Indian Institute of Science
Bangalore, India
भारतीय विज्ञान संस्थान
बंगलौर, भारत

SE256:Jan16 (2:1)

L12:Distributed Graph Processing

Yogesh Simmhan

23/30 Mar, 2016

©Yogesh Simmhan & Partha Talukdar, 2016
This work is licensed under a Creative Commons Attribution 4.0 International License
Copyright for external content used with attribution is retained by their original authors

Graphs are commonplace

- Web & Social Networks
  - Web graph, Citation Networks, Twitter, Facebook, Internet
- Knowledge networks & relationships
  - Google's Knowledge Graph, NELL
- Cybersecurity
  - Telecom call logs, financial transactions, Malware
- Internet of Things
  - Transport, Power, Water networks
- Bioinformatics
  - Gene sequencing, Gene expression networks

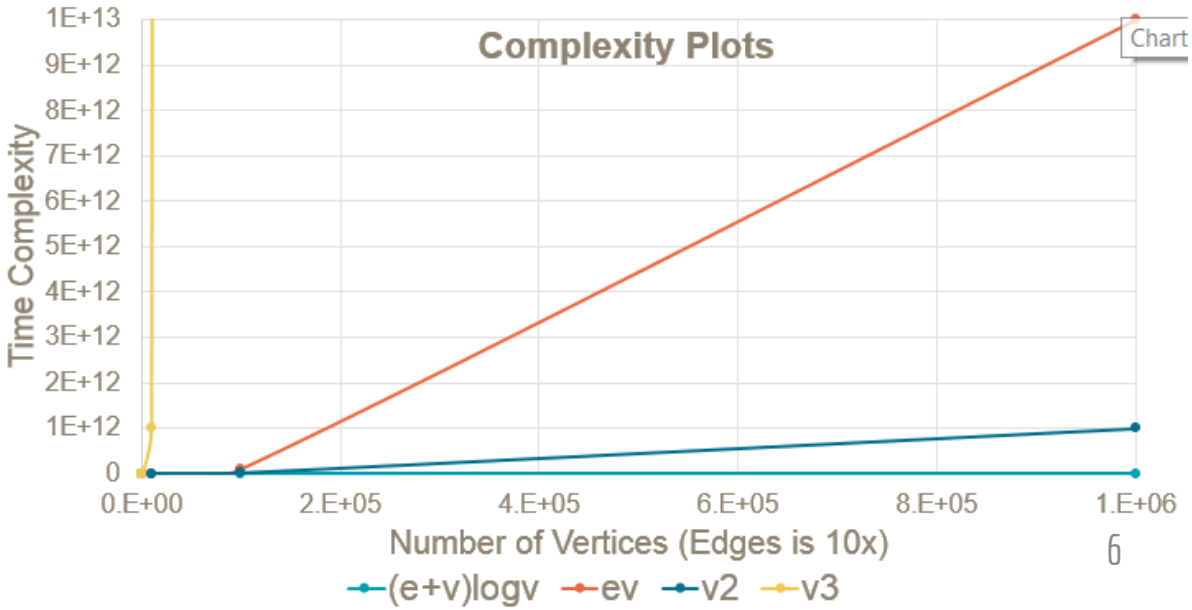
Graph Algorithms

- Traversals: *Paths & flows between different parts of the graph*
  - Breadth First Search, Shortest path, Minimum Spanning Tree, Eulerian paths, MaxCut
- Clustering: *Closeness between sets of vertices*
  - Community detection & evolution, Connected components, K-means clustering, Max Independent Set
- Centrality: *Relative importance of vertices*
  - PageRank, Betweenness Centrality

But, Graphs can be challenging

- Computationally complex algorithms
  - Shortest Path: O((E+V) log V) ~ O(EV)
  - Centrality: O(EV) ~ O(V^3)
  - Clustering: O(V) ~ O(V^3)
- *And these are for "shared-memory" algorithms*

Graph500.org's fastest supercomputer, ***K computer*** *with 524,288 cores performed at* ***17E+12 TEPS***



But, Graphs can be challenging

- Graphs sizes can be huge
  - Google's index contains 50B pages
  - Facebook has around 1.1B users
  - Twitter has around 530M users
  - Google+ has around 570M users

Apache Giraph, Claudio Martella, Hadoop Summit, Amsterdam, April 2014

But, Graphs can be challenging

- Shared memory algorithms don't scale!
- Do not fit naturally to *Hadoop/MapReduce*
  - Multiple MR jobs (iterative MR)
  - Topology & Data written to HDFS each time
  - *Tuple*, rather than graph-centric, abstraction
- Lot of work on *parallel graph libraries* for HPC
  - Boost Graph Library, Graph500
  - Storage & compute are (loosely) coupled, not fault tolerant
  - But everyone does not have a supercomputer ☺
- Processing and *querying* are different
  - Graph DBs not suited for analytics
  - Focus on large simple graphs, complex "queries"
  - *E.g. Neo4J, FlockDB, 4Store, Titan*

Google's Pregel

- Google, to overcome, these challenges came up with Pregel.
  - Provides scalability
  - Fault-tolerance
  - Flexibility to express arbitrary algorithms

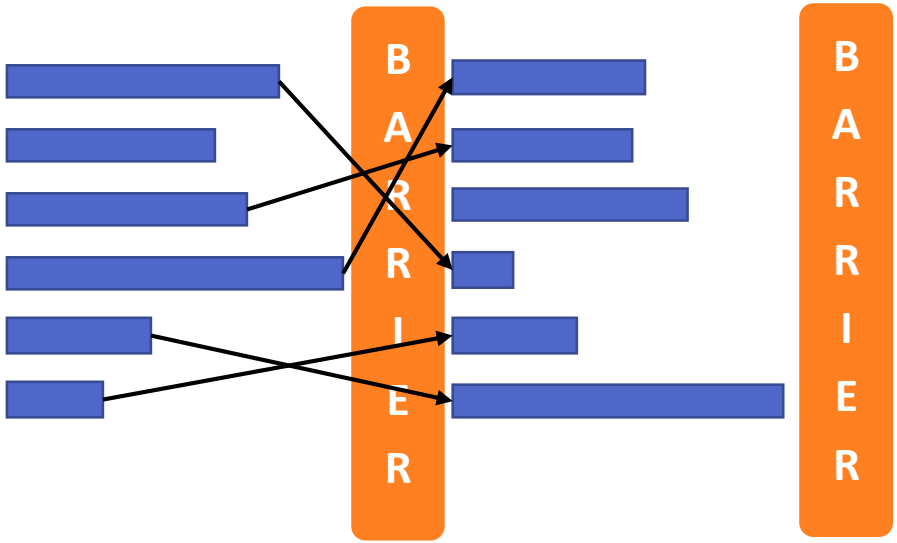
- The high level organization of Pregel programs is inspired by Valiant's Bulk Synchronous Parallel (BSP) model [1].

Slides courtesy "Pregel: A System for Large-Scale Graph Processing, Malewicz, et al, SIGMOD 2010"

[1] Leslie G. Valiant, A Bridging Model for Parallel Computation. Comm. ACM 33(8), 1990

Bulk Synchronous Parallel (BSP)

- Distributed execution model
  - Compute → Communicate → Compute → Communicate → …
  - Bulk messaging avoids comm. costs



Vertex-centric BSP

- Series of iterations (*supersteps*) .
- Each vertex *V* invokes a function in parallel.
- Can read messages sent in previous superstep (*S-1*).
- Can send messages, to be read at the next superstep (*S+1*).
- Can modify state of outgoing edges.

Pregel: A System for Large-Scale Graph Processing, Malewicz, et al, SIGMOD 2010

Advantage?

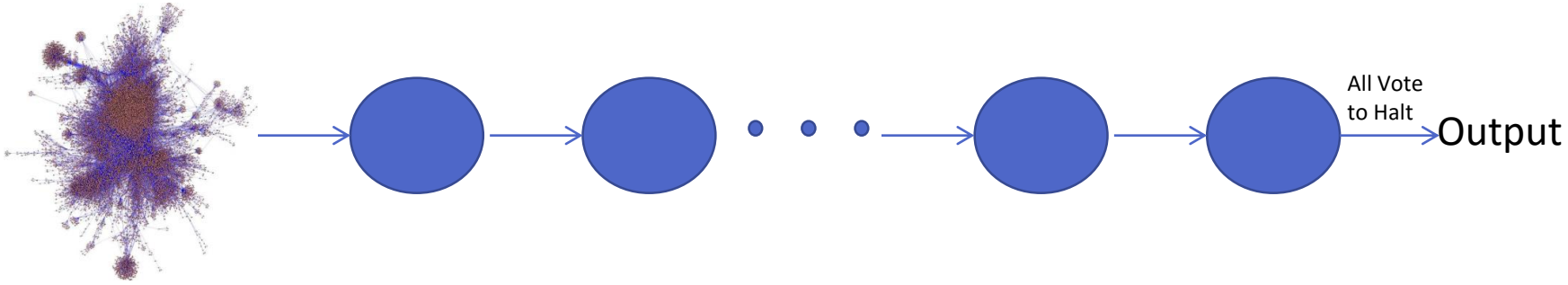
- In Vertex-Centric Approach
- Users focus on a *local action*
  - Think of **Map** method over tuple
- Processing each item *independently*.
- Ensures that Pregel programs are inherently free of *deadlocks* and *data races* common in asynchronous systems.

Pregel: A System for Large-Scale Graph Processing, Malewicz, et al, SIGMOD 2010

Apache Giraph Implements *Pregel* Abstraction

- Google's Pregel, SIGMOD 2010
  - Vertex-centric Model
  - Iterative BSP computation
- Apache Giraph donated by Yahoo
  - Feb 6, 2012: Giraph 0.1-incubation
  - May 6, 2013: Giraph 1.0.0
  - Nov 19, 2014: Giraph 1.1.0
- Built on Hadoop Ecosystem

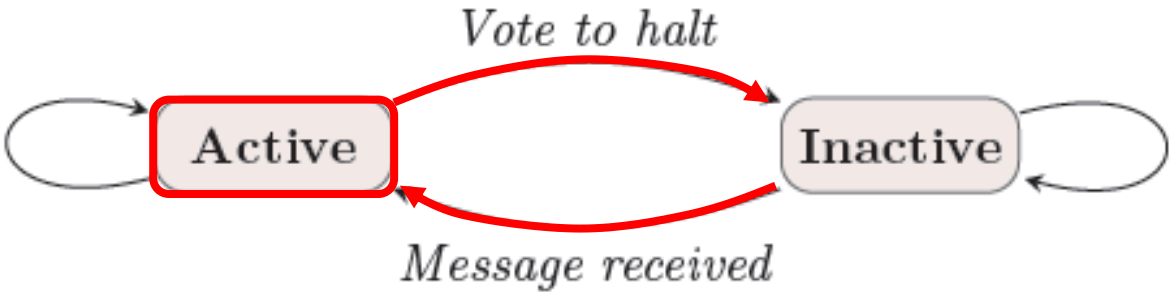
Model of Computation



- A Directed Graph is given to Pregel.
- It runs the computation at each vertex.
- Until all nodes vote for halt.
- Pregel gives you a directed graph back.

