Wiley Precise Textbook Series

Big Data Analytics

Radha Shankarmani
M. Vijayalakshmi

WILEY

Big Data Analytics

Big Data Analytics

Dr. Radha Shankarmani
Prof. & HOD, Dept. of Information Technology,
Sardar Patel Institute Of Technology,
Affiliated to Mumbai University,
Andheri–West, Mumbai

Dr. M. Vijayalakshmi
Professor, Department of Information Technology, VESIT
Vivekanand Education Society Institute of Technology,
Affiliated to Mumbai University

WILEY

Big Data Analytics

Other Wiley Editorial Offices:

John Wiley & Sons, Inc. 111 River Street, Hoboken, NJ 07030, USA

Wiley-VCH Verlag GmbH, Pappellaee 3, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 1 Fusionopolis Walk #07-01 Solaris, South Tower, Singapore 138628

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada, M9W 1L1

Edition: 2016

ISBN: 978-81-265-5865-0
ISBN: 978-81-265-8224-2 (ebk)
www.wileyindia.com

Printed at:

Dedicated to my husband, Shankaramani,
and son Rohit who were kind enough to
understand my busy schedule and patiently waited
for a holiday together.

—Radha Shankarmani

Dedicated to my family who steadfastly supported me and
my crazy schedule during the months of writing this book –
my mother-in-law Mrs G. Sharada, my husband G. Murlidhar and
my elder son Goutam. A special dedication to Pranav my long
suffering younger son, whose computer I hijacked to speedup
my writing for the last two months.

—M. Vijayalakshmi

Preface

Importance of Big Data Analytics

The recent explosion of digital data has made organizations to learn more about their businesses, and directly use that knowledge for improved decision making and performance. When shopping moved online, understanding of customers by business managers increased tremendously. E-business not only could track what customers bought, but also track the way they navigated through the site; how much they are influenced by price discounts; review of products bought by their family and friends. The whole look and feel of the contents and its organization in the site is valuable. This information was not available to business managers a decade ago, and the sale-predictions were restricted to the buying pattern of their customers by looking into their past records.

What is New in Analytics?

Analytics can be reactive or proactive. In traditional methods, reactive analytics are done through business intelligence tools and OLAP. For proactive analytics techniques like optimization, predictive modeling, text mining, forecasting and statistical analysis are used. But these tools and techniques cannot be used for Big Data Analytics.

In case of big data, volume, variety and velocity are the three main drivers that gave a new dimension to the way analytics had to be performed. For instance, in Walmart, data collected cross the Internet every hour from its customer transactions is in the range of petabyes. The speed in which data is created is more important than the volume. Real-time or nearly real-time analysis makes a company more agile to face the demand and competitions. For example, a lot of decisions have to be made in real time during the sale on Thanksgiving day. Big data takes the form of messages and images posted to social networks; readings from sensors; GPS signals from cell phones; and more. There is huge volume and variety of information available from social networks, namely Facebook and Twitter. Mobile devices provide streams of data related to people, activities and locations. All the above said data are unstructured and so cannot be stored in structured databases.

To process large amount of unstructured data, technologies and practices were developed. The first step is to fragment such data and store it in a cluster of commodity servers. Here computing is moved to data and not otherwise. The activities of the commodity servers are coordinated by an open-source

software framework called Hadoop. A NoSQL database is used to capture and store the reference data that are diverse in format and also change frequently.

With the advent of Big Data, applying existing traditional data mining algorithms to current real-world problems faces several tough challenges due to the inadequate scalability and other limitations of theses algorithms. The biggest limitation is the inability of these existing algorithms to match the three Vs of the emerging big data. Not only the scale of data generated today is unprecedented, the produced data is often continuously generated in the form of high-dimensional streams which require decisions just in time. Further, these algorithms were not designed to be applicable to current areas like web based analytics, social network analysis, etc.

Thus, even though big data bears greater value (i.e., hidden knowledge and more valuable insights), it brings tremendous challenges to extract these hidden knowledge and insights from big data since the established process of knowledge discovering and data mining from conventional datasets was not designed to and will not work well with big data.

One solution to the problem is to improve existing techniques by applying massive parallel processing architectures and novel distributed storage systems which help faster storage and retrieval of data. But this is not sufficient for mining these new data forms. The true solution lies in designing newer and innovative mining techniques which can handle the three V's effectively.

Intent of the Book

The book focuses on storage and processing of Big Data in the first four chapters and discusses newer mining algorithms for analytics in the rest of the chapters.

The first four chapter focus on the business drivers for Big Data Analytics, Hadoop distributed file system, Hadoop framework and Hadoop eco-systems. The four main architectural patterns for storing Big Data and its variations are also discussed.

The latter chapters cover extensions and innovations to traditional data mining like clustering and frequent itemset mining. The book further looks at newer data forms like web-based data, social network data and discusses algorithms to effectively mine them. One popular practical application of Big Data Analytics, Recommendations systems, is also studied in great detail.

Journey of the Reader

The book takes through the theoretical and practical approach to teaching readers the various concepts of Big Data management and analytics. Readers are expected to be aware of traditional classification, clustering and frequent pattern mining based algorithms. The laboratory exercises are given at the end of relevant chapters will help them to perform laboratory exercises. The readers are expected to have knowledge of database management and data mining concepts. The review questions and/or exercises

given at the end of the chapters can be used to test the readers understanding of the content provided in the chapter. Further, a list of suggested programming assignments can be used by a mature reader to gain expertise in this field.

Organization of the Book

1. Chapter 1 contains introduction to Big Data, Big Data Characteristics, Types of Big Data, comparison of Traditional and Big Data Business Approach, Case Study of Big Data Solutions.
2. Chapter 2 contains introduction to Hadoop, Core Hadoop Components, Hadoop Ecosystem, Physical Architecture, and Hadoop Limitations.
3. Chapter 3 discusses about No SQL, NoSQL Business Drivers, Case Studies on NoSQL, No SQL Data Architecture Patterns, Variations of NoSQL Architectural Patterns, Using NoSQL to Manage Big Data, Understanding Types of Big Data Problems, Analyzing Big Data with a Shared-Nothing Architecture, Choosing Distribution Models, Master–Slave vs Peer-to-Peer, the way NoSQL System Handles Big Data Problems.
4. Chapter 4 covers MapReduce and Distributed File Systems; Map Reduce: The Map Tasks and The Reduce Tasks; MapReduce Execution, Coping with Node Failures, Algorithms Using MapReduce: Matrix-Vector Multiplication and Relational Algebra operations.
5. Chapter 5 introduces the concept of similarity between items in a large dataset which is the foundation for several big data mining algorithms like clustering and frequent itemset mining. Different measures are introduced so that the reader can apply the appropriate distance measure to the given application.
6. Chapter 6 introduces the concept of a data stream and the challenges it poses. The chapter looks at a generic model for a stream-based management system. Several Sampling and Filtering techniques which form the heart of any stream mining technique are discussed; among them the popularly used is Bloom filter. Several popular steam-based algorithms like Counting Distinct Elements in a Stream, Counting Ones in a Window, Query Processing in a Stream are discussed.
7. Chapter 7 introduces the concept of looking at the web in the form of a huge webgraph. This chapter discusses the ill effects of "Spam" and looks at Link analysis as a way to combat text bead "Spam". The chapter discusses Google's PageRank algorithm and its variants in detail. The alternate ranking algorithm HITS is also discussed. A brief overview of Link spam and techniques to overcome them are also provided.
8. Chapter 8 covers very comprehensively algorithms for Frequent Itemset Mining which is at the heart of any analytics effort. The chapter reviews basic concepts and discusses improvements to the popular A-priori algorithm to make it more efficient. Several newer big data frequent itemset mining algorithms like PCY, Multihash, Multistage algorithms are discussed. Sampling-based

algorithms are also dealt with. The chapter concludes with a brief overview of identifying frequent itemsets in a data stream.

9. Chapter 9 covers clustering which is another important data mining technique. Traditional clustering algorithms like partition-based and hierarchical are insufficient to handle the challenges posed by Big Data clustering. This chapter discusses two newer algorithms, BFR and CURE, which can cluster big data effectively. The chapter provides a brief overview of stream clustering.
10. Chapter 10 discusses Recommendation Systems, A Model for Recommendation Systems, Content-Based Recommendations and Collaborative Filtering.
11. Chapter 11 introduces the social network and enumerates different types of networks and their applications. The concept of representing a Social Network as a Graph is introduced. Algorithms for identifying communities in a social graph and counting triangles in a social graph are discussed. The chapter introduces the concept of SimRank to identify similar entities in a social network.
12. **Appendix:** This book also provides a rather comprehensive list of websites which contain open datasets that the reader can use to understand the concept and use in their research on Big Data Analytics.
13. Additionally each chapter provides several exercises based on the chapters and also several programming assignments that can be used to demonstrate the concepts discussed in the chapters.
14. References are given for detail reading of the concepts in most of the chapters.

Audience

This book can be used to teach a first course on Big Data Analytics in any senior undergraduate or graduate course in any field of Computer Science or Information Technology. Further it can also be used by practitioners and researchers as a single source of Big Data Information.

Acknowledgements

First and foremost, I would like to thank my mother for standing beside me throughout my career and writing this book. My sincere thanks to Principal, Dr. Prachi Gharpure, too. She has been my inspiration and motivation for continuing to improve my knowledge and move my career forward. My thanks to M.E. research students in writing installation procedures for laboratory exercises.

Radha Shankarmani

Several people deserve my gratitude for their help and guidance in making this book a reality. Foremost among them is Prof. Radha Shankaramani, my co-author who pushed and motivated me to start this venture. My sincere thanks to my principal Dr. J.M. Nair (VESIT) who has supported me full heartedly in this venture. My thanks to Amey Patankar and Raman Kandpal of Wiley India for mooting the idea of this book in the first place.

M. Vijayalakshmi

Together,

We would like to express our gratitude to the many people who inspired us and provided support.

Our sincere thanks to the Dean, Ad hoc Board of Studies, Information Technology, Dr. Bakal for introducing the course in under graduate program and providing us an opportunity to take this venture. Our sincere thanks to the publishers, Wiley India and the editorial team for their continuing support in publishing this book.

Radha Shankarmani

M. Vijayalakshmi

About the Authors

Dr. Radha Shankarmani is currently working as Professor and Head at Department of Information Technology, Sardar Patel Institute of Technology, Mumbai. Her areas of interest include Business Intelligence, Software Engineering, Software Testing, Databases, Data Warehousing and Mining, Computer Simulation and Modeling, Management Information System and SOA. Dr. Radha Shankarmani holds a PhD degree from JNTUH; Masters degree in Computer Science and Engineering from NIT, Trichy and Bachelors degree from PSG College of Technology in Electronics and Communication Engineering. She has more than 20 years of teaching experience and 4 years of industry experience where she has held designations such as Programmer, Software Engineer and Manager. She did her sabbaticals for two months in Infosys, Pune in 2005 and has published a number of papers in National, International conferences and International journal.

Dr. M. Vijayalakshmi is Professor of Information Technology at VES Institute of Technology Mumbai. Currently she is also the Vice Principal of the college. She has more than 25 years of teaching experience both at undergraduate and postgraduate engineering level. Dr. M. Vijayalakshmi holds a Master of Technology and Doctorate Degree in Computer Science and Engineering from the Indian Institute of Technology Mumbai, India. During her career at VESIT, she has served on syllabus board of Mumbai University for BE of Computer Science and Information Technology departments. She has made several contributions to conferences, national and international in the field of Data Mining, Big Data Analytics and has conducted several workshops on data mining related fields. Her areas of research include Databases, Data Mining, Business Intelligence and designing new algorithms for Big Data Analytics.

Contents

HDFS stores large files in the range of gigabytes to terabytes across multiple machines. It achieves reliability by replicating the data across multiple hosts. Data is replicated on three nodes: two on the same rack and one on a different rack. Data nodes can communicate with each other to re-balance data and to move copies around. HDFS is not fully POSIX-compliant to achieve increased performance for data throughput and support for non-POSIX operations such as Append.

The HDFS file system includes a so-called secondary NameNode, which regularly connects with the primary NameNode and builds snapshots of the primary NameNode directory information, which the system then saves to local or remote directories. These check-pointed images can be used to restart a failed primary NameNode without having to replay the entire journal of file-system actions, then to edit the log to create an up-to-date directory structure.

An advantage of using HDFS is data awareness between the JobTracker and TaskTracker. The JobTracker schedules map or reduce jobs to TaskTrackers with an awareness of the data location. For example, if node *A* contains data (*x, y, z*) and node *B* contains data (*a, b, c*), the JobTracker schedules node *B* to perform map or reduce tasks on (*a,b,c*) and node *A* would be scheduled to perform map or reduce tasks on (*x,y,z*). This reduces the amount of traffic that goes over the network and prevents unnecessary data transfer.

When Hadoop is used with other file system, this advantage is not always available. This can have a significant impact on job-completion times, which has been demonstrated when running data-intensive jobs. HDFS was designed for mostly immutable files and may not be suitable for systems requiring concurrent write-operations.

2.6 Hadoop Limitations

HDFS cannot be mounted directly by an existing operating system. Getting data into and out of the HDFS file system can be inconvenient. In Linux and other Unix systems, a file system in Userspace (FUSE) virtual file system is developed to address this problem.

File access can be achieved through the native Java API, to generate a client in the language of the users' choice (C++, Java, Python, PHP, Ruby, etc.), in the command-line interface or browsed through the HDFS-UI web app over HTTP.

2.6.1 Security Concerns

Hadoop security model is disabled by default due to sheer complexity. Whoever's managing the platform should know how to enable it; else data could be at huge risk. Hadoop does not provide encryption at the storage and network levels, which is a major reason for the government agencies and others not to prefer to keep their data in Hadoop framework.

2.6.2 Vulnerable By Nature

Hadoop framework is written almost entirely in Java, one of the most widely used programming languages by cyber-criminals. For this reason, several experts have suggested dumping it in favor of safer, more efficient alternatives.

2.6.3 Not Fit for Small Data

While big data is not exclusively made for big businesses, not all big data platforms are suitable for handling small files. Due to its high capacity design, the HDFS lacks the ability to efficiently support the random reading of small files. As a result, it is not recommended for organizations with small quantities of data.

2.6.4 Potential Stability Issues

Hadoop is an open-source platform necessarily created by the contributions of many developers who continue to work on the project. While improvements are constantly being made, like all open-source software, Hadoop has stability issues. To avoid these issues, organizations are strongly recommended to make sure they are running the latest stable version or run it under a third-party vendor equipped to handle such problems.

2.6.5 General Limitations

Google mentions in its article that Hadoop may not be the only answer for big data. Google has its own Cloud Dataflow as a possible solution. The main point the article stresses is that companies could be missing out on many other benefits by using Hadoop alone.

Summary

- MapReduce brings compute to the data in contrast to traditional distributed system, which brings data to the compute resources.
- Hadoop stores data in a replicated and distributed way on HDFS. HDFS stores files in chunks which are physically stored on multiple compute nodes.
- MapReduce is ideal for operating on very large, unstructured datasets when aggregation across large datasets is required and this is accomplished by using the power of Reducers.
- Hadoop jobs go through a map stage and a reduce stage where
  - the mapper transforms the input data into key–value pairs where multiple values for the same key may occur.
  - the reducer transforms all of the key–value pairs sharing a common key into a single key–value.
- There are specialized services that form the Hadoop ecosystem to complement the Hadoop modules. These are HBase, Hive, Pig, Sqoop, Mahout, Oozie, Spark, Ambari to name a few.

Review Questions

1. What is Hadoop?
2. Why do we need Hadoop?
3. What are core components of Hadoop framework?
4. Give a brief overview of Hadoop.
5. What is the basic difference between traditional RDBMS and Hadoop?
6. What is structured, semi-structured and unstructured data? Give an example and explain.
7. What is HDFS and what are its features?
8. What is Fault Tolerance?
9. If replication causes data redundancy, then why is it pursued in HDFS?
10. What is a NameNode?
11. What is a DataNode?
12. Why is HDFS more suited for applications having large datasets and not when there are small files?
13. What is a JobTracker?
14. What is a TaskTracker?
15. What is a "block" in HDFS?
16. What are the benefits of block transfer?
17. What is a secondary NameNode? Does it substitute a NameNode?
18. What is MapReduce? Explain how do "map" and "reduce" work?
19. What sorts of actions does the JobTracker process perform?

Laboratory Exercise

A. Instructions to learners on how to run wordcount program on Cloudera

To start working with Hadoop for beginners, the best choice is to download ClouderaVM from their official website (and it is free).

Download Link:
http://www.cloudera.com/content/cloudera/en/downloads/quickstart_vms/cdh-5-4-x.html

Pre-requisite: Install VM Ware Player or Oracle Virtual Box.
For the above version of Cloudera we need virtual box.

INSTALLING AND OPENNING Cloudera in VIRTUAL BOX

STEP1: EXTRACT the downloaded zip file in the same folder or in home directory.

STEP2: Open virtualbox. Then

- Click New button which is at toolbar menu.
- A new window will open. Type name in the Name field, for example, "Cloudera". Next in Type field select the type as "Linux". In the version field select "Other Linux(64 bit)".

- Click on Next Button. A new window will open. Select the RAM size. Click on Next Button.
- Here you have three options, out of which select "use an existing virtual Hard drive file". Browse your Cloudera folder for file with .vmdk extension. Select that file and press ENTER.

Now as we have successfully created vm we can start Cloudera. So start it by clicking on start button. It will take some time to open. Wait for 2 to 3 minutes. Here the operating system is CentOS.

Once the system gets loaded we will start with the simple program called "wordcount" using MapReduce function which is a simple "hello world" kind of program for Hadoop.

STEPS FOR RUNNING WORDCOUNT PROGRAM:

1. OPEN the Terminal. Install a package "wget" by typing the following command:

   $ sudo yum -y install wget

2. Make directory:

   $ mkdir temp

3. Goto temp:

   $cd temp

4. Create a file with some content in it:

   $ echo "This is SPIT and you can call me Sushil. I am good at statistical modeling and data analysis" > wordcount.txt

5. Make input directory in the HDFS system:

   $ hdfsdfs -mkdir /user/cloudera/input

6. Copy file from local directory to HDFS file system:

   $ hdfsdfs -put /home/cloudera/temp/wordcount.txt /user/cloudera/input/

7. To check if your file is successfully copied or not, use:

   $ hdfsdfs -ls /user/cloudera/input/

8. To check hadoop-mapreduce-examples, use:

   $ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar

9. Run the wordcount program by typing the following command:

   $ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount /user/cloudera/input/wordcount.txt /user/cloudera/output

   Note: The output will be generated in the output directory in HDFS file system and stored in part file "part-r-00000".

10. Check output directory:

$hdfsdfs -ls /user/cloudera/output

11. Open the part file to check the output:

$ hdfsdfs -cat /user/cloudera/output/part-r-00000

Note: The above command will display the file content on the terminal. If you want to open it in the text editor then we need to copy it to our local file system. To do so, use the following command:

$ hdfsdfs -copyToLocal /user/cloudera/output/part-r-00000 /home/cloudera

B. Guidelines to Install Hadoop 2.5.2 on top of Ubuntu 14.04 and write WordCount Program in Java using MapReduce structure and test it over HDFS

Pre-requisite: Apache, JAVA, ssh packages must be installed. If not then follow the following steps.

B1. Steps for Installing Above Packages

1. Before installing above packages, Create a new user to run the Hadoop (hduser or huser) and give it sudo rights:

- Create group name hadoop:
  $ sudoaddgrouphadoop
- To create user and add it in group named Hadoop use
  $ sudoadduser --ingrouphadoophduser
- To give sudo rights to hduser use
  $ sudoadduserhdusersudo
- To switch user to hduser use
  $ suhduser

2. Install the following software:

Update the source list

$ sudo apt-get update

2.1 Apache

$ sudo apt-get install apache2

The OpenJDK project is the default version of Java.

It is provided from a supported Ubuntu repository.

2.2 Java

$ sudo apt-get install default-jdk

$ java -version

2.3 Installing SSH: ssh has two main components, namely,

- ssh: The command we use to connect to remote machines – the client.
- sshd: The daemon that is running on the server and allows clients to connect to the server.

