

Frameworks for Distributed Computing

Alessio Guerrieri

University of Trento, Italy

2011/11/09

Growth of data size

Data sets

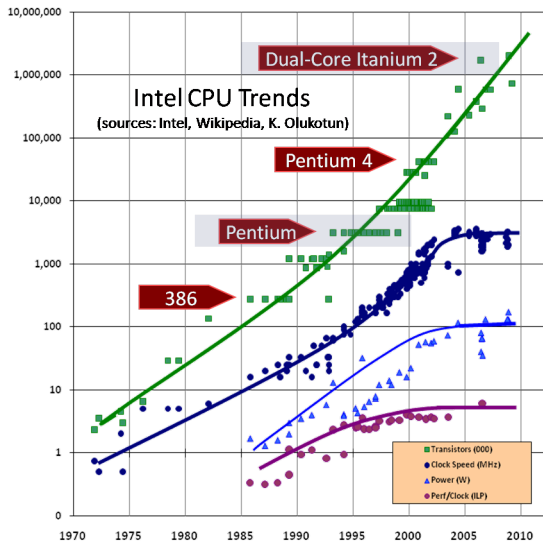
We need to analyze bigger and bigger datasets:

- Web datasets: web topology
- Network datasets: routing efficiency analysis, tcp-ip logs
- Biological datasets: ontologies, genome sequencing data, protein folding patterns

Cpu speed of new computers is not growing fast enough.

Sequential algorithms are not a viable answer

Growth of cpu speed



Valid approaches

Parallel algorithm

Use the power of all cores in the cpu.

Distributed algorithms!

Pros

- Cheaper
- Scalable
- More reliable

Cons

- More difficult to write
- Different challenges

Parallel computing

Parallel vs Distributed

A parallel algorithm can have:

- A global clock
- Shared memory

Are there “easy way” to write parallel computing programs?

Hands-on approach

Manually start threads, distribute data, etc. . .

Parallel computing: OpenMP

Easy way to parallelize a sequential program in shared memory:

```
#pragma omp parallel for private(w) reduction(+:sum)
for(i = 0; i < N; i++){
    w = i*i;
    sum = sum + w*a[i];
}
```

- Can be tested as a sequential program
- Easy to understand
- Can be applied “mostly” on for cycles.

Frameworks for distributed algorithms

- Define a **Programming Model** that is inherently parallelizable.
- Give an interface to the programmer to implement the algorithm
- Create a **framework** that runs the program and gives fault-tolerance and scalability out of the box

Target

We want a programming model that is:

- Transparent: the algorithm designer should not think about “processes” and “messages”
- Expressive: it should be possible to solve a wide range of problems
- Intuitive

We want a framework that guarantees:

- Failure resistance
- Scalability
- Locality

Distributed File System

We assume the existence of a Distributed File System across our network.

Overview of Google's DFS:

- Data is replicated across the network.
- A “namenode” contains information about which nodes contain which file

Caveat:

- We want to ensure **locality**
- If the DFS and the Framework do not collaborate locality is unattainable

Scenarios

Big companies

Google, Yahoo, Microsoft have huge clusters of machines and huge data sets to analyze

A framework helps the developers use the cluster without expertise in distributed computing

Cloud computing

The framework can be offered as a service by a provider.

See Amazon EC2

History

MapReduce is a programming model and an associated implementation for processing and generating large data sets.

- Developed at Google
- First paper published: 2004
- Created to help google developers to analyze huge datasets with google's clusters

The MapReduce programming model was inspired by functional programming.

Map and Reduce in functional programming

Map function

Map: $(\alpha list, \alpha \rightarrow \beta) \rightarrow \beta list$

Example: $\text{Map}([1,2,3], \text{sqr}) = [1,4,9]$

Reduce (or fold) function

Reduce: $(\alpha list, \beta, (\beta, \alpha) \rightarrow \beta) \rightarrow \beta$

Example: $\text{Reduce}([1,2,3], 0, +) = 6$

Compute Sum of squares of a list:

$\text{Reduce}(\text{Map}(\text{list}, \text{sqr}), 0, +)$

Map and Reduce in MapReduce

The user must define two functions:

Mapper

Mapper: $(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$

Reducer

Reducer: $(k_2, \text{list}(v_2)) \rightarrow \text{list}(v_3)$

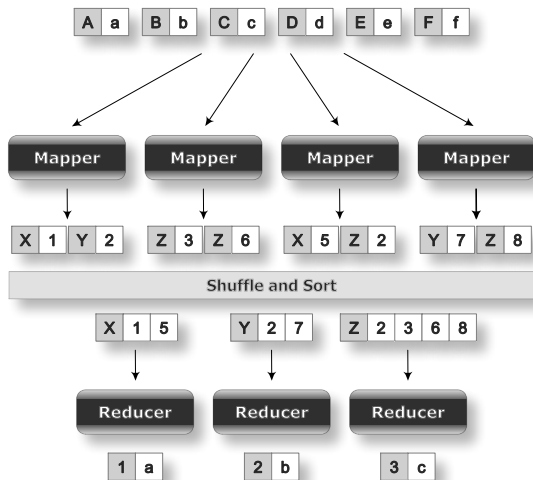
On each pair in the input data set we call the Mapper. It will output a list of *intermediate* pairs.

For each key in the intermediate pairs, we group all the values assigned to it and we call the Reducer.

Each call of the Mapper function can be executed concurrently.

After all Mapper have finished, each call of the Reducer function can be executed concurrently.

MapReduce flow (simplified)



Example: Frequencies

Input

A set of documents in a DFS

Output

The number of occurrences of each word in the set of documents.

Idea: we can work on each document concurrently and then “reduce” the results for each word.

Example: Frequencies

Algorithm 1: Mapper

Input: String filename, String content

foreach word w **in** content **do**

\lfloor EMITINTERMEDIATE($w,1$);

Algorithm 2: Reducer

Input: String key, Iterator values

result $\leftarrow 0$;

foreach v **in** values **do**

\lfloor result \leftarrow result + v ;

EMIT (key,result);

Possible optimization

- Creating a pair for each occurrence of each word looks like a waste of time

Algorithm 3: Optimized-mapper

Input: String filename, String content

$H \leftarrow$ new HashTable;

foreach word w **in** content **do**

$H[w] \leftarrow H[w] + 1$

foreach word w **in** H **do**

 EMITINTERMEDIATE($w, H[w]$)

Combiner and Partitioner

Combiner

After the mapper, each node sorts the output of its mapper.

The combiner is applied before the global sort.

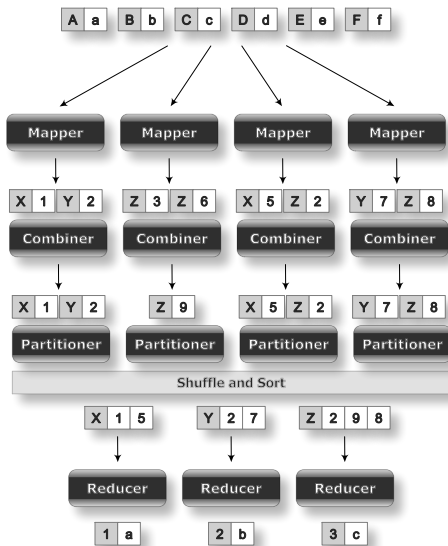
Combiner : $(k_2, \text{list}(v_2)) \rightarrow (k_2, \text{list}(v_2))$

See frequency estimator.

Partitioner

Before the global sort, it is possible to specify how to partition the data between the nodes.

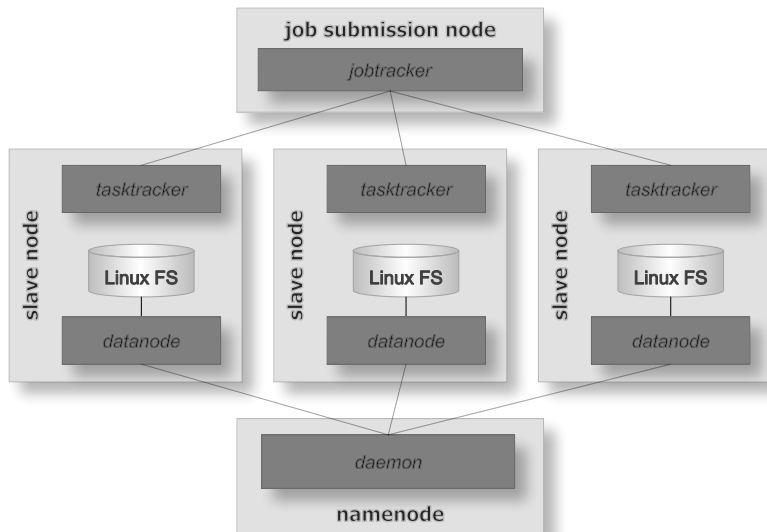
Complete MapReduce flow



Framework

- Google's framework is not available
- Open source alternative: Apache's Hadoop
- Offered as service in Amazon Elastic Computing