Pregel: A System for Large-Scale Graph Processing, Malewicz, et al, SIGMOD 2010

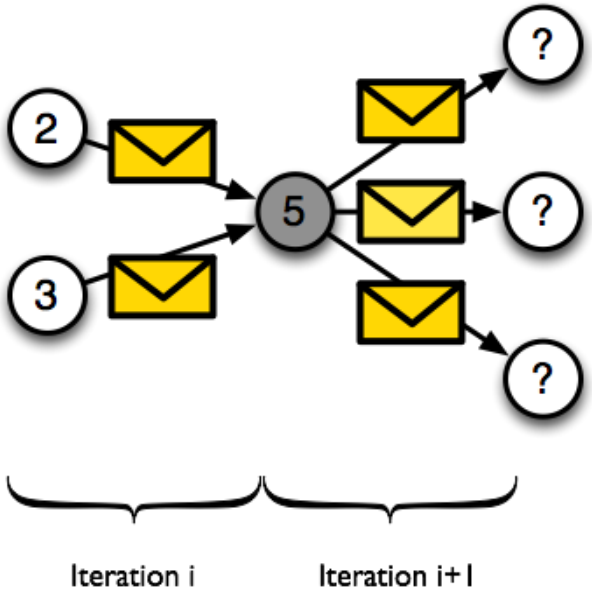
Vertex State Machine



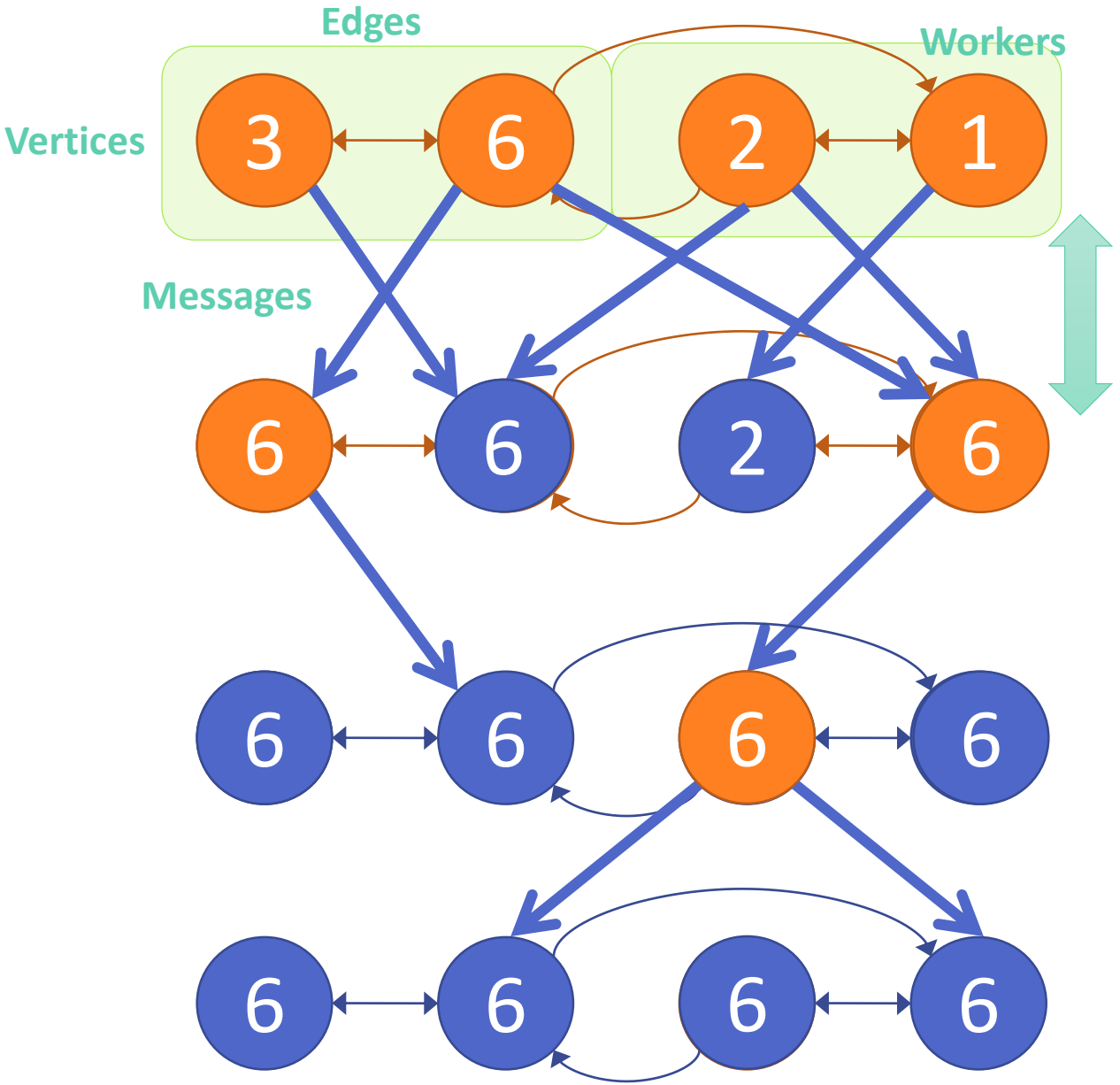
- Algorithm termination is based on every vertex *voting to halt*.
- In superstep 0, every vertex is in the *active* state.
- A vertex deactivates itself by voting to halt.
- It can be reactivated by receiving an (external) message.

Vertex Centric Programming

- Vertex Centric Programming Model
  - Logic written from perspective on a single vertex. Executed on all vertices.
- Vertices know about
  - Their own value(s)
  - Their outgoing edges



Apache Giraph, Claudio Martella, Hadoop Summit, Amsterdam, April 2014



Blue Arrows are messages.

Blue vertices have voted to halt.

Pregel: A System for Large-Scale Graph Processing, Malewicz, et al, SIGMOD 2010

Max Vertex

Algorithm 1 Max Vertex Value using Vertex Centric Model

```
1:  procedure COMPUTE(Vertex myVertex, Iterator⟨Message⟩ M)
2:      hasChanged = (superstep == 1) ? true : false
3:      while M.hasNext do            ▶ Update to max message value
4:          Message m ← M.next
5:          if m.value > myVertex.value then
6:              myVertex.value ← m.value
7:              hasChanged = true
8:      if hasChanged then            ▶ Send message to neighbors
9:          SENDTOALLNEIGHBORS(myVertex.value)
10:     else
11:         VOTETOHALT( )
```

GoFFish: A Sub-Graph Centric Framework for Large-Scale Graph Analytics, Simmhan, et al, EuroPar 2014

Advantages

- Makes distributed programming easy
  - No locks, semaphores, race conditions
  - Separates computing from communication phase
- Vertex-level parallelization
  - Bulk message passing for efficiency
- Stateful (in-memory)
  - Only messages & checkpoints hit disk

Apache Giraph: API

```
void compute(Iterator<IntWritable> msgs)
    getSuperstep()
    getVertexValue()
    edges = iterator()
    sendMsg(edge, value)
    sendMsgToAllEdges(value)
    voteToHalt()
```

Message passing

- No guaranteed message delivery order.
- Messages are delivered exactly once.
- Can send messages to any node.
  - Though, typically to neighbors

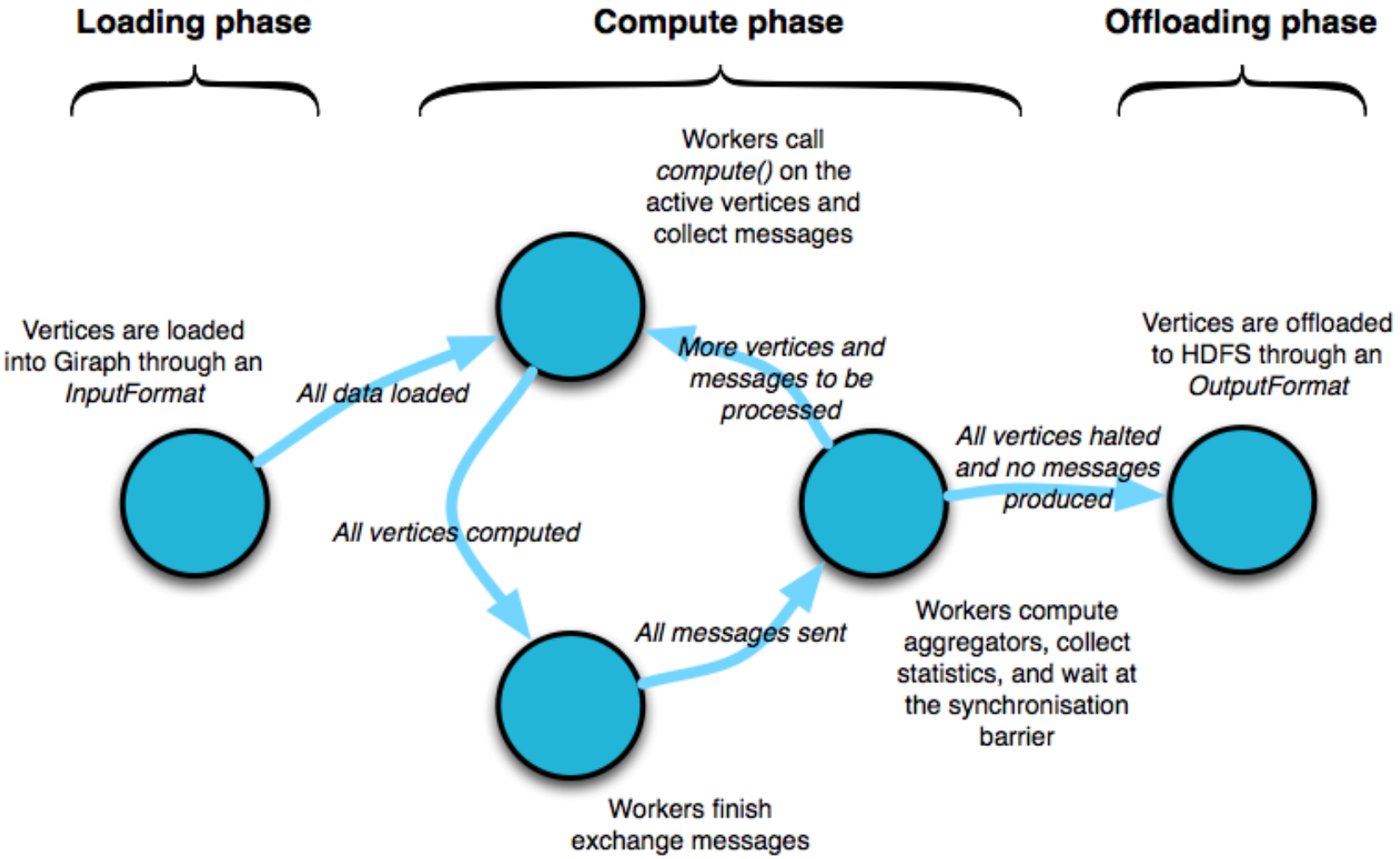
```
public class MaxVertexVertex extends IntIntNullIntVertex {
    public void compute(Iterator<IntWritable> messages)
            throws IOException {
        int currentMax = getVertexValue().get();
        // first superstep is special,
        // because we can simply look at the neighbors
        if (getSuperstep() == 0) {
            for (Iterator<IntWritable> edges =
                    iterator(); edges.hasNext();) {
                int neighbor = edges.next().get();
                if (neighbor > currentMax) {
                    currentMax = neighbor;
                }
            } ...
```

Based on org.apache.giraph.examples.ConnectedComponentsVertex

```
    ...
    // only need to send value if it is not the own id
    if (currentMax != getVertexValue().get()) {
         setVertexValue(new IntWritable(currentMax));
         for (Iterator<IntWritable> edges = iterator();
              edges.hasNext();) {
              int neighbor = edges.next().get();
              if (neighbor > currentMax) {
                sendMsg(new IntWritable(neighbor),
                   getVertexValue());
              }
         }
    }
    voteToHalt();
    return;
  } // end getSuperstep==0
```

```
        boolean changed = false; // getSuperstep != 0
        // did we get a smaller id?
        while (messages.hasNext()) {
          int candidateMax = messages.next().get();
          if (candidateMax > currentMax) {
                currentMax = candidateMax;
                changed = true;
             }
        }
        // propagate new component id to the neighbors
        if (changed) {
             setVertexValue(new IntWritable(currentMax));
             sendMsgToAllEdges(getVertexValue());
        }
        voteToHalt();
} // end compute()
```

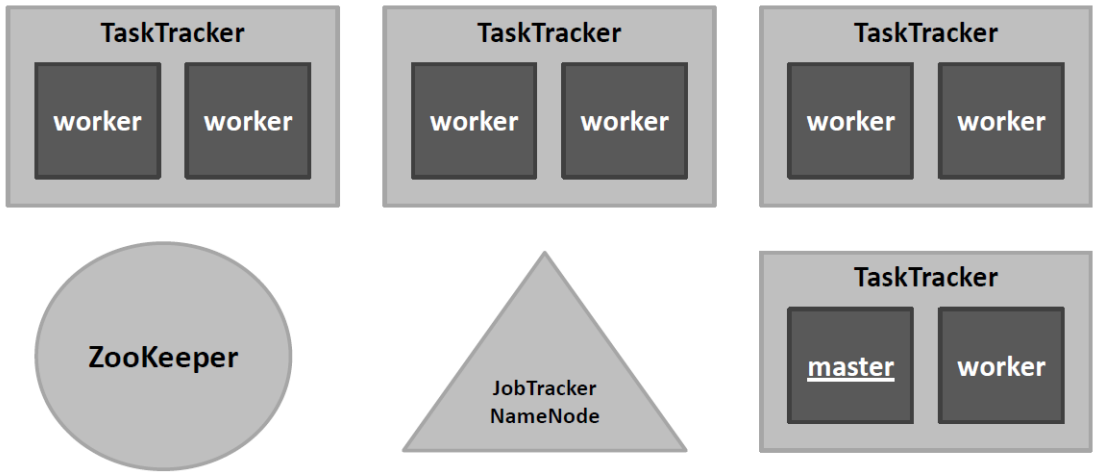
Apache Giraph



Apache Giraph, Claudio Martella, Hadoop Summit, Amsterdam, April 2014

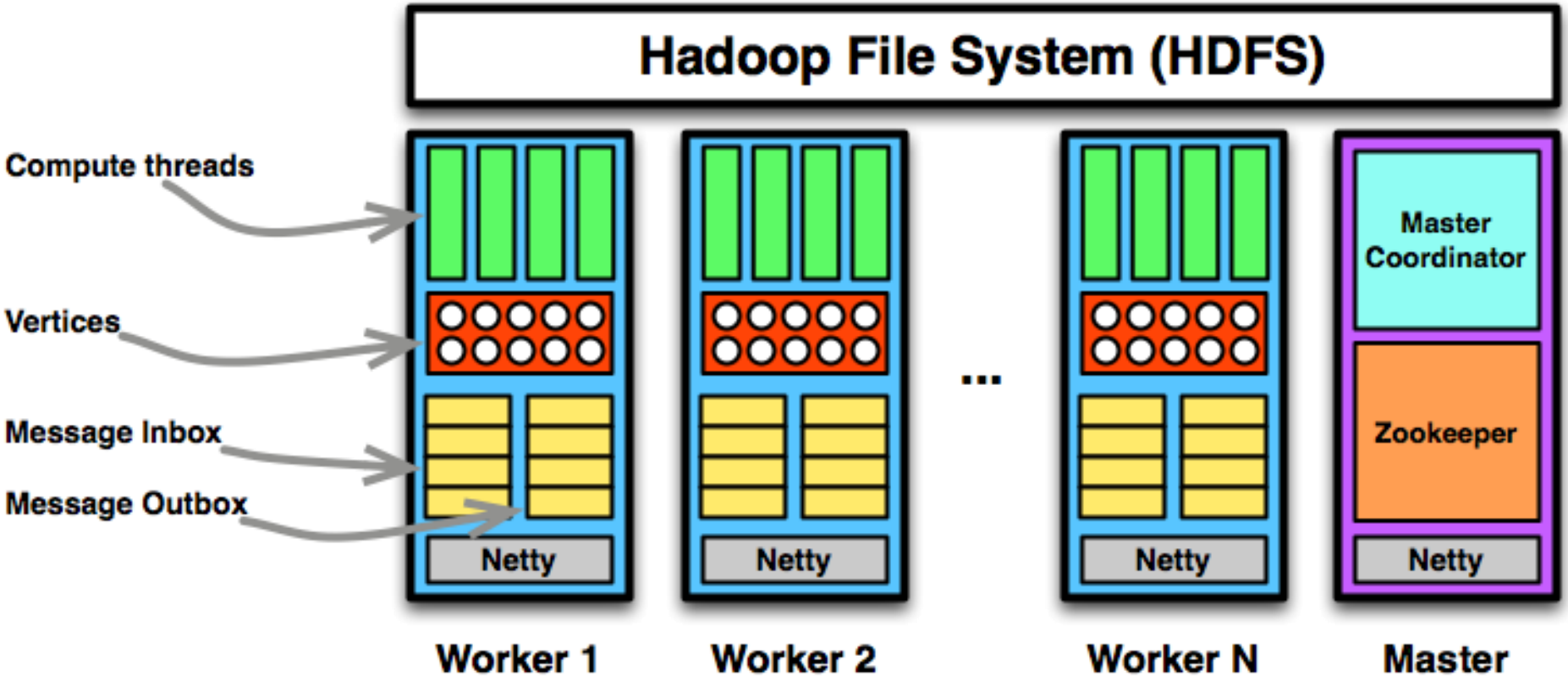
Giraph Architecture

- Hadoop Map-only Application
- **ZooKeeper**: responsible for **computation state**
  - Partition/worker mapping, global #superstep
- **Master**: responsible for **coordination**
  - Assigns partitions to workers, synchronization
- **Worker:** responsible for **vertices**
  - Invokes active vertices compute() function, sends, receives and assigns messages



