

Large-Scale Graph Processing 1: Pregel, Apache Hama/Giraph & Others

Shiow-yang Wu (吳秀陽)

CSIE, NDHU, Taiwan, ROC



© Can Stock Photo - csp15204367

Outline

- Challenges of big graphs
- Problems with MapReduce on large graphs
- **Pregel**: A system for large scale graph computing
 - Computing model
 - Architecture
 - Fault-tolerance
- Mizan
- **Apache Hama**
- **Apache Giraph**



© Can Stock Photo - csp15204367

Graph-based Modeling



- People, devices, processes, and other entities have been more **connected** than at any other point in history.
- **Graph** is a natural, neat, and flexible structure to model the complex **relationships, interactions, and interdependencies** between objects.



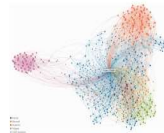
Importance of Graphs



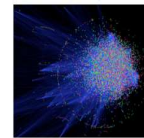
- Graphs abstract application-specific algorithms into generic problems represented as interactions using **vertices** and **edges**



Max flow in road network



Ranking in social networks



Simulating protein interactions

- Algorithms vary in their computational requirements

Graph-based Applications



- Graphs have been used in a wide range of applications.



Social Networks



Web Graph



Map Networks



Chemical and Bio. Networks



Computer Networks



Financial Networks

Graph Algorithms

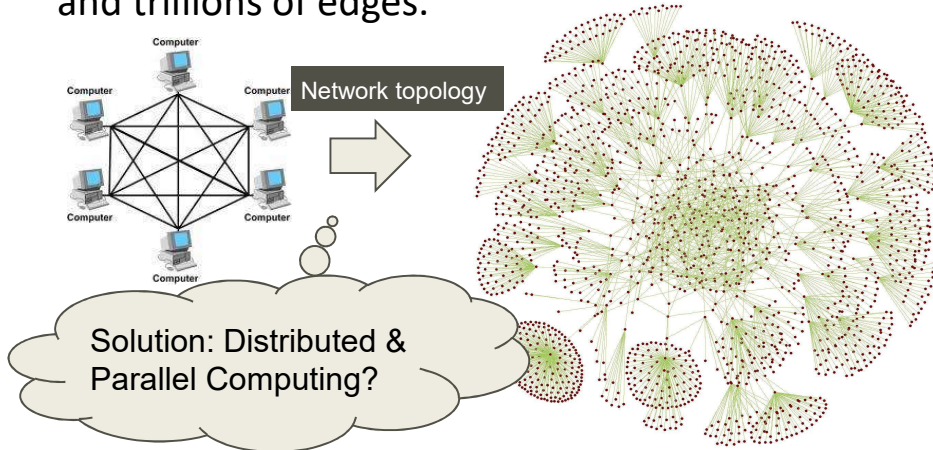


- Algorithms for solving problems on graphs
 - Graph Coloring, Route Problem, Network Flow...etc.
- Many modern applications can be reduced to graph problems
 - Network load balancing, PageRank, Shortest Path...
- Many (approximation) algorithms with polynomial time complexity have been developed to solve the problems.
 - They should be able to solve the problem “efficiently”

Problems: when the graph is BIG



- Graphs become very large, billions of vertices and trillions of edges.



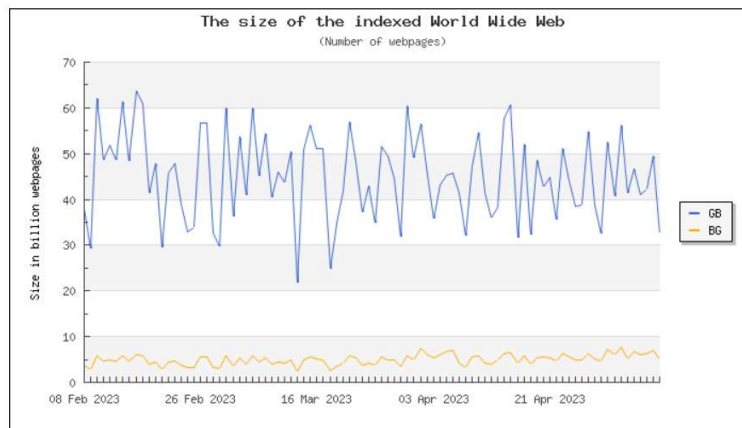
CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 7

Examples of Real-World Big Graph



- World Wide Web is probably the largest graph if pages and links are modeled as vertices and edges



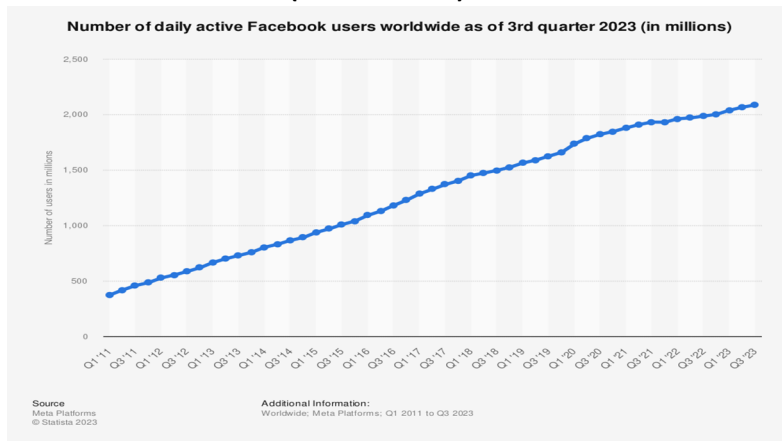
CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 8

Examples of Real-World Big Graph



- Over 3.049/2.09 billion monthly/daily active Facebook users (Q3, 2023).



CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 9

Examples of Real-World Big Graph



- As of 18 December 2023, there are 6,758,734 articles in the English Wikipedia, 59,609,200 Wiki pages with even more links between them. Consider modeling each pages/links as vertices/edges.
- In Q3 of 2023, there are 482 million Pinterest monthly active users, over 240+ billion Pins and much more Related Pins links between them.
- ...

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 10

Problems: when the graph is BIG (2)



- Common problems in large-scale graph computing in distributed environments:
 - Poor locality of memory access
 - Little work per vertex
 - Changing degree of parallelism
 - Running over many machine makes those problems even worse

Existing Solutions and Limitations



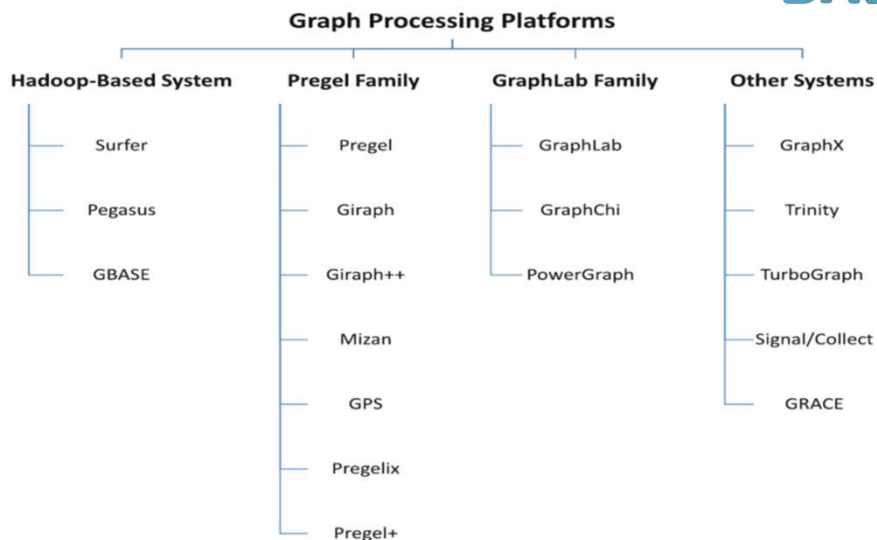
- Custom distributed infrastructure
 - Design a model and system for each graph algorithm.
 - Too expensive
- Distributed computing platforms
 - e.g. MapReduce
 - Often ill-suited for graph algorithms that often have:
 - Many iterations
 - Many intermediate results
 - Too many unnecessary operation(e.g. Disk I/O)

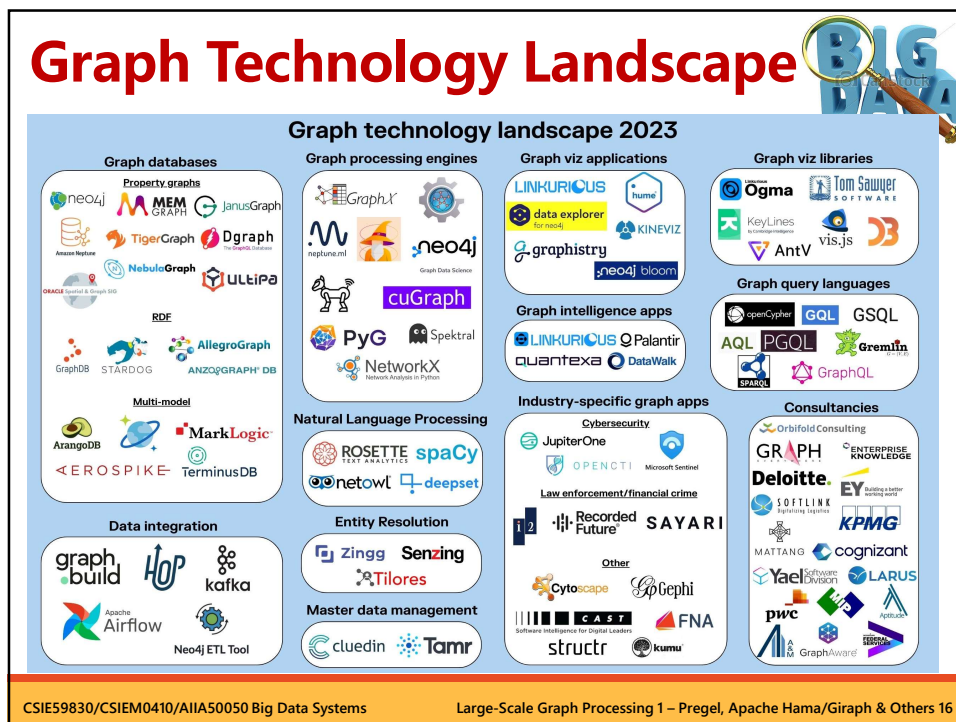
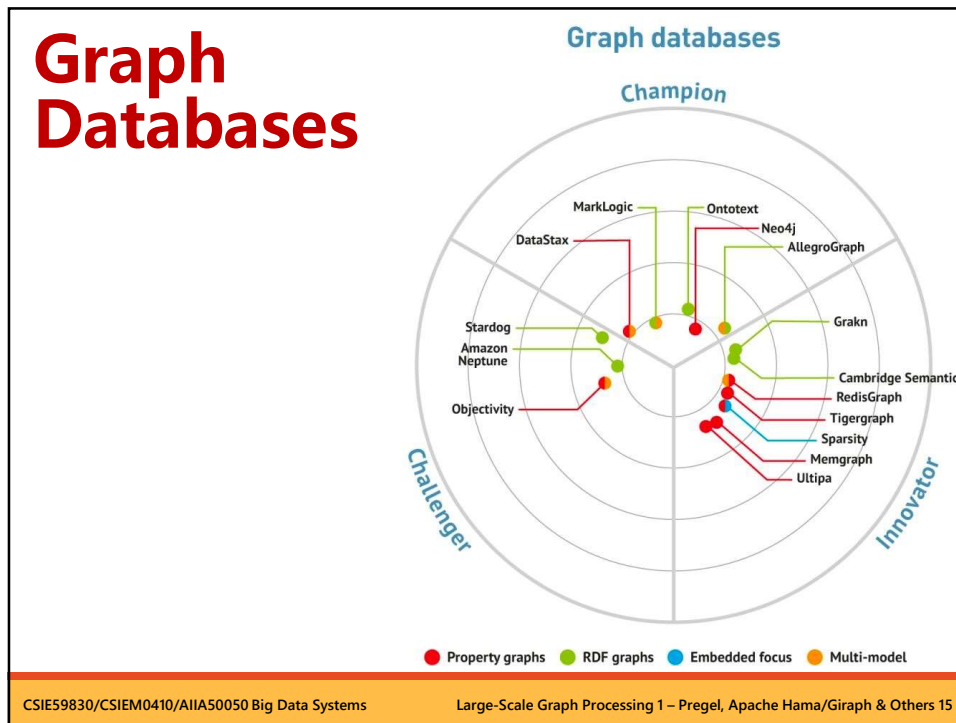
Existing Solutions and Limitations(2)



