Practical Issues using Distributed Computing Environments – Apache Hadoop

Cristian TOMA
IT&C Security Master,
Department of Economic Informatics and Cybernetics,
The Bucharest University of Economic Studies,
ROMANIA
www.ism.ase.ro
cristianvtoma@gmail.com

Abstract: The paper presents practical results obtained with sequential standard programming, RPC/RMI mechanism and Apache Hadoop distributed computing platform for a problem of computation time and power that might be used in e-mail text searching. First section is about Distributed Computing technologies and middleware introduction. In second and third section are shown few details about RPC/RMI and Apache Hadoop approaches. The fourth section presents the results of the computation for a classic problem such as word counting from large text files using standard versus remote procedure call versus map-reducing approach. In the end are shown the main advantages of the distributed systems and computing environments.

Key-Words: Apache Hadoop, HTC – High Throughput Computing/HPC – High Performance Computing, distributed computing, map-reduce, distributed file system.

1. Distributed Computing Intro

The notion of distributed system or distributed computing system has several definitions in the literature, more or less equivalent. Thus, a distributed system might be defined, according with some researchers, as a collection of nodes that are:

- computers;
- processors;
- autonomous processes;

which are interconnected. Each node within the system has one memory of its own. Also, the nodes must be able to exchange information between them. In parallel computing the memory is shared by the processors and there is a common approach for mainframes, and partially for Type 1 Cloud systems – "bare metal".

In Tannenbaum the definition is a more restrictive one, considering that a system is distributed only if the existence of autonomous nodes is transparent to the users. He sees the whole as an independent entity the entire distribution process being handled by the kernel of the operating systems within the operational nodes.

In other words, the user of a distributed system is not aware that there are multiple processors; the system looks like a single virtual processor. The allocation of work on processors and disk files, the transfer of files between where they are stored and where they are needed and any other system function - all these must run automatically.

Considering the criteria regarding the type of connection between nodes, then we distinguish two types of such systems. The first type is that of *closely connected/coupled systems,* where multiple processors usually share the same memory and consult the same clock (supercomputers, etc. mainframes.).
The second type is that of *loosely connected/coupled systems*, where each system has its own memory and its own clock. *Without a unanimous acceptance, loosely connected/coupled systems are identified with distributed systems.*
The most important difference between a distributed and parallel system is in Figure 1:

ISSN 2067 – 4074

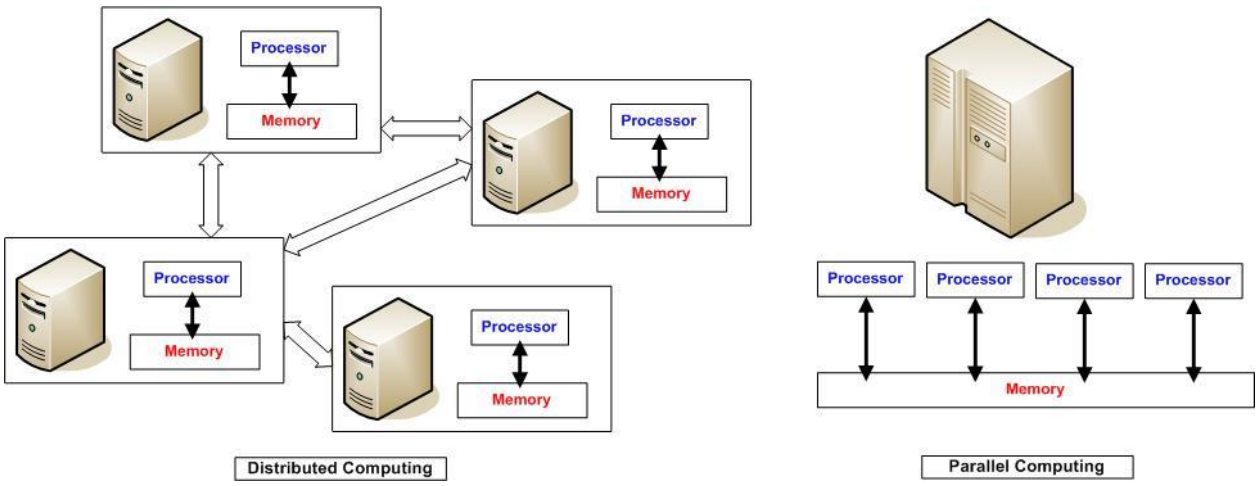

Figure 1. Difference between distributed and parallel system

Of course, some developers may use a distributed system to process in parallel manner a computational problem. So, parallel and distributed computing are interfering together.