© Sebastian Schelter

Giraph Architecture



- Checkpointing of supersteps possible

Apache Giraph, Claudio Martella, Hadoop Summit, Amsterdam, April 2014

Additional Features

Combiners

- Sending a message to another vertex that exists on a different machine has some overhead.
- User specifies a way to reduce many messages into one value (ala Reduce in MR).
  - by overriding the **Combine()** method.
  - Must be commutative and associative.
- Exceedingly useful in certain contexts (e.g., 4x speedup on shortest-path computation).

Additional Features

Aggregators

- A mechanism for global communication, monitoring, and data.
  - Each vertex can produce a value in a superstep S for the Aggregator to use.
  - The Aggregated value is available to all the vertices in superstep S+1.
- Aggregators can be used for statistics and for global communication.
  - E.g., **Sum** applied to out-edge count of each vertex.
    - generates the total number of edges in the graph and communicate it to all the vertices.

Pregel: A System for Large-Scale Graph Processing, Malewicz, et al, SIGMOD 2010

Additional Features

Topology mutations:

- Some graph algorithms need to change the graph's topology.
  - E.g. A clustering algorithm may need to replace a cluster with a node
- Vertices can create / destroy vertices at will.
- Resolving conflicting requests:
  - Partial ordering:

    E Remove,V Remove,V Add, E Add.
  - User-defined handlers:

    You fix the conflicts on your own.

Additional Features

- `MasterCompute`: Executed on master
- `WorkerContext`: Executed per worker
- `PartitionContext`: Executed per partition

Apache Giraph, Claudio Martella, Hadoop Summit, Amsterdam, April 2014

Fault Tolerance

- Checkpointing
  - The master periodically (alternate supersteps) instructs the workers to save the state of their partitions to HDFS.
    - e.g., Vertex values, edge values, incoming messages.
- Failure detection
  - Using regular “ping” messages.
- Recovery
  - The master reassigns graph partitions to the currently available workers.
  - The workers all reload their partition state from most recent available checkpoint.

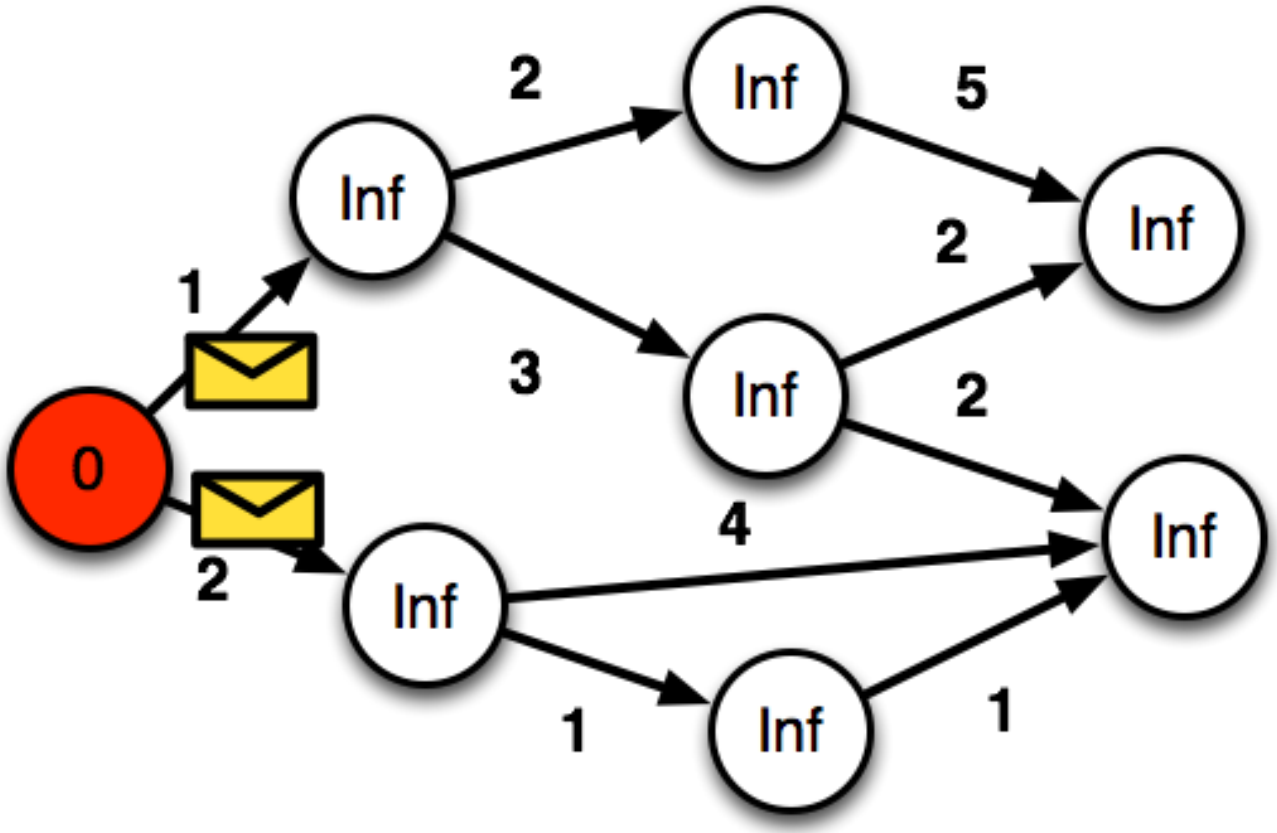
Pregel: A System for Large-Scale Graph Processing, Malewicz, et al, SIGMOD 2010

Shortest Path

```
class ShortestPathVertex
        : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
        int mindist = IsSource(vertex_id()) ? 0 : INF;
        for (; !msgs->Done(); msgs->Next())
            mindist = min(mindist, msgs->Value());
        if (mindist < GetValue()) {
            *MutableValue() = mindist;
            OutEdgeIterator iter = GetOutEdgeIterator();
            for (; !iter.Done(); iter.Next())
                SendMessageTo(iter.Target(),
                    mindist + iter.GetValue());
        }
        VoteToHalt();
  }
};
```

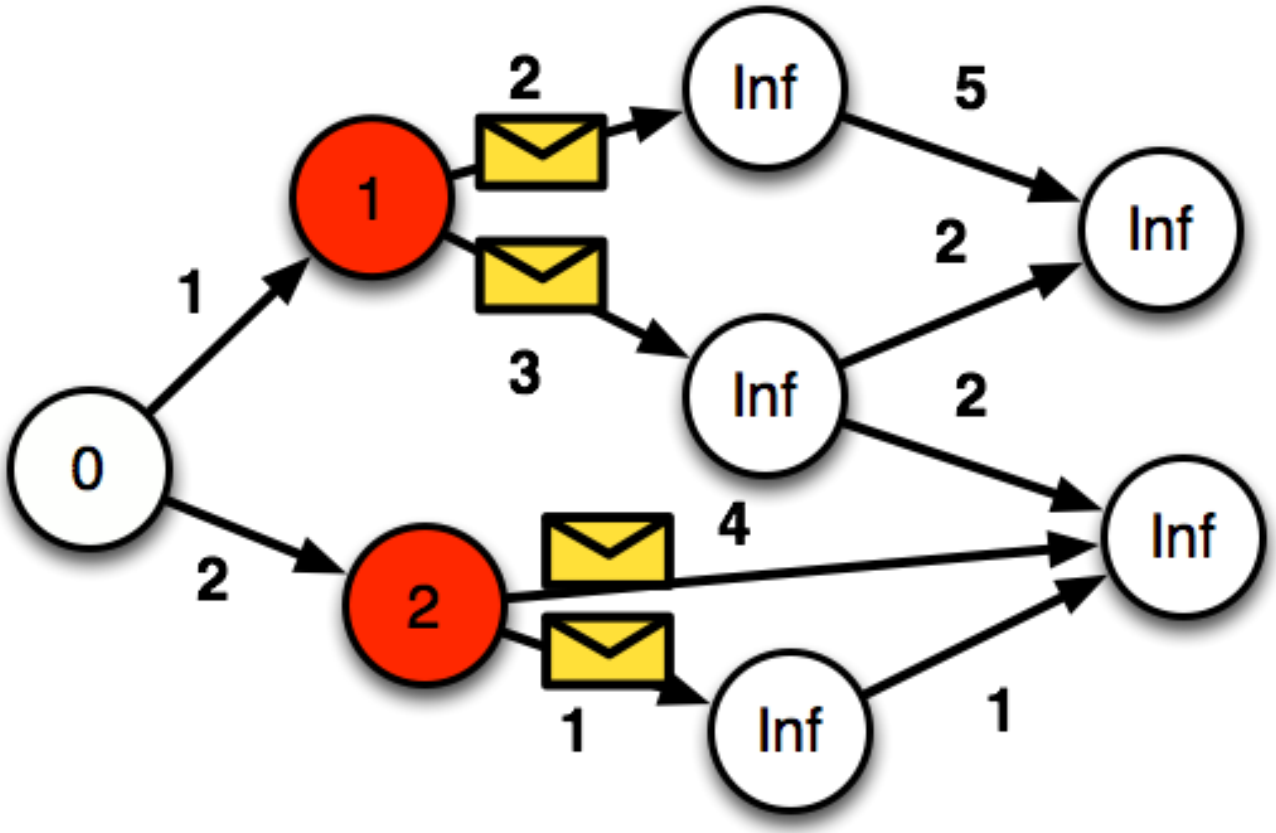
In the 1st superstep, only the source vertex will update its value (from INF to zero)

