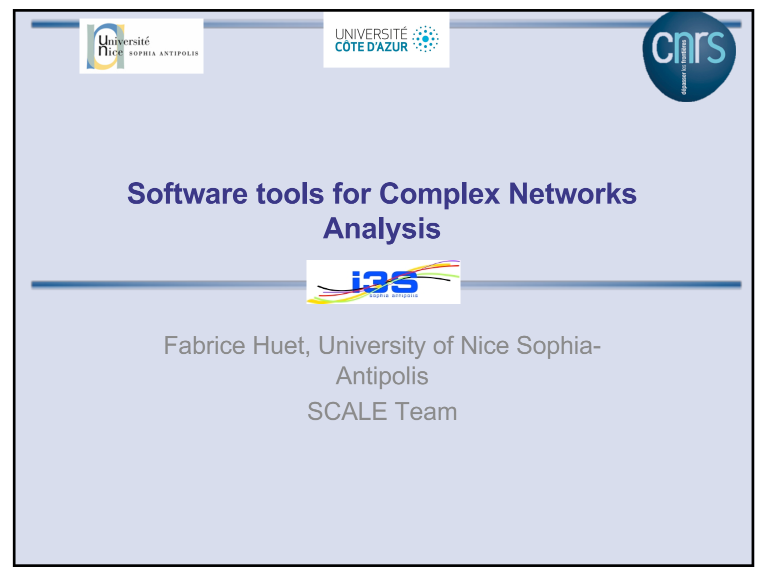
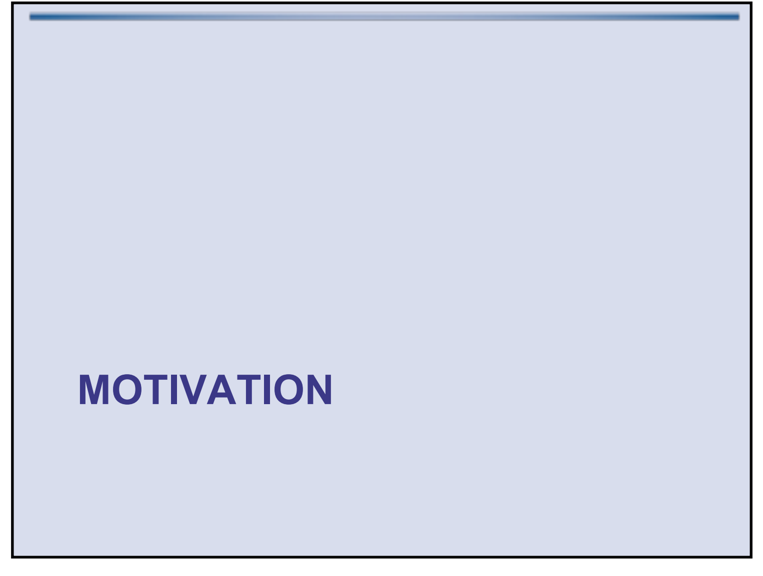
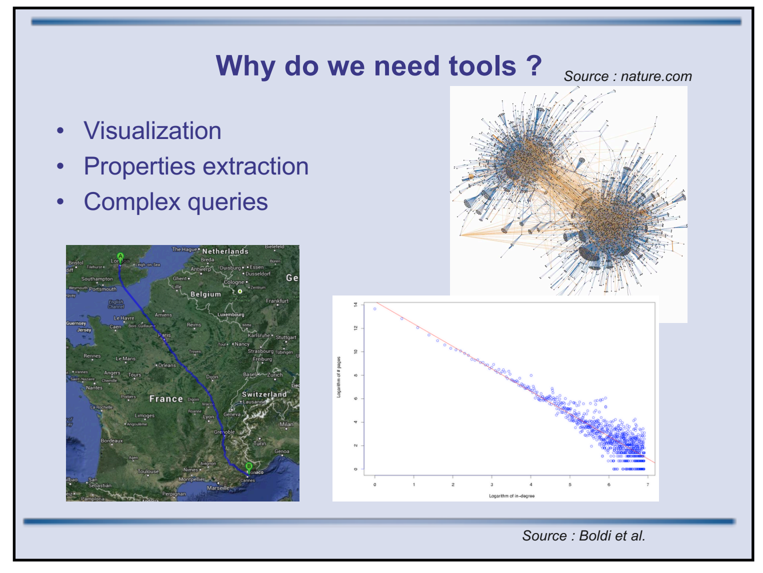
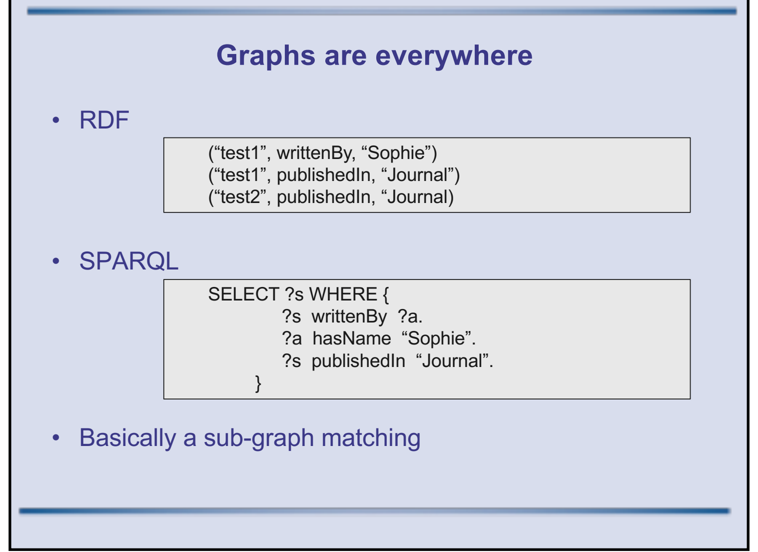

Université Nice Sophia Antipolis
UNIVERSITÉ CÔTE D'AZUR
cnrs
Software tools for Complex Networks Analysis
i3S
Fabrice Huet, University of Nice Sophia-Antipolis
SCALE Team


MOTIVATION


Why do we need tools ?
Source : nature.com
Visualization
Properties extraction
Complex queries
Source : Boldi et al.


Graphs are everywhere
RDF
("test1", writtenBy, "Sophie")
("test1", publishedIn, "Journal")
("test2", publishedIn, "Journal)
SPARQL
SELECT ?s WHERE {
?s writtenBy ?a.
?a hasName "Sophie".
?s publishedIn "Journal".
}
Basically a sub-graph matching

Why are graphs different ?

- Graphs can be large
  - Facebook : 720M users, 69B friends in 2011
    - 1.4 billions vertices, 1 trillion edges (2017)
  - Twitter : 537M accounts, 23.95B links in 2012
- Low memory cost per vertex
  - 1 ID, 1 pointer/edge
- Low computation per vertex
- Graphs are not memory friendly
  - Random jumps to memory
- They are not hardware friendly!

