

HERIOT-WATT UNIVERSITY

MENG SOFTWARE ENGINEERING

4TH YEAR DISSERTATION

DELIVERABLE ONE
REQUIREMENTS DOCUMENT

Map Reduce in Java

Author:

Gordon Govan

Supervisor:

Greg Michaelson

Second Reader:

Fairouz Kamareddine

November 17, 2008

Contents

1	Aims and Objectives	2
1.1	Aims	2
1.2	Objectives	2
2	The Map-Reduce Framework	4
2.1	The Map Function	4
2.2	The Reduce Function	4
2.3	Processing Elements	5
2.4	Example Run Through	5
3	Beowulf Cluster	9
4	Technical Literature	10
4.1	MapReduce: Simplified Data Processing on Large Clusters	10
4.2	Parallel Programming on Linux Networks Using MPI & PVM	10
4.3	Algorithmic Skeletons: Structured Management of Parallel Computation . .	11
4.4	Skeleton Realisations from Functional Prototypes	12
4.5	Bringing Skeletons out of the Closet	13
4.6	Explicit and Implicit Parallel Functional Programming: Concepts and Im- plementation	13
5	Requirements	14
5.1	Implement A Map-Reduce Framework	14
5.2	Fault Tolerance	14
6	Testing Strategy	15
6.1	Test Applications	15
6.2	Data	15
6.3	Unit Testing	16
6.4	Fault Tolerance Testing	16
7	Time Table	17
	References	18

1 Aims and Objectives

1.1 Aims

The goal of this project is to implement and evaluate a Java skeleton that acts similarly to Google's Map-Reduce framework [4]. The Map-Reduce framework is discussed in detail in Section 2. The framework will be programmed in Java and communicate between machines using mpiJava¹. It will be able to speed up operations on large datasets, if the operations do not have to be sequential, by splitting the data among many machines, having each machine perform the operations on their subset of the data, and then collecting the results from each machine and operating on them to reduce them into a few values.

1.2 Objectives

The project can be broken down into several smaller tasks.

Research: The first activity is to find out how Google's Map-Reduce framework works and find technologies in Java that would help in implementing this.

Building Framework: A framework to implement the Map-Reduce concepts in Java will be programmed.

Building Test Applications: In order to test and evaluate the framework then applications will have to be written to use the framework in order to benchmark it. For more information on testing see Section 6.

Testing The Framework: Test applications will be run on the framework, changing the size of data and number of processors that it is run across. The results from these tests will be stored for later evaluation.

Evaluating Results: The test results will be evaluated to find choke points in the system and other areas for improvement.

¹<http://www.hpjava.org/mpiJava.html>

Improving The Framework: The framework can then be improved based upon the results of the evaluation in order to allow for faster execution on larger datasets.

The last few stages can be seen as being iterative, the results of the testing feeding more improvements into both the Map-Reduce framework and the test applications.

2 The Map-Reduce Framework

Google's Map-Reduce framework is described in Dean and Ghemawat [4]. The framework is intended to process very large datasets across large numbers of clustered machines. Google use the framework across thousands of machines and terabytes of data.

2.1 The Map Function

The map function is a common function found in functional programming languages [6]. It takes as its arguments a function and a list of values. The map function then applies the supplied function to each of the elements in the list in turn. As the function is only applied to a single element in the list at a time then the map function can be split and executed on multiple processing elements.

The map function is typically written in SML as:

```
fun map f [] = [] |
    map f (h::t) = f h :: map f t;
```

The map function in Google's map reduce is slightly different as it operates on pairs, not just single values. The function is able to give a different key for output than it took as the input, this can be useful for finding links to pages from a list of links from pages.

2.2 The Reduce Function

The reduce function is also commonly found in high level programming languages [6]. It is used to reduce all the elements of a list into a single value. It takes as arguments: the function to be used to reduce, the value for the list to be reduced into, and the list to be reduced. The reduce function can also be split on to multiple processing elements and then the values returned from those reductions can then be reduced together.

The reduce function is written in SML as:

```
fun reduce f v [] = v |
    reduce f v (h::t) = reduce f (f v h) t;
```

Google's reduce function is again used on key-value pairs. The reduce function is only used on pairs with the same key, this is good for counting how many times something (the key) happens.