Shortest Path

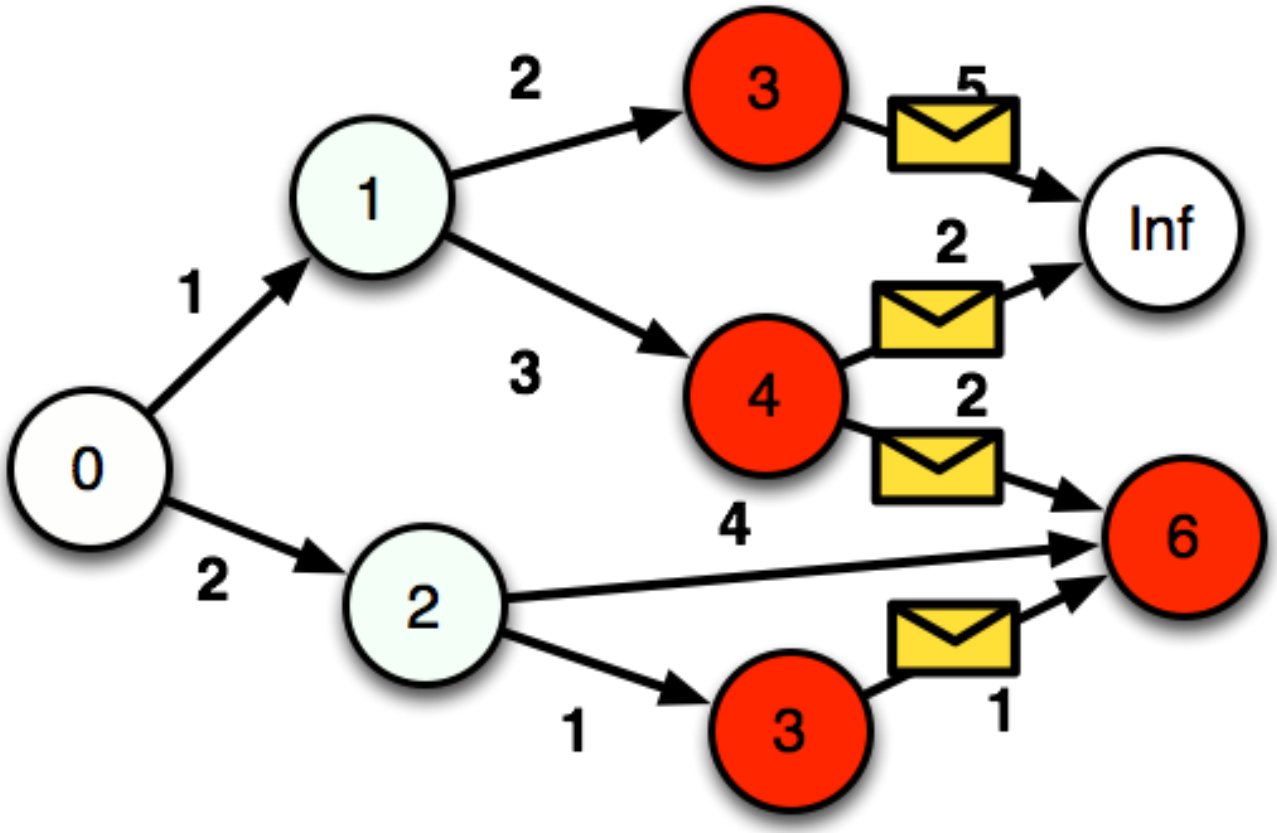


Apache Giraph, Claudio Martella, Hadoop Summit, Amsterdam, April 2014

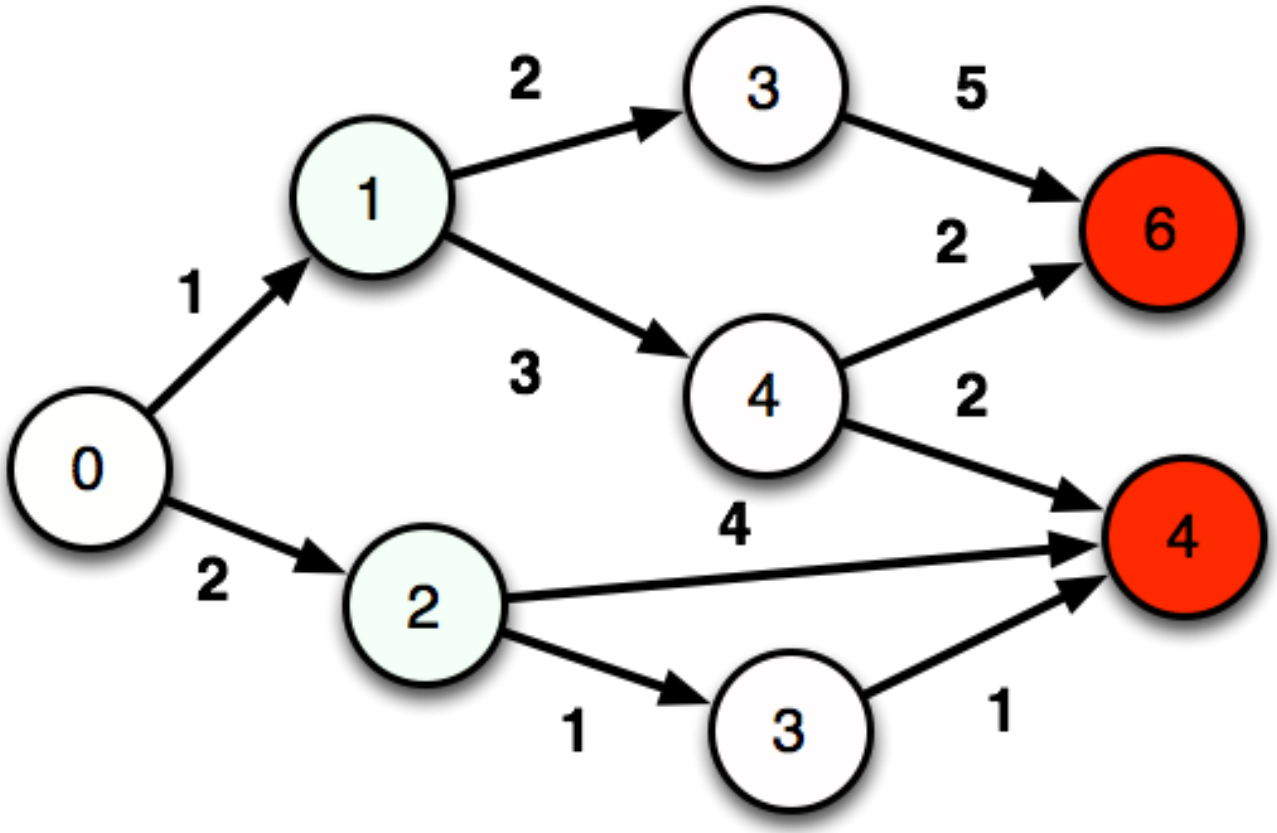
Shortest Path



Shortest Path



Shortest Path



PageRank, recursively

$$P(n) = \alpha \left(\frac{1}{|G|} \right) + (1 - \alpha) \sum_{m \in L(n)} \frac{P(m)}{C(m)}$$

- P(n) is PageRank for webpage/URL 'n'
  - Probability that you're in vertex 'n'
- |G| is number of URLs (vertices) in graph
- α is probability of random jump
- L(n) is set of vertices that link to 'n'
- C(m) is out-degree of 'm'

PageRank using MapReduce

```
1: class Mapper
2:     method Map(nid n, node N)
3:         p ← N.PageRank/|N.AdjacencyList|
4:         Emit(nid n, N)                          ▷ Pass along graph structure
5:         for all nodeid m ∈ N.AdjacencyList do
6:             Emit(nid m, p)                      ▷ Pass PageRank mass to neighbors

1: class Reducer
2:     method Reduce(nid m, [p1, p2, ...])
3:         M ← ∅
4:         for all p ∈ counts [p1, p2, ...] do
5:             if IsNode(p) then
6:                 M ← p                           ▷ Recover graph structure
7:             else
8:                 s ← s + p                       ▷ Sum incoming PageRank contributions
9:         M.PageRank ← s
10:        Emit(nid m, node M)
```

Lin, Fig 5.8

Application – Page Rank

Store and carry PageRank

```
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();
  }
};
```

K-Means Clustering

- Master-Compute Model

```
1  Input: undirected G(V, E), k, τ
2   int  numEdgesCrossing = INF;
3   while (numEdgesCrossing > τ)
4     int []  clusterCenters = pickKRandomClusterCenters(G)
5     assignEachVertexToClosestClusterCenter(G, clusterCenters)
6     numEdgesCrossing = countNumEdgesCrossingClusters(G)
```

Figure 3: A simple k-means like graph clustering algorithm.

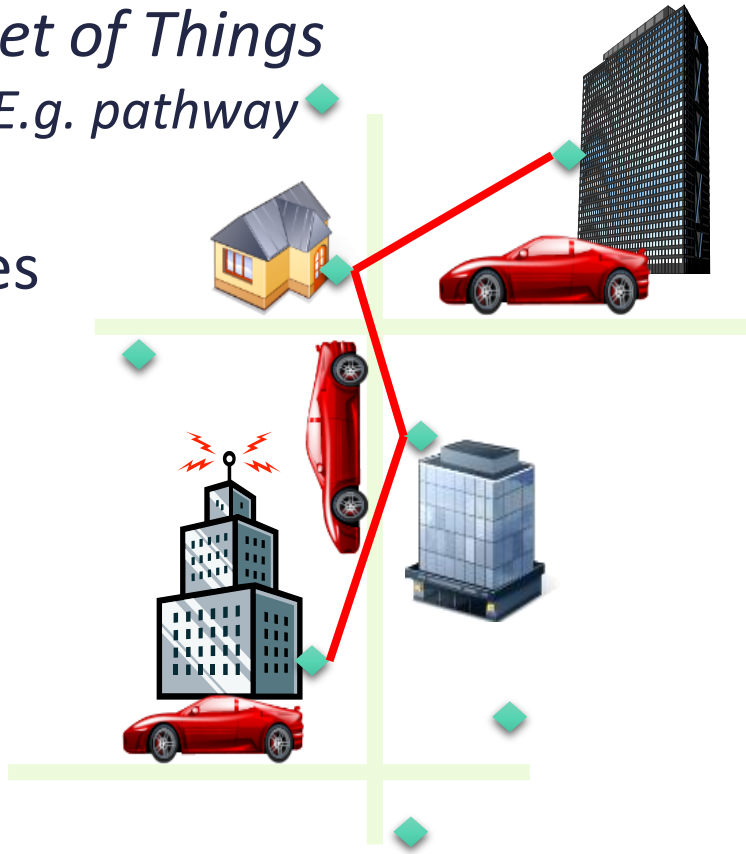
GPS: A Graph Processing System, Semih Salihoglu, et al, SSDBM, 2013

GoFFish

Subgraph-centric, Time-series graph processing

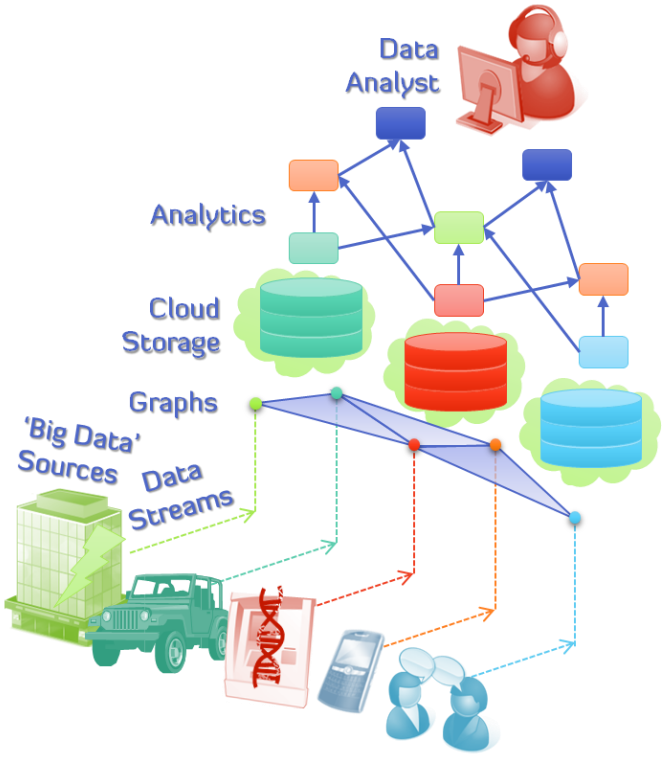
Analysis over Time-series Graphs

- Fixed graph of event sources…*Internet of Things*
  - Known relationships between sources *E.g. pathway*
  - E.g. *"red car" event from a camera*
- Event Streams form graph time-series
- Track path of the "red car"
  - Mine for interesting trajectory
- *Inferring track from micro-paths*
- Other analysis
  - Wide Area outage management
  - Event clustering and aggregation
  - Topic propagation in social networks
  - Time dependent shortest path



GoFFish: Temporal Graph Analytics

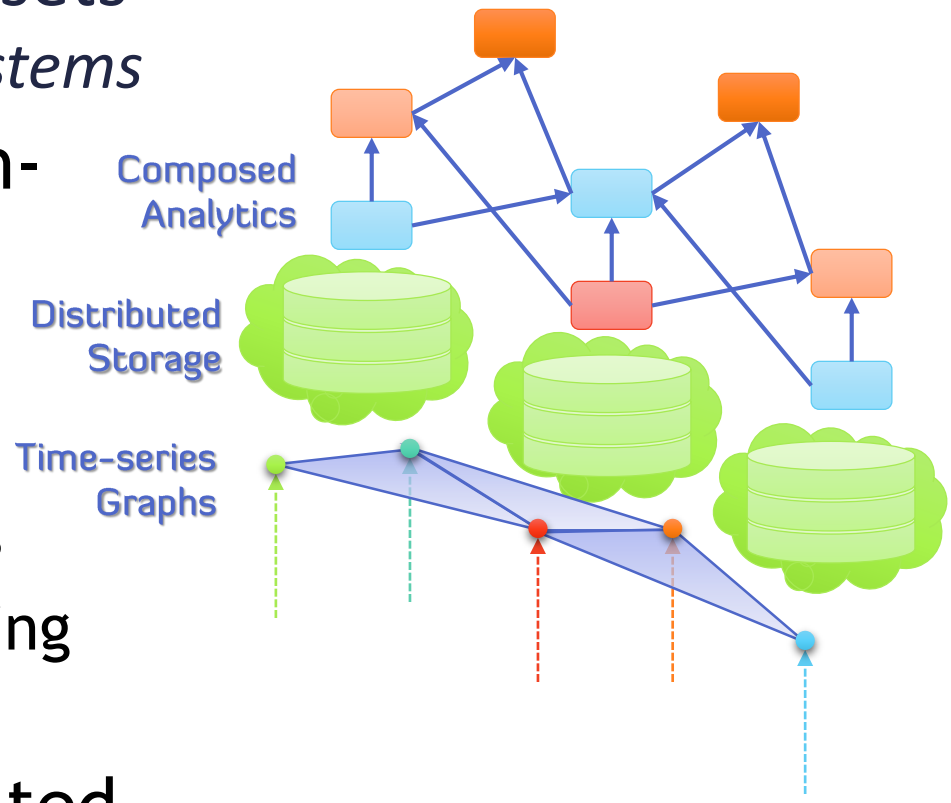
- Scalable platform for *graph-oriented* event analytics
- Designed for commodity H/W: clusters & clouds
- Novel data mapping to *time-series graphs*, with efficient layout on distributed storage
- Compose and efficiently execute *dataflow applications* over time-series graphs
- Funded by DARPA XDATA Grant



GoFFish: A Sub-Graph Centric Framework for Large-Scale Graph Analytics, Simmhan, et al, *EuroPar*, 2014

GoFFish Software Platform

- Platform to *store*, *compose & execute* analytics on *time-series graph* datasets
  - *At scale, on distributed systems*
- **GoFS**: Distributed Graph-oriented File System
- **Gopher**
  - Compose sub-graph centric complex analytics
  - Executed on ***Floe*** streaming dataflow engine
- Data & Compute collocated



Design Insights – *Mitigate Weak Links*

1) Disk I/O is the weakest link for big data
   - Do more with less disk reads, parallel disk I/O
   - *Graph **layout** on distributed disks is key!*

Think *Hive*

2) Network I/O
   - Limit network communicate. Transfer in chunks.
   - *Graph **partitioning** on distributed hosts is key!*

Think *Pregel*

3) Memory Capacity
   - Not all data fits in distributed memory
   - ***Incremental** loading & computation*

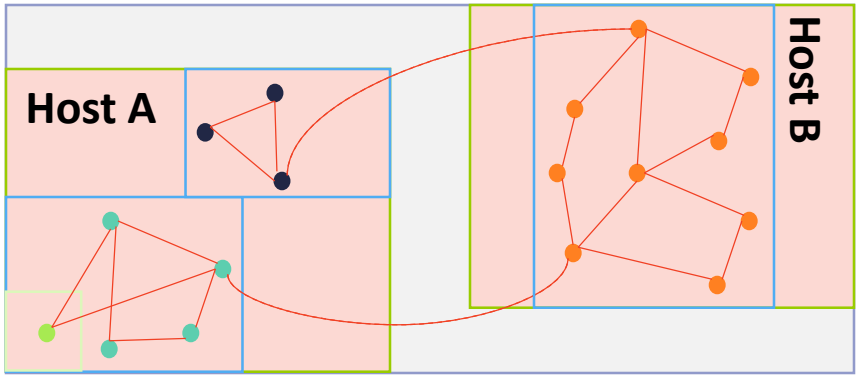
Think *Hadoop*

4) CPU
   - ***Elastic** Cloud execution, Leverage many-core*

USP!

