



Large-Scale Graph Processing 1: Pregel & Apache Hama

Shiow-yang Wu (吳秀陽)

CSIE, NDHU, Taiwan, ROC

Lecture material is mostly home-grown, partly taken with permission and courtesy from Professor Shih-Wei Liao of NTU.

Outline

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

Graph-based Modeling



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



Importance of Graphs



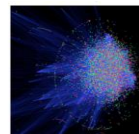
- Graphs abstract application-specific algorithms into generic problems represented as interactions using **vertices** and **edges**



Max flow in road network



Ranking in social networks



Simulating protein interactions

- Algorithms vary in their computational requirements

Graph Algorithms

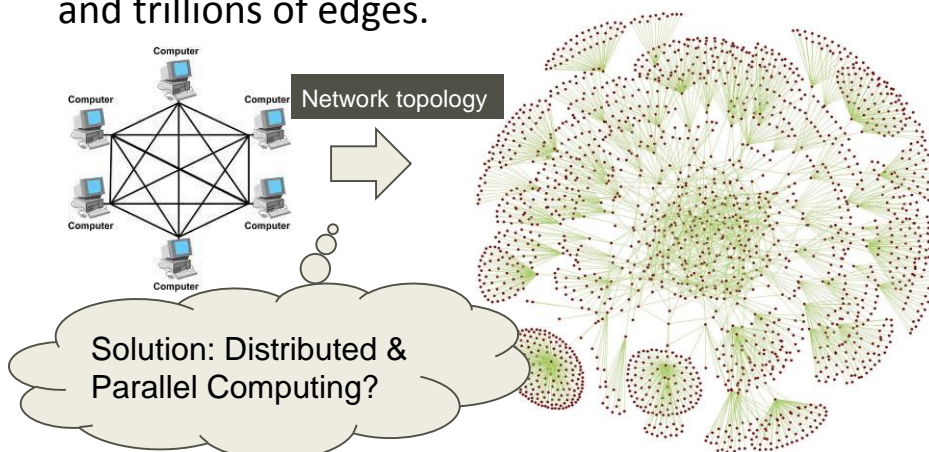


- Algorithms solving problems on graphs
 - Graph Coloring, Route Problem, Network Flow...etc.
- Many modern applications can be reduced to graph problems
 - Network load balancing, PageRank, Shortest Path...
- Many (approximation) algorithms with polynomial time complexity have been developed to solve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

Problems: when the graph is BIG



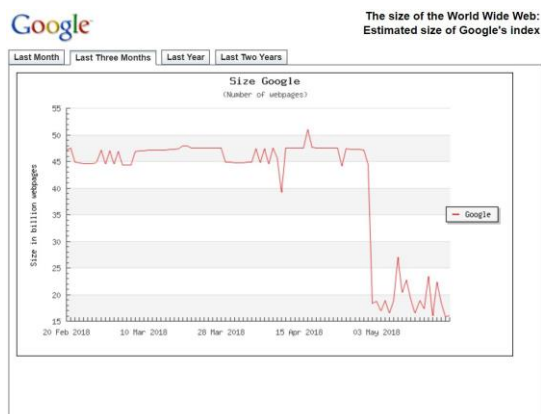
- Graphs become very large, billions of vertices and trillions of edges.



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 Big Data Systems

Large-Scale Graph Processing 1 – Pregel & Apache Hama 7

Examples of Real-World Big Graph



- Over **2.20 billion** monthly active Facebook **users** (Q1 2018). Average FB user has **155 friends**.
- As of 21 May 2018, there are **5,653,299** articles in the **English Wikipedia** and **130+ million links** between them.
- On Jan 2018, the total number of **Pinterest Pins** is over 50+ billion with much more **Related Pins** links between them.
- ...

CSIE59830 Big Data Systems

Large-Scale Graph Processing 1 – Pregel & Apache Hama 8

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 worse**

Existing Solutions and Limitations



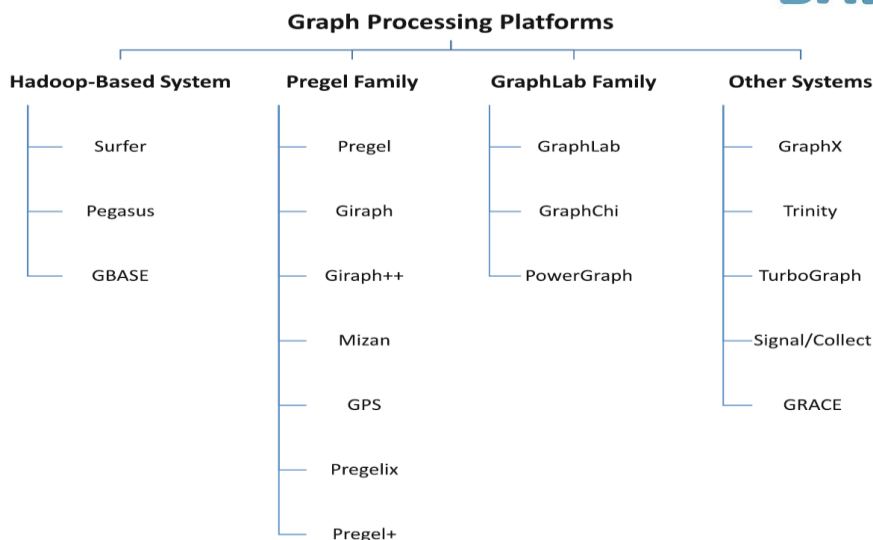
- Custom Distributed Infrastructure
 - Design a model and system for each graph algorithm.
 - **Too expensive**
- Distributed Computing Platform
 - e.g. MapReduce
 - Often **ill-suited for graph algorithms** that often have:
 - ▣ Many iterations
 - ▣ Many intermediate results
 - ▣ Too many unnecessary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

Existing Solutions and Limitations(2)



- Single-computer Graph Algorithm Library
 - e.g. BGL, LEDA, NetworkX, JDSL, GraphBase, FGL...
 - **Not scalable**, cannot afford big data
- Parallel Graph System
 - e.g. Parallel BGL, CGMgraph
 - **No fault tolerance**

