An Implementation of Sawzall on Hadoop

Hidemoto Nakada[1], Tatsuhiko Inoue[2,1], Hirotaka Ogawa[1], Kudoh Tomohiro[1]

1. National Institute of Advanced Institute of Science and Technology

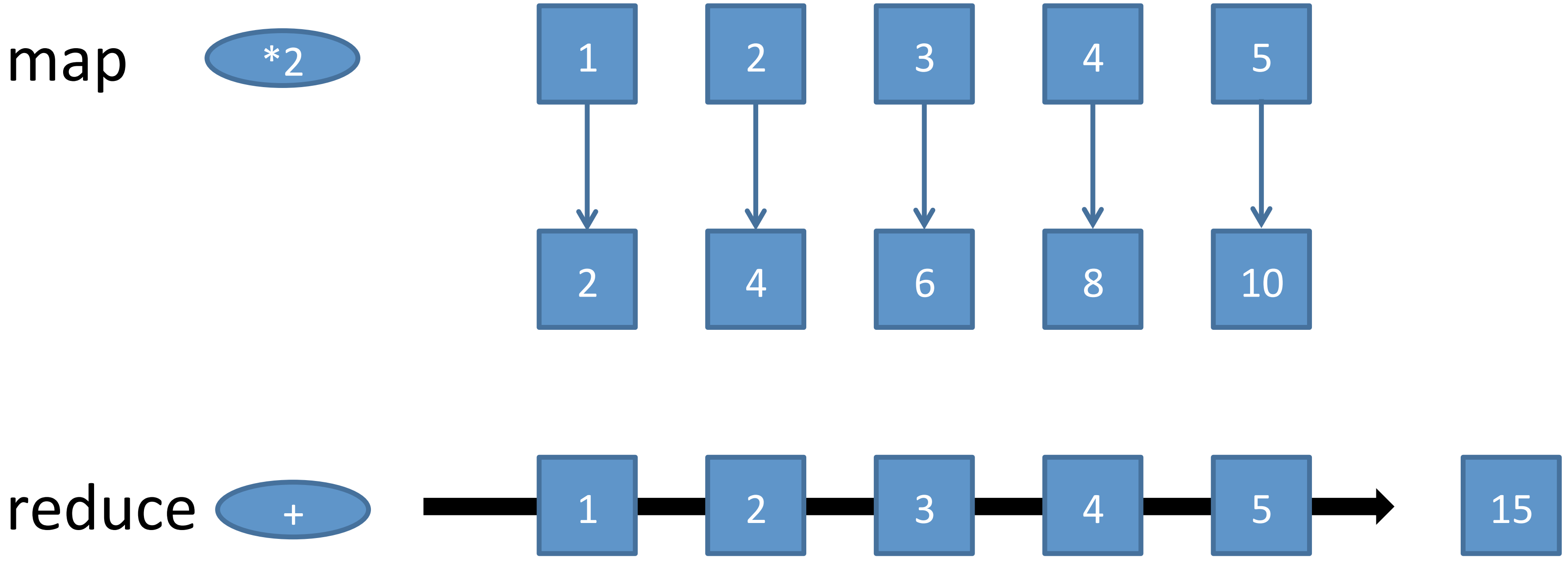
2. Soum, Corporation.

Background

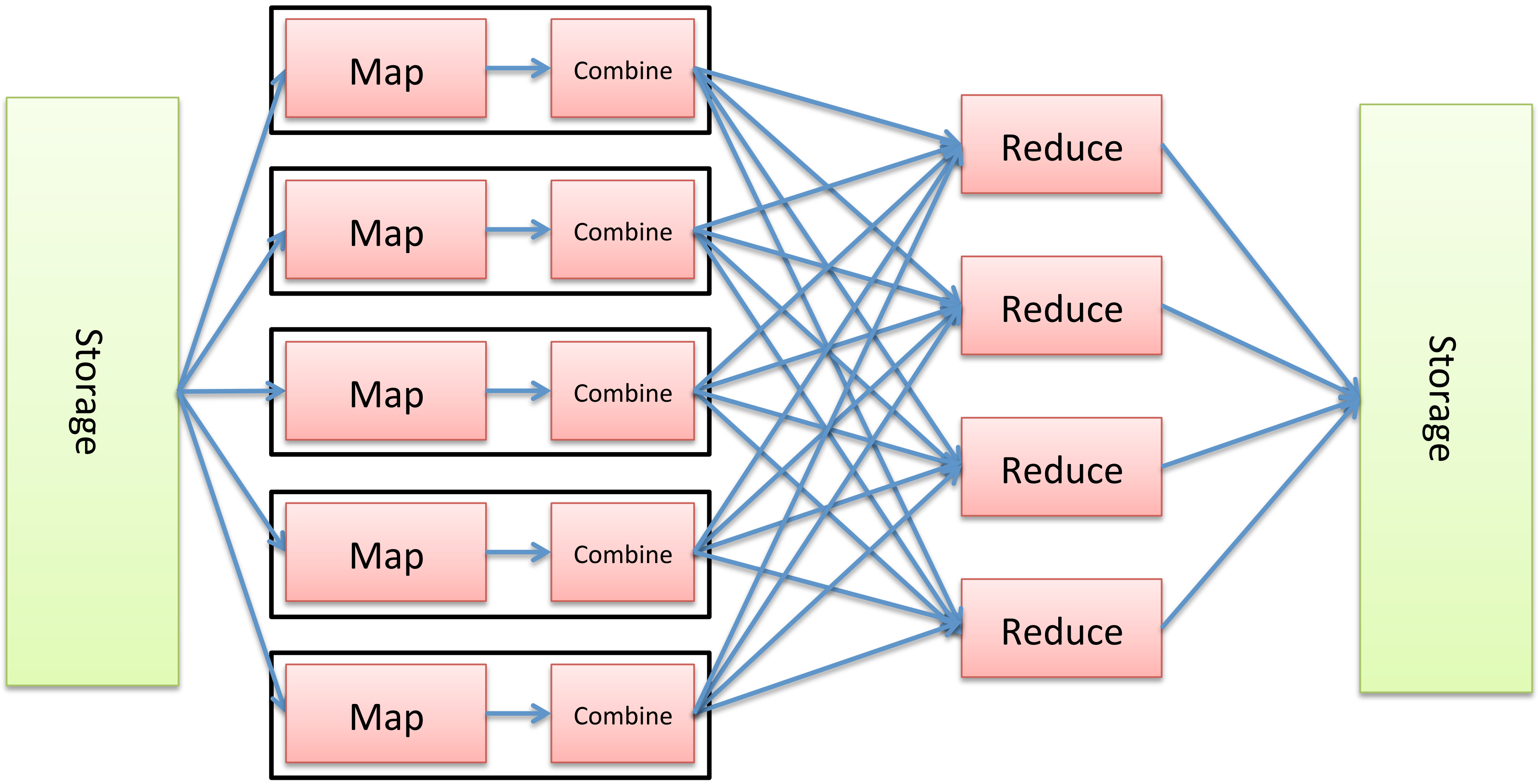
- Broad acceptance of MapReduce
  - As a new programming paradigm
- Native MapReduce program is not that easy.
  - In Hadoop Mapper, Reducer, and driving program is required.
  - Not suitable for prototyping or incremental data mining by end users
- High level languages for MapReduce are proposed
  - HiveQL, PigLatin, Jaql, Sawzall
  - Easy MapReduce programming for end users.