2.3 Processing Elements

Google's Map-Reduce framework is used across clusters of processing elements (PEs) where a processing element can be a machine or a core in a multi-core processor, several PEs can even exist on a single core. Each PE is sent a section of the input data to the system. The PE then performs the map operation on the list it was given. Before the reduction stage the pairs in the list are sorted by their keys so that reduction upon pairs with the same key can be done without any searching. It then reduces the new list before sending it back to another PE for further reduction as shown in Figure 1. The PE that is responsible for distributing the data and for performing the final reductions is called the master, all other PEs are referred to as slaves.

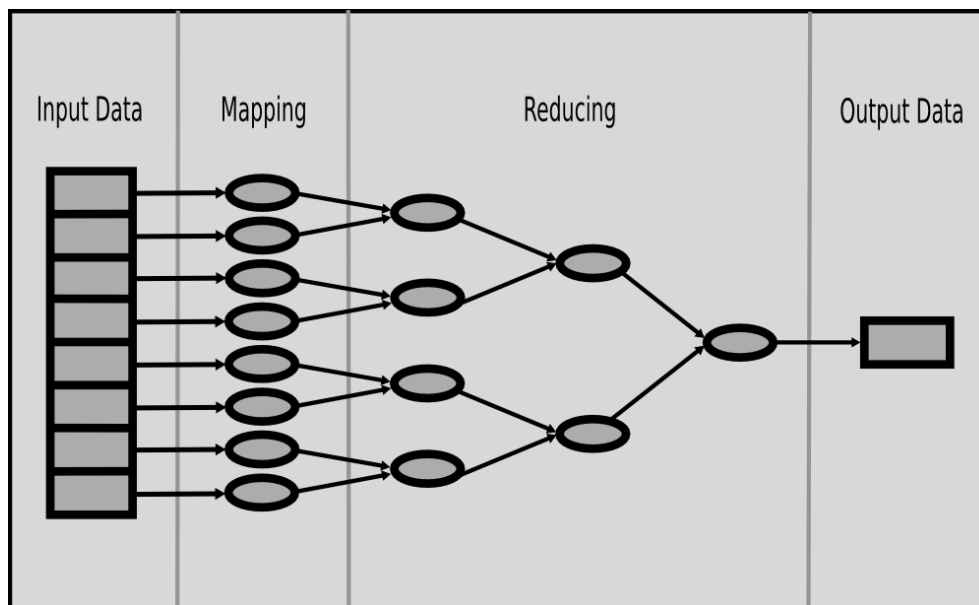


Figure 1: Layout of Processing Elements

2.4 Example Run Through

To better illustrate the Map-Reduce process I will give an example. In this example the framework will be used to sort lists of numbers. The numbers are given a list of pairs where each pair gives the which list it belongs as a key and a list of unordered numbers as the value.

2.4.1 The Functions

MAP : Sorting Function A sorting function is used by the map function to turn the unordered lists, that are the value parts of the input pairs, into ordered lists. A good sort for this would be a quicksort.

REDUCE : Merge Function A function is used to merge two ordered lists into a single ordered list.

2.4.2 Input Data

The input data is a list of key-value pairs where each pair has an integer key and a list of integers as the value.

A real world example of this would be sports clubs with many players spanning different divisions and bringing their scores all together into one table. Each sports club would put out one key-value pair per division with the division as the key and the scores of the players as a list in the value.

```
(2,(14, 12, 8, 16, 23)) (1,(13, 2, 23, 12, 12)) (1,(15, 12, 18, 7, 17)) (1,(3, 22, 13, 16, 14))
```

Figure 2: Input Data

2.4.3 Splitting to PEs

The first operation that is performed is splitting the input data among the PEs. In this example there are 4 PEs. All the data starts at the master machine and then the data is split into 4 equal subsets. Each subset of the data is sent to a different PE with one subset staying with the master machine. The input data is shown in Figure 3 as being partitioned between the 4 PEs.