- Single-computer graph algorithms (libraries)
 - e.g. BGL, LEDA, NetworkX, JDSL, GraphBase, ...
 - **Not scalable**, cannot afford big data
- Parallel graph systems
 - e.g. Parallel BGL, CGMgraph, ...
 - **No fault tolerance**
 - **Lack elasticity**

Graph Processing Platforms

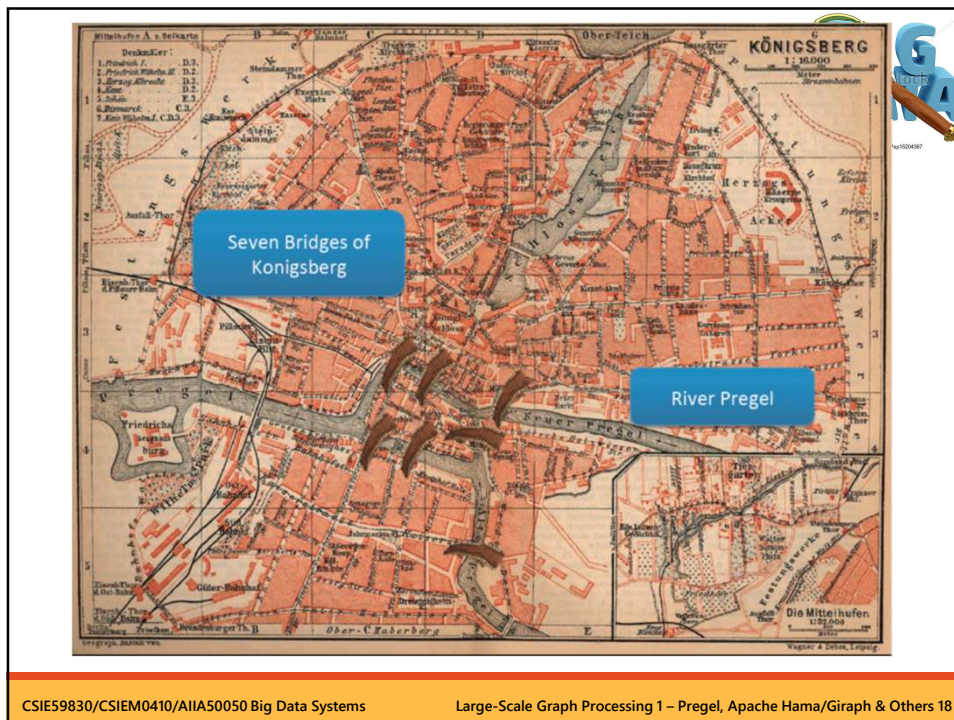
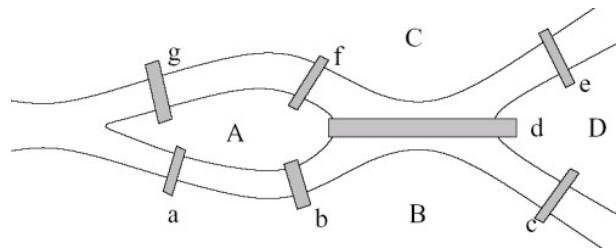




Pregel



- The name comes from shortest path Euler's circuits.
- The **Bridges of Königsberg**, which inspired his famous theorem, spanned the **Pregel** river



How to Scale Graph Processing



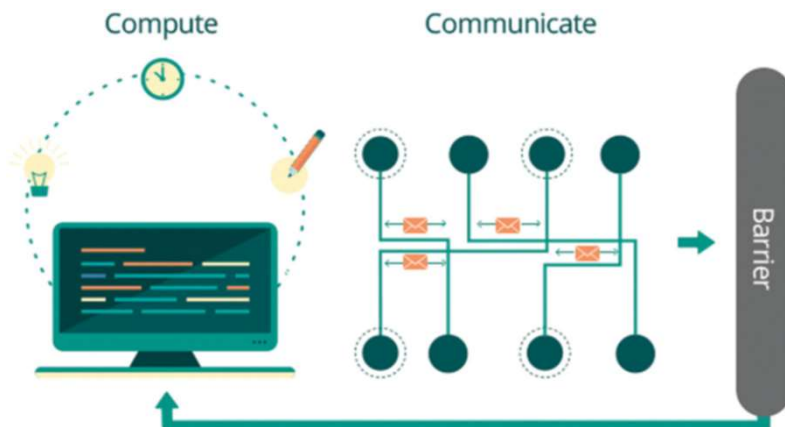
- Pregel was introduced by Google as a scalable abstraction for large-scale graph processing
 - Overcomes the limitations of processing graphs on MapReduce
 - Based on **vertex-centric** computation
 - Utilizes **bulk synchronous parallel (BSP)**

System	Programming Abstraction	Data Exchange
MapReduce	Map(), Reduce()	Key based grouping
Pregel	Compute(), Aggregate()	Message passing


Overview of BSP Model



- In BSP, computation is modeled as a **loop** of **three-stage** processing.




Computation Model

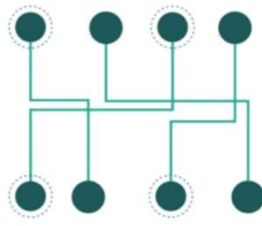


- **Bulk Synchronous Parallel (BSP) model**
 - Bridging model for designing parallel algorithms.
 - **Super steps**
 - ▣ Vertices **compute** in parallel
 - ▣ Messages can be **exchanged** at the end of each step

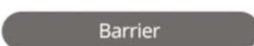
Local Compute



Communication




Barrier Synchronisation



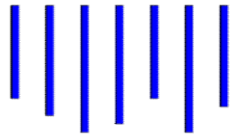
CSIE59830/CSIEM0410/AIIA50050 Big Data Systems
Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 21

Computation Model

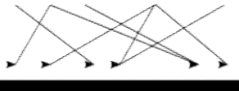


- **Bulk Synchronous Parallel (BSP) model**
 - Bridging model for designing parallel algorithms.
 - **Super steps**
 - ▣ Vertices **compute** in parallel
 - ▣ Messages can be **exchanged** at the end of each step


Local Compute



Communication




Barrier Synchronization

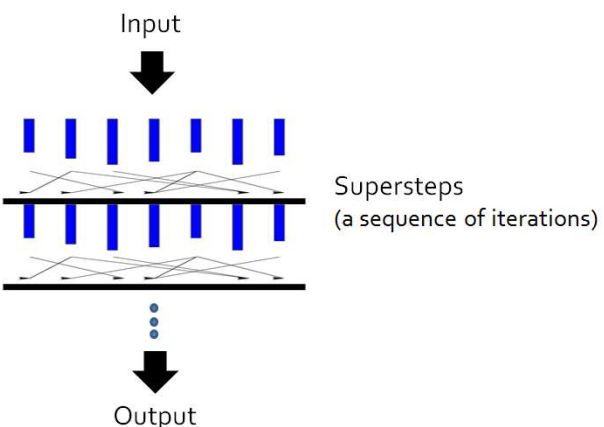


CSIE59830/CSIEM0410/AIIA50050 Big Data Systems
Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 22

Computation Model



- **Messages** sent at superstep N can only be received by superstep N+1




Input

Supersteps
(a sequence of iterations)

Output

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 23

Computation Model



- Every vertex executes the **same** function
- What can a vertex do for each Superstep:
 - **Receives messages** sent in the previous superstep
 - **Modifies its value** or that of its **outgoing edges**
 - **Sends messages** to other vertices
 - **Mutates the topology** of the graph
 - **Votes to halt** if it has no further work to do

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 24

Computation Model



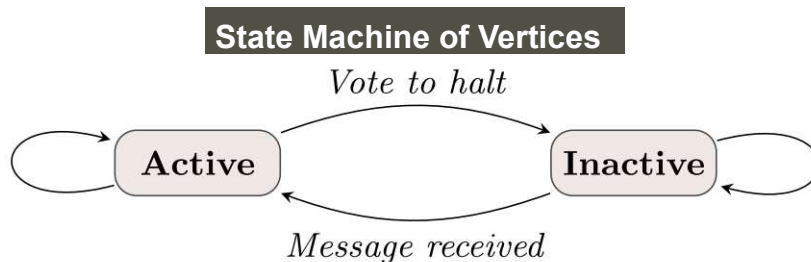
- In vertex-based computation model, each vertex is used to keep **processing state** and **local processing**.
- The **application logic** is represented by the **link structure** and **message exchange** between vertices.



Computation Model



- Vertex computes when **active**
- **Termination condition**
 - All vertices are simultaneously **inactive**
 - There are **no messages** in transit

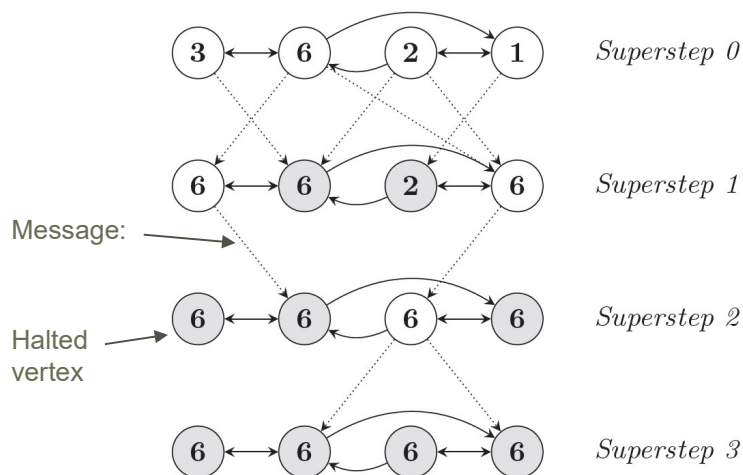


Notes on Computation Model



- During a superstep
 - Receive messages from previous superstep (from the past)
 - Send messages to next superstep (to the future)
 - No interactive communication during rounds
- No deadlocks are possible. Why?
- All vertex-to-vertex operations are totally synchronous
 - Makes fault tolerance simple
- Vertices run their local compute function asynchronously

Example: Maximum Value



C++ API



- Writing a Pregel program

- Subclassing the predefined **Vertex** class

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
public:
    virtual void Compute(MessageIterator* msgs) = 0;

    const string& vertex_id() const;
    int64 superstep() const;

    const VertexValue& GetValue();
    VertexValue* MutableValue();
    OutEdgeIterator GetOutEdgeIterator();

    void SendMessageTo(const string& dest_vertex,
                      const MessageValue& message);
    void VoteToHalt();
};
```

Annotations in the original image:

- An arrow points to `Compute` with the text "Override this!".
- An arrow points to `msgs` with the text "in msgs".
- An arrow points to `dest_vertex` with the text "out msg".

C++ API (2)



Virtual void Compute(MessageIterator* msgs)=0

- The user overrides the virtual Compute() method in each active vertices and every super-step

Constant VertexValue& GetValue();

VertexValue* MutableValue()

- Compute() can inspect the value associated with its vertex via GetValue() or modify it via MutableValue()

OutEdgelterator GetOutEdgelterator();

- It can inspect and modify the out-edges using the GetOutEdgelterator()

Void SendMessageTo(dest_vertex, message)

Void VoteToHalt()

- SendMessageTo() send message to all destination vertex otherwise execute VoteToHalt()

Program Example: Maximum Value



```

Class MaxFindVertex: public Vertex<double, void, double> {
    public:
        virtual void Compute(MessageIterator* msgs) {
            int currMax = GetValue();
            SendMessageToAllNeighbors(currMax);
            for ( ; !msgs->Done(); msgs->Next()) {
                if (msgs->Value() > currMax)
                    currMax = msgs->Value();
            }
            if (currMax > GetValue())
                setValue(currMax);
            else VoteToHalt();
        }
};

```

Send msg. to others

Check msg. queue & do the works

Decide whether to halt

Program Example: SSSP