What is MapReduce?

- (relatively) new paradigm for parallel programming
- Inspired by higher order functions of functional languages



How MapReduce Works



Background

- Broad acceptance of MapReduce
  - As a new programming paradigm
- Native MapReduce program is not that easy.
  - In Hadoop Mapper, Reducer, and driving program is required.
  - Not suitable for prototyping or incremental data mining by end users
- High level languages for MapReduce are proposed
  - HiveQL, PigLatin, Jaql, Sawzall
  - Easy MapReduce programming for end users.

Hadoop Program Example

```
public class WordCount {
 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(LongWritable key, Text value, Context context)
   throws IOException, InterruptedException {
      String line = value.toString();
      StringTokenizer tokenizer = new M;
      while (tokenizer.hasMoreTokens()) {
         word.set(tokenizer.nextToken());
         context.write(word, one);
      }
   }
 }
```

Mapper

```
 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
   public void reduce(Text key, Iterator<IntWritable> values, Context context)
     throws IOException, InterruptedException {
      int sum = 0;
      while (values.hasNext())
         sum += values.next().get();
      context.write(key, new IntWritable(sum));
   }
 }
```

Reducer

```
 public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   Job job = new Job(conf, "wordcount");

   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   job.setMapperClass(Map.class);
   job.setReducerClass(Reduce.class);
   job.setInputFormatClass(TextInputFormat.class);
   job.setOutputFormatClass(TextOutputFormat.class);
   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   job.waitForCompletion(true);
 }
}
```

Driver

Hadoop Java API

Background

- Broad acceptance of MapReduce
  - As a new programming paradigm
- Native MapReduce program is not that easy.
  - In Hadoop Mapper, Reducer, and driving program is required.
  - Not suitable for prototyping or incremental data mining by end users
- High level languages for MapReduce are proposed
  - HiveQL, PigLatin, Jaql, Sawzall
  - Easy MapReduce programming for end users.

Contribution

- Provide open source Sawzall implementation
  - Compiler targeting Java code written in Scala
  - Generates MapReduce codes
    - Hadoop / SSS (AIST MapReduce)
- Evaluation
  - Compilation speed
  - Comparison with Szl – Google's open source implementation of Sawzall for sequential execution.
  - Comparison with Native Hadoop code

Outline

- Overview of Sawzall
- Design and Implementation of SawzallClone
- Evaluation
  - Comparison with Szl
  - Comparison with native Hadoop
- Conclusion and Future work

Sawzall

- A language designed by Google for MapReduce [‘05 Pike, et al.]
- Programmers take care of only the Mappers.
- Reducers are provided by the language runtime.
  - Map function just see one input. Operation on each input is completely independent from others.
    - c.f. Awk
  - Reducers are abstracted out as ‘tables’
    - Map functions just ’emit’ data into tables
    - Tables are provided by language

Sawzall Example

Log analysis

Proto statement

Table declaration

```
proto "p4stat.proto”
submitsthroughweek: table sum[minute: int] of count: int;

log: P4ChangelistStats = input;

t:      time = log.time;
minute: int  = minuteof(t)+
               60*(hourof(t) + 24*(dayofweek(t)-1))

emit submitsthroughweek[minute] <- 1;
```

Cast ‘input’