The most used "core" distributed computing technologies are:

- RPC – Remote Procedure Call / RMI – Remote Method Invocation
- CORBA – Common Broker Request Broker Architecture and DCOM – Distributed Component Object Model
- Web Services – SOAP – Simple Object Access Protocol), WSDL – WebService Description Language
- EJB – Enterprise Java Beans (synchronous versus asynchronous processing using MOM – Message Oriented Middleware implementation provided by JMS – Java Message Service used in MDB – Message Driven Bean)

The packaged use of the above programming technologies from above in the real solutions, may reveal as distributed computing models, systems, frameworks or SDKs such as: JADE – A.I. Agents Programming Framework; GlobusTookit – software kit for developing Grid Networks using mainly web services; Condor – integrated system for Grid Computing; Apache Hadoop – simplified programming model which allows the developer to write and test distributed systems; Nimbus – Cloud Computing Platform/Infrastructure for science (so can be used also by universities).

Grid computing is the use of computation power from various computer resources from different places across multiple administrative domains, in order to achieve a common goal. In the universities, it is used for distributed computing. Grid technologies, paradigms and services are mixing and mashing with SOA – Services Oriented Architecture (web services processing) orchestrated by BPEL/BPM engines and cloud computing concepts, and there are some certain services which are migrating from grids to the clouds environments.

According to [12], Cloud Computing is the hardware and software delivered for processing as a service over a network. As comparison of Cloud systems to the Grid systems, there are the following major differences:

- Grid Systems run processes vs. Cloud Systems run Virtual Machines
- Grid Systems are focused on job delegation vs. Cloud Systems are focused on remote resource allocation (provisioning)
- Grid Systems take in to account On-the-fly dynamic deployment vs. Cloud Systems have Hot-Deploy issues.

The argument for performance of the Grid Computing System Condor over Cloud Computing systems is that virtual machines (VM) can be considered

jobs/processes because: VM image is executable, instance data is input and modified image is output.

Cloud computing is a "reshape" and a new technological approach (mainly through operating system virtualization) that use mature concepts inherited from Grid Computing included in a larger context of HTC – High Throughput Computing, HPC – High Performance Computing and MTC – Many Task Computing.
The main difference between HPC (e.g. SuperComputer) and HTC (e.g. Grid Computing System) is given by the length of the period of time used for massive processing, from point of view of large amounts of computing power. HPC is focused for using large computing power for short periods of time – hours/days (e.g. Supercomputers - tightly coupled parallel jobs), but HTC is focused for using also large computing power for longer times – months/years (e.g. Grids - independent, sequential, distributed and loosely jobs). MTC is a bridge between HPC and HTC.

2. RPC/RMI Mechanism as Core Middleware for Distributed Computing

The basic mechanism for distributing tasks and computing is to use RPC/RMI – Remote Procedure Call / Remote Method Invocation mechanism. A more advanced approach is to use specific frameworks, such as Open MPI or GRID or Cloud IaaS/PaaS for the distributed computing, but each solution is good for a certain type of a set of problems. A comprehensive comparison between the "standard" technologies for distributed processing like – RMI, CORBA and DCOM is presented in [6]. The companies and communities have been developed specific platforms in Java and/or C++ for distributed computing starting from the "standard" technologies. Figure 2 shows the RMI items.