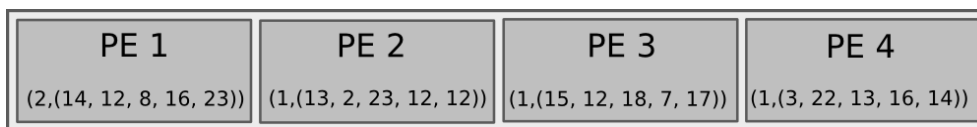


Figure 3: Partitioned Input Data

2.4.4 Mapping

Now that each PE has a set of data to work with they can begin the mapping stage. Each PE takes their set of data and applies the sorting function on each one in turn. The output from this is a set of pairs with the second value being a sorted list. The data after mapping is shown in Figure 4

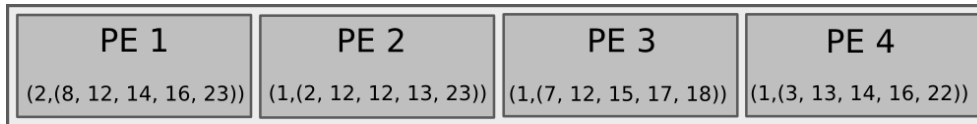


Figure 4: Data After Mapping Operations

2.4.5 Merging

Once all the data has been mapped to new values and as there is only one pair on each PE then data is passed to PEs which will perform reductions on it. As there are 4 PEs in this system then the data will be moved onto two machines where the first reductions will take place. PE 2 sends its data to PE 1, PE 1 must now merge PE 2s data with its own based on the keys of the pairs in the data sets. PE 4 sends its data to PE 3 which merges those two data sets together. Figure 5 shows the merged data across the two PEs still in use.

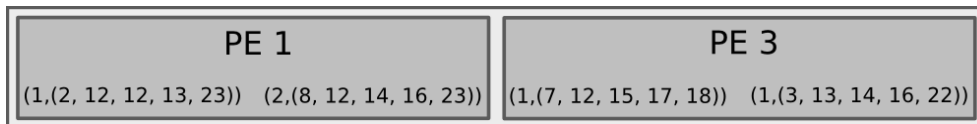


Figure 5: Merged Data Across Two Processing Elements

2.4.6 Reducing

PEs 1 and 3 must now reduce the data that they are storing using the merge function that was supplied to reduce the data. As the two pairs that PE 1 has have got different keys then they will not be reduced together. PE 3 does have two pairs with the same key so they will be reduced. The merge function is performed on the two pairs and the output is a single pair with a sorted list as its second value as shown in Figure 6.

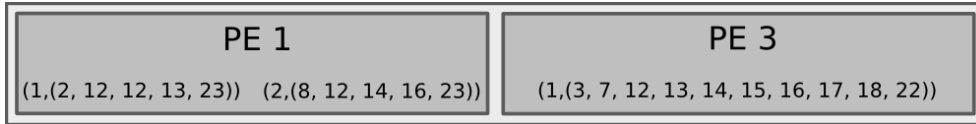


Figure 6: Data After The First Reduction

2.4.7 Merging

As before the data must be collected so that another reduction can be performed upon it. PE 3 passes its data to PE 1 which merges it in with its own to give the list as shown in Figure 7.

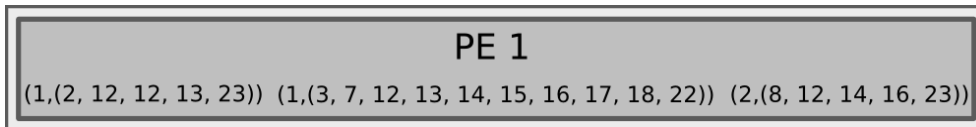


Figure 7: Data Merged Onto A Single Processing Element

2.4.8 Reducing to The Final Output

The final stage in the Map-Reduce is the reduction on the master PE which in this case was PE 1. The merge function is used to reduce the two pairs with key 1 into a single pair. This gives the output of the Map-Reduce framework on this data as the two pairs as shown in Figure 8. In the context of the sports club as given above then the output represents the sorted scores of players in the first and second divisions from a number of clubs.

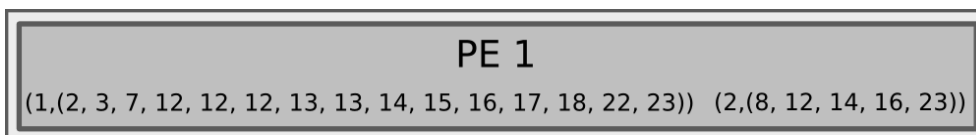


Figure 8: The Final Reduced Output

3 Beowulf Cluster

The Map-Reduce framework that I have to develop will be built and tested on the schools Beowulf cluster. A Beowulf cluster is a small network of computers that are dedicated to solving a problem in parallel [8]. They are typically built on commodity hardware running a Unix based operating system. They normally use libraries such as MPI and PVM (see Section 4.2) to exchange data between each other.