Overview



Details

- Library in Java
- Driver program must be written in Java
- Possibility to use external programs as mapper and reducers
- Relatively stable

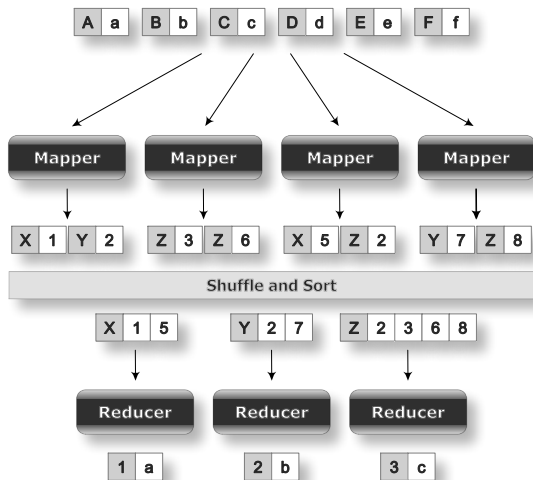
Inverted Index

Problem

We are given a set of webpages $P = \{P_1, \dots, P_n\}$.

We want to know for each word in the dataset, in which documents it appears (and how many times)

MapReduce flow (simplified)



Baseline solution

Algorithm 4: Mapper

Input: String filename, String content

$H \leftarrow \text{new HashTable};$

foreach word w **in** content **do**

$H[w] \leftarrow H[w] + 1$

foreach word w **in** H **do**

$\text{EMITINTERMEDIATE}(w, (\text{filename}, H[w]))$

Algorithm 5: Reducer

Input: String key, Iterator values

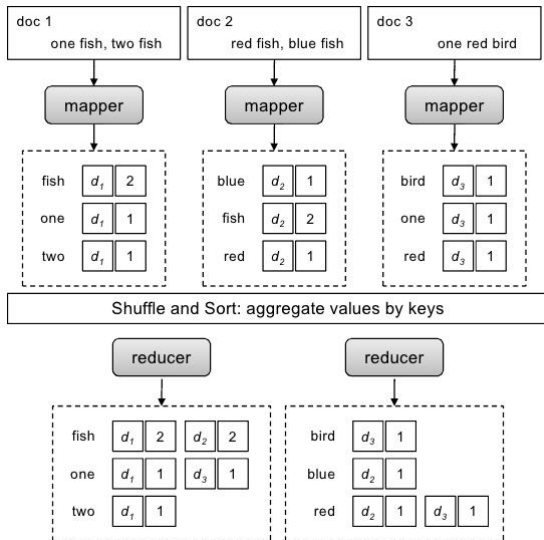
result $\leftarrow [];$

foreach v **in** values **do**

$\text{result.add}(v);$

result.sortbyfreq();

Simple run



K-Means clustering

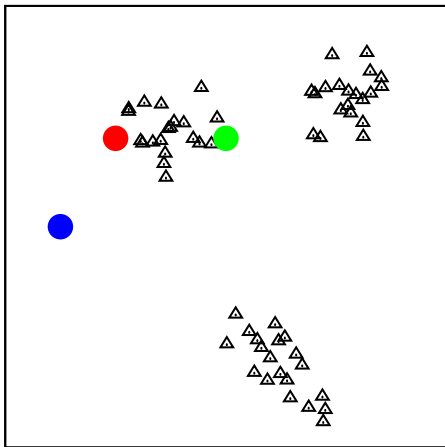
Problem definition

Given a set of points, partition them to minimize the within-cluster sum of squares

