Graph Processing & Bulk Synchronous Parallel Model

CompSci 590.03
Instructor: Ashwin Machanavajjhala
Recap: Graph Algorithms

• Many graph algorithms need iterative computation

• No native support for iteration in Map-Reduce
  – Each iteration writes/reads data from disk leading to overheads
  – Need to design algorithms that can minimize number of iterations
This Class

• Iteration Aware Map-Reduce

• Pregel (Bulk Synchronous Parallel Model) for Graph Processing
ITERATION AWARE MAP-REDUCE
Iterative Computations

PageRank:

\[
do \\
p^{next} = (cM + (1-c) U)p^{cur} \\
while(p^{next} \neq p^{cur})
\]

- Loops are not supported in Map-Reduce
  - Need to encode iteration in the launching script
- M is a loop invariant. But needs to written to disk and read from disk in every step.
- M may not be co-located with mappers and reducers running the iterative computation.
HaLoop

- Iterative Programs

\[ R_{i+1} = R_0 \cup (R_i \bowtie L) \]

- Initial Relation
- Invariant Relation

The iterative programs that HaLoop supports can be distilled into the following core construct:

\[ R_{i+1} = R_0 \cup (R_i \bowtie L) \]

This equation represents the general form of the recursive programs we support and presents the general form of the types of iterative programs that HaLoop supports. Here, we introduce the following functions.

- **SetMaxNumOfIterations**: This function sets the maximum number of iterations of the loop. To write a HaLoop program, a programmer specifies the loop body, the loop control, and the maximum number of iterations.

- **AddMap**: This function adds a new parameter for cached invariant values associated with the key.

- **Reduce**: This function reduces the tuple into intermediate tuples. The interface contains a termination condition that is used to control the loop.

- **SetFixedPointThreshold**: This function sets the approximate fixpoint threshold. To specify the loop control, the programmer needs to specify a termination condition and loop-invariant data.

- **ResultDistance**: This function calculates the distance between the input and the last iteration, since the input files to different iterations may be different. For example, in Example 1, at each iteration, since the input files to different iterations may be different, the input is calculated differently. For example, in Example 1, at each iteration, since the input files to different iterations may be different, the input is calculated differently.

- **Remote communication**: This function provides further control of the loop.

- **MapReduce framework**: This function leverages data locality in these applications. In order to accommodate the requirements of iterative data analysis and complex analytics, both kinds of approximate fixpoints are useful for expressing convergence conditions in machine learning and complex analytics.

- **ResultDistance**: This function calculates the distance between the input and the last iteration, since the input files to different iterations may be different.
Loop aware task scheduling

- Inter-Iteration Locality
- Caching and Indexing of invariant tables

![Diagram of loop aware task scheduling]

Lecture 13: 590.02 Spring 13
iMapReduce

- Reduce output is directly sent to mappers, instead of writing to distributed file system.
- Loop invariant is loaded onto the maps only once.
PREGEL
Seven Bridges of Konigsberg
Pregel Overview

• Processing occurs in a series of supersteps

• In superstep S:
  Vertex may read messages sent to V in superstep S-1
  Vertex may perform some computation
  Vertex may send messages to other vertices

• Vertex computation within a superstep can be arbitrarily parallelized.

• All communication happens between two supersteps
Pregel

- Input: A directed graph $G$. Each vertex is associated with an id and a value. Edges may also contain values.

- Edges are not a first class citizen – they have no associated computation
  - Vertices can modify its state/edge state/edge set

- Computation finishes when all vertices enter the inactive state
### Example

**Figure 2: Maximum Value Example.** Dotted lines are messages. Shaded vertices have voted to halt.

![Graph Diagram](image)

- **Superstep 0**
  - Vertex values: 3, 6, 2, 1

- **Superstep 1**
  - Vertex values: 6, 6, 2, 6

- **Superstep 2**
  - Vertex values: 6, 6, 6, 6

- **Superstep 3**
  - Vertex values: 6, 6, 6, 6

Each vertex can send messages to its neighbors, and messages are received by their destination vertex. The maximum value is propagated to all vertices in each superstep.
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();
};
Vertex API

• MessageIterator contains all the messages received.

• Message ordering is not guaranteed, but all messages are guaranteed to be delivered without duplication.

• Vertices can also send messages to other vertices (whose id it knows from prior messages)

• No need to explicitly maintain an edgeset.
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();
        }
    }
};
Combiners

• If messages are aggregated ("reduced") using an associative and commutative function, then the system can combine several messages intended for a vertex into 1.

• Reduces the number of messages communicated/buffered.
Single Source Shortest Paths

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();
  }
};

class MinIntCombiner : public Combiner<int> {
  
  virtual void Combine(MessageIterator* msgs) {
    int mindist = INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    Output("combined_source", mindist);
  }
};

Lecture 14 : 590.02 Spring 13
Aggregation

• Global communication

• Each vertex can provide a value to an aggregator in a superstep $S$. Resulting value is made available to all vertices in superstep $S+1$.

• System aggregates these values using a reduce step.
Topology Mutations

- Compute function can add or remove vertices
- But this can cause race conditions
  - Vertex 1 creates an edge to vertex 100
    Vertex 2 deletes vertex 100
  - Vertex 1 creates vertex 100 with value 10
    Vertex 2 also creates vertex 100 with value 12

- Partial Order on operations
  - Edge removal < vertex removal < vertex add < edge add (< means earlier)

- Handlers for conflicts
  - Default: Pick a random action
  - Can specify more complex handlers
PREGEL ARCHITECTURE
Graph Partitioning

- Vertices are assigned to machines based on \( \text{hash}(\text{vertex.id}) \mod N \)

- Can define other partitions: co-locate all web pages from the same site

- Sparsest Cut Problem: minimize the edges across partitions
Processing

• Master coordinates a set of workers.
  – Determines the number of partitions
  – Determines assignment of partitions to workers

• Worker processes one or more partitions
  – Workers know the entire set of partition to worker assignments and the partition function
  – All vertices in Worker’s partition are initialized to active
  – Worker loops through vertex list and sends any messages asynchronously
  – Worker notifies master of # active vertices at the end of a superstep
Fault Tolerance

• Checkpoint: master instructs workers to save state to persistent storage (e.g. HDFS)
  – Vertex values
  – Edge values
  – Incoming messages

• Master saves to disk aggregator values

• Worker failure is detected using a heartbeat.

• New worker is created using state from previous checkpoint (which could be several supersteps before current superstep)
Summary

• Map-reduce has no native support for iterations
  – No Loop construct
  – Write to disk and read from disk in each step, even if the data is an invariant in the loop.

• Systems like HaLoop introduce inter-iteration locality and caching to help iterations on map-reduce.

• Pregel is a vertex oriented programming model and system for graph processing with built in features for iterative processing on graphs.