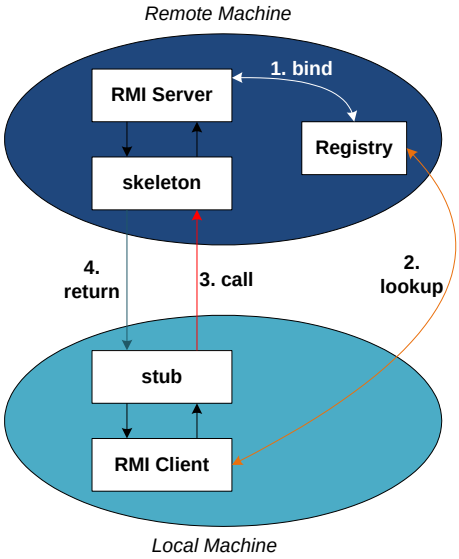

Figure 2. RMI – Remote Method Invocation Architecture

Java RMI allows the software developers to call functions/methods from an object that is inside the memory of another virtual machine/computer than the holder object of the method. For instance if the one needs to compute the sum between 1 and 2, because the client does not need the computational power to achieve the sum, may transmit over the network the parameter 1 and 2; the server has resources to compute the result of 3 and to send back to the client.
In figure 1.2 the logical flow is the following:

- The RMI Server object should register its name and virtual machine address to the Registry – **bind**
- The RMI Client object searches the name and the address of RMI Server object in Registry – **lookup**
- The RMI Stub object serializes – "marshaling" the parameters to the RMI Skeleton object using JRMP protocol – set of rules. The RMI Skeleton object and de-serializes the received parameters – "un-marshaling". The RMI Skeleton object calls the method from the server object – **call**.
- The RMI Skeleton object returns the results back to RMI Stub object, by "marshaling" the fields and methods prototype of the result's object - **return**.

ISSN 2067 – 4074

The overall principle requests to have the following main items to be involved in any kind of distributed platform (the "core" logic is "quite the same"):

- Name Service:
  - rmiregistry – for RMI;
  - COS – CORBA Object Service: tnameserv.exe (Non-persistent) & orbd.exe (Persistent) – for CORBA;
  - UDDI/AC – Active Directory or a LDAP Server for Web Services)
- Communication objects for the client and server side:
  - Stub (Client) and Skeleton (Server, starting with RMI 1.2 the skeleton is called also stub) for RMI;
  - ORB – Object Request Broker (Client) and POA – Portable Object Adapter/ORB (Server)
  - Web Service Object/Process/Application for the client side capable to send XML messages over HTTP encapsulated in SOAP Requests AND Java Servlets or objects from the addnotated classes for the server side.
- Interfaces/"Independent" languages as input for compilers in order to generate server and client objects that communicate over the communications protocol:
  - Java interface that extends Remote interface for RMI
  - IDL – Intermediary Definition Language for CORBA
  - WSDL – Web Services Definition Language for Web Services
- Communication Protocols:
  - JRMP – Java Remote Method Protocol for RMI
  - GIOP or IIOP - Internet Inter-ORB Protocol for CORBA
  - SOAP – Simple Object Access Protocol for web services
- Compilers and Products

3. Apache Hadoop Core Components

Apache Hadoop is used for implementing "a large-scale distributed batch processing infrastructure" [8]. For testing purposes, it maybe be used on a single machine, but in production may be deployed to scale to hundreds or thousands of PCs/servers, each with several processor cores. Apache Hadoop provides an efficient solution to distribute large amounts of data and work across a set of machines.

The computation of large volumes of data has been done with other solutions before, usually in a distributed setting. Apache Hadoop, differently by others solutions, is unique is its **simplified programming model** which allows the developer to write and test distributed systems very easy. It is **efficient for automatic distribution of data and work across machines** and in turn utilizing the underlying parallelism of the CPU cores.

The core components of the Apache Hadoop are:

- HDFS – Hadoop Distributed File System.
- Map-Reduce Implementation

3.1 HDFS – Hadoop Distributed File System for DATA DISTRIBUTION

In Apache Hadoop cluster, the data is distributed to all the nodes of the cluster at the loading time within the HDFS. The large data files are split into chunks which are managed by different nodes in the cluster of HDFS – Hadoop Distributed File System. If a computer/machine will have a serious failure, the data is still available. The data is re-replicated due to an active monitoring system responsible to repair the system failures which can result in partial storage. The file chunks are replicated and distributed across several machines within the cluster. A single namespace within the Hadoop cluster contains one or more data nodes, so their contents are accessible to any node. A simplified view of the splitting process in HDFS is in figure 3.