Graph Data Model

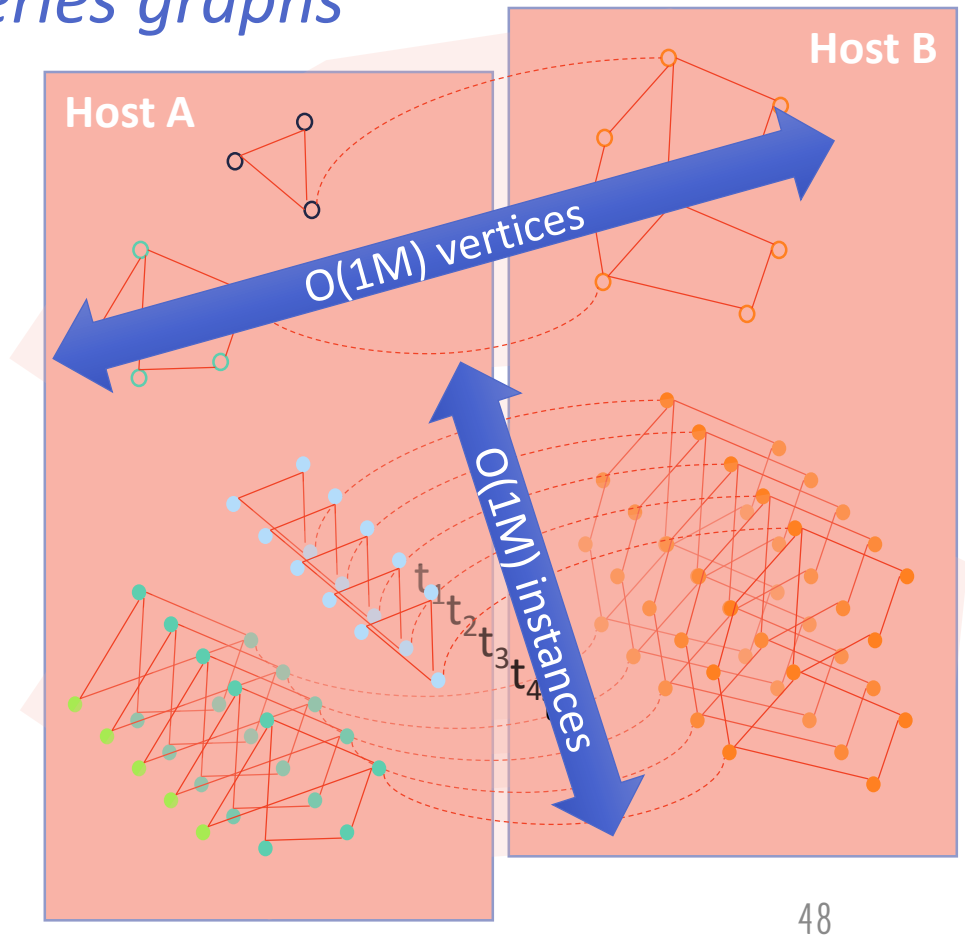
- Designed for "sub-graph" centric *distributed* computing
  - Graphs
    - Partitions … *Distributed evenly across machines*
      - Sub-graphs … *Logical unit of operation*
        - Vertices



- Sub-graph is unit of *distributed* data access & operation
  - Extends Google Pregel/Apache Giraph's vertex-centric BSP model … *no global view*

Time Series Graph Data Model

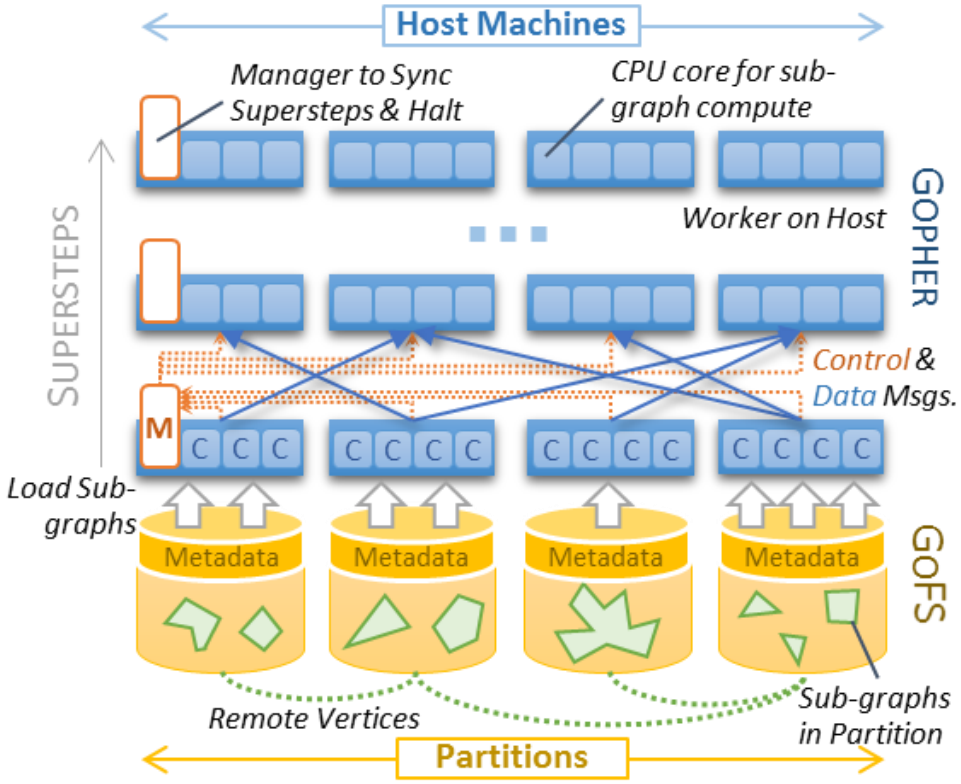
- Designed for "sub-graph" distributed centric computing … *over time-series graphs*
- Graph template
  - *Common* features
- Graph instances
  - *Time-variant* features



- Instance vertex follows the template vertex's partition

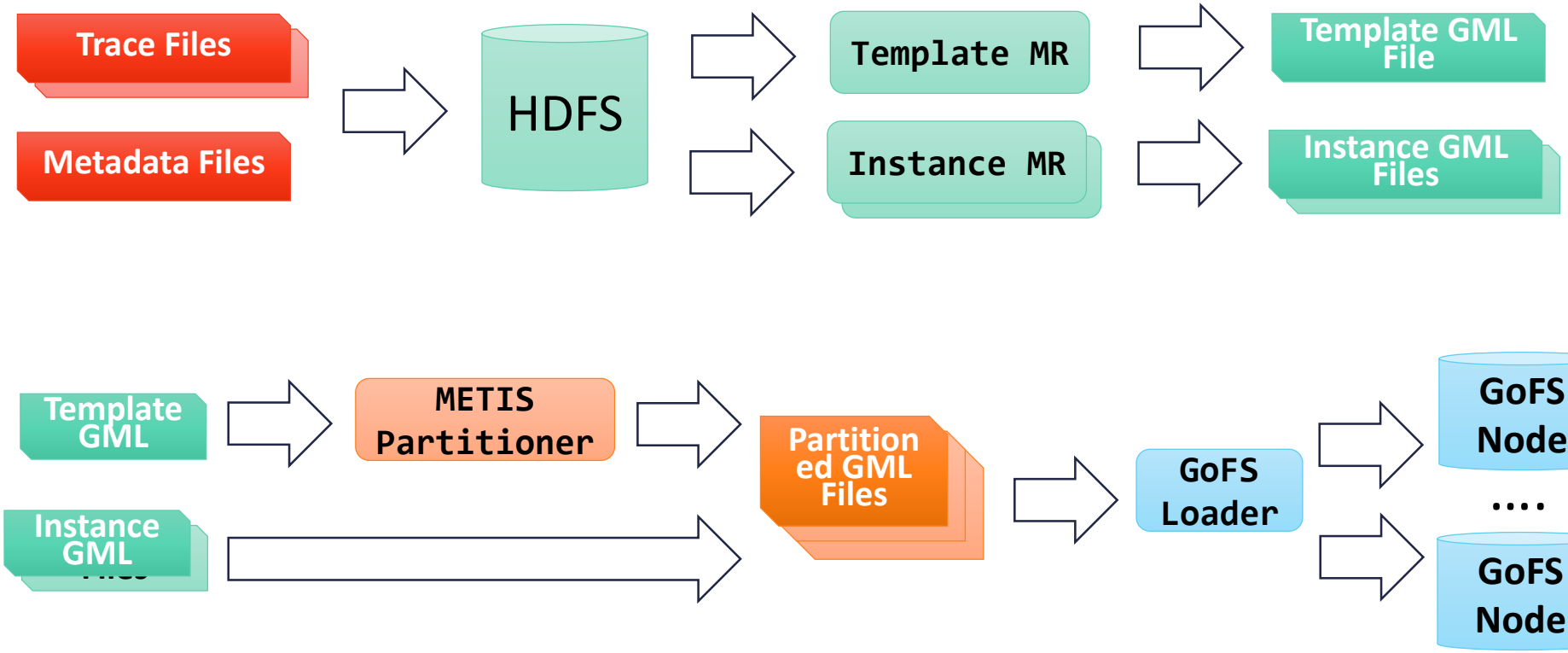
GoFS: Distributed TS Graph Store

- Data+Compute Co-design
- Graphs partitioned to reduce edge cuts
- Subgraphs identified within partitions
- Slice is unit of (file) storage on disk



- Exploits spatio-temporal locality in a slice
  - Bin pack small subgraphs in a partition
  - Temporal packing of different instances

E.g. Data Loading of NW Traces



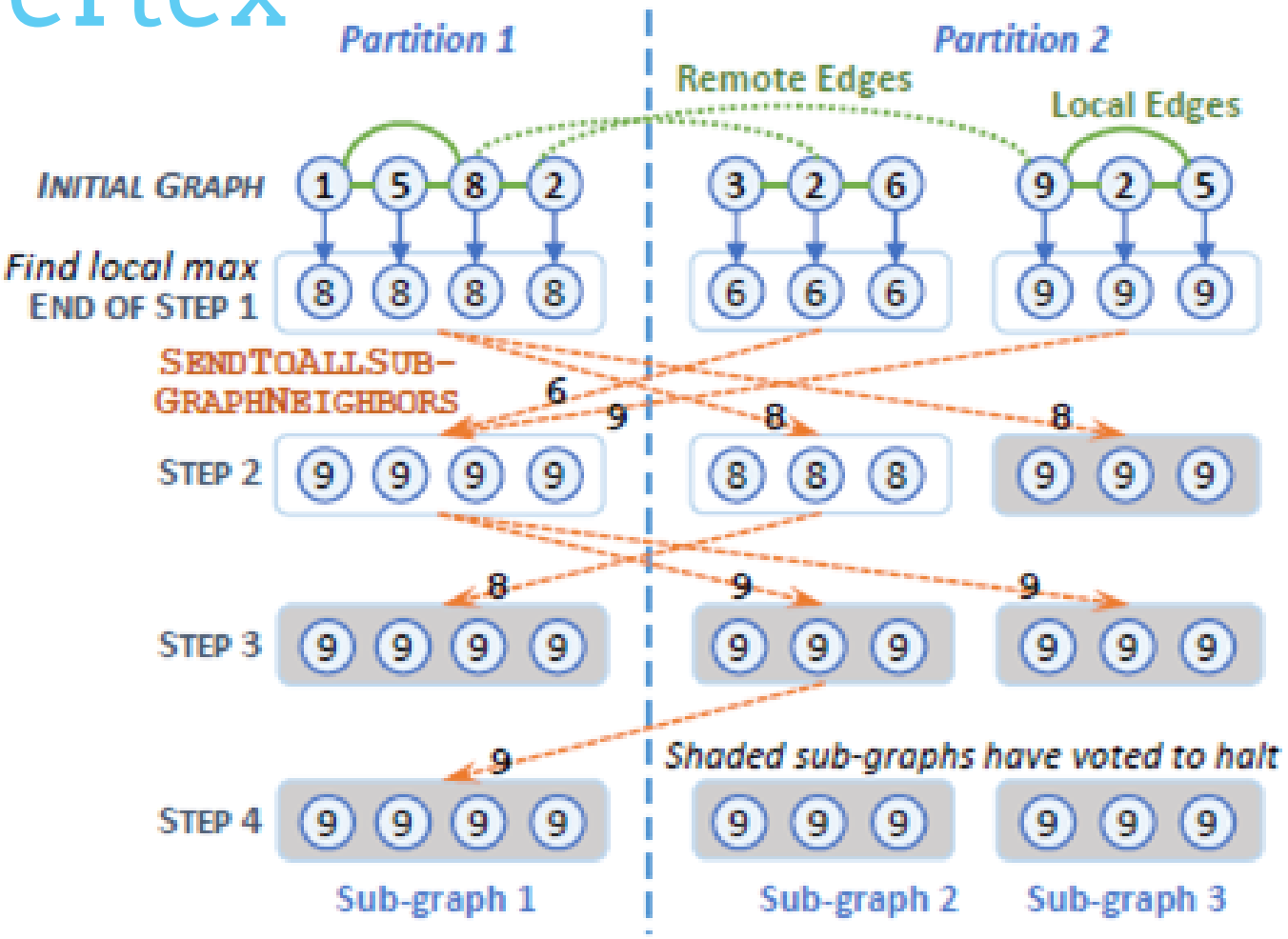
- Write once, read many model: Bulk load instances