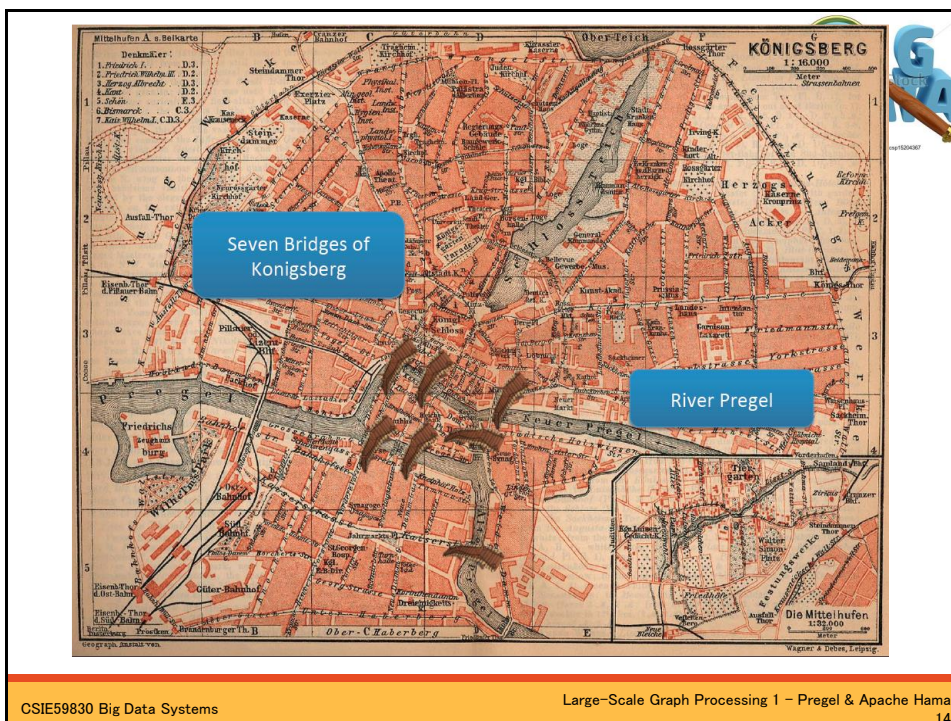
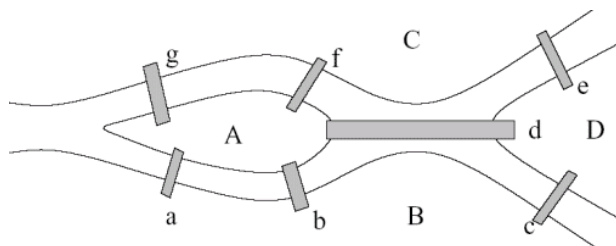
Graph Processing Platforms



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



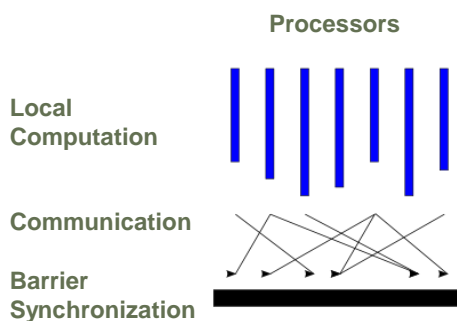
- Pregel was introduced by Google as a scalable abstraction for large-scale graph processing
 - Overcomes the limitations of processing graphs on MapReduce
 - Based on **vertex-centric** computation
 - Utilizes **bulk synchronous parallel (BSP)**

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

Computation Model

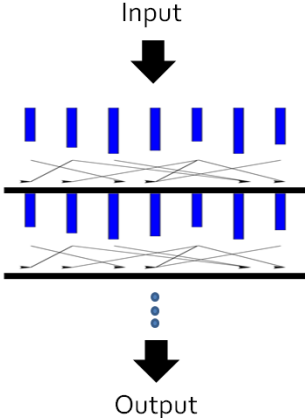


- **Bulk Synchronous Parallel (BSP)** model
 - Bridging model for designing parallel algorithms.
 - **Super steps**
 - Vertices compute in parallel
 - Messages can be sent at the end of each step



Computation Model (2)

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



Input

Supersteps
(a sequence of iterations)

Output

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama
17

Computation Model (3)

- 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

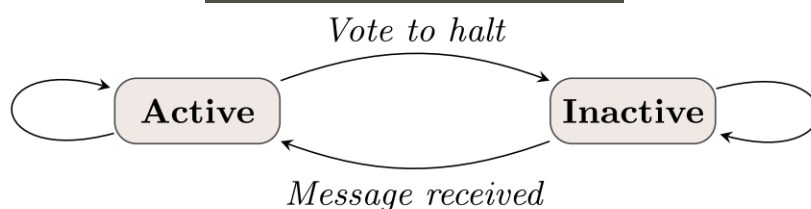
CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama
18

Computation Model (4)



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

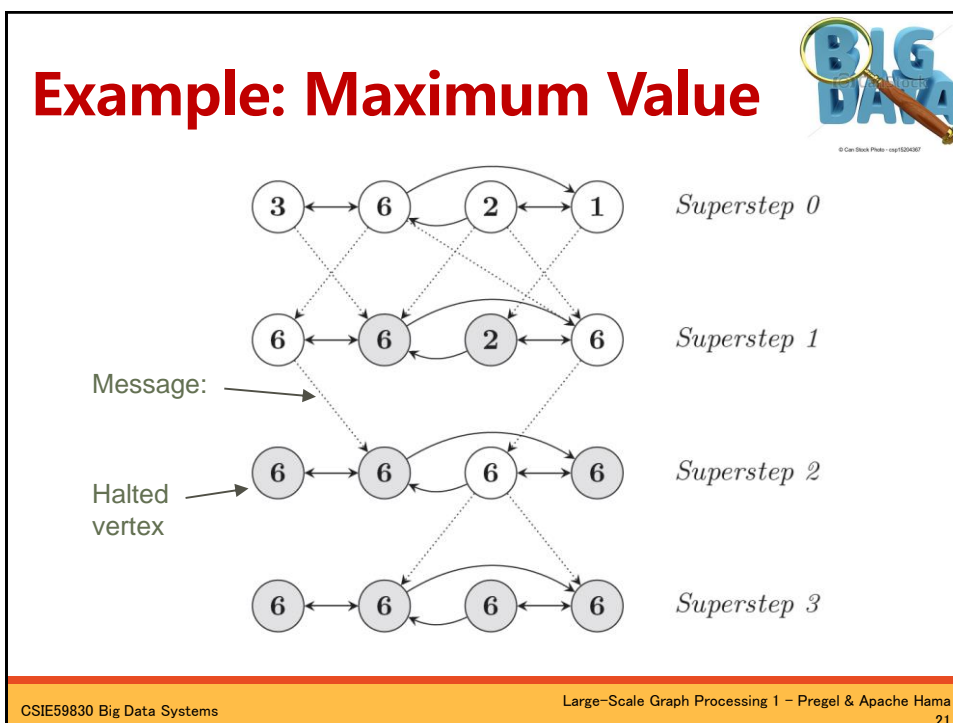
State Machine of Vertices



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



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 code:

- Override this! (pointing to `Compute`)
- in msgs (pointing to `msgs`)
- out msg (pointing to `dest_vertex`)

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 22

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()

OutEdgeIterator GetOutEdgeIterator();