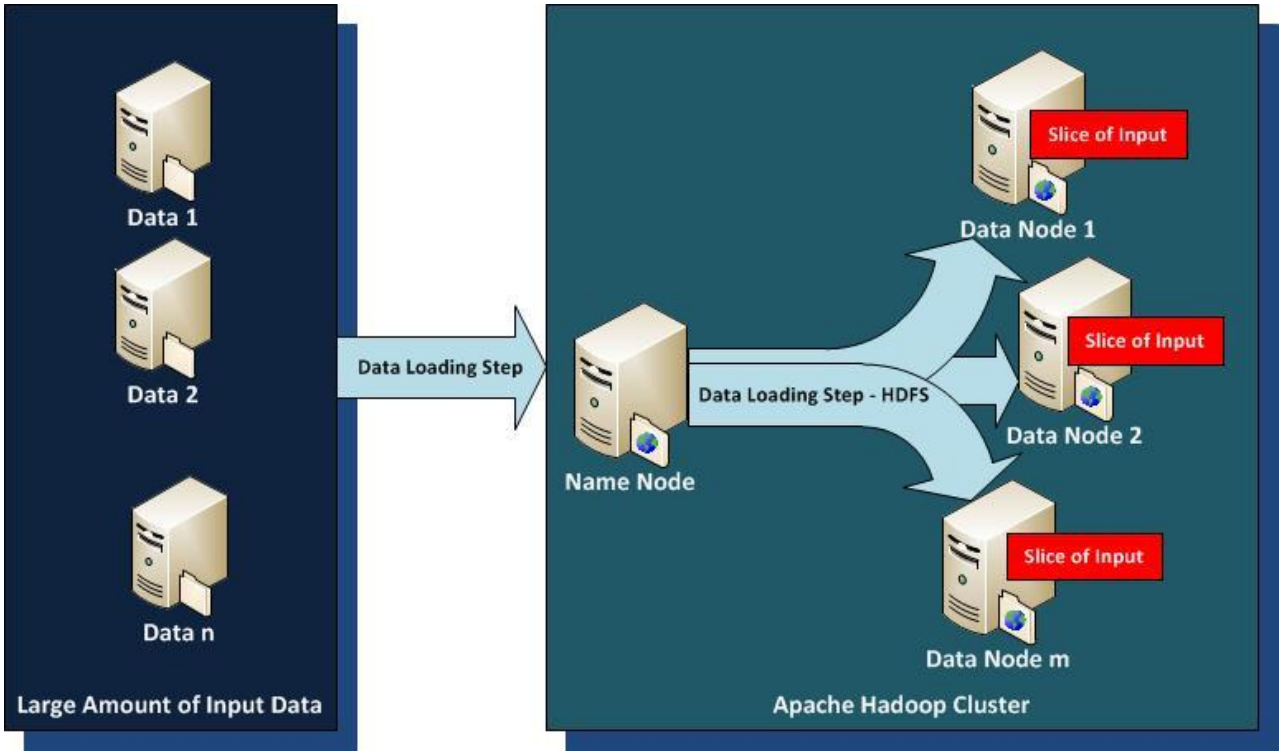

Figure 3. HDFS – Hadoop Distributed File System – Data is distributed at load time of the file

Data contains **records** in the Hadoop programming framework. Every input file is split into one or more lines, alternatively into other formats specific to the application. A subset of these records may be processed independently by each process running on a node in the cluster. The Hadoop framework goal is to schedule these processes NEAR to the location of data/records using knowledge from the distributed file system. Each process running on a node operates on a subset of the data, since the files are spread across the distributed file system as chunks. The data location of chunk is an important factor for choosing the operating process/node. Most data is read from the local disk directly into the CPU, preventing unnecessary network transfers. This approach to **move computation to the data**, instead of moving the data to the computation allows Hadoop to achieve high performance.
Most block-structured file systems use a block size around 4 or 8 KB, but for Hadoop, the default block size in HDFS is 64MB – much larger. This allows HDFS to decrease the amount of metadata storage required per file.