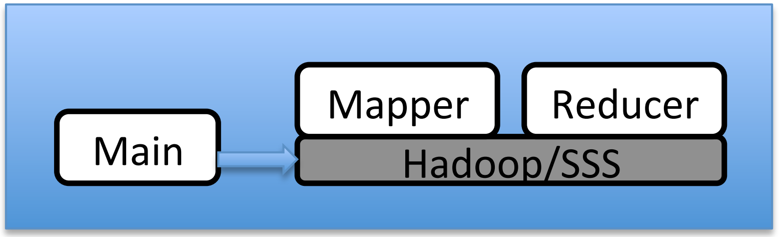
‘emit’ into the table

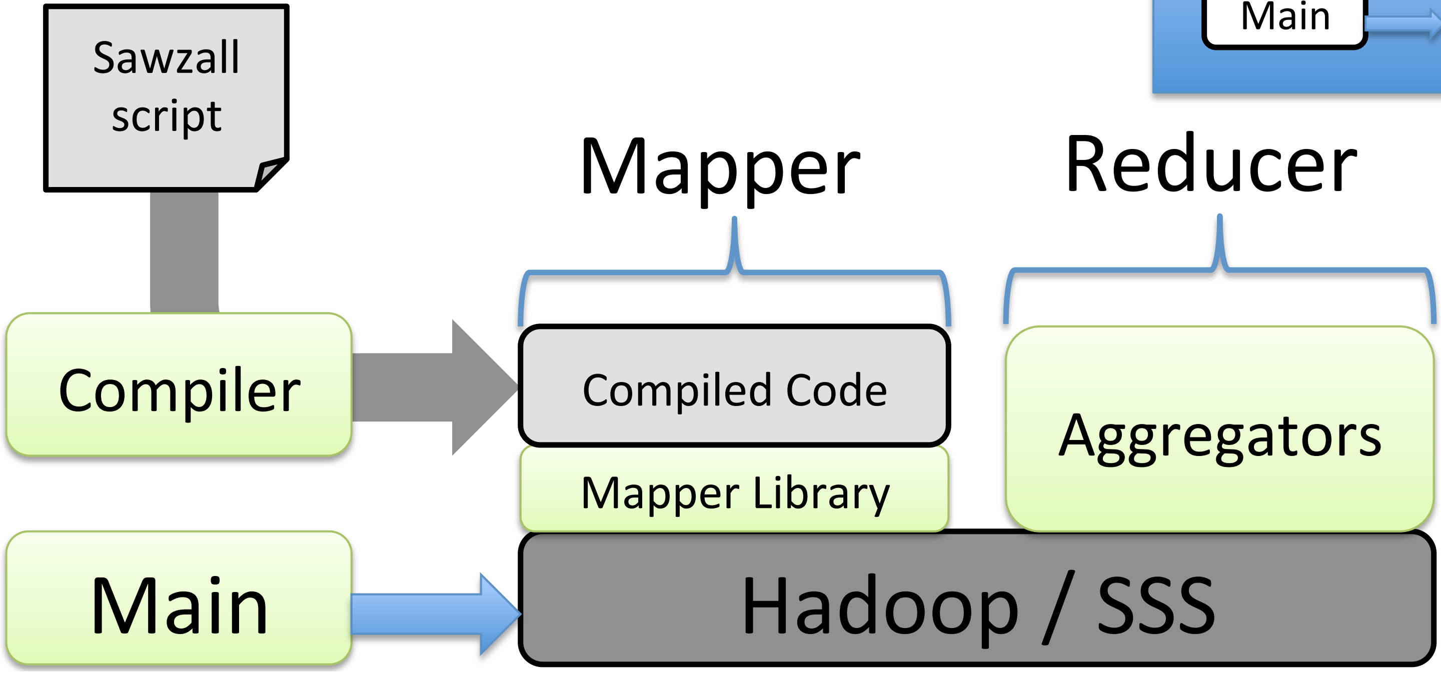
Sawzall tables

- Collection
- Maximum
- Sample
- Sum
- Top
- Quantile
- Unique

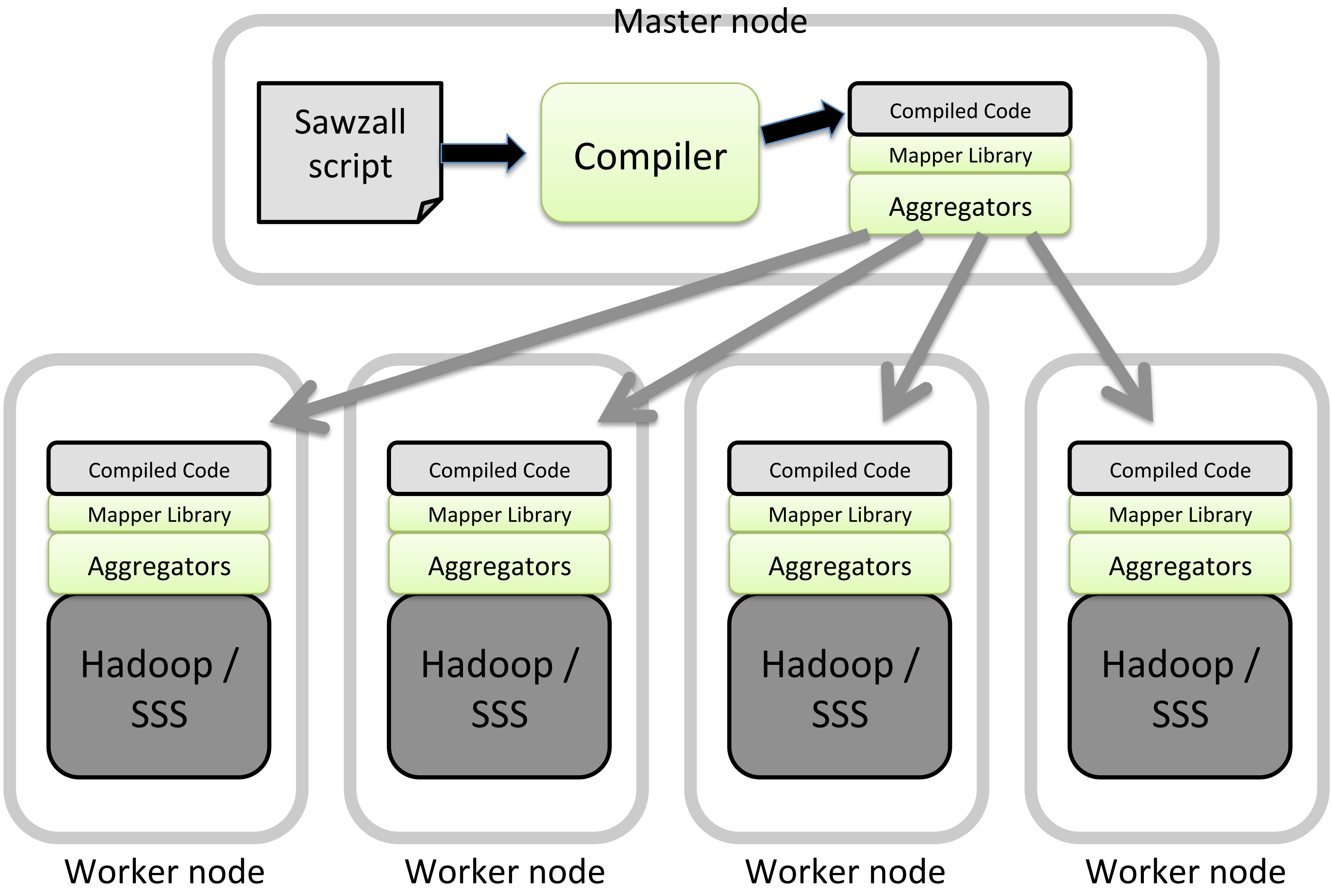
Overview of SawzallClone

- Sawzall script is just for Mappers
  Reducer is provided as the aggregator libraries
- Implemented as a compiler in Scala, targeting Java code
