Pig Latin has extensive support for user-defined functions (UDFs).
- All aspects of processing in Pig Latin including grouping, filtering, joining, and per-tuple processing can be customized through the use of UDFs.
- Originally, only Java UDFs were supported, now Javascript and Python UDFs are supported as well.

# An Example of UDF

- `exp_q = FOREACH queries GENERATE myudfs.UPPER(qString);`

```
1  package myudfs;
2  import java.io.IOException;
3  import org.apache.pig.EvalFunc;
4  import org.apache.pig.data.Tuple;
5  import org.apache.pig.impl.util.WrappedIOException;
6
7  public class UPPER extends EvalFunc<String>
8  {
9      public String exec(Tuple input) throws IOException {
10         if (input == null || input.size() == 0)
11             return null;
12         try{
13             String str = (String)input.get(0);
14             return str.toUpperCase();
15         }catch(Exception e){
16             throw WrappedIOException.wrap("Caught exception processing input row ", e);
17         }
18     }
19 }
```

# Specifying Input Data: **LOAD**

```
queries = LOAD 'query_log.txt'
          USING myLoad()
          AS    (userId,
                 queryString,
                 timestamp);
```

# Per-tuple Processing: **FOREACH**

- `exp_q = FOREACH queries GENERATE Exp(qString);`

**GENERATE** accepts Pig expressions.

$t = \left(\text{'alice'}, \left\{\begin{matrix}(\text{'lakers'}, 1)\\(\text{'iPod'}, 2)\end{matrix}\right\}, [\text{'age'} \rightarrow 20]\right)$

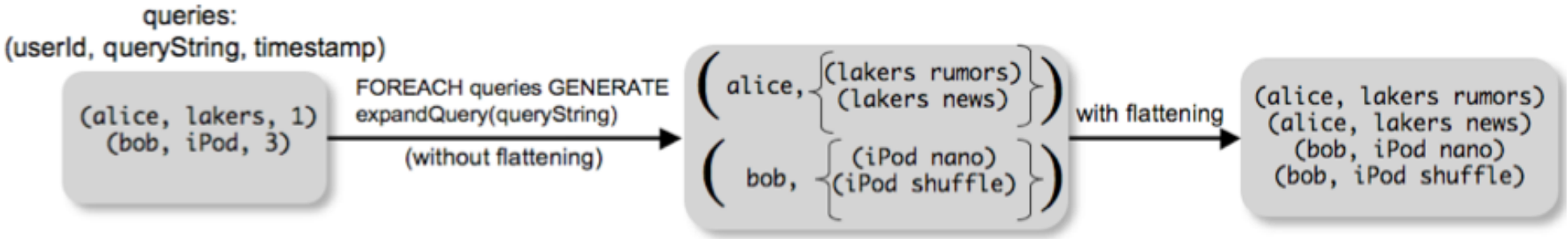
Let fields of tuple `t` be called `f1, f2, f3`

| Expression Type | Example | Value for `t` |
|---|---|---|
| Constant | `'bob'` | Independent of `t` |
| Field by position | `$0` | `'alice'` |
| Field by name | `f3` | `['age' → 20]` |
| Projection | `f2.$0` | `{('lakers') ('iPod')}` |
| Map Lookup | `f3#'age'` | 20 |
| Function Evaluation | `SUM(f2.$1)` | `1 + 2 = 3` |
| Conditional Expression | `f3#'age'>18? 'adult':'minor'` | `'adult'` |
| Flattening | `FLATTEN(f2)` | `'lakers', 1 'iPod', 2` |

# Tuple flattening: **FLATTEN**

- Flattening operates on bags by extracting the fields of the tuples in the bag, and making them fields of the tuple being output by GENERATE, thus removing one level of nesting. For example, the output of the following command is shown as the second step in the figure below.

```
exp_q = FOREACH    queries
        GENERATE   userID, FLATTEN(Exp(qString));
```



# Discarding Unwanted Data: **FILTER**

- While scanning a dataset we might need to filter out lines not matching a given boolean expressi
- For exa
queries
`real_q`
- Using U
`real_q`

Filtering conditions in Pig Latin can involve a combination of expressions, comparison operators such as **==**, **eq**, **!=**, **neq**, and the logical connectors **AND**, **OR**, and **NOT**.

