Simple Parallel Computing in R Using Hadoop

Stefan Theußl

WU Vienna University of Economics and Business
Augasse 2–6, 1090 Vienna
Stefan.Theuss1@wu.ac.at

30.06.2009
Problem & Motivation

The MapReduce Paradigm

Package hive

Distributed Text Mining in R

Discussion
Motivation

Recap of my talk last year:

- Computer architecture: distributed memory (DMP) and shared memory platforms (SMP)
- Types of parallelism: functional or data-driven
- Parallel computing strategies: threads (on SMP), message-passing (on DMP), high-level abstractions (e.g., packages snow, nws)
Main motivation: large scale data processing

- Many tasks, i.e. we produce output data via processing lots of input data
- Want to make use of many CPUs
- Typically this is not easy (parallelization, synchronization, I/O, debugging, etc.)
- Need for an integrated framework
- preferably usable on large scale distributed systems
The MapReduce Paradigm
The MapReduce Paradigm

- Programming model inspired by functional language primitives
- Automatic parallelization and distribution
- Fault tolerance
- I/O scheduling
- Examples: document clustering, web access log analysis, search index construction, ...

Jeffrey Dean and Sanjay Ghemawat.
MapReduce: Simplified data processing on large clusters.

Hadoop ([http://hadoop.apache.org/core/](http://hadoop.apache.org/core/)) developed by the Apache project is an open source implementation of MapReduce.
The MapReduce Paradigm

![Conceptual Flow Diagram]

**Figure:** Conceptual Flow
The MapReduce Paradigm

A MapReduce implementation like Hadoop typically provides a distributed file system (DFS):

- Master/worker architecture (Namenode/Datanodes)
- Data locality
- Map tasks are applied to partitioned data
- Map tasks scheduled so that input blocks are on same machine
- Datanodes read input at local disk speed
- Data replication leads to fault tolerance
- Application does not care whether nodes are OK or not
The MapReduce Paradigm

Figure: HDFS architecture

Source: http://hadoop.apache.org/core/docs/r0.20.0/hdfs_design.html
Hadoop Streaming

- Utility allowing to create and run MapReduce jobs with any executable or script as the mapper and/or the reducer

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar
  -input inputdir
  -output outputdir
  -mapper ./mapper
  -reducer ./reducer
```
Hadoop Interactive (hive)
Hadoop InteractiVE (hive)

hive provides:

- Easy-to-use interface to Hadoop
- Currently, only Hadoop core (http://hadoop.apache.org/core/) supported
- High-level functions for handling Hadoop framework (hive_start(), hive_create(), hive_is_available(), etc.)
- DFS accessor functions in R (DFS_put(), DFS_list(), DFS_cat(), etc.)
- Streaming via Hadoop (hive_stream())
- Available on R-Forge in project RHadoop
hive: How to Get Started?

Prerequisites:

- Java 1.6.x
- Passwordless `ssh` within nodes
- Best practice: provide everything via shared filesystem (e.g., NFS).
hive: How to Get Started?

Installation:

► Download and untar Hadoop core from http://hadoop.apache.org/core/

► Modify JAVA_HOME environment variable in /path/to/hadoop-0.20.0/conf/hadoop-env.sh appropriately

► Further (optional) configuration:
  - mapred-site.xml e.g., configuration of number of parallel tasks (mapred.tasktracker.map.tasks.maximum)
  - core-site.xml e.g., where is the DFS managed? (fs.default.name)
  - hdfs-site.xml e.g., number of chunk replication (dfs.replication)

► Export HADOOP_HOME environment variable

► Start Hadoop either via command line or with hive_start()
Example: Word Count

Data preparation:

```r
> library("hive")
Loading required package: rJava
Loading required package: XML
> hive_start()
> hive_is_available()
[1] TRUE
> DFS_put("~/Data/Reuters/minimal", "/tmp/Reuters")
> DFS_list("/tmp/Reuters")
[1] "reut-00001.xml" "reut-00002.xml" "reut-00003.xml"
[4] "reut-00004.xml" "reut-00005.xml" "reut-00006.xml"
[7] "reut-00007.xml" "reut-00008.xml" "reut-00009.xml"
> head(DFS_read_lines("/tmp/Reuters/reut-00002.xml"))
[1] "<?xml version="1.0"?>"
[2] "<REUTERS TOPICS="NO" LEWISSPLIT="TRAIN" [...]
[4] " <TOPICS/>
[5] " <PLACES>"
[6] "  <D>usa</D>"
```
Example: Word Count