- 4 modules, compiler, main driver program, mapper libraries, and aggregator libraries.



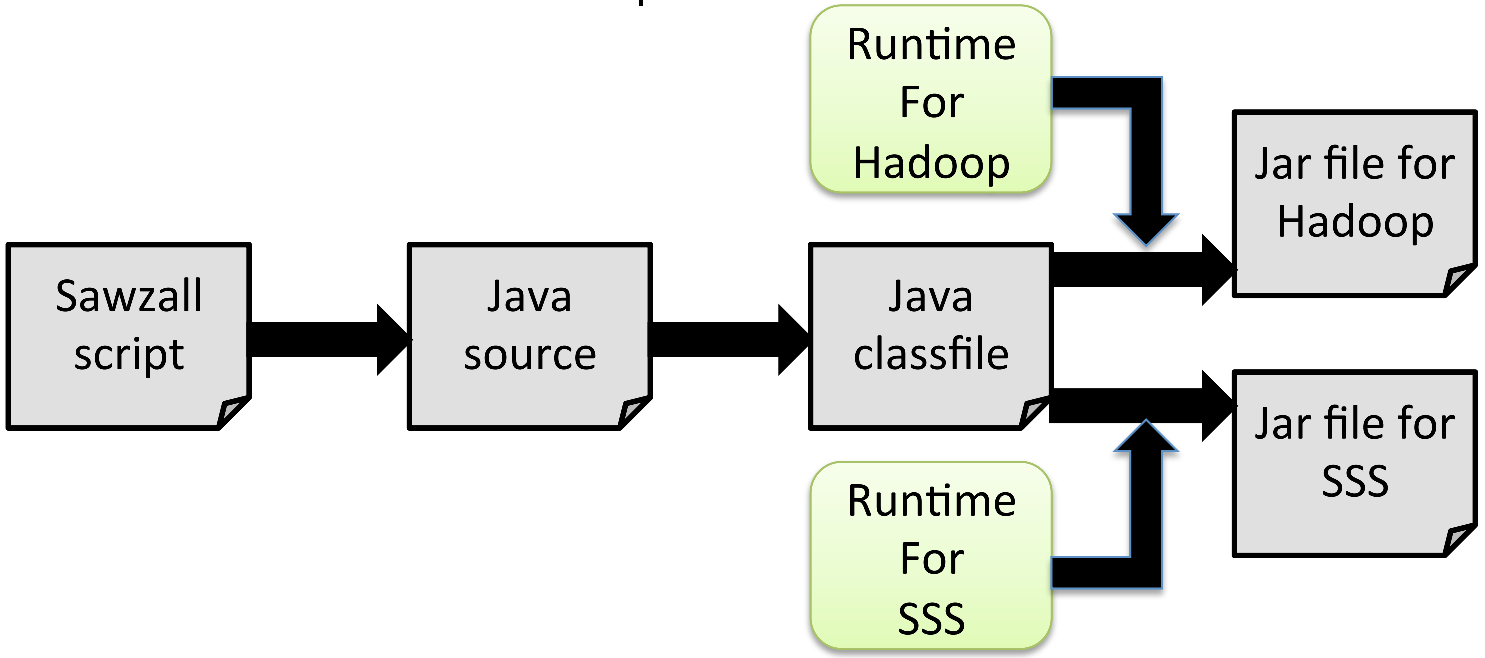


SawzallClone execution



Compilation of Sawzall

- Output Java code, not byte code
  - to ease implementation
- Output code is system neutral thanks to runtime libraries
  - Runtime libraries hide the underlying system from the output code
  - Same code works for Hadoop and SSS