Gopher Design

- Vertex-centric graph model
  - *Google Pregel (Apache Giraph)*
  - ☒ Message overhead, dynamic graphs†
- Sub-graph centric streaming dataflows
  - Sub-graphs reduce messaging, more local ops
  - Streaming allows incremental execution
  - Dataflow composition more flexible than BSP
- Giraph++, Blogel use partition & block centric

† "Optimizations and Analysis of BSP Graph Processing Models on Public Clouds", Redekopp, et al, *IPDPS* 2013

Sub-graph Centric Max Vertex

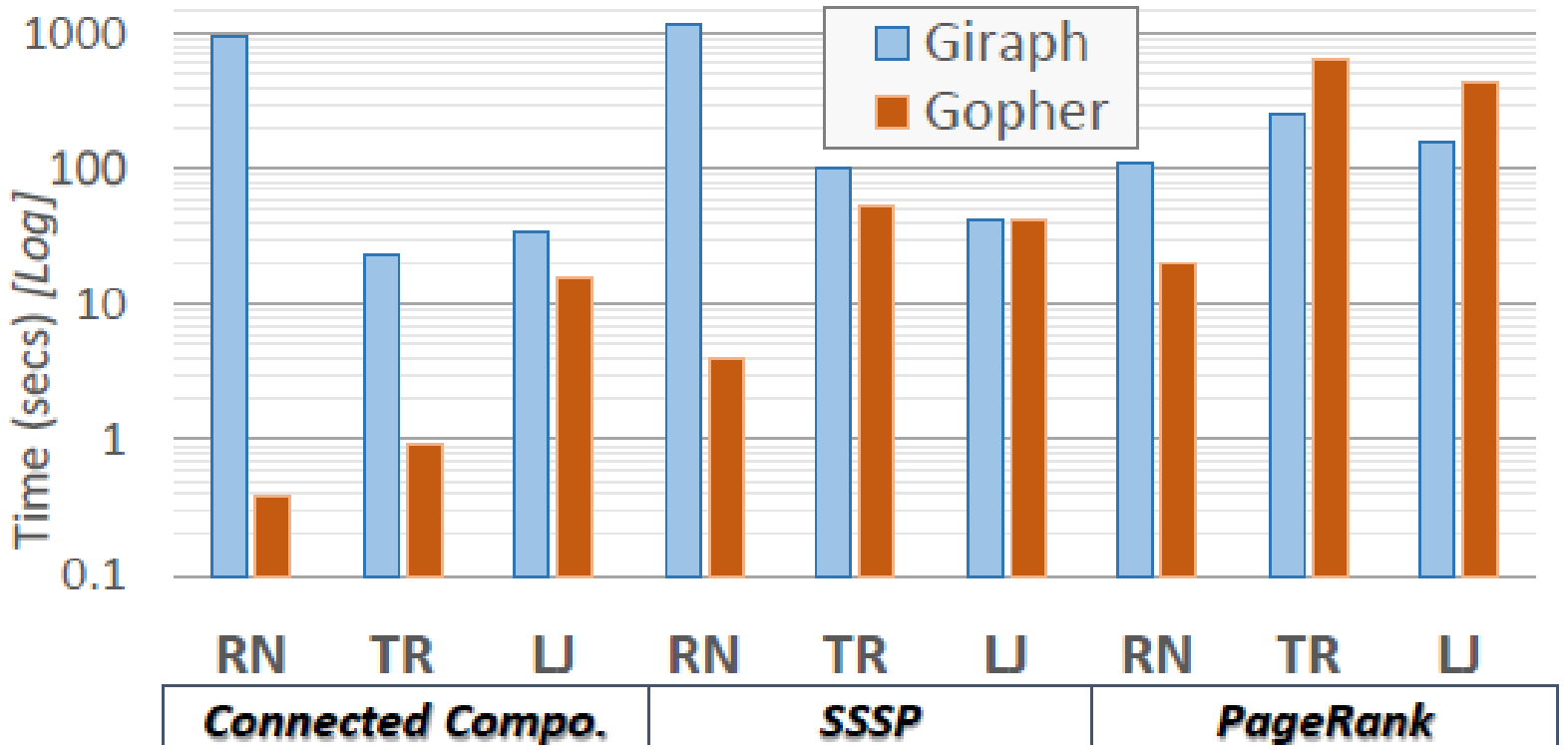


Sub-graph Centric Max Vertex

Algorithm 2 Max Vertex using Sub-Graph Centric Model

```
1:  procedure COMPUTE(SubGraph mySG, Iterator⟨Message⟩ M)
2:      if superstep = 1 then                ▶ Find local max in subgraph
3:          mySG.value ← −∞
4:          for all Vertex myVertex in mySG.vertices do
5:              if mySG.value < myVertex.value then
6:                  mySG.value ← myVertex.value
7:      hasChanged = (superstep == 1) ? true : false
8:      while M.hasNext do
9:          Message m ← M.next
10:         if m.value > mySG.value then
11:             mySG.value ← m.value
12:             hasChanged = true
13:     if hasChanged then
14:         SENDTOALLSUBGRAPHNEIGHBORS(mySG.value)
15:     else
16:         VOTETOHALT( )
```

Performance on Single Graphs

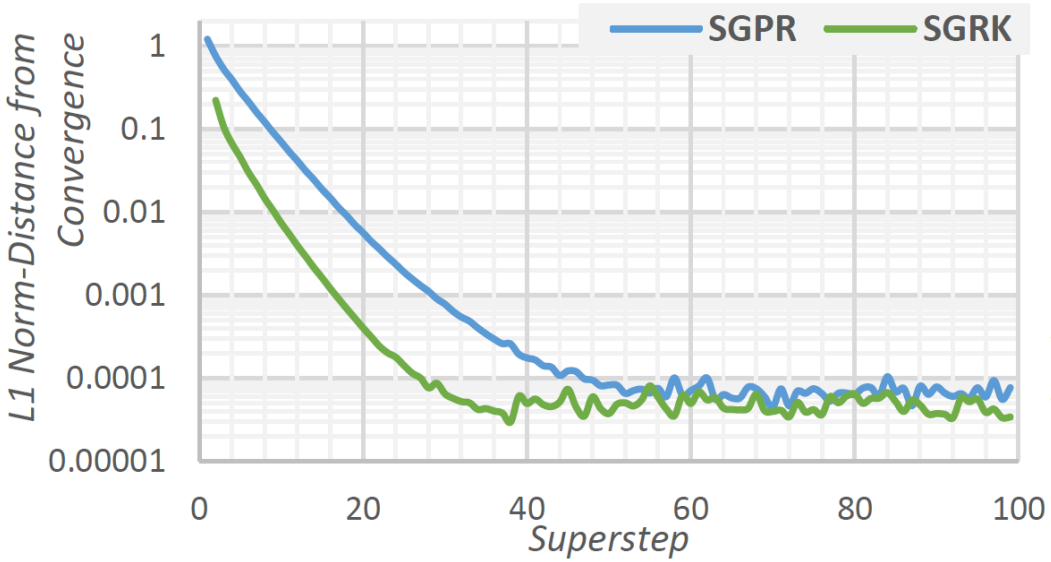


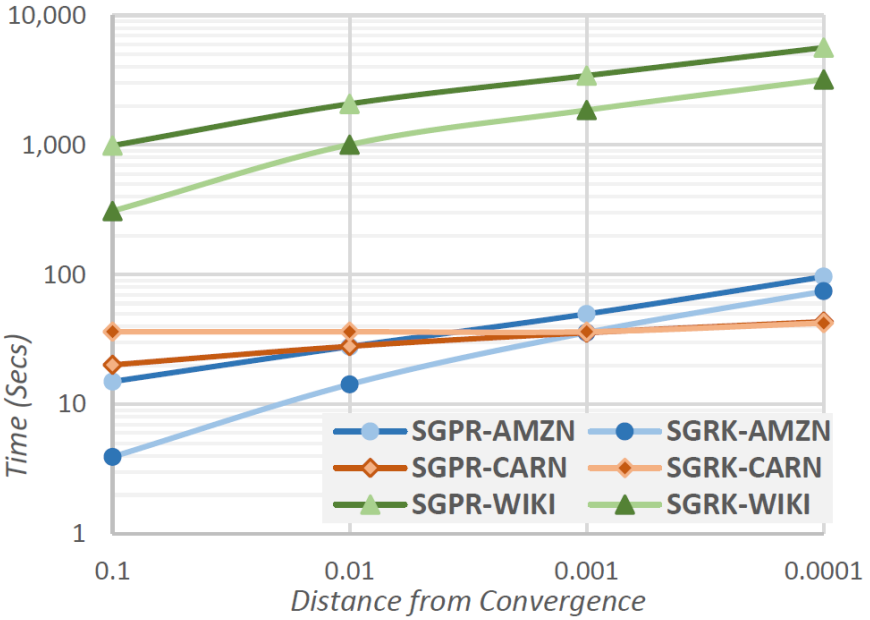
Data Set	Vertices	Edges	Diameter
RN: CA Road Network	1,965,206	2,766,607	849
TR: Internet Tracesroutes	19,442,778	22,782,842	25
LJ: LiveJournal Social N/W	4,847,571	68,475,391	10

GoFFish: A Sub-Graph Centric Framework for Large-Scale Graph Analytics, Simmhan, et al, *EuroPar*, 2014

Algorithmic Benefits on PageRank

- PageRank → Block Rank → Subgraph Rank
  - Coarse-grained rank for "good" initialization

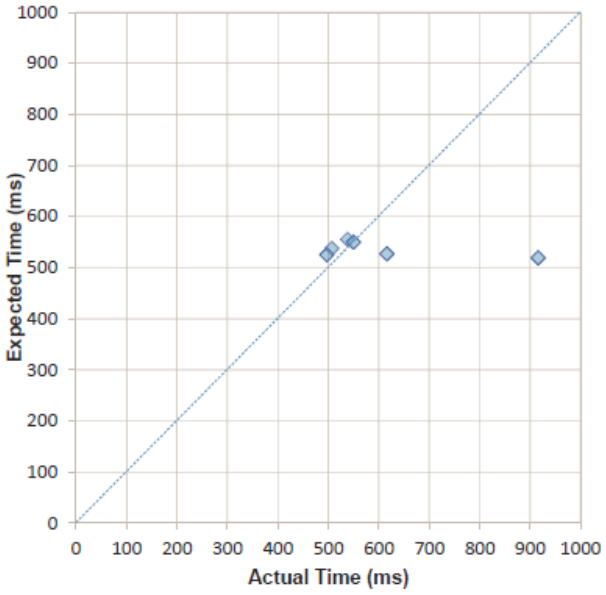




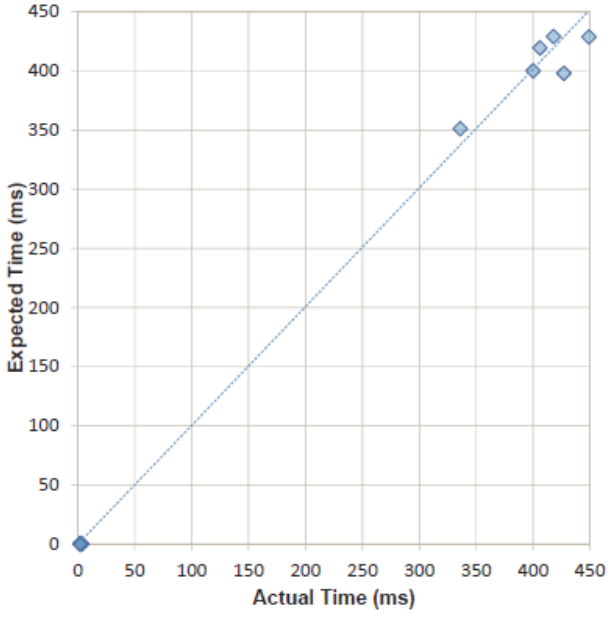
Subgraph Rank: PageRank for Subgraph-Centric Distributed Graph Processing, Badam & Simmhan, *COMAD*, 2014

