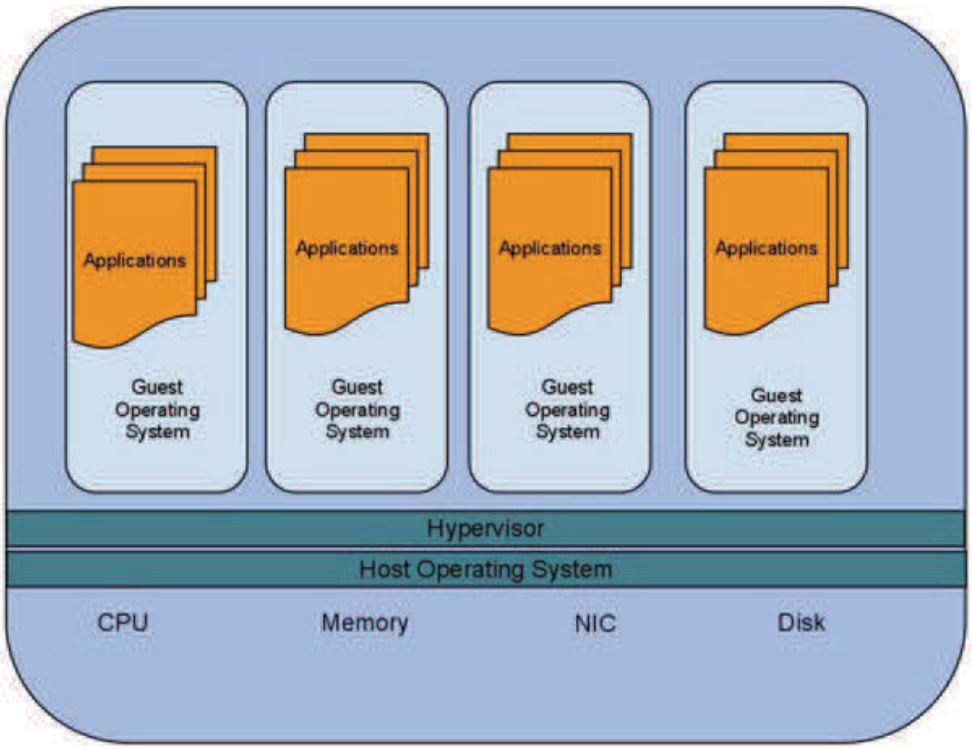

Fig. 4.3 Hosted hypervisor

4.5 MapReduce

MapReduce was originally introduced by Google as a distributed programming model using large server clusters to process massive (multi-terabyte) data sets. The model can be applied to many large-scale computing problems and offers a number of attractive features such as automatic parallelisation, load balancing, network and disk transfer optimisation and robust handling of machine failure. MapReduce works by breaking a large problem into smaller parts, solving each part in parallel and then combining results to produce the final answer (White 2009).

MapReduce runs on top of a specialised file system such as the Google File System (GFS) or the Hadoop File System (HDFS). Data is loaded, partitioned into chunks (commonly 64 MB) such that each chunk can replicate. A key feature of MapReduce is that data processing is collocated with data storage, and as a distributed programming paradigm, a number of advantages are evident when compared to the traditional approach of moving the data to the computation including:

1. Scalability
2. Reliability
3. Fault tolerance
4. Simplicity
5. Efficiency

All these advantages are obviously very relevant to the cloud, where processing large amounts of data using distributed resources is a central task. Unsurprisingly, MapReduce is a programming model used widely in cloud computing environments for processing large data sets in a highly parallel way.

