Integrating Pig with Harp to Support Iterative Applications with Fast Cache and Customized Communication

Tak Lon (Stephen) Wu, Abhilash Koppula, Judy Qiu

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

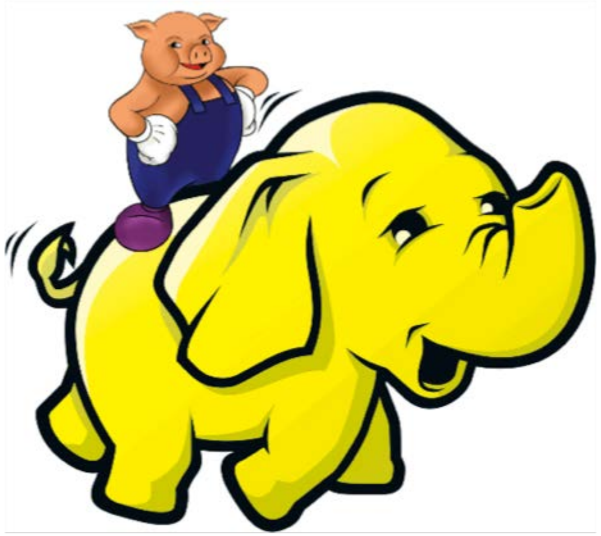
Outline

- Apache Hadoop and Pig background
- Performance issue
  – Pig's overhead
  – Pig in supporting iterative applications
- Solution
  – Pig with Harp (Pig+Harp) integration and performance
- Conclusion

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Hadoop and Pig

- Hadoop
  - Hadoop has been widely used by many fields of research and commercial companies
    - Machine Learning, Text Mining, Bioinformatics, etc.
    - Facebook, Amazon, LinkedIn, etc.
  - Java is one of the main stream languages for distributed systems
    - Apache Storm, Apache HBase, Apache Cassandra, etc.
- Pig
  - Procedural language and straightforward syntax
  - Runs directly on top of Hadoop
  - Automatic parallelism
  - Works with HDFS and HBase



Types of Pig Application

- Exact, Transform, Load (ETL)
  - Join, (Co)Group, Union, etc.
  - Raw Data analysis: daily log analysis
  - NoSQL Database queries
- Statistical data analysis
  - Means, median, standard deviation, etc.
- Data mining
  - K-means clustering

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

WordCount Example in Hadoop

```java
public class WordCount {
  public static class Map
          extends Mapper<LongWritable, Text, Text, IntWritable>{
   private final static IntWritable one = new IntWritable(1); // type of output value
   private Text word = new Text();   // type of output key
   public void map(LongWritable key, Text value, Context context
               ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString()); // line to string token
    while (itr.hasMoreTokens()) {
     word.set(itr.nextToken());    // set word as each input keyword
     context.write(word, one);     // create a pair <keyword, 1>
    }
   }
  }
```

Map

```java
  public static class Reduce
      extends Reducer<Text,IntWritable,Text,IntWritable> {
   private IntWritable result = new IntWritable();
   public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0; // initialize the sum for each keyword
    for (IntWritable val : values)
     { sum += val.get();  }
    result.set(sum);
    context.write(key, result); // create a pair <keyword, number of occurences>
   }
  }
```

Reduce

```java
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs(); // get all
args
    for (int i = 0; i < otherArgs.length; i++)
                      System.out.println(i + " " +otherArgs[i]);
    Job job = new Job(conf, "wordcount");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    job.setCombinerClass(Reduce.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[1]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[2]));
    //Wait till job completion
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Driver

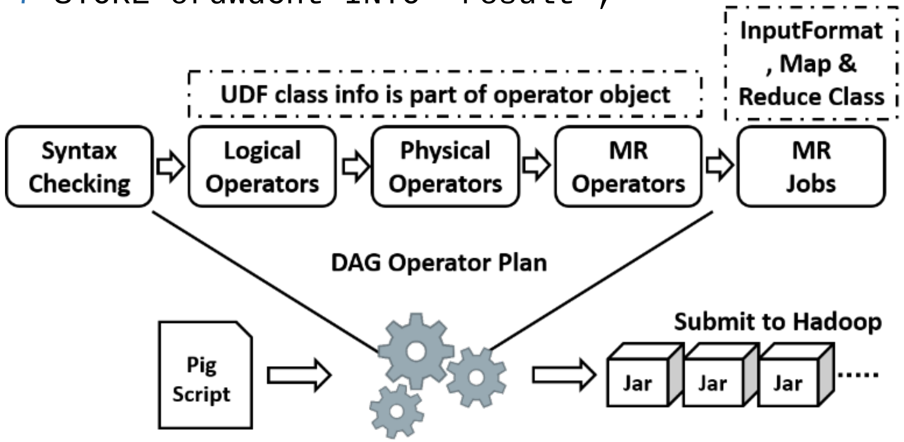
SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

- 48 lines of code not including library import lines

Pig WordCount

- Fewer lines of code
- Data is converted into Pig data types: bag, tuple and field.
- Data transformation is handled by built-in operators or UDF.
- Compile into Hadoop job(s) as jar file(s)
- DAG execution dataflow/pipeline
- Jobs are submitted sequentially