# Getting Related Data Together: **GROUP**

- If we want to merge together rows from a dataset

```
grouped
query_
                                                    ;
```

To group all tuples of a data set together (e.g., to compute the overall total revenue), one uses the syntax

**GROUP revenue ALL**;

# **GROUP** Results

- Warning. Results of **GROUP** operations are non-intuitive.

```
B = GROUP A BY age;

DESCRIBE B;
B: {group: int, A: {name: chararray,age: int,gpa: float}}

ILLUSTRATE B;
etc …
-------------------------------------------------------------------------
| B     | group: int | A: bag({name: chararray,age: int,gpa: float}) |
-------------------------------------------------------------------------
|       | 18         | {(John, 18, 4.0), (Joe, 18, 3.8)}             |
|       | 20         | {(Bill, 20, 3.9)}                             |
-------------------------------------------------------------------------

DUMP B;
(18,{(John,18,4.0F),(Joe,18,3.8F)})
(19,{(Mary,19,3.8F)})
(20,{(Bill,20,3.9F)})
```

# Merging Datasets: **JOIN**

- Given two datasets, normal equi-join operations can be carried out by the **JOIN** command.

```
join_result =
    JOIN result BY qString,
         revenue BY qString;
```

# Stream Processing: **STREAM**

- Sends data to an external script or program.

```
A = LOAD 'data';
B = STREAM A THROUGH 'stream.pl -n 5';
```

# Asking for Output: **STORE**

- Materialization of results is obtained through the Pig statement **STORE**:

```
STORE query_revenues INTO
'output' USING myStore();
```

# Other Commands

- **`UNION`**: Returns the union of two or more bags.
- **`CROSS`**: Returns the cross product of two or more bags.
- **`ORDER`**: Orders a bag by the specified field(s).
- **`DISTINCT`**: Eliminates duplicate tuples in a bag. This command is just a shortcut for grouping the bag by all fields, and then projecting out the groups.

# Nested Operations

```
grouped_revenue = GROUP revenue BY qString;
query_revenues =
    FOREACH grouped_revenue{
        top_slot = FILTER revenue BY adSlot eq 'top';
        GENERATE queryString,
                 SUM(top_slot.amount),
                 SUM(revenue.amount);
    };
```

# PageRank on Pig

- Input format:

```
www.A.com    1    {(www.B.com),(www.C.com),(www.D.com),(www.E.com)}
www.B.com    1    {(www.D.com),(www.E.com)}
www.C.com    1    {(www.D.com)}
www.D.com    1    {(www.B.com)}
www.E.com    1    {(www.A.com)}
www.F.com    1    {(www.B.com),(www.C.com)}
```

- We use python scripts with Pig embedded. Useful for implementing iterative Pig scripts.

# PageRank.py

```
#!/usr/bin/python
from org.apache.pig.scripting import *

P = Pig.compile("""
- PR(A) = (1-d) + d (PR(T1)/C(T1) + … + PR(Tn)/C(Tn))

previous_pagerank =
    LOAD '$docs_in'
    USING PigStorage('\t')
    AS ( url: chararray, pagerank: float, links:{ link: ( url: chararray ) } );

outbound_pagerank =
    FOREACH previous_pagerank
    GENERATE
        pagerank / COUNT ( links ) AS pagerank,
        FLATTEN ( links ) AS to_url;

new_pagerank =
    FOREACH
        ( COGROUP outbound_pagerank BY to_url, previous_pagerank BY url INNER )
    GENERATE
        group AS url,
        ( 1 - $d ) + $d * SUM ( outbound_pagerank.pagerank ) AS pagerank,
        FLATTEN ( previous_pagerank.links ) AS links;

STORE new_pagerank
    INTO '$docs_out'
    USING PigStorage('\t');
""")
```

# LinkedIn's DataFu

DataFu is a collection of user-defined functions for working with large-scale data in Hadoop and Pig. This library was born out of the need for a stable, well-tested library of UDFs for data mining and statistics. It is used at LinkedIn in many of our off-line workflows for data derived products like "People You May Know" and "Skills & Endorsements". It contains functions for:

- PageRank
- Quantiles (median), variance, etc.
- Sessionization
- Convenience bag functions (e.g., set operations, enumerating bags, etc)
- Convenience utility functions (e.g., assertions, easier writing of EvalFuncs)