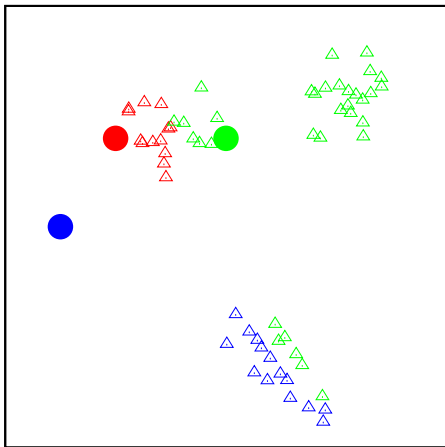
Standard algorithm (Lloyd 1982)

- Choose K centroids at random
- While changes:
 - ▶ Assign each point to closest centroid
 - ▶ Move each centroid to the average of the points assigned to it

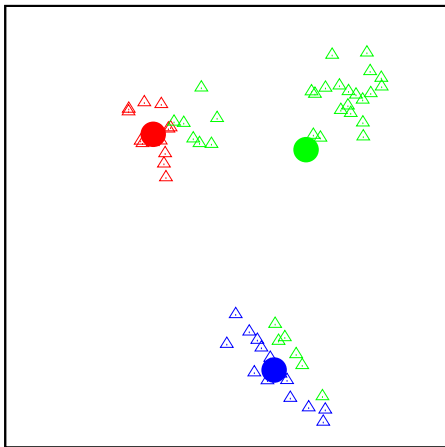
K-Means run



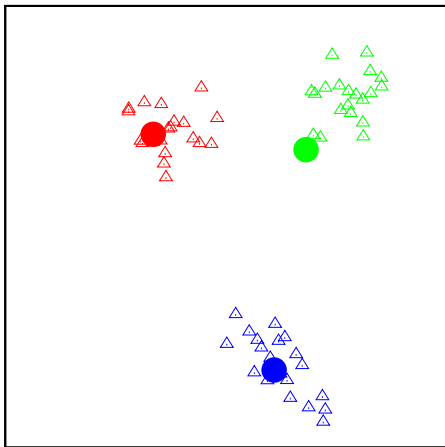
K-Means run



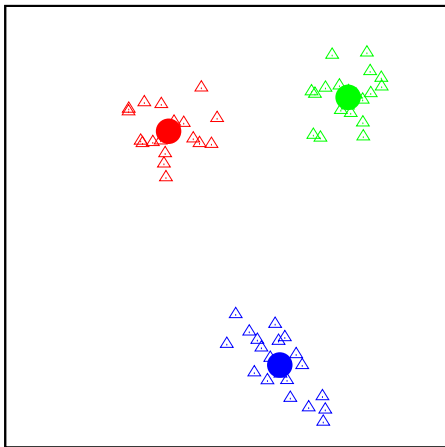
K-Means run



K-Means run



K-Means run



MapReduce Algorithm

- Impossible to solve in a single job
- Each MapReduce job corresponds to a KMeans step
- We need a “driver” program.

Driver

- Let C' be the set of starting centroids
- Do:
 - ▶ Set $C = C'$
 - ▶ Send C to all nodes
 - ▶ Run MapReduce algorithm
 - ▶ Extract new centroids C' from output
- While $C \neq C'$

MapReduce Algorithm

Algorithm 6: Mapper

Input: Int id, Point p

$k \leftarrow \mathbf{argmin}_i(\mathit{dist}(p, C_i))$;

EMITINTERMEDIATE(k,p);

Algorithm 7: Reducer

Input: Int id, Iterator values

result $\leftarrow \mathbf{avg}(\mathit{values})$;

EMIT (id,result);

Extensions over Hadoop MapReduce

- Give higher level abstraction
- No need to think in terms of maps and reduces
- Easier to use

Sawzall

- Developed by Google
- Used for simple queries
- Target: a distributed awk.

Example

Problem

Given a dataset of real numbers, compute how many numbers there are, the sum and the sum of squares

Code

```
count: table sum of int;
total: table sum of float;
sum_of_squares: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
emit sum_of_squares <- x * x;
```

Other Aggregators

```
c: table collection of string;  
s: table sample(100) of string;  
m: table maximum(10) of string weight length: int;  
q: table quantile(101) of response_in_ms: int;  
t: table top(10) of language: string;
```

Pig-Latin

- Developed and used at Yahoo.
- “designed to fit in a sweet spot between the declarative style of SQL, and the low-level, procedural style of map-reduce.”

