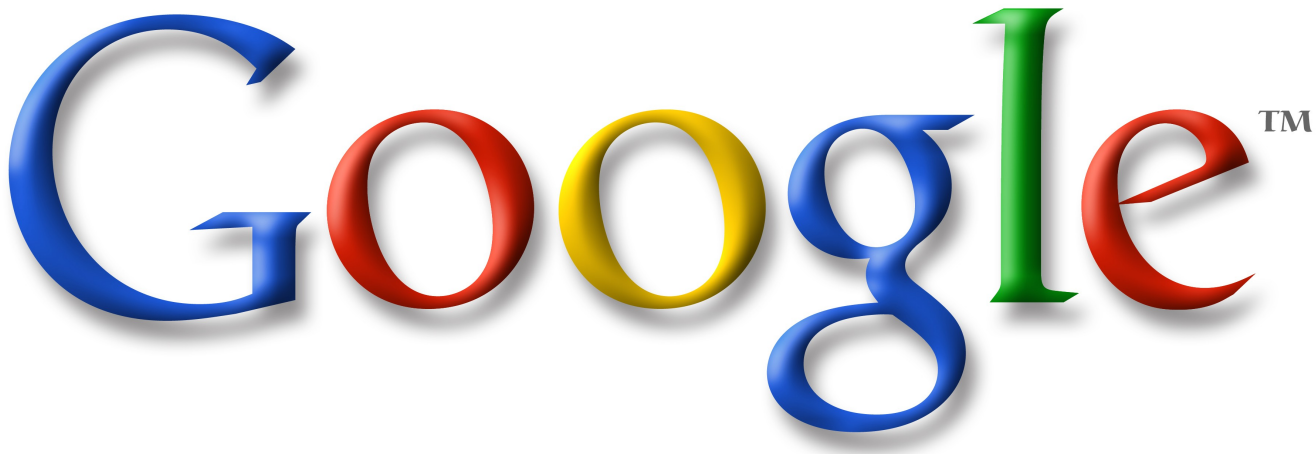
MapReduce a distribuované výpočty



5. listopadu 2009

Prostředí

Máme k dispozici následující počítače:

1. dvoujaderné x86 procesory,
2. běžné IDE disky s distribuovaným souborovým systémem,
3. běžný síťový hardware – 100Mb/s nebo 1Gb/s.

Cluster obsahuje tisíce takových počítačů.

Programovací model

Map

Z jedné dvojice `(klíč, hodnota)` vytvoří seznam dvojic `(klíč, dočasná hodnota)`.

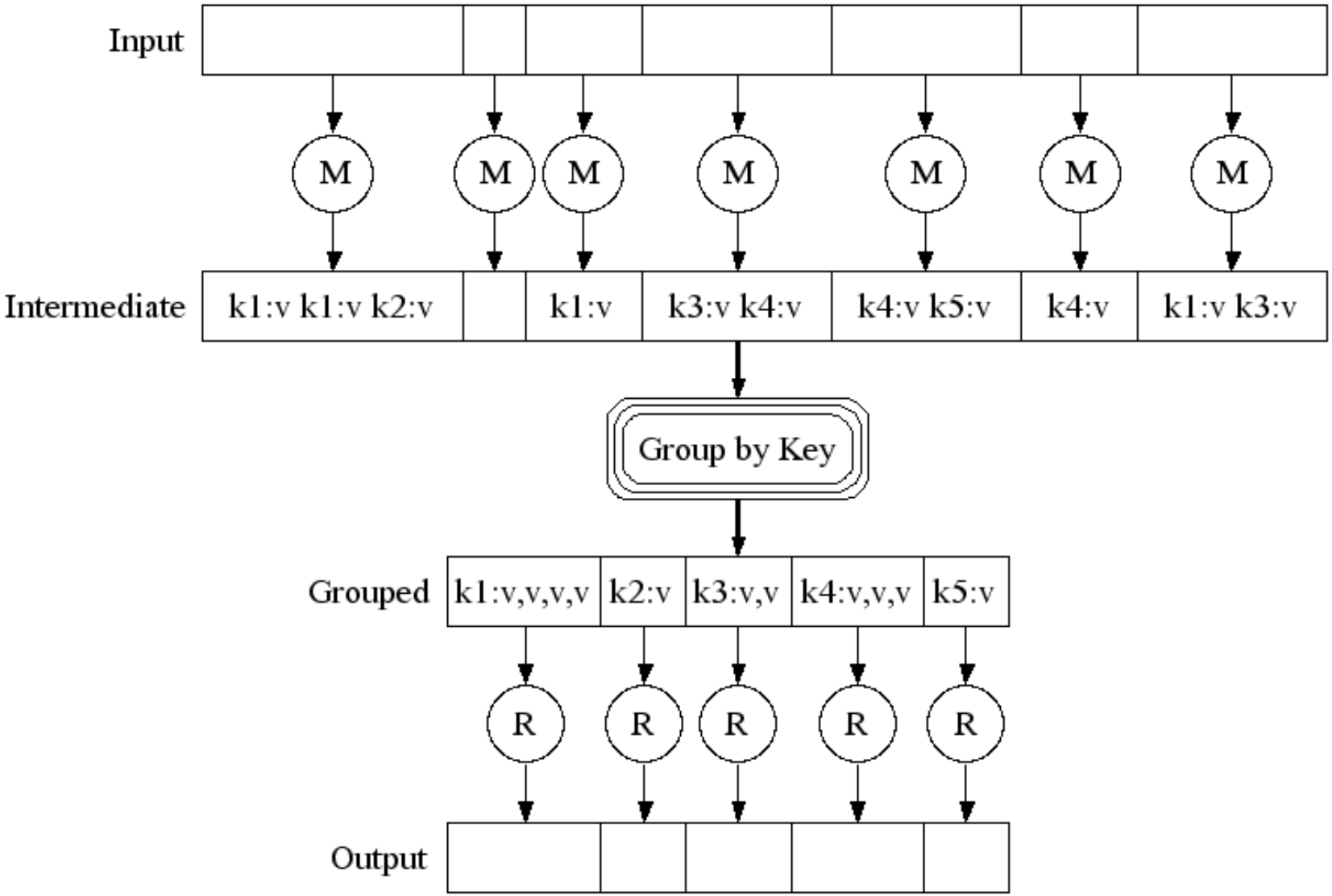
```
map (in_key, in_value) -> list(out_key, intermediate_value)
```