Lots of frameworks

- Really lots of them
  - Matlab, NetworkX, GraphChi, Hadoop, Twister, Piccolo, Maiter, Pregel, Giraph, Hama, GraphLab, Pegasus, Snap, Neo4J, Gephi, Tulip, any DBMS,…
- Why so many ?
  - Not one size fits all
  - Different computational models
  - Different architecture

Possible taxonomy

- Generic vs Specialized
  - Hadoop vs GraphChi (or Giraph, GraphX…)
- Shared vs Distributed Memory
  - GraphChi vs Pregel
- Synchronous vs Asynchronous
  - Giraph vs Maiter
- Single vs Multi threaded
  - NetworkX vs GraphiChi

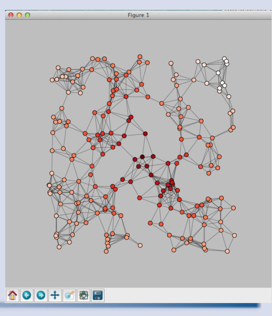
NETWORKX

Overview

- A Python package for complex network analysis
- Simple API
- Very flexible
  - Can attach any data to vertices and edges
  - Supports visualization
- Graphs generators
- http://networkx.github.io/

Dependencies

- Supports Python 2.7 (preferred) or 3.0
- If drawing support required
  - Numpy (http://www.numpy.org/)
  - Mathplotlib (http://matplotlib.org/)
  - Graphivz (http://graphviz.org/)



Examples

- Creating an empty graph

```
>>> import networkx as nx
>>> G=nx.Graph()
```

- Adding nodes

```
>>> G.add_node(1)
>>> G.add_nodes_from([2,3])
```

- Adding edges

```
>>> G.add_edge(2,3)
>>> G.add_edges_from([(1,2),(1,3)])
```

Examples (2)

- Graph generators

```
>>> K_5=nx.complete_graph(5)
>>> K_3_5=nx.complete_bipartite_graph(3,5)
```

- Stochastic graph generators

```
>>> er=nx.erdos_renyi_graph(100,0.15)
>>> ws=nx.watts_strogatz_graph(30,3,0.1)
>>> ba=nx.barabasi_albert_graph(100,5)
>>> red=nx.random_lobster(100,0.9,0.9)
```

- Reading from files

```
>>> mygraph=nx.read_gml("path.to.file")
```

Examples (3)

- Graph analysis

```
>>> nx.connected_components(G)
```

```
>>> nx.degree(G)
```

```
>>> pr=nx.pagerank(G,alpha=0.9)
```

- Graph drawing

```
>>> import matplotlib.pyplot as plt
>>> nx.draw(G)
>>> plt.show()
```

NetworkX - Conclusion

- Easy to use
  - Very good for prototyping/testing
- Centralized
  - Limited scalability
- Efficiency
  - Memory overhead

PROGRAMING MODELS FOR SCALABLE GRAPH PROCESSING

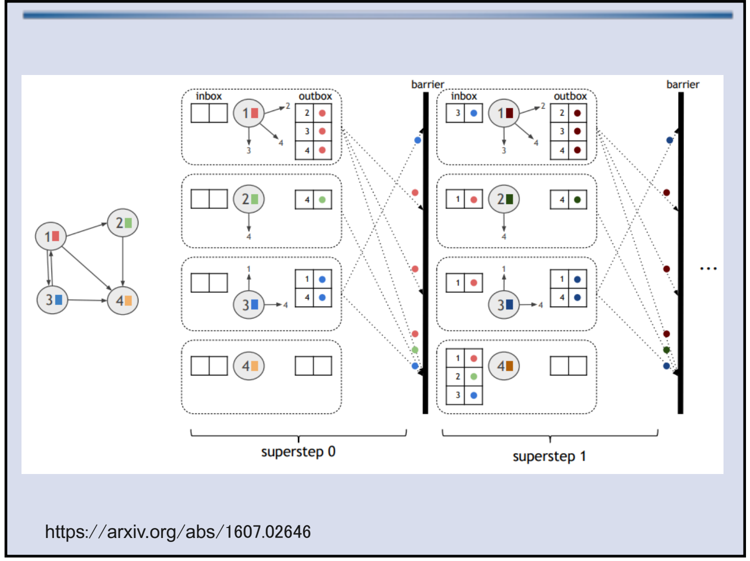
15

Overview

- Edges and vertices can have values
  - *Some state, weight…*
- Values propagate along edges
  - From source to destination vertex
- Vertices values are (can be) computed using incoming values
- Source :
  - **High-Level Programming Abstractions for Distributed Graph Processing"** by Vasiliki Kalavri, Vladimir Vlassov, and Seif Haridi.

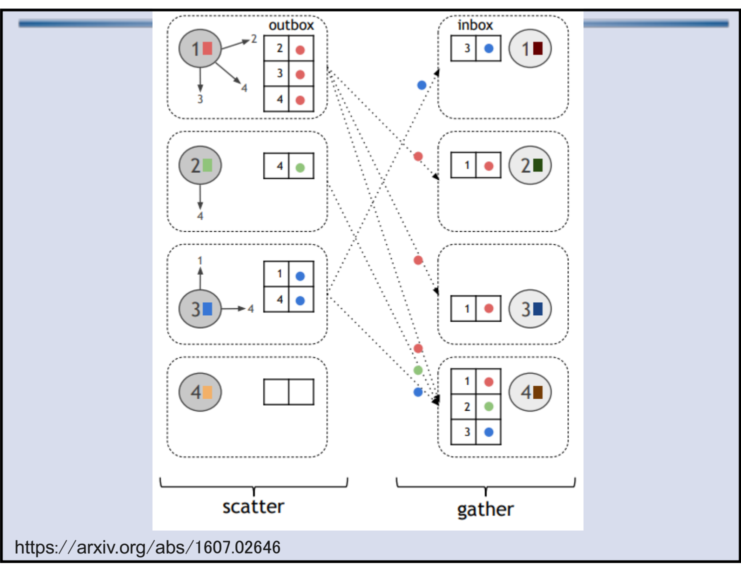
Vertex Centric model (”think like a vertex”)

- Input :
  - Directed graph
  - A function to execute on each vertex (aka Vertex Function)
- Execution model
  - At step *i*
    - Receive values from ingoing edges sent at *i-1*
    - Compute local state
    - Push new values
- Introduced in Pregel



Gather Scatter

- Iteration based
- Two operations to implement
  - Gather : receive values
  - Scatter : send values
- Very similar to vertex centric
  - Except message sending/receiving occurs in the same step
- Read/Write synchronization is done with inbox/outbox
  - No write during gather, no read during scatter



The GAS Model

- Gather, Sum, Apply, Scatter
- Try to address work imbalance in power-law graph
  - Mainly low-degree vertices, few high-degree
- Gather phase
  - A user defined function is applied to each edge of a vertex in parallel
- Sum phase
  - Apply an associative and commutative function to processed edges
- Apply
  - Compute new vertex state
- Scatter
  - User defined function on edges in paralle