```r
mapper <- function(){
  mapred_write_output <- function(key, value)
    cat(sprintf("%s\t%s\n", key, value), sep = "")

  trim_white_space <- function(line)
    gsub("(^ +)|( +$)", "", line)
  split_into_words <- function(line)
    unlist(strsplit(line, "\[[:space:]]+"))

  con <- file("stdin", open = "r")
  while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
    line <- trim_white_space(line)
    words <- split_into_words(line)
    if(length(words))
      mapred_write_output(words, 1)
  }
  close(con)
}
```
Example: Word Count

```r
reducer <- function(){
  [
  env <- new.env(hash = TRUE)
  con <- file("stdin", open = "r")
  while (length(line <- readLines(con, n = 1,
    warn = FALSE)) > 0) {
    split <- split_line(line)
    word <- split$word
    count <- split$count
    if(nchar(word) > 0){
      if(exists(word, envir = env, inherits = FALSE)) {
        oldcount <- get(word, envir = env)
        assign(word, oldcount + count, envir = env)
      } else assign(word, count, envir = env)
    }
  }
  close(con)
  for (w in ls(env, all = TRUE))
    cat(w, "\t", get(w, envir = env), "\n", sep = "")
}
```
Example: Word Count

```r
> hive_stream(mapper = mapper,
              reducer = reducer,
              input = "/tmp/Reuters",
              output = "/tmp/Reuters_out")

> DFS_list("/tmp/Reuters_out")
[1] "_logs" "part-00000"

> results <- DFS_read_lines(
                     "/tmp/Reuters_out/part-00000")

> head(results)
[1] " -	2" "--	7" " :	1" ".	1" "0064 < /UNKNOWN >	1" "0066 < /UNKNOWN >	1"

> tmp <- strsplit(results, "\t")
> counts <- as.integer(unlist(lapply(tmp, function(x)
                   x[[2]])))
> names(counts) <- unlist(lapply(tmp, function(x)
                   x[[1]]))
> head(sort(counts, decreasing = TRUE))
the to and of at said
58 44 41 30 25 22
```
hive: Summary

- Further functions: DFS_put_object(), DFS_cat(), hive_create(), hive_get_parameter(), ...
- Currently, heavy usage of command line tools
- Java interface in preparation (presentation @ useR 2009)
- Use infrastructure of package HadoopStreaming?
- Higher-level abstraction (e.g., variants of apply())
Why Distributed Text Mining?

- Highly interdisciplinary research field utilizing techniques from computer science, linguistics, and statistics
- Vast amount of textual data available in machine readable format:
  - scientific articles, abstracts, books, ...
  - memos, letters, ...
  - online forums, mailing lists, blogs, ...
- Data volumes (corpora) become bigger and bigger
- Steady increase of text mining methods (both in academia as in industry) within the last decade
- Text mining methods are becoming more complex and hence computer intensive
- Thus, demand for computing power steadily increases
Why Distributed Text Mining?

- High Performance Computing (HPC) servers available for a reasonable price
- Integrated frameworks for parallel/distributed computing available (e.g., Hadoop)
- Thus, parallel/distributed computing is now easier than ever
- Standard software for data processing already offer extensions to use this software
Text Mining in R

- tm Package
- Tailored for
  - Plain texts, articles and papers
  - Web documents (XML, SGML, ...)
  - Surveys
- Methods for
  - Clustering
  - Classification
  - Visualization
Text Mining in R

I. Feinerer

*tm: Text Mining Package*, 2009
URL http://CRAN.R-project.org/package=tm
R package version 0.3-3

I. Feinerer, K. Hornik, and D. Meyer
Text mining infrastructure in R
ISSN 1548-7660
URL http://www.jstatsoft.org/v25/i05
Distributed Text Mining in R

Example: Stemming

- Erasing word suffixes to retrieve their radicals
- Reduces complexity
- Stemmers provided in packages Rstem\(^1\) and Snowball\(^2\)

Data:

- *Reuters-21578*: one of the most widely used test collection for text categorization research
- (New York Times corpus)

---

\(^1\)Duncan Temple Lang (version 0.3-0 on Omegahat)  
\(^2\)Kurt Hornik (version 0.0-3 on CRAN)
Motivation:

- Large data sets
- Corpus typically loaded into memory
- Operations on all elements of the corpus (so-called *transformations*)

Available transformations: `stemDoc()`, `stripWhitespace()`, `tmTolower()`, ...
Strategies:

- Text mining using tm and Hadoop/hive\textsuperscript{1}
- Text mining using tm and MPI/snow\textsuperscript{2}

\textsuperscript{1}Stefan Theußl (version 0.1-1)
\textsuperscript{2}Luke Tierney (version 0.3-3)
Distributed Text Mining in R

Solution (Hadoop):

- Data set copied to DFS (‘DistributedCorpus’)
- Only meta information about the corpus in memory
- Computational operations (Map) on all elements in parallel
- Work horse `tmMap()`
- Processed documents (revisions) can be retrieved on demand
> library("tm")
Loading required package: slam
> input <- "~/Data/Reuters/reuters_xml"
> co <- Corpus(DirSource(input), [...])
> co
A corpus with 21578 text documents
> print(object.size(co), units = "Mb")
65.5 Mb

> source("corpus.R")
> source("reader.R")
> dc <- DistributedCorpus(DirSource(input), [...])
> dc
A corpus with 21578 text documents
> dc[[1]]
Showers continued throughout the week in
[...]  
> print(object.size(dc), units = "Mb")
1.9 Mb
Mapper (called by tmMap):