Beowulf clusters are a cheap way of achieving super computer like performance from commodity hardware at a fraction of the cost of a super computer[10]. They are a great way of achieving parallelism even from old machines. Beowulf clusters can have more machines added to them easily meaning that a small cluster can be built upon as the need, or funding, for parallel computing increases.

Beowulfs are used for many high performance computing tasks. Common applications that they are used for include simulations, biotechnology, financial market modeling and data mining[5]. Their cheap set-up and many online resources dedicated to them will help Beowulf clusters find more tasks that they can be used for.

The schools cluster is made up of 32 computers each with a 3GHz Pentium 4 Processor. Each machine also has 512 MB of RAM and they are connected by fast ethernet (100Mbs).

4 Technical Literature

4.1 MapReduce: Simplified Data Processing on Large Clusters

Written by Jeffrey Dean and Sanjay Ghemawat from Google, this paper describes their Map-Reduce framework in detail and provides most of the ideas that this dissertation is based off of. See Section 2.

4.2 Parallel Programming on Linux Networks Using MPI & PVM

Written by Graeme S. McHale[7] this paper proved a good introduction to Message Passing Interface (MPI) and Parallel Virtual Machine (PVM) libraries that programs use to communicate between machines in a cluster.

MPI is a very low level way of dealing with parallel programs. The first MPI function that must be called by a program using the library is `MPI_Init()`. When `MPI_Init()` is called then the MPI library starts up copy's of the program on the other machines in the cluster, starts them all at the `MPI_Init()` call and gives them all task IDs (`tid`).

Most of the other functions in the MPI library are responsible for passing data between the processes. When sending data to another process the function `MPI_Send()` is used, amongst the arguments that it takes are a pointer to the start of the message, the size of the message, the datatype of the message, the `tid` of the destination process, an integer tag that the programmer can use to identify data, or the order of data, and the communication group that the message is to be sent through. Another function used to send data between processes is `MPI_Bcast()` which is used to send a message to all other processes in the communication group.

To receive data the function `MPI_Recv()` is called. When called it will take control of the process and wait until a message is available to be received. The function takes as arguments: a pointer to a buffer where the data received is to be placed, the size of the buffer, the datatype of the buffer, the `tid` of process that sent the message, the tag that is expected, the communicator used to send the message, and a status structure that supplies details of which process sent the message with what tag and also contains a field where an error code can be placed. When receiving messages wildcards can be used instead of the process `tid` and tag.

The final method call to be made in a MPI program is to `MPI_Finalize()`. This function closes down all the copy's of the programs that are running except from the original so that the program returns to a single sequential process.

PVM is used to allow a cluster of computers to be represented as single parallel virtual machine. PVM has a `pvm_spawn()` function that is used to start new tasks on the virtual machine. These tasks can be any executable, not just the same program that called the function. This allows a program to be split into a number of tasks meaning that tasks can be well defined and encapsulated.

Like MPI PVM has different functions for sending data to either one or many machines and the concept of tids and tags is the same for PVM as well. PVM does this slightly differently to MPI though. While using MPI to send data there is only a single function call to be made but PVM uses a few functions. The first function `pvm_initsend()` creates a send buffer where data can then be packed using functions such as `pvm_pklong()` which packs a long value into the buffer. Data is then sent to another task using the `pvm_send()` function, which takes the tid of the task to be sent to and a tag for identification.

The data is received at the other end by using the `pvm_recv()` function. This loads the packed data into a buffer. To unpack the data functions such as `pvm_upkint()` are used to get a pointer to an integer that is packed in the buffer.

In the paper McHale finds that, using his test program, MPI allows for a much greater speedup than PVM when looking at large numbers of processors. For this reason, along with the low level control that MPI allows, I will be using MPI for this dissertation.

4.3 Algorithmic Skeletons: Structured Management of Parallel Computation

Murray Cole's book[2] introduces the ideas of algorithmic skeletons to aid in the parallel implementation of programs.