```
input     = LOAD 'input.txt' AS
            (line:chararray);
words     = FOREACH input GENERATE
            FLATTEN(TOKENIZE(line)) AS word;
filWords = FILTER words BY word MATCHES '\\w+';
wdGroups = GROUP filWords BY word;
wdCount  = FOREACH wdGroups GENERATE group AS
            word, COUNT(filWords) AS count;
ordWdCnt = ORDER wdCount BY count DESC;
STORE ordWdCnt INTO 'result';
```



SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

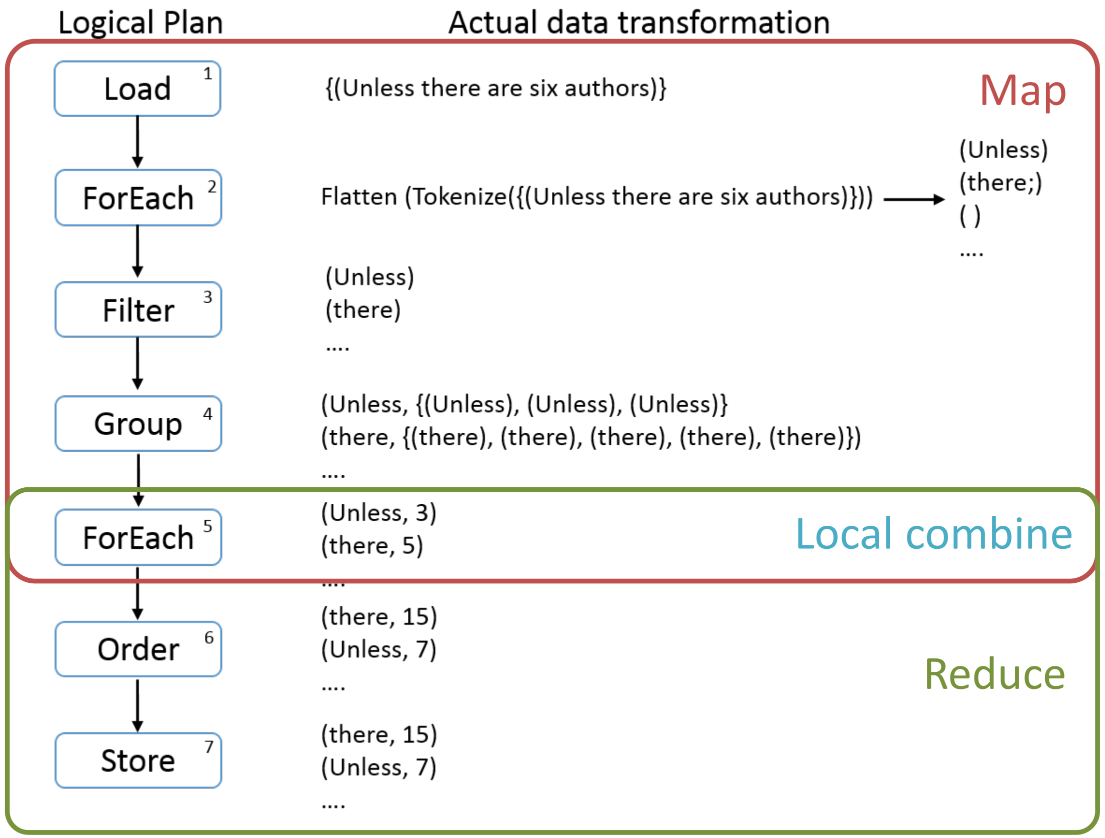
Outline

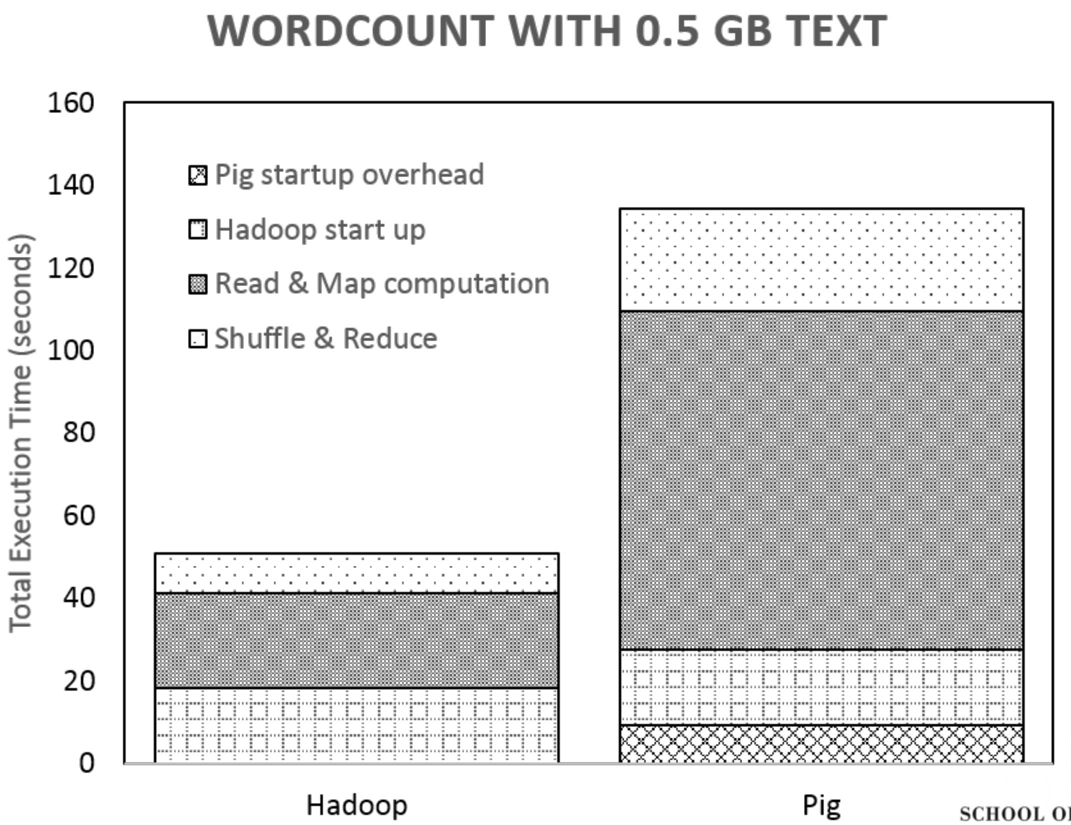
- Apache Hadoop and Pig background
- Performance issue
  – Pig's overhead
  – Pig in supporting iterative applications
- Solution
  – Pig with Harp (Pig+Harp) integration and performance
- Conclusion

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Pig's Computation Overhead

- Pig's Tuple-based (record-based) computation is slower than Hadoop
  - Overall execution time is about 2+ times slower

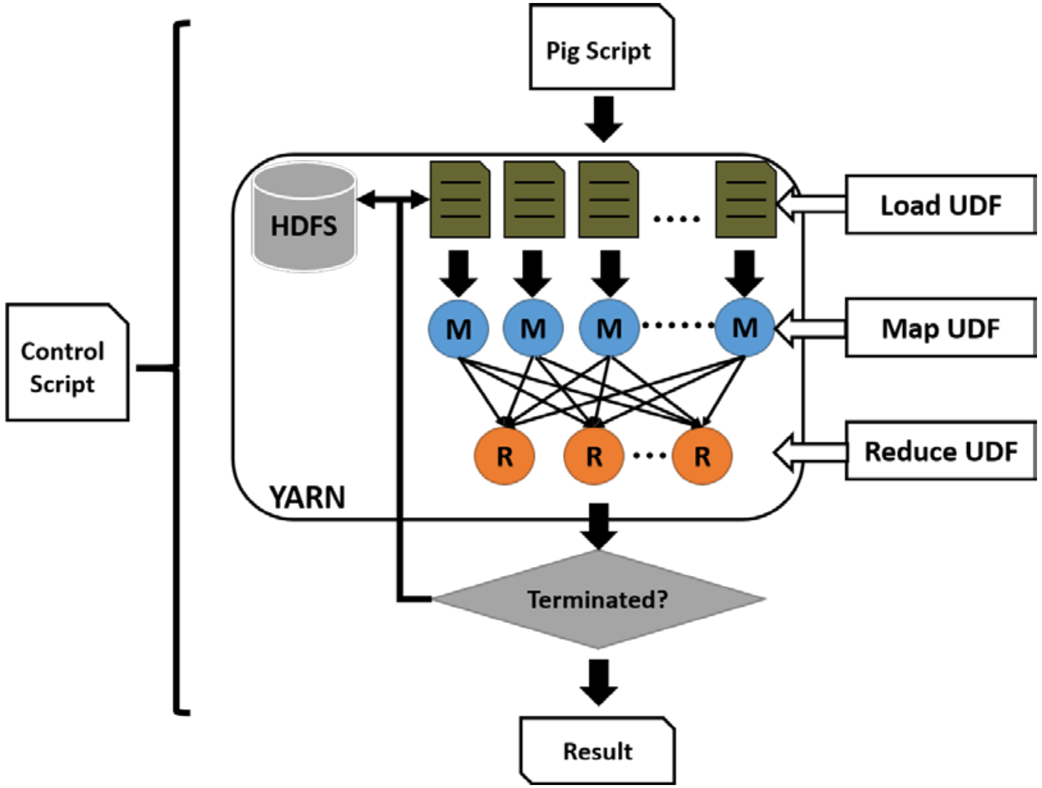




SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Pig and Iterative Applications