```

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();
}
};

```

Example: SSSP – Parallel BFS in Pregel

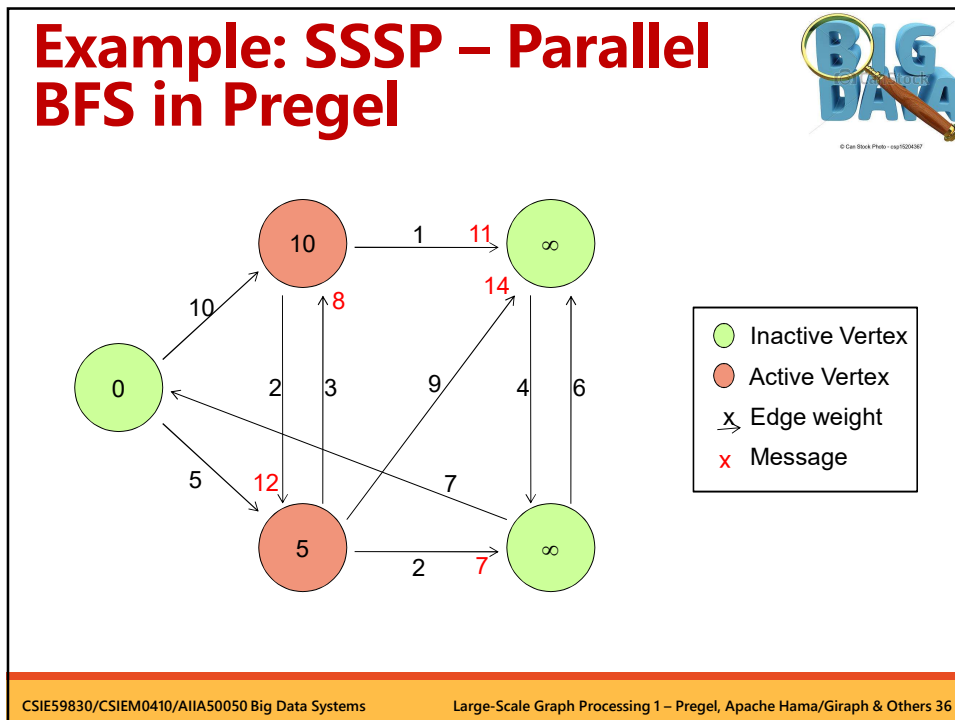
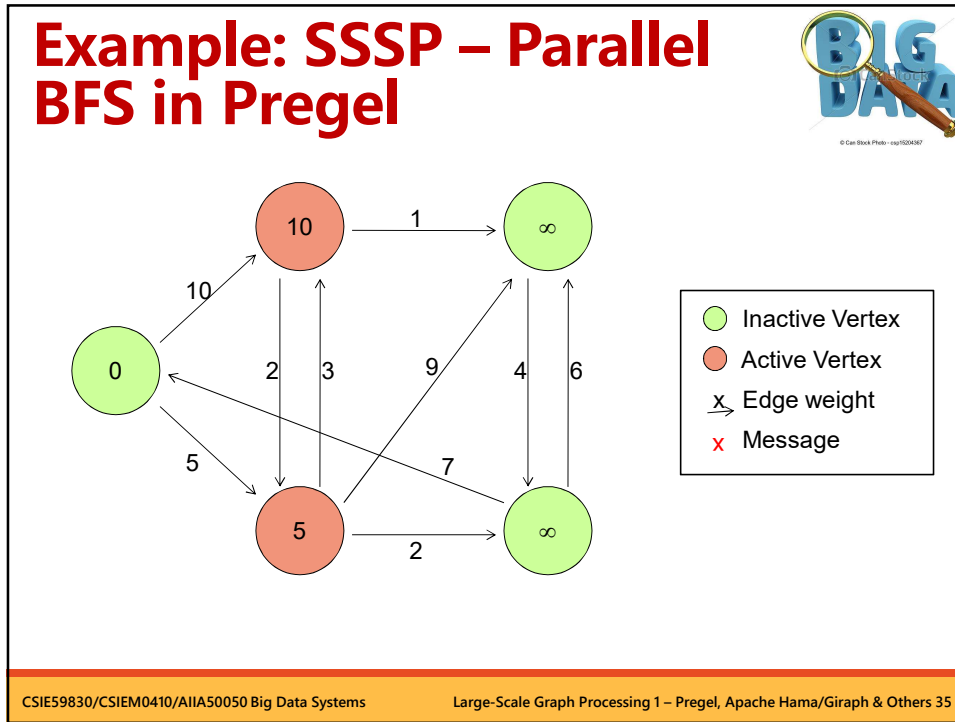
- Inactive Vertex
- Active Vertex
- $\overset{x}{\rightarrow}$ Edge weight
- x Message

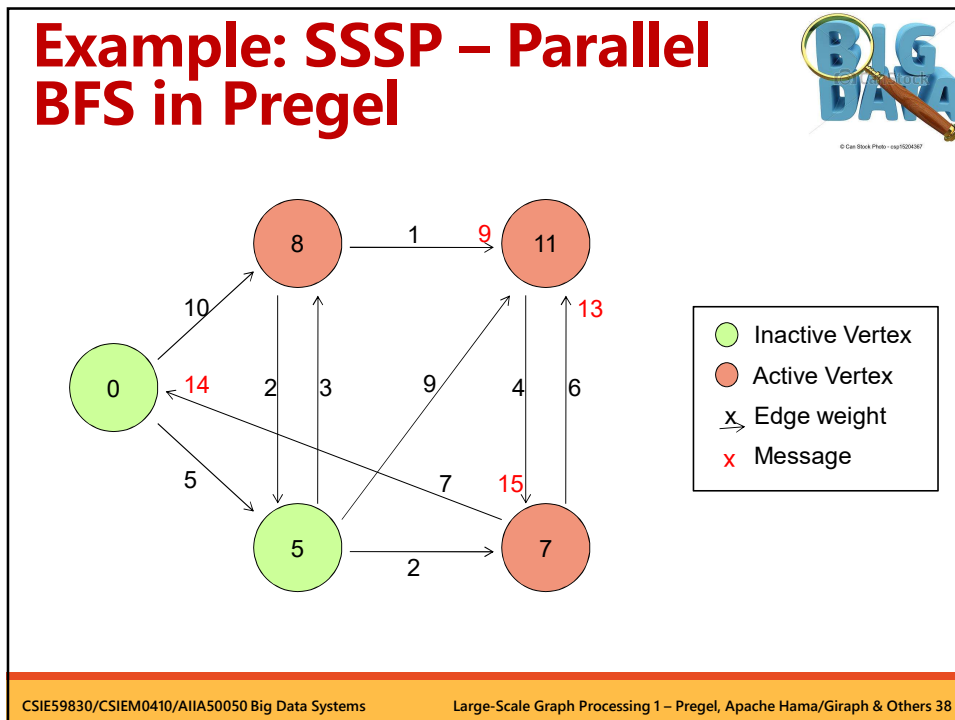
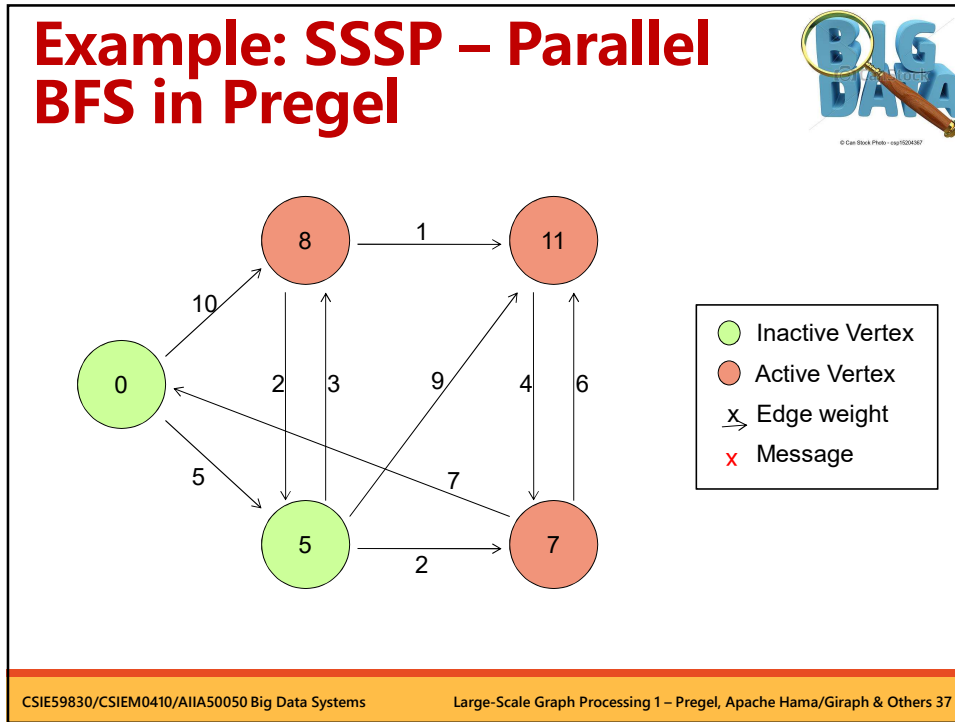
CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 33

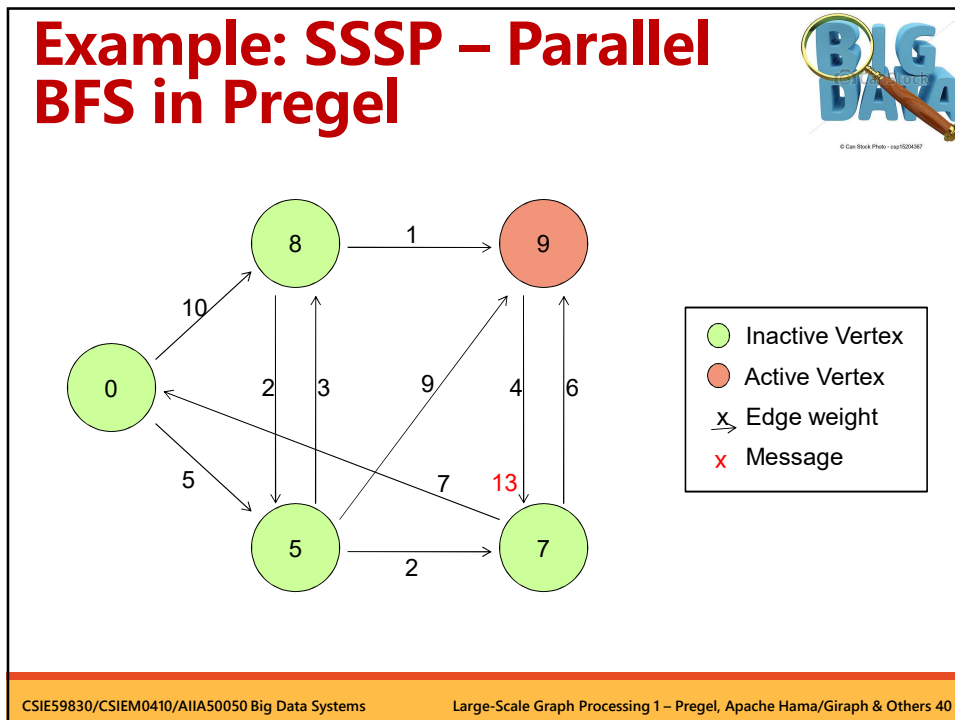
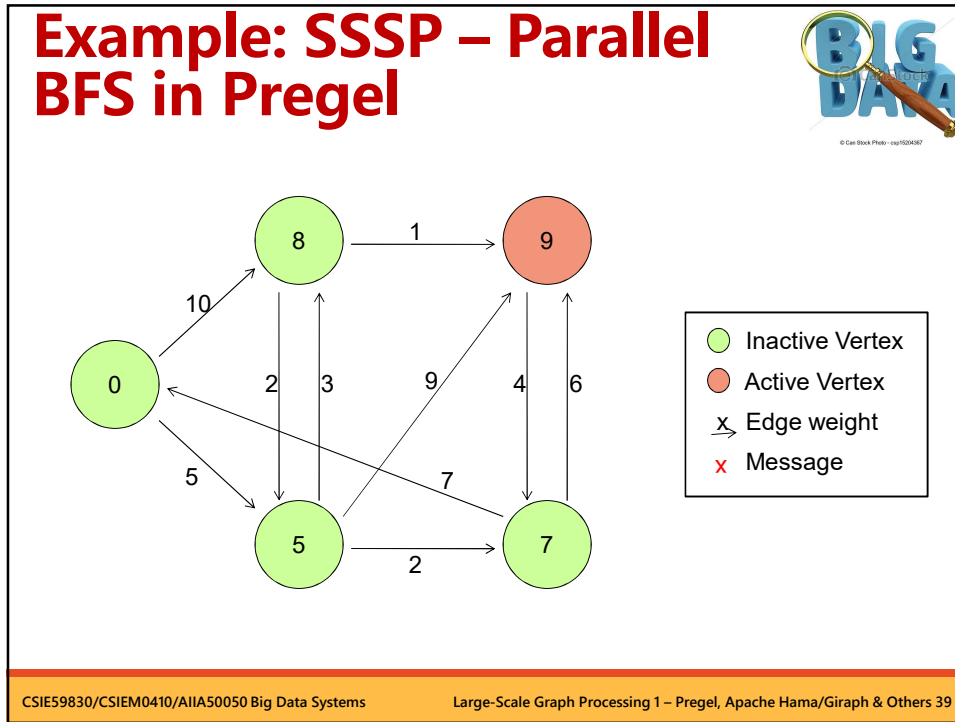
Example: SSSP – Parallel BFS in Pregel

- Inactive Vertex
- Active Vertex
- $\overset{x}{\rightarrow}$ Edge weight
- x Message

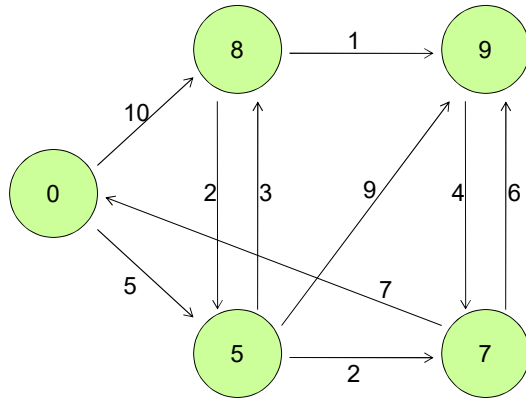
CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 34







Example: SSSP – Parallel BFS in Pregel



- Inactive Vertex
- Active Vertex
- \xrightarrow{x} Edge weight
- x Message

Program Example: 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();
    }
  }
};
```

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

PageRank

BIG DATA
© Cal Stock Photo - iap120437

Courtesy: Wikipedia

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 43

Pregel Architecture

- Pregel consists of Master/Workers on top of GFS.