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

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

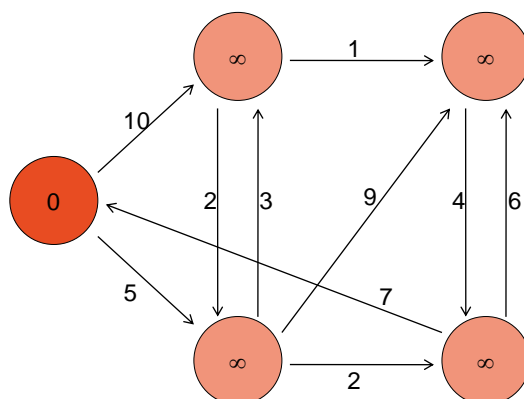
Decide whether to halt

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



Example: SSSP – Parallel BFS in Pregel

The diagram shows a graph with 5 nodes. Node 0 is the source and is active (red). All other nodes (1, 2, 3, 4) have an infinity value and are active (red). Edges and their weights are: (0,1) weight 10, (0,2) weight 5, (1,2) weight 2, (1,3) weight 3, (2,3) weight 3, (2,4) weight 7, (3,4) weight 9, (3,5) weight 2, (4,5) weight 6. Messages (red 'x') are shown on edges (1,2), (2,3), (3,4), (4,5), (2,4), and (3,5). A legend on the right defines the symbols: Inactive Vertex (green circle), Active Vertex (red circle), $\overset{x}{\rightarrow}$ Edge weight, and \times Message.

BIG DATA
© Can Stock Photo - iag1524037

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama
27

Example: SSSP – Parallel BFS in Pregel

The diagram shows the same graph as slide 27, but with updated values. Node 0 is inactive (green) with value 0. Node 1 is active (red) with value 10. Node 2 is active (red) with value 5. Nodes 3 and 4 are inactive (green) with value infinity. Edges and their weights are: (0,1) weight 10, (0,2) weight 5, (1,2) weight 2, (1,3) weight 3, (2,3) weight 3, (2,4) weight 7, (3,4) weight 9, (3,5) weight 2, (4,5) weight 6. A legend on the right defines the symbols: Inactive Vertex (green circle), Active Vertex (red circle), $\overset{x}{\rightarrow}$ Edge weight, and \times Message.

BIG DATA
© Can Stock Photo - iag1524037

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama
28

Example: SSSP – Parallel BFS in Pregel

Legend:

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

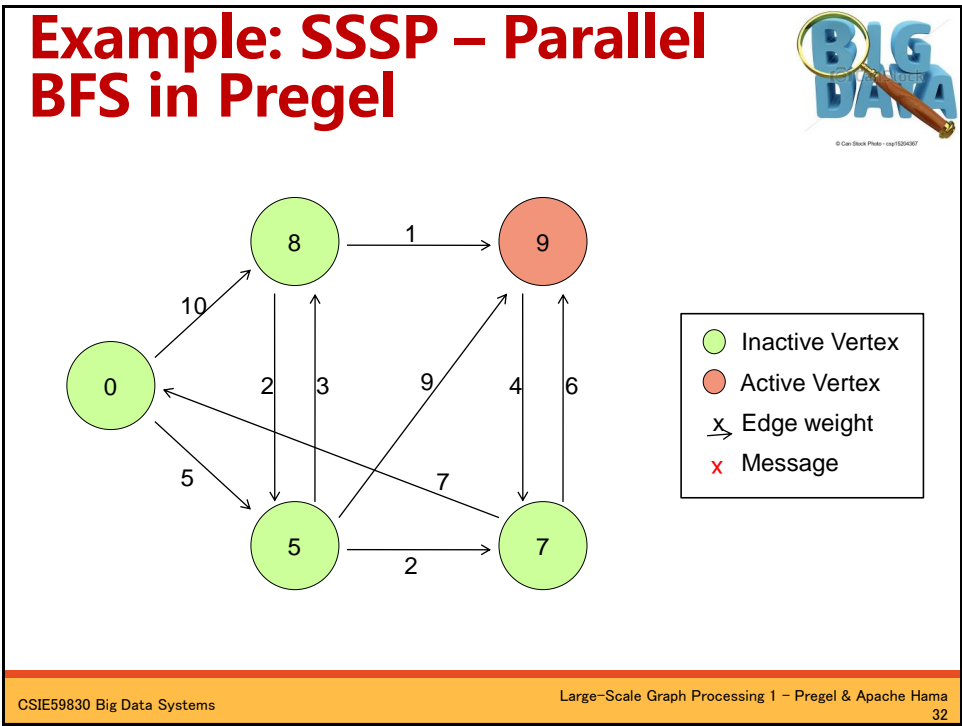
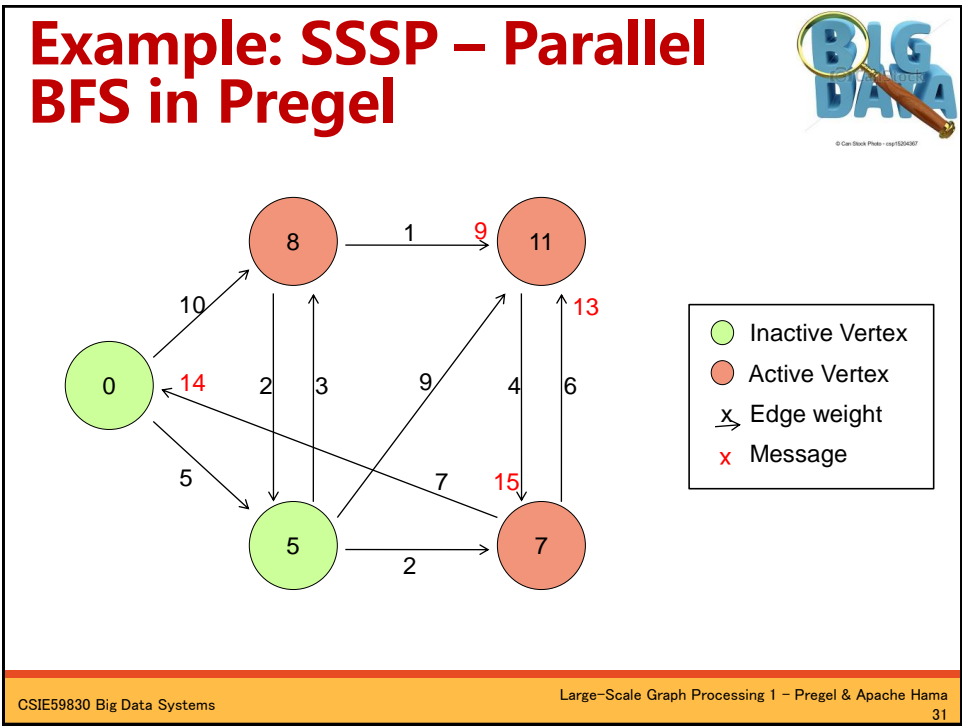
CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 29

Example: SSSP – Parallel BFS in Pregel

Legend:

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

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 30



Example: SSSP – Parallel BFS in Pregel

Legend:

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

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 33

Example: SSSP – Parallel BFS in Pregel

Legend:

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

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 34

Program Example: PageRank

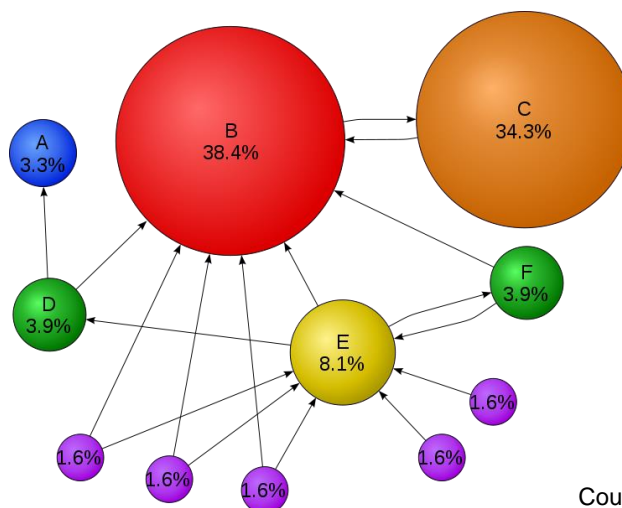


```
class PageRankVertex
  : public Vertex<double, void, double> {
public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
        0.15 / NumVertices() + 0.85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



Courtesy: Wikipedia

System Architecture

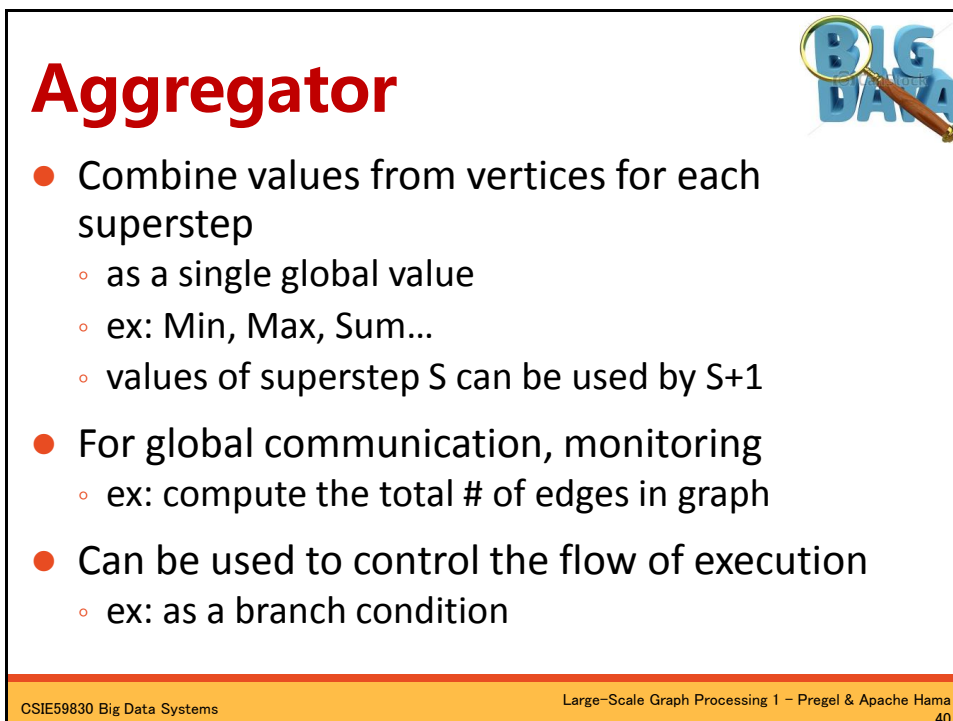
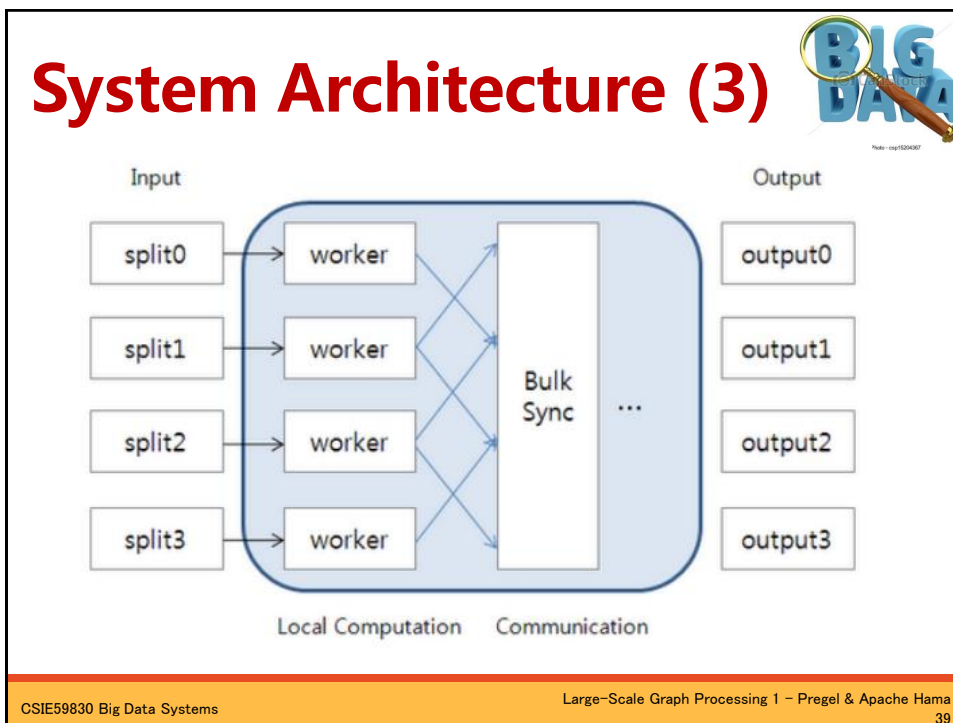


- User program is copied to many machines
- One machine become **Master**
 - Be responsible to coordinate activities
 - Partition the whole graph
 - Assign each partitions to workers
- Others become **Workers**
 - Operate computation
 - Maintain the state of partition(s)

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



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

Fault Tolerance (3)

The diagram illustrates a fault-tolerant execution model. It features three horizontal blue arrows representing workers, labeled "3 workers" on the left. Two vertical red bars represent checkpoints, labeled "checkpoint" at the bottom. Two vertical black bars represent supersteps, labeled "supersteps" at the bottom. The workers' arrows are positioned between the checkpoints and supersteps, indicating that work is performed between checkpoints and supersteps.

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 43

Fault Tolerance (4)

The diagram illustrates a fault-tolerant execution model with logging. It features three horizontal blue arrows representing workers, labeled "3 workers" on the left. Two vertical red bars represent checkpoints, labeled "checkpoint" at the bottom. Two vertical black bars represent supersteps, labeled "supersteps" at the bottom. The workers' arrows are positioned between the checkpoints and supersteps, and each arrow has a small blue arrow pointing to the right labeled "logs...", indicating that logs are generated during the execution of supersteps.