sum
apply
gather
scatter

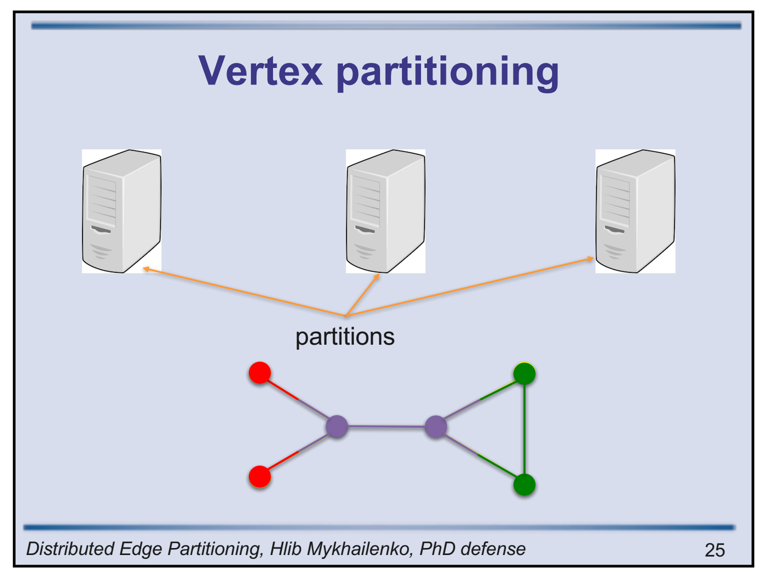
https://arxiv.org/abs/1607.02646

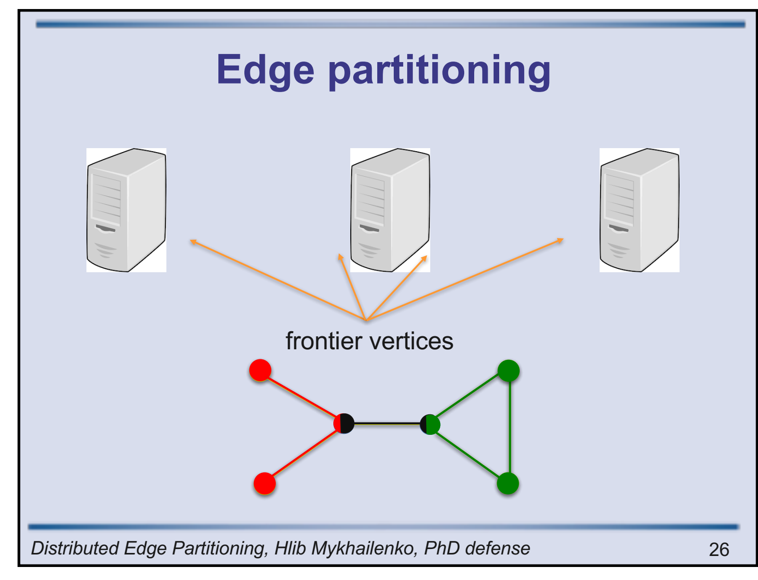
GRAPH PARTITIONING

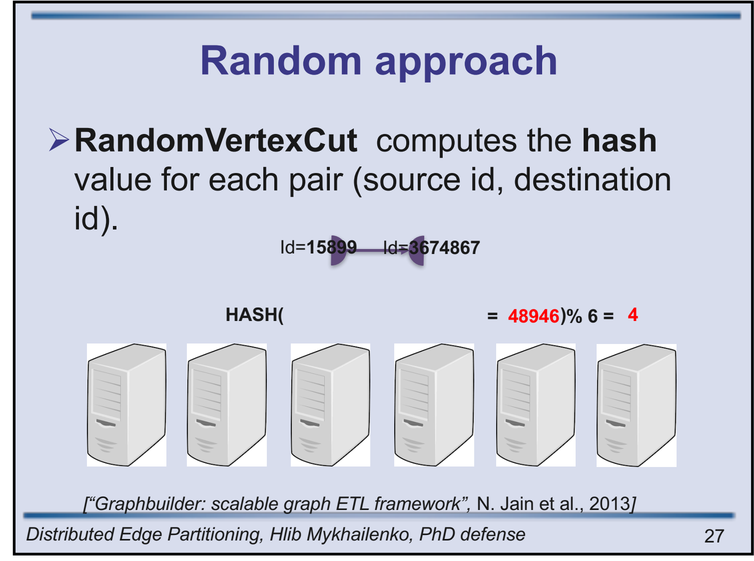
23

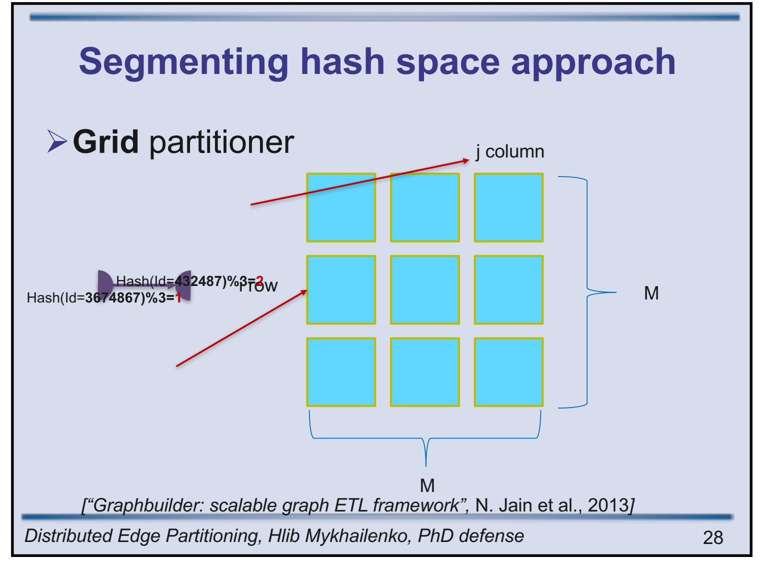
Motivation

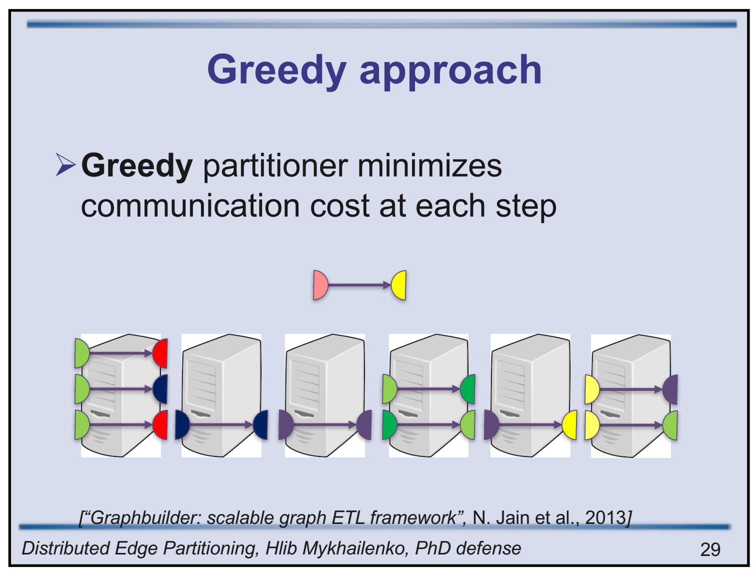
- Divide a graph into partitions
- Allow for parallel/distributed processing
- Optimal partitioning ?
  - NP Complete :(
- Good partitioning ?
  - Based on heuristics
  - Try to optimize some metrics (std dev of size partition, inter partitions communications…)
- How to partition ?
  - Vertex vs Edges

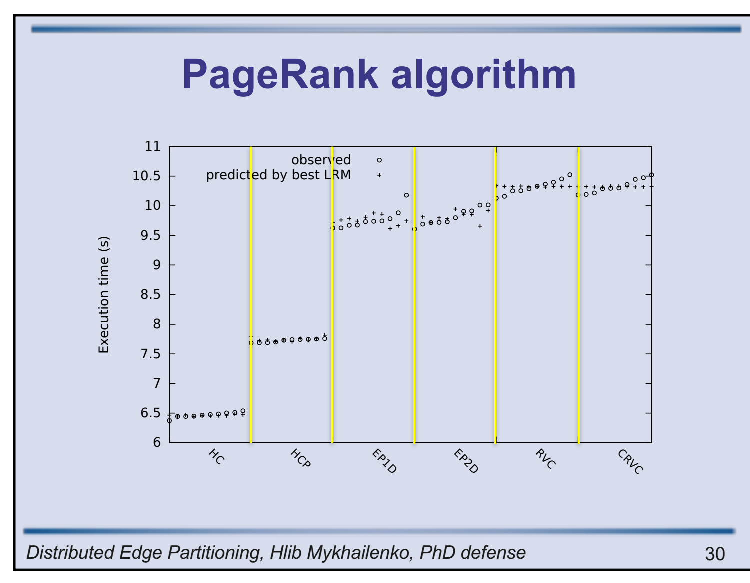

Vertex partitioning
partitions
Distributed Edge Partitioning, Hlib Mykhailenko, PhD defense
25


Edge partitioning
frontier vertices
Distributed Edge Partitioning, Hlib Mykhailenko, PhD defense
26


Random approach
RandomVertexCut computes the hash value for each pair (source id, destination id).
Id=15899
Id=3674867
HASH(
= 48946)% 6 = 4
["Graphbuilder: scalable graph ETL framework", N. Jain et al., 2013]
Distributed Edge Partitioning, Hlib Mykhailenko, PhD defense
27


Segmenting hash space approach
Grid partitioner
j column
M
M
["Graphbuilder: scalable graph ETL framework", N. Jain et al., 2013]
Distributed Edge Partitioning, Hlib Mykhailenko, PhD defense
28