The ssh is pre-enabled on Linux, but in order to start sshd daemon, we need to install ssh first. Use the following command to do so:

$ sudo apt-get install ssh

This will install ssh on our machine. Verify if ssh is installed properly with which command:

$ whichssh
o/p:usr/bin/ssh

$ whichsshd
o/p:/usr/sbin/sshd

Create and Setup SSH Certificates: Hadoop requires SSH access to manage its nodes, that is, remote machines plus our local machine. For our single-node setup of Hadoop, we therefore need to configure SSH access to local host. So, we need to have SSH up and running on our machine and configured to allow SSH public key authentication. Hadoop uses SSH (to access its nodes) which would normally require the user to enter a password. However, this requirement can be eliminated by creating and setting up SSH certificates using the following commands. If asked for a filename just leave it blank and press the enter key to continue.

$ ssh-keygen -t rsa -P “”

Note: After typing the above command just press Enter two times.

$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys

The second command adds the newly created key to the list of authorized keys so that Hadoop can use ssh without prompting for a password.

We can check if ssh works using the following command:

$ ssh localhost

o/p:

The authenticity of host ‘localhost (127.0.0.1)’ cannot be established.

ECDSA key fingerprint is e1:8b:a0:a5:75:ef:f4:b4:5e:a9:ed:be:64:be:5c:2f.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added ‘localhost’ (ECDSA) to the list of known hosts.

Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-40-generic x86_64)

B2. Installing Hadoop

1. Download and extract the hadoop-2.5.2.tar.gz to the Downloads directory from the link given below:
   https://archive.apache.org/dist/hadoop/core/hadoop-2.5.2/
2. To switch user to hduser use
   $ sudosuhduser
3. To move hadoop to/usr/local/Hadoop use
   $ sudo mv /home/admin/Downloads/* /usr/local/hadoop
4. To change the access rights use
   sudochown -R hduser:hadoop /usr/local/hadoop

B3. Setup Configuration Files

The following files will have to be modified to complete the Hadoop setup:
~/.bashrc
/usr/local/hadoop/etc/hadoop/hadoop-env.sh
/usr/local/hadoop/etc/hadoop/core-site.xml
/usr/local/hadoop/etc/hadoop/mapred-site.xml.template
/usr/local/hadoop/etc/hadoop/hdfs-site.xml

1. ~/.bashrc: Before editing the .bashrc file in our home directory, we need to find the path where Java has been installed to set the JAVA_HOME environment variable using the following command:

   $ update-alternatives --config java

   Now we can append the following to the end of ~/.bashrc:

   $ vi ~/.bashrc

```
#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS=”-Djava.library.path=$HADOOP_INSTALL/lib”
#HADOOP VARIABLES END
```

$ source ~/.bashrc

Note that the JAVA_HOME should be set as the path just before the '.../bin/':

$ javac -version
$ whichjavac
$ readlink -f /usr/bin/javac

2. /usr/local/hadoop/etc/hadoop/hadoop-env.sh: We need to set JAVA_HOME by modifying hadoop-env.sh file.

   $ vi /usr/local/hadoop/etc/hadoop/hadoop-env.sh

   Add the following configuration:

   export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

3. /usr/local/hadoop/etc/hadoop/core-site.xml: This file contains configuration properties that Hadoop uses when starting up.This file can be used to override the default settings that Hadoop starts with.

   $ sudomkdir -p /app/hadoop/tmp

   $ sudochownhduser:hadoop /app/hadoop/tmp

   Open the file and enter the following in between the <configuration></configuration> tag:

```
$ vi /usr/local/hadoop/etc/hadoop/core-site.xml
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>A base for other temporary directories.</description>
</property>

<property>
<name>fs.default.name</name>
<value>hdfs://localhost:54310</value>
<description>The name of the default file system. A URI whose
scheme and authority determine the FileSystem implementation. The
uri's scheme determines the config property (fs.SCHEME.impl) naming
the FileSystem implementation class. The uri's authority is used to
determine the host, port, etc. for a filesystem.</description>
</property>
</configuration>
```

4. /usr/local/hadoop/etc/hadoop/mapred-site.xml: By default, the /usr/local/hadoop/etc/hadoop/ folder contains /usr/local/hadoop/etc/hadoop/mapred-site.xml.template file which has to be renamed/copied with the name mapred-site.xml.

   $ cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml

   The mapred-site.xml file is used to specify which framework is being used for MapReduce.

   We need to enter the following content in between the <configuration></configuration> tag:

   ```
   <configuration>
   <property>
   <name>mapred.job.tracker</name>
   <value>localhost:54311</value>
   <description>The host and port that the MapReduce job tracker runs
   at. If "local", then jobs are run in-process as a single map
   and reduce task.
   </description>
   </property>
   </configuration>
   ```

5. /usr/local/hadoop/etc/hadoop/hdfs-site.xml: This file needs to be configured for each host in the cluster that is being used. It is used to specify the directories which will be used as the NameNode and the DataNode on that host. Before editing this file, we need to create two directories which will contain the NameNode and the DataNode for this Hadoop installation. This can be done using the following commands:

   $ sudomkdir -p /usr/local/hadoop_store/hdfs/namenode

   $ sudomkdir -p /usr/local/hadoop_store/hdfs/datanode

   $ sudochown -R hduser:hadoop /usr/local/hadoop_store

   Open the file and enter the following content in between the <configuration></configuration> tag:

   $ vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml

   ```
   <configuration>
   <property>
   <name>dfs.replication</name>
   <value>1</value>
   <description>Default block replication.
   The actual number of replications can be specified when the file is created.
   The default is used if replication is not specified in create time.
   ```

```
</description>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/usr/local/hadoop_store/hdfs/namenode</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file:/usr/local/hadoop_store/hdfs/datanode</value>
</property>
</configuration>
```

B4. Format the New Hadoop File System: Now, the Hadoop file system needs to be formatted so that we can start to use it. The format command should be issued with write permission since it creates current directory:

under /usr/local/hadoop_store/hdfs/namenode folder:

$ hadoopnamenode -format

Note that hadoopnamenode -format command should be executed once before we start using Hadoop. If this command is executed again after Hadoop has been used, it will destroy all the data on the Hadoop file system.

Starting Hadoop: Now it is time to start the newly installed single node cluster. We can use start-all.sh or (start-dfs.sh and start-yarn.sh)

$ cd /usr/local/hadoop/sbin

$ start-all.sh

We can check if it is really up and running using the following command:

$ jps

o/p:

9026 NodeManager
7348 NameNode
9766 Jps
8887 ResourceManager
7507 DataNode

The output means that we now have a functional instance of Hadoop running on our VPS (Virtual private server).

$ netstat -plten | grep java

Stopping Hadoop

$ cd /usr/local/hadoop/sbin

$ stop-all.sh

B5. Running Wordcount on Hadoop 2.5.2: Wordcount is the hello_world program for MapReduce. Code for wordcount program is:

```
packageorg.myorg;
importjava.io.IOException;
importjava.util.*;
importorg.apache.hadoop.fs.Path;
importorg.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
importorg.apache.hadoop.util.*;
importorg.apache.hadoop.mapreduce.Mapper;
importorg.apache.hadoop.mapreduce.Reducer;
importorg.apache.hadoop.conf.Configuration;
importorg.apache.hadoop.conf.Configured;
importorg.apache.hadoop.mapreduce.Job;
importorg.apache.hadoop.mapreduce.lib.input.FileInputFormat;
importorg.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
importorg.apache.hadoop.mapred.JobConf;
public class myWordCount {
public static class Map extends Mapper
<LongWritable, Text, Text, IntWritable> {
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
public void map(LongWritable key, Text value, Context context) throws IOException, Inter-
ruptedException {
            String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
word.set(tokenizer.nextToken());
context.write(word, one);
            }
        }
    }
public static class Reduce extends Reducer
<Text, IntWritable, Text, IntWritable> {
public void reduce(Text key, Iterator<IntWritable> values, Context context) throws IOExcep-
tion, InterruptedException {
```

```
int sum = 0;
while (values.hasNext()) {
sum += values.next().get();
            }
context.write(key, new IntWritable(sum));
        }
   }
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
conf.set("mapreduce.job.queuename", "apg_p7");
System.out.println("This is a new version");
        Job job = new Job(conf);
job.setJarByClass(myWordCount.class);
job.setJobName("myWordCount");
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setMapperClass(myWordCount.Map.class);
job.setCombinerClass(myWordCount.Reduce.class);
job.setReducerClass(myWordCount.Reduce.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.waitForCompletion(true);
    }
}
```

Note: Copy the above code and save it with .java extension. To run the program under MapReduce, the following steps needs to be done:

1. Put source code under this location

   /project/src/org/myorg/myWordCount.java

2. Compile java code

   ```
   $ mkdir /project/class;
   $ cd /project;
   $ javac -classpath `yarn classpath` -d ./class ./src/org/myorg/*.java
   ```

3. Create manifest.txt file

   ```
   $ cd project/class;
   $ sudo vim manifest.txt;
   ```
   The content of manifest.txt is
   Main-Class: org.myorg.myWordCount
   Leave an empty line at the end of manifest.txt

4. To Generate jar file

   $ jar -cvmf manifest.txt myWordCount.jar org

5. Put input data on HDFS

   $ mkdir input
   $ echo “hadoop is fast and hadoop is amazing, hadoop is new Technology for Big Data Processing” > input/file1
   $ hadoop fs -put input /user/hadoop

6. Run the program

   hadoop jar myWordCount.jar /user/hadoop/input /user/hadoop/output

3 What is NoSQL?

LEARNING OBJECTIVES

After reading this chapter, you will be able to:

- Understand NoSQL business drivers.
- Learn the desirable features of NoSQL that drive business.
- Learn the need for NoSQL through case studies.
- Learn NoSQL data architectural pattern.
- Learn the variations in NoSQL architectural pattern.
- Learn how NoSQL is used to manage big data.
- Learn how NoSQL system handles big data problems.

3.1 What is NoSQL?

NoSQL is database management system that provides mechanism for storage and retrieval of massive amount of unstructured data in a distributed environment on virtual servers with the focus to provide high scalability, performance, availability and agility.

In other words, NoSQL was developed in response to a large volume of data stored about users, objects and products that need to be frequently accessed and processed. Relational databases are not designed to scale and change easily to cope up with the needs of the modern industry. Also they do not take advantage of the cheap storage and processing power available today by using commodity hardware.

NoSQL database is also referred as **N**ot **o**nly **SQL**. Most NoSQL systems are entirely non-relational; they do not have fixed schemas or JOIN operations. Instead they use objects, key-value pairs, or tuples.

Some of the NoSQL implementations are SimpleDB, Google BigTable, Apache Hadoop, MapReduce and MemcacheDB. There are approximately 150 NoSQL databases available in the market. Companies that largely use NoSQL are NetFlix, LinkedIn and Twitter for analyzing their social network data.

In short:

1. NoSQL is next generation database which is completely different from the traditional database.
2. NoSQL stands for Not only SQL. SQL as well as other query languages can be used with NoSQL databases.

3. NoSQL is non-relational database, and it is schema-free.
4. NoSQL is free of JOINs.
5. NoSQL uses distributed architecture and works on multiple processors to give high performance.
6. NoSQL databases are horizontally scalable.
7. Many open-source NoSQL databases are available.
8. Data file can be easily replicated.
9. NoSQL uses simple API.
10. NoSQL can manage huge amount of data.
11. NoSQL can be implemented on commodity hardware which has separate RAM and disk (shared-nothing concept).

3.1.1 Why NoSQL?

The reality is that a traditional database model does not offer the best solution for all scenarios in applications. A relational database product can cater to a more predictable, structured data. NoSQL is required because today's industry needs a very agile system that can process unstructured and unpredictable data dynamically. NoSQL is known for its high performance with high availability, rich query language, and easy scalability which fits the need. NoSQL may not provide atomicity, consistency, isolation, durability (ACID) properties but guarantees eventual consistency, basically available, soft state (BASE), by having a distributed and fault-tolerant architecture.

3.1.2 CAP Theorem

Consistency, **A**vailability, **P**artition tolerance (**CAP) theorem**, also called as **Brewer's theorem**, states that it is not possible for a distributed system to provide all three of the following guarantees simultaneously:

1. Consistency guarantees all storage and their replicated nodes have the same data at the same time.
2. Availability means every request is guaranteed to receive a success or failure response.
3. Partition tolerance guarantees that the system continues to operate in spite of arbitrary partitioning due to network failures.

3.2 NoSQL Business Drivers

Enterprises today need highly reliable, scalable and available data storage across a configurable set of systems that act as storage nodes. The needs of organizations are changing rapidly. Many organizations operating with single CPU and Relational database management systems (RDBMS) were not able to

cope up with the speed in which information needs to be extracted. Businesses have to capture and analyze a large amount of variable data, and make immediate changes in their business based on their findings.

Figure 3.1 shows RDBMS with the business drivers velocity, volume, variability and agility necessitates the emergence of NoSQL solutions. All of these drivers apply pressure to single CPU relational model and eventually make the system less stable.

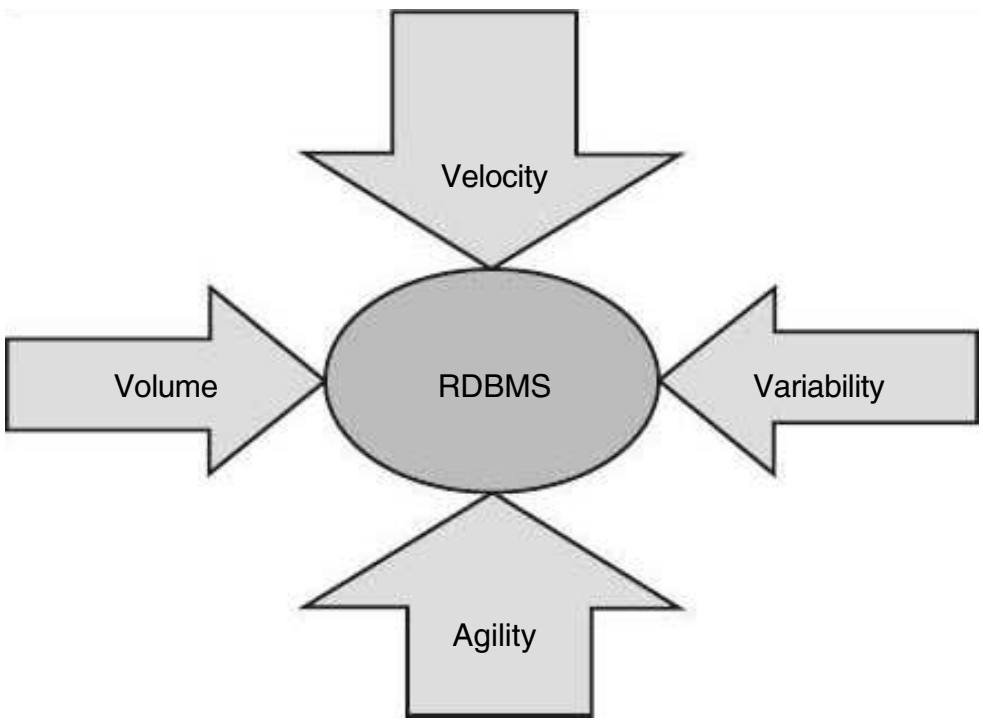


Figure 3.1 NoSQL business drivers.

3.2.1 Volume

There are two ways to look into data processing to improve performance. If the key factor is only speed, a faster processor could be used. If the processing involves complex (heavy) computation, Graphic Processing Unit (GPU) could be used along with the CPU. But the volume of data is limited to on-board GPU memory. The main reason for organizations to look at an alternative to their current RDBMSs is the need to query big data. The need to horizontal scaling made organizations to move from serial to distributed parallel processing where big data is fragmented and processed using clusters of commodity machines. This is made possible by the development of technologies like Apache Hadoop, HDFS, MapR, HBase, etc.

3.2.2 Velocity

Velocity becomes the key factor when frequency in which online queries to the database made by social networking and e-commerce web sites have to be read and written in real time. Many single CPU, RDBMS systems are unable to cope up with the demands of real-time inserts. RDBMS systems frequently index on many columns that decrease the system performance. For example, when online shopping sites introduce great discount schemes, the random bursts in web traffic will slow down the

response for every user and tuning these systems as demand increases can be costly when both high read and write is required.

3.2.3 Variability

Organizations that need to capture and report on certain uncommon data, struggle when attempting to use RDBMS fixed schema. For example, if a business process wants to store a few special attributes for a few set of customers,then it needs to alter its schema definition. If a change is made to the schema, all customer rows within the database will also have this column. If there is no value related to this for most of the customers, then the row column representation will have sparse matrix. In addition to this, new columns to an RDBMS require to execute ALTER TABLE command. This cannot be done on the fly since the present executing transaction has to complete and database has to be closed, and then schema can be altered. This process affects system availability, which means losing business.

3.2.4 Agility

The process of storing and retrieving data for complex queries in RDBMS is quite cumbersome. If it is a nested query, data will have nested and repeated subgroups of data structures that are included in an object-relational mapping layer. This layer is responsible to generate the exact combination of SELECT, INSERT, DELETE and UPDATE SQL statements to move the object data from and to the backend RDBMS layer. This process is not simple and requires experienced developers with the knowledge of object-relational frameworks such as Java Hibernate. Even then, these change requests can cause slow-downs in implementation and testing.

Desirable features of NoSQL that drive business are listed below:

1. **24 × 7 Data availability:** In the highly competitive world today, downtime is equated to real dollars lost and is deadly to a company's reputation. Hardware failures are bound to occur. Care has to be taken that there is no single point of failure and system needs to show fault tolerance. For this, both function and data are to be replicated so that if database servers or "nodes" fail, the other nodes in the system are able to continue with operations without data loss. NoSQL database environments are able to provide this facility. System updates can be made dynamically without having to take the database offline.
2. **Location transparency:** The ability to read and write to a storage node regardless of where that I/O operation physically occurs is termed as "Location Transparency or Location Independence". Customers in many different geographies need to keep data local at those sites for fast access. Any write functionality that updates a node in one location, is propagated out from that location so that it is available to users and systems at other locations.
3. **Schema-less data model:** Most of the business data is unstructured and unpredictable which a RDBMS cannot cater to. NoSQL database system is a schema-free flexible data model that can easily accept all types of structured, semi-structured and unstructured data. Also relational model has scalability and performance problems when it has to manage large data volumes.

NoSQL data model can handle this easily to deliver very fast performance for both read and write operations.

4. **Modern day transaction analysis:** Most of the transaction details relate to customer profile, reviews on products, branding, reputation, building business strategy, trading decisions, etc. that do not require ACID transactions. The data consistency denoted by "C" in ACID property in RDBMSs is enforced via foreign keys/referential integrity constraints. This type of consistency is not required to be used in progressive data management systems such as NoSQL databases since there is no JOIN operation. Here, the "Consistency" is stated in the CAP theorem that signifies the immediate or eventual consistency of data across all nodes that participate in a distributed database.

5. **Architecture that suits big data:** NoSQL solutions provide modern architectures for applications that require high degrees of scale, data distribution and continuous availability. For this multiple data center support with which a NoSQL environment complies is one of the requirements. The solution should not only look into today's big data needs but also suit greater time horizons. Hence big data brings four major considerations in enterprise architecture which are as follows:
   - *Scale of data sources*: Many companies work in the multi-terabyte and even petabyte data.
   - *Speed is essential*: Overnight extract-transform-load (ETL) batches are insufficient and real-time streaming is required.
   - *Change in storage models*: Solutions like Hadoop Distributed File System (HDFS) and unstructured data stores like Apache Cassandra, MongoDb, Neo4j provide new options.
   - Multiple compute methods for Big Data Analytics must be supported.

Figure 3.2 shows the architecture that suits big data.