Reduce

Z klíče a seznamu dočasných hodnot vytvoří výsledný seznam hodnot tohoto klíče.

```
reduce (out_key, list(intermediate_value)) -> list(out_value)
```

Programovací model



Příklad: počítání výskytů slov

Map

```
map (String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");
```

Reduce

```
reduce (String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Další příklady

Počet přístupů na dané URL

Stejně jako počítání výskytů slov.

Výběr vyhovujících položek

Funkce `map` vrátí pouze vyhovující položky, funkce `reduce` nemění seznam hodnot.

Matice webu

```
map (String url, String content):
  for each referenced h in content:
    EmitIntermediate(href, url);

reduce (String url, Iterator sources):
  Emit(sources);
```

Další příklady

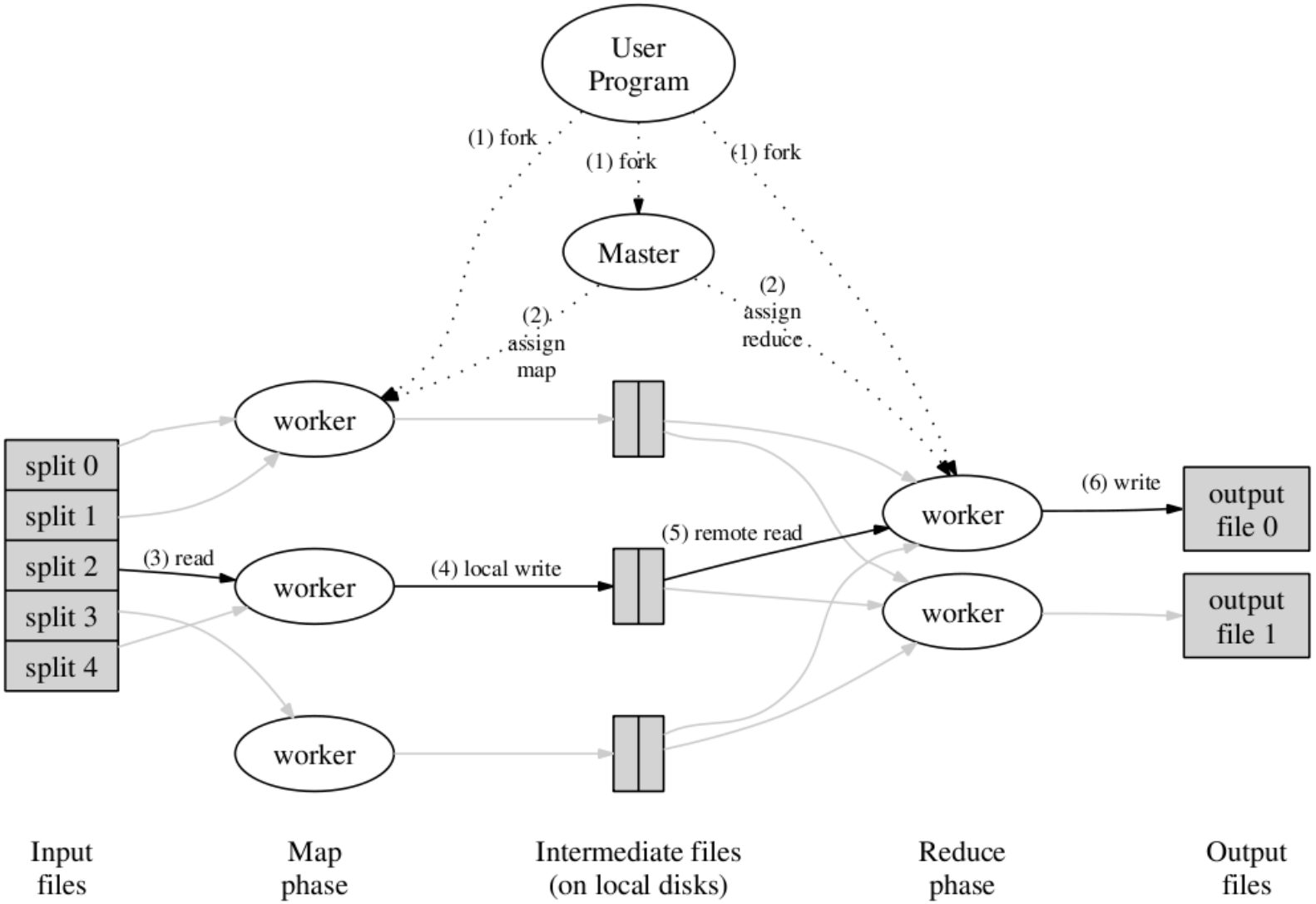
Invertovaný index