Frameworks

GRAPHCHI

31

Overview

- Single machine
  - Distributed systems are complicated!
- Disk-based system
  - Memory is cheap but limited
- Supports both static and dynamic graph
- Kyrola, Aapo and Blelloch, Guy and Guestrin, Carlos, ***GraphChi: Large-scale Graph Computation on Just a PC***, Proceedings of OSDI'12

Computational Model

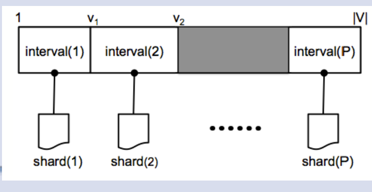
- Vertex centric
  - Vertices and Edges have associated values
  - Update a vertex values using edges values
- Typical update
  - Read values from edges
  - Compute new value
  - Update edges
- Asynchronous model
  - Always get the most recent value for edges
  - Schedule multiple updates

Storing graphs on disk

- Compressed Sparse Row (CSR)
  - Equivalent to adjacency sets
  - Store out-edges of vertex consecutively on Disk
  - Maintain index to adjacency sets for each vertex
- Very efficient for out-edges, not so for in-edges
  - Use *Compressed Sparse Column (CSC)*
- Changing edges values
  - On modification of out-edge : write to CSC
  - On reading of in-edge : read from CSR
  - Random read or random write ☹

Parallel Sliding Windows

- Minimize non sequential disk access
- 3 stages algorithm
- Storing graph on disk
  - Vertices *V* are split into *P* disjoints intervals
  - Store all edges that have **destination** in an interval in a *Shard*
  - Edges are stored by source order



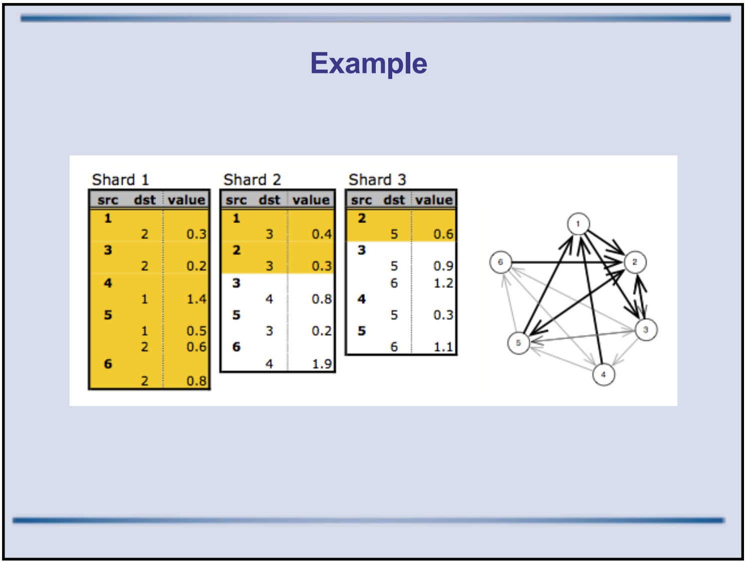
From ***Kyrola and al.***

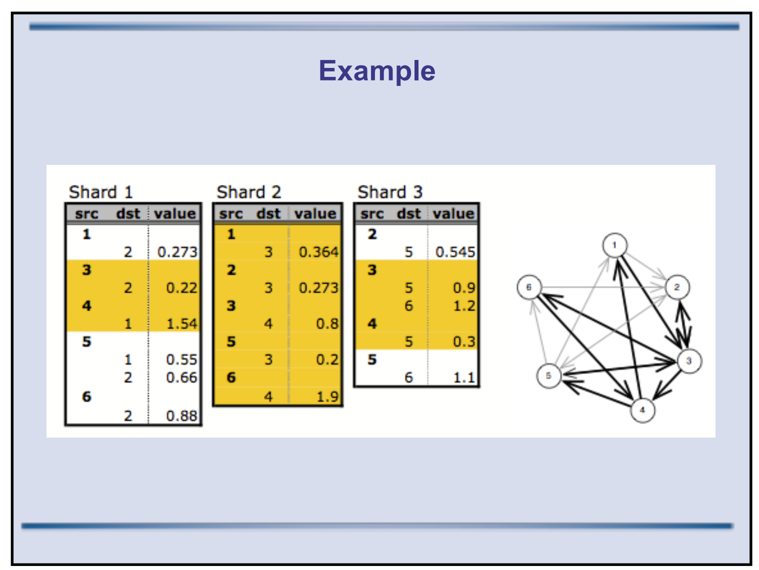
Parallel Sliding Windows (2)

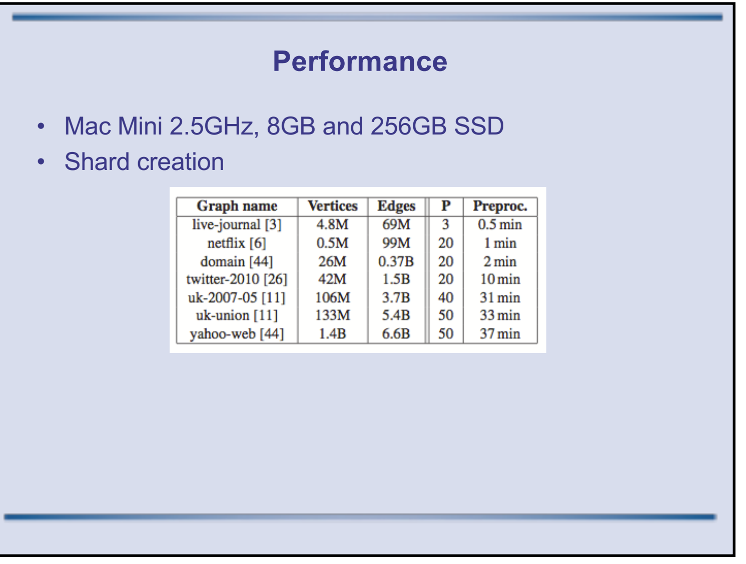
- Loading subgraph of vertices in interval *p*
  - Load Shard(p) in memory
    - Get in-edges immediately
  - Out-edges are stored in the P-1 other shards
    - But ordered by sources, so easy to find
- Loading subgraph p+1
  - Slide a window over all shards
- Each interval requires *P* sequential reads

Parallel updates

- Once interval loaded, update in parallel
- Data races
  - Only a problem if considering edge with both endpoints in interval
  - Enforce sequential update
- Write back result to disk
  - Current shard totally rewritten
  - Sliding window of other shards rewritten







Graph name	Vertices	Edges	P	Preproc.
live-journal [3]	4.8M	69M	3	0.5 min
netflix [6]	0.5M	99M	20	1 min
domain [44]	26M	0.37B	20	2 min
twitter-2010 [26]	42M	1.5B	20	10 min
uk-2007-05 [11]	106M	3.7B	40	31 min
uk-union [11]	133M	5.4B	50	33 min
yahoo-web [44]	1.4B	6.6B	50	37 min

Performance (2)

Application & Graph	Iter.	Comparative result	GraphChi (Mac Mini)	Ref
Pagerank & domain	3	GraphLab[30] on AMD server (8 CPUs) **87 s**	**132 s**	-
Pagerank & twitter-2010	5	Spark [45] with 50 nodes (100 CPUs): **486.6 s**	**790 s**	[38]
Pagerank & V=105M, E=3.7B	100	Stanford GPS, 30 EC2 nodes (60 virt. cores), **144 min**	approx. **581 min**	[37]
Pagerank & V=1.0B, E=18.5B	1	Piccolo, 100 EC2 instances (200 cores) **70 s**	approx. **26 min**	[36]
Webgraph-BP & yahoo-web	1	Pegasus (Hadoop) on 100 machines: **22 min**	**27 min**	[22]
ALS & netflix-mm, D=20	10	GraphLab on AMD server: **4.7 min**	**9.8 min** (in-mem) **40 min** (edge-repl.)	[30]
Triangle-count & twitter-2010	-	Hadoop, 1636 nodes: **423 min**	**60 min**	[39]
Pagerank & twitter-2010	1	PowerGraph, 64 x 8 cores: **3.6 s**	**158 s**	[20]
Triange-count & twitter- 2010	-	PowerGraph, 64 x 8 cores: **1.5 min**	**60 min**	[20]