3.2 MapReduce – ISOLATED PROCESSES

Hadoop limits the bandwidth of the network communication performed by the processes, as each individual record is processed by an isolated task. This may induce a major limitation at first glance, but this makes the whole framework more reliable. Hadoop will run "special" programs and distribute them across a cluster. The programs must be written in a special Java "flavor" in order to be compliant to a particular programming model, named "MapReduce."
Figure 4 shows the process of Map-Reduce process:

ISSN 2067 – 4074

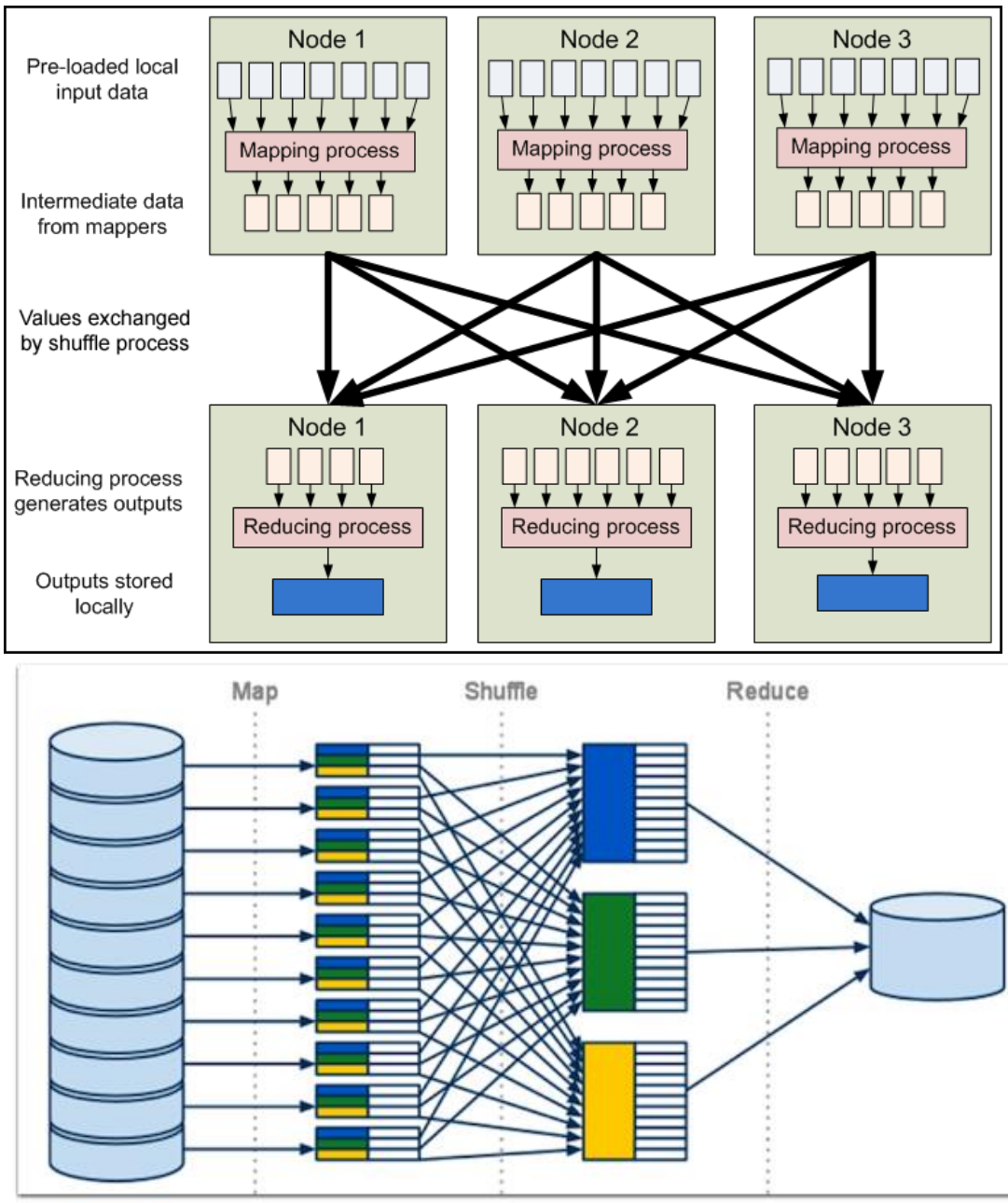

Figure 4. Map-Reduce Process (copyright from Yahoo [4] and Google [13])

In MapReduce, the records from the input file are processed in isolation by tasks called *Mappers*. The output from the Mappers is then brought together into a second set of tasks called *Reducers*, where the results from different mappers could be merged together.