Google Pregel Architecture

Master: Workers, Global Aggr

Workers in lock-step

Global / Local Aggregation

State: S, S+1, Aggr

Graph Partition: Vertices, Edges-Out

Combiner

Worker

Message passing (dashed arrows)

GFS (persistent storage)

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 44

System Architecture

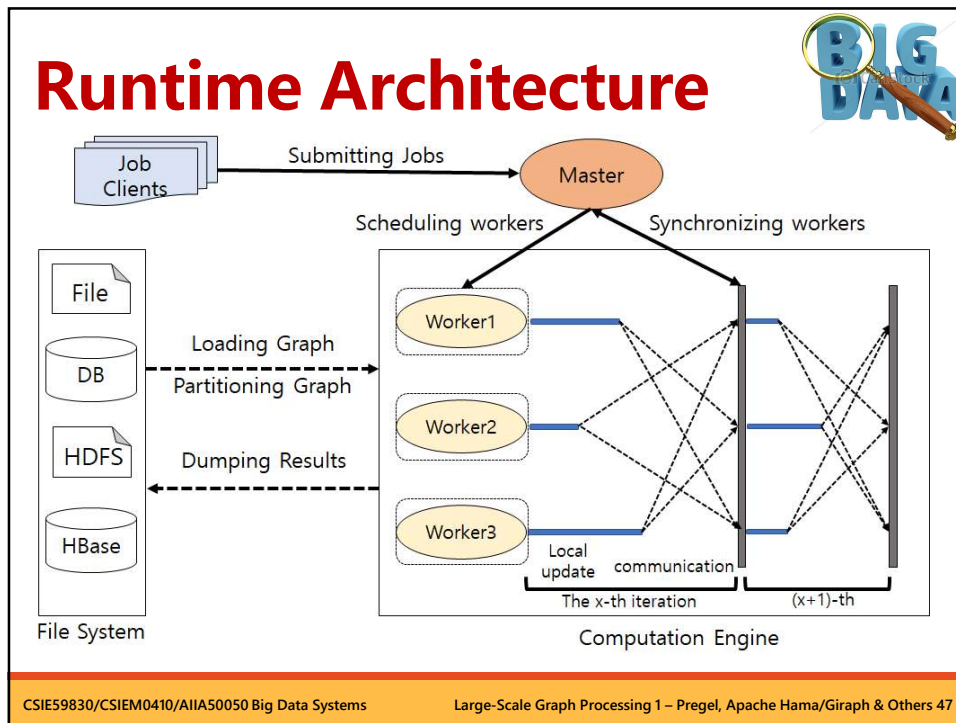


- User program is copied to many machines
- One machine become **Master**
 - Be responsible to coordinate activities
 - Partition the whole graph
 - Assign each partitions to workers
- Others become **Workers**
 - Operate computation
 - Maintain the state of partition(s)

System Architecture (2)



- The input graph is partitioned
 - Each partition has vertices and their out-links
- Each worker has one or many partition(s)
 - Machines are responsible for maintaining the state of vertices in their partitions



Aggregator

- Combine values from vertices for each superstep
 - as a single global value
 - ex: Min, Max, Sum...
 - values of superstep S can be used by $S+1$
- For global communication, monitoring
 - ex: compute the total # of edges in graph
- Can be used to control the flow of execution
 - ex: as a branch condition

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 48

Fault Tolerance

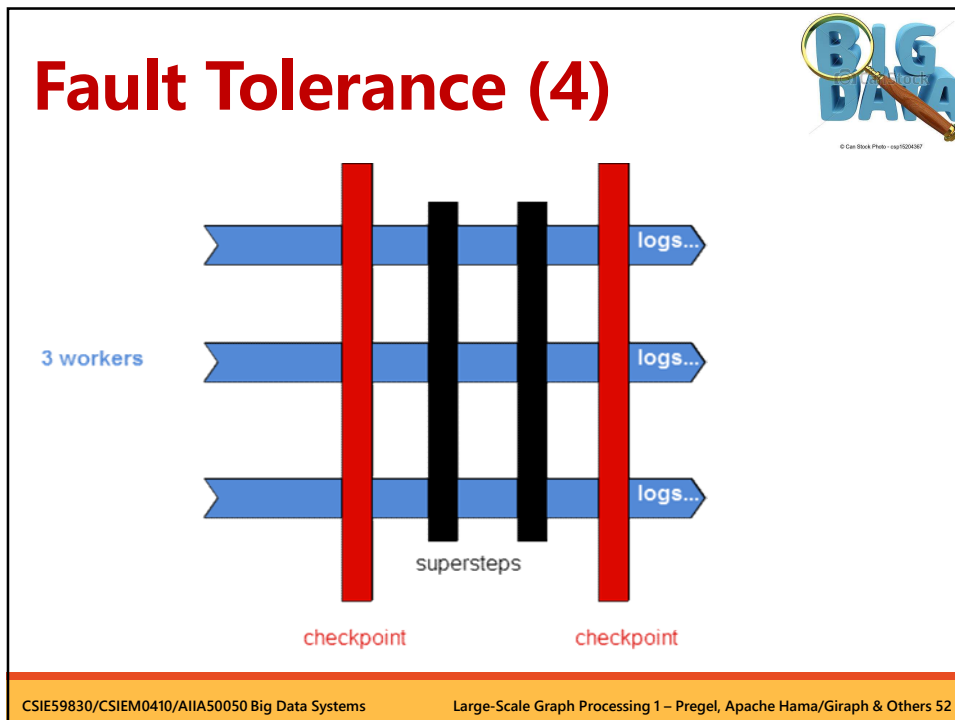
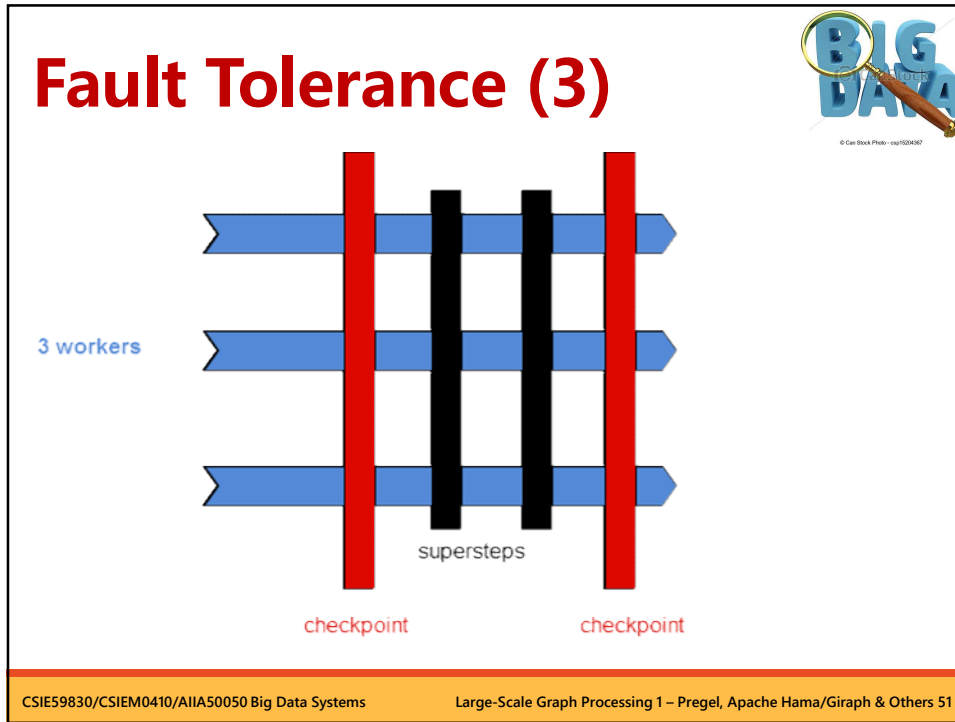


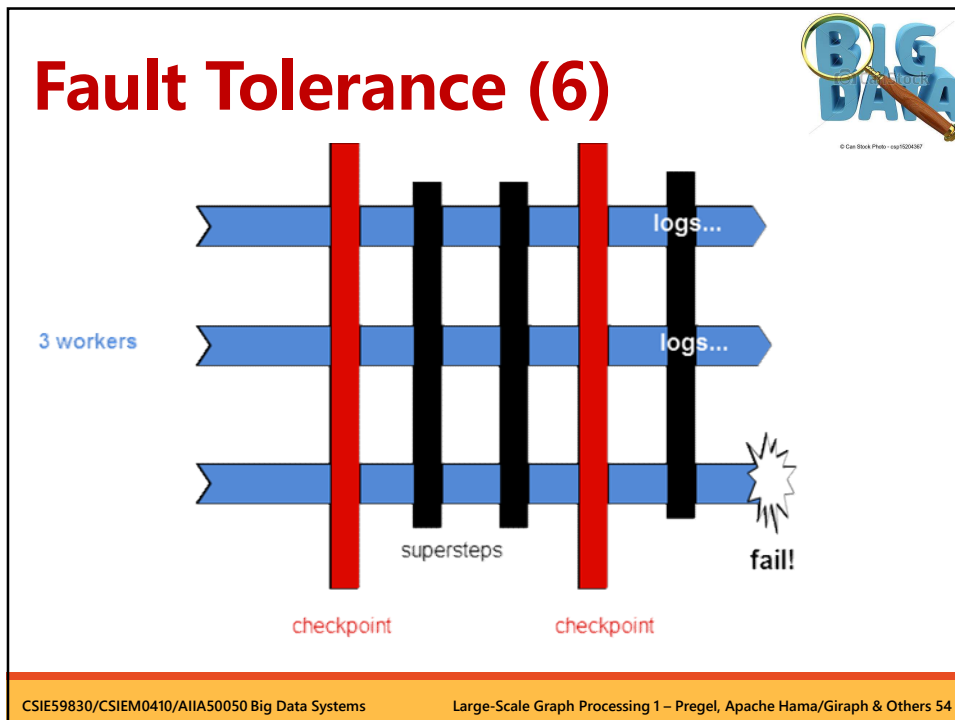
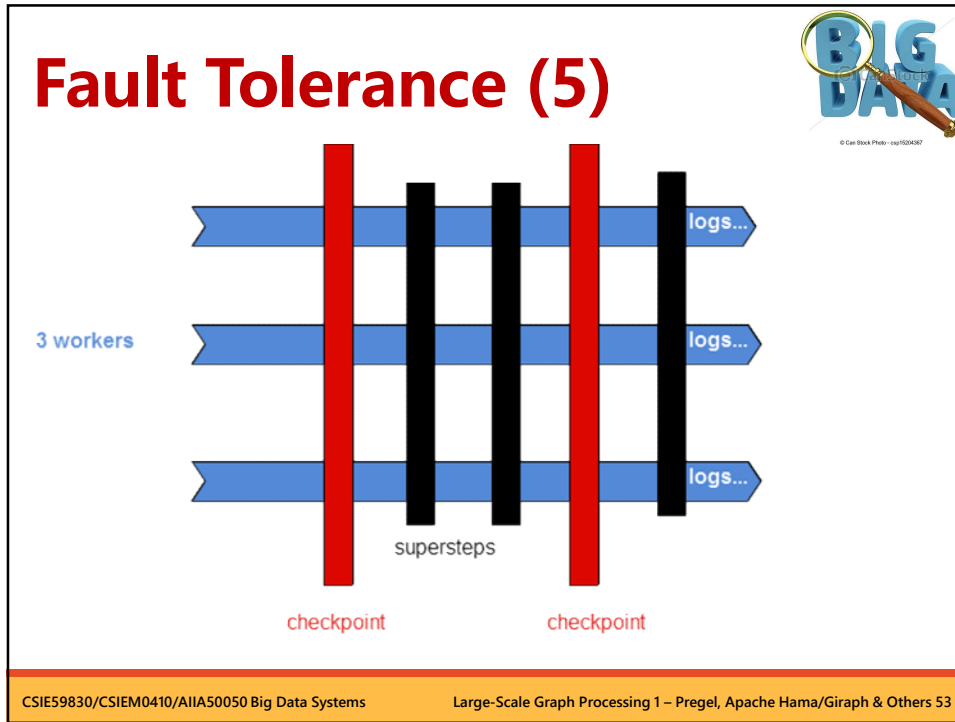
- **Check points** can be set after several supersteps:
 - Worker checkpoint: V, E and Msg
 - Master checkpoint: Aggregator
- **Worker failure**
 - Detected by regular heartbeat
 - All workers start over at most recent available checkpoint
- **Frequency of checkpoint**
 - Need to balance the backup cost against recovery cost