```r
mapper <- function () {
    require("tm")
    fun <- some_tm_method
    [...] 
    con <- file("stdin", open = "r")
    while (length(line <- readLines(con, n = 1L, 
                                  warn = FALSE)) > 0) {
        input <- split_line(line)
        result <- fun(input$value)
        if (length(result))
            mapred_write_output(input$key, result)
    }
    close(con)
}
```
Distributed Text Mining in R

Infrastructure:

▶ Development platform: 8-core Power 6 shared memory system

<table>
<thead>
<tr>
<th>IBM System p 550</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 128</td>
</tr>
<tr>
<td>2-core IBM POWER6 @ 3.5 GHz GB RAM</td>
</tr>
</tbody>
</table>

▶ Computers of PC Lab used as worker nodes
  ▶ 8 PCs with an Intel Pentium 4 CPU @ 3.2 GHz and 1 GB of RAM
  ▶ Each PC has > 20 GB reserved for DFS

MapReduce framework:

▶ Hadoop (implements MapReduce + DFS)
▶ R (2.9.0) with tm (0.4) and hive (0.1-1)
▶ Code implementing ‘DistributedCorpus’
▶ Cluster installation coming soon (loose integration with SGE)
BENCHMARK

Reuters-21578:

- Single processor runtime (\texttt{lapply()}): 133 min.
- \texttt{tm/hive on 8-core SMP (hive\_stream())}: 4 min.
- \texttt{tm/snow on 8 nodes of cluster@WU (parLapply())}: 2.13 min.
Benchmark

Runtime

Speedup

# CPU

Runtime [s]

Speedup

1 2 3 4 5 6 7 8

500 1500 2500 3500

1 2 3 4 5 6 7 8
Excursion: How to store text files in the DFS

- **Requirement:** access text documents in R via `[]`
- **Difficult to achieve:** almost random output after calling `map` in Hadoop
- **Output chunks automatically renamed to `part-xxxxx`**.
- **Solution:** add meta information to each chunk (chunk name, position in the chunk)
- **Update `DistributedCorpus` after `Map` process**
Lessons Learned

- Problem size has to be sufficiently large
- Location of texts in DFS (currently: ID = file path)
- Thus, serialization difficult (how to update text IDs?)
- Remote file operation on DFS around 2.5 sec. (will be significantly reduced after Java implementation)
Conclusion

- MapReduce has proven to be a useful abstraction
- Greatly simplifies distributed computing
- Developer focus on problem
- Implementations like Hadoop deal with messy details
  - different approaches to facilitate Hadoop’s infrastructure
  - language- and use case dependent
Thank You for Your Attention!

Stefan Theußl
Department of Statistics and Mathematics
e-mail: Stefan.Theussl@wu.ac.at
URL: http://statmath.wu.ac.at/~theussl

WU Vienna University of Economics and Business
Augasse 2–6, A-1090 Wien