Analysis of SSSP Algorithm

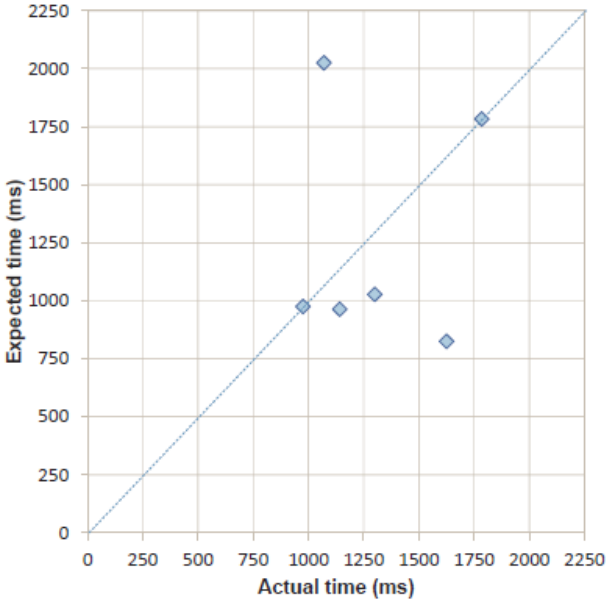
- Modeling & predicting behaviour of SSSP
  - Predictions of Runtime, Optimization heuristics



(a) California Road Network (CARN)



(b) Web Google (WEBG)



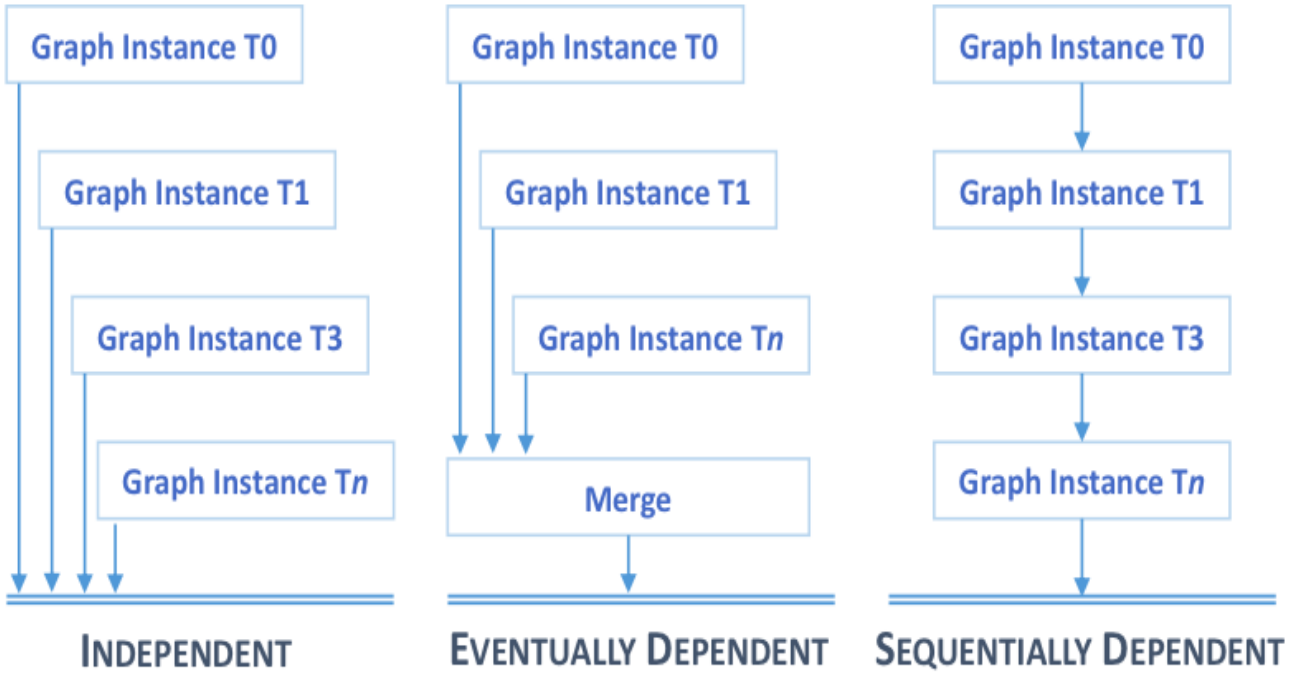
(c) Wiki Talk (WIKI)

Analysis of Subgraph-centric Distributed Shortest Path Algorithm, Ravikant, Neel & Simmhan, *ParLearning*, 2015

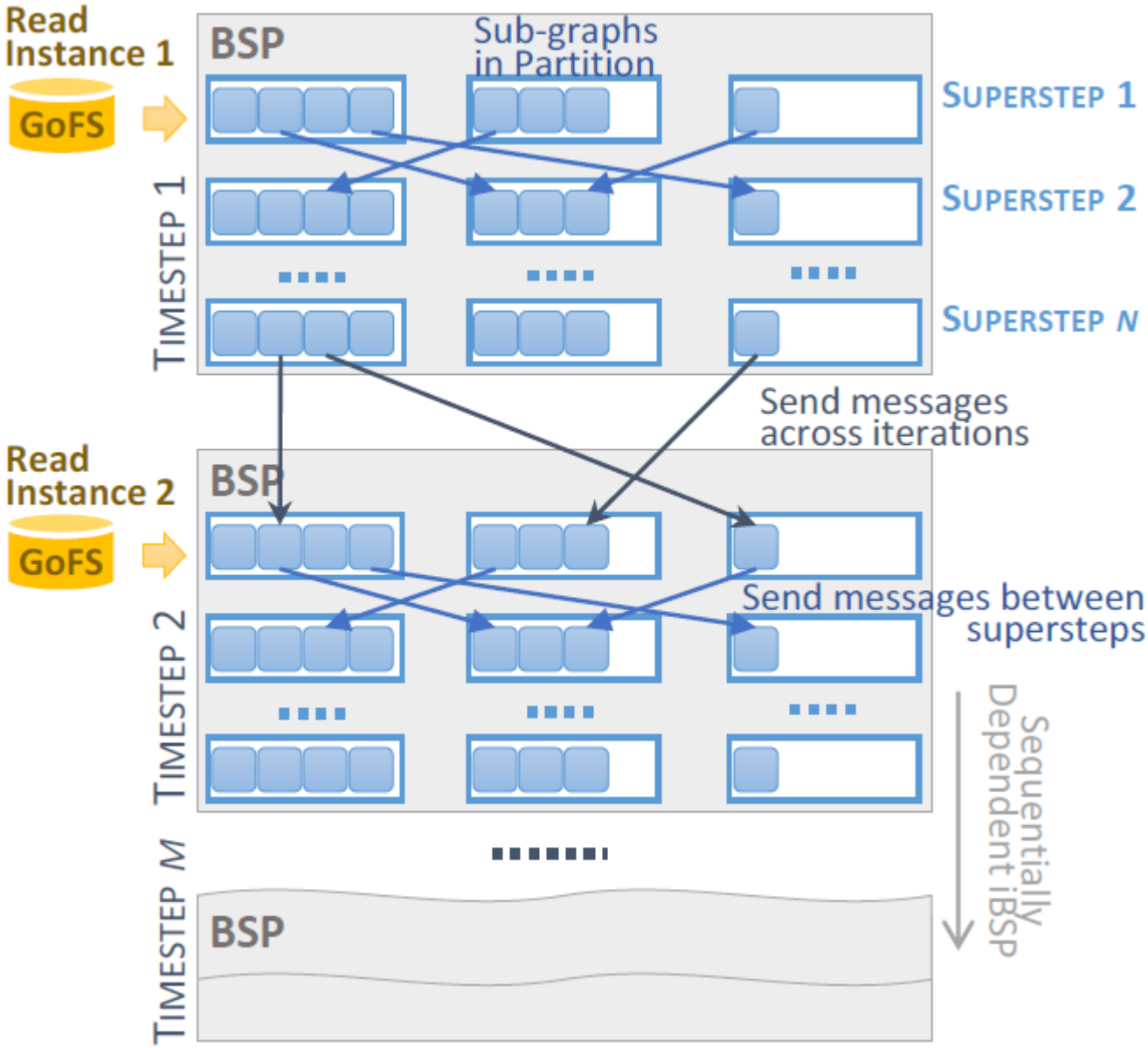
Time Series Graph on GoFFish

TS Graph Programming Model

- Independent Instances
- Eventually dependent Instances
- Sequentially dependent Instances
- Timesteps form “outer loop” of supersteps



Temporally Iterative BSP



User Logic

- **Compute**(Subgraph sg, int timestep, int superstep, Message[] msgs)
- **EndOfTimestep**(Subgraph sg, int timestep)
- **Merge**(SubgraphTemplate sgt, int superstep, Message[] msgs)

Messaging & Termination

- **SendToNextTimestep**(msg)
- **SendToSubgraphInNextTimestep**(sgid, msg)
- **SendMessageToMerge**(msg)

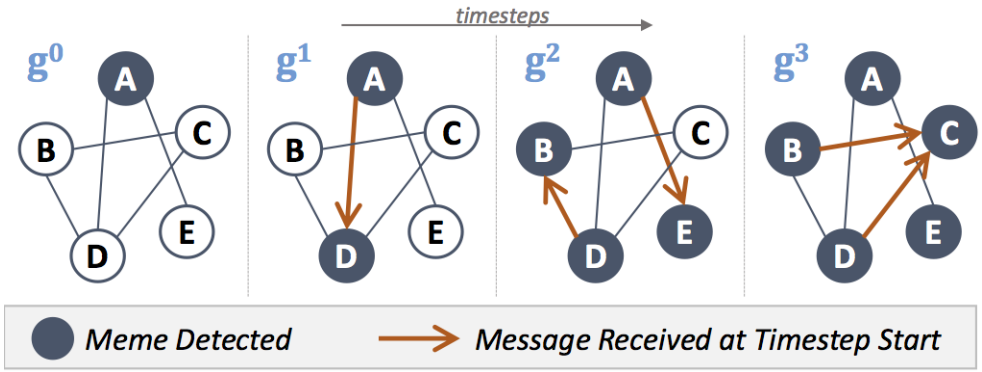
- **VoteToHaltTimestep**()
- **VoteToHalt**()

Hash Tag Aggregation

- We have the structure of a social network and the hashtags shared by each user in different timesteps.
- Find the statistical summary of a particular hashtag: count, rate
- eventually dependent pattern
- Each subgraph calculates the frequency of occurrence of the hashtags among its vertices, sends the result to Merge step.
- In the Merge method, each subgraph receives the messages sent to it from its own predecessors at different timesteps creates a vector from it and send it to largest subgraph in 1st partition.
- In the next superstep, this largest subgraph in the 1st partition aggregates all lists it receives as messages

Meme Propagation Application

- Meme tracking helps analyze the spread of *ideas or memes* (e.g. viral videos, hashtags) through a social network



- Given snapshots of a social network after everyδtime, and each vertex contain memes received between successive intervals
- We need to track a meme μ across the network.
- As meme spread in a short span the superset of structures can be taken as template

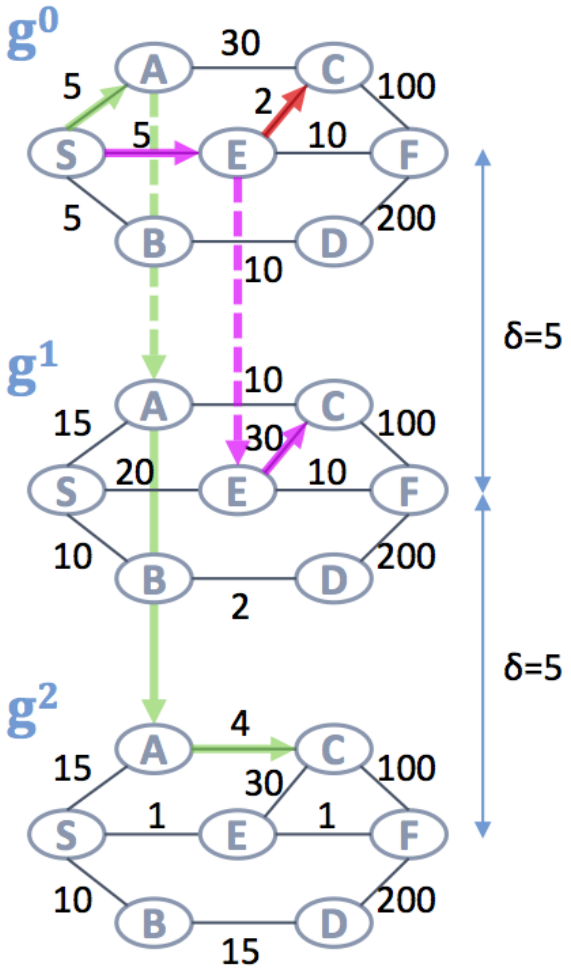
Meme Propagation Algorithm

- BFS across space and time

1. In timestep 0 find all source vertices in $\mathbf{g^0}$. Let the set be $\mathbf{C^0}$
2. In timestep 1 start multisource BFS from $\mathbf{C^0}$ in $\mathbf{g^1}$ to find all vertices touched by meme μ and create set $\mathbf{C}^1$.
   - During BFS, stop at vertices which don't have any neighbor touched by μ
3. In timestep *i* start multisource BFS from all vertices touched in previous timesteps and create $\mathbf{C^i}$
   - Result of one timestep is used as an input for the next time step hence follows sequentially dependent pattern

Time Dependent Shortest Path

- SSSP when edge values changes over time
- Road network in a city
- Data available at discrete time , waiting on vertices allowed.
- SSSP on initial graph gives suboptimal solution. For eg(S→C)
- SSSP on 3-d graph created by stacking instances over each other.
- Edge $\mathbf{v_j^i \rightarrow v_j^{i+1}}$ (idle edge), Unidirectional
- Let ***tdsp[v_j]*** be the final tdsp value from source **s** → $\mathbf{v_j}$