Sample Compiled Code

```
public class Mapper implements SCHelpers.Mapper {
...
  @Override
  public void map(SCHelpers.Emitter emitter,
                Helpers.ByteStringWrapper global_0_input)
  throws java.lang.Throwable {
    String local_0_document = BuildIn.func_string(global_0_input);
    List<String> local_1_words =
                          BuildIn.func_split(local_0_document);
    {
      Long local_2_i = 0l;
      for (; (((((local_2_i) <
                 (BuildIn.func_len(local_1_words)))?1l:0l)) != 0l);
                 (local_2_i) = (((local_2_i) + (1l)))) {
        emitter.emit(statics.static_0_t,
                     BuildIn.func_bytes(
                      (local_1_words).get((local_2_i).intValue())),
                     BuildIn.func_bytes(1l));
      }
    }
  }
}
```

Evaluation

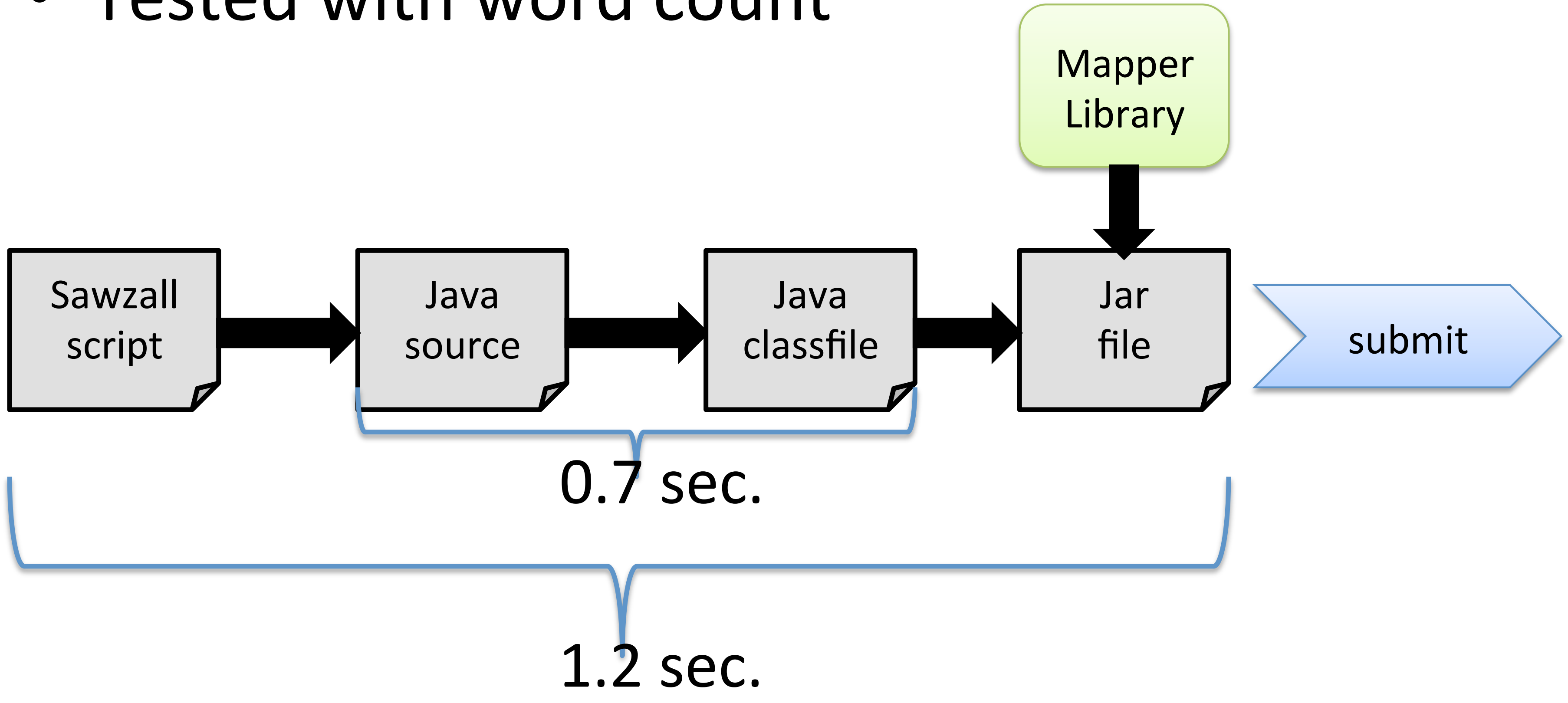
- Compilation time
- Comparison with Szl
  - Szl : Google’s open source implementation of Sawzall language
    - Written in C++: both of the compiler and runtime
    - No parallel implementation so far
  - Comparison in Sequential execution
- Comparison with Hadoop Native
  - In Parallel execution

Evaluation Environment

- Small Cluster
  - Number of nodes: 16 + 1 (master)
  - CPUs per node: Intel Xeon W5590 3.33GHz x 2
  - Memory per node: 48GB
  - OS: CentOS 5.5 x86_64
  - Storage: Fusion-io ioDrive Duo 320GB
- Hadoop
  - Hadoop 0.20.2
    - HDFS replication = 3
    - Mapper /node = 7