GOOGLE PREGEL

Overview

- Directed graphs
- Distributed Framework Based on the *Bulk Synchronous Parallel* model
- *Vertex Centric* computation model
- Private framework with C++ API
- Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz 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)

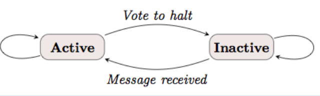
Model of Computation (1)

- BSP : model for parallel programming
  - Takes into account communication/synchronization
  - Series of super-steps (iterations)
    - Performs local computations
    - Communicate with others
    - Barrier

From : http://www.multicorebsp.com/

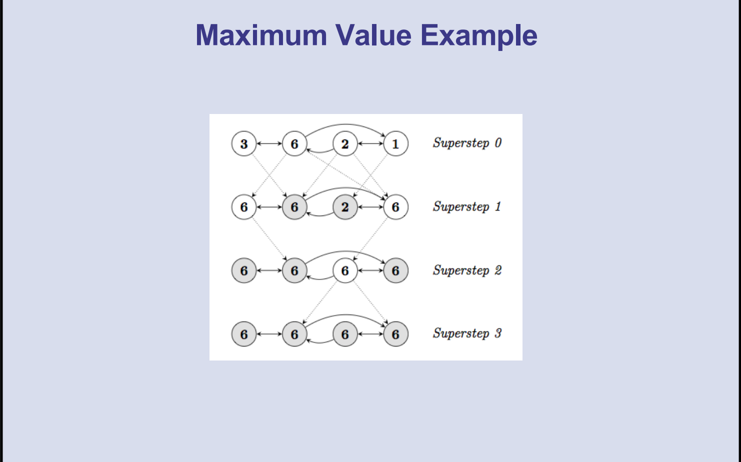
Model of Computation (2)

- Vertex Centric
  - Each vertex execute a function in parallel
- Can read messages sent at previous super-step
- Can send messages to be read at next super-step
  - Not necessarily following edges
- Can modify state of outgoing edges
- Run until all vertices agree to stop and no message in transit



From **Malewicz and al.**

Maximum Value Example



From **Malewicz and al.**

Implementation and Execution (1)

- User provides a graph, some input (vertex and edges values) and a program
- The program is executed on all nodes of a cluster
  - One node become the master, other are workers
- The graph is divided into partitions by the master
  - Vertex Id used to compute partition index (*e.g. hash(Id) mod N*)
- Partitions are assigned to workers
- User input file is partitioned (no fancy hash) and sent to workers
  - If some input is not for the worker, it will pass it along

Implementation and Execution (2)

- The master request worker to perform superstep
  - At the end, each worker reports the number of active vertices for next superstep
- Aggregators can be used at end of super-step to reduce communications
  - Perform reduction on values before sending
- If no more active vertices, Master can halt computation
- What about failures ?
  - Easy to checkpoint workers at end of superstep
  - If failure, rollback to previous checkpoint
  - If master fails… too bad ☹

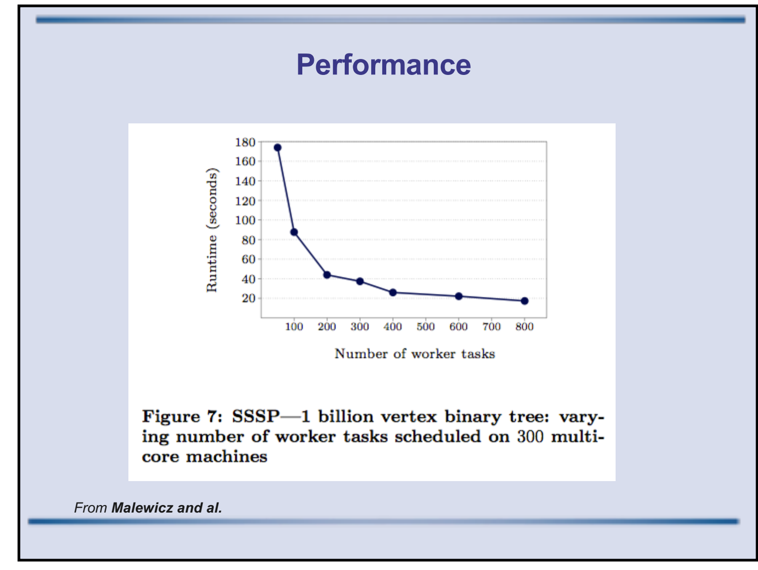
PageRank in Pregel

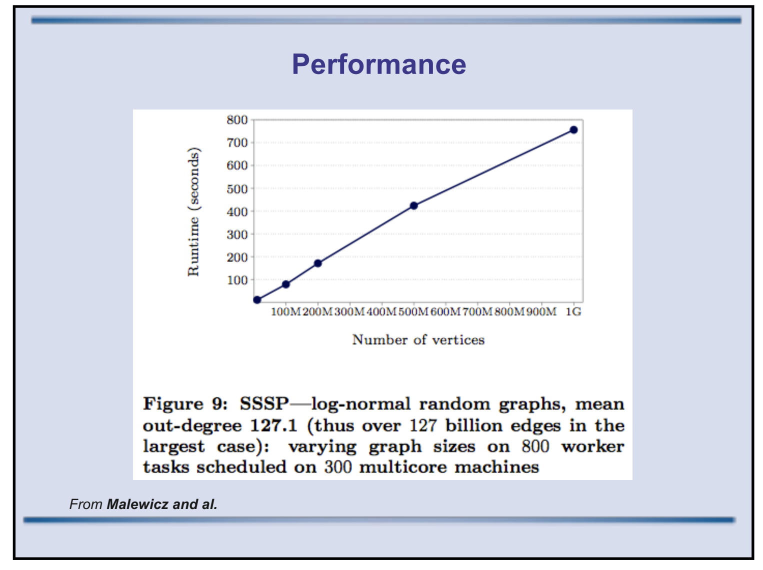
$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