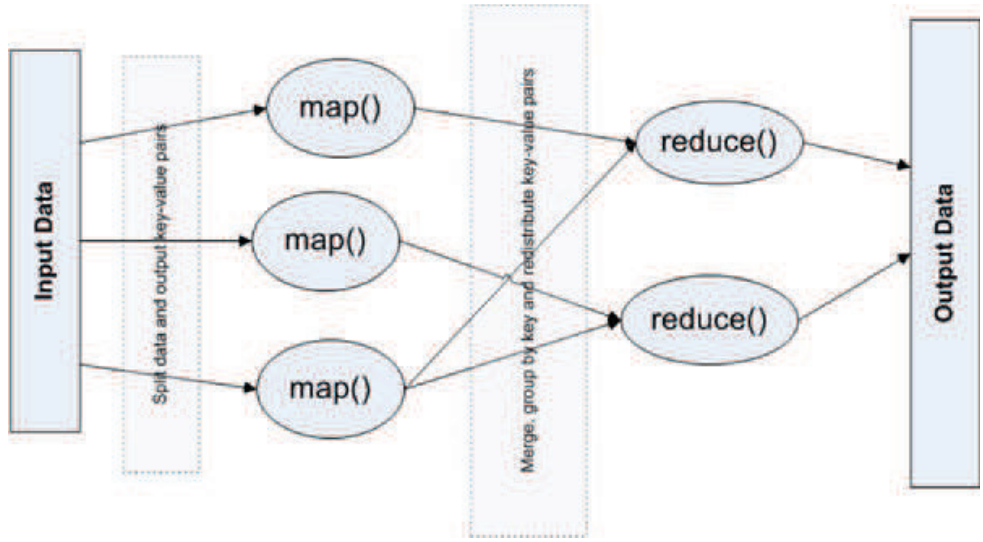

Fig. 4.4 MapReduce overview

MapReduce was inspired by the map and reduce functions which are commonly found in functional program languages like LISP. In LISP, a map takes an input function and a sequence of values and then applies the function to each value in the sequence. Reduce combines all the elements in the sequence using an operator such as * or +. In MapReduce, the functions are not so rigidly defined, but programmers still specify the computation in terms of the two functions map and reduce, which can be carried out on subsets of total data under analysis in parallel. The map function is used to generate a list of intermediate key/value pairs, and the reduce function merges all the intermediate values associated with same intermediate key.

When all the tasks have been completed, the result is returned to the user. In MapReduce, input data is portioned across multiple worker machines executing in parallel; intermediate values are output from the map worker machines and fed to a set of 'reduce' machines (there may be some intermediate steps such as sorting). It is, perhaps, useful to think of MapReduce as representing a data flow as shown in Figure 4.4 rather than a procedure. Jobs are submitted by a user to a master node that selects idle workers and assigns each one a MapReduce task to be performed in parallel. The process of moving map outputs to the reducers is known as 'shuffling'.

4.5.1 MapReduce Example

The canonical MapReduce example takes a document or a set of documents and outputs a listing of unique words and the number of occurrences throughout the text data. The map function takes as its key/value pair the document name and document contents. It then reads through the text of the document and outputs an intermediate key/value listing for each word encountered together with a value of 1. The reduce phase then counts up each of these individual 1's and outputs the total value for each word. The pseudocode for the functions is shown below. The map function takes the name of the document as the key and the contents as the value.