```
map (String docid, String content):
  for each word w in content:
    EmitIntermediate(word, docid);

reduce (String word, Iterator docids):
  Emit(sort(docids));
```

Třídění dat

```
map (String _, String value):
  EmitIntermediate(extract_key(value), value);

reduce (String key, Iterator values):
  Emit(values);
```

Přehled provádění



Průběh provádění

Průběh provádění:

1. Master získá `map`, `reduce`, data a počítače.
2. Vstup je rozdělen na M bloků.
3. `Map` na jednotlivé bloky dat.
4. Dočasné výsledky se rozdělí do R skupin.
5. `Reduce` na jednotlivé skupiny.

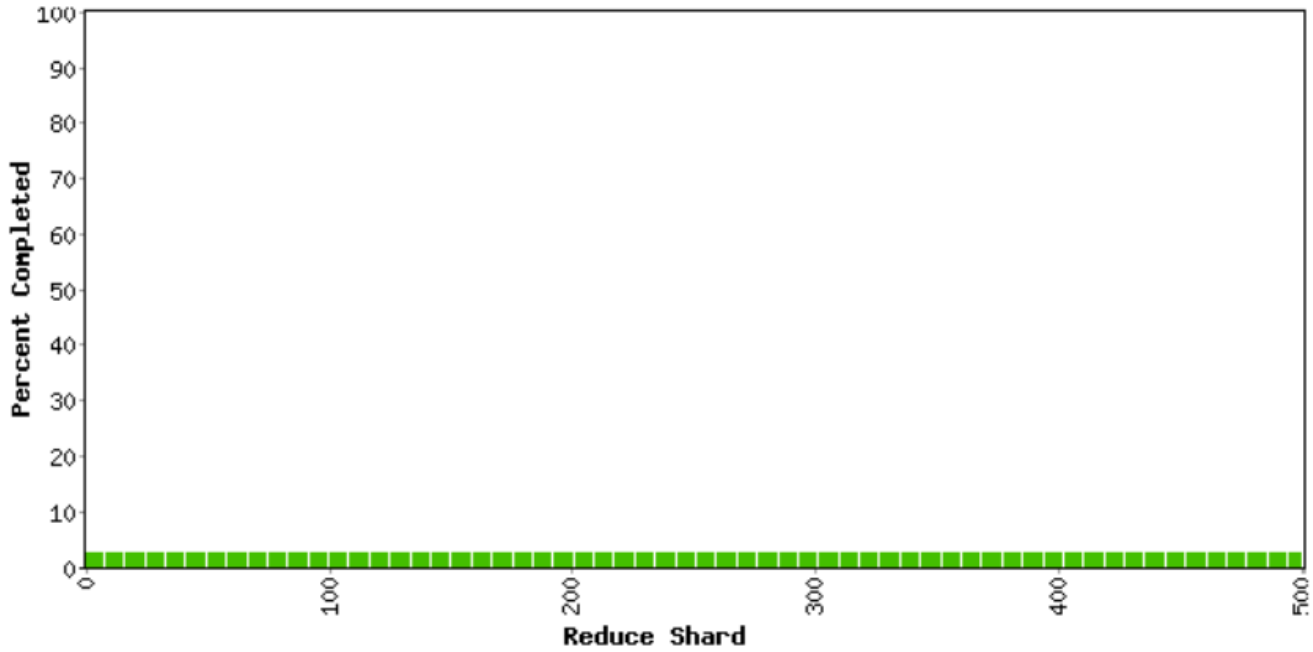
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0



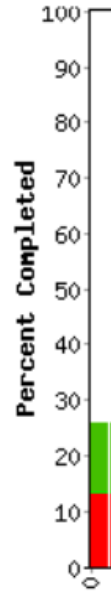
Counters

Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-	506631

Started:

1707 w

Type
Map
Shuffle
Reduce



Detaily implementace

Detaily implementace:

1. Master a jeho datové struktury.
2. Lokalita zpracovávaných dat.
3. Volba M, R a vyrovnávání zátěže.
4. Výpadky použitých počítačů.
5. Synchronizace, zamykání a atomicita.
6. Záložní úlohy před koncem.

Rozšíření modelu

Model výpočtu může být rozšířen mnoha způsoby:

1. Vlastní rozdělovací funkce před `reduce` fází.
2. Zaručení pořadí dat při `reduce`.
3. Sekundární klíče.
4. Funkce `combine`.
5. Přeskakování problematických částí vstupu.
6. Vlastní čítače.
7. Lokální spouštění.
8. Stringové a strukturované rozhraní.

Zdrojový kód

Zveřejněný zdrojový kód:

```
// User's map function
class WordCounter : public Mapper {
 public:
  virtual void Map(const MapInput& input) {
    const string& text = input.value();
    const int n = text.size();
    for (int i = 0; i < n; ) {
      // Skip past leading whitespace
      while ((i < n) && isspace(text[i]))
        i++;
      // Find word end
      int start = i;
      while ((i < n) && !isspace(text[i]))
        i++;
       if (start < i)
         Emit(text.substr(start,i-start),"1");
     }
   }
};
REGISTER_MAPPER(WordCounter);
```

```
// User's reduce function
class Adder : public Reducer {
 public:
   virtual void Reduce(ReduceInput* input) {
     // Iterate over all entries with the
     // same key and add the values
     int64 value = 0;
     while (!input->done()) {
       value += StringToInt(input->value());
       input->NextValue();
     }
     // Emit sum for input->key()
     Emit(IntToString(value));
   }
};
REGISTER_REDUCER(Adder);
```

Zdrojový kód

Zveřejněný zdrojový kód:

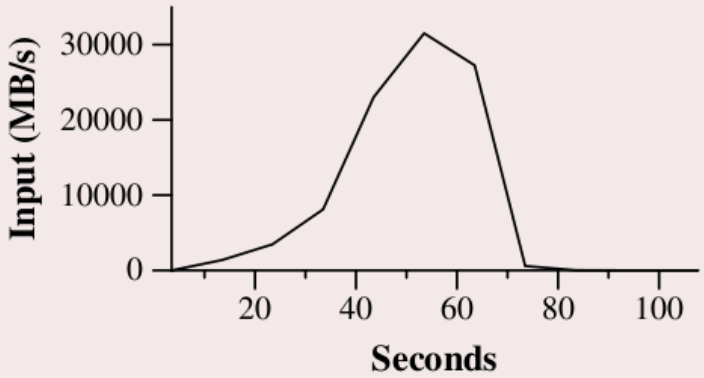
```
int main(int argc, char** argv) {
   ParseCommandLineFlags(argc, argv);
   MapReduceSpecification spec;
   // Store list of input files into "spec"
   for (int i = 1; i < argc; i++) {
     MapReduceInput* input = spec.add_input();
     input->set_format("text");
     input->set_filepattern(argv[i]);
     input->set_mapper_class("WordCounter");
   }