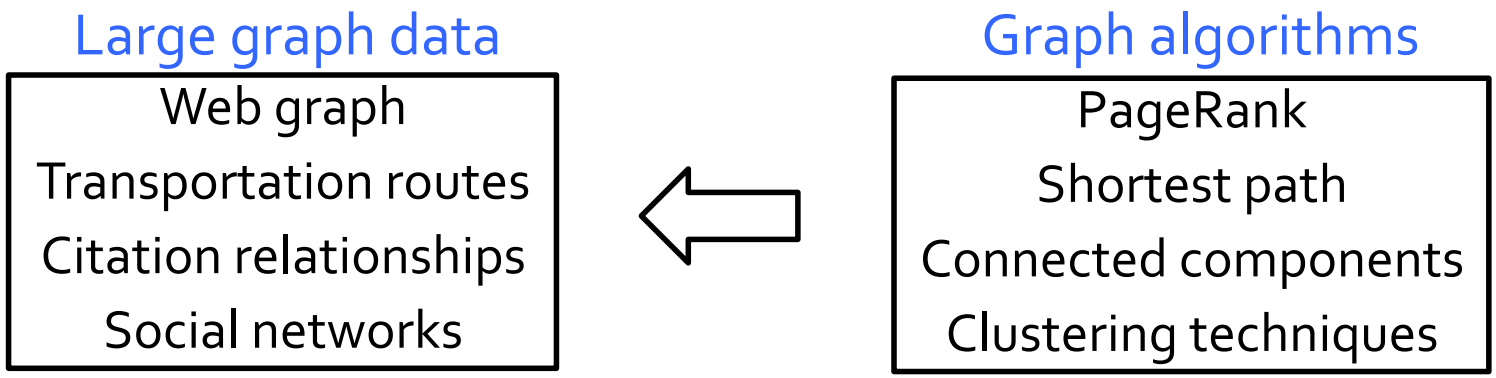
- https://github.com/linkedin/datafu

# Is Map-Reduce Enough?

- Map-Reduce is a functional-like easy-to-understand paradigm.
- Complex programs are not easily portable in Map-Reduce.
- Other programming models exists.

# Is Map-Reduce Enough?

- Many practical computing problems concern large graphs



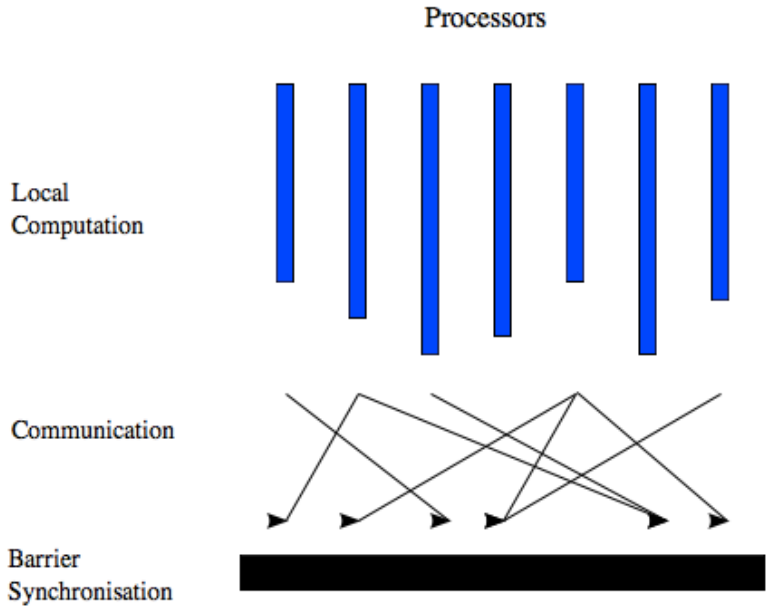

- Map-Reduce is ill-suited for graph processing
  - Many iterations are needed for parallel graph processing
  - Materializations of intermediate results at every Map-Reduce iteration harm performance

# Bulk Synchronous Parallel (BSP) Model

- Developed during 80s by Leslie G. Valiant (2010 Turing Award winner)
- Published in 1990:
  - Leslie G. Valiant, **A bridging model for parallel computation**, *Communications of the ACM*, Volume 33 Issue 8, Aug. 1990
- Is a very simple, yet powerful, bridging model.
  - A bridging model "is intended neither as a hardware nor a programming model but something in between".
  - It serves a purpose similar to the Parallel Random Access Machine (PRAM) model.
  - BSP differs from PRAM by not taking communication and synchronization for granted.

# The BSP Computer

A BSP computer consists of **processors** connected by a communication network.
Each processor has a fast local memory, and may follow different threads of computation.
A BSP computation proceeds in a series of global supersteps.
A superstep consists of three components:




- **Local** (concurrent) **computation**: Several computations take place on every participating processor Each process only uses values stored in the local memory of the processor. The computations are independent in the sense that they occur asynchronously of all the others.
- **Communication**: The processes exchange data between themselves. This exchange takes the form of one-sided put and get calls, rather than two-sided send and receive calls.
- **Barrier synchronization**: When a process reaches this point (the barrier), it waits until all other processes have finished their communication actions.