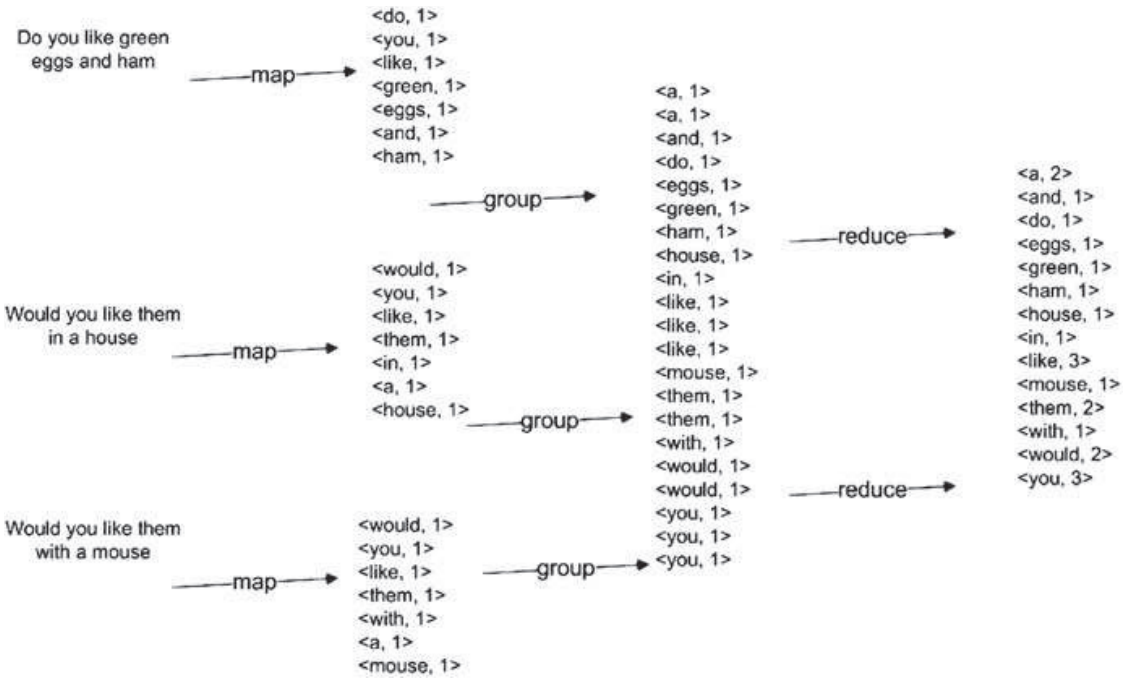

Fig. 4.5 MapReduce word count example

```
map(document_name, document_contents) {
    for each word w in document_contents
      emit_intermediate(w, 1)
}
reduce(a_word, intermediate_vals) {
    result=0
    for each value v in intermediate_vals
      result+= v
    emit result
}
```

Figure 4.5 shows a simple example with the three input files shown on the left and the output word counts shown on the right.

The same kind of process can be applied to many other problems and is particularly useful where we need to process a huge amount of raw data, for example, from documents which have been returned by a web crawl. To create an inverted index (see Chap. 7) after a web crawl, we can create a map function to read each document and output a sequence of <word, documentID> pairs. The reduce function accepts all pairs for a given word and output pairs of <word, documentIDList> such that for any word, we can quickly identify all the documents in which the word occurs. The amount of data may be simply too big for traditional systems and needs to be distributed across hundreds or thousands of machines in order to be processed in a reasonable time frame. Other problems which are well suited to the MapReduce approach include but are not limited to:

1. Searching
2. Classification and clustering

3. Machine learning (Apache Mahout is an open source library for solving machine learning problems with MapReduce)
4. *tf-idf*

We will discuss the above tasks together with web crawling and inverted indexes in Chap. 7.

4.5.2 Scaling with MapReduce

To scale vertically (scale up) refers to the process of adding resources to a single computing resource. For example, we might add more memory, processing power or network throughput which can then be used by virtual machines on that node. MapReduce is designed to work on a distributed file system and uses horizontal scaling (scale out) which refers to the process of achieving scaling by the addition of computing nodes.

The overarching philosophy is often summarised in the adage 'don't move data to workers – move workers to data'; in other words, store the data on the local disks of nodes in the cluster and then use the worker on the node to process its local data. For the kind of problem typical of MapReduce, it is simply not possible to hold all the data in memory. However, throughput can remain reasonable through use of multiple nodes, and reliability is achieved through redundancy. Rather than use expensive 'high end' machines, the approach generally benefits from standard commodity hardware—a philosophy sometimes summarised as 'scale out not up'. In any MapReduce task, coordination is needed, and a 'master' is required to create chunks, balance and replicate and communicate with the nodes.

4.5.3 Server Failure

Where processing occurs on one powerful and expensive machine, we might reasonably expect to run the machine for several years without experiencing hardware failure. However, in a distributed environment using hundreds or thousands of low-cost machines, failures are an expected and frequent occurrence. The creators of MapReduce realised they could combat machine failure simply by replicating jobs across machines. Server failure of a worker is managed by re-executing the task on another worker. Alternatively, several workers can be assigned the same task, and the result is taken from the first one to complete, thus also improving execution time.

4.5.4 Programming Model

The MapReduce model targets a distributed parallel platform with a large number of machines communicating on a network without any explicit shared memory. The programming model is simple and has the advantage that programmers without

expertise in distributed systems are able to create MapReduce tasks. The programmer only needs to supply the two functions, map and reduce, both of which work with key value pairs (often written as <k, v>). Common problems can be solved by writing two or more MapReduce steps which feed into each other.

4.5.5 Apache Hadoop

Hadoop is a popular open-source Java implementation of MapReduce and is used to build cloud environments in a highly fault tolerant manner. Hadoop will process web-scale data of the order of terabytes or petabytes by connecting many commodity computers together to work in parallel. Hadoop includes a complete distributed batch processing infrastructure capable of scaling to hundreds or thousands of computing nodes, with advanced scheduling and monitoring capability. Hadoop is designed to have a 'very flat scalability curve' meaning that once a program is created and tested on a small number of nodes, the same program can then be run on a huge cluster of machines with minimal or no further programming required. Reliable performance growth should then be in proportion to the number of machines available.

The Hadoop File System (HDFS) splits large data files into chunks which are managed by different nodes on the cluster so that each node is operating on a subset of the data. This means that most data is read from the local disk directly into the CPU, thus reducing the need to transfer data across the network and therefore improving performance. Each chunk is also replicated across the cluster so that a single machine failure will not result in data becoming inaccessible.