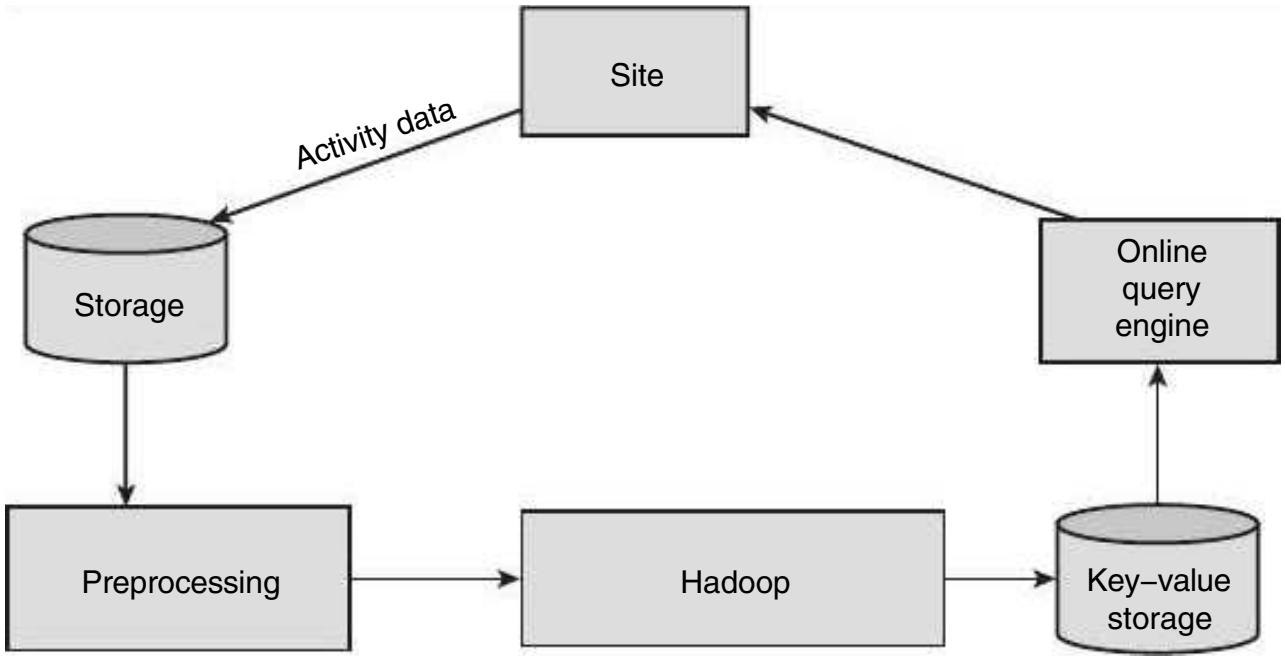


Figure 3.2 The architecture that suits big data.

6. **Analytics and business intelligence:** A key business strategic driver that suggests the implementation of a NoSQL database environment is the need to mine the data that is being collected in order to derive insights to gain competitive advantage. Traditional relational database system poses great difficulty in extracting meaningful business intelligence from very high volumes of data. NoSQL database systems not only provide storage and management of big data but also deliver integrated data analytics that provides instant understanding of complex datasets and facilitate various options for easy decision-making.

3.3 NoSQL Case Studies

Four case studies are discussed in the following subsections and each of them follow different architectural pattern, namely, key–value store, Column Family/BigTable, Document store and Graph store.

3.3.1 Amazon DynamoDB

Amazon.com has one of the largest e-commerce operations in the world. Customers from all around the world shop all hours of the day. So the site has to be up 24×7. Initially Amazon used RDBMS system for shopping cart and checkout system. Amazon DynamoDB, a NoSQL store brought a turning point.

DynamoDB addresses performance, scalability and reliability, the core problems of RDBMS when it comes to growing data. Developers can store unlimited amount of data by creating a database table and DynamoDB automatically saves it at multiple servers specified by the customer and also replicates them across multiple "Available" Zones. It can handle the data and traffic while maintaining consistent, fast performance. The cart data and session data are stored in the key–value store and the final (completed) order is saved in the RDBMS as shown in Fig. 3.3.

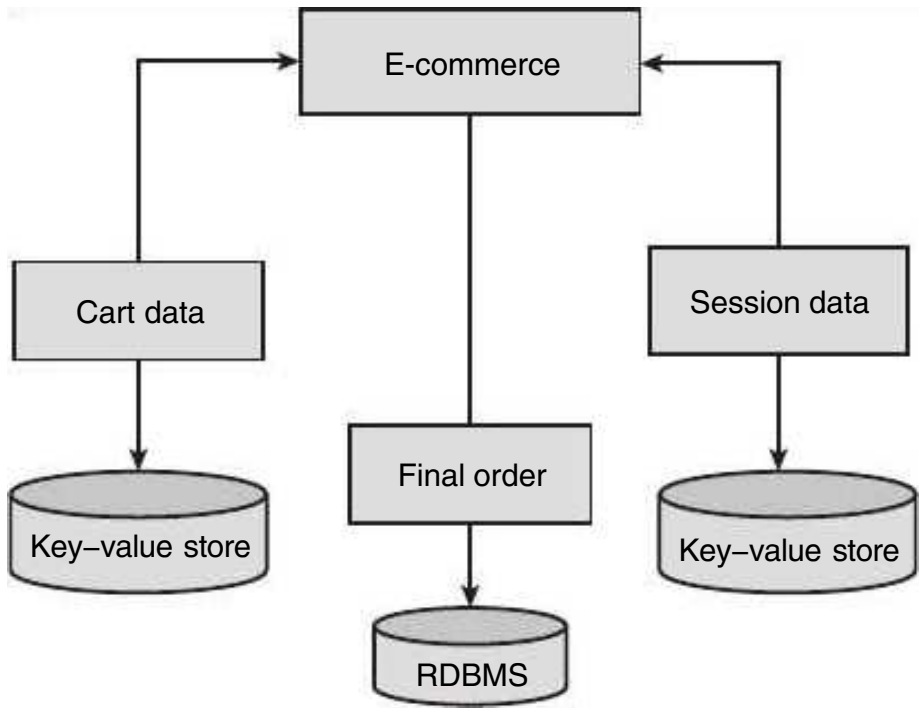


Figure 3.3 E-commerce shopping cart uses key–value store.

7. **Start MongoDB:** Issue the following command to start mongod:

   $ sudo service mongod start

8. **Stop MongoDB:** As needed, we can stop the mongod process by issuing the following command:

   $ sudo service mongod stop

9. **Restart MongoDB:** Issue the following command to restart mongod:

   $ sudo service mongod restart

II. Working with MongoDB

(a) Once the installation of MongoDB is completed type the following command to run it:

$ mongo
O/P:
MongoDB shell version: 2.6.11
connecting to: test
>

(b) To check the different commands available in MongoDB type the command as shown below:

>db.help()
O/P:

(c) DB methods:

db.adminCommand(nameOrDocument) – switches to 'admin' db, and runs command [just calls db.runCommand(...)]
db.auth(username, password)
db.cloneDatabase(fromhost)
db.commandHelp(name)– returns the help for the command
db.copyDatabase(fromdb, todb, fromhost)
db.createCollection(name, { size : ..., capped : ..., max : ... })
db.createUser(userDocument)
db.currentOp()–displays currently executing operations in the db
db.dropDatabase()
db.eval(func, args)– runs code server-side
db.fsyncLock()–flushes data to disk and lock server for backups
db.fsyncUnlock()–unlocks server following a db.fsyncLock()
db.getCollection(cname) same as db['cname'] or db.cname
db.getCollectionInfos()
db.getCollectionNames()
db.getLastError()– just returns the err msg string

db.getLastErrorObj()– returns full status object
db.getMongo()–gets the server connection object
db.getMongo().setSlaveOk()–allows queries on a replication slave server
db.getName()
db.getPrevError()
db.getProfilingLevel()– deprecated
db.getProfilingStatus()– returns if profiling is on and slow threshold
db.getReplicationInfo()
db.getSiblingDB(name) –gets the db at the same server as this one
db.getWriteConcern()– returns the write concern used for any operations on this db, inherited from server object if set
db.hostInfo()–gets details about the server's host
db.isMaster() –checks replica primary status
db.killOp(opid) –kills the current operation in the db
db.listCommands()–lists all the db commands
db.loadServerScripts()– loads all the scripts in db.system.js
db.logout()
db.printCollectionStats()
db.printReplicationInfo()
db.printShardingStatus()
db.printSlaveReplicationInfo()
db.dropUser(username)
db.repairDatabase()
db.resetError()
db.runCommand(cmdObj)– runs a database command. If cmdObj is a string, turns it into { cmdObj : 1 }
db.serverStatus()
db.setProfilingLevel(level,<slowms>) 0=off 1=slow 2=all
db.setWriteConcern(<write concern doc>) – sets the write concern for writes to the db
db.unsetWriteConcern(<write concern doc>) – unsets the write concern for writes to the db
db.setVerboseShell(flag) –displays extra information in shell output
db.shutdownServer()
db.stats()
db.version()–current version of the server
>

(d) To check the current statistic of database type the command as follows:

>db.stats()
O/P:

```
{
    "db" : "test",
    "collections" : 0,
    "objects" : 0,
    "avgObjSize" : 0,
    "dataSize" : 0,
    "storageSize" : 0,
    "numExtents" : 0,
    "indexes" : 0,
    "indexSize" : 0,
    "fileSize" : 0,
    "dataFileVersion" : {
    },
    "ok" : 1
}
>
```

Note: In the output we can see that everything is "0". This is because we haven't yet created any collection.

Some considerations while designing schema in MongoDB:

1. Design your schema as per the user's requirements.
2. Combine the objects into one document if you are going to use them together. Otherwise separate them.
3. Duplicate the data (but in limit) because disk space is cheap as compare to compute time.
4. Optimize your schema for the most frequent use cases.
5. Do join while write and not on read.
6. Do complex aggregation in the schema.

For example: Let us say that a client needs a database design for his blog and see the differences between RDBMS and MongoDB schema. Website has the following requirements:

1. Every post can have one or more tag.
2. Every post has the unique title, description and url.
3. Every post has the name of its publisher and total number of likes.
4. On each post there can be zero or more comments.

In RDBMS schema design for above requirements will have minimum three tables.
Comment(comment_id,post_id,by_user,date_time,likes,messages)
post(id,title,description,like,url,post_by)

tag_list(id,post_id,tag)

While in MongoDB schema design will have one collection (i.e., Post) and has the following structure:

```
{
    _id: POST_ID
title: TITLE_OF_POST,
description: POST_DESCRIPTION,
by: POST_BY,
url: URL_OF_POST,
tags: [TAG1, TAG2, TAG3],
likes: TOTAL_LIKES,
comments: [
    {
user:'COMMENT_BY',
message: TEXT,
dateCreated: DATE_TIME,
like: LIKES
    },
    {
user:'COMMENT_BY',
message: TEXT,
dateCreated: DATE_TIME,
like: LIKES
    }
  ]
}
```

The table given below shows the basic terminology in MongoDB in relation with RDBMS:

RDBMS	***MongoDB***
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)

Use command is used to create a new database or open an existing one.

1. **The use command:** In MongoDB use command is used to create the new database.The command creates new database,if it doesnot exist; otherwise it will return the existing database.

   Syntax:
   use DATABASE_NAME

   Example: If you want to create a database with name <mydatabase1>, then use DATABASE statement as follows:
   > use mydatabase1
   switched to db mydatabase1

   To check your currently selected database type "db".
   >db
   mydatabase1

   To check your database list type the following command:
   > show dbs
   admin (empty)
   local 0.078GB
   test (empty)

   Our created database (mydatabase1) is not present in list. To display database we need to insert atleast one document into it.
   >db.students.insert({"name":"Sushil","place":"Mumbai"})
   WriteResult({ "nInserted" : 1 })
   > show dbs
   admin (empty)
   local 0.078GB
   mydatabase1 0.078GB
   test (empty)

2. **The dropDatabase() Method:** In MongoDB, db.dropDatabase() command is used to drop an existing database.

   Syntax:
   db.dropDatabase()

   Basically it will delete the selected database, but if you have not selected any database, then it will delete the default test database.

   For example, to do so, first check the list of available databases by typing the command:

   > show dbs
   admin (empty)
   local 0.078GB
   mydatabase1 0.078GB
   test (empty)

Suppose you want to delete newly created database (i.e. mydatabase1) then

```
> use mydatabase1
switched to db mydatabase1
>db.dropDatabase()
{ "dropped" : "mydatabase1", "ok" : 1 }
```

Now just check the list of databases. You will find that the database name mydatabase1 is not present in the list. This is because it got deleted.

```
> show dbs
admin (empty)
local 0.078GB
test (empty)
```

3. **The createCollection() Method:** In MongoDB, **db.createCollection(name, options)** is used to create collection.

   Syntax:
   db.createCollection(name,options)

Parameter	*Type*	*Description*
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

Following is the list of options you can use:

Field	*Type*	*Description*
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexID	Boolean	(Optional) If true, automatically creates index on _id field.s Default value is false.
Size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
Max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

For example, basic syntax of **createCollection()** method without options is as follows:

```
> use test1
switched to db test1
>db.createCollection("mycollection1")
{ "ok" : 1 }
```

-To check the created collection:

```
> show collections
mycollection1
system.indexes
```

Note: In MongoDB you donot need to create collection.It creates collection automatically when you insert some document. For example,

```
>db.students.insert({"name" : "sushil"})
WriteResult({ "nInserted" : 1 })
> show collections
mycollection1
students
system.indexes
```

4. **The drop() Method:** MongoDB's **db.collection.drop()** is used to drop a collection from the database.
   Syntax:
   db.COLLECTION_NAME.drop()

   For example,

   -First check the available collections:

```
> show collections
mycollection1
students
system.indexes
```

   -Delete the collection named mycollection1:

```
>db.mycollection1.drop()
true
```

   -Again check the list of collection:

```
> show collections
students
system.indexes
>
```

5. The supported datatype in MongoDB are as follows:
   - **String:** This is the most commonly used datatype to store the data. String in mongodb must be UTF-8 valid.
   - **Integer:** This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
   - **Boolean:** This type is used to store a Boolean (true/ false) value.
   - **Double:** It is used to store floating point values.
   - **Min/Max keys:** This type is used to compare a value against the lowest and highest BSON elements.
   - **Arrays:** This type is used to store arrays or list or multiple values into one key.
   - **Timestamp:** ctimestamp. This can be handy for recording when a document has been modified or added.
   - **Object:** This datatype is used for embedded documents.
   - **Null:** This type is used to store a Null value.
   - **Symbol:** Its use is identically to a string. However, it is generally reserved for languages that use a specific symbol type.
   - **Date:** This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
   - **Object ID:** This datatype is used to store the document's ID.
   - **Binary data:** This datatype is used to store binary data.
   - **Code:** This datatype is used to store javascript code into document.
   - **Regular expression:** This datatype is used to store regular expression.
   - **RDBMS where Clause Equivalents in MongoDB:** To query the document on the basis of some condition, you can use following operations:

Operation	***Syntax***	***Example***	***RDBMS Equivalent***
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol. find({"likes":{$lt:50}}).pretty()	where likes <50

(*Continued*)

(Continued)

Operation	*Syntax*	*Example*	*RDBMS Equivalent*
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes >50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50

4 MapReduce

LEARNING OBJECTIVES

After reading this chapter, you will be able to:

- Learn the need for MapReduce.
- Understand Map task, Reducer task and Combiner task.
- Learn various MapReduce functions.
- Learn MapReduce algorithm for relational algebra operations.
- Learn MapReduce algorithm for matrix multiplication.

4.1 MapReduce and The New Software Stack

Businesses and governments need to analyze and process a tremendous amount of data in a very short period of time. The processing is to be done on a large amount of data, which will take a huge amount of time if done on a single machine. So the idea is to divide the data into smaller chunks and send to a cluster of machines, where they can be processed simultaneously and then the results can be combined.

Huge increase in data generated from social network and other blogging sites, for example, "Friends" on social networking sites, has led to increase in the volume of graphic data with millions of nodes and edges. This led to the creation of a new software stack. This new software stack provides parallelism by using several commodity hardware connected by Ethernet or switches. Hadoop is a framework for large-scale distributed batch processing. Hadoop can be deployed on a single machine if the data can be handled by the machine, but it is mainly designed to efficiently distribute a large amount of data for processing across a set of machines. Hadoop includes a distributed file system (DFS) that splits the input data and sends these portions of the original data to several machines in the defined cluster to hold. Main focus of this new software stack is MapReduce, a high-level programming system. This helps in doing the computation of the problem in parallel using all the connected machines so that the output, results are obtained in an efficient manner. DFS also provides data replication up to three times to avoid data loss in case of media failures.

Figure 4.1 shows the role of client machines, Master and Slave Nodes in Hadoop deployment. The MasterNode stores the huge data Hadoop Distributed File System (HDFS) and runs parallel computations on all that data (MapReduce).

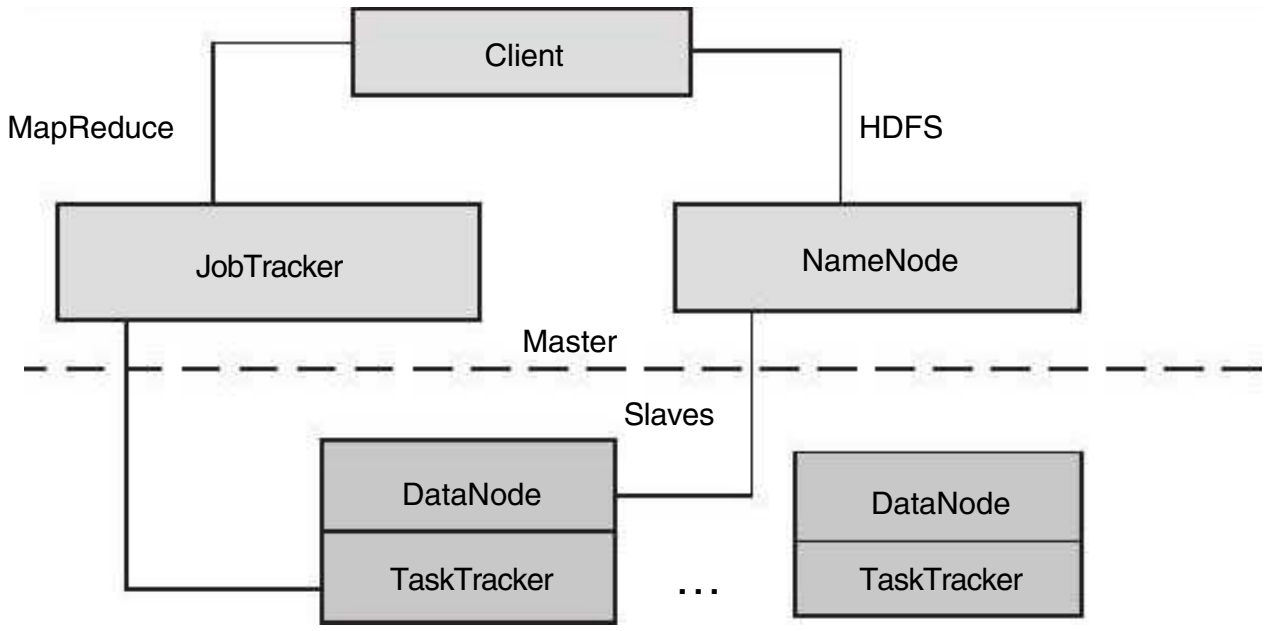


Figure 4.1 Hadoop high-level architecture.

1. The NameNode coordinates and monitors the data storage function (HDFS), while the JobTracker coordinates the parallel processing of data using MapReduce.
2. SlaveNode does the actual work of storing the data and running the computations. MasterNodes give instructions to their SlaveNodes. Each slave runs both a DataNode and a TaskTracker daemon that communicate with their respective MasterNodes.
3. The DataNode is a slave to the NameNode.
4. The TaskTracker is a slave to the JobTracker.

