Cloud Computing

Beyond MapReduce

Dell Zhang
Birkbeck, University of London
2018/19

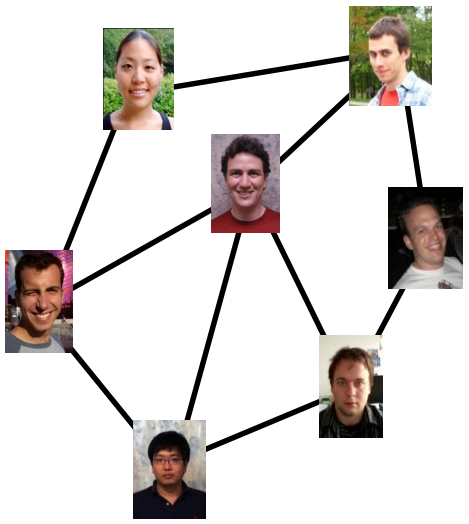
Is MapReduce enough?

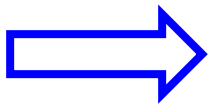
- MapReduce is a functional-like easy-to-understand paradigm
- Complex programs are not easily portable in MapReduce
- Other programming models exists

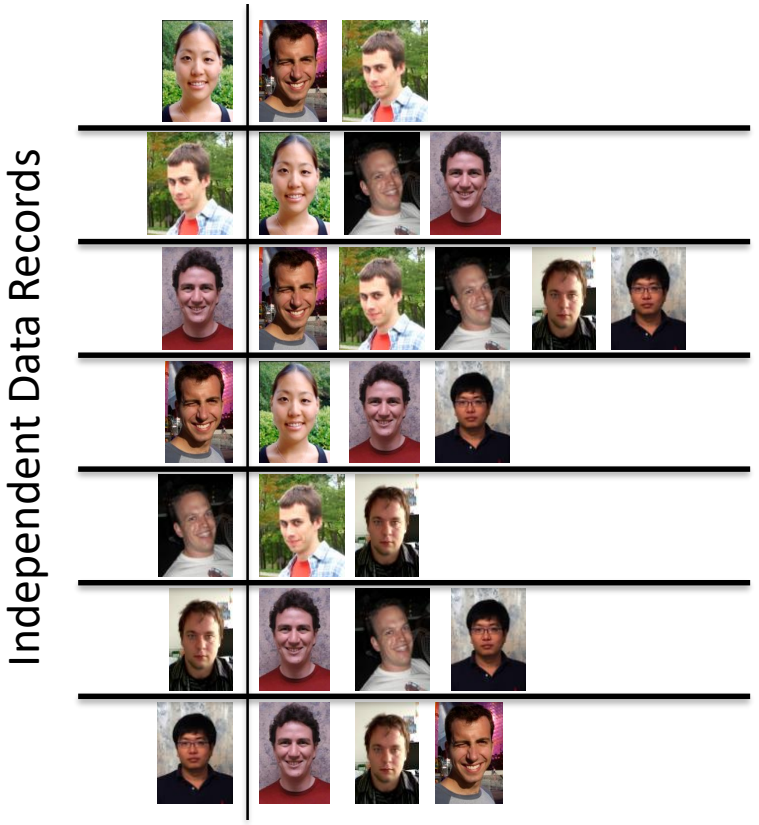
Is MapReduce enough?

- MapReduce is ill-suited for graph processing
  - Data dependencies are difficult to express
    - Substantial data transformations
    - User managed graph structure
    - Costly data replication
  - Iterative computation is difficult to optimise
    - Many iterations are needed for parallel graph processing, but Materializations of intermediate results at every MapReduce iteration harm performance.

