Apache HAMA: An Introduction to Bulk Synchronization Parallel on Hadoop (Ver 0.1)

Hyunsik Choi <hyunsik.choi@gmail.com>,
Edward J. Yoon <edwardyoon@apache.org>,

Apache Hama Team

Table Of Contents

An Overview of the BSP Model

The BSP (Bulk Synchronous Parallel) is a parallel computational model that performs a series supersteps.

A superstep consists of three ordered stages:

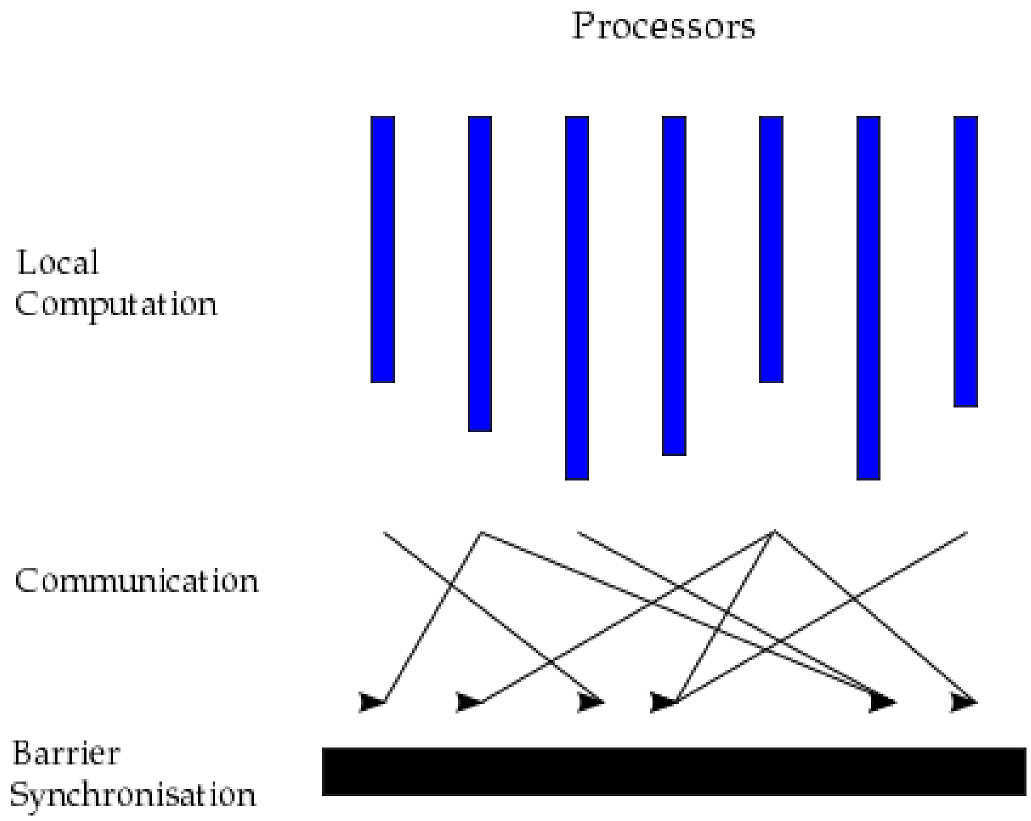
- Concurrent Computation: Computation on locally stored data
- Communication: Send and Receive messages in a point-to-point manner
- Barrier Synchronization: Wait and Synchronize all processors at end of superstep

A BSP system is composed of a number of networked computers with both local memory and disk.

More detailed information

- http://en.wikipedia.org/wiki/Bulk_synchronous_parallel

A Vertical Structure of Superstep



Why the BSP on Hadoop?

We want to use Hadoop cluster for more complicated computations

- But, the MapReduce is not easy to program complex mathematical and graphcial operations
  - e.g., Matrix Multiply, Page-Rank, Breadth-First Search, ..., etc
- Preserving data locality is an important factor in distributed/parallel environments.
  - Data locality is also important in MapReduce for efficient processing.
- However, MapReduce frameworks do not preserve data locality in consecutive operations due to its inherent natures.