  // Specify the output files:
  //    /gfs/test/freq-00000-of-00100
  //    /gfs/test/freq-00001-of-00100
  //    ...
  MapReduceOutput* out = spec.output();
  out->set_filebase("/gfs/test/freq");
  out->set_num_tasks(100);
  out->set_format("text");
  out->set_reducer_class("Adder");
  // Optional: do partial sums within map
  // tasks to save network bandwidth
  out->set_combiner_class("Adder");
  // Tuning parameters: use at most 2000
  // machines and 100 MB of memory per task
  spec.set_machines(2000);
  spec.set_map_megabytes(100);
  spec.set_reduce_megabytes(100);
  // Now run it
  MapReduceResult result;
  if (!MapReduce(spec, &result)) abort();
  // Done: 'result' structure contains info
  // about counters, time taken, number of
  // machines used, etc.
  return 0;
}
```

Výkon MapReduce

- Cluster s 1800 počítačí, každý dva 2GHz Xeony s HT, 4GB paměti, dva 160GB IDE disky, gigabitový Ethernet.

Hledání záznamů

Celkem 10^{10} 100B záznamů, $M = 15000$ (64MB kusy), $R = 1$.

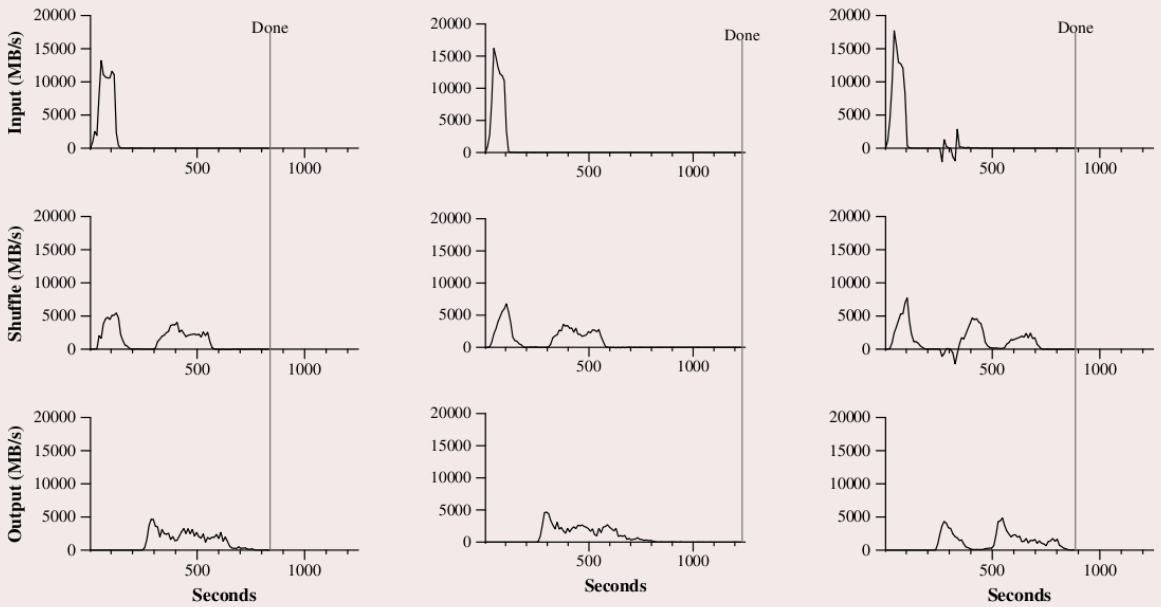


Výkon MapReduce

- Cluster s 1800 počítačí, každý dva 2GHz Xeony s HT, 4GB paměti, dva 160GB IDE disky, gigabitový Ethernet.

Třídění

Celkem 10^{10} 100B záznamů, $M = 15000$ (64MB kusy), $R = 1$.



(a) Normal execution (b) No backup tasks (c) 200 tasks killed

Používání MapReduce uvnitř firmy Google

MapReduce spuštěné ve firmě Google v srpnu 2004:

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days
Input data read	3,288 TB
Intermediate data produced	758 TB
Output data written	193 TB
Average worker machines per job	157
Average worker deaths per job	1.2
Average map tasks per job	3,351
Average reduce tasks per job	55
Unique *map* implementations	395
Unique *reduce* implementations	269
Unique *map/reduce* combinations	426

Výhody a nevýhody modelu MapReduce

Výhody MapReduce

1. jednoduchost
2. vysoká úroveň paralelismu
3. veliká škálovatelnost
4. nenáročnost na HW

Vhodné typy problémů pro MapReduce

1. vytváření indexů
2. strojové učení na velkých datech, například automatický překlad
3. extrahování dat
4. statistiky a analýza dat
5. clusterizace
6. manipulace s velkými grafy

Hadoop

Hadoop

1. Opensource implementace MapReduce Hadoop.
2. Opensource implementace GFS – HDFS.
3. Zastřešuje Apache, pod Apache licencí.
4. V jazyce Java. . .
5. . . . ale `map` a `reduce` lze díky projektu Hadoop Pipes psát v mnoha jazycích, především v `C++`.

Kdo používá Hadoop

Kdo používá Hadoop:

- Yahoo, $\sim$ 100 000 procesorů v $\sim$ 25 000 počítačích
- Facebook, $\sim$ 4 800 procesorů v $\sim$ 600 počítačích, 2PB dat
- Quantcast, $\sim$ 3 000 procesorů, 3.5PB dat
- IBM, dává univerzitám přístup k Hadoop clusterům
- ImageShack, Last.fm, The New York Times, Baidu, . . .

Haddoop API – Mapper a Reducer

Jmenný prostor `org.apache.hadoop.mapreduce`.

Context<KeyIn, ValueIn, KeyOut, ValueOut>

- `write(KeyOut key, ValueOut value)`
- `getCounter(string groupName, string counterName)`

Mapper<KeyIn, ValueIn, KeyOut, ValueOut>

- `setup(Context context)`
- `map(KeyIn key, ValueIn value, Context context`
- `cleanup(Context context)`

Reducer<KeyIn, ValueIn, KeyOut, ValueOut>

- `setup(Context context)`
- `reduce(KeyIn key, Iterable<ValueIn> value, Context context`
- `cleanup(Context context)`

Haddoop API – Serializovatelná data

Jmenný prostor `org.apache.hadoop.io`.

Writable

- `readFields(DataInput in)`
- `write(DataOutput out)`

Spoustu implementací:

- `ArrayWritable`
- `BooleanWritable`
- `ByteWriteble`
- `BytesWriteble`
- `DoubleWriteble`
- `FloatWriteble`
- `IntWriteble, VIntWritable`
- `LongWriteble, VLongWritable`
- `NullWriteble`
- `Text`

Zdrojový kód