Fault tolerance is achieved mainly through active monitoring and restarting tasks when necessary. Individual nodes communicate with a master node known as a 'JobTracker'. If a node fails to communicate with the job tracker for a period of time (typically 1 min), the task may be restarted. A system of speculative execution is often employed such that once most tasks have been completed, the remaining tasks are copied across a number of nodes. Once a task has completed, the job tracker is informed, and any other nodes working on the same tasks can be terminated.

4.5.6 A Brief History of Hadoop

Hadoop was created by Doug Cutting (also responsible for Lucene) who named it after his son's toy elephant. Hadoop was originally developed to support distribution of tasks associated with the Nutch web crawler project. In Chap. 7, we will discuss Lucene and Nutch in more detail and will use Nutch to perform a web crawl and create a searchable Lucene index in the end of chapter exercise.

Yahoo was one of the primary developers of Hadoop, but the system is now used by many companies including Facebook, Twitter, Amazon and most recently Microsoft. Hadoop is now a top-level Apache project and benefits from a global community of contributors.

4.5.7 Amazon Elastic MapReduce

Elastic MapReduce runs a hosted Hadoop instance on an EC2 (see Chap. 5) instance master which is able to provision other pre-configured EC2 instances to distribute the MapReduce tasks. Amazon currently allows you to specify up to 20 EC2 instances for data intensive processing.

4.5.8 Mapreduce.NET

Mapreduce.NET is an implementation of MapReduce for the .Net platform which aims to provide support for a wide variety of compute-intensive applications. Mapreduce.Net is designed for the Windows platform and is able to reuse many existing Windows components. An example of Mapreduce.Net in action is found in MRPGA (MapReduce for Parallel GAs) which is an extension of MapReduce specifically for parallelizing genetic algorithms which are widely used in the machine learning community.

4.5.9 Pig and Hive

Pig, originally developed at Yahoo research, is a high-level platform for creating MapReduce programs using a language called 'Pig Latin' which compiles into physical plans which are executed on Hadoop. Pig aims to dramatically reduce the time required for the development of data analysis jobs when compared to creating Hadoops. The creators of Pig describe the language as hitting 'a sweet spot between the declarative style of SQL and the low-level, procedural style of MapReduce'. You will create a small Pig Latin program in the tutorial section.

Apache Hive which runs on Hadoop offers data warehouse services.

4.6 Chapter Summary

In this chapter, we have investigated some of the key technology underlying computing clouds. In particular, we have focused on web application technology, virtualisation and the MapReduce model.

4.7 End of Chapter Exercises

Exercise 1: Create your own VMWare virtual machine

In this exercise, you will create your own virtual machine and install the Ubuntu version of the Linux operating system. In later chapters, we will use this same virtual machine to install more software and complete further exercises. Note that the virtual

machine is quite large and ideally you will have at least 20 GB of free disk space and at least 2 GB of memory. The machine may run with less memory or disk space, but you may find that performance is rather slow.

4.8 A Note on the Technical Exercises (Chaps. 4, 5, 6, 7)

As you will already be aware, the state of the cloud is highly dynamic and rapidly evolving. In these exercises, you are going to download a number of tools and libraries, and we are going to recommend that you choose the latest stable version so that you will always be working at the 'cutting edge'. The downside of this is that we cannot guarantee that following the exercise notes will always work exactly as described as you may well be working with a later version than the one we used for testing. You may need to adapt the notes, refer to information on the source web pages or use web searches to find solutions to any problems or inconsistencies you encounter. The advantage of this is that you will be developing critical skills as you go through the process which would be an essential part of your everyday work should you be involved in building or working with cloud systems. Anyone who has developed software systems will know that it can be very frustrating and at times may feel completely stuck. However, you are often closer to a solution than you might think, and persisting through difficulties to solve a problem can be very rewarding and a huge learning opportunity.

4.9 Create Your Ubuntu VM

(1) Install VMWare Player on your machine: http://www.vmware.com/products/player/ You may need to register with VMWare but VMWare Player is free.
   (a) Click on download and follow the instructions until VM Player is installed.
(2) Go to www.ubuntu.com
   (a) Go to the Ubuntu download page.
   (b) Download the latest version of Ubuntu (e.g. ubuntu-11.10-desktop-i386.iso).
   (c) Save the .iso file to a folder on your machine.
(3) Open VMplayer. Be aware that this process can be quite lengthy.
   (a) Select 'Create New Virtual Machine' (Fig. 4.6).
   (b) Select 'Installer disc image file(iso)' and browse to the folder where you saved your Ubuntu .iso file. Select next.
   (c) Enter a name for your VM and a username and password for the main user on your Linux VM.
   (d) Select a location where your VM will be stored: You will need plenty of disk space (~5 GB).
   (e) You now need to select additional disk space for the VM to use. If you have plenty of space you can go with the recommendation (e.g. 20 GB). It is probably best to select at least 8 GB. Select 'Split virtual disk into 2 GB files'. Select next.