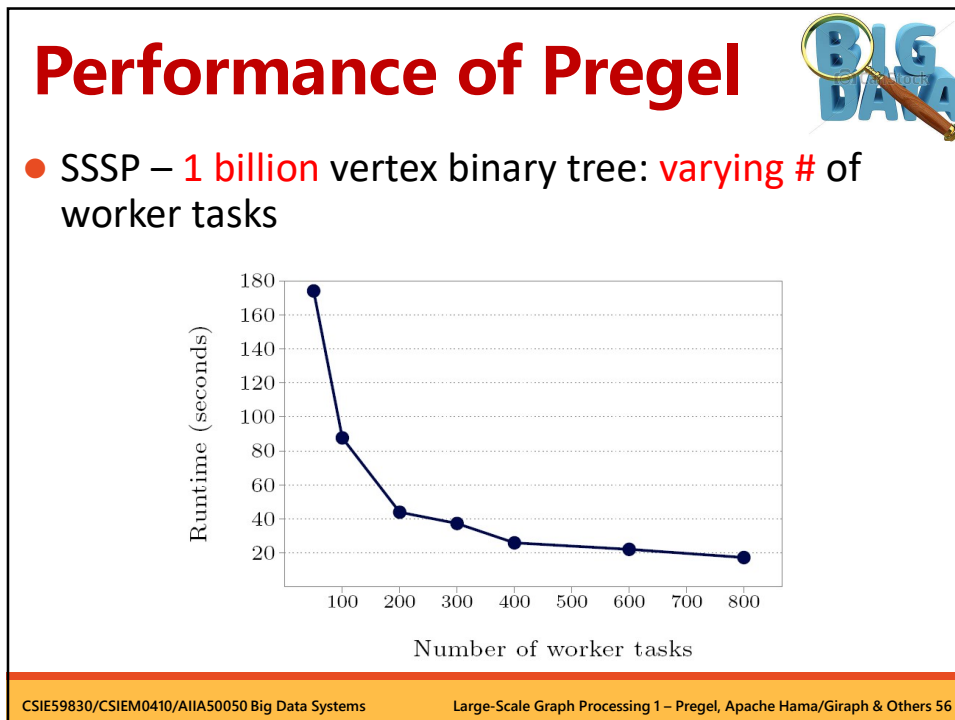
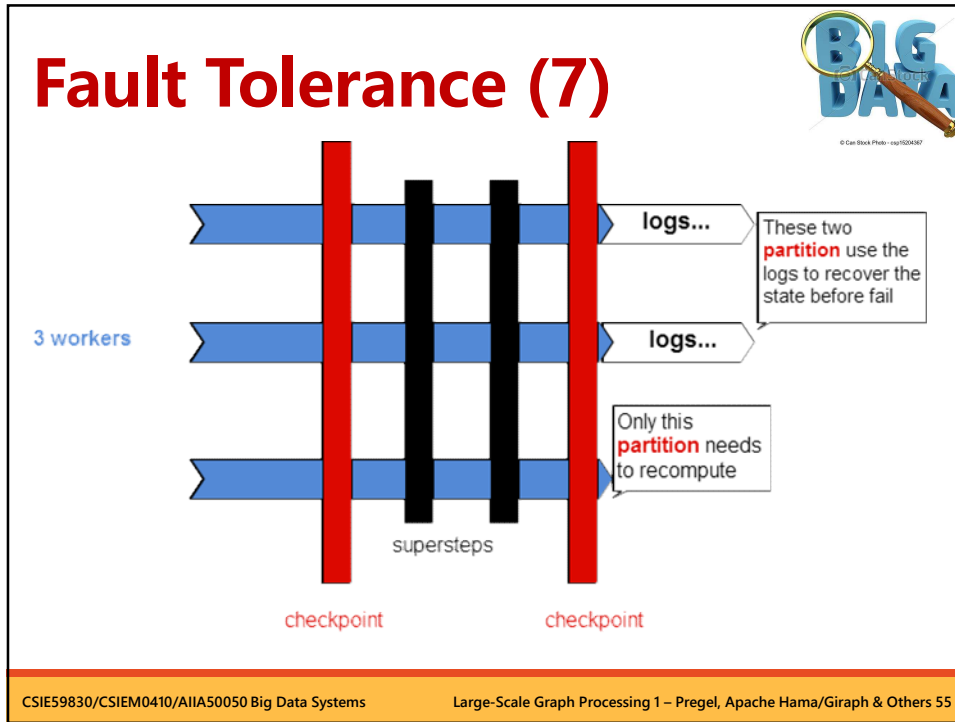
Fault Tolerance (2)



- **Confined recovery**
 - Workers log outgoing Msg from their assigned partitions
 - Only the failed worker need to recompute => Save compute resource (e.g. network overhead from communication)
=> Improve the cost and latency from recovery



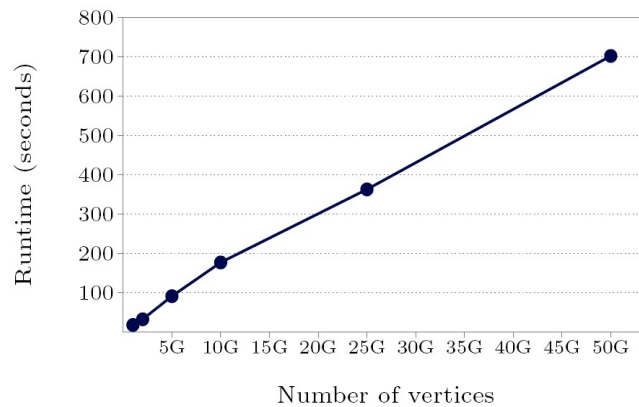




Performance of Pregel



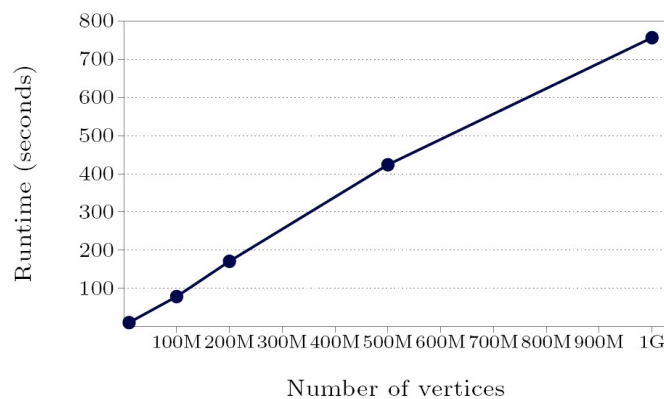
- SSSP – **binary trees**: varying graph sizes on **800** worker tasks



Performance of Pregel



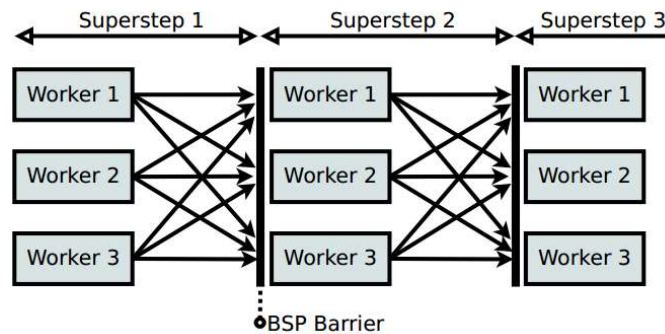
- SSSP – **Random graphs**: varying graph sizes on **800** worker tasks



Remind: BSP



- Balanced computation and communication is fundamental to Pregel's efficiency



Problems with Pregel



- Agnostic to the properties of the input graph
 - **High-degree vertices** receive exponentially more msgs than others
 - => **workload imbalance**
 - => **communication overhead**
- Agnostic to the underlying computation infrastructure
 - Physical links may be overloaded by redundant messages due to the network topology

The Problem of Pregel

....

worker

worker

worker

BSP Barrier

BSP Barrier

....

Computation

Communication

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 61

Optimize Graph Processing

- Existing work focus on **optimizing for graph structure** (static optimization):
 - Optimize graph partitioning:
 - Simple graph partitioning schemes (e.g., hash or range)
 - User-defined partitioning function
 - Sophisticated partitioning techniques (e.g., min-cuts)
- What about **algorithm behavior**?
- Pregel provides coarse-grained load balancing, is it enough?

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 62

Types of Algorithms: Stationary & Non-Stationary



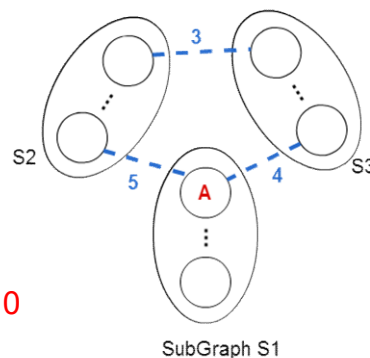
- Algorithms behaves differently, we classify algorithms in two categories depending on their behavior

Algorithm Type	In/Out Vertices Examples Messages	Variable state	Examples
Stationary	Predictable	Fixed	PageRank, Weakly connected component
Non-stationary	Variable	Variable	Distributed minimum spanning tree (DMST)

Non-stationary Algorithm : Topology Changed



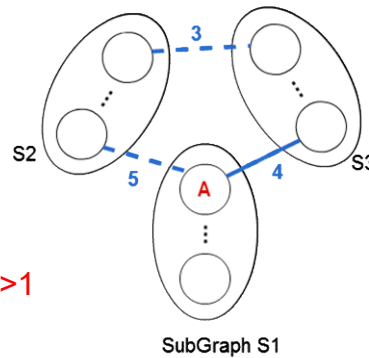
- SuperStep k:



Communication # of vertex A:0

Non-stationary Algorithm :

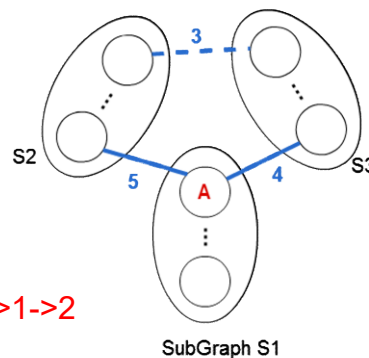
- SuperStep k:
- SuperStep k+1:
 - S1 connects to S3 by A



Communication # of vertex A:0->1

Non-stationary Algorithm :

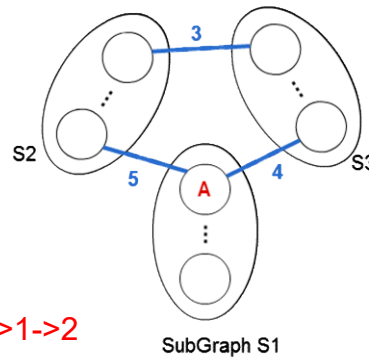
- SuperStep k:
- SuperStep k+1:
- SuperStep k+2:
 - S1 connects S2 by A



Communication # of vertex A:0->1->2

Non-stationary Algorithm :

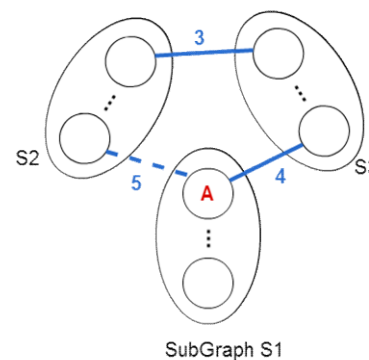
- SuperStep k:
- SuperStep k+1:
- SuperStep k+2:
- SuperStep k+m:
 - S2 connects to S3