The computation and communication actions do not have to be ordered in time. The barrier synchronization concludes the superstep: it has the function of ensuring that all one-sided communications are properly concluded. This global synchronization is not needed in models based on two-sided communication, since these synchronize processes implicitly.

# Cost of Communications

- The BSP model considers communication actions *en masse*.
  - All messages have fixed size
  - Communications happen at the beginning and at the end of a superstep
- The maximum number of incoming or outgoing messages for a superstep is denoted by *h*.
- The ability of a communication network to deliver data is captured by a parameter *g*, defined such that it takes time *hg* for a processor to deliver *h* messages of size 1.
  - A message of length *m* obviously takes longer to send than a message of size 1. However, the BSP model does not make a distinction between a message length of *m* or *m* messages of length 1. In either case the cost is said to be *mgh*.

# The Cost of a BSP Algorithm

- The cost of a superstep is determined as the sum of three terms:
  - the cost of the longest running local computation *w*
  - the cost of global communication between the processors *hg*
  - the cost of the barrier synchronization at the end of the superstep *l*
- Hence, the total cost of a BSP program is given by

$$W + Hg + Sl = \sum_{s=1}^{S} w_s + g \sum_{s=1}^{S} h_s + Sl$$

# Why BSP?

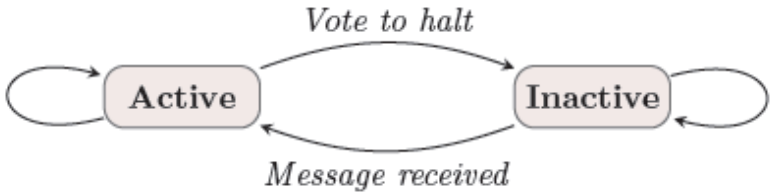
- Google's Pregel is based on the BSP model:
  - G. Malewicz, M. H. Austern, A. J.C Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. 2010. **Pregel: a system for large-scale graph processing**. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10).*
- Pregel is suitable for computations on graph.
  - In Pregel vertexes of a graph are abstracted as processors.
  - **Vertex-centric** approach.
- Pregel computations consist of a sequence of iterations, called supersteps.

# What's in a Pregel's Superstep?

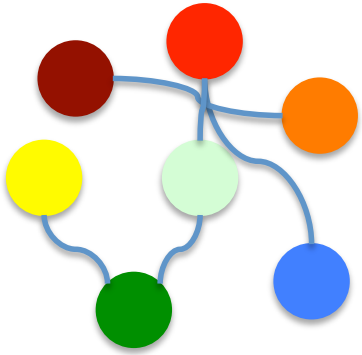
- During a superstep the framework invokes a user-defined function for each vertex, conceptually in parallel.
  - The function specifies behavior at a single vertex *V* and a single superstep *S*.
  - It can read messages sent to *V* in superstep *S − 1*, send messages to other vertices that will be received at superstep *S + 1*, and modify the state of V and its outgoing edges.
  - Messages are typically sent along outgoing edges, but a message may be sent to any vertex whose identifier is known.

# Recap: Model of Computation

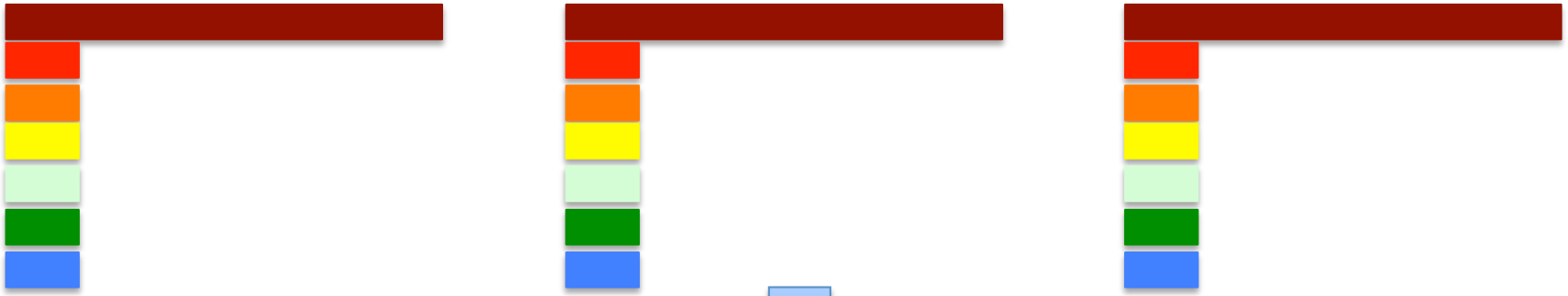
- Superstep: the vertices compute in parallel
  - Each vertex
    - Receives messages sent in the previous superstep
    - Executes the same user-defined function
    - Modifies its value or that of its outgoing edges
    - Sends messages to other vertices (to be received in the next superstep)
    - Mutates the topology of the graph
    - Votes to halt if it has no further work to do
  - Termination condition
    - All vertices are simultaneously inactive
    - There are no messages in transit