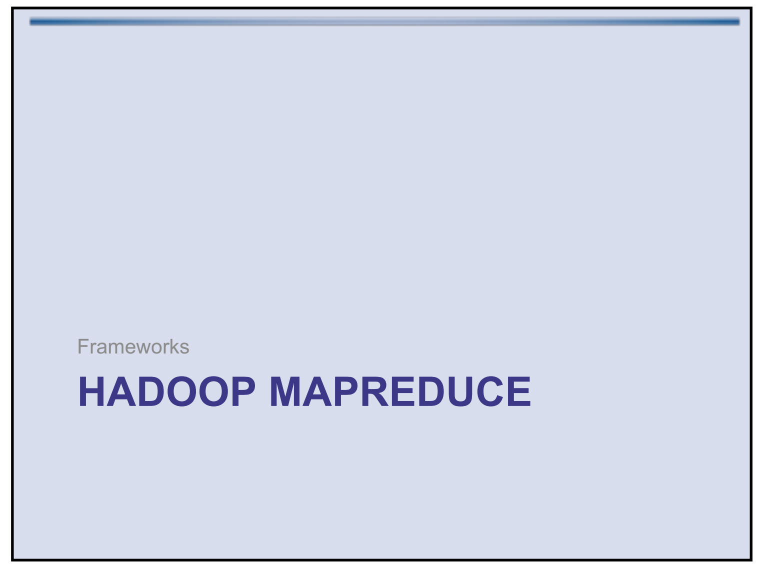
```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

From **Malewicz and al.**







Map Reduce operations

- Input data are (key, value) pairs
- 2 operations available : map and reduce
- Map
  - Takes a (key, value) and generates other (key, value)
- Reduce
  - Takes a key and all associated values
  - Generates (key, value) pairs
- A map-reduce algorithm requires a mapper and a reducer
- Re-popularized by Google
  - **MapReduce: Simplified Data Processing on Large Clusters** Jeffrey Dean and Sanjay Ghemawat, OSDI'04

Map Reduce example

- Compute the average grade of students
  - For each course, the professor provides us with a text file
  - Text file format : lines of "student grade"
- Algorithm (non map-reduce)
  - For each student, collect all grades and perform the average
- Algorithm (map-reduce)
  - Mapper
    - Assume the input file is parsed as (student, grade) pairs
    - So … do nothing!
  - Reducer
    - Perform the average of all values for a given key

Map Reduce example

Course 1
Bob 20
Brian 10
Paul 15

Course 2
Bob 15
Brian 20
Paul 10

Course 3
Bob 10
Brian 15
Paul 20

Map

(Bob , 20)
(Brian, 10)
(Paul, 15)
(Bob , 15)
(Brian, 20)
(Paul, 10)
(Bob , 10)
(Brian, 15)
(Paul, 20)

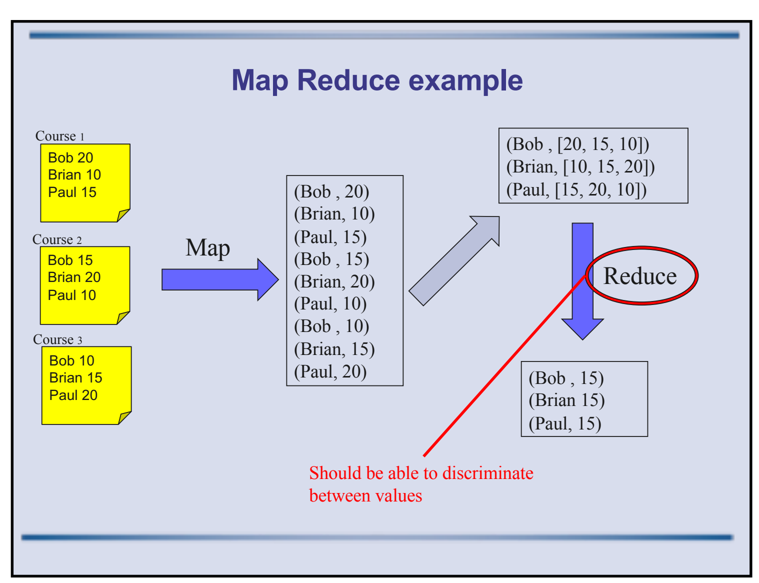
(Bob , [20, 15, 10])
(Brian, [10, 15, 20])
(Paul, [15, 20, 10])

Reduce

(Bob , 15)
(Brian 15)
(Paul, 15)

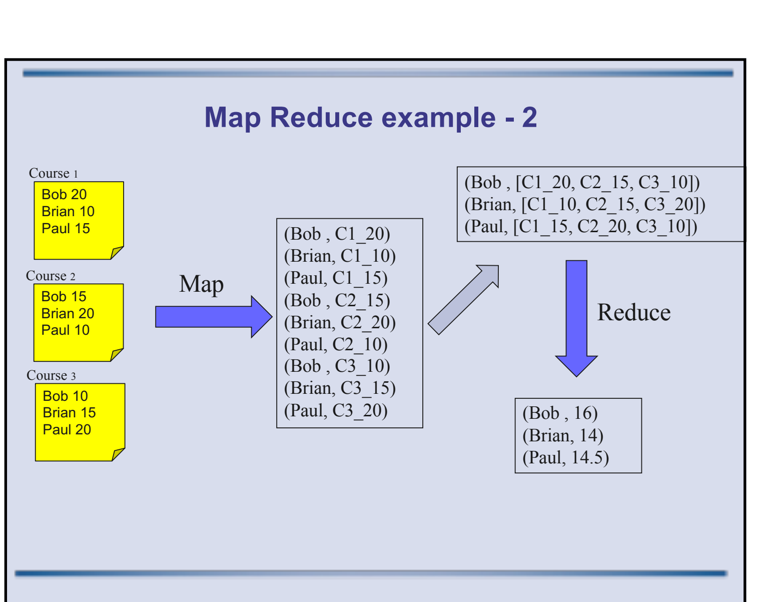
Map Reduce example… too easy ☺

- Ok, this was easy because
  - We didn't care about technical details like reading inputs
  - All keys are "equals", no weighted average
- Now can we do something more complicated ?
- Let's computed a weighted average
  - Course 1 has weight 5
  - Course 2 has weight 2
  - Course 3 has weight 3
- What is the problem now ?



Map Reduce example - advanced

- How discriminate between values for a given key
  - We can't … unless the values look different
- New reducer
  - Input : (Name, [course1_Grade1, course2_Grade2, course3_Grade3])
  - Strip values from course indication and perform weighted average
- So, we need to change the input of the reducer which comes from… the mapper
- New mapper
  - Input : (Name, Grade)
  - Output : (Name, courseName_Grade)
  - The mapper needs to be aware of the input file



What is Hadoop ?

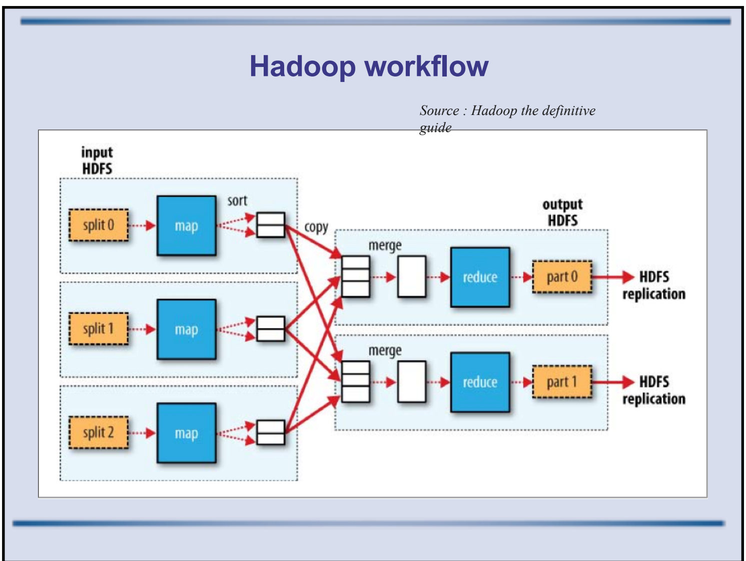
- A set of software developed by Apache for distributed computing
- Many different projects
  - MapReduce
  - HDFS : Hadoop Distributed File System
  - Hbase : Distributed Database
  - ….
- Written in Java
  - Bindings for your favorite languages available
- Can be deployed on any cluster easily

Hadoop Job

- An Hadoop job is composed of a map operation and (possibly) a reduce operation
- Map and reduce operations are implemented in a *Mapper* subclass and a *Reducer* subclass
- Hadoop will start many instances of *Mapper* and *Reducer*
  - Decided at runtime but can be specified
- Each instance will work on a subset of the keys called a *Splits*

Hadoop workflow

Source : Hadoop the definitive guide



Graphs and MapReduce

- How to write a graph algorithm in MapReduce?
- Graph representation ?
  - Use adjacency matrix

	V_1	V_2	V_3
V_1	0	0	1
V_2	1	0	1
V_3	1	1	0

- Line based representation
  - V_1 : 0, 0, 1
  - V_2 : 1, 0, 1
  - V_3 : 1, 1, 0
- Size $|V|^2$ with tons of 0 …

63

Sparse matrix representation

- Only encode useful values, i.e. non 0
  - V_1 : (V_3 ,1)
  - V_2 : (V_1,1), (V_3 ,1)
  - V_3 : (V_1,1), (V_2,1)