Fig. 4.6 VMware Player

(f) Check the options you have selected and select Finish. If you want to run the VM as soon as it is ready leave the 'Power on this virtual machine after creation' selected.

(4) Your machine will now be created. This will take some time depending on the configuration of your underlying hardware.

(5) You may be asked to select your country so that the keyboard layout can be set appropriately. You can test your keyboard to check it works as expected.

4.10 Getting Started

Your machine is now ready to use.

1. Once your Ubuntu VM has loaded up inside VMWare, log on using the ID and username you gave when creating the VM. VMWare tools should be automatically installed and you may need to restart your machine a number of times. Once complete, you should now be able to maximise the size of your VM screen.
2. Also in the Virtual Machine menu you should see 'power options' where you can 'power off', 'reset' or 'suspend' your virtual machine. 'Suspend' is often a useful option as it will maintain the particular state that your virtual machine is in and restore it to that state next time you 'resume' the virtual machine.
3. If you wish to copy your virtual machine, perhaps to a USB memory stick, you should first suspend or power off the virtual machine. You then need to locate the folder where you created the virtual machine and copy the entire folder to the required destination. You can then start the virtual machine on another computer, providing that VMware Player is installed by locating the file ending with .vmx and double clicking on that file. Note that if something goes wrong later on, you can switch to using this copy. You can of course take a copy of the virtual machine at any point during your development.

4.11 Learn How to Use Ubuntu

In this tutorial and others that follow, we will be using the Ubuntu virtual machine. If you are not familiar with Ubuntu, it will be well worth getting familiar with the basics. There are many guides available, and a good starting point is the Ubuntu home page (http://www.ubuntu.com/ubuntu) where you can take a 'tour' of the important features. In your virtual machine, there is also help available, just click on the 'dash home' icon on the top left and type help and select the help icon (Fig. 4.7).

So take some time to explore the environment and get used to Ubuntu. Actually most of the features are quite intuitive, and you can learn by just 'trying'. In terms of the tutorials in this book, we will only be using a small subset of the available commands and applications. In particular you will need to:

- Be able to browse, copy, move, delete and edit files.
- Use the archive manager to extract zipped files.
- Use the terminal to send basic Unix commands including:
- ls to list the contents of a folder (traditionally referred to as 'Directory' in Unix)
- cd to change directory
- pwd to identify your location in the directory structure

Fig. 4.7 Finding assistance from the Ubuntu home screen

4.12 Install Java

We will be using Java for a number of the tutorial exercises. Java is a freely available programming language. We are going to install the java SDK from Oracle which is suitable for all the tutorial material in this book. These notes are partly based on information from the Ubuntu community at https://help.ubuntu.com/community/Java. If you encounter problems you may want to check the site for any updates.

1. Download the latest Oracle JDK 7 for Linux from http://www.oracle.com/technetwork/java/javase/downloads/java-se-jdk-7-download-432154.html (e.g. jdk-7-linux-i586.tar.gz) in your Ubuntu VM.
2. You will have to agree to Oracle Binary Code License to download this.
3. Locate the file by clicking on the home folder icon and then selecting the 'Downloads' folder (Fig. 4.8).