- Need a wrapper program to support conditional loop
- Intermediate results of iterations are mapped from disk to next iteration
  - Disk cache and Disk I/O are substantial
- Hadoop Jobs restart overhead
- No in-memory caching mechanism
- Data partitions are based on Pig Input Format
- Tuple-based data transformation/computation is slow



SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Outline

- Apache Hadoop and Pig background
- Performance issue
  - Pig's overhead
  - Pig in supporting iterative applications
- Solution
  - Pig with Harp (Pig+Harp) integration and performance
- Conclusion

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

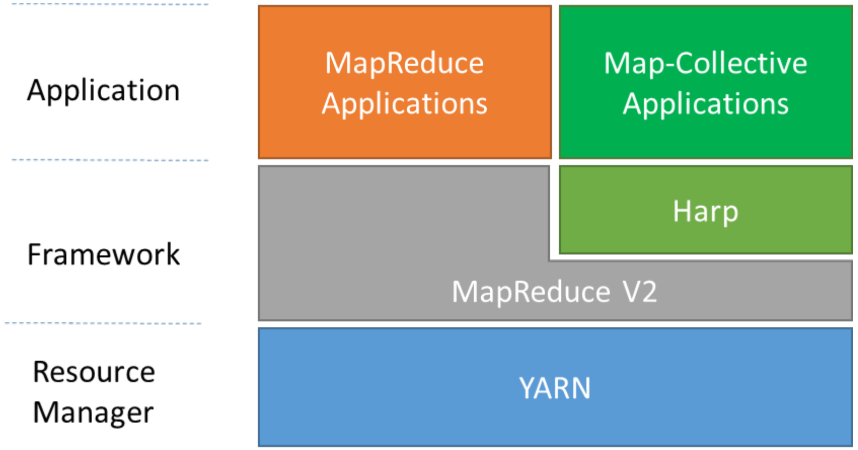
Improvement?

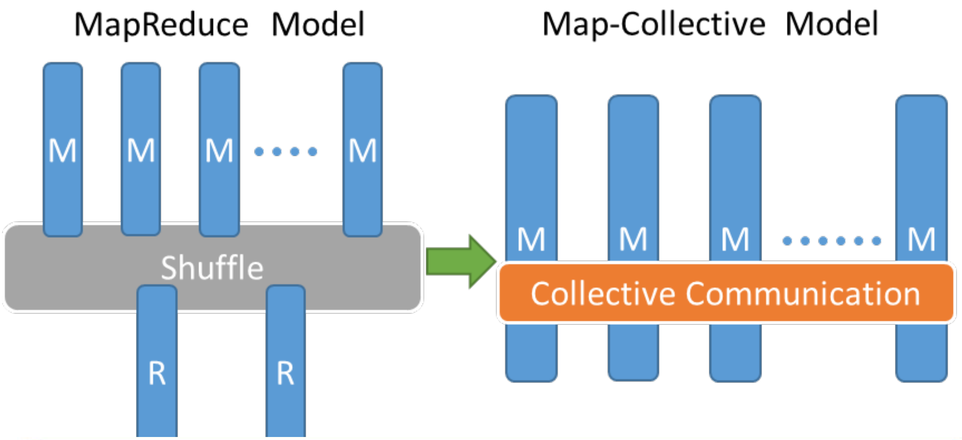
- Avoid tuple-based computation
  - Easy fix by optimizing LOAD UDF
- Need loop-awareness support
- In-memory caching for reused data among iterations

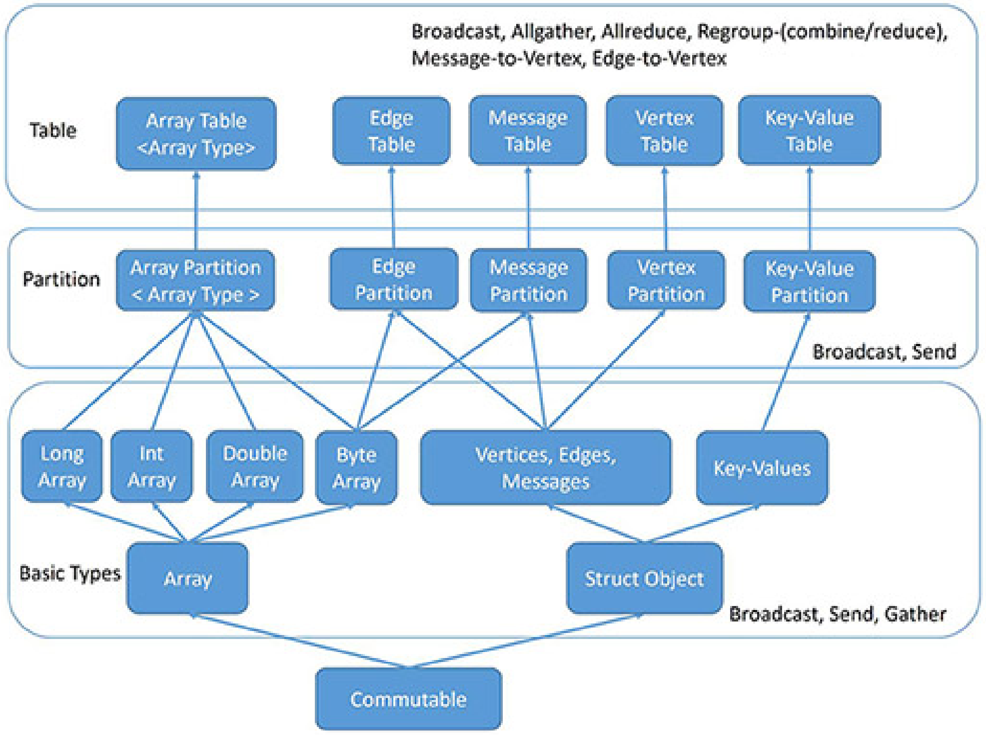
SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Harp: A Hadoop Plugin

- Plug-and-play Hadoop plugin
- Enable loop awareness for iterative applications
- Multi-thread and Multi-process computing