- And if equal weights
  - V_1 : V_3
  - V_2 : V_1, V_3
  - V_3 : V_1,V_2

Single Source Shortest Path

- Find the shortest path from one source node *S* to others
- Assume edges have weight 1
- General idea is BFS
  - Distance(*S*) = 0
  - For all nodes *N* reachable from S
    - Distance(*N*) = 1
  - For all nodes *N* reachable from other set of nodes *M*
    - Distance(N) = 1+ min(Distance(M))
  - And start next iteration

MapReduce SSSP

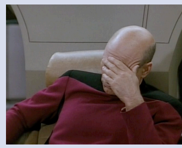
- Data
  - Key : node N
  - Value : (d, adjacency list of N)
    - d distance from S so far
- Map :
  - $\forall m \in$ adjacency list: emit (m, d + 1)
- Reduce :
  - Keep minimum distance for each node
- This basically advances the frontier by one hop
  - Need more iterations

MapReduce SSSP (2)

- How to maintain graph structure between iterations
  - Output adjacency list in mapper
  - Have special treatment in reducer
- Termination ?
  - Eventually ☺
  - Stops when no new distance is found… (any idea how?)

Seriously ?

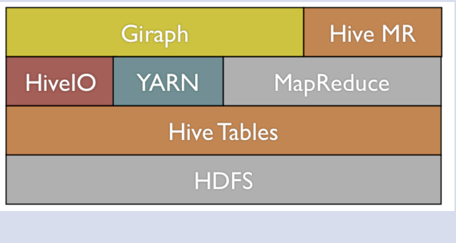
- MapReduce + Graphs is easy



- But everyone is MapReducing the world!
  - Because they are forced to
  - And because of Hadoop
- Hadoop gives
  - A scalable infrastructure (computation and storage)
  - Fault tolerance
- So let's use Hadoop as an underlying infrastructure

Giraph

- Built on top of Hadoop
- Vertex centric and BSP model ☺
  - Giraph jobs run as MapReduce



Source : https://m.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920/

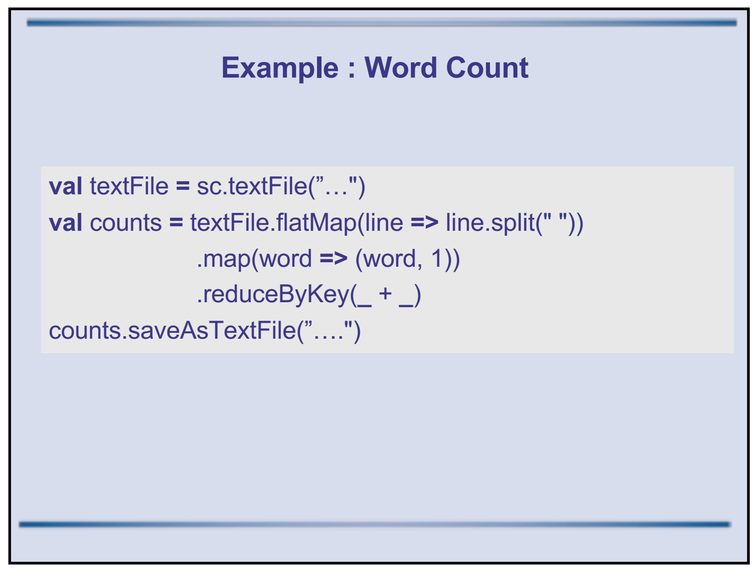
SPARK AND GRAPHX

Spark

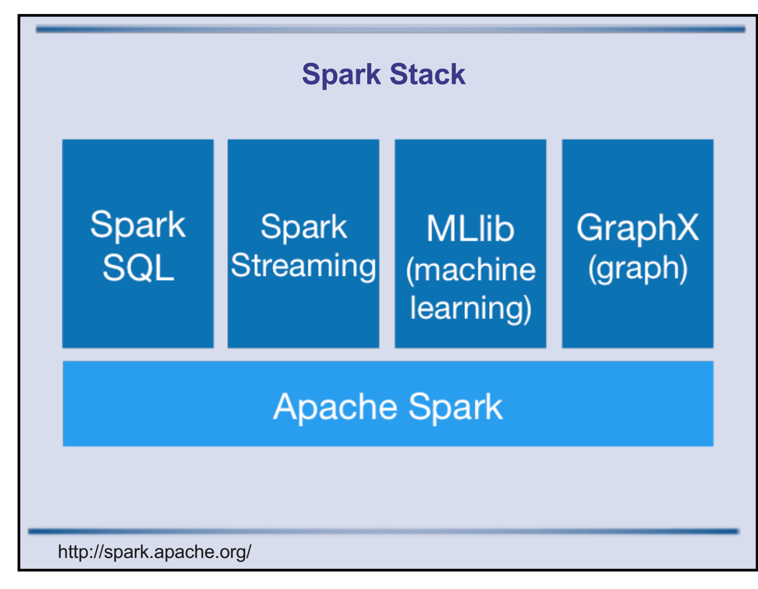
- Addresses limitations of Hadoop
  - Disk intensive
  - No support for iteration (cycles)
- Spark
  - In-Memory computation
  - Workflows with cycles
  - Still relies on Map-Reduce like operations
  - Multi languages support : Scala, Java, Python, R
- https://spark.apache.org/

Resilient Distributed Datasets

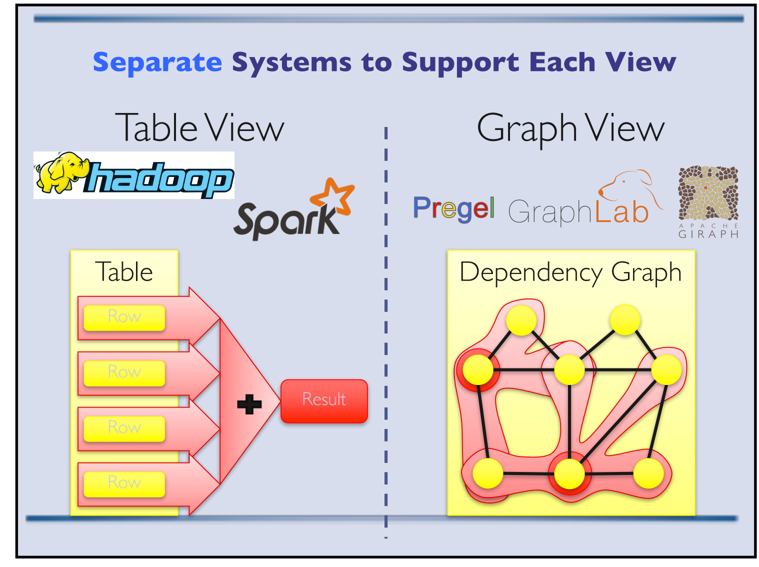
- RDDs
  - Array-like data structure
  - Mostly in-memory
  - Partitioned
  - Fault tolerant
  - Immutable ← very important !
- RDDs are created through **transformations**
  - Of raw data or another RDD
  - Example : map, filter, reduceByKey, groupBy…
- RDDs support **actions**
  - Example : collect, count, reduce, save…
- Transformations are lazy



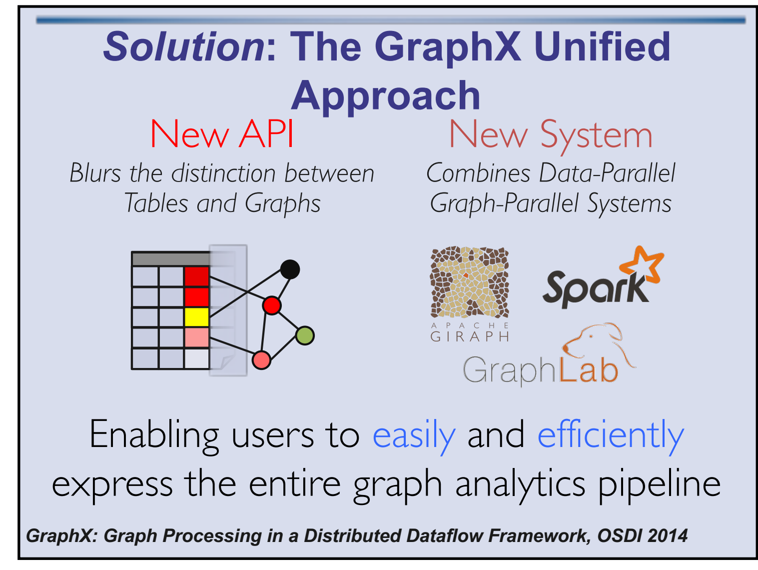
Example : Word Count
val textFile = sc.textFile("…")
val counts = textFile.flatMap(line => line.split(" "))
.map(word => (word, 1))
.reduceByKey(_ + _)
counts.saveAsTextFile("….")



Spark Stack
Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark
http://spark.apache.org/



Separate Systems to Support Each View
Table View
Graph View
hadoop
Spark
Pregel
GraphLab
APACHE GIRAPH
Table
Row
Row
Row
Row
Result
Dependency Graph



Solution: The GraphX Unified Approach
New API
Blurs the distinction between Tables and Graphs
New System
Combines Data-Parallel Graph-Parallel Systems
APACHE GIRAPH
Spark
GraphLab
Enabling users to easily and efficiently express the entire graph analytics pipeline
GraphX: Graph Processing in a Distributed Dataflow Framework, OSDI 2014

Abstractions

- Graphs are represented by 2 collections
  - Vertex RDD (IDs, Properties)
  - Edges RDD(sIDs, dIDs, Properties)
- Graphs have multiple properties
  - edges, vertices
- Most graphs operations can be expressed as analyzing or joining collections
  - *Join stage* (build a triple view)
  - *Group-by-stage* (reduce-like)
  - *Map operations*

Building a Graph