Example

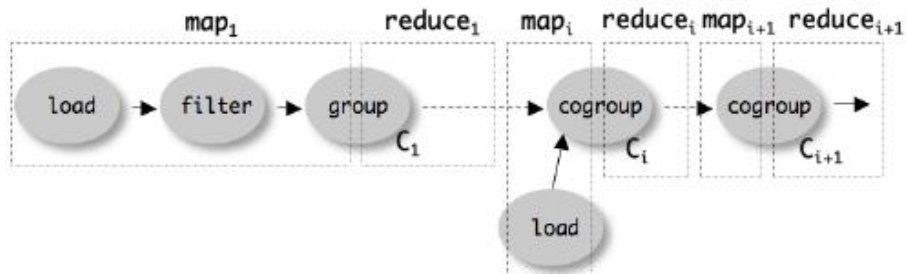
SQL query

```
SELECT category, AVG(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 1000000
```

Pig-Latin program

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)>1000000;
output = FOREACH big_groups GENERATE
category, AVG(good_urls.pagerank);
```

Notes

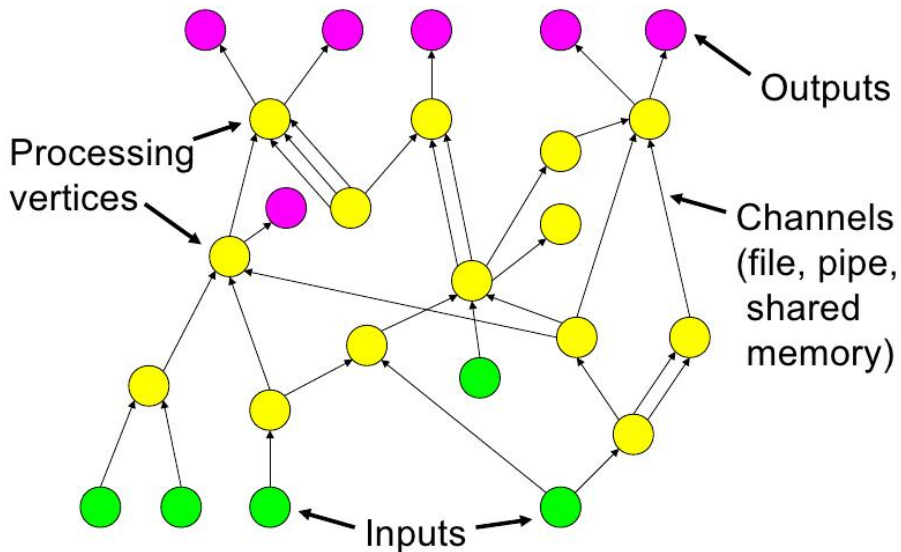


- The program is decomposed in different MapReduce jobs
- Full support for User Defined Function
- Has a debugging tool

Main idea

- Developed by Microsoft
- In Map-Reduce all Maps must be completed before the Reduce
- Dryad generalize the concept
- The computation is modelled as a Direct Acyclic Graph

Structure



Example: Gravitational lens effect

Input

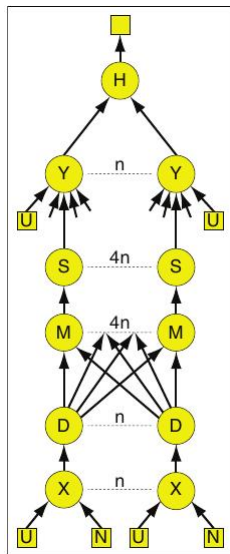
- Table U: (starId, color)
- Table N: (starId, neighbourId)

Problem

Finding the pair of stars with similar colors.

Example: Gravitational lens effect

- X: join on starId
- D: distribute the data according to neighbourId
- M: merge inputs
- S: sort according to neighbourId
- Y: join neighbourId with starId of U
- H: filter distinct pairs



Implementation

- Nodes are described as classes in C++
- The graph must be manually described in the code.
- The properties of the channels can be specified in the graph creation program

Framework characteristics

- Vertices are spawned across the network
- If a vertex fail, it can be restarted at another node
- There may be different copies of the same vertex running at the same time
- Messages across channels are buffered
- Graph can be changed at runtime to optimize performance

Dryad-Linq

- Dryad is a low-level framework
- We need extensions to simplify usage
- Dryad-Linq is specialized on queries

Linq

Part of the .NET framework. Adds support to queries in all .NET languages

Linq example