```
public class WordCount {

  public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context)
                                                  throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer 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;
      for (IntWritable val : values)
        sum += val.get();

      result.set(sum);
      context.write(key, result);
    }
  }
```

Haddoop API – Nastavení výpočtu

org.apache.hadoop.mapreduce.Job

- `setJarByClass(Class<?> cls)`
- `setMapperClass(Class<? extends Mapper> cls)`
- `setReducerClass(Class<? extends Reducer> cls)`
- `setCombinerClass(Class<? extends Combiner> cls)`
- `setOutputKeyClass(Class<?> theClass)`
- `setOutputValueClass(Class<?> theClass)`
- `setInputFormatClass(Class<? extends InputFormat> cls)`
  - `TextInputFormat, SequenceFileInputFormat`
- `setOutputFormatClass(Class<? extends OutputFormat> cls)`
  - `TextOutputFormat, SequenceFileOutputFormat`

Zdrojový kód

```
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Haddoop API – Podrobné řízení výpočtu

Jmenný prostor `org.apache.hadoop.mapreduce`.

Job

- `setPartitionerClass(Class<? extends Partitioner> cls)`
- `setSortComparatorClass(Class<? extends RawComparator> cls)`
- `setGroupingComparatorClass(Class<? extends RawComparator> cls)`

Partitioner<Key, Value>

- `int getPartition(Key key, Value value, int numPartitions)`

RawComparator<T>

- `int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)`

Haddoop API – Užitečné drobnosti

org.apache.hadoop.mapreduce.Counter

- vytvářejí se pomocí `Context.getCounter()`
- `long getValue()`
- `void increment(long incr)`

org.apache.hadoop.filecache.DistributedCache

- distribuuje soubory, archivy a JARy na každý mapper a reducer
- při vytváření výpočtu:

```
Configuration conf = new Configuration();
DistributedCache.addCacheFile(new URI("test/words.dat#words.dat"), conf);
```