4. Comparative results in practical sample

A sample proof of concept solution has been tested by the paper's author. The solution should provide the most efficient way to index the frequency of the words from some inputs file like literature books for search engines optimizations.
The Word Count program is written in 3 approaches types:

- Standard C++,
- Java RMI – Remote Method Invocation,
- "Special flavor" of Java for Apache Hadoop.

The program versions are processing texts from books and log files of with lengths from 1.4 MB to 16 MB.

4.1 Standard C++ stand-alone program development

Next table contains the C++ source code for processing the word count using a tree data structure from STL – Standard Template Library.

Table 1. C++ Source Code for Word Counting stand-alone program

```
#include <iostream>
#include <fstream>
#include <map>
#include <string>
#include <time.h>

using namespace std;

int main(int argc, char* argv[])
{
	time_t secStart = time(NULL);
	ifstream
fin(argv[1],ios_base::in);
	ofstream out("out.txt",
ios_base::out);
	map<string,int> f;
	map<string,int>::iterator it;

	string linie;
	char* cuv,*sep = " ;:.\n";

	while(getline(fin,linie))
	{
		cuv =
strtok((char*)linie.c_str(),sep);

		while(cuv)
		{
			string t(cuv);
			it = f.find(t);

	if(it!=f.end())(*it).second++
;
			else

	f.insert(pair<string,int>(t,1
));
			cuv =
strtok(NULL,sep);
		}
	}
	for(it = f.begin();it !=
f.end(); it++)
		out<<"\n
"<<(*it).first<<": "<<(*it).second;

	fin.close();
	out.close();
	time_t secStop = time(NULL);
	cout<<"\n The processing of
the file have been made
within"<<(secStop - secStart)<<"
seconds."<<endl;
	return 0;
}
```

The C/C++ program generates the results in 45 seconds after processing on a dual core computer/PC and with 3 GB of RAM.

4.2 Java RMI program development

For Java RMI, it has been used an PC as dispatcher and RMI client and two RMI servers. The table 2 shows the source code for RMI server and RMI client:

Table 2. Java RMI interface, client and server source code

```
//RMI Interface:
package eu.ase.rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;

import java.util.Hashtable;

public interface WordCountInterface
extends Remote {
	public Hashtable<String,
Integer> doWordCount(byte[]
fileContent) throws
RemoteException;
}

////////////////////////////////
//RMI Server Implementation
package eu.ase.rmi;

import
java.rmi.server.UnicastRemoteObject
;
import java.rmi.RemoteException;
import java.util.HashMap;
import java.util.Scanner;
```