- In-memory object caching
- MPI-like and graph collective communication
- Pure Java implementation



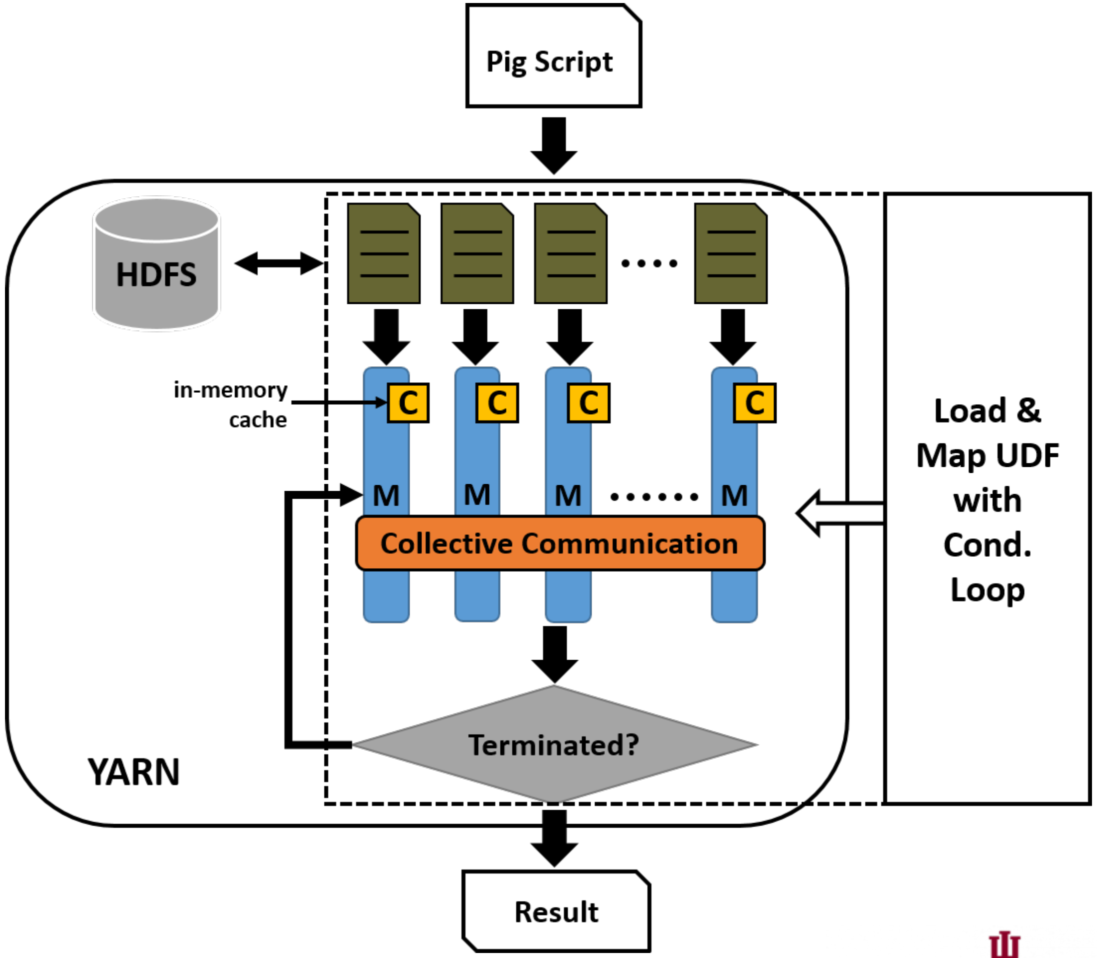




*Apache Harp project: http://salsaproj.indiana.edu/harp

Solution: Pig+Harp

- Replace default mapper interface with Harp's MapCollective long-running mapper
- Read once, Compute many
- In-memory objects caching in LOAD & MAP stages' UDF
- Shuffle data by calling Harp's collective communication API
- UDF controls loop termination
- No-hassle plugins
  - Same as general Pig if collective communication is not written in UDF



SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Applications and Benchmarking

- Madrid Cluster (before update)
  - 8-node cluster with an extra head node
  - 4 x AMD Opteron 8356 2.30GHz with 4 cores
  - 16GB RAM per node
  - 1Gbps Ethernet network
  - Red Hat Enterprise 6.5s
- Hadoop 2.2.0
- Harp 0.1.0
- Pig 0.12.0
- K-means clustering on large dataset
  - Fixed computation ratios (50 Billion 4D data points computation per node) but various memory and communication usage aspects
- PageRank