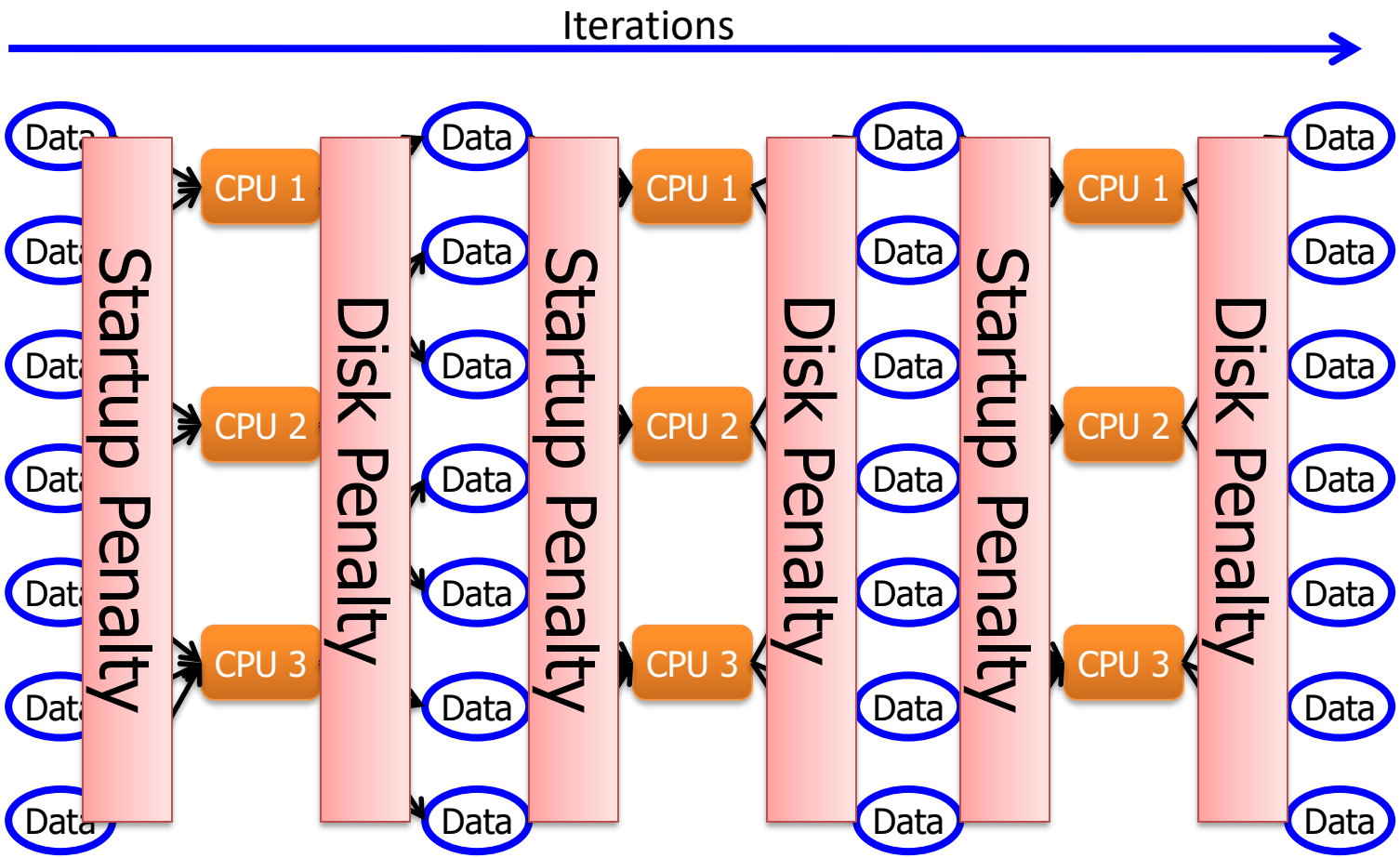
Data dependencies are difficult







Iterative computation is difficult



Example: Label Propagation

- Social Arithmetic:

  50% What I list on my profile
  40% Sue Ann Likes
  \+ 10% Carlos Like

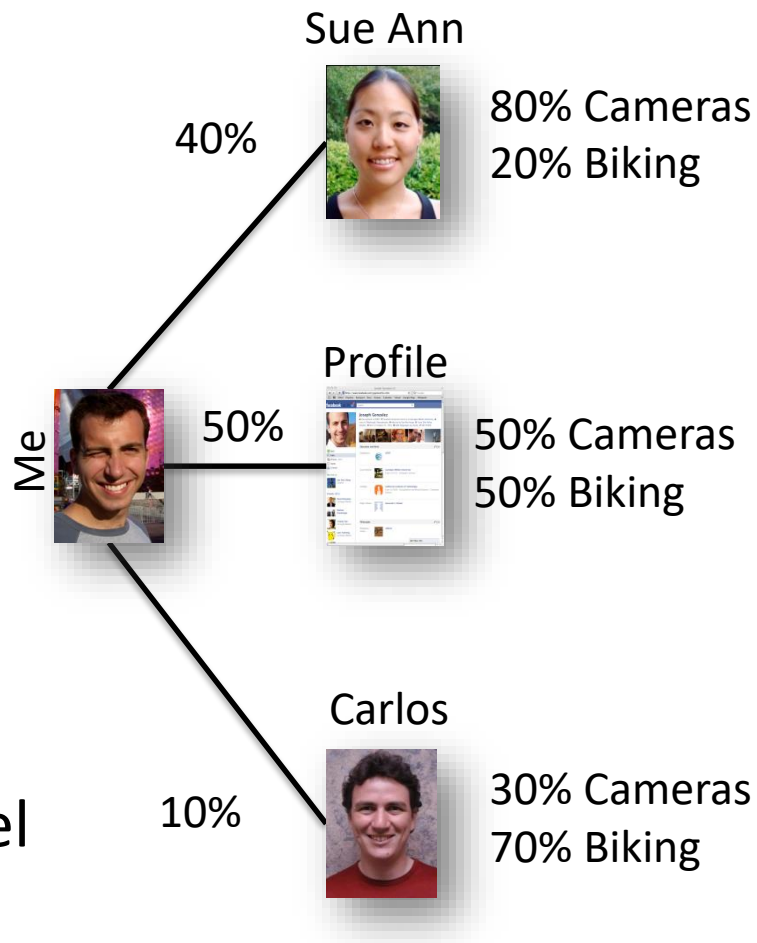
  I Like: 60% Cameras, 40% Biking

- Recurrence Algorithm:

$$Likes[i] = \sum_{j \in Friends[i]} W_{ij} \times Likes[j]$$

  – iterate until convergence

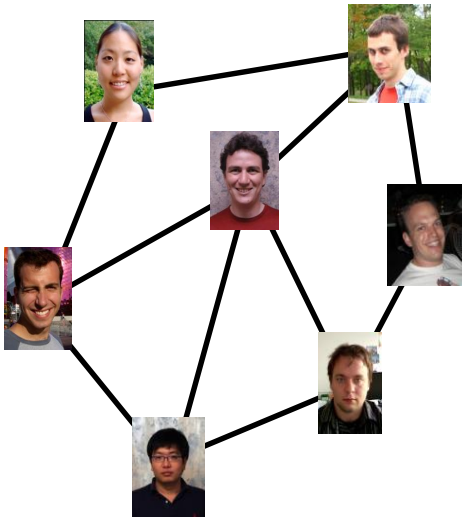
- Parallelism:
  – Compute all *Likes[i]* in parallel



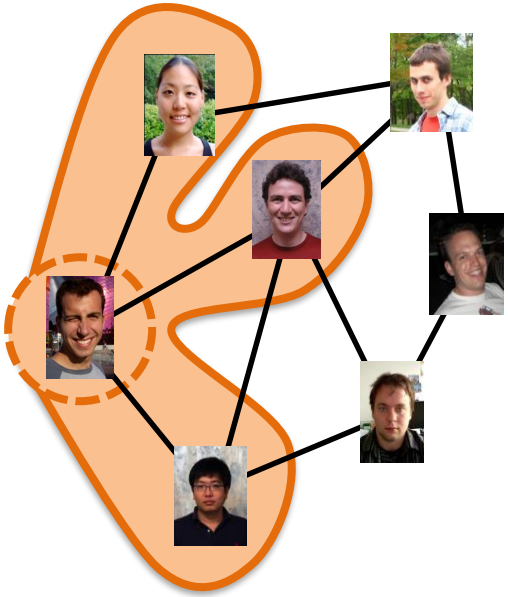
http://www.cs.cmu.edu/~zhuxj/pub/CMU-CALD-02-107.pdf