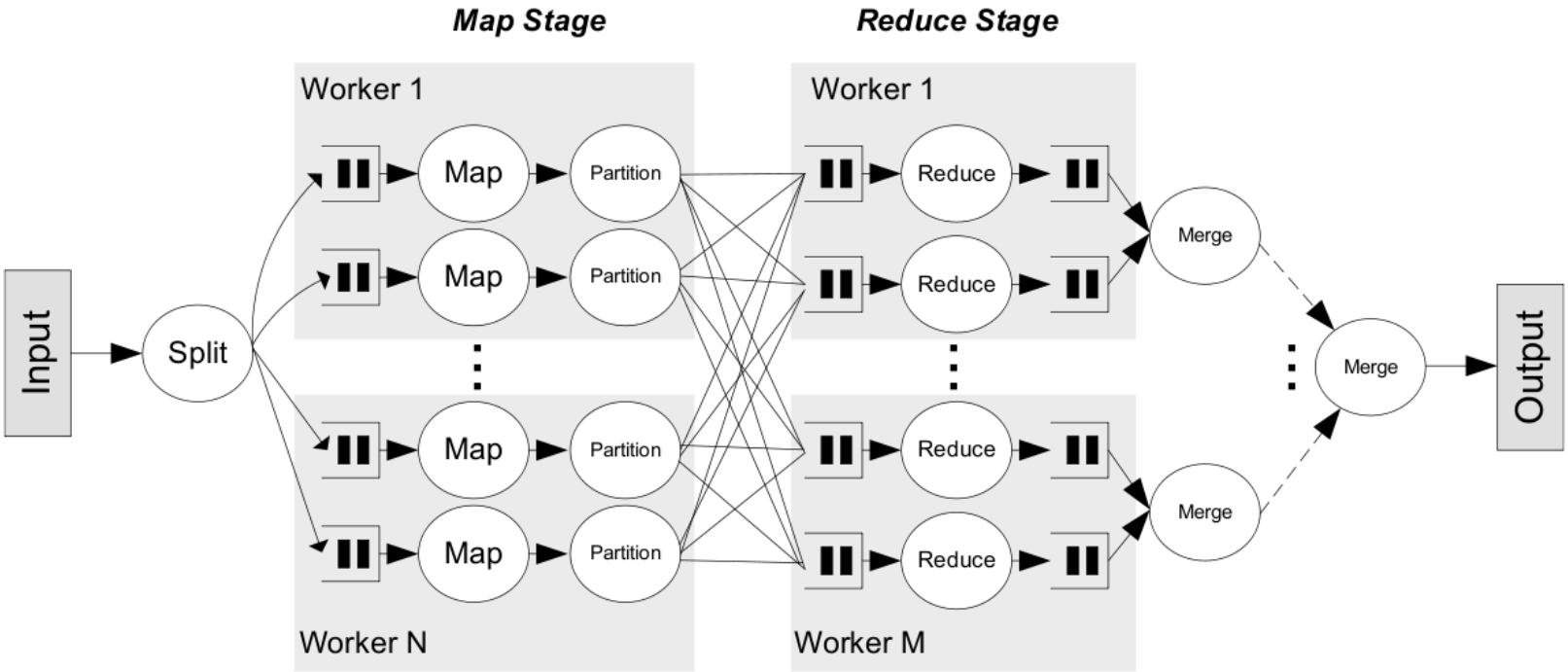
- v implementaci třídy `Mapper` nebo `Reducer`:

```
void setup(Mapper.Context context) {
    Path[] localFiles = DistributedCache.getLocalCacheFiles(context);
    // práce s lokálně uloženými soubory localFiles
}
```

Phoenix

- Implementace MapReduce v prostředí se sdílenou pamětí.

Přehled zpracování dat ve Phoenixu



Phoenix API

- `int phoenix_scheduler(scheduler args t * args)`
- `void emit_intermediate(void *key, void *val, int key size)`
- `void emit(void *key, void *val)`
- `int (*splitter_t)(void *, int, map args t *)`
- `void (*map_t)(map args t*)`
- `int (*partition_t)(int, void *, int)`
- `void (*reduce_t)(void *, void **, int)`
- `int (*key_cmp_t)(const void *, const void*)`

Detaily implementace Phoenixu

Detaily implementace Phoenixu

1. Předávání dat, dočasné buffery.
2. Dynamické přidělování úkolů.
3. Počty souběžně běžících vláken / procesů.
4. Velikosti vstupů pro `map` řádově jako L1 cache.
5. Detekce chyb, záložní úlohy před koncem.

Výkon Phoenixu

	CMP	SMP
Model	Sun Fire T1200	Sun Ultra-Enterprise 6000
CPU Type	UltraSparc T1 single-issue in-order	UltraSparc II 4-way issue in-order
CPU Count	8	24
Threads/CPU	4	1
L1 Cache	8KB 4-way SA	16KB DM
L2 Size	3MB 12-way SA shared	512KB per CPU (off chip)
Clock Freq.	1.2 GHz	250 MHz

	Description	Data Sets	Code Size Ratio	
			Pthreads	Phoenix
Word Count	Determine frequency of words in a file	S:10MB, M:50MB, L:100MB	1.8	0.9
Matrix Multiply	Dense integer matrix multiplication	S:100x100, M:500x500, L:1000x1000	1.8	2.2
Reverse Index	Build reverse index for links in HTML files	S:100MB, M:500MB, L:1GB	1.5	0.9
Kmeans	Iterative clustering algorithm to classify 3D data points into groups	S:10K, M:50K, L:100K points	1.2	1.7
String Match	Search file with keys for an encrypted word	S:50MB, M:100MB, L:500MB	1.8	1.5
PCA	Principal components analysis on a matrix	S:500x500, M:1000x1000, L:1500x1500	1.7	2.5
Histogram	Determine frequency of each RGB component in a set of images	S:100MB, M:400MB, L:1.4GB	2.4	2.2
Linear Regression	Compute the best fit line for a set of points	S:50M, M:100M, L:500M	1.7	1.6

Mars

Implementace MapReduce v prostředí se sdílenou pamětí pomocí GPU.