ISSN 2067 – 4074

```
public class WordCountImpl extends
UnicastRemoteObject implements
WordCountInterface {
      private static final long
serialVersionUID = 1L;

      public WordCountImpl() throws
RemoteException {
            super();
      }

      public HashMap<String,
Integer> doWordCount(String
txtFileContent) throws
RemoteException {
            Scanner scan = new
Scanner(txtFileContent);
        HashMap<String, Integer>
listOfWords = new HashMap<String,
Integer>();

        String word = scan.next();
        int countWord = 0;

if(!listOfWords.containsKey(word)){
            listOfWords.put(word,
1);
        } else {
            countWord =
listOfWords.get(word) + 1;

listOfWords.remove(word);
            listOfWords.put(word,
countWord);
        }

        return listOfWords;
      }
}

////////////////////////////////
//RMI Server Running:
package eu.ase.rmi;

import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;

public class ProgMainWordCount {
      public static void
main(String args[]) {
          //set the security
manager
          try {

System.setSecurityManager(new
RMISecurityManager());

            //create a local
instance of the RMI server object
            WordCountImpl Server
= new WordCountImpl();

            //put the local
instance in the registry

Naming.rebind("rmi://127.0.0.1:1099
/SAMPLE-SERVER-WORDC", Server);


System.out.println("Server
waiting.....");
          } catch
(java.net.MalformedURLException me)
{

System.out.println("Malformed URL:
" + me.toString());
          } catch (RemoteException
re) {

System.out.println("Remote
exception: " + re.toString());
          }
      }
}

////////////////////////////////
//RMI-Client running:
package eu.ase.rmi;

import java.io.File;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;

import java.rmi.Naming;
import java.rmi.RMISecurityManager;

import java.util.HashMap;
import java.util.Iterator;

public class ProgMainClientRmiWordC
{
      public static void
main(String[]  args) {
            // set the security
manager for the client

System.setSecurityManager(new
RMISecurityManager());
            //get the remote object
from the registry
            try {
              File f = new
File(args[0]);
```

```
                FileInputStream fis
= new FileInputStream(f);
                DataInputStream in
= new DataInputStream(fis);
                byte[] fileContent
= new byte[(int)f.length()];

in.readFully(fileContent);
                String
txtFileContent = new
String(fileContent);
                in.close();
                fis.close();


System.out.println("Security
Manager loaded");
                String url =
"rmi://127.0.0.1:1099/SAMPLE-
SERVER-WORDC";
                WordCountInterface
remoteObject =
(WordCountInterface)Naming.lookup(u
rl);

System.out.println("Got remote
object");

                HashMap<String,
Integer> result =
remoteObject.doWordCount(txtFileCon
tent);
                Iterator<String>
iterator =
result.keySet().iterator();

                while
(iterator.hasNext()) {
                    String key =
iterator.next().toString();
                    String value =
result.get(key).toString();


System.out.println(key + " = " +
value);
                }
            } catch (IOException
ioe) {

ioe.printStackTrace();
            } catch
(java.rmi.NotBoundException exc) {

System.out.println("NotBound: " +
exc.toString());
            }

        } //end method
}
```

The time for processing was around 26 seconds using the same input file as in stand-alone sample, but with two RMI servers (dual core computer/PC-server and with 2 GB of RAM) and one RMI client.

4.3 Java on Apache Hadoop – Map-Reduce solution development

On a Linux Ubuntu 8 distribution the same problem is solved using Apache Hadoop platform and the one can observe the time of execution half of 45 seconds on a virtual machine from VMWare with 512MB RAM and 1 processor core.
The advantages are obviously highlighted in the print-screens, but a lot of test on different infrastructures should be made in order to establish certain meaningful results.
The sample of processing with Map-Reduce approach for word count process is presented in figure 5.

5. Conclusions

The best performance in terms of speed has been obtained in Apache Hadoop environment, second has been the RMI approach and third was stand-alone program. There are not taken into account the delay of the networks links, HDD speed, or processors cores compatibility. The best results have been obtained with Hadoop, also because of the underlying parallelism provided by the Apache framework.
The need of the design of distributed systems is motivated by some specific advantages. The most important of these advantages are:

- *Exchange of information*. The strong increase in the amount of information and need to exchange information quickly between different points in distant geographical locations are necessary to connect between autonomous computers.