Graph Parallel Algorithms

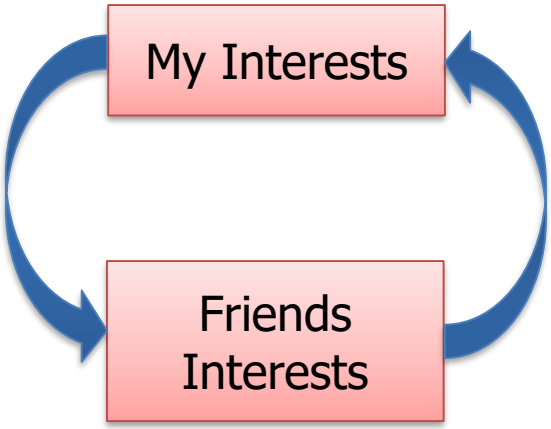
Dependency **Graph**



Local Updates



Iterative Computation



Bulk Synchronous Parallel

- The Bulk Synchronous Parallel (BSP) computing model was developed during 1980s by Leslie G. Valiant (2010 Turing Award winner)
- The definitive article:
  - Leslie G. Valiant, **A Bridging Model for Parallel Computation**, *Communications of the ACM*, Volume 33 Issue 8, Aug 1990.

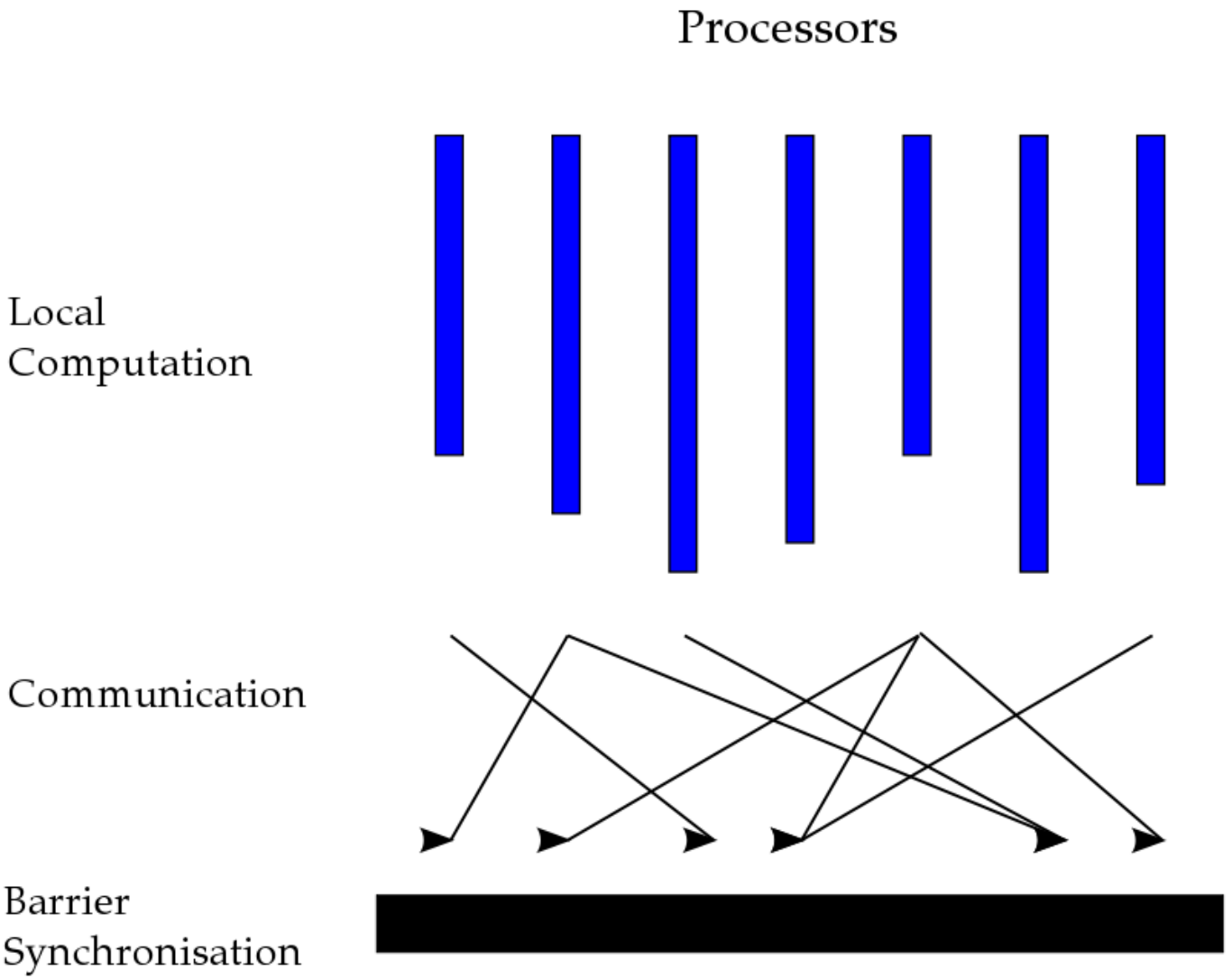
The BSP Model

- A BSP computer consists of *processors* connected by a communication network.
- Each processor has a fast local memory, and may follow different threads of computation.
- A BSP computation proceeds in a series of global *supersteps*.

The BSP Model

- A superstep consists of three components:
  - Concurrent computation
  - Communication
  - Barrier synchronisation

The BSP Model



The BSP Applications

- The BSP model has been used in the creation of a number of programming models
  - Google Pregel, Apache Giraph, Golden Orb
  - Apache Hama
- An asynchronous variant:
  - GraphLab

Pregel

- Inside Google,
  - MapReduce is used for about **80%** of all the data processing needs:
    - indexing web content, running the clustering engine for news articles, generating reports for popular queries, processing satellite imagery , language model processing for statistical machine translation, and even mundane tasks like data backup and restore.
  - The other **20%** is handled by a lesser known infrastructure called “Pregel”
    - optimized to mine relationships from graphs.

Pregel

- This system views its data as a graph
  - Each node of the graph corresponds roughly to a task (although in practice many nodes of a large graph would be bundled into a single task).
  - Each graph node generates output messages that are destined for other nodes of the graph.
  - Each graph node processes the inputs it receives from other nodes.

Pregel: Example

- All-pairs shortest path
  - Suppose our data is a collection of weighted arcs of a graph, and we want to find, for each node of the graph, the length of the shortest path to each of the other nodes.
  - Initially, each graph node a stores the set of pairs [b: w] such that there is an arc from a to b of weight w.
  - These facts are first sent to all other nodes, as triples (a, b, w).

Pregel: Example

- All-pairs shortest path (continued)
  - When the node a receives a triple (c, d, w), it looks up its current distance to c; that is, it finds the pair [c: v] stored locally, if there is one. It also finds the pair [d: u] if there is one.
  - If $w+v < u$, then the pair [d: u] is replaced by [d: $w+v$]; and if there was no pair [d: u], then the pair [d: $w+v$] is stored at the node a.
  - Also, the other nodes are sent the message (a, d, $w+v$) in either of these two cases.