TDSP Algorithm

$$idle[v_j^i] = \begin{cases} \delta & \text{if } tdsp[v_j] \leq i\delta \\ (i+1)\delta - tdsp[v_j] & \text{if } i\delta < tdsp[v_j] \leq (i+1)\delta \\ \text{N/A} & \text{otherwise} \end{cases}$$

1. Start SSSP on g^0 from ***s*** and find all vertices whose shortest path value is less than ***δ*** . Let the set be F^0
2. In g^1 give label ***δ***to all vertices in F^0 and use SSSP to find all vertices whose shortest path value is less than ***2* δ*** (F^1)
3. Similarly in g^i start with all the vertices whose tdsp value is already calculated and label them as ***i*δ*** and use SSSP to find vertices having shortest path ***(i+1)*δ***

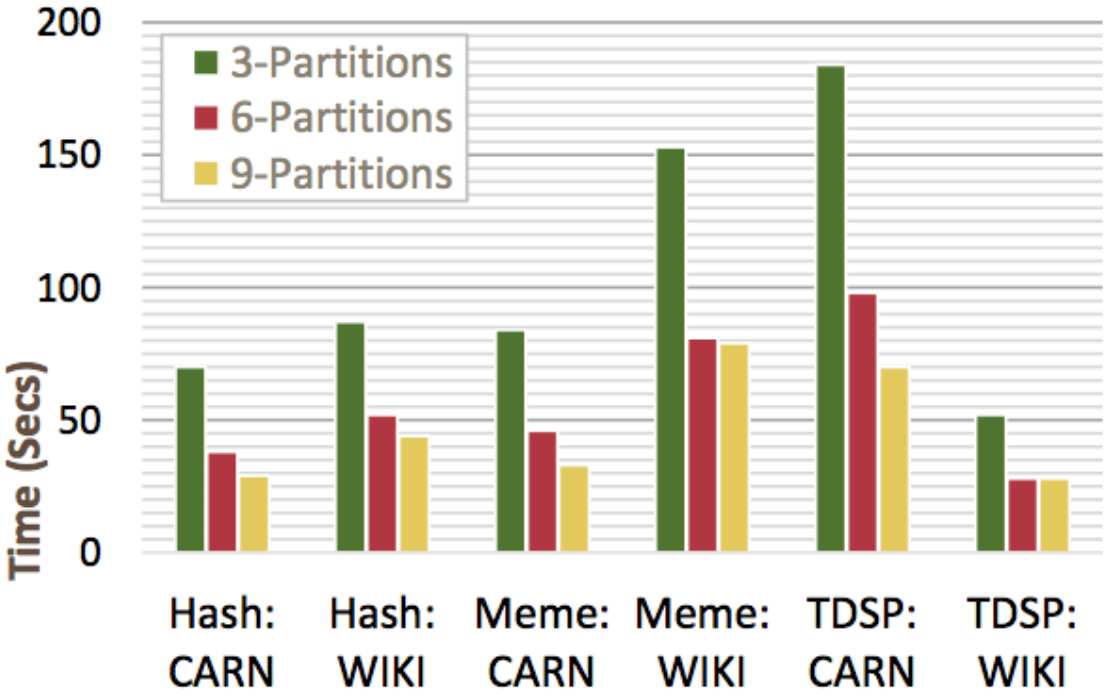
- Result of one timestep is used as an input for the next time step hence follows sequentially dependent pattern

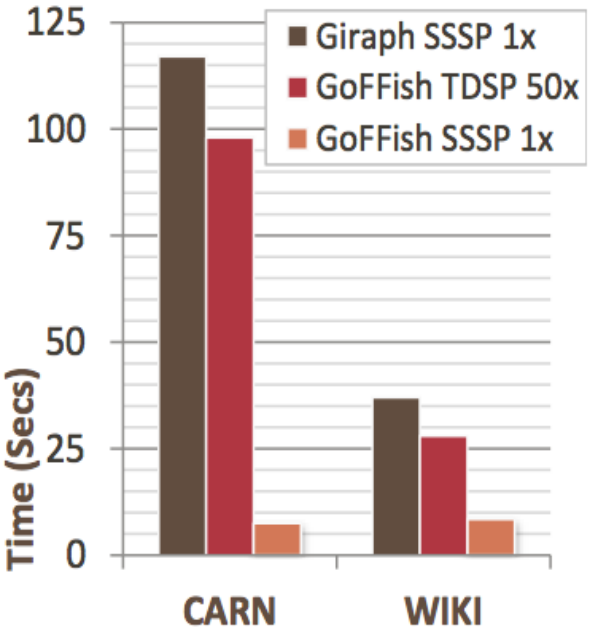
Experimental Setup

Name	No of Vertices	No of Edges	Diameter
California Road Network (CARN)	1,965,206	2,766,607	849
Wikipedia Talk Network (WIKI)	2,394,385	5,021,410	9

- Generated data for 50 timesteps
  - Meme and Hash: SIR Model of epidemiology
    - TDSP: Random weight for edge latencies
- Ran 3 algorithms on two graphs partitioned on 3,6,9 machines
- Partitioning: Metis(min edge cut, load factor 1.03)
- Amazon AWS, m3.large(dual core, 7.5GB RAM, 100 GBSSD)

Results: Makespan

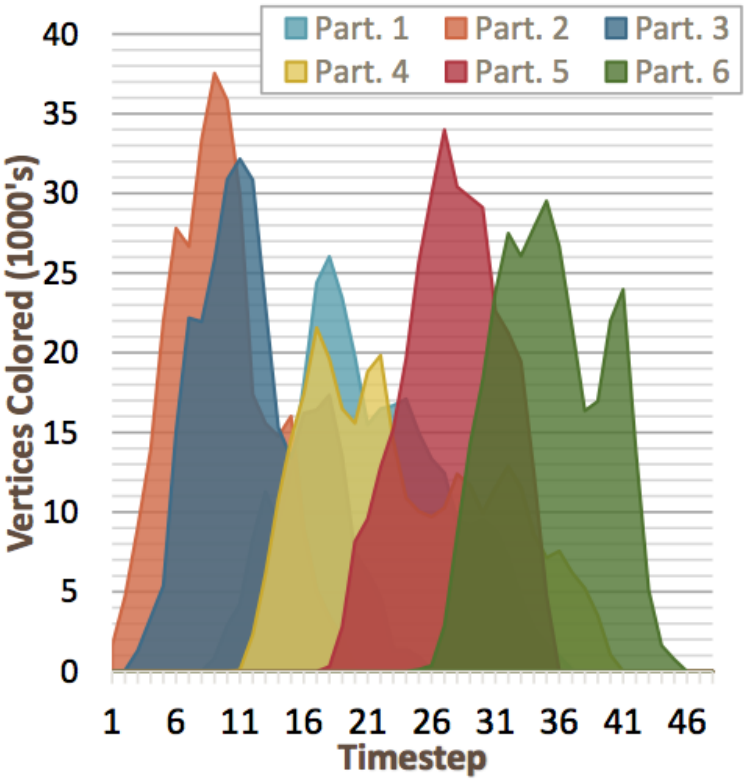


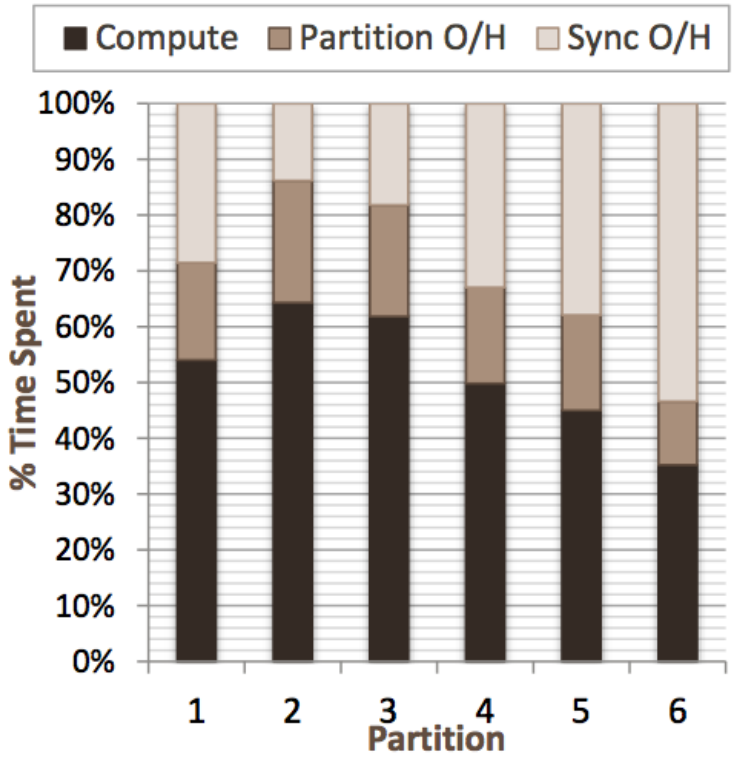


- CARN more scalable than WIKI (larger diameter, better partitioning)
- Baseline comparison with Apache Giraph
  - Giraph SSSP on one time step worse than TDSP in GoFFish in 50 timestep

Y. Simmhan, N. Choudhury, et al. Distributed programming over time-series graphs, IEEE IPDPS 2015

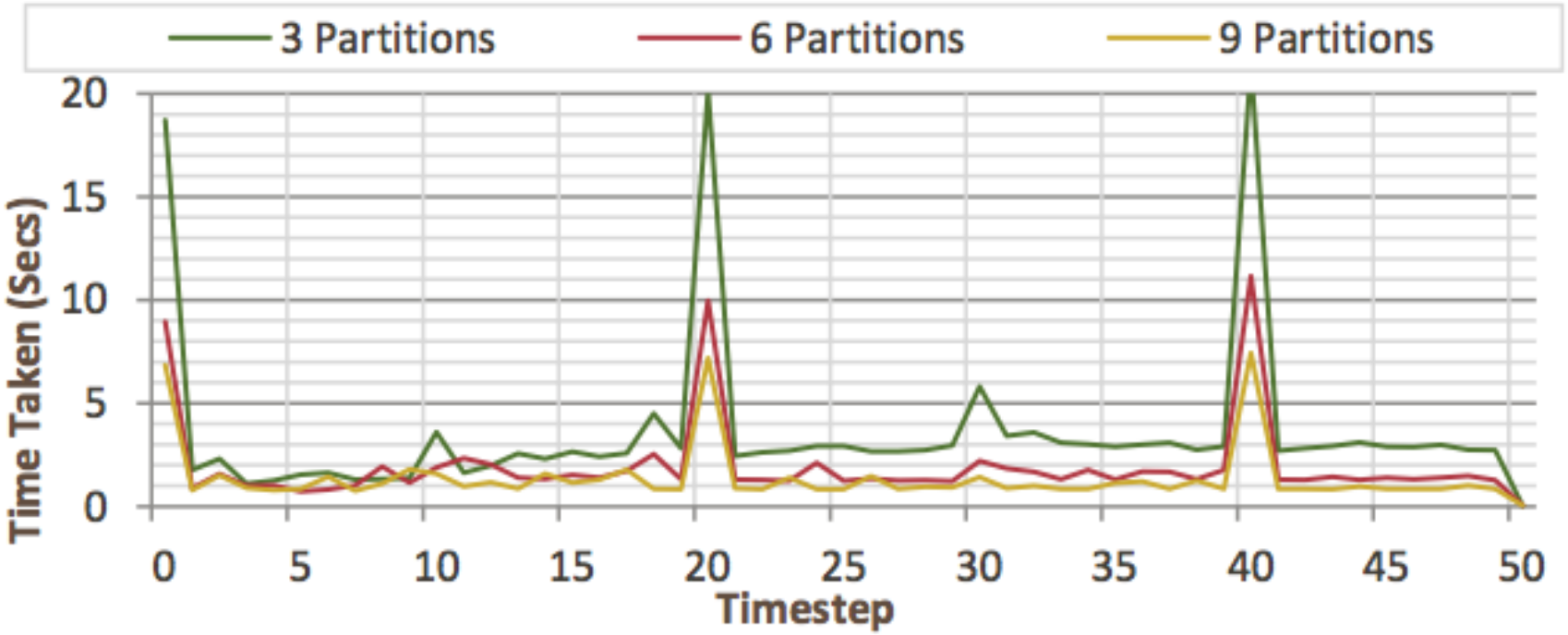
Results: Profiling TDSP for CARN





- Uneven distribution of load across timesteps
- Partition 6 not touched till 26th timestep
- Effect visible in CPU utilization

Results: Time across timesteps



- Large spike in every 20th time steps (Forced Garbage Collection)
- Subtle spike in 10th timestep
- Lazy loading of slices with load factor 10
- Average time for 3 much larger than 6 and 9

Research Problems

1. New vertex/subgraph-centric graph & time-series graph algorithms
2. Better partitioning of subgraph tasks to improve utilization, makespan
3. Realtime scheduling of subgraph tasks for runtime balancing of load
4. Graph databases for querying using Giraph, GoFFish
5. Managing dynamism in graph structure

Other Distributed Graph Platforms

- GraphLab, CMU [1]
- GraphX, UC Berkeley [2]
- Trinity, Microsoft Research [3]

[1] Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud, Low, et al, *VLDB*, 2012
[2] GraphX: Graph Processing in a Distributed Dataflow Framework, Gonzalez, et al, *USENIX OSDI,* 2014
[3] Trinity: A distributed graph engine on a memory cloud, B Shao, et al, *ACM SIGMOD*, 2013

Reading

- Pregel: A System for Large-Scale Graph Processing, Malewicz, et al, *SIGMOD* 2010
- GPS: A Graph Processing System, Salihoglu and Widon, *SSDBM*, 2013
- GoFFish: A Sub-Graph Centric Framework for Large-Scale Graph Analytics, Simmhan, et al, *EuroPar*, 2014