Compilation time

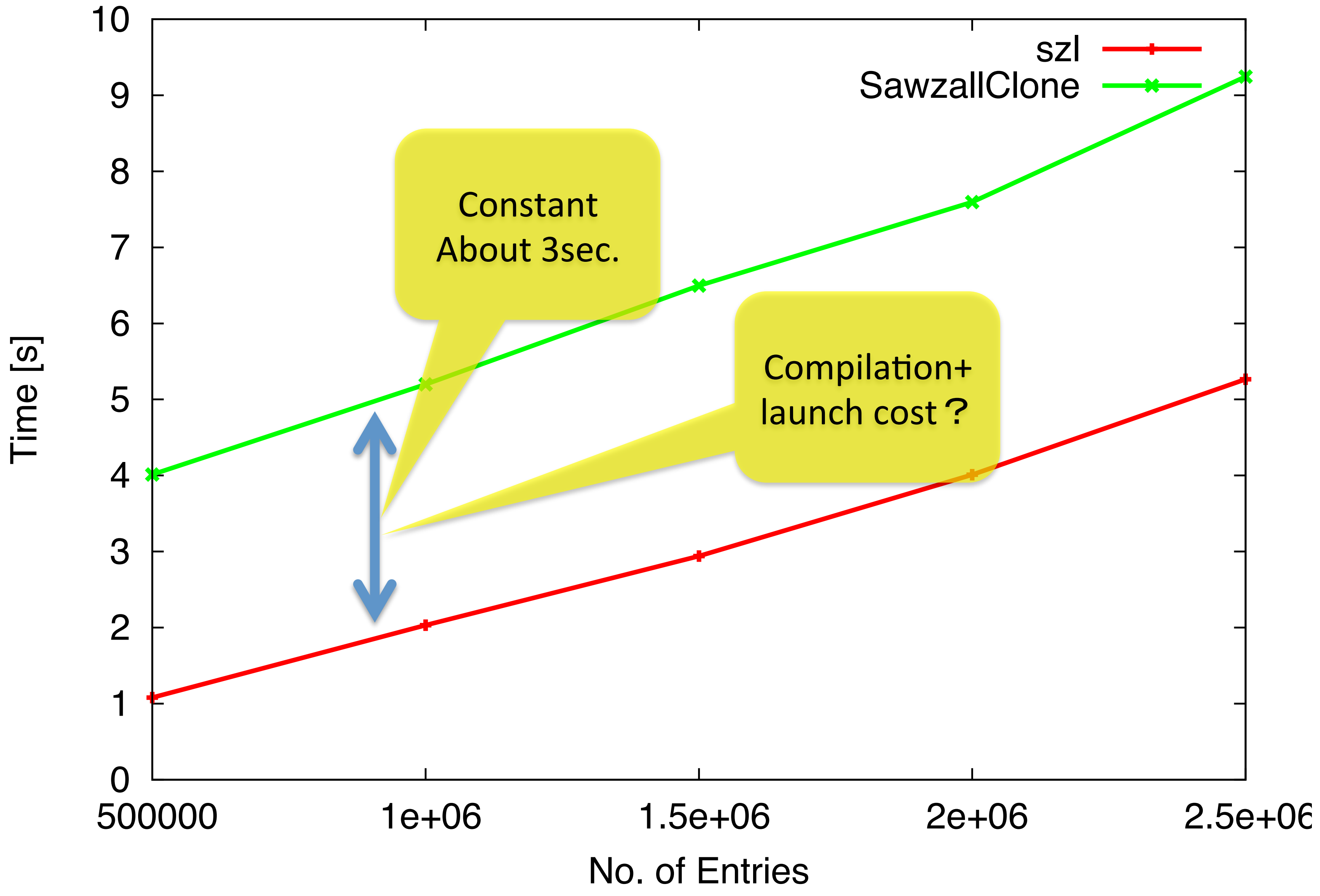
- SawzallClone compiles script on the fly
  - 2 compilation and creation of a Jar file
- Tested with word count