PageRank in Giraph/Pregel

```
public void compute(Iterator<DoubleWritable> msgIterator) {
  double sum = 0;
  while (msgIterator.hasNext())
    sum += msgIterator.next().get();
  DoubleWritable vertexValue =
    new DoubleWritable(0.15 + 0.85 * sum);
  setVertexValue(vertexValue);

  if (getSuperstep() < getConf().getInt(MAX_STEPS, -1)) {
    long edges = getOutEdgeMap().size();
    sentMsgToAllEdges(
      new DoubleWritable(getVertexValue().get() / edges));
  } else voteToHalt();
}
```

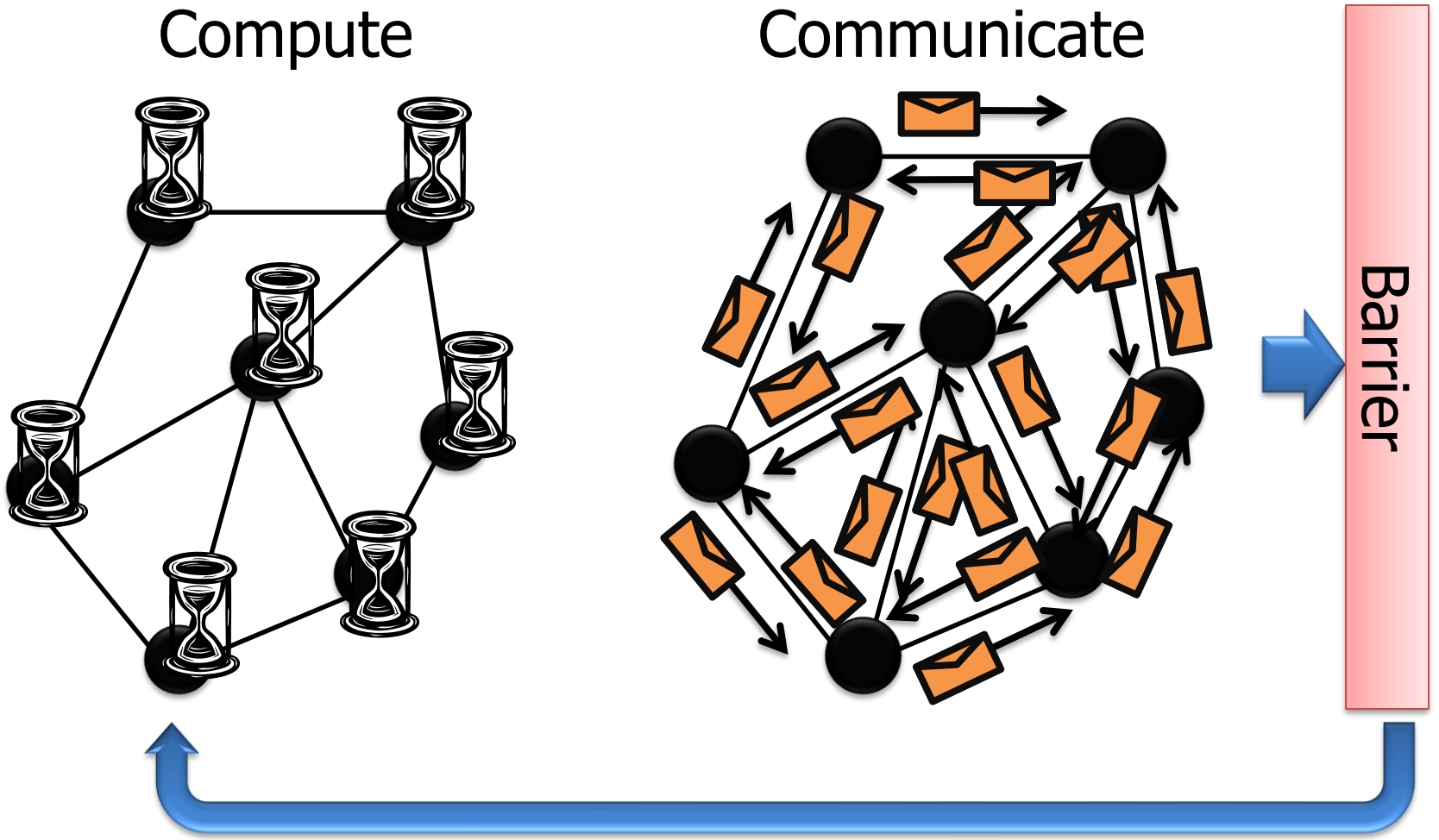
Sum PageRank over incoming messages

http://incubator.apache.org/giraph/

Pregel: Supersteps

- Computations in Pregel are organized into supersteps
- In one superstep:
  - all the messages that were received by any of the nodes at the previous superstep (or initially, if it is the first superstep) are processed,
  - all the messages generated by those nodes are sent to their destination.

Pregel: Supersteps

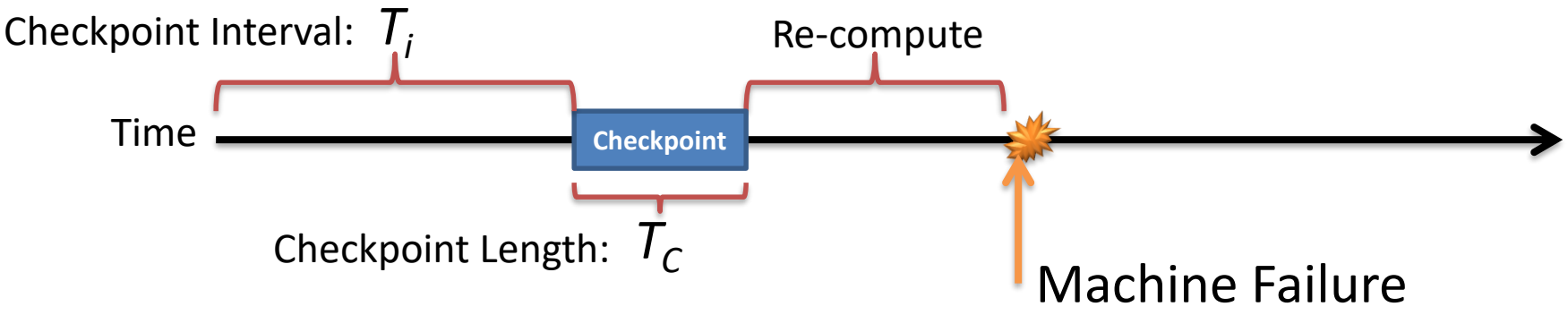


http://dl.acm.org/citation.cfm?id=1807184

Pregel: Checkpoints

- Pregel *checkpoints* its entire computation after some of the supersteps.
  - The probability of a failure during that number of supersteps should be low.