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 44

Fault Tolerance (5)

BIG DATA
© Can Stock Photo - ray1524007

3 workers

logs...

logs...

logs...

supersteps

checkpoint

checkpoint

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama
45

Fault Tolerance (6)

BIG DATA
© Can Stock Photo - ray1524007

3 workers

logs...

logs...

fail!

supersteps

checkpoint

checkpoint

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama
46

Fault Tolerance (7)

3 workers

logs...

logs...

Only this partition needs to recompute

These two partition use the logs to recover the state before fail

supersteps

checkpoint

checkpoint

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 47

Performance of Pregel

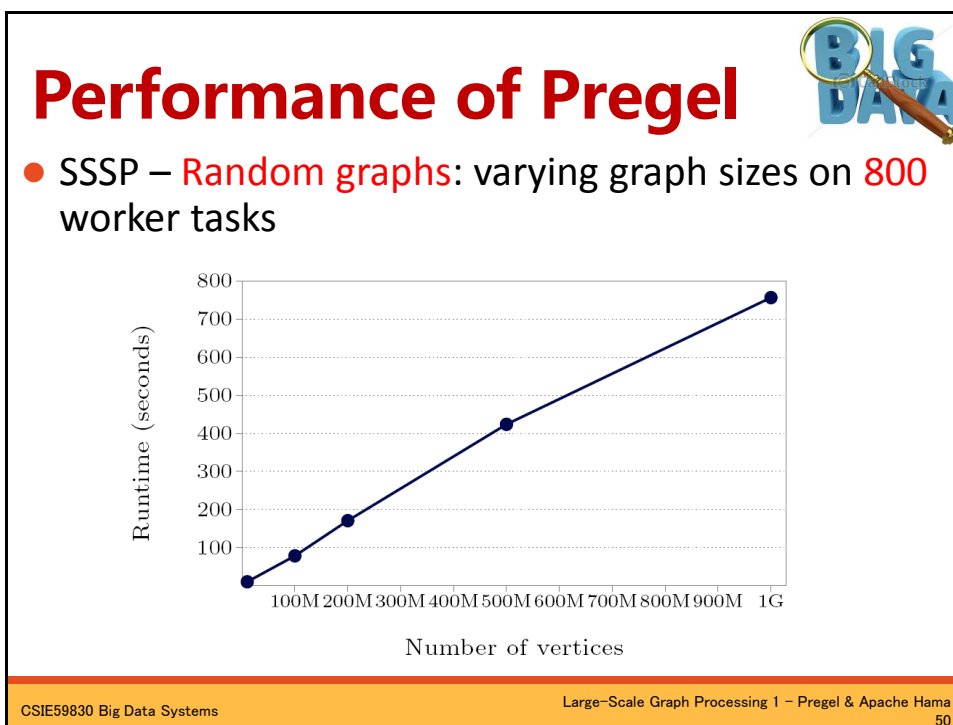
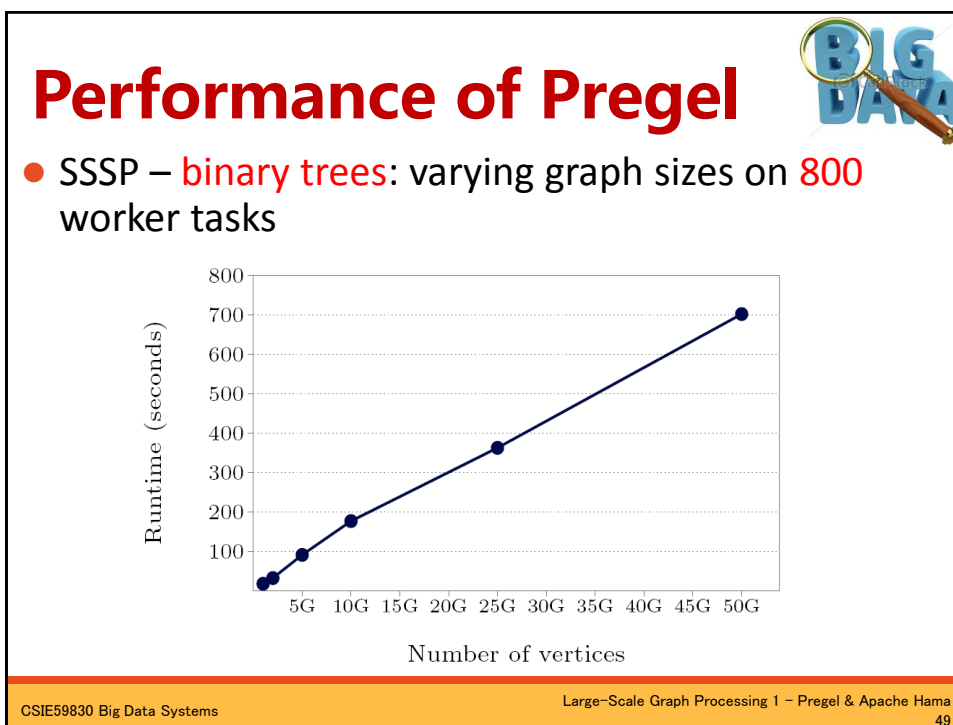
- SSSP – 1 billion vertex binary tree: varying # of worker tasks

Number of worker tasks	Runtime (seconds)
100	175
200	45
300	38
400	28
600	22
800	18

Runtime (seconds)

Number of worker tasks

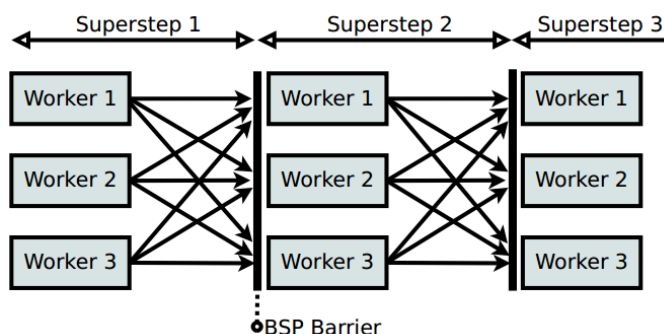
CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 48



Remind: BSP



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

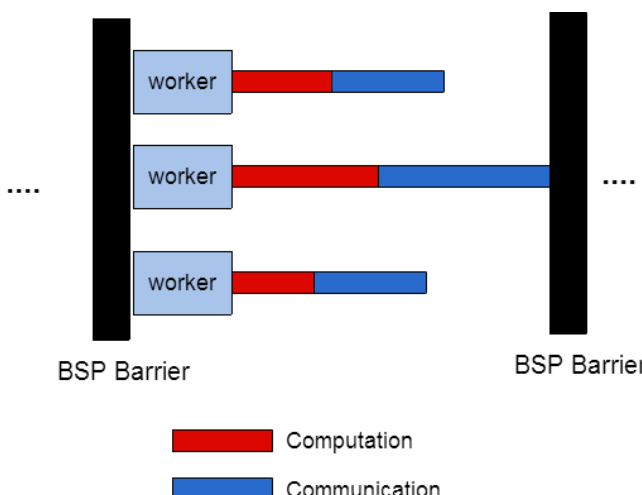


Problems with Pregel



- 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 Big Data Systems

Large-Scale Graph Processing 1 – Pregel & Apache Hama

53

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 Big Data Systems

Large-Scale Graph Processing 1 – Pregel & Apache Hama

54

Types of Algorithms: Stationary & Non-Stationary



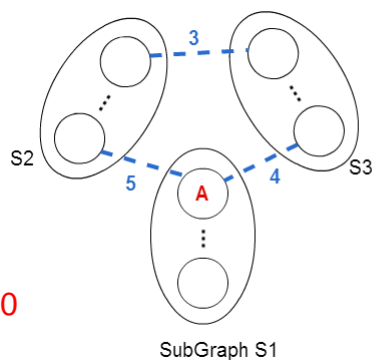
- Algorithms behaves differently, we classify algorithms in two categories depending on their behavior

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