  - Strong scaling test on a dataset with 2 million random vertices

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

K-means

- Pig K-means
  - An external python loop-control wrapper
  - Data points and centroids are reloaded each iteration
  - Batch computation right after data loading
  - Default GROUP BY aggregation
- Pig+Harp K-means
  - Extends from Pig's LOAD interface
  - Reads data as file directly from HDFS.
  - Data points and centroids are cached as in-memory objects
  - Batch computation right after data loading
  - Sync intermediate centroids by using AllReduce communication

Pig K-means

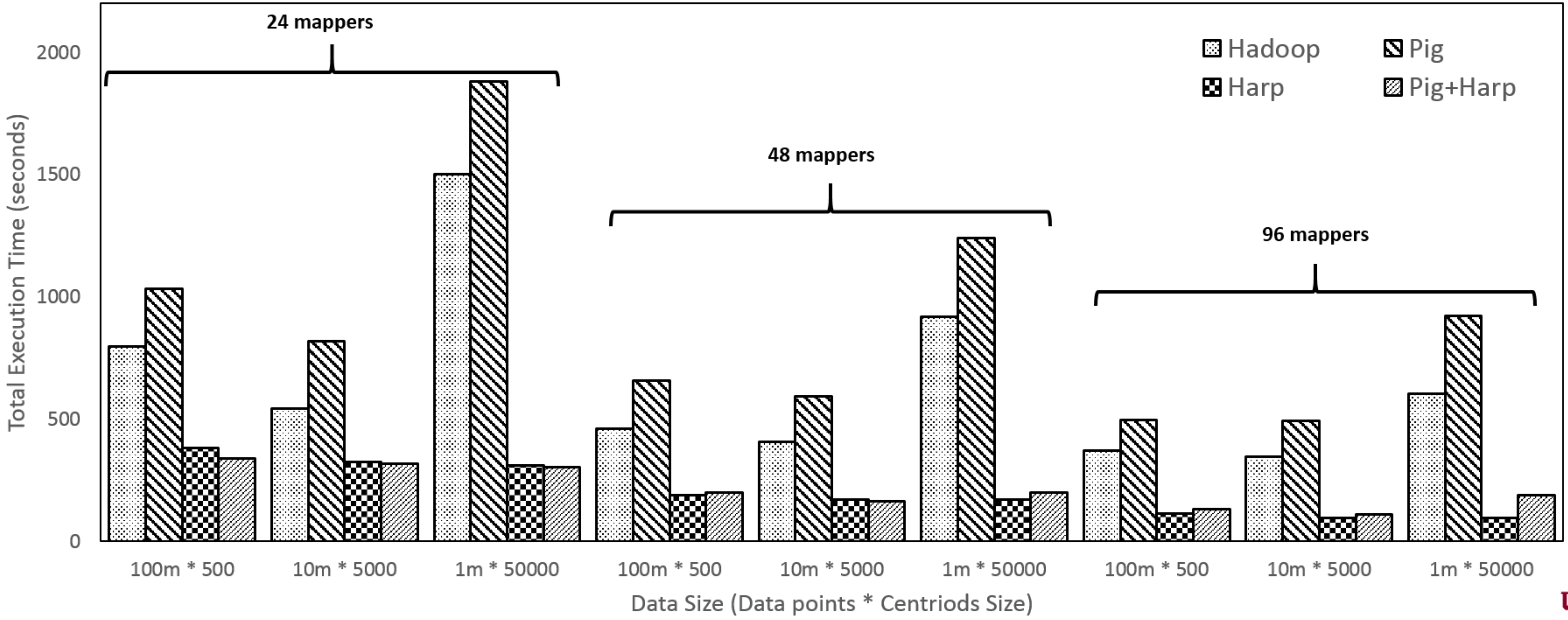
```
raw     = LOAD $hdfsInputDir using
          PigKmeans('$centroids',
          '$numOfCentroids') AS (datapoints);
dptsBag = FOREACH raw GENERATE
          FLATTEN(datapoints) as dptInStr;
dpts    = FOREACH dptsBag GENERATE
          STRSPLIT(dptInStr, ',', 5) AS
          splitedDP;
grouped = GROUP dpts BY splitedDP.$0;
newCens = FOREACH grouped GENERATE
          CalculateNewCentroids($1);
STORE newCens INTO 'output';
```

Pig+Harp K-means