4.1.1 Distributed File Systems

Most scientific applications in the past, required to do parallel processing for fast computing, used special-purpose computers. Web services enabled the use of commodity nodes (having RAM, CPU and hard disk) to execute the chosen services independently on the nodes and this reduced the cost of using special-purpose machines for parallel computing. In recent times, the new parallel-computing architecture called *cluster computing* is in use. Compute nodes typically in the range of 8–64 are stored in racks and are connected with each other by Ethernet or switch to the network. Failure at the node level (disk failure) and at the rack level (network failure) is taken care of by replicating data in secondary nodes. All the tasks are completed independently and so if any task fails, it can be re-started without affecting the other tasks.

File system stores data permanently. The system has logical drives and is layered on top of physical storage medium. It is addressed by a file name under a directory that supports hierarchical nesting. Access to the file is through file path consisting of drive, directory(s) and filename.

DFS supports access to files that are stored on remote servers. It also offers support for replication and local caching. Concurrent access to files read/write has to be taken care of using locking conditions. Different types of implementations are available based on the complexity of applications.

4.1.1.1 Google File System

Google had to store a massive amount of data. It needs a good DFS with cheap commodity computers to reduce cost. These commodity computers are unreliable, hence redundant storage is required to manage failures. Most of the files in Google file system (GFS) are written only once and sometimes appended. But it needs to allow large streaming reads and so high-sustained throughput is required over low latency. File sizes are typically in gigabytes and are stored as chunks of 64 MB each. Each of these chunks is replicated thrice to avoid information loss due to the failure of the commodity hardware. These chunks are centrally managed through a single master that stores the metadata information about the chunks. Metadata stored on the master has file and chunk namespaces, namely, mapping of file to chunks and location of the replicas of each chunk. Since Google users do a lot of streaming read of large data sets, caching has no importance or benefit. What if the master fails? Master is replicated in shadow master. Also the master involvement is reduced by not moving data through it; metadata from master is cached at clients. Master chooses one of the replicas of chunk as primary and delegates the authority for taking care of the data mutations.

4.1.1.2 Hadoop Distributed File System

HDFS is very similar to GFS. Here, the master is called NameNode and shadow master is called Secondary NameNode. Chunks are called blocks and chunk server is called DataNode. DataNode stores and retrieves blocks, and also reports the list of blocks it is storing to NameNode. Unlike GFS, only single-writers per file is allowed and no append record operation is possible. Since HDFS is an open-source, interface, libraries for different file systems are provided.

4.1.2 Physical Organization of Compute Nodes

Hadoop runs best on Linux machines. Hadoop is installed in client machines with all the cluster settings. The client machine loads data and MapReduce program into the cluster, and then retrieves or views the results once the program is executed. For smaller clusters, where the number of nodes is less than 40, a single physical server can host both JobTracker and NameNode. For medium and large clusters,both of them can be in different physical servers. The "server virtualization" or "hypervisor layer" adds to overhead and impedes the Hadoop performance. Hadoop does work in a virtual machine. Cluster (with a few nodes) can be up and running in VMware Workstation on a laptop machine.

4.1.2.1 Case Study

What problem does Hadoop solve? Businesses and governments have a large amount of data that needs to be analyzed and processed very quickly. If this data is fragmented into small chunks and spread over many machines, all those machines process their portion of the data in parallel and the results are obtained extremely fast.

For example, a huge data file containing feedback mails is sent to the customer service department. The objective is to find the number of times goods were returned and refund requested. This will help the business to find the performance of the vendor or the supplier.

It is a simple word count exercise. The client will load the data into the cluster (Feedback.txt), submit a job describing how to analyze that data (word count), the cluster will store the results in a new file (Returned.txt), and the client will read the results file.

The client is going to break the data file into smaller "Blocks", and place those blocks on different machines throughout the cluster. Every block of data is on multiple machines at once to avoid data loss. So each block will be replicated in the cluster as it is loaded. The standard setting for Hadoop is to have (three) copies of each block in the cluster. This can be configured with the dfs.replication parameter in the file hdfs-site.xml.

The client breaks Feedback.txt into three blocks. For each block, the client consults the NameNode and receives a list of three DataNodes that should have a copy of this block. The client then writes the block directly to the DataNode. The receiving DataNode replicates the block to other DataNodes, and the cycle repeats for the remaining blocks. Two of these DataNodes, where the data is replicated, are in the same rack and the third one is in another rack in the network topology to prevent loss due to network failure. The NameNode as it is seen is not in the data path. The NameNode only provides the metadata, that is, the map of where data is and where data should be in the cluster (such as IP address, port number, Host names and rack numbers).

The client will initiate TCP to DataNode 1 and sends DataNode 1 the location details of the other two DataNodes. DataNode 1will initiate TCP to DataNode 2, handshake and also provide DataNode 2 information about DataNode 3. DataNode 2 ACKs and will initiate TCP to DataNode 3, handshake and provide DataNode 3 information about the client which DataNode 3 ACKs.

On successful completion of the three replications, "Block Received" report is sent to the NameNode. "Success" message is also sent to the Client to close down the TCP sessions. The Client informs the NameNode that the block was successfully written. The NameNode updates its metadata info with the node locations of Block A in Feedback.txt. The Client is ready to start the process once again for the next block of data.

The above process shows that Hadoop uses a lot of network bandwidth and storage.

The NameNode not only holds all the file system metadata for the cluster, but also oversees the health of DataNodes and coordinates access to data. The NameNode acts as the central controller of HDFS. DataNodes send heartbeats to the NameNode every 3 seconds via a TCP handshake using the same port number defined for the NameNode daemon. Every 10th heartbeat is a Block Report, where the DataNode tells the NameNode about all the blocks it has.

1. DataNode sends "hearts beat" or "block report".
2. NameNode ACK.
3. DataNode acknowledges the ACK.

Every hour, by default the Secondary NameNode connects to the NameNode and copies the in-memory metadata information contained in the NameNode and files that used to store metadata (both

may and may not be in sync). The Secondary NameNode combines this information in a fresh set of files and delivers them back to the NameNode, while keeping a copy for itself.

4.1.2.2 Receiving the Output

When a Client wants to retrieve the output of a job, it again communicates to the NameNode and asks for the block locations of the results file. The NameNode in turn provides the Client a unique list of three DataNodes for each block. Client chooses the first DataNode by default in each list. Blocks are read sequentially. Subsequent blocks are read only after the previous block is read completely.

DataNode requests the NameNode for location of block data. The NameNode will first check for DataNode in the same rack. If it is present, the NameNode provides the in-rack location from which to retrieve the data. This prevents the flow from traversing two more switches and congested links to find the data (in another rack). With the data retrieved quicker in-rack, data processing can begin sooner and the job completes that much faster.

4.1.2.3 Map Process

MapReduce is the parallel processing framework along with Hadoop, named after two important processes: Map and Reduce.

Map process runs computation on their local block of data. The MapReduce program needs to count the number of occurrences of the word "Refund" in the data blocks of Feedback.txt. Following are the steps to do this:

1. Client machine submits the MapReduce job to the JobTracker, asking "How many times does Refund occur in Feedback.txt?"
2. The JobTracker finds from the NameNode which DataNodes have blocks of Feedback.txt.
3. The JobTracker then provides the TaskTracker running on those nodes with the required Java code to execute the Map computation on their local data.
4. The TaskTracker starts a Map task and monitors the tasks progress.
5. The TaskTracker provides heartbeats and task status back to the JobTracker.
6. As each Map task completes, each node stores the result of its local computation as "intermediate data" in temporary local storage.
7. This intermediate data is sent over the network to a node running a Reduce task for final computation.

Note: If the nodes with local data already have too many other tasks running and cannot accept anymore, then the JobTracker will consult the NameNode whose Rack Awareness knowledge can suggest other nodes in the same rack. In-rack switching ensures single hop and so high bandwidth.

Google was the pioneer in this field with the use of a PageRank measure for ranking Web pages with respect to a user query. Spammers responded with ways to manipulate PageRank too with what is called Link Spam. Techniques like TrustRank were used for detecting Link Spam. Further, various variants of PageRank are also in use to evaluate the Web pages.

This chapter provides the reader with a comprehensive overview of Link Analysis techniques.

7.2 History of Search Engines and Spam

The huge volume of information on the Web is essentially useless unless information can be discovered and consumed by users. Earlier search engines fell into two broad categories:

1. *Full-text index search engines* such as AltaVista, Lycos which presented the user with a keyword search interface. Given the scale of the Web and its growth rate, creating indexes becomes a herculean task.
2. *Taxonomies based search engines* where Web pages were organized in a hierarchical way based on category labels. Example: Yahoo!. Creating accurate taxonomies requires accurate classification techniques and this becomes impossible given the size of the Web and also its rate of growth.

As the Web became increasingly used in applications like e-selling, opinion forming, information pushing, etc., web search engines began to play a major role in connecting users to information they require. In these situations, in addition to fast searching, the quality of results returned by a search engine also is extremely important. Web page owners thus have a strong incentive to create Web pages that rank highly in a search query. This led to the first generation of ***spam***, which means "manipulation of Web page content for the purpose of appearing high up in search results for selected keywords". Earlier search engines came up with several techniques to detect spam, and spammers responded with a richer set of spam techniques.

Spamdexing is the practice of search engine spamming. It is a combination of Spamming with Indexing. Search Engine Optimization (SEO) is an industry that attempts to make a Website attractive to the major search engines and thus increase their ranking. Most SEO providers resort to Spamdexing, which is the practice of creating Websites that will be illegitimately indexed with a high position in the search engines.

Two popular techniques of Spamdexing include "Cloaking" and use of "Doorway" pages.

1. Cloaking is the technique of returning different pages to search engines than what is being returned to the people. When a person requests the page of a particular URL from the Website, the site's normal page is returned, but when a search engine crawler makes the same request, a special page that has been created for the engine is returned, and the normal page for the URL is hidden from the engine – it is cloaked. This results in the Web page being indexed by the search engine under misleading keywords. For every page in the site that needs to be cloaked, another page is created that will cause this page to be ranked highly in a search engine. If more than one

search engine is being targeted, then a page is created for each engine, based on the criteria used by the different engines to rank pages. Thus, when the user searches for these keywords and views the selected pages, the user is actually seeing a Web page that has a totally different content than that indexed by the search engine. Figure 7.1 illustrates the process of cloaking.

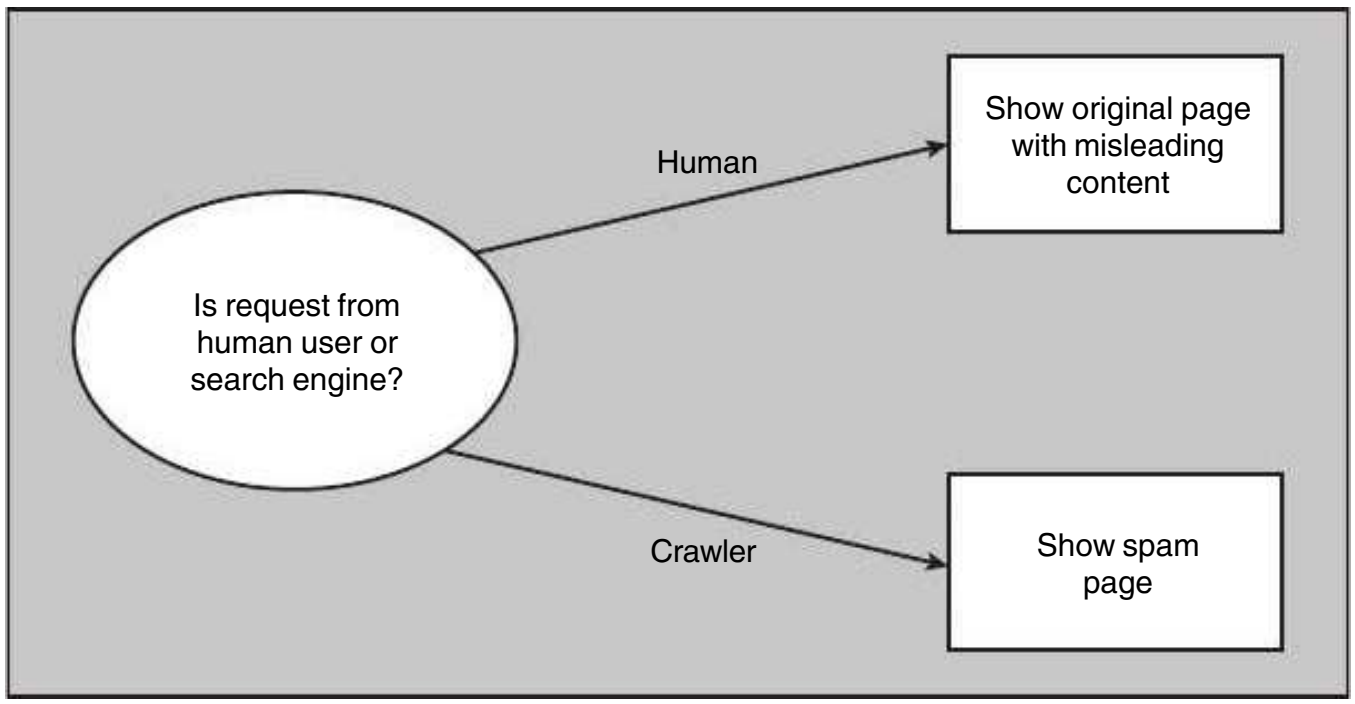


Figure 7.1 The process of cloaking.

Some example situations for cloaking include:

- Rendering a page of HTML text to search engines which guarantee its high ranking. For example, a Web site may be in the business of selling writing instruments. When a search engine browses the site, all it can see would be text indicating history of writing instruments, paper, etc. Thus, the site gets ranked highly for these concepts. The same site then shows a page of images or Flash ads of writing instruments like pens, pencils to users (people) visiting the site.
- Stuffing relevant extra text or keywords into a page to increase its ranking only when the user-agent requesting the page is a search engine, not a human visitor. Adding a term like "music" to a page several thousand times increases its ranking in the music sub topic, and thus any query about music would lead an user to the pen selling site as first choice.

2. "Doorway" pages are low-quality Web pages created with very little content, but are instead stuffed with very similar keywords and phrases. They are designed to rank highly within the search results, but serve no purpose to visitors looking for information. When a browser requests the doorway page, it is redirected to a page containing content of a more commercial nature. A dummy page with very specific keyword density is created and submitted with a redirect to a commercial site. These pages are generally very ugly and would never pass human scrutiny. The most recent method is to create a single frame and display the entire site through it. Software can create thousands of pages for a single keyword in minutes.

Other techniques used by spammers include meta-tag stuffing, scraper sites, article spinning, etc. The interested reader can go through the references for more information on term spam.

The techniques used by spammers to fool search engines into ranking useless pages higher are called as "Term Spam". Term spam refers to spam perpetuated because search engines use the visibility of terms or content in a Web page to rank them.

As a concerted effort to defeat spammers who manipulate the text of their Web pages, newer search engines try to exploit the link structure of the Web – a technique known as *link analysis*. The first Web search engine known to apply link analysis on a large scale was Google, although almost all current Web search engines make use of it. But the war between search engines and spammers is far from over as spammers now invest considerable effort in trying to manipulate the link structure too, which is now termed *link spam*.

7.3 PageRank

One of the key concepts for improving Web search has been to analyze the hyperlinks and the graph structure of the Web. Such link analysis is one of many factors considered by Web search engines in computing a composite score for a Web page on any given query.

For the purpose of better search results and especially to make search engines resistant against term spam, the concept of link-based analysis was developed. Here, the Web is treated as one giant graph: The Web page being a node and edges being links pointing to this Web page. Following this concept, the number of inbound links for a Web page gives a measure of its importance. Hence, a Web page is generally more important if many other Web pages link to it. Google, the pioneer in the field of search engines, came up with two innovations based on link analysis to combat term spam:

1. Consider a random surfer who begins at a Web page (a node of the Web graph) and executes a random walk on the Web as follows. At each time step, the surfer proceeds from his current page A to a randomly chosen Web page that A has hyperlinks to. As the surfer proceeds in this random walk from node to node, some nodes will be visited more often than others; intuitively, these are nodes with many links coming in from other frequently visited nodes. As an extension to this idea, consider a set of such random surfers and after a period of time find which Web pages had large number of surfers visiting it. The idea of PageRank is that pages with large number of visits are more important than those with few visits.
2. The ranking of a Web page is not dependent only on terms appearing on that page, but some weightage is also given to the terms used in or near the links to that page. This helps to avoid term spam because even though a spammer may add false terms to one Website, it is difficult to identify and stuff keywords into pages pointing to a particular Web page as that Web page may not be owned by the spammer.

The algorithm based on the above two concepts first initiated by Google is known as PageRank. Since both number and quality are important, spammers just cannot create a set of dummy low-quality Web pages and have them increase the number of in links to a favored Web page.

In Google's own words: *PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the Website is. The underlying assumption is that more important Websites are likely to receive more links from other Websites.*

This section discusses the PageRank algorithm in detail.

7.3.1 PageRank Definition

PageRank is a link analysis function which assigns a numerical weighting to each element of a hyperlinked set of documents, such as the WWW. PageRank helps in "measuring" the relative importance of a document (Web page) within a set of similar entities. The numerical weight that it assigns to any given element E is referred to as the *PageRank of* E and denoted by PR(E). The PageRank value of a Web page indicates its importance – *higher the value more relevant is this Webpage*.

A hyperlink to a page counts as a vote of support. The PageRank of a page is defined recursively and depends on the number and PageRank metric of all pages that link to it ("incoming links"). A page that is linked to by many pages with high PageRank receives a *high rank itself*.

We can illustrate the simple computation for pageRank using the Web graph model depicted in Fig. 7.2. The figure shows a tiny portion of the Web with five pages 1, 2, 3, 4, and 5. Directed arcs indicate links between the pages. For example, in the figure, the links coming into page 5 are *backlinks* (inlinks) for that page and links going out from page 5 are called *outlinks*. Pages 4 and 1 have a single backlink each, pages 2 and 3 have two backlinks each, and page 5 has three backlinks.

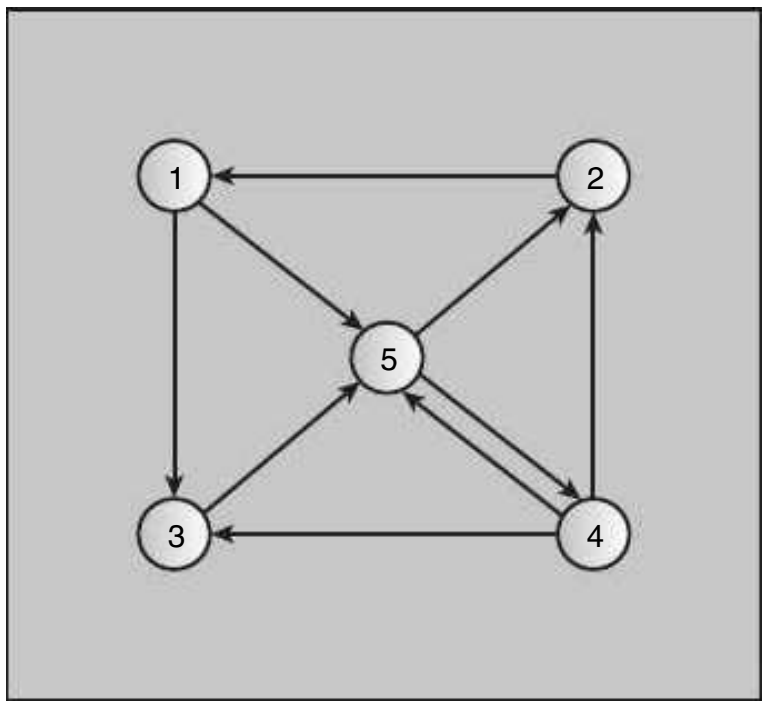


Figure 7.2 A hypothetical web graph.

Let us consider a random surfer who begins at a Web page (a node of the Web graph) and executes a random walk on the Web as follows. At each time step, the surfer proceeds from his current page to a randomly chosen Web page that it has hyperlinks to. So in our figure, the surfer is at a node 1, out of which there are two hyperlinks to nodes 3 and 5; the surfer proceeds at the next time step to one of these two nodes, with equal probabilities 1/2. The surfer has zero probability of reaching 2 and 4.