He first designs three levels of abstraction for parallel skeletons. The first is a high level of abstraction where the user has no idea of the underlying parallelism. This would be an example of a declarative method as the user only declares what should be used to perform the operations, not how the operations should be performed.

The next level of abstraction he calls an idealised level. In this the user is aware of

the parallelism and is responsible for telling the framework which part of the program can be parallelised, but they are still unaware of how the program and data is split up among different machines. Examples of this are PVM and MPI.

The last level of abstraction is the low level abstraction. This is very involved in hardware, the user has to consider the underlying network topology and has to manually map tasks to processors and data transmission channels to physical links. This is much more difficult to use and has to be manually changed in response to changes in the network.

The two of these levels that are the most important to this dissertation are the high and idealised levels. The Map-Reduce framework is a high level skeleton as the user only declares what they want to happen, not how it is to happen. The idealised level is also important as I will be using MPI to implement the framework.

Cole also looks at 4 different parallel skeletons, one of which is a divide and conquer skeleton. This is very similar to the reduce function in the Map-Reduce framework as it first breaks down tasks in to a number of small pieces (the distributed data in Map-Reduce) and then operates on the small tasks, solving those and then using them to solve larger tasks in the same way that the reduce function operates.

4.4 Skeleton Realisations from Functional Prototypes

Written by Greg Michaelson and Norman Scaife[9] this paper explains the use of higher order functions (in particular map and reduce functions) and how they can be used for algorithmic skeletons that can be easily parallelised. It gives good examples of such parallelism using a functional language and shows how functional languages are well suited to parallelised skeletons.

One of main topics of the paper was showing how functional code can be inspected to be adapted to use higher order functions so that they can be easily parallelised. It even discuss the automatic extraction of higher order functions from functional code so that users do not have to design specifically for the skeleton, the skeleton will try and fit the program to itself.

4.5 Bringing Skeletons out of the Closet

Written by Murray Cole[3] this paper sees that there are skeletons that are used in software engineering in the form of object oriented programming and design patterns but that skeletal programming has not had a large effect on mainstream parallel programming. MPI, although operating at a lower level of abstraction than other skeletons, has proved to be very popular. Cole then introduces the Edinburgh Skeleton library and how it means to take parallel skeletons into the mainstream.

He talks about the Edinburgh Skeleton library (eSkel) which is a C library that uses MPI to communicate between machines. The ideas behind eSkel are that it must:

- Be simple to use.
- Be able to mix skeletons with manual parallelisation techniques.
- Be flexible with constraints placed on functions.

The eSkel library uses a farm as a skeleton. A farm is a process or task that takes in a value and outputs another. This is the same as the map function in the MapReduce framework.

4.6 Explicit and Implicit Parallel Functional Programming: Concepts and Implementation

Jost Berthold's thesis[1] covers a parallel implementation of the Eden language (another functional language). In this he describes Google's Map-Reduce system and also another implementation of Google's Map-Reduce by Ralf Lämmal [6]. Lämmal's version of Map-Reduce is slightly different from Google's as he uses the traditional forms of map and reduce but he stops short of an implementation. He does show that the framework would be expected to perform the same with the added ability to have the input to the framework as just a list of elements instead of a list of pairs.

Berthod then implements Lämmal's design in Eden. He also describes the tests that he ran on his implementation of the framework. He provides many results of his tests that I should be able to use to help bench mark my framework against his. This thesis also helps explain the Map-Reduce system in another way that will help me design and build my own framework.

5 Requirements

The main requirement of this dissertation is to implement and evaluate the framework. There are other features that can be added to the framework to improve efficiency and reliability. While some of these ideas will make themselves known through experimenting with early builds of the system, others are already apparent. I will outline these features here.

5.1 Implement A Map-Reduce Framework

Build a Map-Reduce framework in Java that will take two functions (a map function and a reduce function) and a collection of data as arguments. The framework will then distribute the data over the cluster and operate on it using the map and reduce functions as described in Section 2.

5.2 Fault Tolerance

The framework should be fault tolerant. It should be reliable in the sense that it will be able to perform operations as normal even if some of the machines in the cluster cannot be communicated with, or the communication is dropped, as long as the master stays on-line. This means the master will have to know what data each of the machines is working with and will have to redistribute the data to other machines in order for the operation to complete successfully.