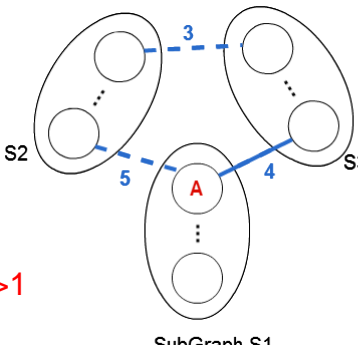
- SuperStep k:



Communication # of vertex A:0

Non-stationary Algorithm :

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

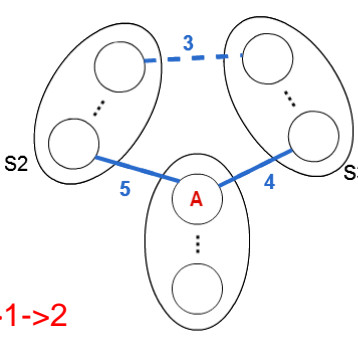


Communication # of vertex A: 0->1

CSIE59830 Big Data Systems
Large-Scale Graph Processing 1 – Pregel & Apache Hama
57

Non-stationary Algorithm :

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

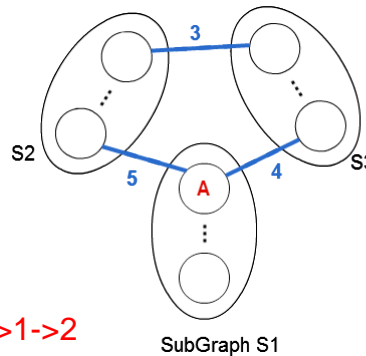


Communication # of vertex A: 0->1->2

CSIE59830 Big Data Systems
Large-Scale Graph Processing 1 – Pregel & Apache Hama
58

Non-stationary Algorithm : Topology Changed

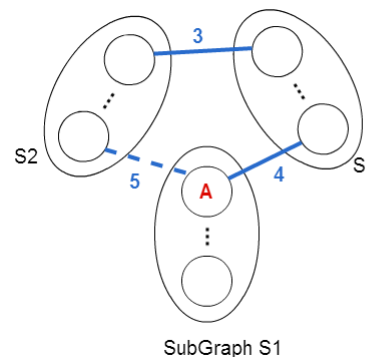
- SuperStep k:
- SuperStep k+1:
- SuperStep k+2:
- SuperStep k+m:
 - S2 connects to S3



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


Non-stationary Algorithm : Topology Changed

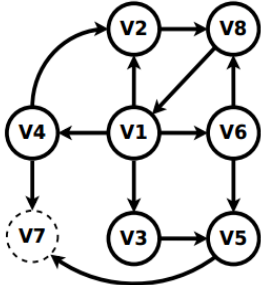
- SuperStep k:
- SuperStep k+1:
- SuperStep k+2:
- SuperStep k+m:
- SuperStep k+m+n:
 - the edge from A to S2 is not necessary



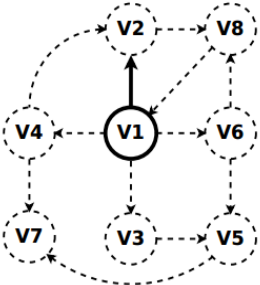
Communication # of vertex A: 0->1->2->1

Types of Algorithms - First Superstep






PageRank

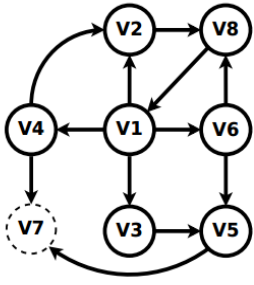


DMST
(Distributed Minimal Spanning Tree)

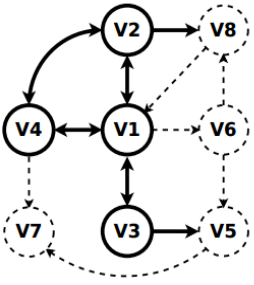
CSIE59830 Big Data Systems
Large-Scale Graph Processing 1 – Pregel & Apache Hama
61

Types of Algorithms - Superstep k






PageRank

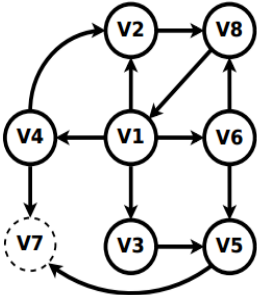


DMST

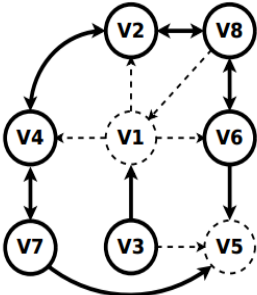
CSIE59830 Big Data Systems
Large-Scale Graph Processing 1 – Pregel & Apache Hama
62

Types of Algorithms - Superstep $k+m$






PageRank



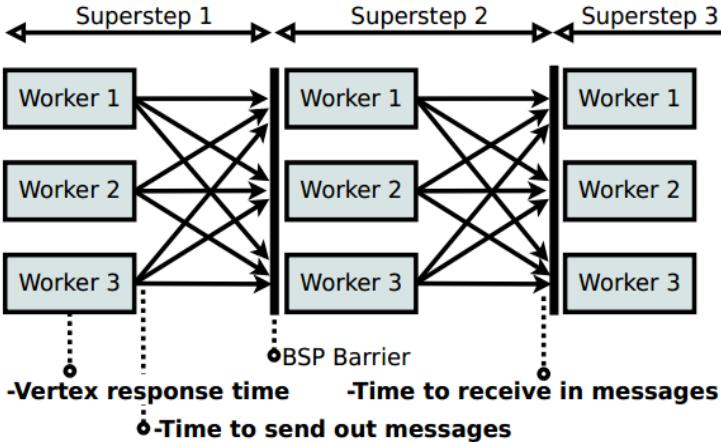
DMST

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 63

What Causes Computation Imbalance in Non-stationary Algorithms?



Superstep 1 Superstep 2 Superstep 3