We can create a "Transition Matrix" "M" of the Web similar to an adjacency matrix representation of a graph, except that instead of using Boolean values to indicate presence of links, we indicate the probability of a random surfer reaching that node from the current node. The matrix M is an $n \times n$ matrix if there are n Web pages. For a Web page pair (P_i, P_j), the corresponding entry in M (row i column j) is

$$M(i,j)=\frac{1}{k}$$

where k is the number of outlinks from P_j and one of these is to page P_i, otherwise $M(i, j) = 0$. Thus, for the Web graph of Fig. 7.2, the following will be the matrix M_5:

$$M_5 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 1 & \frac{1}{3} & 0 \end{bmatrix} \tag{7.1}$$

We see that column 2 represents node 2, and since it has only one outlink to node 1, only first row has a 1 and all others are zeroes. Similarly, node 4 has outlinks to node 2, 3, and 5 and thus has value 1/3 to these nodes and zeroes to node 1 and 4.

7.3.2 PageRank Computation

We know that the PageRank value of a page depends on how important that page is. To compute how important a page is we need to know the probability that a random surfer will land at that page and higher the probability, the more important the page.

To determine the location of a random surfer, we use a column vector v of size n where n is the number of WebPages. The j^{th} component of this vector v is the probability that the surfer is at page j. This probability is nothing but an indication of PageRank value of that page.

1. Initially the surfer can be at any of the n pages with probability $1/n$. We denote it as follows:

$$v_0 = \begin{bmatrix} 1/n \\ 1/n \\ \vdots \\ \vdots \\ 1/n \\ 1/n \end{bmatrix}$$

2. Consider M, the transition matrix. When we look at the matrix M_5 in Eq. (7.1), we notice two facts: the sum of entries of any column of matrix M is always equal to 1. Further, all entries have values greater or equal to zero. Any matrix possessing the above two properties is called as a matrix of a *Markov chain process*, also called *Markov transition matrix.* At any given instant of time, a process in a Markov chain can be in one of the N states (in a Web set up a state is a node or Web page). Then, the entry m_{ij} in the matrix M gives us the probability that i will be the next node visited by the surfer, provided the surfer is at node j currently. Because of the Markov property, the next node of the surfer only depends on the current node he is visiting. Recall that this is exactly the way we have designed the Transition matrix in Section 7.3.1.

3. If vector v shows the probability distribution for the current location, we can use v and M to get the distribution vector for the next state as $x = Mv$. Say currently the surfer is at node j. Then, we have

$$x = M \times v_j = \sum_j m_{ij} \times v_j \tag{7.2}$$

   Here v_j is the column vector giving probability that current location is j for every node 1 to n. Thus after first step, the distribution vector will be Mv_0. After two steps, it will be $M(Mv_0)$. Continuing in this fashion after k steps the distribution vector for the location of the random surfer will be $M^k(Mv_0)$.

4. This process cannot continue indefinitely. If a Markov chain is allowed to run for many time steps, the surfer starts to visit certain Web pages (say, a popular stock price indicator site) more often than other pages and slowly the visit frequency converges to fixed, steady-state quantity. Thus, the distribution vector v remains the same across several steps. This final equilibrium state value in v is the PageRank value of every node.

5. For a Markov chain to reach equilibrium, two conditions have to be satisfied, the graph must be strongly connected and there must not exist any dead ends, that is, every node in the graph should have at least one outlink. For the WWW this is normally true. When these two conditions are satisfied, for such a Markov chain, there is a unique steady-state probability vector, that is, the principal left eigenvector of the matrix representing the Markov chain.

In our case we have v as the principal eigenvector of matrix M. (An eigenvector of a matrix M is a vector v that satisfies $v = \beta Mv$ for some constant eigenvalue β.) Further, because all columns of the matrix M total to 1, the eigenvalue associated with this principle eigenvector is also 1.

Thus to compute PageRank values of a set of WebPages, we must compute the principal left eigenvector of the matrix M with eigenvalue 1. There are many algorithms available for computing left eigenvectors. But when we are ranking the entire Web, the size of the matrix M could contain a billion rows and columns. So a simple iterative and recursive algorithm called the Power method is used to compute the eigenvalue. It is calculated repetitively until the values in the matrix converge. We can use the following equation to perform iterative computations to calculate the value of PageRank:

$$x_k = M^k(Mv_0) \tag{7.3}$$

After a large number of steps, the values in x_k settle down where difference in values between two different iterations is negligible below a set threshold. At this stage, the values in vector x_k indicate the PageRank values of the different pages. Empirical studies have shown that about 60–80 iterations cause the values in x_k to converge.

Example 1

Let us apply these concepts to the graph of Fig. 7.2 represented by the matrix M_5 as shown before. As our graph has five nodes

$$v_0 = \begin{bmatrix} 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \end{bmatrix}$$

If we multiply v_0 by matrix M_5 repeatedly, after about 60 iterations we get converging values as

$$\begin{bmatrix} 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \end{bmatrix} \begin{bmatrix} 1/5 \\ 1/6 \\ 1/6 \\ 1/10 \\ 11/30 \end{bmatrix} \begin{bmatrix} 1/6 \\ 13/60 \\ 2/15 \\ 11/60 \\ 3/10 \end{bmatrix} \cdots\cdots\cdots \begin{bmatrix} 0.4313 \\ 0.4313 \\ 0.3235 \\ 0.3235 \\ 0.6470 \end{bmatrix}$$

Thus Page 1 has PageRank 0.4313 as does Page 2. Pages 3 and 4 have PageRank 0.3235 and Page 5 has the highest PageRank of 0.6470.

7.3.3 Structure of the Web

One of the assumptions made for using the concept of Markov processes to compute PageRank is that the entire Web is one giant strongly connected entity. Theoretically a surfer starting at any random Web page can visit any other Web page through a set of links.

But one study conducted in 2000 threw up some surprising results. Researchers from IBM, the AltaVista search engine and Compaq Systems in 2000 conducted a comprehensive study to map the structure of the Web. They analyzed about 200 million Web pages and 1.5 billion hyperlinks. Andrei Broder, AltaVista's Vice President of research at that time and lead author of the study proclaimed that "The old picture – where no matter where you start, very soon you will reach the entire Web – is not quite right."

Older models of the Web had always portrayed its topology as a cluster of sites connected to other clusters and all forming one Strongly Connected Component (SCC). An SCC can be defined as a sub-graph of a graph where a path exists between every pair of nodes in the sub-graph. Further, this is the maximal sub-graph with this property. But the results of the study present a different picture of the Web. The Web, according to the study, consists of three distinct regions:

1. One large portion called as "Core" which is more or less strongly connected so as to form an SCC. Web surfers in the Core can reach any Webpage in the core from any other Webpage in the core. Mostly this is the region of the Web that most surfers visit frequently.
2. A portion of the Web consisted of Web Pages that had links that could lead to the SCC but no path from the SCC led to these pages. This region is called the IN-Component and then pages are called IN pages or "Origination" Pages. New Web Pages or Web Pages forming closed communities belong to this component.
3. Another set of nodes exist that can be reached from the SCC but do not have links that can ultimately lead to a Webpage in the SCC. This is called the "Out" component and the Web Pages "Out" pages or "termination" pages. Many corporate sites, e-commerce sites, etc. expect the SCC to have links to reach them but do not really need links back to the core.

Figure 7.3 shows the original image from the study conducted by Broder *et al*. Because of the visual impact the picture made, they termed this as the "bow-tie picture" of the Web, with the SCC as the central "knot". Figure 7.3 also shows some pages that belong to none of IN, OUT, or SCC. These are further classified into:

1. **Tendrils:** These are pages that do not have any inlinks or outlinks to/from the SCC. Some tendrils consist of pages reachable from the in-component but not SCC and some other tendrils can reach the out-component but not from the SCC.
2. **Tubes:** These are pages that reach from in-component to the out-component without linking to any pages in the SCC.

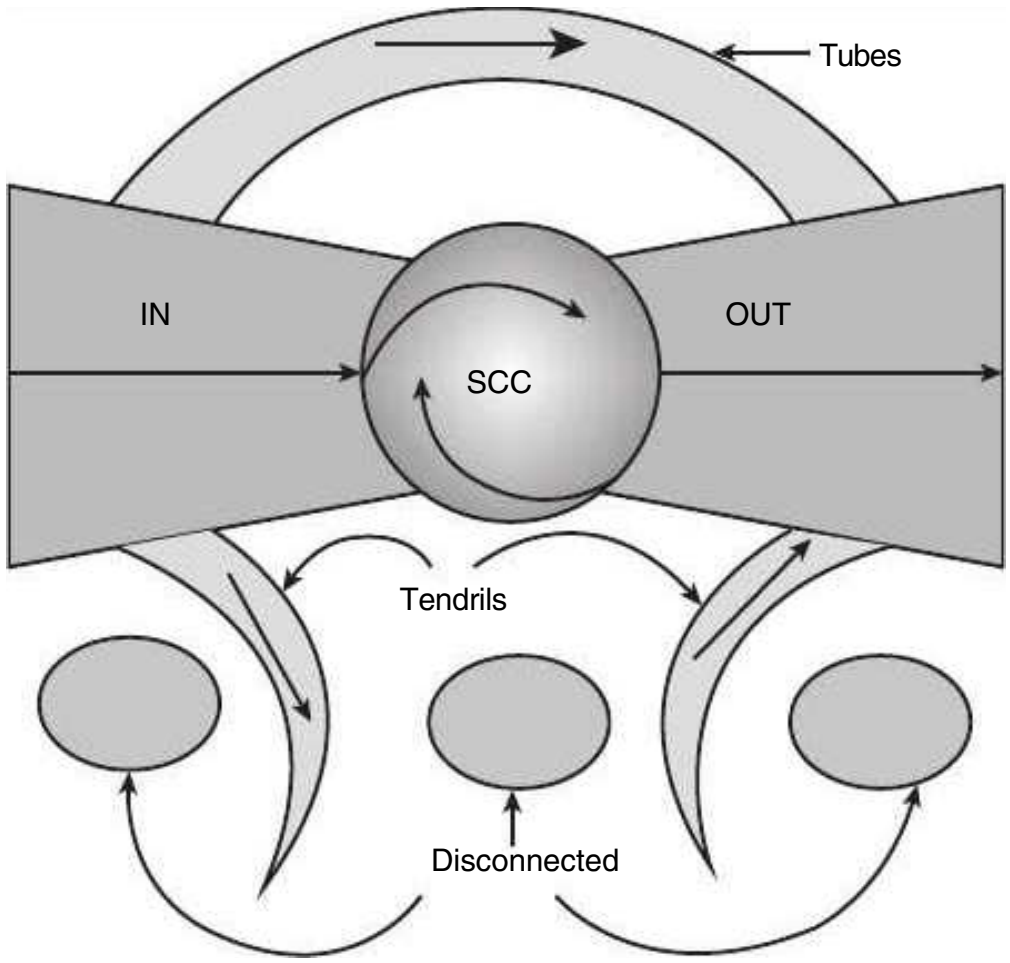


Figure 7.3 Bow-tie structure of the web (Broder *et al.*).

The study further also discussed the size of each region and it was found that perhaps the most surprising finding is the size of each region. Intuitively, one would expect the core to be the largest component of the Web. It is, but it makes up only about one-third of the total. Origination and termination pages both make up about a quarter of the Web, and disconnected pages about one-fifth.

As a result of the bow-tie structure of the Web, assumptions made for the convergence of the Markov process do not hold true causing problems with the way the PageRank is computed. For example, consider the out-component and also the out-tendril of the in-component; if a surfer lands in either of these components he can never leave out, so probability of a surfer visiting the SCC or the in-component is zero from this point. This means eventually pages in SCC and in-component would end up with very low PageRank. This indicates that the PageRank computation must take the structure of the Web into consideration.

There are two scenarios to be taken care of as shown in Fig. 7.4:

1. **Dead ends:** These are pages with no outlinks. Effectively any page that can lead to a dead end means it will lose all its PageRank eventually because once a surfer reaches a page that is a dead end no other page has a probability of being reached.
2. **Spider traps:** These are a set of pages whose outlinks reach pages only from that set. So eventually only these set of pages will have any PageRank.

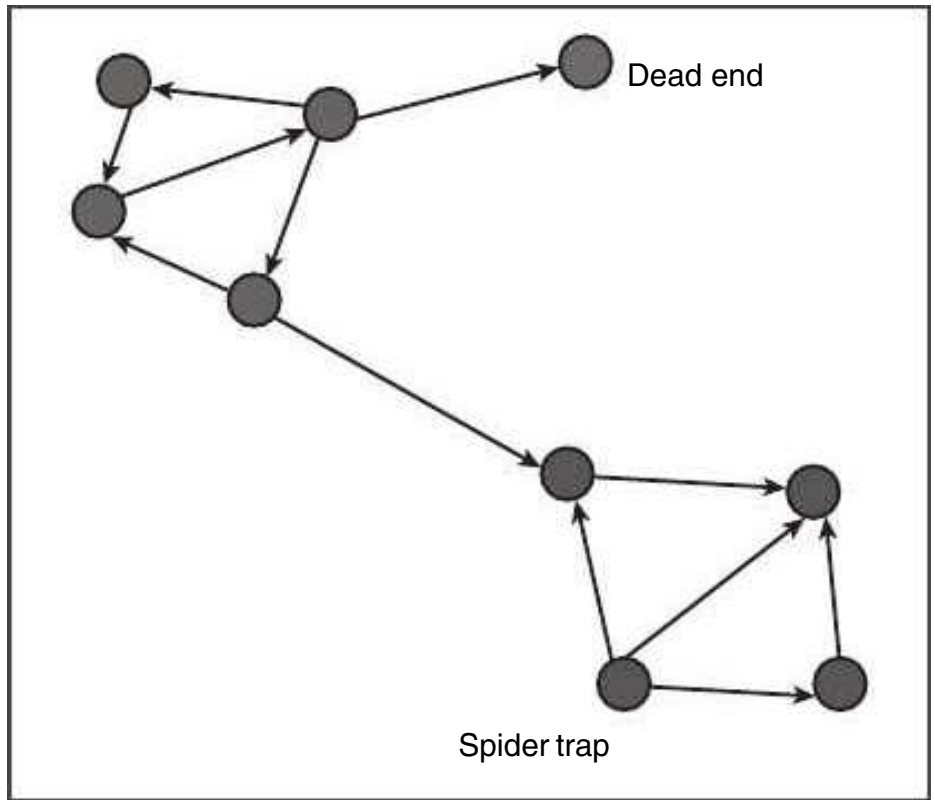


Figure 7.4 Dead end and spider trap.

In both the above scenarios, a method called "taxation" can help. Taxation allows a surfer to leave the Web at any step and start randomly at a new page.

7.3.4 Modified PageRank

One problem we have to solve is the problem of dead ends. When there are dead ends, some columns of the transition matrix M will not sum to 1, they may be 0. For such a matrix where values may sum up to at most 1, the power operation used to generate the ranking vector will result in some or all components reaching 0.

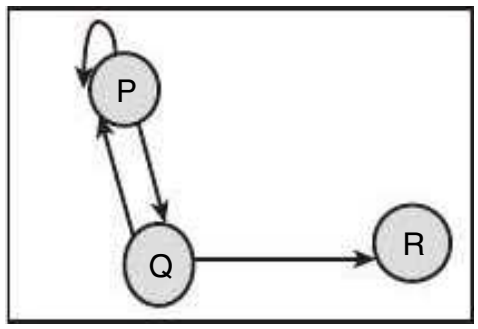
$$M^k v \text{ results in } V \to 0$$

A simple example can illustrate this. Consider a 3-node network P, Q and R and its associated transition matrix. Since R is dead, eventually all PageRank leaks out leaving all with zero PageRank.

$$\begin{bmatrix} X_P \\ X_Q \\ X_R \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \begin{bmatrix} 2/6 \\ 1/6 \\ 3/6 \end{bmatrix} \begin{bmatrix} 3/12 \\ 2/12 \\ 7/12 \end{bmatrix} \begin{bmatrix} 5/24 \\ 3/24 \\ 16/24 \end{bmatrix} \cdots\cdots \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

All the PageRank is trapped in R. Once a random surfer reaches R, he can never leave. Figure 7.5 illustrates this.

We now propose modifications to the basic PageRank algorithm that can avoid the above two scenarios as described in the following subsections.



	P	Q	R
P	1/2	1/2	0
Q	1/2	0	0
R	0	1/2	0

Figure 7.5 A simple Web Graph and its associated transition matrix.

7.3.4.1 Dealing with Dead Ends

We can remove nodes that are dead ends. "Remove" all pages with no outgoing links and remove their in links too. This is a repetitive operation; removing some pages may lead to other pages becoming dead ends. So, recursively we may require to remove more pages ultimately stopping when we land up with an SCC. Name this graph G.

Keep track of the order in which the pages were removed, and in what iteration, because we must eventually restore the graph in reverse order. Now using any method, compute the PageRank of graph G. Now we have to restore the graph by putting back the links and the nodes removed as dead ends. We restore nodes in the reverse order in which they were deleted. The last set of nodes to be deleted will be those whose in links are directly from SCC. All the nodes in the SCC have their PageRank computed.

When we put a dead end page back in, we can compute its PageRank as the sum of PageRanks it receives from each of its inlinks in the SCC. Each inlink will contribute to the dead end page its own PageRank divided by the number of outgoing links for that page.

Example 2

Consider Fig. 7.6.

1. Part (a) shows a portion of a Web graph where A is part of the SCC and B is a dead end. The self-loop of A indicates that A has several links to pages in SCC.
2. In part (b), the dead end B and its links are removed. PageRank of A is computed using any method.
3. In part (c), the dead end last removed, that is B, is put back with its connections. B will use A to get its PageRank. Since A now has two outlinks, its PageRank is divided into 2 and half this rank is propagated to B.
4. In part (d), A has two outlinks and C has three outlinks and both propagate 1/2 and 1/3 of their PageRank values to B. Thus, B gets the final PageRank value as shown. In part (d), A is having a PR value of 2/5 and C has 2/7 leading to B obtaining a PR value of 31/105.

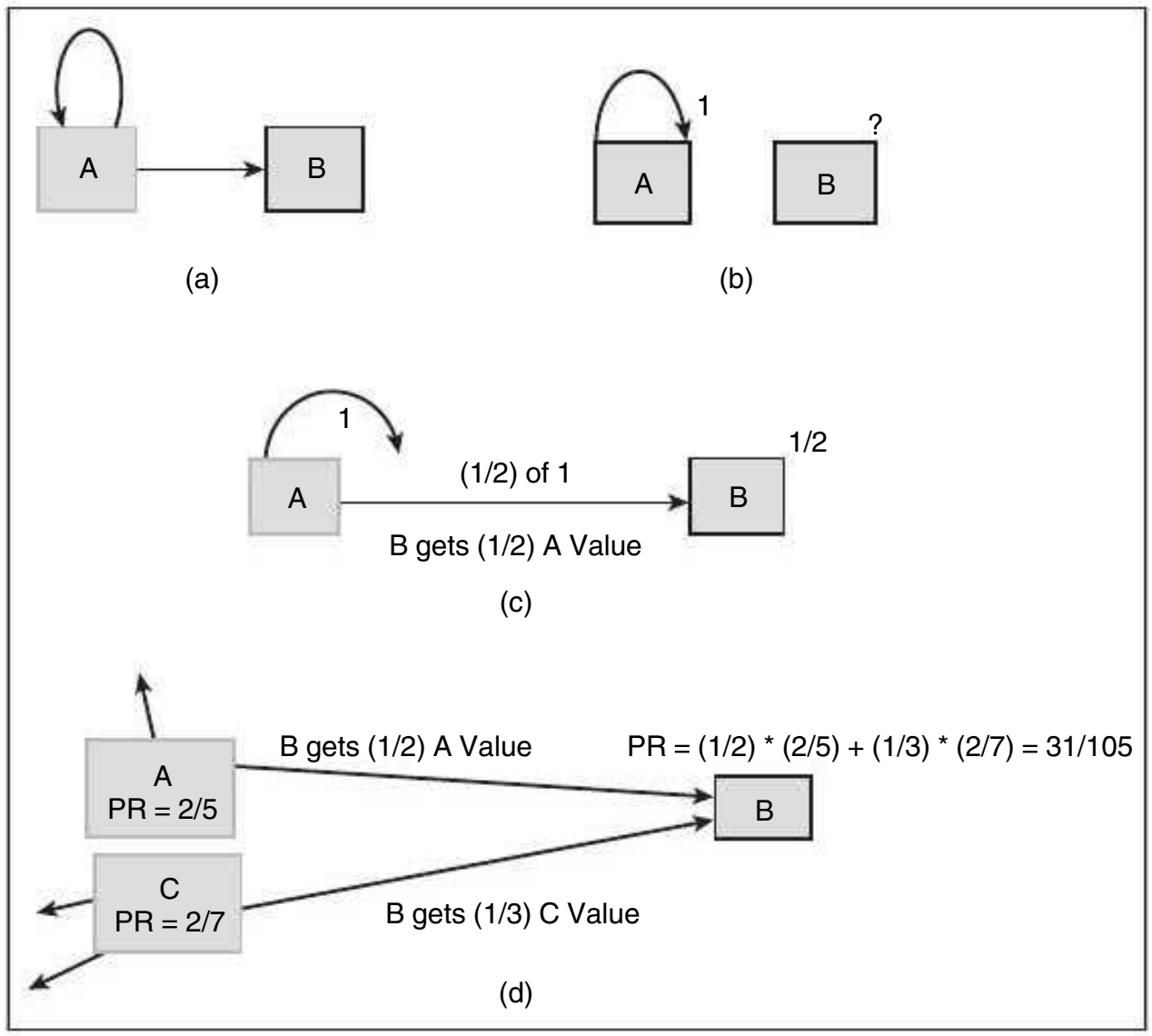


Figure 7.6 Computing PageRank for dead end pages.