A Comparison to M/R Programming model

Map/Reduce does any filtering and/or transformations while each mapper is reading input data split by InputFormat.

- Simple API
  - Map and Reduce
- Automatic parallelization and distribution
- During MapReduce processing, it generally passes input data through either many passes of MapReduce or MapReduce iteration in order to derive final results.
- Partitioner is too simple
  - MR does not provide an way to preserve data locality either a transition from Map to Reduce or between MapReduce iteration.
  - It incurs not only intensive communication cost but also unnecessary processing cost.

A Comparison to M/R Programming model

The BSP is tailored towards processing data with locality.

- The BSP enables each peer to communicate only necessary data to other peers while peers are preserving data locality.
- Simple API and Easy Synchronization
- Like MapReduce, it makes programs to be automatically parallelized and distributed across cluster nodes.

What Applications are Appropriate for BSP?

- BSP will show its innate capability in applications with following characteristics:
  - Important to data locality
  - Processing data with complicated relations
  - Using many iterations and recursions

The BSP Implementation on Hadoop

- Written in Java
- The BSP package is now available in the Hama repository.
  - Implementation available for Hadoop version greater than 0.20.x
  - Allows to develop new applications
- Hadoop RPC is used for BSP peers to communicate each other.
- Barrier Sync mechanism is helped by zookeeper.

Serialize Printing of Hello World (shows synchronization)

```
// BSP start with 10 threads.
// Each thread will have a shuffled ID number from 0 to 9.

for (int i = 0; i < numberOfPeer; i++) {
    if (myId == i) {
      System.out.println("Hello BSP world from " + i + " of " + numberOfPeer);
    }

    peer.sync();
}

// BSP end
```

Breath-first Search on Graph

```
public void expand(Map<Vertex, Integer> input, Map<Vertex, Integer> nextQueue) {
    for (Map.Entry<Vertex, Integer> e : input.entrySet()) {
      Vertex vertex = e.getKey();
      if (vertex.isVisited() == true) { // only visit a vertex that has never been visited.
        continue;
      } else {
        vertex.visit();
      }

      // Put vertices adjacent to current vertex into nextQueue with increment distance.
      for (Integer i : vertex.getAdjacent()) {
        if (needToVisit(i, vertex, input.get(e.getKey()))) {
          nextQueue.put(getVertexByKey(i), e.getValue() + 1);
        }
      }
    }
}
```

Breath-first Search on Graph

```
public void process() {
     currentQueue = new HashMap<Vertex, Integer>();
     nextQueue = new HashMap<Integer, Integer>();
     currentQueue.put(startVertex,0); // initially put a start vertex into the current queue.

     while (true) {
       expand(currentQueue, nextQueue);

       // The peer.sync method can determine if certain vertex resides
       // on local disk according to vertex id.
       peer.sync(nextQueue); // Synchronize nextQueue with other peers.
                             // At this step, the peer communicates vertex IDs corresponding to
                             // vertices that resides on remote peers.
      if (peer.state == TERMINATE)
        break;
      // Convert nextQueue that contains vertex IDs into currentQueue queue that contains vertices.
      currentQueue = convertQueue(nextQueue);
      nextQueue = new HashMap<Vertex, Integer>();
    }
}
```

What's Next?

- Novel matrix computation algorithms using BSP
- Angrapa
  - A graph computation framework based on BSP
  - API familiar with graph features
- More improved fault tolerance for BSP
  - Now, BSP has poor fault tolerance mechanisms.
- Support collective and compressed communication mechanisms for high performance computing

Who We Are

- **Apache Hama** Project Team
  - (http://incubator.apache.org/hama)
- Mailing List
  - hama-user@incubator.apache.org
  - hama-dev@incubator.apache.org