Communication # of vertex A: 0->1->2

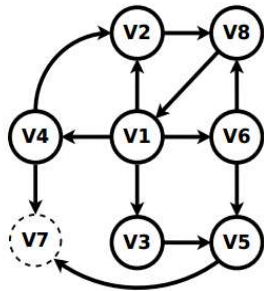
Non-stationary Algorithm :

- SuperStep k:
- SuperStep k+1:
- SuperStep k+2:
- SuperStep k+m:
- SuperStep k+m+n:
 - the edge from A to S2 is not necessary

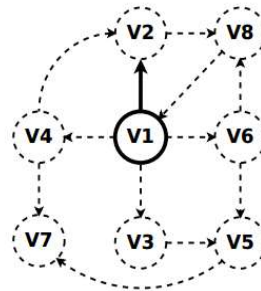


Communication # of vertex A: 0->1->2->1

Types of Algorithms - First Superstep

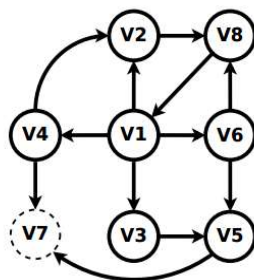


PageRank

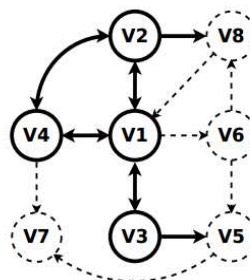


DMST
(Distributed Minimal Spanning Tree)

Types of Algorithms - Superstep k




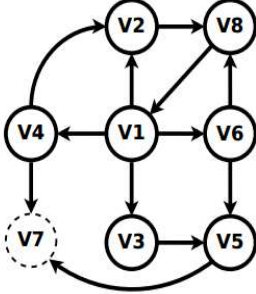
PageRank



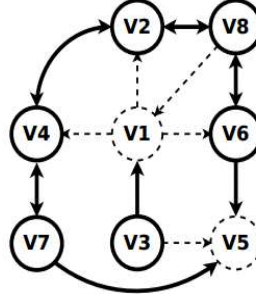
DMST

Types of Algorithms - Superstep $k+m$






PageRank



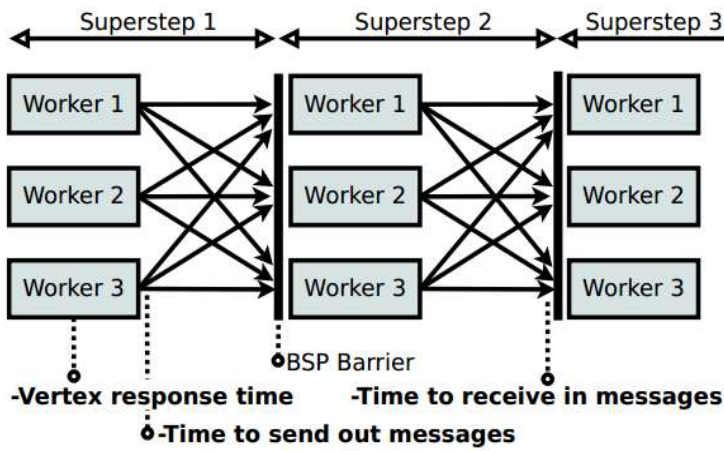
DMST

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 71

What Causes Computation Imbalance in Non-stationary Algorithms?



← Superstep 1 Superstep 2 Superstep 3 →



Worker 1, Worker 2, Worker 3

• BSP Barrier

• Vertex response time • Time to receive in messages

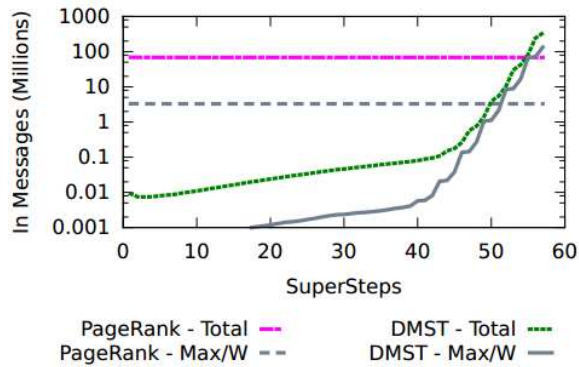
• Time to send out messages

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 72

Why to Optimize for Algorithm Behavior?



- Difference between stationary and non-stationary algorithms



Mizan

What is Mizan?

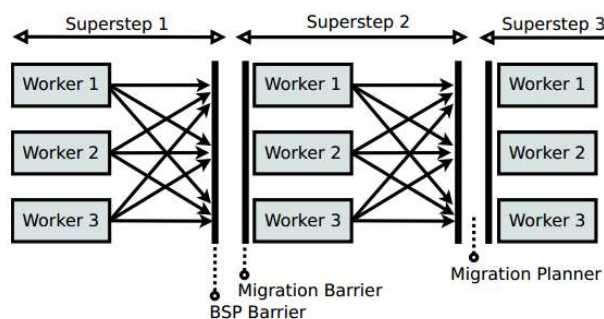


- BSP-based graph processing framework
- Uses **runtime fine-grained vertex migrations** to balance computation and communication
- Follows the Pregel programming model
 - So focus on efficient dynamic load balance
- Open source, written in C++

Mizan's Migration Barrier



- Mizan performs both **planning** and **migrations** after all workers reach the BSP barrier



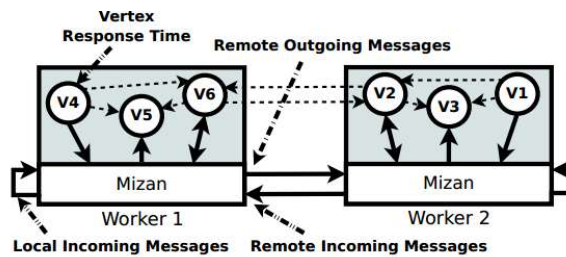
Mizan's Migration Planning Steps



1. Identify the **source of imbalance**: By comparing the worker's execution time against a normal distribution and flagging outliers

■ Mizan monitors for each vertex:

- Remote outgoing messages
- All incoming messages
- Response time
- High level summaries are broadcast to each worker



Mizan's Migration Planning Steps



1. Identify the source of imbalance

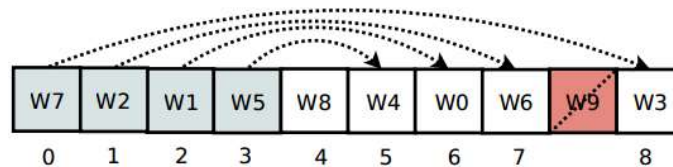
2. **Select the migration objective:**

- Mizan finds the **strongest cause** of workload imbalance by comparing statistics of all workers for
 - outgoing messages
 - incoming messages
 - execution time
- The migration objective is either:
 - Optimize for outgoing messages, or
 - Optimize for incoming messages, or
 - Optimize for response time

Mizan's Migration Planning Steps



1. Identify the source of imbalance
2. Select the migration objective
3. **Pair over-utilized workers with under-utilized ones:**
 - All workers create and execute the migration plan in parallel without centralized coordination
 - Each worker is paired with one other worker at most



CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 79

Mizan's Migration Planning Steps



1. Identify the source of imbalance
2. Select the migration objective
3. Pair over-utilized workers with under-utilized ones
4. **Select vertices to migrate:**
 - Depending on the migration objective (same as step 2)

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

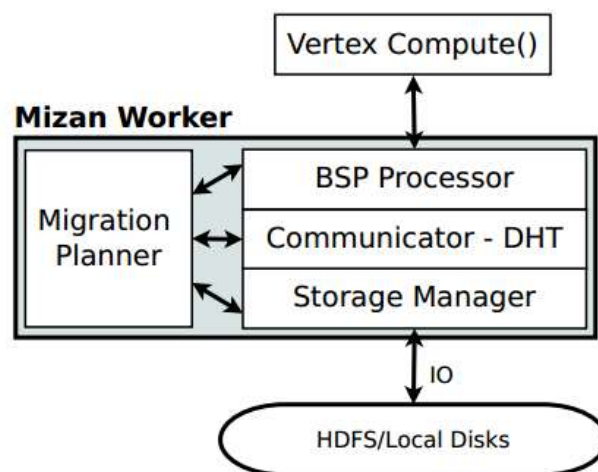
Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 80

Mizan's Migration Planning Steps



1. Identify the source of imbalance
2. Select the migration objective
3. Pair over-utilized workers with under-utilized ones
4. Select vertices to migrate
5. **Migrate vertices:**
 - How to migrate vertices freely across workers while maintaining vertex ownership and fast updates?
 - How to minimize migration costs for large vertices?

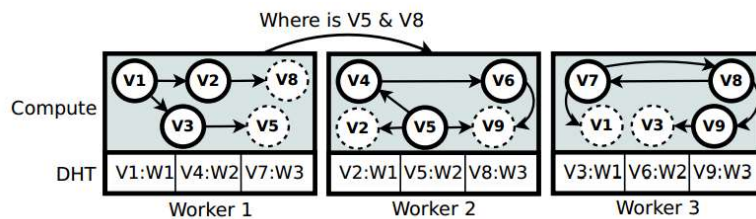
Mizan Architecture



Vertex Ownership



- Mizan uses **distributed hash table (DHT)** to implement a distributed lookup service:
 - V can execute at any worker
 - V's home worker ID = $(\text{hash}(\text{ID}) \bmod N)$
 - Workers ask the home worker of V for its current location
 - The home worker is notified with the new location as V migrates



Migrating Vertices with Large Message Size

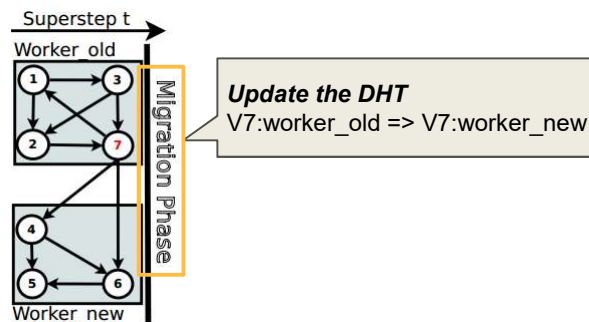


- When the graph is very large, some vertices may have tremendous number of edges.
 - The message queue may be too large to migrate

Migrating Vertices with Large Message Size



- Introduce **delayed migration** for the vertices with very large message queues
- After SS t (first migration):
Only **ownership** of vertex 7 is moved to Worker_new



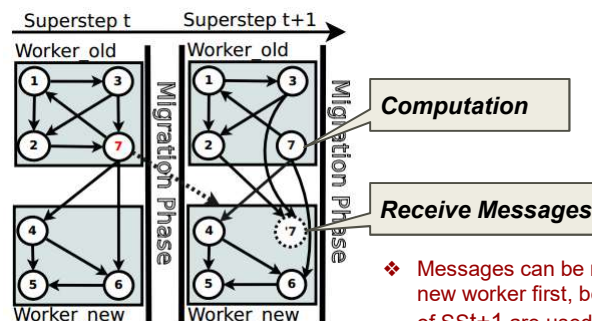
CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 85

Migrating Vertices with Large Message Size



- At SS $t + 1$:
 - Worker_new receives messages for vertex 7
 - Worker_old do the computation of vertex 7
 - compute new value, mutate the graph...