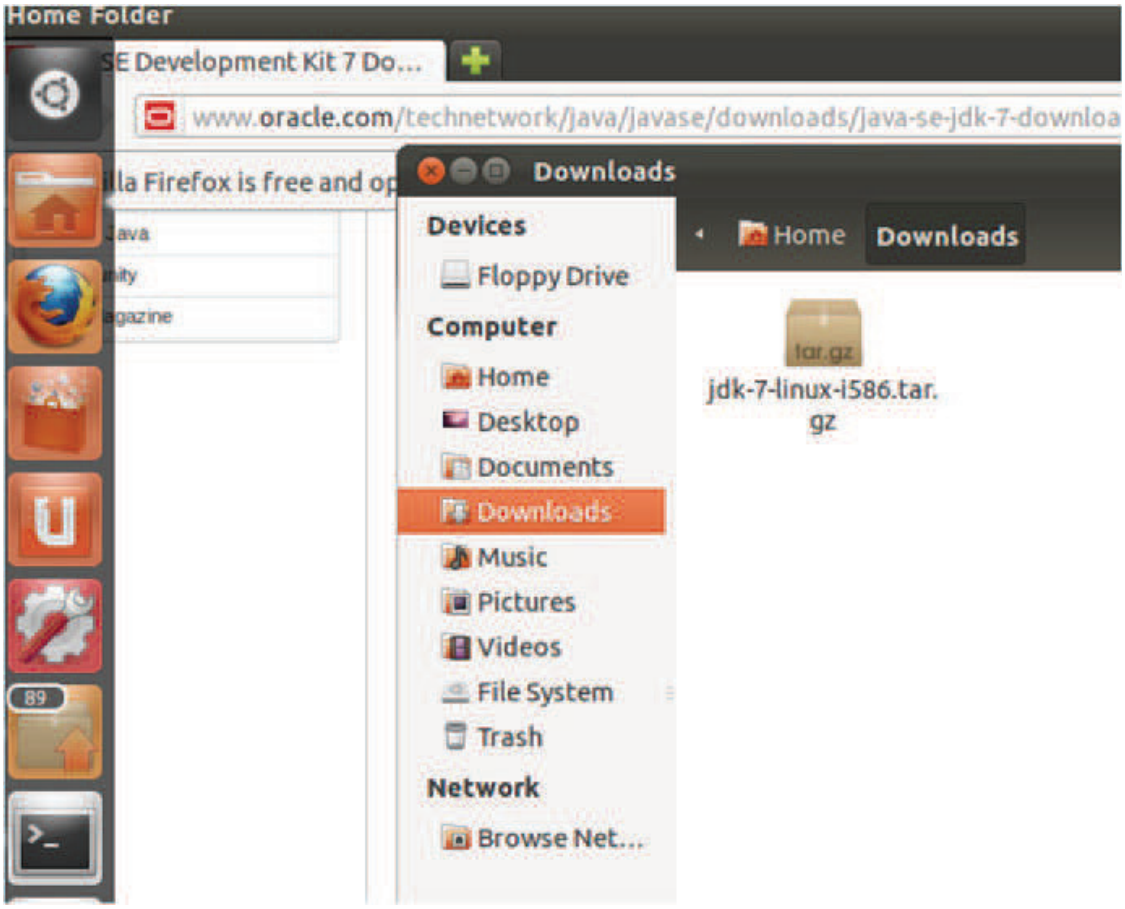

Fig. 4.8 Downloads folder illustrating the Oracle Java Development Kit archive

4. Right click on the file and select 'open with archive manager'.
5. In the archive manager select 'extract' and save. Select 'show the files' once the extraction is complete.
6. A new folder called 'jdk1.7.0' or similar should have been created. Rename this folder to 'java-7-oracle' by right clicking and selecting 'rename'.
7. In your Ubuntu virtual machine open a terminal session. To get a terminal session just click the Ubuntu logo in the side bar and type 'terminal' and then click on the terminal icon (Fig. 4.9). Move to the directory where you extracted the files.
8. Now type the following commands one at a time. Enter your password if prompted and follow on-screen instructions where required. Make sure you type these exactly, including the case.

```
user@ubuntu:~$ sudo mkdir -p/usr/lib/jvm/
user@ubuntu:~$ sudo mv java-7-oracle//usr/lib/jvm/
user@ubuntu:~$sudoadd-apt-repositoryppa:nilarimogard/webupd8
user@ubuntu:~$ sudo apt-get update
user@ubuntu:~$ sudo apt-get install update-java
 user@ubuntu:~$ sudo update-java
```