In the most extreme case that the master cannot communicate with any other machine then it must be able to perform the functions of the Map-Reduce framework by itself. The other extreme case is that the master fails, in this situation the system will fail as the master is responsible for communicating with user that is using the framework so without the master there is no means of the system giving an output.

6 Testing Strategy

6.1 Test Applications

The testing of the Map-Reduce framework will require special applications to be written to use it. Many ideas for such applications have been suggested by others. I will use some of these ideas here and some of my own as well.

Distributed Grep: This is taken from Google's paper on the Map-Reduce framework [4]. It has the same effect as the grep command found on Unix based systems, it scans through text files and prints any line with content that matches a supplied phrase or pattern.

Distributed Sort: This is the application that was used in the example run through in Section 2.4. It sorts list of lists into a single list by performing a sort as the map function and merging lists together as the reduce function.

Links To A Page: This application can be used to find all the links to a page from a list of links from pages. It uses a function to swap the keys and values in key-value pairs in the map function. The reduce function will concatenate all the values for a given key.

6.2 Data

The Map-Reduce framework is intended to work on large amounts of data, so in-order for and testing to be meaningful a large supply of data will have to be used in test applications. For applications requiring numerical data then random data could be produced for the system to use, either at runtime or previously. Large amounts of meaningful textual data, however, cannot be randomly generated so easily. However it is possible to find large amounts of text data on-line that would be free to use under the circumstances of this project. It is possible to download dumps of Wikipedia (<http://en.wikipedia.org>), Project Gutenberg (<http://www.gutenberg.org/>) or other sites that would provide enough data for the testing of the framework.

6.3 Unit Testing

Unit testing will be used to ensure that each part of the program operates as expected. The unit tests will cover both normal and erroneous inputs to parts of the framework to check both that the expected output for a function is received and that it can handle any errors that could occur.

Unit tests will be drawn up as the framework is designed.

6.4 Fault Tolerance Testing

To test the fault tolerance of the system the user must be able to stop processes or communication from certain machines. This can be done by killing the process of a program from the command line. If the framework manages to successfully adapt to this and gives the same output as it would if the process had not been killed then it passes the test. This would have to be repeated, changing the number of processes killed and when they are killed to make sure that the framework can survive major faults. The master process will not be killed as it is the single point of failure that the system should have.

7 Time Table

Development	Documentation	Week Ending	
Design	Specification	31 st October	
Prototype	Design	7 th November	
Application 1	Draw up Testing	14 th November	
		21 st November	
	Design Document	28 th November	
		5th December	
Refinement	Dissertation	16 th January	
Application 2		23 rd January	
		30 th January	
		6 th February	
Refinement		13 th February	
Application 3		20 th February	
		27 th February	
		6 th March	
Evaluation		13 th March	
		20 th March	
		27 th March	
			3rd April

References

- [1] Jost Berthold. *Explicit and Implicit Parallel Functional Programming: Concepts and Implementation*. PhD thesis, Philipps Universität Marburg, 2008.
- [2] Murray Cole. *Algorithmic Skeletons: Structured Management Of Parallel Computation*. MIT Press, Cambridge, MA, USA, 1991.
- [3] Murray Cole. Bringing Skeletons Out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming. *Parallel Comput.*, 30(3):389–406, 2004.
- [4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified Data Processing on Large Clusters. In *OSDI*, pages 137–150, 2004.
- [5] <http://www.beowulf.org/overview/index.html>.
- [6] Ralf Lämmel. Google’s MapReduce Programming Model — Revisited. *Sci. Comput. Program.*, 68(3):208–237, 2007.
- [7] Graeme S. McHale. Parallel Programming on Linux Networks Using MPI & PVM, 2000.
- [8] Marsha Meredith, Teresa Carrigan, James Brockman, Timothy Cloninger, Jaroslav Privoznik, and Jeffery Williams. Exploring Beowulf clusters. *J. Comput. Small Coll.*, 18(4):268–284, 2003.
- [9] Greg Michaelson and Norman Scaife. Skeleton Realisations from Functional Prototypes. pages 129–153, 2003.
- [10] Stuart Steiner. Building and installing a Beowulf cluster. *J. Comput. Small Coll.*, 17(2):78–87, 2001.