Detaily implementace Marsu

1. Velmi pobobné Phoenixu, rozdíly pouze kvůli vlastnostem GPU.
2. Základní API:
   - `void map_count(void *key, void *val, int keySize, int valSize)`
   - `void map(void *key, void* val, int keySize, int valSize)`
   - `void reduce_count(void* key, void* vals, int keySize, int valCount)`
   - `void reduce(void* key, void* vals, int keySize, int count)`
   - `void emit_intermediate_count(int keySize, int valSize)`
   - `void emit_intermediate(void* key, void* val, int keySize, int valSize)`
   - `void emit_count(int keySize, int valSize)`
   - `void emit(void *key, void* val, int keySize, int valSize)`

Výkon Marsu++ v porovnání s Marsem

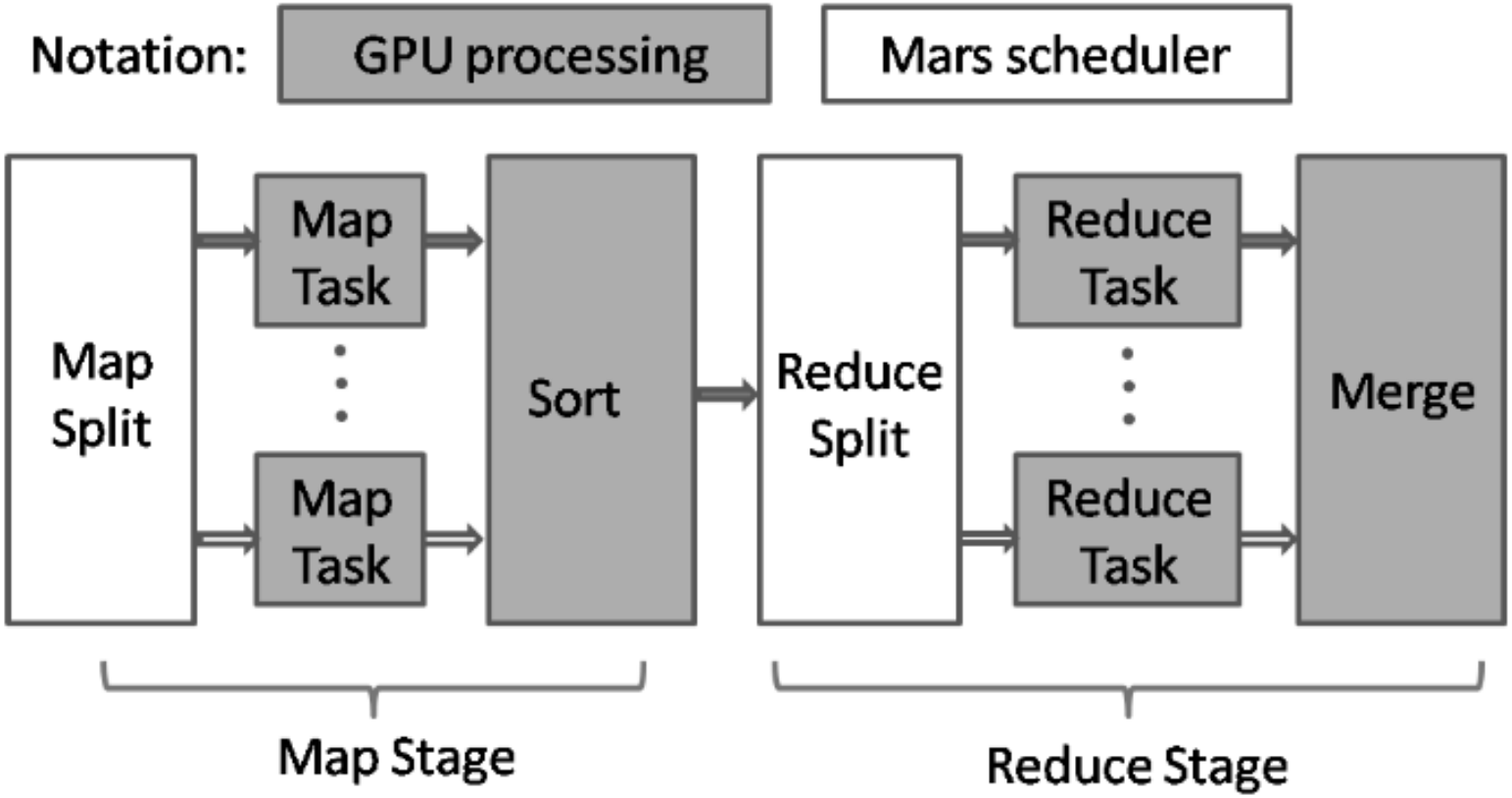


Figure 2. The work flow of Mars on the GPU.

	GPU	CPU
Processors	1350MHz × 8 × 16	2.4 GHz × 4
Data cache (shared memory)	16KB × 16	L1: 32KB × 4, L2: 4096KB × 2
Cache latency (cycle)	2	L1: 2 , L2: 8

Otázky?