```
int [] numbers = new int [7] { 0, 1, 2, 3, 4, 5, 6 };  
var numQuery = from num in numbers  
                where (num % 2) == 0  
                select num;  
foreach (int num in numQuery) {  
    Console.WriteLine(num);  
}
```

DryadLinq overview

DryadLinq processes the Linq code and:

- creates the vertex code
- generates the graph
- passes the graph and the vertices to Dryad

Orleans

- Developed at Microsoft
- Aimed at integration Cloud+Client
- Actor-based computing
- The basic unit of computation is the **grain**
- Grains collaborate in a **transaction**
- Each concurrent transaction uses a separate **activation** of each grain
- When a transaction has finished, the state of each grain is updated.

Details

Grains

- The developer defines a Grain Interface
- Many grains are created and their state is saved in permanent storage
- Whenever a process (maybe on a client) starts a transaction:
 - ▶ each grain in the transaction is instantiated in a activation
 - ▶ each activation is private to the transaction
 - ▶ when the transaction has terminated the state of each activation is written to storage
 - ▶ if conflicting changes across different transaction, changes are reconciled automatically

Example

Grain interface

```
public interface ISimpleGrain : IGrain {  
    AsyncValue<int> A { get; }  
    AsyncValue<int> B { get; }  
    AsyncCompletion SetA(int a);  
    AsyncCompletion SetB(int a);  
    AsyncValue<int> GetA();  
    AsyncValue<int> GetB();  
    AsyncValue<int> GetAxB();  
}
```

Example

Promises

```
AsyncValue<int> intPromise = grain.GetA();  
intPromise.ContinueWith((int a) => {  
    Console.WriteLine("Result: " + a.ToString());  
}, (Exception exc) => {  
    Console.WriteLine("Error: " + exc.Message);  
}).Ignore();
```

Example

Promises advanced

```
AsyncCompletion setA = grain.SetA(3);
AsyncCompletion setB = grain.SetB(4);
AsyncCompletion set=AsyncCompletion.Join(setA, setB);
AsyncValue<int> get=
    set.ContinueWith( () => { return grain.GetAxB(); });

AsyncCompletion result = get.ContinueWith(
    (int x) => { Console.WriteLine("Result:_" + x.ToString()); },
    (Exception e) => { Console.WriteLine("Error:_" + e.Message)}
);

try {
    result.Wait();
} catch(Exception exc) {
    Console.WriteLine("Error:_" + exc.Message);
}
```

Motivations

Many big data sets are graphs:

- World wide web graphs
- Social network topology
- Protein folding graphs

Pregel

- Developed at Google.
- Based on the Bulk Synchronous Parallel model
- Algorithm based on a sequence of supersteps
- Each vertex in the graph is seen as a different process
- At each superstep, each vertex:
 - ▶ Receives the list of messages sent to it during the last superstep
 - ▶ Computes its new state
 - ▶ Sends messages to other vertices
 - ▶ Decides if it votes to halt
- When all nodes vote to halt, the algorithm stops.

Framework

- Each process take control of a part of the graph
- The vertices are divided between the nodes
- The order of computation inside each superstep is not important
- Can be built over MapReduce or other frameworks

SSSP

Single source shortest path

Given a graph and a source vertex s , compute the distance from s to all other vertices.

Centralized solution

Run a breadth-first search.

Pregel solution

```
class SSSPVertex{
    int dist=inf;

    void compute(Collection<Integer> Messages){
        int dist2=min(Messages);
        if(dist2<dist){
            dist=dist2;
            for(Vertex v:this.getNeighbours())
                sendMessage(v, dist+1);
        }else{
            voteToHalt();
        }
    }
}
```

Available implementation

Both based on Apache Hadoop

GoldenOrb

- Developed by an independent company, sponsored by Ravel (Austin TX)
- Launched June 2011

Apache Giraph

- Developed in Apache Incubator

```
public class MyVertex extends Vertex<IntWritable, IntWritable, IntMessage>{
    ...
    public void compute(Collection<IntMessage> messages) {
        ...
        sendMessage(new IntMessage(dest, new IntWritable(42)));
    }
}
```