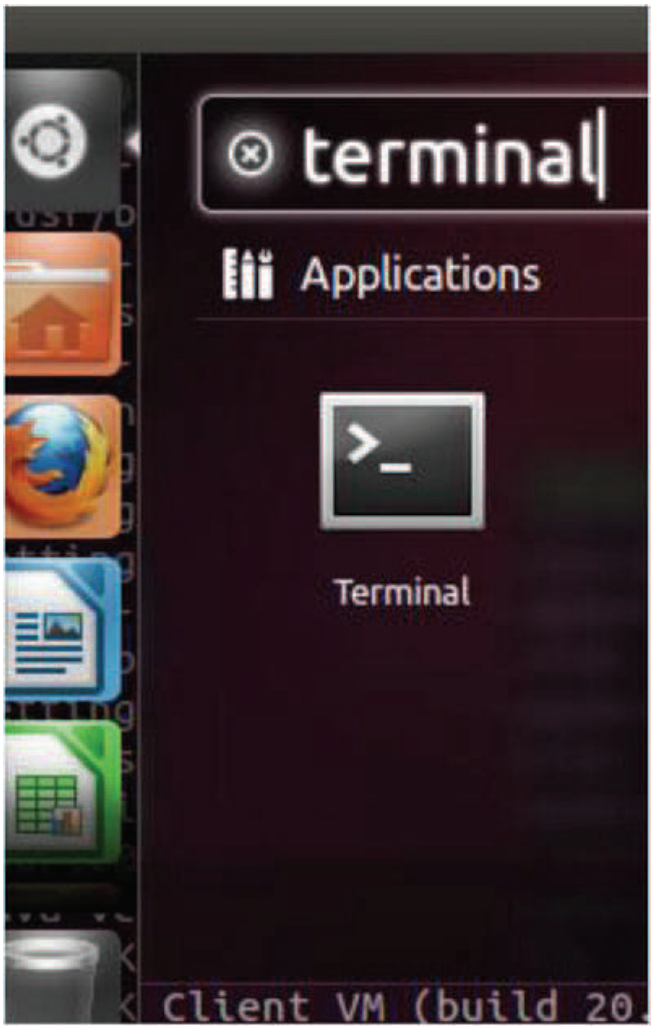

Fig. 4.9 Invoking a terminal session in Ubuntu

9. If prompted, select the Java version you just updated (/usr/lib/jvm/jdk1.7.0).
10. Answer Y if prompted to continue with the Java install.
11. Type the following commands to set the JAVA_HOME

```
user@ubuntu:~$ JAVA_HOME=/usr/lib/jvm/java-7-oracle
user@ubuntu:~$ export JAVA_HOME
user@ubuntu:~$ PATH=$PATH:$JAVA_HOME/bin
user@ubuntu:~$ export PATH
```

4.13 MapReduce with Pig

In this tutorial we are going to implement the classic word count program using Pig Latin.

1. Open your home folder and create a new folder called 'Pig'.
2. In your virtual machine, use a browser such as Firefox to go to http://www.apache.org/dyn/closer.cgi/pig, select a suitable mirror and then download the latest stable release of pig e.g. pig-0.9.2.tar.gz.