# Why Vertex-centric for Massive Graphs?



One node per processor

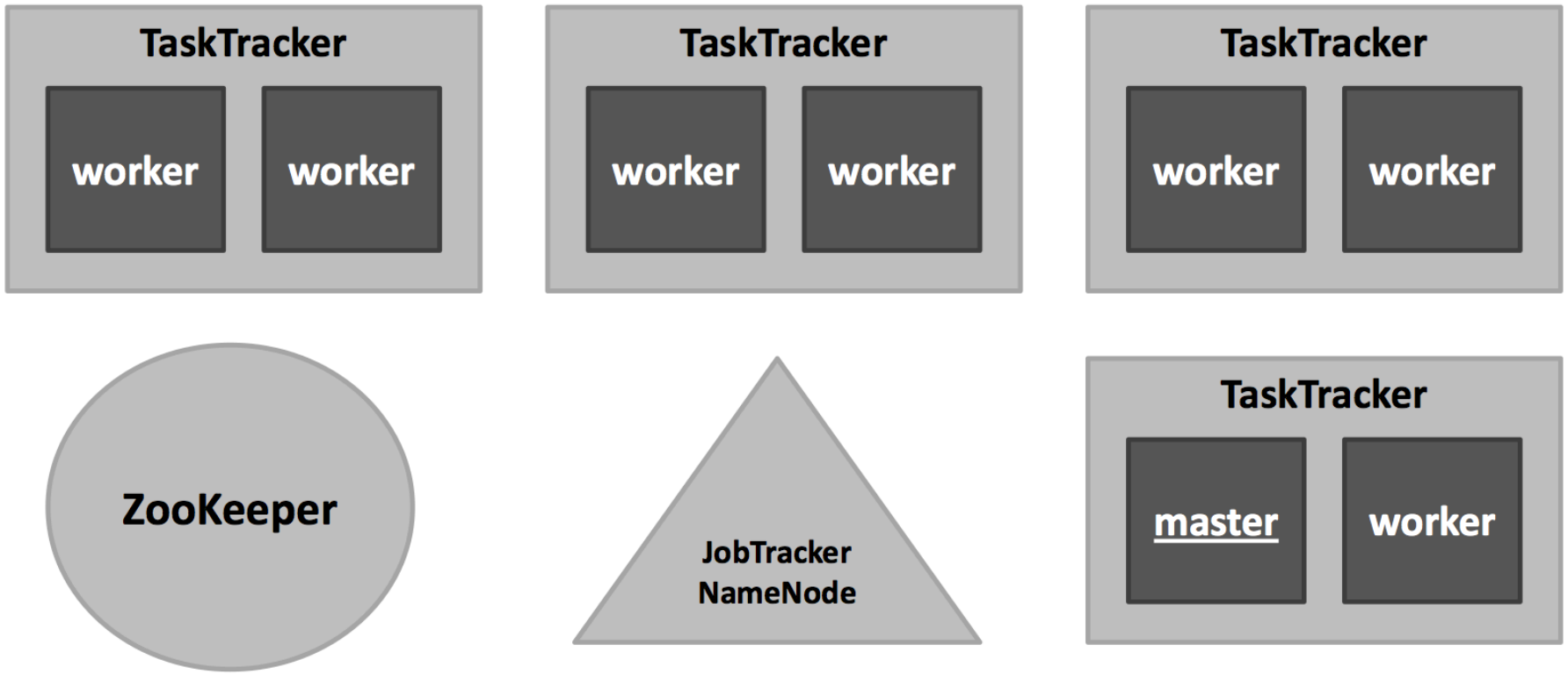


Only two processors

# Apache Giraph

- Warning!
  - Giraph is currently in Apache incubator
  - Modifications are continuous:
    - Some of them might cause a complete program rewrite…
      - … it happened to me! :)