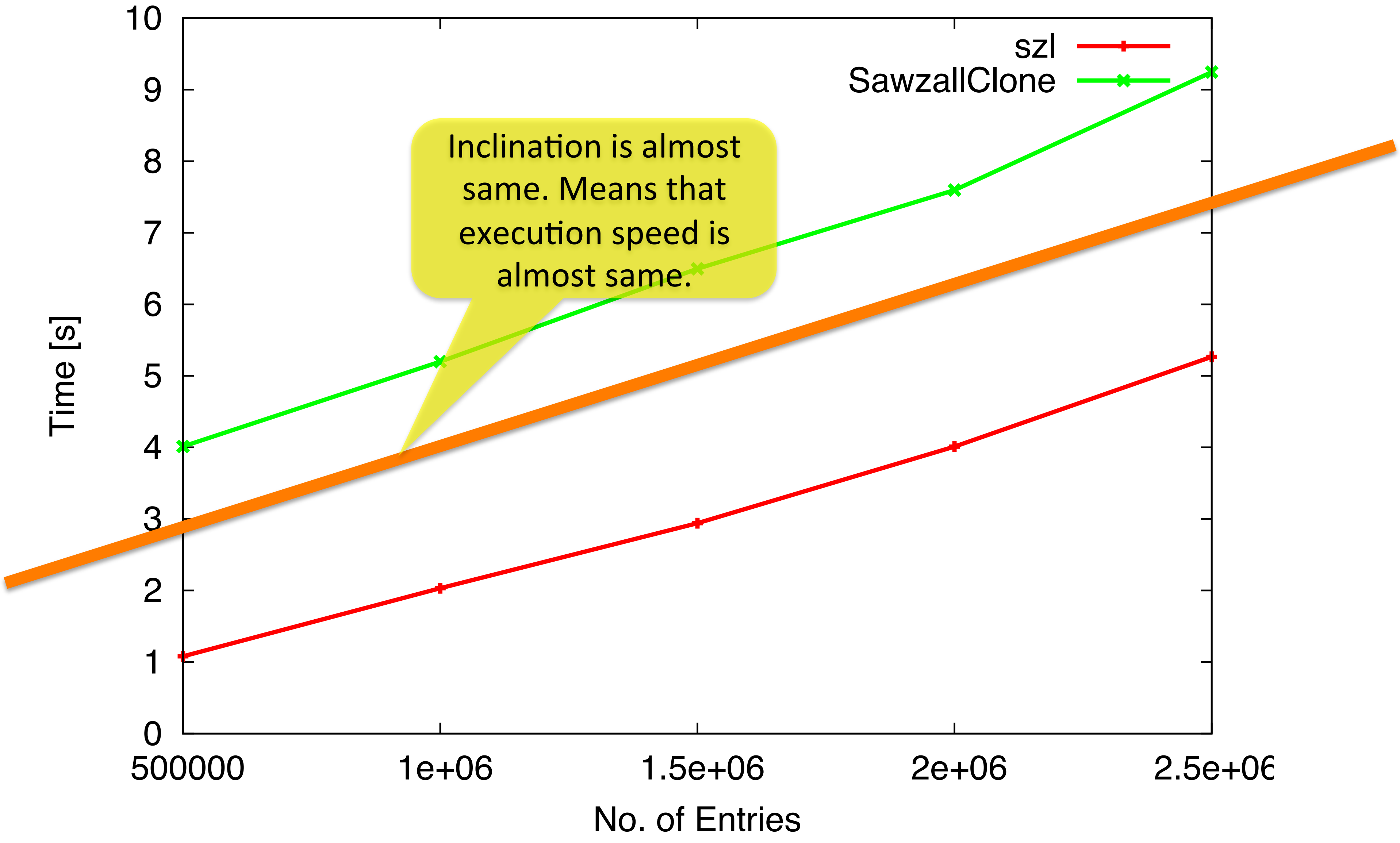
Comparison with Szl

- Target Program: Log analysis
- Sequential execution
  - SawzallClone in a single VM
- For log items from 0.5 million to 2.5 million