  - A checkpoint consists of making a copy of the entire state of each task.
  - In case of a compute-node failure, the entire job (rather than the failed tasks) is restarted from the most recent checkpoint.

Pregel: Checkpoints



Tradeoff:

– **Short** T_i: Checkpoints become too costly

– **Long** T_i : Failures become too costly

Pregel: Checkpoints

- Optimal Checkpoint Intervals

$$T_i \approx \sqrt{2 T_c T_{\text{mtbf}}}$$

Checkpoint Interval — T_i

Length of Checkpoint — T_c

Mean time between failures — T_{mtbf}

  - For example:
    - 64 machines with a per machine MTBF of 1 year
    - T_{mtbf} = 1 year / 64 ≈ 130 Hours
    - T_c = of 4 minutes
    - T_i ≈ of 4 hours

http://dl.acm.org/citation.cfm?id=361115

- It addresses the limitations of BSP (Pregel)
  - Use graph structure
    - Automatically manage the movement of data
  - Focus on **Asynchrony**
    - Computation runs as resources become available
    - Use the most recent information
  - Support Adaptive/Intelligent Scheduling
    - Focus computation to where it is needed
  - Preserve Serializability
    - Provide the illusion of a sequential execution
    - Eliminate “race-conditions”

Take Home Messages

- Bulk Synchronous Parallel (BSP)
  - Google Pregel