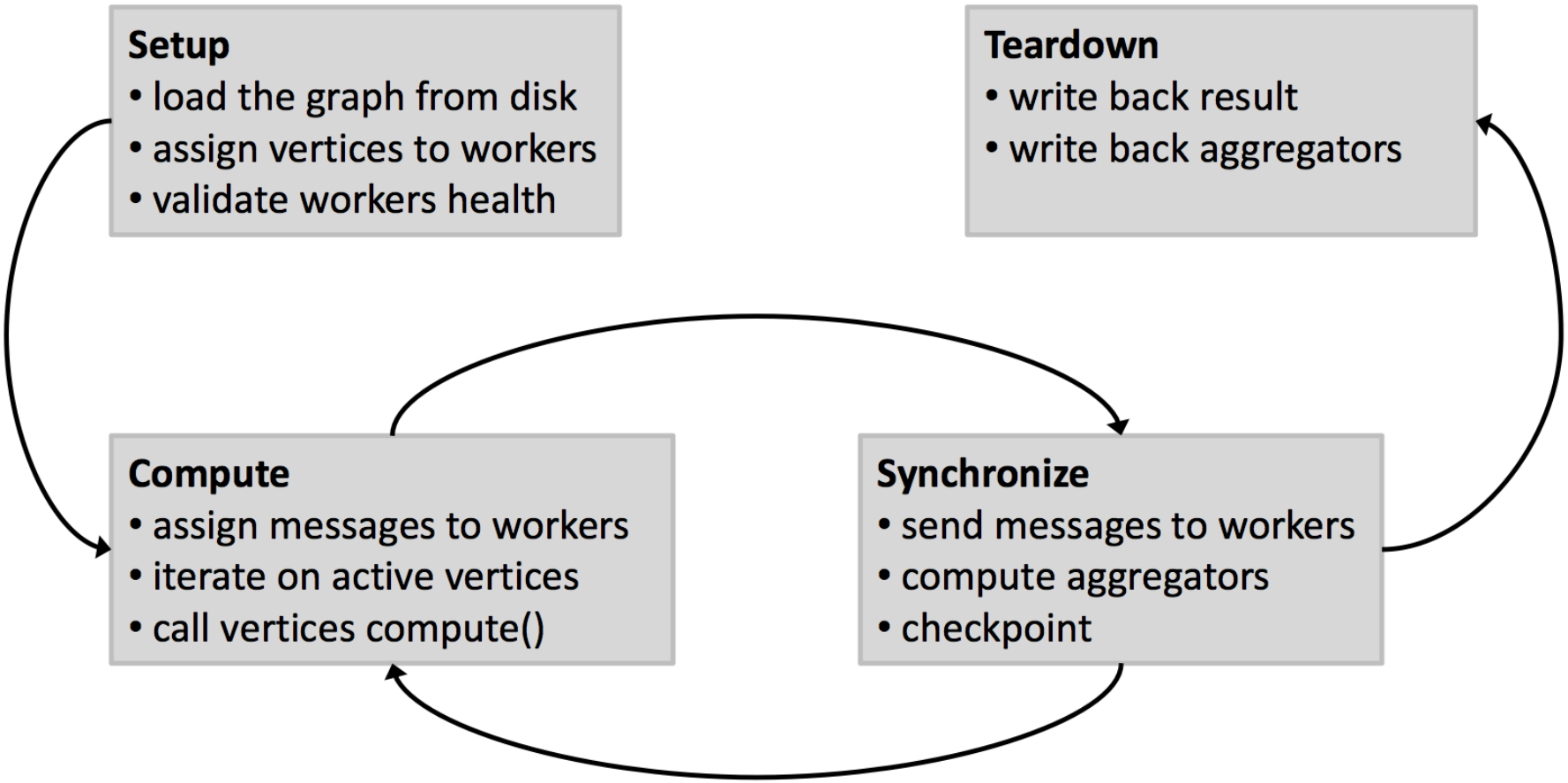
# Giraph Framework



# Tasks Assignment

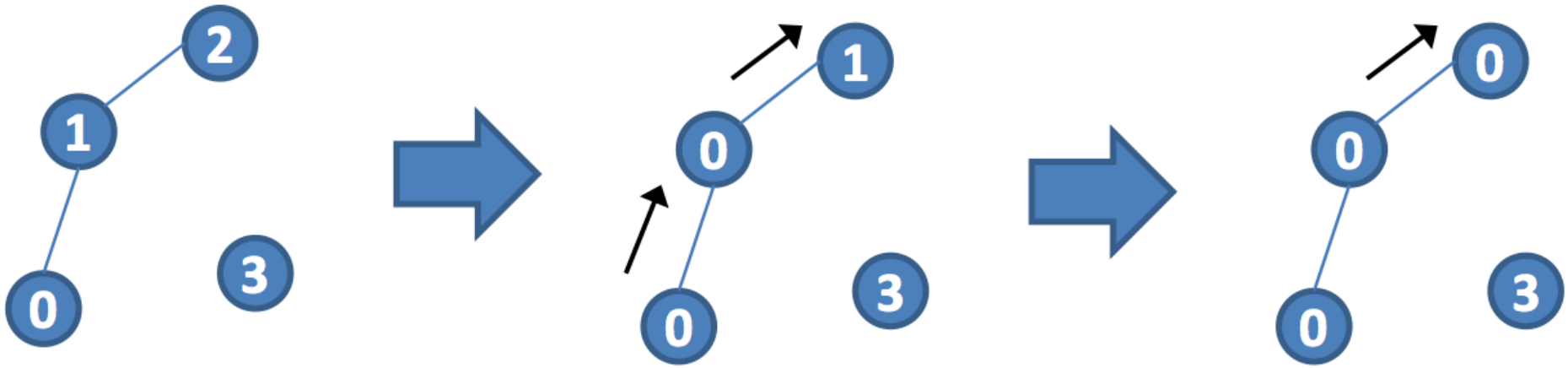
- **ZooKeeper**: responsible for **computation state**
  - partition/worker mapping
  - global state: #superstep
  - checkpoint paths, aggregator values, statistics
- **Master**: responsible for **coordination**
  - assigns partitions to workers
  - coordinates synchronization
  - requests checkpoints
  - aggregates aggregator values
  - collects health statuses
- **Worker**: responsible for **vertices**
  - invokes active vertices compute() function
  - sends, receives and assigns messages
  - computes local aggregation values

# Anatomy of an Execution



# Giraph Example: Connected Components of an Undirected Graph

- *algorithm*: propagate smallest vertex label to neighbors until convergence




- in the end, all vertices of a component will have the same label

# Create a Custom Vertex

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.giraph.examples;

import org.apache.giraph.graph.IntIntNullIntVertex;
import org.apache.hadoop.io.IntWritable;

import java.io.IOException;
```

# Create a Custom Vertex

```
/**
 * Implementation of the HCC algorithm that identifies connected components and
 * assigns each vertex its "component identifier" (the smallest vertex id
 * in the component)
 *
 * The idea behind the algorithm is very simple: propagate the smallest
 * vertex id along the edges to all vertices of a connected component. The
 * number of supersteps necessary is equal to the length of the maximum
 * diameter of all components + 1
 *
 * The original Hadoop-based variant of this algorithm was proposed by Kang,
 * Charalampos, Tsourakakis and Faloutsos in
 * "PEGASUS: Mining Peta-Scale Graphs", 2010
 *
 * http://www.cs.cmu.edu/~ukang/papers/PegasusKAIS.pdf
 */
@Algorithm(
    name = "Connected components",
    description = "Finds connected components of the graph"
)
```

# Create a Custom Vertex

```
public class ConnectedComponentsVertex extends IntIntNullIntVertex {
  /**
   * Propagates the smallest vertex id to all neighbors. Will always choose to