ISSN 2067 – 4074

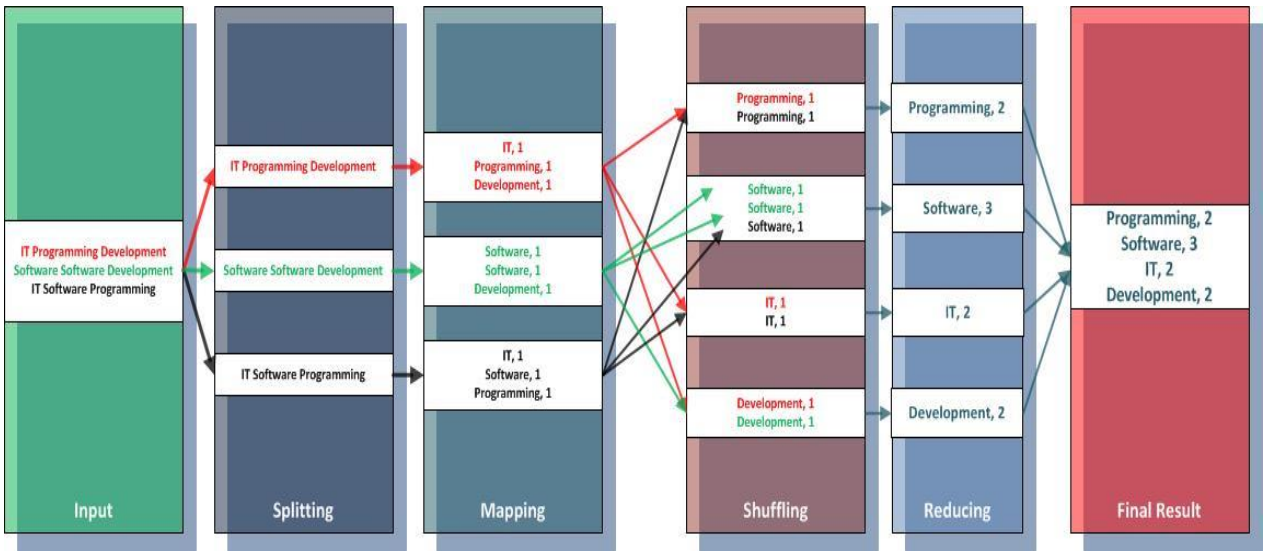

Figure 5. Sample of Map-Reduce for Word Count process.

- *Sharing of resources*. An organization prefers to buy cheaper, reasonably powerful computers than to buy one, much more powerful but more expensive. In this way it becomes necessary to interconnect these small computers, possibly with a small number of more powerful computers whose resources (memory, processing power, and large capacity peripherals) will be shared between them. The cost of such a network capacity increase is much smaller than of a single strong computer connected to resources.
- *Increased safety in operation*. If a computer system consists of a single computer malfunction makes it impossible to use the whole system. When designing a distributed system, its reliability is mainly taken into account. Thus, the failure of a node does not disturb one operation of the other, but the nodes take, in many cases, the burden of a failure.
- *Increased performance*. The presence of multiple processors in a distributed system makes it possible to reduce the time to build a massive calculation. This is done by dividing the tasks among different processors, collecting the partial results and determining the final result. This process is known as the parallelization of the calculations.
- *Specialization nodes*. Designing an autonomous computer system with more functionality can be very difficult and practical reasons. This design can be simplified by dividing the system into modules, each module implementing some of the features and communicating with other modules.

Algorithms used in distributed systems (as in all systems) must be fair, flexible and efficient. Development of a distributed algorithm differs significantly from the development of a centralized algorithm peculiarity due mainly distributed systems.

A research approach maybe stimulating for distributed and parallel computing involving Apache Hadoop in Cloud platforms and web based environments.

Acknowledgement

Parts from this paper have been presented in The 11th International Conference on Informatics in Economy, IE 2012 (www.conferenceie.ase.ro).

References

[1] Tom White, *Hadoop – The Definitive Guide*, O'Reilly Media, 528 pp, ISBN-10: 0596521979, ISBN-13: 978-0596521974, US 2009.

[2] Chuck Lam, *Hadoop in Action*, Manning Publishing, 325 pp, ISBN-10: 1935182196 , ISBN-13: 978-1935182191, US 2010.

[3] Apache Hadoop Project, http://hadoop.apache.org/

[4] Hadoop Project, HDFS Architecture, available at http://hadoop.apache.org/docs/stable/hdfs_design.html

[5] Vincent McBurney, So what is better, ETL or ELT?, Toolbox.com, Available at: http://it.toolbox.com/blogs/

[6] Gopalan Suresh Raj, A Detailed Comparison of CORBA, DCOM and Java/RMI, available at: http://my.execpc.com/~gopalan/misc/compare.html

[7] Wikipedia, Distributed computing, available at: http://en.wikipedia.org/wiki/Distributed_computing

[8] Yahoo, Yahoo! Hadoop Tutorial, available at: http://developer.yahoo.com/hadoop/tutorial/

[9] The Outline of Science, Vol. 1 (of 4) by J. Arthur Thomson - http://www.gutenberg.org/ebooks/20417

[10] The Notebooks of Leonardo Da Vinci — Complete by Leonardo da Vinci - http://www.gutenberg.org/ebooks/5000

[11] James Joyce, Ulysses, available at: http://www.gutenberg.org/ebooks/4300

[12] Wikipedia, Cloud Computing, avaialble at: https://en.wikipedia.org/wiki/Cloud_computing

[13] Google, Map-Reduce Programming, available at: https://developers.google.com/appengine/docs/python/dataprocessing/