```
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD

val vertices :  VertexRDD[String] = ….

val edges : EdgeRDD[Int] = ….

val graph : Graph(vertices, edges) = …

graph.edges.count()
```

Triplets Join Vertices and Edges

- The ***triplets*** view joins vertices and edges:

Vertices Triplets Edges

A B C D

A C, B C, C D

A B, A C, B C, C D

```
SELECT src.id, dst.id, src.attr, e.attr, dst.attr
FROM edges AS e
   LEFT JOIN vertices AS src, vertices AS dst ON e.srcId = src.Id AND
    e.dstId = dst.Id
```

GraphX: Graph Processing in a Distributed Dataflow Framework, OSDI 2014

Triplet view

```
graph.triplets
```

- Each Triplet contains
  - srcId and srcAttr
  - dstId and dstAttr
  - attr

Aggregate Messages

- Applies a user defined function to each edge triplet
  - The messages
- Applies a user defined function to aggregate the messages at destination vertex

```
def aggregateMessages[Msg: ClassTag](
   sendMsg: EdgeContext[VD, ED, Msg] => Unit,
   mergeMsg: (Msg, Msg) => Msg,
   tripletFields: TripletFields = TripletFields.All) : VertexRDD[Msg]
}
```

Example : get largest incoming edge

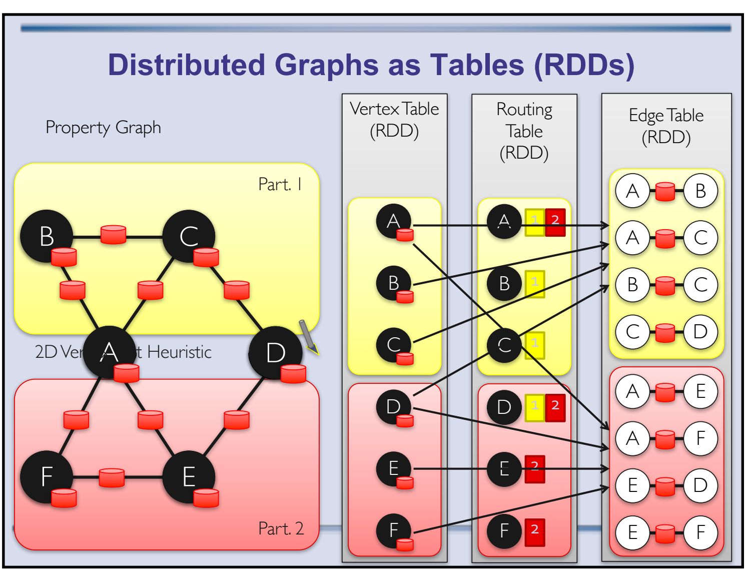
- For each vertex compute the largest incoming edge
  - Message is edge attribute value
  - Merge function is max

```
graph.aggregateMessages[Int](
         triplet => {
             triplet.sendToDst(triplet.attr)
         },
        (a,b) => Math.max(a,b)
)
```

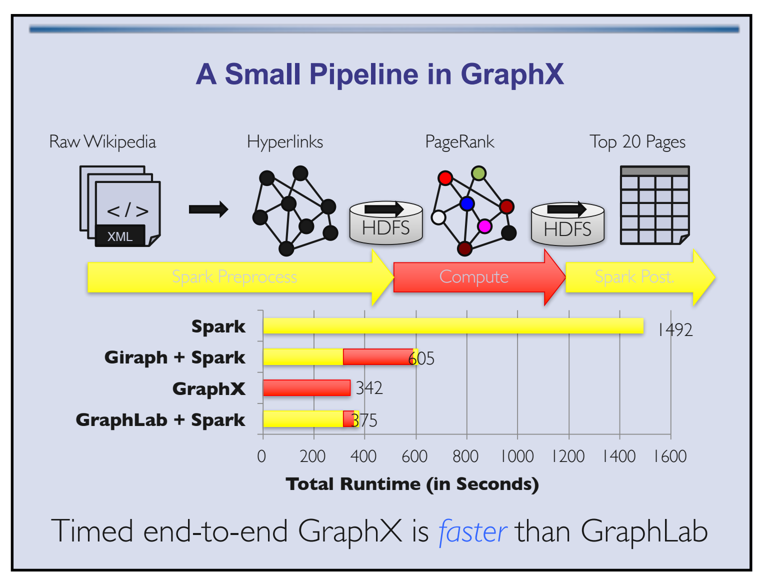
Misc operations

- RDD -> Array
  - take(n)
- Compute the degree of each vertex
  - graph.inDegrees/outDegrees
- Collect edges for all vertices
  - **val** coll = graph.collectNeighborIds(EdgeDirection.X) with X In, Out, Either
- Get all edges of a given vertice id
  - coll.lookup(id)

Distributed Graphs as Tables (RDDs)



A Small Pipeline in GraphX



Timed end-to-end GraphX is *faster* than GraphLab

Conclusion

- So many frameworks to choose from…
- Criteria
  - What is the size of your graph ?
  - What algorithms do you want to run ?
  - How fast do you want your results ?
- Distributed frameworks are no silver bullet
  - Steeper learning curve
  - Add new problems (data distribution, faults…)

Food for thought

- Distributed partitioning is a hot topic
  - But what is a good partitioner ?
- New hardware is massively parallel
  - GPGPU, Xeon Phi …
- The network might not be a bottleneck anymore
  - RDMA + Infiniband == profit !
  - *The end of slow networks: it's time for a redesign,* Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian, Proc. VLDB Endow. 2016,
- Hardware contention is an issue
  - *Performance Impact of Resource Contention in Multicore Systems*, R. Hood, H. Jin, P. Mehrotra, J. Chang, J. Djomehri, S. Gavali, D. Jespersen, K. Taylor, R. Biswas, IPDPS 2010

Resources

- Slides
  - http://www.slideshare.net/shatteredNirvana/pregel-a-system-for-largescale-graph-processing
  - *http://courses.cs.washington.edu/courses/cse490h/08au/lectures/algorithms.pdf*
  - http://www.cs.kent.edu/~jin/Cloud12Spring/GraphAlgorithms.pptx
  - https://amplab.cs.berkeley.edu/wp-content/uploads/2014/02/graphx@strata2014_final.pptx