```
centds = LOAD $hdfsInputDir using
         HarpKmeans('$initCentroidOnHDFS',
         '$numOfCentroids', '$numOfMappers',
         '$iteration', '$jobID', '$Comm') as
         (result);
STORE centroids INTO '$output';
```

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

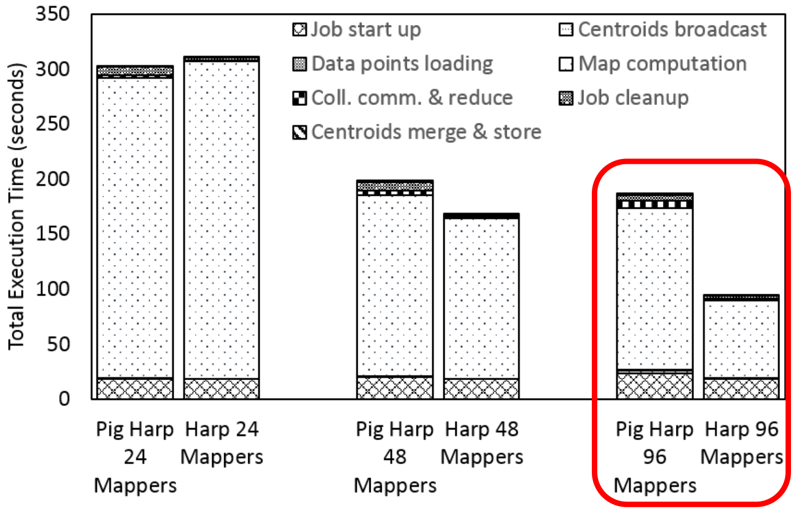
K-means Performance



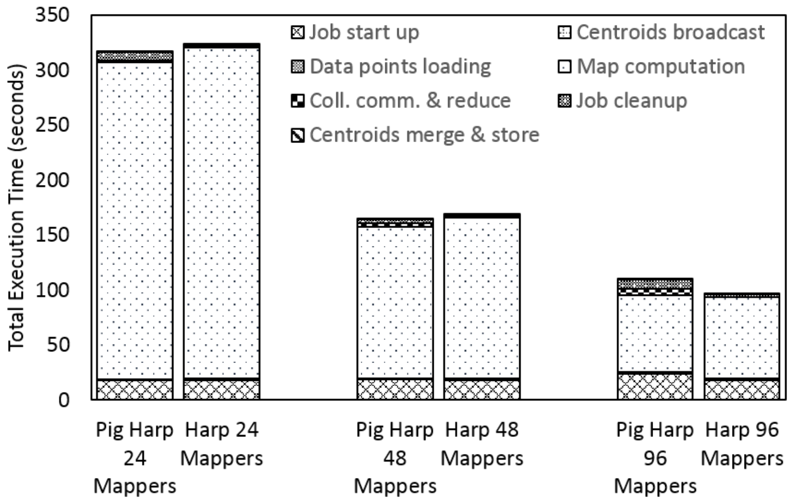
SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY Bloomington

K-means Performance (cont.)

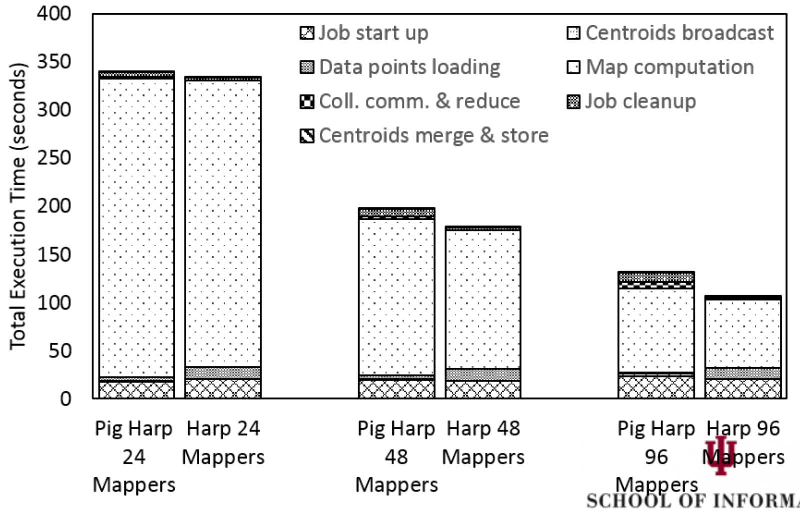
- Harp K-means is written in multi-thread model; meanwhile, Pig+Harp is written in multi-process model
- Pig+Harp 1m 50K 96 mappers runs 2 times slower than Harp’s multi-thread computation
  - L2 & L3 cache effect of in-memory caching



1m data points 50k centroids



10m data points 5k centroids



100m data points 500 centroids

SCHOOL OF INFORMATICS AND COMPUTING

PageRank

- Pig PageRank
  - An external java loop-control wrapper
  - PageRank adjacent matrix is reloaded each iteration
  - Compute with built-in operators except data loading
  - Tuple-based computation

- Pig+Harp PageRank
  - Extend from Pig's LOAD interface
  - Reads data as file directly from HDFS
  - Data points are cached as in-memory objects
  - Batch computation right after loading
  - Sync intermediate page rank values by using AllGather communication