   * halt and only reactivate if a smaller id has been sent to it.
   *
   * @param messages Iterator of messages from the previous superstep.
   * @throws IOException
   */
  @Override
  public void compute(Iterable<IntWritable> messages) throws IOException {
    int currentComponent = getValue().get();

    // First superstep is special, because we can simply look at the neighbors
    if (getSuperstep() == 0) {
      for (IntWritable neighbor : getNeighbors()) {
        if (neighbor.get() < currentComponent) {
          currentComponent = neighbor.get();
        }
      }
      // Only need to send value if it is not the own id
      if (currentComponent != getValue().get()) {
        setValue(new IntWritable(currentComponent));
        for (IntWritable neighbor : getNeighbors()) {
          if (neighbor.get() > currentComponent) {
            sendMessage(new IntWritable(neighbor.get()), getValue());
          }
        }
      }

      voteToHalt();
      return;
    }

    boolean changed = false;
    // did we get a smaller id ?
    for (IntWritable message : messages) {
      int candidateComponent = message.get();
      if (candidateComponent < currentComponent) {
        currentComponent = candidateComponent;
        changed = true;
      }
    }

    // propagate new component id to the neighbors
    if (changed) {
      setValue(new IntWritable(currentComponent));
      sendMessageToAllEdges(getValue());
    }
    voteToHalt();
  }
}
```

# Additional Stuff

- To read a custom data format we need to create a custom input format class extending an input format class available in Giraph, e.g.,
  - public class ConnectedComponentsInputFormat extends TextVertexInputFormat<IntWritable, IntWritable, NullWritable, IntWritable>
- To output using a custom data format we need to create a custom output format class extending an output format class available in Giraph, e.g.,
  - public class VertexWithComponentTextOutputFormat extends TextVertexOutputFormat<IntWritable, IntWritable, NullWritable>

# How To Run a Giraph Job