Worker 1 Worker 1 Worker 1
 Worker 2 Worker 2 Worker 2
 Worker 3 Worker 3 Worker 3

• BSP Barrier

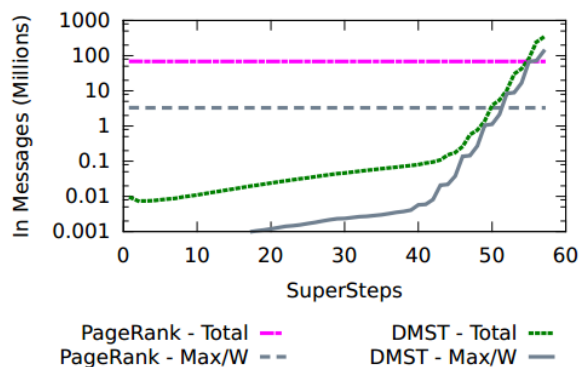
• -Vertex response time • -Time to receive in messages
 • -Time to send out messages

CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 64

Why to Optimize for Algorithm Behavior?



- Difference between stationary and non-stationary algorithms




What is Mizan?

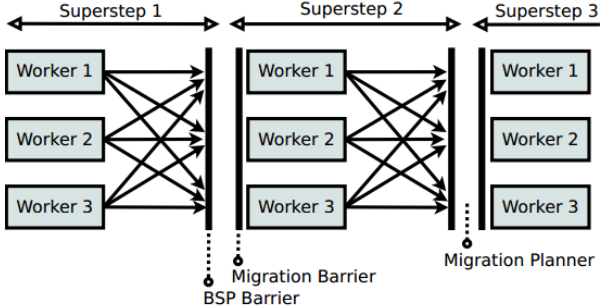


- BSP-based graph processing framework
- Uses runtime fine-grained vertex migrations to balance computation and communication
- Follows the Pregel programming model
 - So focus on efficient dynamic load balance
- Open source, written in C++

Mizan's Migration Barrier




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



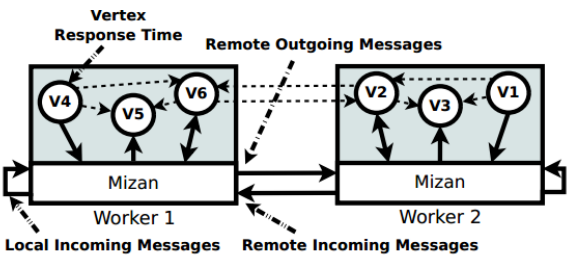
CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 67

Mizan's Migration Planning Steps



- 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



CSIE59830 Big Data Systems Large-Scale Graph Processing 1 – Pregel & Apache Hama 68

Mizan's Migration Planning Steps

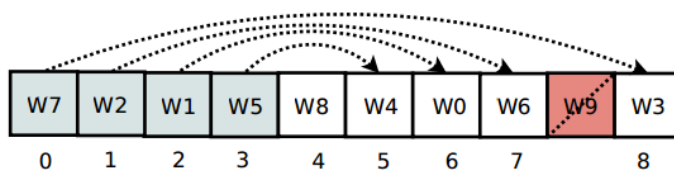


1. Identify the source of imbalance
2. **Select the migration objective:**
 - Mizan finds the **strongest cause** of workload imbalance by comparing statistics of all workers for
 - outgoing messages
 - incoming messages
 - execution time
 - The migration objective is either:
 - Optimize for outgoing 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



Mizan's Migration Planning Steps

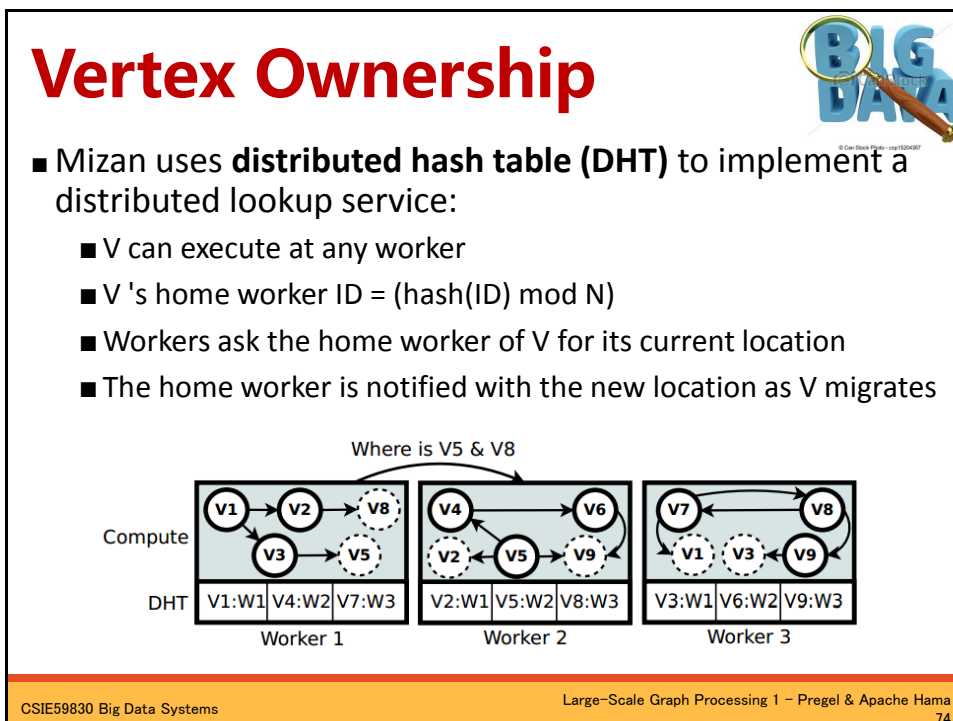
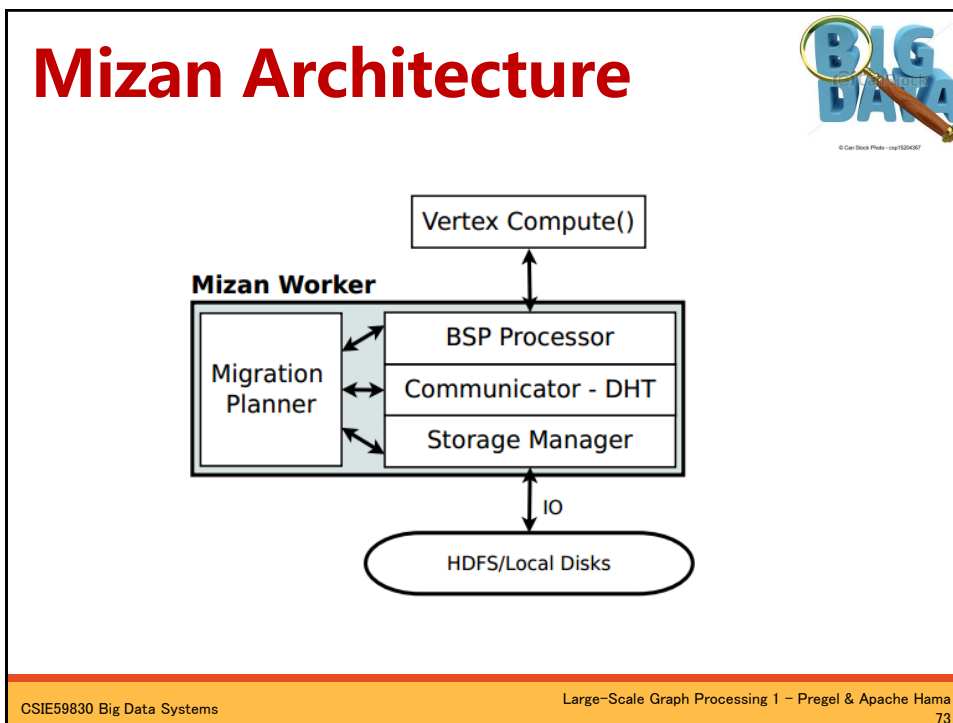


1. Identify the source of imbalance
2. Select the migration objective
3. Pair over-utilized workers with under-utilized ones
4. **Select vertices to migrate:**
 - Depending on the migration objective (same as step 2)

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?