Comparison with Szl



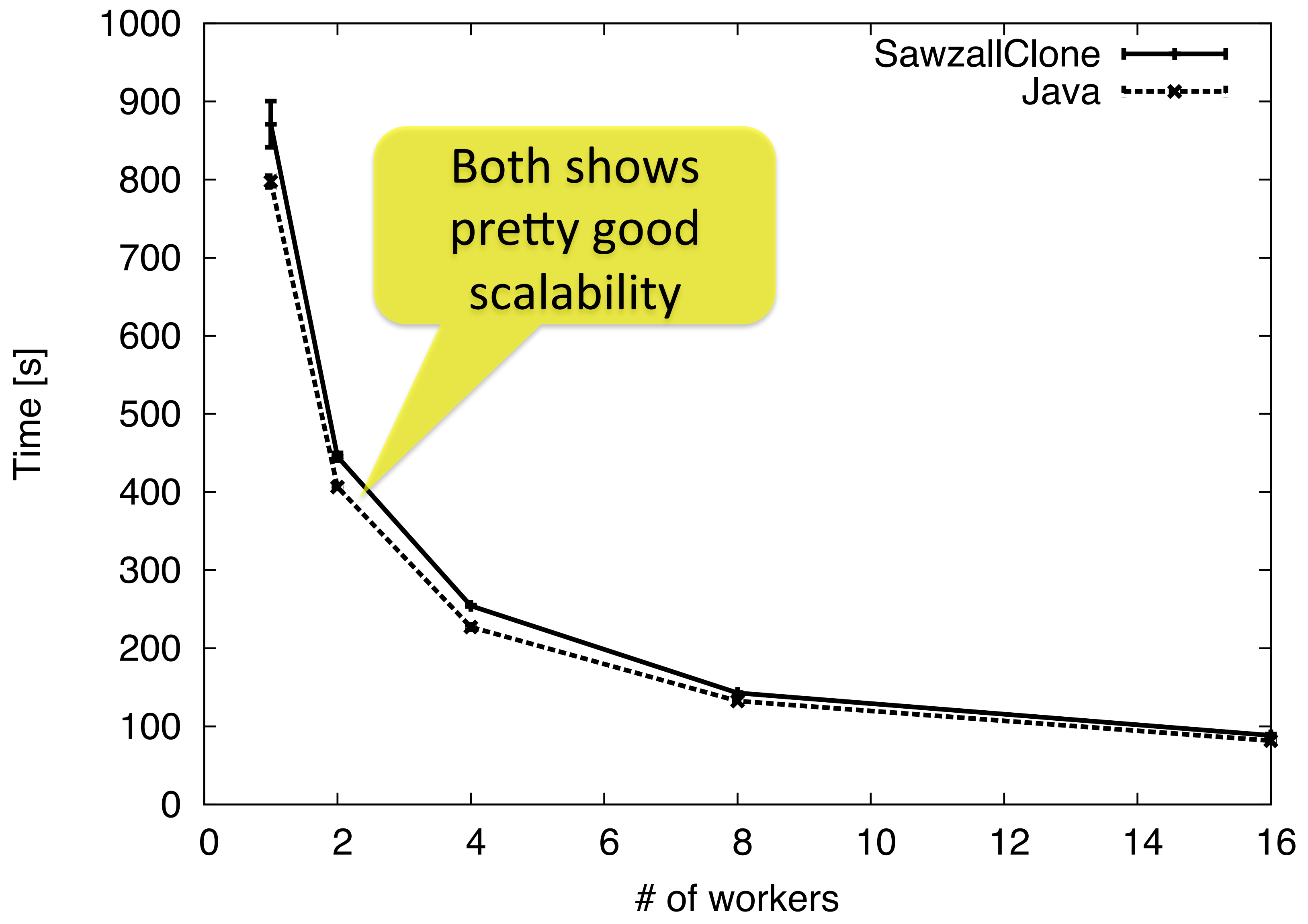
Comparison with Szl



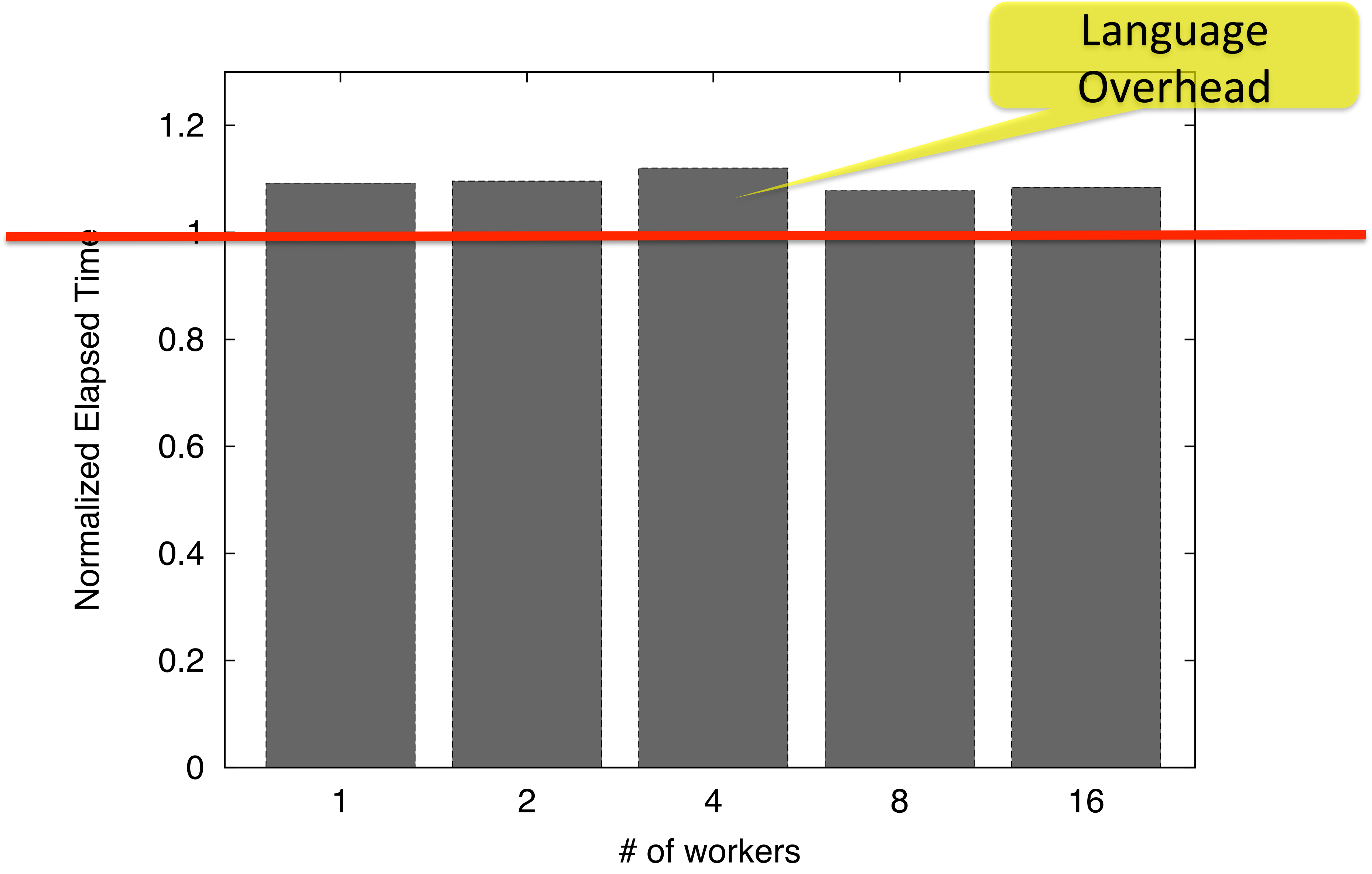
Comparison with Hadoop

- Log analysis
  - Data size is fixed
    - 0.1 billion records
    - Divided into 64 files, 40MB each, 2.5GB in total.
    - Replicated 3 times on HDFS
  - # of nodes : 1,2,4,8,16

Comparison with Hadoop



Normalized with Native Hadoop



Conclusion

- Sawzall implementation for Hadoop / SSS
  - Compiler targeting Java

- Evaluation
  - Compilation – Constant 1.2sec. overhead
  - Comparison with Szl
    - Constant overhead due to slow compilation
    - Execution speed is comparable
  - Comparison with Hadoop Native code
    - Confirmed that it scales well
    - Overhead is not negligible
    - Further optimization required

Acknowledgement

- This work was partly funded by the New Energy and Industrial Technology Development Organization (NEDO) Green-IT project.

Thank you

Available at

http://code.google.com/p/sawzall-clone