7.3.4.2 Avoiding Spider Traps

As we recall, a spider trap is a group of pages with no links out of the group. We have illustrated earlier that in a spider trap all PageRank value will eventually be "trapped" by the group of pages.

To address this issue, we introduce an additional operation for our random surfer: the *teleport* operation. In the teleport operation, the surfer jumps from a node to any other node in the Web graph. Thus, the random surfer, instead of always following a link on the current page, "teleports" to a random page with some probability – Or if the current page has no outgoing links.

In assigning a PageRank score to each node of the Web graph, we use the teleport operation in two ways:

1. When a node has no outlinks, the surfer can invoke the teleport operation with some probability.
2. If a node has outgoing links, the surfer can follow the standard random walk policy of choosing any one of the outlinks with probability $0 < \beta < 1$ and can invoke the teleport operation with probability $1 - \beta$, where β is a fixed parameter chosen in advance.

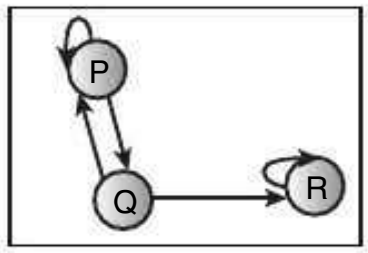
Typical values for β might be 0.8–0.9. So now the modified equation for computing PageRank iteratively from the current PR value will be given by

$$v' = \beta Mv + (1 - \beta)e/n$$

where M is the transition matrix as defined earlier, v is the current PageRank estimate, e is a vector of all ones and n is the number of nodes in the Web graph.

Example 3

Consider the same Web graph example shown earlier in 7.5 (we repeat the graph for clarity). The following shows the new PageRank computations with $\beta = 0.8$.



	P	Q	R
P	1/2	1/2	0
Q	1/2	0	0
R	0	1/2	1

(0.8)*

	P	Q	R
P	1/2	1/2	0
Q	1/2	0	0
R	0	1/2	1

+ (0.2)*

	P	Q	R
P	1/3	1/3	1/3
Q	1/3	1/3	1/3
R	1/3	1/3	1/3

=

	P	Q	R
P	7/15	7/15	1/15
Q	7/15	1/15	1/15
R	1/15	7/15	13/15

Eventually

$$\begin{bmatrix} X_P \\ X_Q \\ X_R \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1.00 \\ 0.60 \\ 1.40 \end{bmatrix} \begin{bmatrix} 0.84 \\ 0.60 \\ 1.56 \end{bmatrix} \begin{bmatrix} 0.776 \\ 0.536 \\ 1.688 \end{bmatrix} \cdots\cdots\cdots \begin{bmatrix} 7/33 \\ 5/33 \\ 21/33 \end{bmatrix}$$

This indicates that the spider trap has been taken care of. Even though R has the highest PageRank, its effect has been muted as other pages have also received some PageRank.

7.3.5 Using PageRank in a Search Engine

The PageRank value of a page is one of the several factors used by a search engine to rank Web pages and display them in order of ranks in response to an user query. When a user enters a query, also called a search, into Google, the results are returned in the order of their PageRank. The finer details of working of Google's working are proprietary information.

Google utilizes a number of factors to rank search results including standard Information Retrieval measures, proximity, anchor text (text of links pointing to Web pages), and PageRank. According to Sergei and Brin, the designers of PageRank measure and creators of Google, at inception, the ranking of

Web pages by the Google search engine was determined by three factors: Page specific factors, Anchor text of inbound links, PageRank.

Page-specific factors include the body text, for instance, the content of the title tag or the URL of the document. In order to provide search results, Google computes an IR score out of page-specific factors and the anchor text of inbound links of a page. The position of the search term and its weightage within the document are some of the factors used to compute the score. This helps to evaluate the relevance of a document for a particular query. The IR-score is then combined with PageRank to compute an overall importance of that page.

In general, for queries consisting of two or more search terms, there is a far bigger influence of the content-related ranking criteria, whereas the impact of PageRank is more for unspecific single word queries. For example, a query for "Harvard" may return any number of Web pages which mention Harvard on a conventional search engine, but using PageRank, the university home page is listed first.

Currently, it is estimated that Google uses about 250 page-specific properties with updated versions of the PageRank to compute the final ranking of pages with respect to a query.

7.4 Efficient Computation of PageRank

Essentially the implementation of PageRank is very simple. The sheer size of the Web, however, requires much greater care in the use of data structures. We need to perform a matrix and vector multiplication of very large matrices about 50–80 times. This is computationally very expensive.

The algorithm naturally lends itself to a MapReduce (MR) type of scenario. But before we use the MR paradigm, it is also possible to have a more efficient representation of the transition matrix M depicting the Web graph. Further, the use of Combiners with MR helps avoid thrashing.

7.4.1 Efficient Representation of Transition Matrices

Let us consider the structure of a Web graph more precisely. It is typical for a search engine to analyze billions of pages with respect to a query. On an average, a page may have a maximum of 10–15 outlinks. If the page happens to be a doorway page to other pages, a maximum of 25–30 outlinks may exist. When a transition matrix is constructed for computation of PageRank, the matrix will be a billion rows and a billion columns wide. Further, most of the entries in this giant matrix will be zero. Thus, the transition matrix tends to be a very sparse matrix. The storage required to store this matrix is quadratic in the size of the matrix. We can definitely find a more efficient way of storing the transition data by considering only the non-zero values.

One way will be to just list all the non-zero outlinks of a node and their values. This method will need space linear in the number of non-zero entries; 4 byte integers for the node value and 8 byte floating point number for the value of the link. But we can perform one more optimization for a transi-

Generally, after a number of iterations, the authority and hub scores do not vary much and can be considered to have "converged".

HITS algorithm and the **PageRank algorithm** both make use of the link structure of the Web graph to decide the relevance of the pages. The difference is that while the **PageRank** is query independent and works on a large portion of the Web, **HITS** only operates on a small subgraph (the seed S_Q) from the Web graph.

The most obvious strength of HITS is the two separate vectors it returns, which allow the application to decide on which score it is most interested in. The highest ranking pages are then displayed to the user by the query engine.

This sub-graph generated as seed is query dependent; whenever we search with a different query phrase, the seed changes as well. Thus, the major disadvantage of HITS is that the query graph must be regenerated dynamically for each query.

Using a query-based system can also sometimes lead to link spam. Spammers who want their Web page to appear higher in a search query, can make spam farm pages that link to the original site to give it an artificially high authority score.

Summary

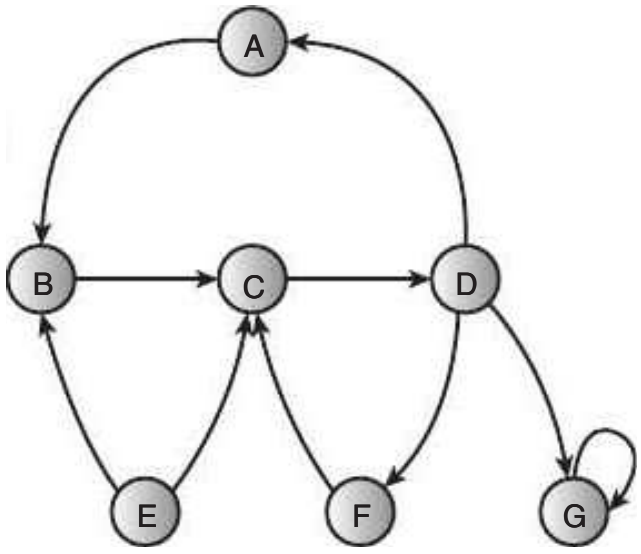
- As search engines become more and more sophisticated, to avoid being victims of spam, spammers also are finding innovative ways of defeating the purpose of these search engines. One such technique used by modern search engines to avoid spam is to analyze the hyperlinks and the graph structure of the Web for ranking of Web search results. This is called Link Analysis.
- Early search engines were mostly text based and susceptible to spam attacks. ***Spam*** means "manipulation of Web page content for the purpose of appearing high up in search results for selected keywords".
- To attack text-based search engines, spammers resorted to term based spam attacks like cloaking and use of Doorway pages.
- Google, the pioneer in the field of search engines, came up with two innovations based on Link Analysis to combat term spam and called their algorithm PageRank.
- The basic idea behind PageRank is that the ranking of a Web page is not dependent only on terms appearing on that page, but some weightage is also given to the terms used in or near the links to that page. Further pages with large no of visits are more important than those with few visits.
- To compute PageRank, "Random Surfer Model" was used. Calculation of PageRank can be thought of as simulating the behavior of many random surfers, who each start at a random page and at any step move, at random, to one of the pages to which their

current page links. The limiting probability of a surfer being at a given page is the PageRank of that page.

- An iterative matrix based algorithm was proposed to compute the PageRank of a page efficiently.
- The PageRank algorithm could be compromised due to the bow–tie structure of the Web which leads to two types of problems, dead ends and spider traps.
- Using a scheme of random teleportation, PageRank can be modified to take care of dead ends and spider traps.
- To compute the PageRank of pages on the Web efficiently, use of MapReduce is advocated. Further schemes of efficiently storing the transition matrix and the PageRank vector are described.
- In Topic-Sensitive PageRank, we bias the random walker to teleport to a set of topic-specific relevant nodes. The topic is determined by the context of the search query. A set of PageRank vectors, biased using a set of representative topics, helps to capture more accurately the notion of importance with respect to a particular topic. This in turn yields more accurate search results specific to a query.
- Link spam can be formally stated as a class of spam techniques that try to increase the link-based score of a target Web page by creating lots of spurious hyperlinks directed towards it. These spurious hyperlinks may originate from a set of Web pages called a Link farm and controlled by the spammer. They may be created from a set of partner Web sites known as link exchange. Sometimes such links could also be placed in some unrelated Websites like blogs or marketplaces. These structures are called Spam farms.
- Search engines can respond to link spam by mining the Web graph for anomalies and propagating a chain of distrust from spurious pages which will effectively lower the PageRank of such pages. TrustRank and Spam mass are two techniques used to combat Link Spam.
- In a parallel development along with PageRank, another algorithm to rank pages in relation to a query posed by a user was proposed. This algorithm also used the link structure of the Web in order to discover and rank pages relevant for a particular topic. The idea was to associate two scores with each Web page, contributions coming from two different types of pages called “hubs” and “authorities”. This algorithm is called hyperlink-induced topic search (HITS). HITS presently is used by the **Ask** search engine (www.Ask.com). Further it is believed that modern information retrieval engines use a combination of PageRank and HITS for query answering.
- Calculation of the hubs and authorities scores for pages depends on solving the recursive equations: “a hub links to many authorities, and an authority is linked to by many hubs”. The solution to these equations is essentially an iterated matrix–vector multiplication, just like PageRank’s.

Exercises

1. Consider the portion of a Web graph shown below.

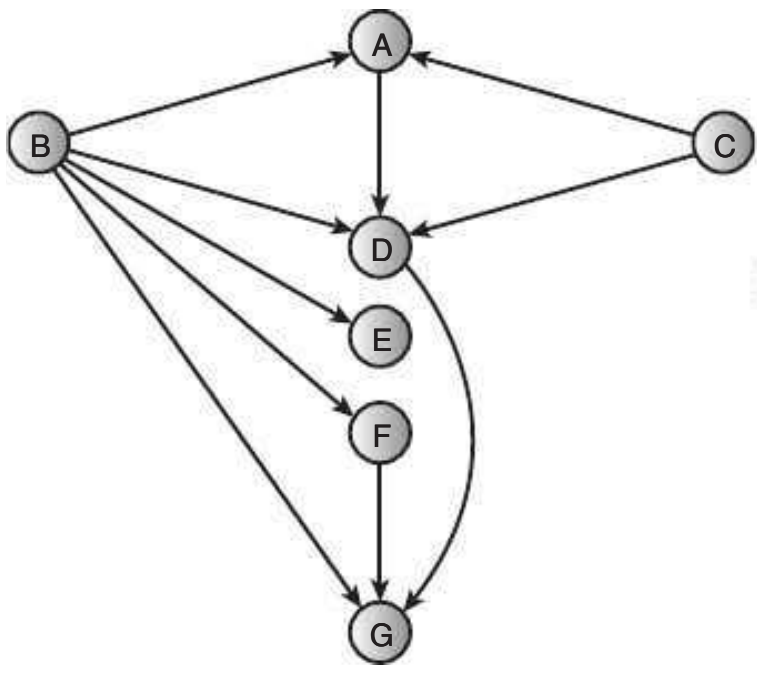


(a) Compute the hub and authority scores for all nodes.

(b) Does this graph contain spider traps? Dead ends? If so, which nodes?

(c) Compute the PageRank of the nodes without teleportation and with teleportation $\beta = 0.8$.

2. Compute hub and authority scores for the following Web-graph:



3. Let the adjacency matrix for a graph of four vertices (n_1 to n_4) be as follows:

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Calculate the authority and hub scores for this graph using the HITS algorithm with $k = 6$, and identify the best authority and hub nodes.

4. Can a Website's PageRank ever increase? What are its chances of decreasing?

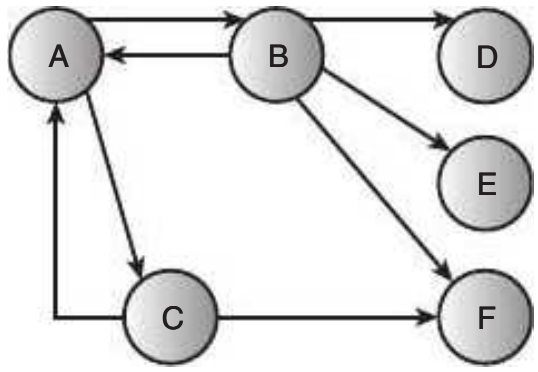
5. Given the following HITS scoring vector, normalize x for the next iteration of the algorithm:

$$x = \begin{bmatrix} 3.12 \\ 4.38 \\ 6.93 \\ 3.41 \\ 1.88 \\ 4.53 \end{bmatrix}$$

6. Consider the Web graph given below with six pages (A, B, C, D, E, F) with directed links as follows:

- A → B, C
- B → A, D, E, F
- C → A, F

Assume that the PageRank values for any page m at iteration 0 is $PR(m) = 1$ and teleportation factor for iterations is $\beta = 0.85$. Perform the PageRank algorithm and determine the rank for every page at iteration 2.



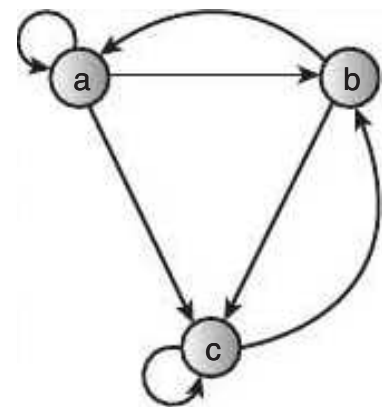
7. Consider a Web graph with three nodes 1, 2, and 3. The links are as follows: $1 \rightarrow 2$, $3 \rightarrow 2$, $2 \rightarrow 1$, $2 \rightarrow 3$. Write down the transition probability matrices for the surfer's walk with teleporting, for the following three values of the teleport probability:

(a) $\beta = 0$

(b) $\beta = 0.5$

(c) $\beta = 1$

8. Compute the PageRank of each page in the following graph without taxation and with taxation factor of 0.8.



9. For all the Web graphs discussed in the exercises, represent the transition matrices and the PageRank vector using methods discussed in Section 7.3.

10. Compute the TopicSensitive PageRank for the graph of Exercise 2 assuming the teleport set is: (A, C, E, F) only.

Programming Assignments

1. Implement the following algorithms on standard datasets available on the web. The input will normally be in the form of sparse matrices representing the webgraph.

(a) Simple PageRank Algorithm

(b) PageRank algorithm with a teleportation factor to avoid dead-ends and spider traps.

2. Describe how you stored the connectivity matrix on disk and how you computed the transition matrix. List the top-10 pages as returned by the algorithms in each case.

3. Now rewrite portions of the code to implement the TrustRank algorithm. The user will specify which pages (indices) correspond to trustworthy pages. It might be good to look at the URLs and identify reasonable candidates for trustworthy pages.

4. Implement Assignments 1 and 3 using MapReduce.

5. Implement HITS algorithm any webgraph using MapReduce.

References

1. C.D. Manning, P. Raghavan, H. Schütze (2008). *Introduction to Information Retrieval.* Website: http://informationretrieval.org/; Cambridge University Press.

2. D. Easley, J. Kleinberg (2010). *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge University Press. Complete preprint on-line at http://www.cs.cornell.edu/home/kleinber/networks-book/.

3. Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry (1999) *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report. Stanford InfoLab.

4. T. Haveliwala. Efficient Computation of PageRank. Tech. rep., Stanford University, 1999.

5. T. Haveliwala (2002). Topic-sensitive PageRank. In *Proceedings of the Eleventh International Conference on World Wide Web*, 2002.

6. J. Kleinberg (1998). Authoritative Sources in a Hyperlinked Environment. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*.

7. A. Broder, R. Kumar, F. Maghoul et al. (2000). Graph structure in the Web, *Computer Networks,* 33:1–6, pp. 309–320.

8 Frequent Itemset Mining

LEARNING OBJECTIVES

After reading this chapter, you will be able to:

- Review your knowledge about frequent itemsets and basic algorithms to identify them.
- Learn about different memory efficient techniques to execute the traditional FIM algorithms.
- Understand how these algorithms are insufficient to handle larger datasets.
- Learn about the algorithm of Park, Chen and Yu, and its variants.
- Understand the sampling-based SON Algorithm and how it can be parallelized using MapReduce.
- Learn some simple stream-based frequent itemset mining methods.

8.1 Introduction

Frequent itemsets play an essential role in many data mining tasks where one tries to find interesting patterns from databases, such as association rules, correlations, sequences, episodes, classifiers, clusters and many more. One of the most popular applications of frequent itemset mining is discovery of association rules. The identification of sets of items, products, symptoms, characteristics and so forth that often occur together in the given database can be seen as one of the most basic tasks in data mining. This chapter discusses a host of algorithms that can be effectively used to mine frequent itemsets from very massive datasets.

This chapter begins with a conceptual description of the "market-basket" model of data. The problem of deriving associations from data was first introduced using the "market-basket" model of data, which is essentially a many-many relationship between two kinds of elements, called "items" and "baskets". The frequent-itemsets problem is that of finding sets of items that appear in (are related to) many of the same baskets.