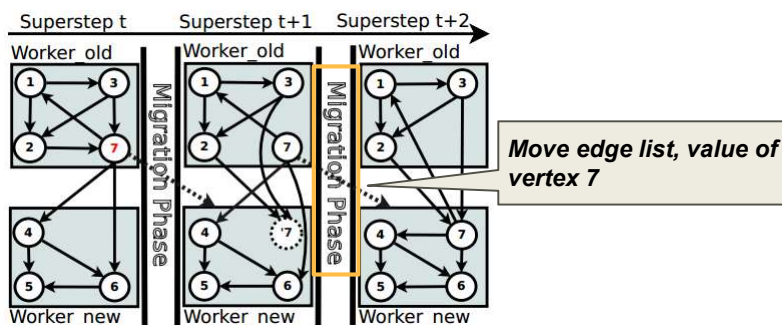
CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 86

Migrating Vertices with Large Message Size



- After SS $t + 1$ (second migration):
worker_old moves state & new value of vertex 7 to Worker_new



CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 87

Mizan Summary



- Mizan is a Pregel system that uses fine-grained vertex migration to load balance computation and communication across supersteps
- Mizan is an open source project developed within InfoCloud group in KAUST in collaboration with IBM, programmed in C++ with MPI
- Mizan improves the overall computation cost between 40% up to two order of magnitudes with less than 10% migration overhead

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

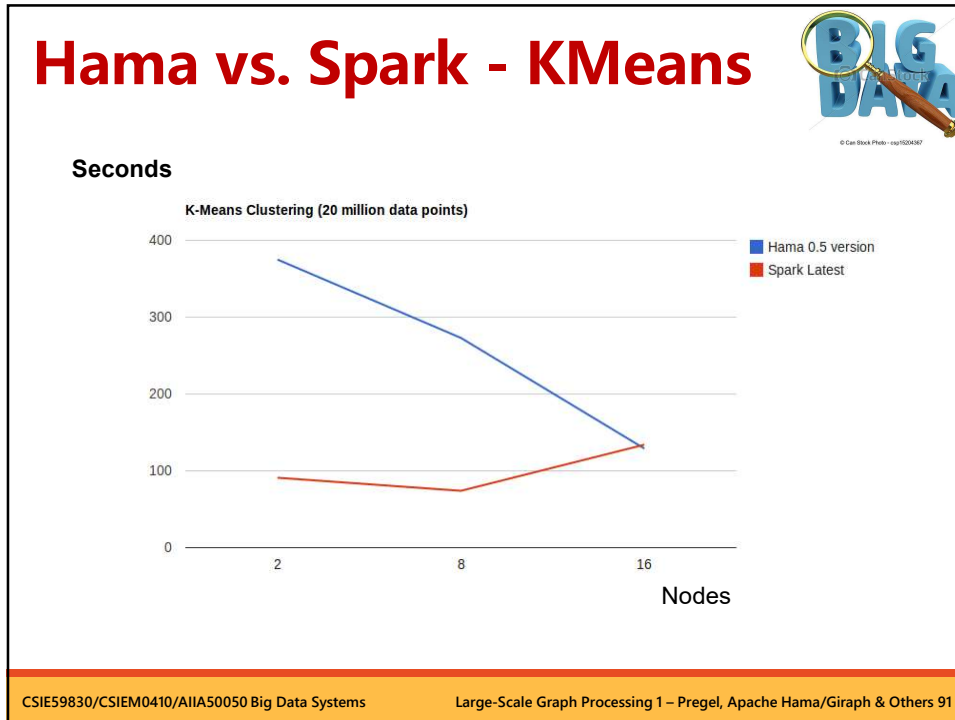
Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 88

Apache Hama

Apache Hama



- A distributed computing framework
- An Top-Level Apache Project(since 2012) for graph processing inspired by Google Pregel and DistBelief
- Based on Bulk Synchronous Parallel(BSP) model
- A general **BSP** computing engine on Top of Hadoop
- Also support vertex and neuron centric programming models
- Written in Java, available for Hadoop greater than 1.0.x and require Java 1.6.x or higher
- **Hama has retired**. The other BSP based project **Apache Giraph** remains active. (June 11, 2020: Giraph 1.3.0)



- ## So, why Hama?
- Simple and Flexible message-passing programming Interface
- And,
- Machine Learning Package
 - K-Means clustering is almost 500x ~ 1000x faster than Mahout MR version
 - Graph Package (Google's Pregel)
 - PageRank is almost 10 ~ 20x faster than MapReduce version
- CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 92

Benchmarks with 256 cores



- SSSP on random 1 Billion edges

400 secs!

- PageRank on Wikipedia link DataSet, contains 5,716,808 pages and 130,160,392 links).

17 secs!

Use Case: Netflow Analytics at Korea Telecom



Weather forecasting for Clouds

- 4 Full Racks
- Used as a Real-time event processing
 - Monitoring the network usage of each VMs as a real time
 - Detecting anomaly traffics
 - Sharing the risks among Servers
 - Billing, ..., etc.

Use Case: SiteRank at Sogou.com



Sogou.com runs SiteRank algorithm on a 7,200 cores Hama cluster.

- SiteRank is the ranking generated by applying the classical PageRank algorithm to the graph of Web sites.
- Dataset is about 400GB contains about 600M vertices and 6B edges



Hama Architecture

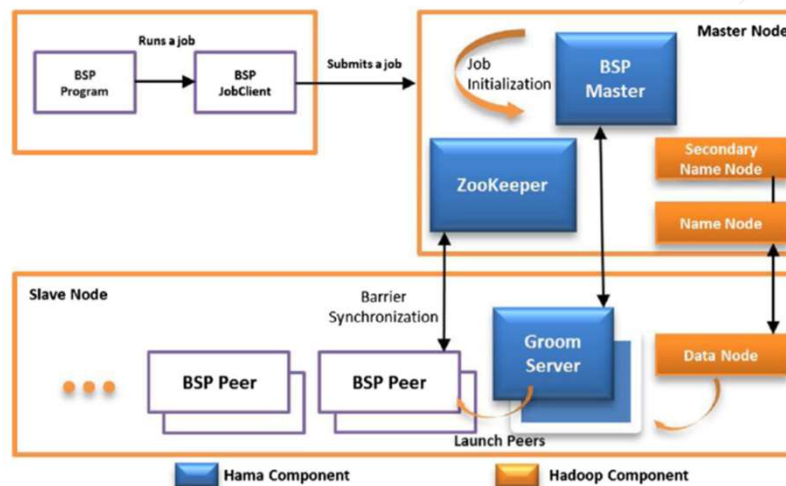


Fig. 2 Apache Hama architecture

BSP Master



- Schedules jobs and assigns the tasks to a Groom Server(next slide)
- Maintains the Groom Server status and job progress information
- Controls faults and supersteps in a cluster
- Distributes execution classes to its slaves
- Provides a cluster control interface for users

Groom Server



- A **slave** component for the BSP computation
- Running **BSP peer tasks** assigned by BSP Master
- Launches one or more BSP peer tasks
- Each task acts as a **worker task** to perform the actual computation
- Send **heartbeat** to BSP Master to report processing status and piggybacks other metrics
- Can run with any distributed storage in addition to HDFS
- A Groom and a data node should run on the same physical node for best performance

Zookeeper



- The synchronization component
- Provides efficient barrier synchronization of the BSP peer tasks
- Zookeeper and BSP Master execute on the same master node

How does it work?



- **BSP Program** runs a **job**
- **BSP JobClient** establishes the **communication channel** with BSP Master using Hadoop RPC framework
- **Partitions the input** and stores the chunks to HDFS before submitting a new job to BSP Master
- Executes locally and periodically sends **status updates** including **memory usage statistics** for each process and **superstep count**.

How does it work?



- BSP Master **updates status** of the Groom Servers on receiving a **heartbeat** messages
- BSP Master efficiently **assigns tasks to idle Grooms** and returns a heartbeat response containing the set of actions to perform
- Once a task is assigned, a Groom Server **continues its execution until the last superstep** is executed
- Upon **task failure**, it is marked failed and gets killed
- **ZooKeeper** manages efficient **barrier synchronization** of the processes

Hama vs Others



- Hama is more focused towards processing complex computation intensive tasks rather data intensive tasks
- It aims to provide a more general purpose framework than Pregel and Apache Hadoop
- It is not restricted to graph processing.
- Supports massive scientific computations such as matrix, graph, machine learning, business intelligence, and network algorithms.

Hama vs Hadoop

Fig. 3 Comparison between Hama and Hadoop architectures

- BSP tasks **can communicate with each other**, leads to better efficiency w/o I/O overheads.

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 103

Key Attribute Comparison

- Hama’s BSP programming model supports versatile processing with diverse application domains.

Table 4 Key attribute comparison of Hama with some well adopted frameworks

Framework	Processing Method	Application Domains			
		Graph	Streaming	Machine Learning	Incremental Learning
Hama	DMV	✓	✓	✓	✓
Hadoop	D	✓	✓	✓	✗
Spark	D	✓	✓	✓	✗
Giraph	V	✓	✗	✗	✗

Processing technique: *D* Directed Acyclic Graph; *M* Matrix; *V* Vertex-centric

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 104

BSP Primitive Operations



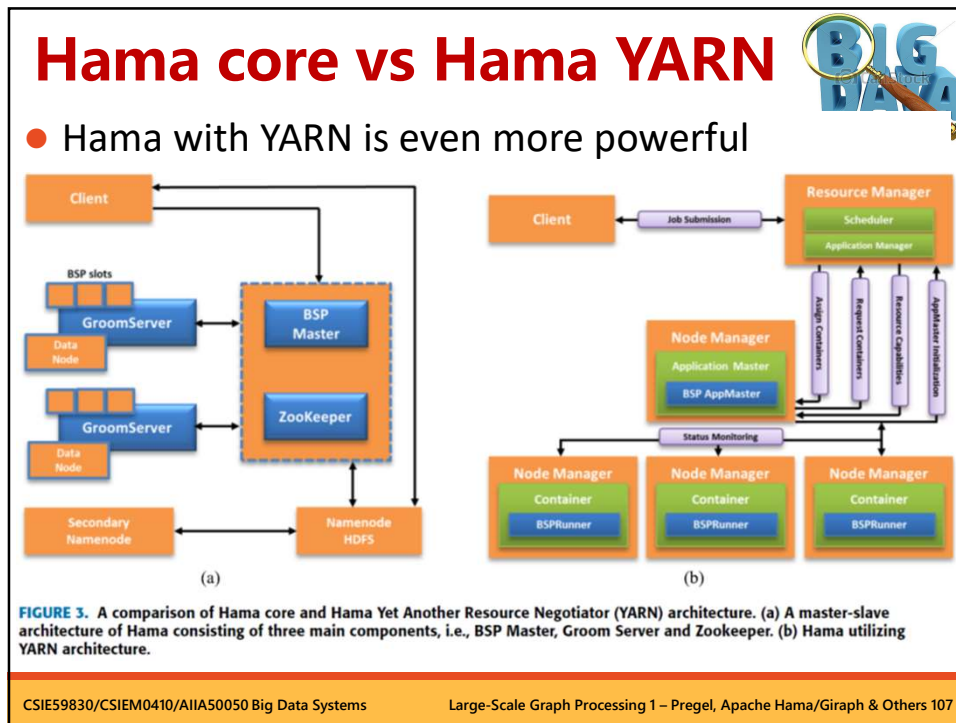
Table 5 Some powerful BSP primitive operations implemented in Hama

Operation	Description
getPeerName()	Returns the name of the peer
getSuperstepCount()	Returns the count of supersteps
send()	Sends a message to another peer
getCurrentMessage()	Returns a message received from the peer
getNumCurrentMessages()	Returns the number of received messages from the peer
clear()	Clears the entries of all queues
write()	Writes a key/value pair to the output interface
sync()	Performs barrier synchronization operations

Some Problems with Hama



- BSP Master is a **single point of failure** and the application will stop if it dies.
- Despite great advances in graph processing performance, the manipulation functions remain somewhat limited.
- The graph partitioning algorithm needs to be customized.



Hama core vs Hama YARN

No.	Hama Core Architecture			Hama on YARN			
	BSP Master	Groom Server	Zookeeper	Resource Manager	BSPApp Master	Node Manager	BSPRunner
	<i>Key Components</i>						
	<i>Working</i>						
1.	The BSP JobClient component partitions the input and stores the input sections to HDFS			BSP JobClient submits an application request including the necessary specifications to the Resource Manager (RM)			
2.	This component is executed locally and keeps track of status updates for the submitted job and superstep count			The RM contacts a Node Manager (NM) for the creation and initialization of BSPApp Master (BAP) instance			
3.	BSP JobClient then establishes a communication channel with BSP Master using Hadoop RPC framework, and submits a job			The BAP determines the required number of resources for entire application execution			
4.	BSP Master determines the required number of resources for job execution			The BAP then requests the necessary resources from the RM			
5.	It then schedules the job to a Groom Server considering the availability of resources			If the requested resources are available, the RM allocates containers to the BAP or otherwise queues up the requests			
6.	The Groom Server periodically sends a heartbeat and task status updates to BSP Master			The BAP sends a container launch context to NM containing the information required to run an application such as environment variables and command strings			
7.	A Groom Server should run on the same machine as a data node for optimal performance based on data locality			The NM then creates the requested container and BSPRunner executes the tasks			
8.	BSP JobClient updates itself with job progress and superstep count			The BAP also performs status monitoring and in case of a container or a node failure, the task is restarted on the next available slot			
9.	Zookeeper remains responsible for barrier synchronization of BSP Peer tasks throughout the whole execution cycle			Once all tasks are complete, the BAP deregisters itself from the RM and shuts down			

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 108

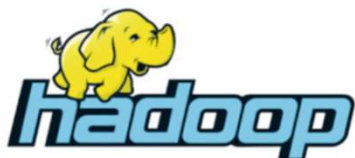
Apache Giraph

Giraph Overview



- Starting in 2012, **Apache Giraph** has been established to clone the ideas of **Pregel** and implement it **on top** of the **Hadoop** framework.