```
1 raw       = LOAD '$InputDir' USING
              CmLoader('$noOfURLs','$itrs') as
              (source,pagerank, out:bag{});
2 prePgRank = FOREACH raw GENERATE FLATTEN(out)
              as source, pagerank/SIZE(out) as
              pagerank;
3 newPgRank = FOREACH (COGROUP raw by source,
              prePgRank by source OUTER)GENERATE
              group as source, (1-$dpFactor) +
              $dpFactor*(SUM(prePgRank.pagerank)
              IS NULL?0:SUM(prePgRank.pagerank))
              as pagerank, FLATTEN(raw.out)
              as out;
4 STORE newPgRank INTO '$outputFile';
```

Pig PageRank

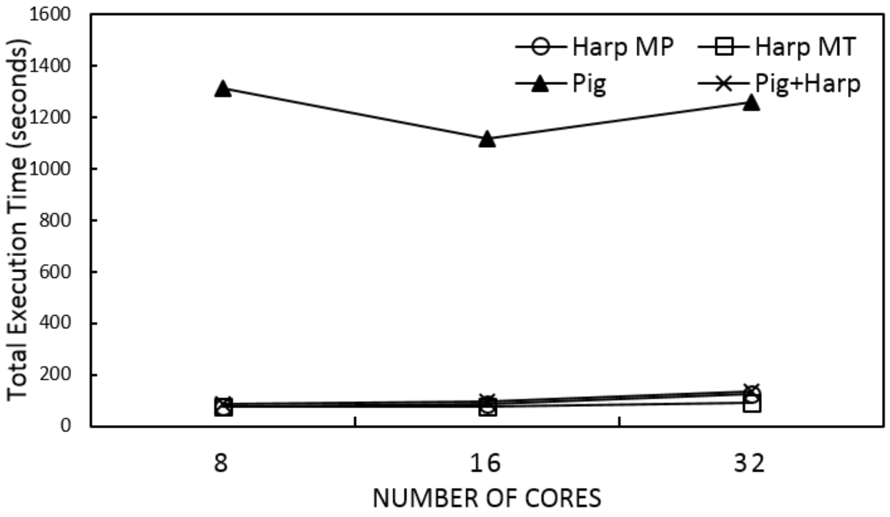
```
1 pagerank = LOAD '$InputDir' using
             HarpPageRank('$totalUrls',
             '$numMaps', '$itrs', '$jobID')
             as (result);
2 STORE pagerank INTO '$output';
```

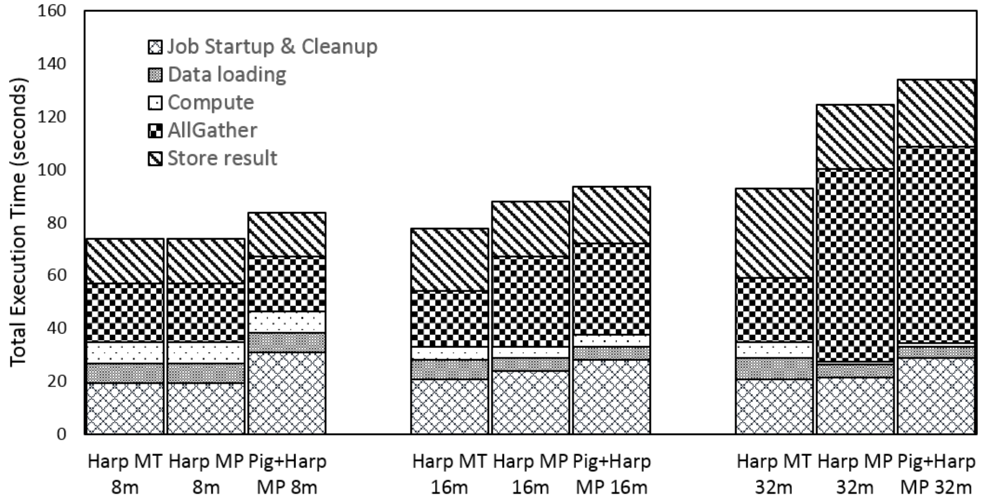
Pig+Harp PageRank

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

PageRank

- Pig+Harp is 5 times faster than native Pig
  - Tuple-based computation
  - Data type conversion time between bags and fields
- Harp's multi-thread shows the advantage in AllGather communication for larger partitions.
  - 2 layer synchronization
  - In-node sync and cross-node sync





SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Lines of code for K-means and PageRank

- Same lines of code for core algorithm
- Zero lines of code for wrapper in Pig+Harp approach

	Hadoop K-means	Pig K-means	Harp K-means	Pig+Harp K-means
K-means	36	36	39	39
Load & Format	261	250	499	662
Reduce / Comm.	142	56	34	34
Pig	0	10	0	3
Driver / Wrapper	341	40	176	0
Total lines	780	393	748	738

	Pig PageRank	Harp PageRank	Pig+Harp PageRank
PageRank	1	56	56
Load & Format	50	386	494
Reduce / Comm.	0	4	4
Pig	4	0	3
Driver / Wrapper	70	90	0
Total lines	125	536	557

Conclusion