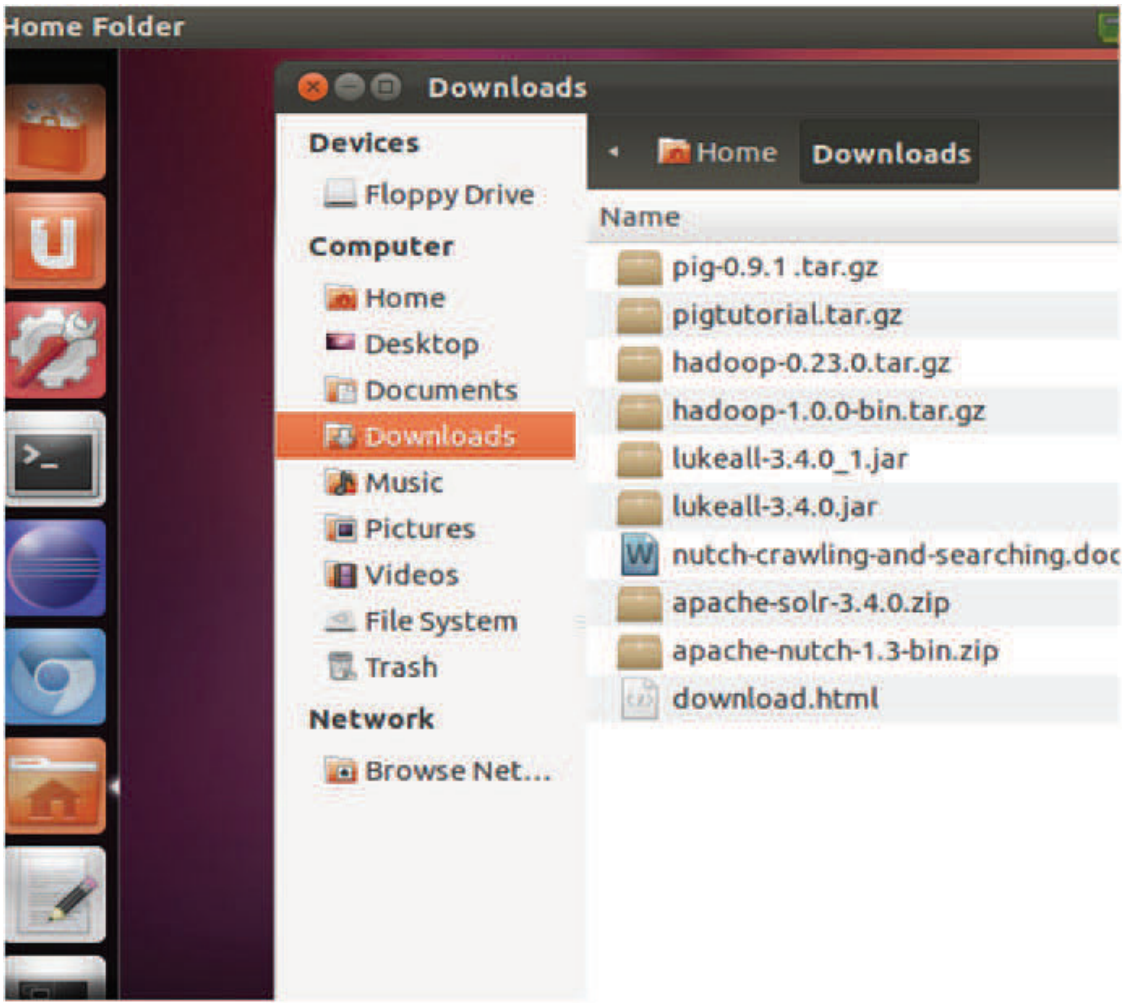

Fig. 4.10 Ubuntu download

3. Open the home folder and then the download folder to see the downloaded pig file (Fig. 4.10).
4. Right click on the file (e.g. pig.0.9.2.tar.gz) and select 'open with archive manager'.
5. In the archive manager select 'extract' and allow the current folder as the destination folder. Select 'show the files' once the extraction is complete.
6. The archive manager will extract the file to a folder called pig-0.9.1 or similar. The only file we need for this tutorial is the .jar file. Locate this file (e.g. pig.0.9.1.jar), right click and select 'copy' and paste the file into the pig folder you created earlier.
7. We need a text file which we can use to count the words. You can use any text file you like but in this example we will copy over the NOTICE.txt file from the extracted folder to the pig folder for testing purposes.
8. We are now going to create our first Pig Latin program. To do this, we first need a text editor so select the 'dash home' icon, type 'text' and select the text editor.
9. Paste in the following code (originally from http://en.wikipedia.org/wiki/Pig_(programming_language)) and save the file as wordCount.pig

```
A=load 'NOTICE.txt';
```

```
   B=foreach A generate flatten(TOKENIZE((chararray)$0))
as word;
   C=filter B by word matches '\\w+';
   D=group C by word;
   E=foreach D generate COUNT(C) as count, group as word;
   F=order E by count desc;
  store F into 'wordcount.txt';
```

10. Open a terminal and navigate to the pig folder.
11. Type in the following command to run your word count program using pig
    `java -Xmx512M -cp pig-0.9.1.jar org.apache.pig.Main -x local wordCount.pig`
12. Once the program has finished running you should see a new folder called wordcount.txt. Navigate into the folder and open the contents of the file. Compare your output with the text file you used and hopefully you will see that the program has been successful and the frequency of the words has been counted and placed in order.
13. Try and run the program with another text file of your choice.
14. Alter the program so that it orders the output by word rather than frequency.
15. Alter the program so that only words starting with a letter above 't' are output.

4.14 Discussion

You might be thinking that actually the program seemed to take quite some time to complete this simple task on a small text file. However, notice that we are running the Pig Latin program using the 'local' mode. This is useful for testing our programs, but if we were really going to use pig for a serious problem such as the results of a web crawl, we would switch to 'hadoop' mode. We could then run the same Pig Latin program, and the task would be automatically parallelized and distributed on a cluster of possibly thousands of nodes. This is actually much simpler than you might expect, especially when vendors such as Amazon offer a set of nodes pre-configured for MapReduce tasks using pig.

4.15 MapReduce with Cloudera

Should you wish to experiment with MapReduce a nice way to start is to visit the Cloudera site (http://www.cloudera.com/) where you can download a virtual machine with Hadoop and Java pre-installed. Cloudera also offer training and support with a range of MapReduce related tasks.

References

Goldne, B.: Virtualization for Dummies. Wiley, Chichester (2008)
Nixon, R.: Learning PHP, MySQL, and JavaScript. O'Reilly Media, Inc, Sebastopol (2009). This is a useful introduction to developing web applications with the popular PHP language
White, T.: Hadoop: The Definitive Guide, 2nd edn. O'Reilly Media, Sebastopol (2009)