Google MapReduce



Google Pregel



Computing Model of Giraph



- Almost the same as Pregel
- Processing is expressed as a **sequence** of **supersteps**.
- In a superstep, the framework starts a **user-defined function** for each **vertex**, conceptually in **parallel**.
- The function specifies the **behavior** at a **single vertex** V and a **single superstep** S .
- Can **read messages** sent to V in superstep $S - 1$, **send messages** to other vertices (received at superstep $S + 1$), and **modify the state** of V and its **outgoing edges**.
- **Messages** are typically sent along **outgoing edges**, but can send a message to any vertex with a **known ID**.

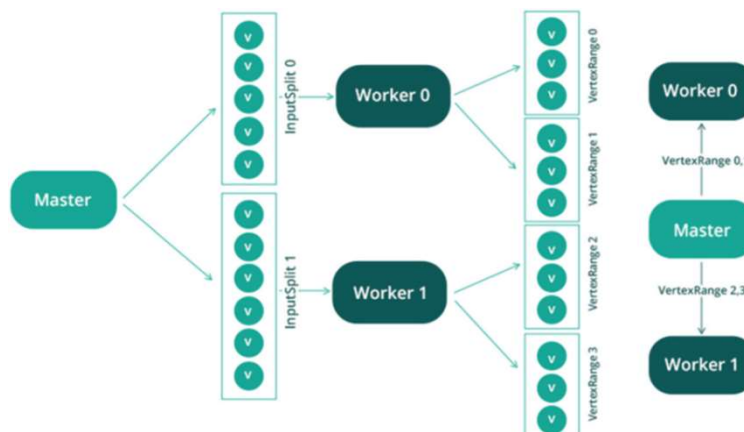
CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 111

Execution Architecture



- Giraph applies a **master/worker** architecture.



CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 112

Execution Architecture



- The **master** node **assigns partitions** to workers, **coordinates** synchronization, requests **checkpoints**, and collects **health statuses**. (Giraph uses Apache ZooKeeper for synchronization.)
- **Workers** are responsible for **vertices**.
- A worker starts the **compute()** function for the active vertices. It also **sends, receives, and assigns messages** with other vertices.

The MaxVertexValue.java Example



```
package example;
import java.io.IOException;
import org.apache.giraph.graph.Vertex;
import org.apache.giraph.graph.BasicComputation;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.LongWritable;

public class MaxVertexValue extends
BasicComputation<LongWritable, DoubleWritable, FloatWritable,
DoubleWritable> {
    @Override
    public void compute(Vertex<LongWritable, DoubleWritable,
FloatWritable> vertex, Iterable<DoubleWritable> messages)
throws IOException {
```

MaxVertexValue.java



```
boolean changed = false;
for (DoubleWritable msg : messages) {
    /* Collect in-messages and update if necessary */
    if ( vertex.getValue().get() < msg.get() ) {
        vertex.setValue( new DoubleWritable( msg.get() ) );
        changed = true;
    }
}
/* Send out-message at Superstep 0 or Vertex value is
changed */
if ( getSuperstep() == 0 || changed ) {
    sendMessageToAllEdges(vertex, getValue());
}
vertex.voteToHalt();
}
```

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 115

Giraph Programming



- Giraph programs are traditionally written mostly in Java. (as previous example)
- To support writing Computation code in **Python**, the Giraph team add **Jython** bindings so that the Python computation code can communicate back with the Java Giraph classes.
- See issue GIRAPH-683 for more details.

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 116

Assignment 4a: Graph Computing with Hama/Giraph



1. Complete the Maximum Value example.
2. Complete the PageRank example. Test your program on at least a connected and a disconnected graph.

Note:

1. Can choose either Hama or Giraph (or both).
2. You may use existing code on the Web for both exercises.

Assignment 4b : Graph Computing with Hama/Giraph



3. Each year, each lab in the CSIE department can propose the **lower** and **upper bounds** on the number of new grad students to recruit at that year. Each new grad student can classify all CSIE labs into **three categories**: highly desirable(3 points), desirable(2 points), acceptable(1 points). Write a graph program to generate a **feasible assignment** of all new grad students.
4. (Optional) Generate an **optimal assignment** with **highest possible score**.

Alternative Graph Systems



- The BSP model and Pregel family of systems are **not the only type** of big graph systems.
- We will only sketch through some of the representative systems.
 - **GraphLab family** of systems: GraphLab, PowerGraph
 - **Spark-based** systems: GraphX
 - **Hadoop-based** systems: Gradoop
- Another type of systems are **graph databases**. (next lecture)

GraphLab



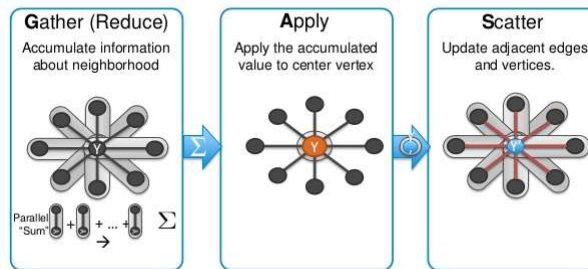
- **GraphLab** is an open-source large graph processing system in **C++** from **CMU**.
- Based on **shared memory** abstraction and the **GAS** (**Gather, Apply, Scatter**) model.
- The GraphLab abstraction has three parts:
 - The **data graph**: represents program **state** and **dependencies**.
 - The **update function**: **user computation** on data graph by **transforming** data in **scopes**.
 - The **sync operation**: **synchronous** and **asynchronous** execution modes.

GAS Model



- **Gather** phase: a vertex **collects** information about its neighborhood.
- **Apply** phase: **performs** user computations
- **Scatter** phase: **updates** its adjacent vertices and edges.

GAS Decomposition



Academy
Invent

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 121

GAS in GraphLab



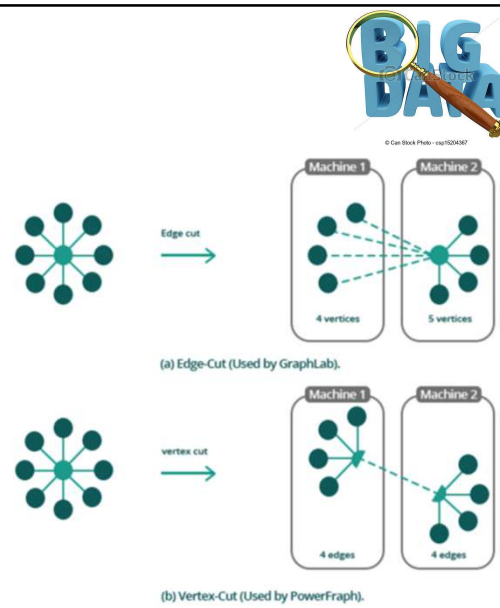
- Vertices can directly **pull** their neighbors' data. (In BSP, a vertex can learn its neighbors' values only via the messages that its neighbors **push** to it.)
- **Synchronous** mode uses **barriers**.
- **Asynchronous** mode uses **distributed locking** to **avoid conflicts** and to maintain **serializability**.
- Many **ML algorithms** fit into **GAS model**: graph analytics, inference in graphical models, matrix factorization, collaborative filtering, clustering, LDA, ...

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 122

PowerGraph

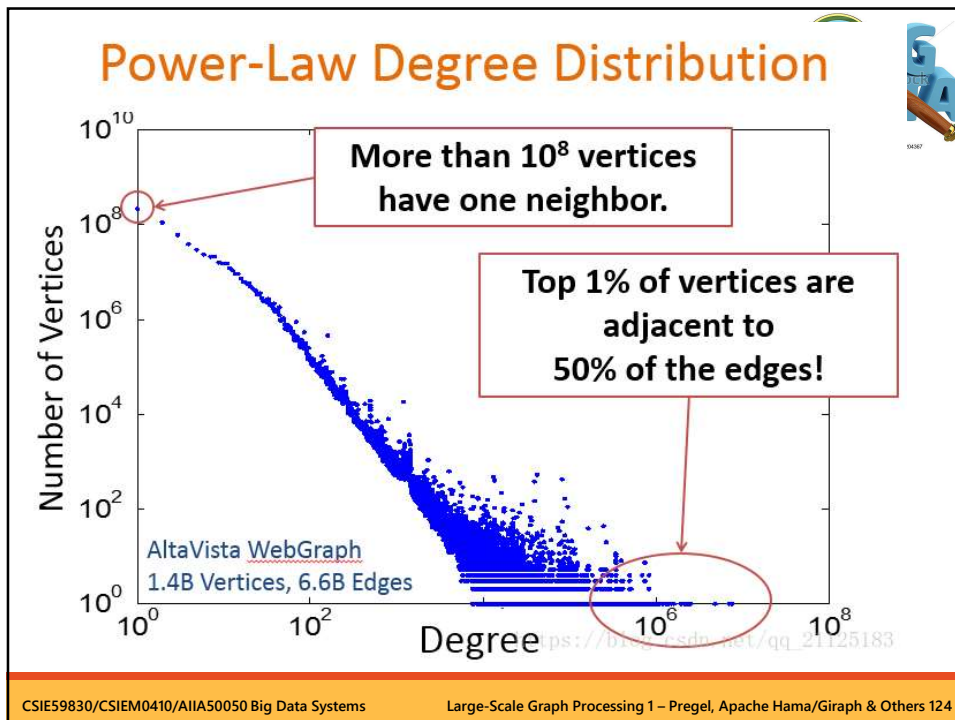
- High-degree vertices results in imbalanced workload during the graph computation.
- PowerGraph tackles the problem with a vertex-cut partitioning scheme so that the edges of a high degree vertex are handled by multiple workers.



© Cal Stock Photo - iap120437

(a) Edge-Cut (Used by GraphLab). (b) Vertex-Cut (Used by PowerGraph).

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 123



PowerGraph



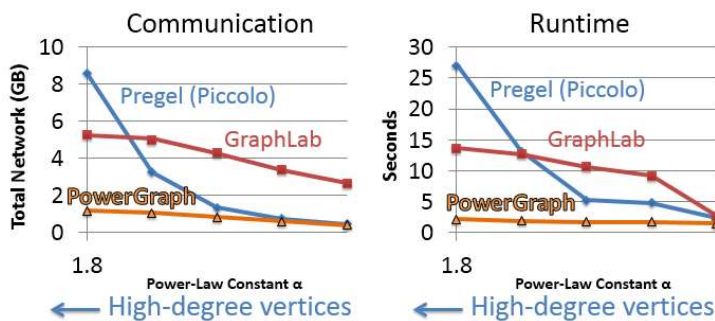
- Vertices are replicated across workers, and communication is required to guarantee that the vertex value on each replica remains consistent.
- Directly exploit the GAS decomposition to factor vertex programs over edges.
- Retain the “think-like-a-vertex” programming style while distributing the computation of a single vertex program over the entire cluster.
- PowerGraph attempts to merge the best features from both Pregel and GraphLab.

Comparison of PowerGraph, GraphLab & Pregel



Comparison with GraphLab & Pregel

- PageRank on Synthetic Power-Law Graphs:



PowerGraph is robust to high-degree vertices.

Spark-based Systems



- **GraphX** is a distributed graph engine on top of Spark.
- Extending RDD to **resilient distributed graph(RDG)** that associates **records** with **vertices** and **edges** and provides expressive **primitives**.
- **Basic GraphX RDG interface** expresses graph transformations, filtering operations, and queries.
- Allow the **construction** of **new graph-parallel APIs**.
- Enables the **composition** of **graphs** with **unstructured** and **tabular data**.
- Permits the **same physical data** to be **viewed both** as a **graph** and as **collections**.

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 127

GraphX



- **Vertex-cut partitioning** encodes graphs as horizontally partitioned collections.
- Achieves **low-cost fault tolerance** with **logical partitioning** and **lineage**.
- Immutability **reuses indices** across **graph** and **collection** views and over **multiple iterations**, **reducing memory overhead** and **improving system performance**.



Table View



GraphX Unified Representation



Graph View

CSIE59830/CSIEM0410/AIIA50050 Big Data Systems

Large-Scale Graph Processing 1 – Pregel, Apache Hama/Giraph & Others 128

Graph on Hadoop



- **Gradoop** is a distributed graph querying and processing framework on Hadoop.
- Support the **extended property graph model (EPGM)**.