The problem of finding frequent itemsets differs from the similarity search discussed in Chapter 5. In the frequent itemset scenario, we attempt to discover sets of items that are found in the same baskets frequently. Further we need the number of such buckets where these items appear together to

be sufficiently large so as to be statistically significant. In similarity search we searched for items that have a large fraction of their baskets in common, even if the absolute number of such baskets is small in number.

Many techniques have been invented to mine databases for frequent events. These techniques work well in practice on smaller datasets, but are not suitable for truly big data. Applying frequent itemset mining to large databases is a challenge. First of all, very large databases do not fit into main memory. For example consider the well-known *Apriori* algorithm, where frequency counting is achieved by reading the dataset over and over again for each size of candidate itemsets. Unfortunately, the memory requirements for handling the complete set of candidate itemsets blows up fast and renders Apriori-based schemes very inefficient to use on large data.

This chapter proposes several changes to the basic Apriori algorithm to render it useful for large datasets. These algorithms take into account the size of the main memory available.

Since exact solutions are costly and impractical to find in large data, a class of approximate algorithms is discussed which exploit parallelism, especially the Map–Reduce concept. This chapter also gives a brief overview of finding frequent itemsets in a data stream.

8.2 Market-Basket Model

"Market-Baskets" is an abstraction that models any many–many relationship between two concepts: "items" and "baskets". Since the term a "market-basket" has its origin in retail applications, it is sometimes called "transactions". Each basket consists of a set of items (an itemset), and usually we assume that the number of items in a basket is small – much smaller than the total number of items. The number of baskets is usually assumed to be very large, bigger than what can fit in main memory.

Items need not be "contained" in baskets. We are interested in the co-occurrences of items related to a basket, not vice-versa. For this purpose, we define basket data in general terms. Let $I = \{i_1, \ldots, i_k\}$ be a set of k elements, called items. Let $B = \{b_1, \ldots, b_n\}$ be a set of n subsets of I. We call each $b_i \subset I$ a basket of items. For example, in a retail market-basket application, the set I consists of the items stocked by a retail outlet and each basket is the set of purchases from one register transaction; on the other hand, in a document basket application, the set I contains all dictionary words and proper nouns, while each basket is a single document in the corpus and each basket consists of all words that occur in that document.

8.2.1 Frequent-Itemset Mining

Let $I = \{i_1, \ldots, i_k\}$ be a set of items. Let D, the task-relevant data, be a set of database transactions where each transaction T is a set of items such that $T \subseteq I$. Each transaction is associated with an

identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$.

A set of items is referred to as an itemset. An itemset that contains k items is a k-itemset. For example, consider a computer store with computer-related items in its inventory. The set {computer, anti-virus software, printer, flash-drive} is a 4-itemset. The occurrence frequency of an itemset is the number of transactions that contain the itemset. This is also known, simply, as the frequency, support count, or count of the itemset. We can call an itemset I a "frequent itemset" only if its support count is sufficiently large. We prescribe a *minimum support* s and any I which has support greater than or equal to s is a frequent itemset.

Example 1

Items = {milk (m), coke (c), pepsi (p), beer (b), juice (j)}
Minimum support $s = 3$
Transactions

1. T1 = {m, c, b}
2. T2 = {m, p, j}
3. T3 = {m, b}
4. T4 = {c, j}
5. T5 = {m, p, b}
6. T6 = {m, c, b, j}
7. T7 = {c, b, j}
8. T8 = {b, c}

Frequent itemsets: {m}, {c}, {b}, {j}, {m, b}, {c, b}, {j, c}.

8.2.2 Applications

A supermarket chain may have 10,000 different items in its inventory. Daily millions of customers will push their shopping carts ("market-baskets") to the checkout section where the cash register records the set of items they purchased and give out a bill. Each bill thus represents one market-basket or one transaction. In this scenario, the identity of the customer is not strictly necessary to get useful information from the data. Retail organizations analyze the market-basket data to learn what typical customers buy together.

Example 2

Consider a retail organization that spans several floors, where soaps are in floor 1 and items like towels and other similar goods are in floor 10. Analysis of the market-basket shows a large number of baskets containing both soaps and towels. This information can be used by the supermarket manager in several ways:

1. Apparently, many people walk from where the soaps are to where the towels is which means they have to move form floor 1, catch the elevator to move to floor 10. The manger could choose to put a small shelf in floor 1 consisting of an assortment of towels and some other bathing accessories that might also be bought along with soaps and towels, for example, shampoos, bath mats etc. Doing so can generate additional "on the spot" sales.
2. The store can run a sale on soaps and at the same time raise the price of towels (without advertising that fact, of course). People will come to the store for the cheap soaps, and many will need towels too. It is not worth the trouble to go to another store for cheaper towels, so they buy that too. The store makes back on towels what it loses on soaps, and gets more customers into the store.

While the relationship between soaps and towels seems somewhat obvious, market-basket analysis may identify several pairs of items that occur together frequently but the connections between them may be less obvious. For example, the analysis could show chocolates being bought with movie CDs. But we need some rules to decide when a fact about co-occurrence of sets of items can be useful. Firstly any useful set (need not be only pairs) of items must be bought by a large number of customers. It is not even necessary that there be any connection between purchases of the items, as long as we know that lots of customers buy them.

Example 3 Online Commerce

An E-Retail store like E-bay or Amazon.com offers several million different items for sale through its websites and also cater to millions of customers. While normal offline stores, such as the supermarket discussed above, can only make productive decisions when combinations of items are purchased by very large numbers of customers, online sellers have the means to tailor their offers even to a single customer. Thus, an interesting question is to find pairs of items that many customers have bought together. Then, if one customer has bought one of these items but not the other, it might be good for Amazon or E-bay to advertise the second item when this customer next logs in. We can treat the purchase data as a market-basket problem, where each "basket" is the set of items that one particular customer has ever bought. But there is another way online sellers can use the same data. This approach, called "collaborative filtering", finds sets of customers with similar purchase behavior. For example, these businesses look for pairs, or even larger sets, of customers who

Example 7

One example of a market-basket file could look like:

{23, 45, 11001} {13, 48, 92, 145, 222} {…

Here, the character "{" begins a basket and the character "}" ends it. The items in a basket are represented by integers, and are separated by commas.

Since such a file (Example 7) is typically large, we can use MapReduce or a similar tool to divide the work among many machines. But non-trivial changes need to be made to the frequent itemset counting algorithm to get the exact collection of itemsets that meet a global support threshold. This will be addressed in Section 8.4.3.

For now we shall assume that the data is stored in a conventional file and also that the size of the file of baskets is sufficiently large that it does not fit in the main memory. Thus, the principal cost is the time it takes to read data (baskets) from the disk. Once a disk block full of baskets is read into the main memory, it can be explored, generating all the subsets of size k. It is necessary to point out that it is logical to assume that the average size of a basket is small compared to the total number of all items. Thus, generating all the pairs of items from the market-baskets in the main memory should take less time than the time it takes to read the baskets from disk.

For example, if there are 25 items in a basket, then there are $\left[\binom{25}{2} = 300\right]$ pairs of items in the basket, and these can be generated easily in a pair of nested for-loops. But as the size of the subsets we want to generate gets larger, the time required grows larger; it takes approximately $n^k / k!$ time to generate all the subsets of size k for a basket with n items. So if k is very large then the subset generation time will dominate the time needed to transfer the data from the disk.

However, surveys have indicated that in most applications we need only small frequent itemsets. Further, when we do need the itemsets for a large size k, it is usually possible to eliminate many of the items in each basket as not able to participate in a frequent itemset, so the value of n reduces as k increases.

Thus, the time taken to examine each of the baskets can usually be assumed proportional to the size of the file. We can thus measure the running time of a frequent-itemset algorithm by the number of times each disk block of the data file is read. This, in turn, is characterized by the number of passes through the basket file that they make, and their running time is proportional to the product of the number of passes they make through the basket file and the size of that file.

Since the amount of data is fixed, we focus only on the number of passes taken by the algorithm. This gives us a measure of what the running time of a frequent-itemset algorithm will be.

8.3.2 Itemset Counting using Main Memory

For many frequent-itemset algorithms, main memory is the critical resource. An important computing step in almost all frequent-itemset algorithms is to maintain several different counts in each pass

of the data. For example, we might need to count the number of times that each pair of items occurs in baskets in pass 2. In the next pass along with maintaining the counts of 2-itemsets, we have to now compute frequency of 3-itemsets and so on. Thus, we need main memory space to maintain these counts.

If we do not have enough main memory to store each of the counts at any pass then adding 1 to count of any previous itemset may involve loading the relevant page with the counts from secondary memory. In the worst case, this swapping of pages may occur for several counts, which will result in thrashing. This would make the algorithm several orders of magnitude slower than if we were certain to find each count in main memory. In conclusion, we need counts to be maintained in the main memory. This sets a limit on how many items a frequent-itemset algorithm can ultimately deal with. This number of different things we can count is, thus, limited by the main memory.

The naive way of counting a frequent k-itemset is to read the file once and count in main memory the occurrences of each k-itemset. Let n be the number of items. The number of itemsets of size $1 \leq k \leq n$ is given by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Example 8

Suppose we need to count all pairs of items (2-itemset) in some step, and there are n items. We thus need space to store $n(n-1)/2$ pairs. **This algorithm will fail if (#items)^2 exceeds main memory.**

Consider an e-commerce enterprise like Amazon. The number of items can be around 100K or 10B (Web pages). Thus, assuming 10^5 items and counts are 4-byte integers, the number of pairs of items is

$$\frac{10^5(10^5-1)}{2} = 5*10^9$$

Therefore, $2*10^{10}$ (20 GB) of memory is needed. Thus, in general, if integers take 4 bytes, we require approximately $\mathbf{2n^2}$ bytes. If our machine has 2 GB, or 231 bytes of main memory, then we require $n \leq 2^{15}$ or approximately $n < 33{,}000$.

It is important to point out here that it is sufficient to focus on counting pairs, because the probability of an itemset being frequent drops exponentially with size while the number of itemsets grows more slowly with size. This argument is quite logical. The number of items, while possibly very large, is rarely so large we cannot count all the singleton sets in main memory at the same time. For larger sets like triples, quadruples, for frequent-itemset analysis to make sense, the result has to be a small number of sets, or these itemsets will lose their significance. Thus, in practice, the support threshold is set high enough that it is only a rare set that is k-frequent ($k \geq 2$). Thus, we expect to find more frequent pairs than frequent triples, more frequent triples than frequent quadruples, and so on. Thus, we can safely conclude that maximum main memory space is required for counting frequent pairs. We shall, thus, only concentrate on algorithms for counting pairs.

8.3.3 Approaches for Main Memory Counting

Before we can discuss approaches for counting of pairs in the main memory we have to first, discuss how items in the baskets are represented in the memory. As mentioned earlier, it is more space-efficient to represent items by consecutive positive integers from 1 to n, where n is the number of distinct items.

But items will mostly be names or strings of the form "pencil", "pen", "crayons", etc. We will, therefore, need a hash table that translates items as they appear in the file to integers. That is, each time we see an item in the file, we hash it. If it is already in the hash table, we can obtain its integer code from its entry in the table. If the item is not there, we assign it the next available number (from a count of the number of distinct items seen so far) and enter the item and its code into the table.

8.3.3.1 The Triangular-Matrix Method

Coding items as integers saves some space in the main memory. Now we need to store the pair counts efficiently. The main issue is that we should count a pair $\{i, j\}$ in only one place. One way is to order the pair so that $i < j$ and only use the entry $A[i, j]$ in a two-dimensional (2-D) array A. But half the array will be useless in this method. Thus, we need either the upper triangular matrix or the lower triangular matrix. A triangular matrix can be very efficiently stored in a one-dimensional (1-D) ***Triangular Array***.

Keep pair counts in lexicographic order:

1. $\{1, 2\}, \{1, 3\}, \ldots, \{1, n\}, \{2, 3\}, \{2, 4\}, \ldots, \{2, n\}, \ldots \{n - 2, n - 1\}, \{n - 2, n\}, \{n - 1, n\}$
2. Pair $\{i, j\}$ is at position $(i-1)(n-i/2)+j-1$

Figure 8.2 shows a sample of a triangular matrix storing a set of pairs.

	1,2	1,3	1,4	1,5
		2,3	2,4	2,5
			3,4	3,5
				4,5

Pair $\{i, j\}$ is at position $(i - 1)(n - i / 2) + j - i$

- **$\{1,2\}$: $0 + 2 - 1 = 1$**
- **$\{1,3\} = 0 + 3 - 1 = 2$**
- **$\{1,4\} = 0 + 4 - 1 = 3$**
- **$\{1,5\} = 0 + 5 - 1 = 4$**
- **$\{2,3\} = (2 - 1) * (5 - 2/2) + 3 - 2 = 5$**
- **$\{2,4\} = (2 - 1) * (5 - 2/2) + 4 - 2 = 6$**
- **$\{2,5\} = (2 - 1) * (5 - 2/2) + 5 - 2 = 7$**
- **$\{3,4\} = (3 - 1) * (5 - 3/2) + 4 - 2 = 8$**
- **$\{3,5\} = 9$**
- **$\{4,5\} = 10$**

Pair	1,2	1,3	1,4	1,5	2,3	2,4	2,5	3,4	3,5	4,5
Position	1	2	3	4	5	6	7	8	9	10

Figure 8.2 Triangular matrix and its equivalent 1-D representation.

8.3.3.2 The Triples Method

We can consider one more approach to storing counts that may be more useful when the fraction of the possible pairs of items that actually appear in some basket may not be too high.

We can use a hash table with pairs $\{i, j\}$, where $1 \leq i \leq j \leq n$, as keys and the counts as values. This results in triplets of the form $[i, j, c]$, meaning that the count of pair $\{i, j\}$ with $i < j$ is c.

Unlike the triangular matrix approach, in this approach we only store pairs with non-zero count value. If items and counts are represented by 4-byte integers, this approach requires 12 bytes for each pair with non-zero value plus some overhead for the hash table.

We call this approach the triples method of storing counts. Unlike the triangular matrix, the triples method does not require us to store anything if the count for a pair is 0. On the other hand, the triples method requires us to store three integers, rather than one, for every pair that does appear in some basket.

It is easy to see that the hash table approach requires less memory than the triangular matrix approach if less than one-third of possible pairs actually occur.

Let us assume that a supermarket has 100,000 items, which means that there are about 5 billion possible pairs. With the triangular matrix approach, we would need about 20 GB of main memory to hold the pair counts. If all possible pairs actually occur, the hash table approach requires about 60 GB of main memory. If only 1 billion of the possible pairs occur, however, the hash table approach would only require about 12 GB of main memory.

How about frequent triples? Since there are $[n(n-1)(n-2)]/6$ triples, the naive algorithm with the triangular matrix approach would require $[4n(n-1)(n-2)]/6$ bytes of main memory, which is more than 666 TB for the example above. For this reason, we want to avoid counting itemsets that will turn out to be infrequent at the end.

Example 9

Suppose there are 100,000 items and 10,000,000 baskets of 10 items each. Then the integer counts required by triangular matrix method are

$$\binom{1000}{2} = 5 \times 10^9 \text{ (approximately)}$$

On the other hand, the total number of pairs among all the baskets is

$$10^7 \binom{n}{k} = 4.5 \times 10^8$$

Even in the extreme case that every pair of items appeared only once, there could be only 4.5×10^8 pairs with non-zero counts. If we used the triples method to store counts, we would need only three times

that number of integers or 1.35×10^9 integers. Thus, in this case, the triples method will surely take much less space than the triangular matrix.

However, even if there were 10 or a 100 times as many baskets, it would be normal for there to be a sufficiently uneven distribution of items that we might still be better off using the triples method. That is, some pairs would have very high counts, and the number of different pairs that occurred in one or more baskets would be much less than the theoretical maximum number of such pairs.

8.3.4 Monotonicity Property of Itemsets

When n is large, a naive approach to generate and count the supports of all sets over the database cannot be achieved within a reasonable period of time. Typically most enterprises deal with thousands of items and thus 2^n, which is the number of subsets possible, is prohibitively high.

Instead, we could limit ourselves to those sets that occur at least once in the database by generating only those subsets of all transactions in the database. Of course, for large transactions, this number could still be too large. As an optimization, we could generate only those subsets of at most a given maximum size. This technique also suffers from massive memory requirements for even a medium sized database. Most other efficient solutions perform a more directed search through the search space. During such a search, several collections of *candidate sets* are generated and their supports computed until all frequent sets have been generated. Obviously, the size of a collection of candidate sets must not exceed the size of available main memory. Moreover, it is important to generate as few candidate sets as possible, since computing the supports of a collection of sets is a time-consuming procedure. In the best case, only the frequent sets are generated and counted. Unfortunately, this ideal is impossible in general. The main underlying property exploited by most algorithms is that support is monotone decreasing with respect to extension of a set.

Property 1 (Support Monotonicity): Given a database of transactions D over I and two sets X, $Y \subseteq I$. Then,

$$X, Y \subseteq I \Rightarrow \text{support}(Y) \leq \text{support}(X)$$

Hence, if a set is infrequent, all of its supersets must be infrequent, and vice versa, if a set is frequent, all of its subsets must be frequent too. In the literature, this monotonicity property is also called the *downward-closure property*, since the set of frequent sets is downward closed with respect to set inclusion. Similarly, the set of infrequent sets is upward closed.

The downward-closure property of support also allows us to compact the information about frequent itemsets. First, some definitions are given below:

1. An itemset is *closed* if none of its immediate itemset has the same count as the itemset.
2. An itemset is *closed frequent* if it is frequent and closed.
3. An itemset is *maximal frequent* if it is frequent and none of its immediate superset is frequent.

For example, assume we have items = {apple, beer, carrot} and the following baskets:

1. {apple, beer}
2. {apple, beer}
3. {beer, carrot}
4. {apple, beer, carrot}
5. {apple, beer, carrot}

Assume the support threshold $s = 3$.

Table 8.4 Indicating Frequent, Closed and Maximal Itemsets

Itemset	*Count*	*Frequent?*	*Closed?*	*Closed Freq?*	*Max Freq?*
{apple}	4	Yes	No	No	No
{beer}	5	Yes	Yes	Yes	No
{carrot}	3	Yes	No	No	No
{apple, beer}	4	Yes	Yes	Yes	Yes
{apple, carrot}	2	No	No	No	No
{beer, carrot}	3	Yes	Yes	Yes	Yes
{apple, beer, carrot}	2	No	Yes	No	No

From Table 8.4, we see that there are five frequent itemsets, of which only three are closed frequent, of which in turn only two are maximal frequent. The set of all maximal frequent itemsets is a subset of the set of all closed frequent itemsets, which in turn is a subset of the set of all frequent itemsets. Thus, maximal frequent itemsets is the most compact representation of frequent itemsets. In practice, however, closed frequent itemsets may be preferred since they also contain not just the frequent itemset information, but also the exact count.

8.3.5 The Apriori Algorithm

In this section we shall we concentrate on finding the frequent pairs only. We have previously discussed that counting of pairs is indeed a more memory-intensive task. If we have enough main memory to count all pairs, using either triangular matrix or triples, then it is a simple matter to read the file of baskets in a single pass. For each basket, we use a double loop to generate all the pairs. Each time we

generate a pair, we add 1 to its count. At the end, we examine all pairs to see which have counts that are equal to or greater than the support threshold *s*; these are the frequent pairs.

However, this naive approach fails if there are too many pairs of items to count them all in the main memory. The Apriori algorithm which is discussed in this section uses the monotonicity property to reduce the number of pairs that must be counted, at the expense of performing two passes over data, rather than one pass.

The Apriori algorithm for finding frequent pairs is a two-pass algorithm that limits the amount of main memory needed by using the downward-closure property of support to avoid counting pairs that will turn out to be infrequent at the end.

Let *s* be the minimum support required. Let *n* be the number of items. In the first pass, we read the baskets and count in main memory the occurrences of each item. We then remove all items whose frequency is lesser than *s* to get the set of frequent items. This requires memory proportional to *n*.