Migrating Vertices with Large Message Size

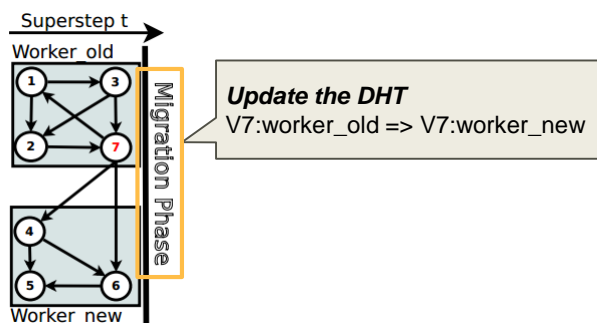


- When the graph is very large, some vertices may have tremendous number of edges.
 - The message queue may be too large to migrate

Migrating Vertices with Large Message Size



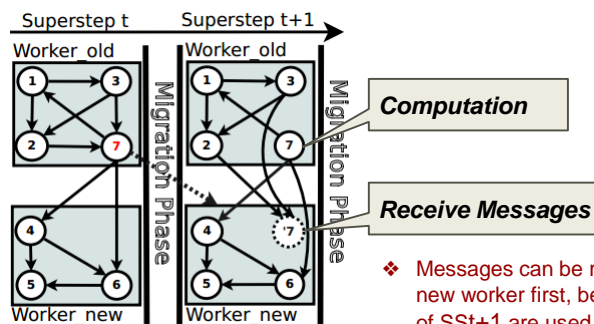
- Introduce **delayed migration** for the vertices with very large message queues
- After SS t (first migration):
Only **ownership** of vertex 7 is moved to Worker_{new}



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...

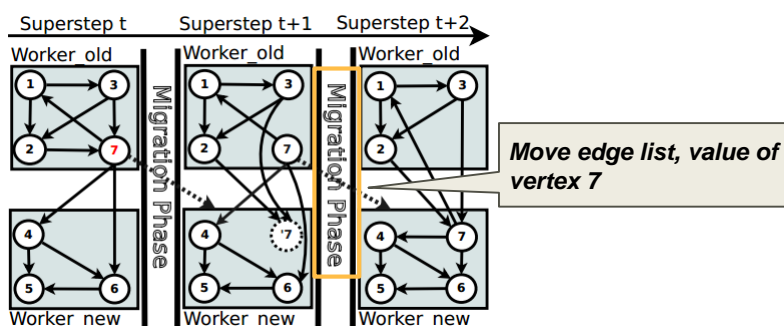


- Messages can be received by the new worker first, because the messages of SS $t+1$ are used only by SS $t+2$.

Migrating Vertices with Large Message Size



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



Mizan Summary

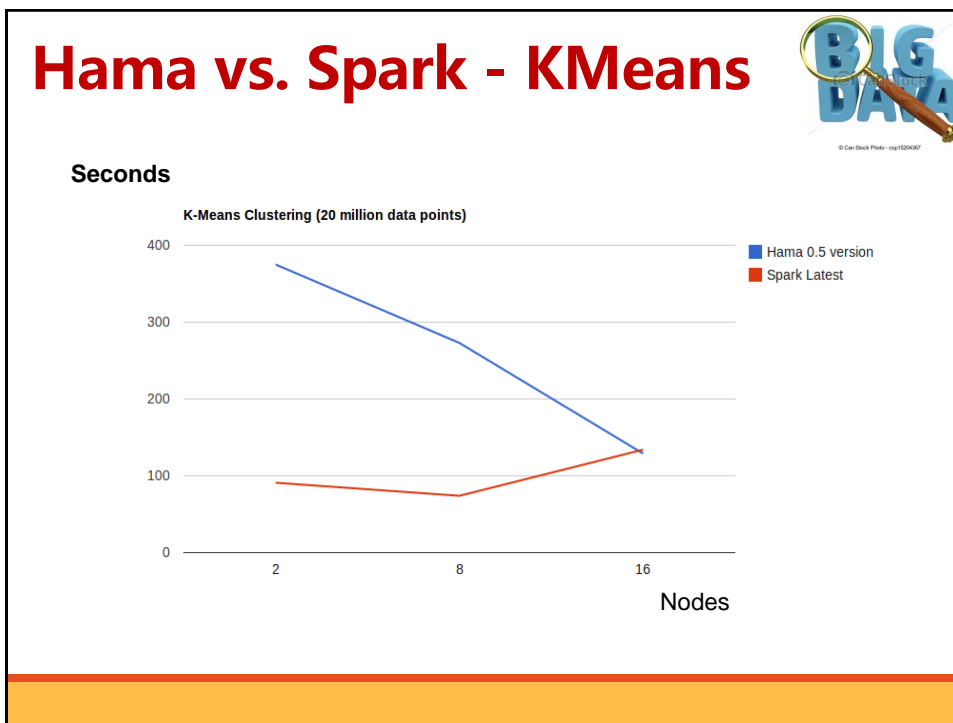


- Mizan is a Pregel system that uses fine-grained vertex migration to load balance computation and communication across supersteps
- Mizan is an open source project developed within InfoCloud group in KAUST in collaboration with IBM, programmed in C++ with MPI
- Mizan improves the overall computation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

Apache Hama



- Apache project for a graph processing system like Pregel
- Based on BSP model
- Written in Java
- A general **BSP** computing engine on Top of Hadoop



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

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



Assignment 3a



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

(You may use existing code on the Web for both exercises above.)

Assignment 3b



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 Hama program to generate a **feasible assignment** of all new grad students.
4. (Optional) Generate an **optimal assignment** with highest possible score.