- A trend of using Apache high level languages for data analytics
- Leverage Apache open source building blocks to maximize the usage of existing features such as expressiveness of data type and data structure, automatic parallelization for applications, and algorithms.
- Easy-to-use Hadoop and Pig plugin written in Java.
- Pig+Harp saves the jobs restart overheads; by utilizing Harp, it provides in-memory objects caching and fast communication for data shuffling.
- Pig+Harp suggests minimizing tuple-based computation by batch computation and replacing data aggregation by writing customized collective communication in UDF.

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Future Work

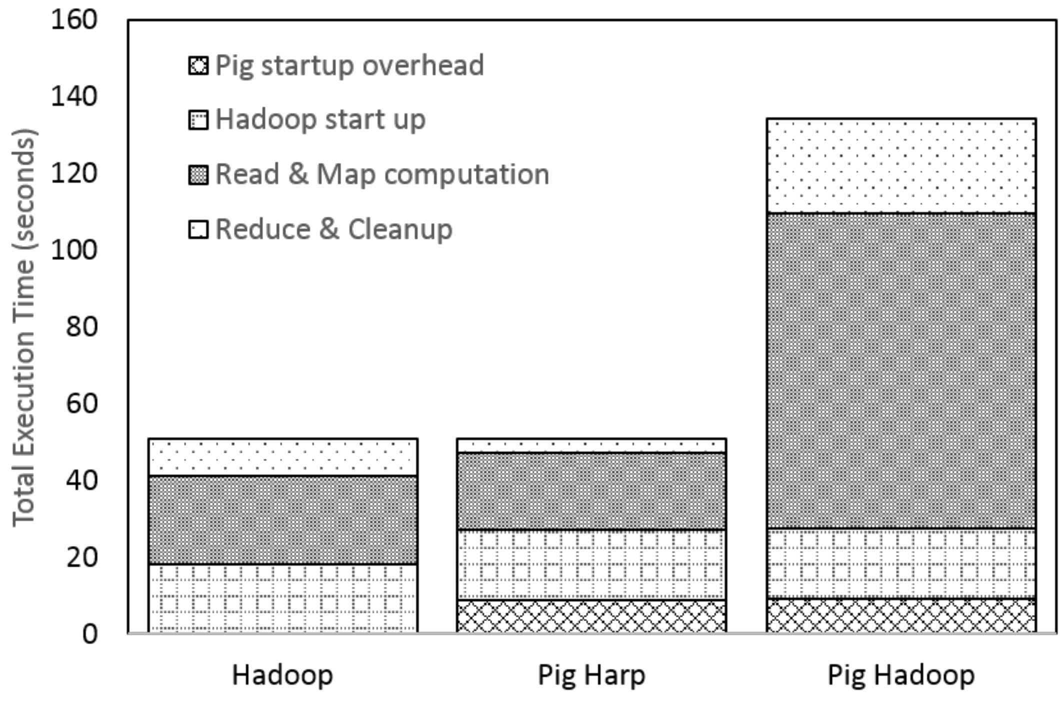
- Link scientific data pipelines as an end-to-end solution in the context of using high-level languages to solve parallel computing problems.
- Investigate Apache Tez, compare to our approach, and optimize in-memory data caching between tasks.
- Benchmark applications at a larger scale.

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Q&A

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

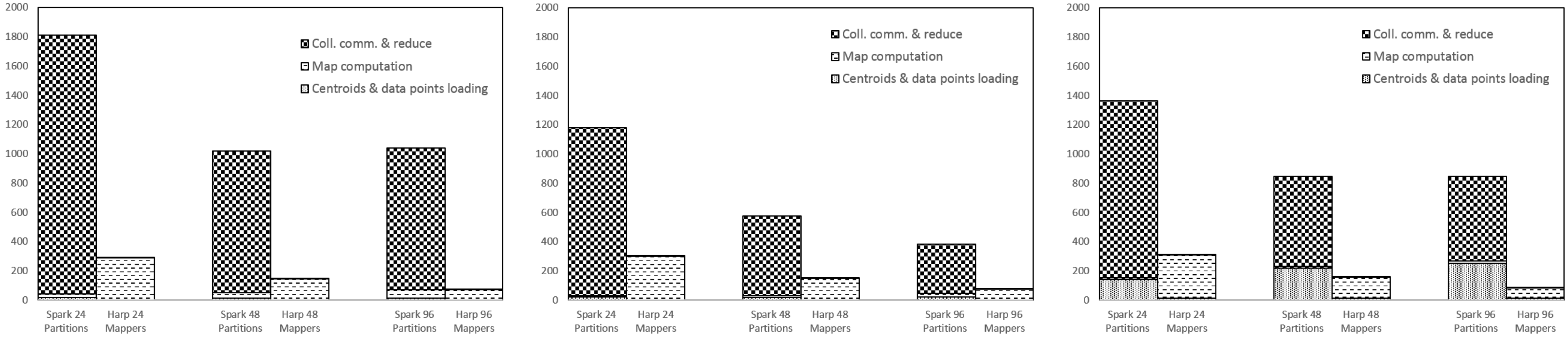
Wordcount without tuple-based computation



SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Harp 0.1.0 vs Spark 1.0.2

- Run Same K-Means clustering data with default Spark Mlib K-Means clustering
- Harp's data communication is highly optimized.
- Spark's computation and collectAsMap has less impact on the overall performance.
- Spark's reduceByKey operation takes much longer than usual with large data points as RDDs.
  - *Large intermediate data are shuffled to disk.



1m data points 50k centroids

10m data points 5k centroids

100m data points 500 centroids

*http://spark-summit.org/2014/training

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY Bloomington