```
fabriziosilvestri@macsilvestri ~/Documents/giraph/trunk/target $ hadoop jar giraph-0.2-SNAPSHOT-for-hadoop-0.20.203.0-jar-with-dependencies.jar
org.apache.giraph.GiraphRunner
usage: org.apache.giraph.GiraphRunner [-aw <arg>] [-c <arg>] [-ca <arg>]
       [-cf <arg>] [-h] [-if <arg>] [-ip <arg>] [-la] [-mc <arg>] [-of
       <arg>] [-op <arg>] [-q] [-w <arg>] [-wc <arg>]
 -aw,--aggregatorWriter <arg>   AggregatorWriter class
 -c,--combiner <arg>            VertexCombiner class
 -ca,--customArguments <arg>    provide custom arguments for the job
                                configuration in the form:
                                <param1>=<value1>,<param2>=<value2> etc.
 -cf,--cacheFile <arg>          Files for distributed cache
 -h,--help                      Help
 -if,--inputFormat <arg>        Graph inputformat
 -ip,--inputPath <arg>          Graph input path
 -la,--listAlgorithms           List supported algorithms
 -mc,--masterCompute <arg>      MasterCompute class
 -of,--outputFormat <arg>       Graph outputformat
 -op,--outputPath <arg>         Graph output path
 -q,--quiet                     Quiet output
 -w,--workers <arg>             Number of workers
 -wc,--workerContext <arg>      WorkerContext class
```

# A PageRank Vertex

```
public class SimplePageRankVertex extends LongDoubleFloatDoubleVertex {
  /** Number of supersteps for this test */
  public static final int MAX_SUPERSTEPS = 30;
  /** Logger */
  private static final Logger LOG =
      Logger.getLogger(SimplePageRankVertex.class);
  /** Sum aggregator name */
  private static String SUM_AGG = "sum";
  /** Min aggregator name */
  private static String MIN_AGG = "min";
  /** Max aggregator name */
  private static String MAX_AGG = "max";

  @Override
  public void compute(Iterable<DoubleWritable> messages) {
    if (getSuperstep() >= 1) {
      double sum = 0;
      for (DoubleWritable message : messages) {
        sum += message.get();
      }
      DoubleWritable vertexValue =
          new DoubleWritable((0.15f / getTotalNumVertices()) + 0.85f * sum);
      setValue(vertexValue);
      aggregate(MAX_AGG, vertexValue);
      aggregate(MIN_AGG, vertexValue);
      aggregate(SUM_AGG, new LongWritable(1));
      LOG.info(getId() + ": PageRank=" + vertexValue +
          " max=" + getAggregatedValue(MAX_AGG) +
          " min=" + getAggregatedValue(MIN_AGG));
    }

    if (getSuperstep() < MAX_SUPERSTEPS) {
      long edges = getNumEdges();
      sendMessageToAllEdges(
          new DoubleWritable(getValue().get() / edges));
    } else {
      voteToHalt();
    }
  }
```


Web
Search
Parallel
Mining
Frequent
Caching
System
Engines
Query
Closed
Retrieval
Pattern
Itemsets
Scheduling
Patterns
Design
Exploiting
Information
Efficient
Data
Grids
Queries
Recommender
Constrained
Enhance
Document
parallelism
Logs
Fast
framework
engine
Modern
Results
Distributed
stream
Discovery
Indexes
Image
Large
Identifiers
Grid
Assigning
Constraint-Based
computing
Content-based
Memory
flow
Sites
Exploitation
Incremental
preserving
applications
Historical
Identifier
Collaborative
perspectives
Databases
Optimize
mapping
Computer
Criteria
Challenges
New
Know
Job
Area
environment
Behavioral
Hybrid
Model
Fulltext
CoPhIR
Muskel
java
Mine
Farms
Interactive
Property
Load
Prefetching
preservation
autonomic
Detection
Comparison
annotations
Ownership
dynamically
computations
jobs
geodesic
Intervention
Balance
Multi
Highly
Personalization
Joint
protection
aspects
macro
GCM
Performance
based
Past
expandable
Condensed
Assignment
Core
Harnessing
components
Grid-enabled
Peer-to-Peer
Capacity
Online
automatic
java/jini
Algorithm
Including
Architectures
Partitioning
heterogeneous
privacy
heuristics
Pal
map
Future
Load-driven
management
Usage
Documents
Knowledge
distance
batch
Problem
Dynamic
Extending
large-scale
Building
Boosting
Collection
Impact
Exploratory
Launch-time
Clustering
Everything
Neighbors
Overlays
Providers
interpreter
Index
Compressibility