In the second pass, we read the baskets again and count in main memory only those pairs where both items are frequent items. This pass will require memory proportional to square of **frequent** items only (for counts) plus a list of the frequent items (so you know what must be counted). Figure 8.3 indicates the main memory in the two passes of the Apriori algorithm.

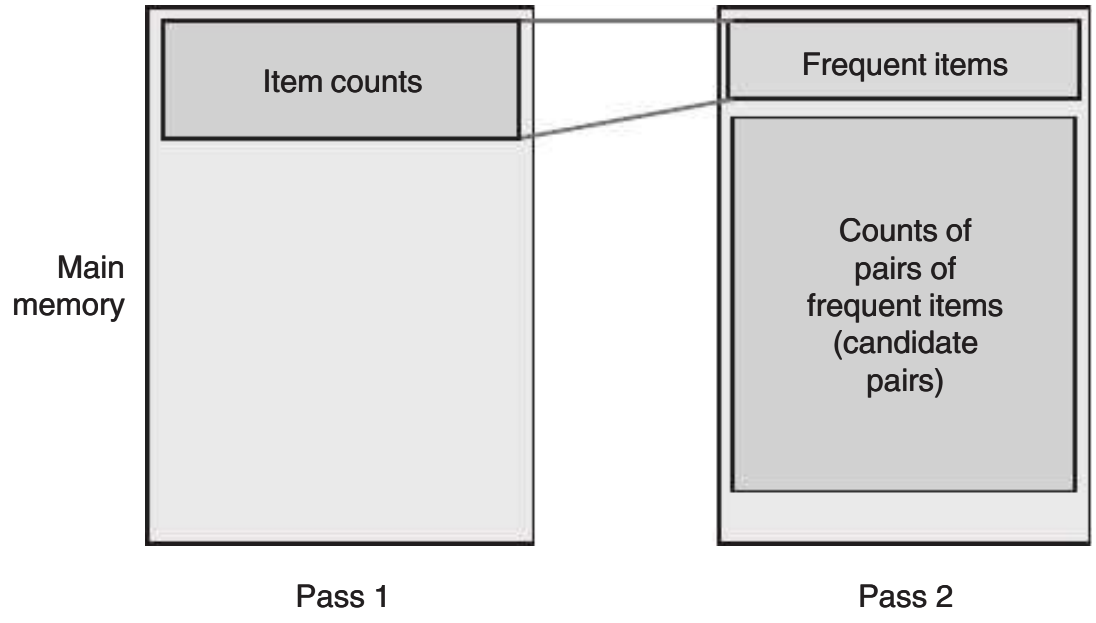


Figure 8.3 Main memory map in Apriori algorithm.

8.3.5.1 The First Pass of Apriori

Two tables are created in this pass. The first table, if necessary, translates item names into integers from 1 to *n*, as it will be optimal to use numbers than long varying length string names. The other table is an array of counts; the i^{th} array element counts the occurrences of the item numbered *i*.

Initially, the counts for all the items are 0. As we read baskets, we look at each item in the basket and translate its name into an integer. Next, we use that integer to index into the array of counts, and we add 1 to the integer found there.

After the first pass, we examine the counts of the items to determine which of them are frequent as singletons. It will normally be the case that the number of frequent singletons will definitely be much lesser than the number of items. Thus, we employ a trick for the next pass. For the second pass of Apriori, we create a new numbering from 1 to m for just the frequent items. This table is an array indexed 1 to n, and the entry for i is either 0, if item i is not frequent, or a unique integer in the range 1 to m if item i is frequent. We shall refer to this table as the frequent-items table.

8.3.5.2 The Second Pass of Apriori

During the second pass, we count all the pairs that consist of two frequent items. The space required on the second pass is $2m^2$ bytes, rather than $2n^2$ bytes, if we use the triangular- matrix method for counting. Notice that the renumbering of just the frequent items is necessary if we are to use a triangular matrix of the right size. The complete set of main-memory structures used in the first and second passes of this improved Apriori algorithm is shown in Fig. 8.4.

Let us quantify the benefit of eliminating infrequent items; if only half the items are frequent we need one quarter of the space to count. Likewise, if we use the triples method, we need to count only those pairs of two frequent items that occur in at least one basket.

The mechanics of the second pass are as follows:

1. For each basket, look in the frequent-items table to see which of its items are frequent.
2. In a double loop, generate all pairs of frequent items in that basket.
3. For each such pair, add one to its count in the data structure used to store counts.

Finally, at the end of the second pass, examine the structure of counts to determine which pairs are frequent.

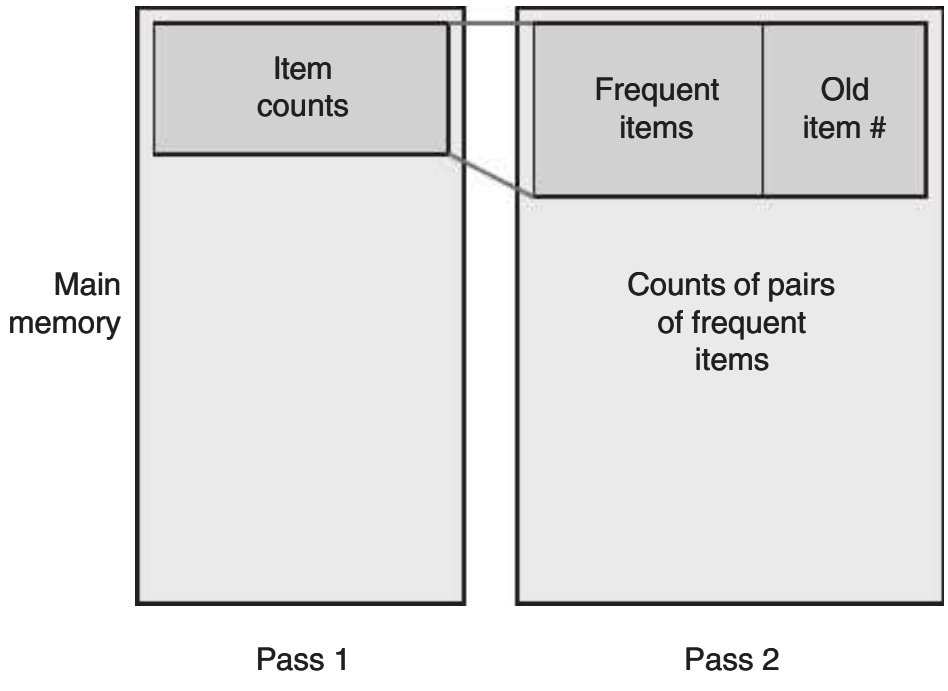


Figure 8.4 Improved Apriori algorithm.

8.3.5.3 Finding All Frequent Itemsets

The same technique as used for finding frequent pairs without counting all pairs can be extended to find larger frequent itemsets without an exhaustive count of all sets. In the Apriori algorithm, one pass is taken for each set-size k. If no frequent itemsets of a certain size are found, then monotonicity tells us there can be no larger frequent itemsets, so we can stop. The pattern of moving from one size k to the next size $k + 1$ can be summarized as follows. For each size k, there are two sets of itemsets:

1. C_k is the set of candidate itemsets of size k – the itemsets that we must count in order to determine whether they are in fact frequent.
2. L_k is the set of truly frequent itemsets of size k.

The pattern of moving from one set to the next and one size to the next is depicted in Fig. 8.5.

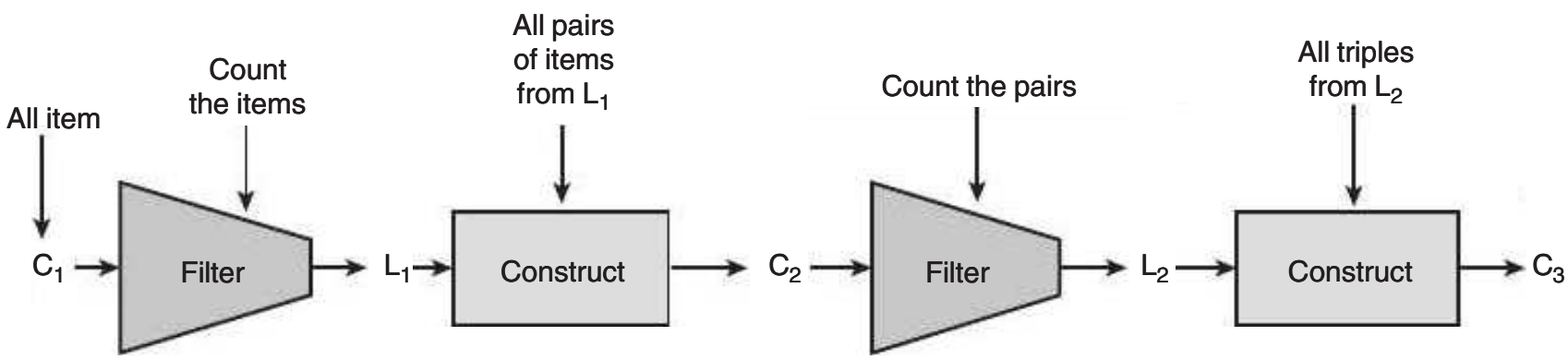


Figure 8.5 Candidate generation: General Apriori algorithm.

Example 10

Let $C_1 = \{ \{b\} \{c\} \{j\} \{m\} \{n\} \{p\} \}$. Then

1. Count the support of itemsets in C_1
2. Prune non-frequent: $L_1 = \{ b, c, j, m \}$
3. Generate $C_2 = \{ \{b,c\} \{b,j\} \{b,m\} \{c,j\} \{c,m\} \{j,m\} \}$
4. Count the support of itemsets in C_2
5. Prune non-frequent: $L_2 = \{ \{b,m\} \{b,c\} \{c,m\} \{c,j\} \}$
6. Generate $C_3 = \{ \{b,c,m\} \{b,c,j\} \{b,m,j\} \{c,m,j\} \}$
7. Count the support of itemsets in C_3
8. Prune non-frequent: $L_3 = \{ \{b,c,m\} \}$

Example 11

Assume we have items = {*a, b, c, d, e*} and the following baskets:

1. {*a, b*}
2. {*a, b, c*}
3. {*a, b, d*}
4. { *b, c, d*}
5. {*a, b, c, d*}
6. {*a, b, d, e*}

Let the support threshold $s = 3$. The Apriori algorithm passes as follows:

1.
   - **(a)** Construct $C_1 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$.
   - **(b)** Count the support of itemsets in C_1.
   - **(c)** Remove infrequent itemsets to get $L_1 = \{\{a\}, \{b\}, \{c\}, \{d\}\}$.
2.
   - **(a)** Construct $C_2 = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}\}$.
   - **(b)** Count the support of itemsets in C_2.
   - **(c)** Remove infrequent itemsets to get $L_2 = \{\{a, b\}, \{a, d\}, \{b, c\}, \{b, d\}\}$.
3.
   - **(a)** Construct $C_3 = \{\{a, b, c\}, \{a, b, d\}, \{b, c, d\}\}$. Note that we can be more careful here with the rule generation. For example, we know {*b, c, d*} cannot be frequent since {*c, d*} is not frequent. That is, {*b, c, d*} should not be in C_3 since {*c, d*} is not in L_2.
   - **(b)** Count the support of itemsets in C_3.
   - **(c)** Remove infrequent itemsets to get $L_3 = \{\{a, b, d\}\}$.
4. Construct C_4 = {empty set}.

8.4 Handling Larger Datasets in Main Memory

We can clearly understand that the initial candidate set generation especially for the large 2-itemsets is the key to improve the performance of the Apriori algorithm. This algorithm is fine as long as the step for the counting of the candidate pairs has enough memory that it can be accomplished without excessive moving of data between disk and main memory.

Another performance related issue is the amount of data that has to be scanned during large itemset discovery. A straightforward implementation would require one pass over the database of all transactions for each iteration. Note that as k increases, not only is there a smaller number of large k itemsets, but

there are also fewer transactions containing any large k itemsets. Reducing the number of transactions to be scanned and trimming the number of items in each transaction can improve the data mining efficiency in later stages.

Several algorithms have been proposed to cut down on the size of candidate set C_2. In this section, we discuss the Park–Chen–Yu (PCY) algorithm, which takes advantage of the fact that in the first pass of Apriori there is typically lots of main memory not needed for the counting of single items. Then we look at the Multistage algorithm, which uses the PCY trick and also inserts extra passes to further reduce the size of C_2.

8.4.1 Algorithm of Park–Chen–Yu

In the Apriori algorithm, in each pass we use the set of large itemsets L_i to form the set of candidate large itemsets C_{i+1} by joining L_i with L_i on $(i-1)$ common items for the next pass. We then scan the database and count the support of each item set in C_{i+1} so as to determine L_{i+1}. As a result, in general, the more itemsets in C_i the higher the processing cost for determining L_i will be. An algorithm designed by Park, Chen, and Yu called as DHP (standing for direct hashing and pruning) is so designed that it will reduce the number of itemsets to be explored in C_i in initial iterations significantly. The corresponding processing cost to determine L_i from C_i is, therefore, reduced. This algorithm is also popularly known as PCY algorithm in literature, named so because of its authors.

In essence, the PCY algorithm uses the technique of hashing to filter out unnecessary itemsets for next candidate itemset generation. When the support of candidate k-itemsets is counted by scanning the database, PCY accumulates information about candidate $(k+1)$-itemsets in advance in such a way that all possible $(k+1)$-itemsets of each transaction after some pruning are hashed to a hash table. Each bucket in the hash table consists of a number to represent how many itemsets have been hashed to this bucket thus far. We note that based on the resulting hash table a bit vector can be constructed, where the value of one bit is set to 1 if the number in the corresponding entry of the hash table is greater than or equal to s, the minimum support value. This bit vector can be used later to greatly reduce the number of itemsets in C_i.

The PCY algorithm does need any extra space to store the hash tables or the bit vectors. It exploits the observation that there may be much unused space in main memory on the first pass. If there are a million items and gigabytes of main memory, which is quite typical these days, we do not need more than 10% of the main memory for the two tables, one a translation table from item names to small integers and second an array to count those integers.

The PCY algorithm then uses the still available space for an unusual type of hash table. The buckets of this hash table store integers rather than sets of keys. Pairs of items are hashed to buckets of this hash table. As we examine a basket during the first pass, we not only add 1 to the count for each item in the basket, but also generate all possible pairs (2-itemsets). We hash each pair using a hash function, and we add 1 to the bucket into which that pair hashes. Note that the pair itself does not go into the bucket; the pair only affects the single integer in the bucket.

Algorithm

FOR (each basket):

FOR (each item in the basket) :

add 1 to item's count;

FOR (each pair of items):

{ hash the pair to a bucket;

add 1 to the count for that bucket;}

At the end of the first pass, each bucket has a count, which is the sum of the counts of all the pairs that hash to that bucket. If the count of a bucket is at least as great as the support threshold s, it is called a frequent bucket. We can say nothing about the pairs that hash to a frequent bucket; they could all be frequent pairs from the information available to us. But if the count of the bucket is less than s (an infrequent bucket), we know no pair that hashes to this bucket can be frequent, even if the pair consists of two frequent items. This fact gives us an advantage on the second pass.

In Pass 2 we only count pairs that hash to frequent buckets.

Algorithm

Count all pairs ***{i, j}*** that meet the conditions for being a **candidate pair**:

1. Both ***i*** and ***j*** are frequent items

2. The pair ***{i, j}*** hashes to a bucket whose bit in the bit vector is **1** (i.e., a **frequent bucket**)

Both the above conditions are necessary for the pair to have a chance of being frequent. Figure 8.6 indicates the memory map.

Depending on the data and the amount of available main memory, there may or may not be a benefit in using the hash table on pass 1. In the worst case, all buckets are frequent, and the PCY algorithm counts exactly the same pairs as Apriori does on the second pass. However, typically most of the buckets will be infrequent. In that case, PCY reduces the memory requirements of the second pass.

Let us consider a typical situation. Suppose we have 1 GB of main memory available for the hash table on the first pass. Let us say the dataset has a billion baskets, each with 10 items. A bucket is an

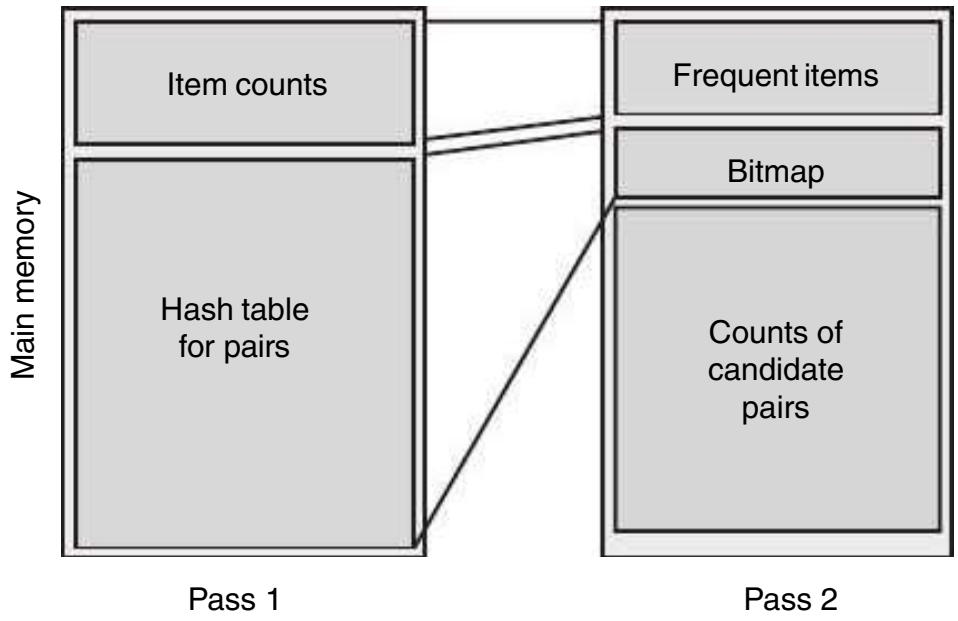


Figure 8.6 Memory map in PCY algorithm.

integer, typically 4 bytes, so we can maintain a quarter of a billion buckets. The number of pairs in all the baskets is

$$10^9 \times \binom{10}{2} = 4.5 \times 10^{10} \text{ pairs}$$

This number is also the sum of the counts in the buckets. Thus, the average count is about 180. If the support threshold s is around 180 or less, we might expect few buckets to be infrequent. However, if s is much larger, say 1000, then it must be that the great majority of the buckets are infrequent. The greatest possible number of frequent buckets is, thus, about 45 million out of the 250 million buckets.

Between the passes of PCY, the hash table is reduced to a bitmap, with one bit for each bucket. The bit is 1 if the bucket is frequent and 0 if it is not. Thus, integers of 4 bytes are replaced by single bits. Thus, the bitmap occupies only 1/32 of the space that would otherwise be available to store counts. However, if most buckets are infrequent, we expect that the number of pairs being counted on the second pass will be much smaller than the total number of pairs of frequent items. Thus, PCY can handle large datasets without thrashing during the second pass, while the Apriori algorithm would have run out of main memory space resulting in thrashing.

There is one more issue that could affect the space requirement of the PCY algorithm in the second pass. In the PCY algorithm, the set of candidate pairs is sufficiently irregular, and hence we cannot use the triangular-matrix method for organizing counts; we must use a table of counts. Thus, it does not make sense to use PCY unless the number of candidate pairs is reduced to at most one-third of all possible pairs. Passes of the PCY algorithm after the second can proceed just as in the Apriori algorithm, if they are needed.

Further, in order for PCY to be an improvement over Apriori, a good fraction of the buckets on the first pass must not be frequent. For if most buckets are frequent, the algorithm does not eliminate many

pairs. Any bucket to which even one frequent pair hashes will itself be frequent. However, buckets to which no frequent pair hashes could still be frequent if the sum of the counts of the pairs that do hash there exceeds the threshold s.

To a first approximation, if the average count of a bucket is less then s, we can expect at least half the buckets not to be frequent, which suggests some benefit from the PCY approach. However, if the average bucket has a count above s, then most buckets will be frequent.

Suppose the total number of occurrences of pairs of items among all the baskets in the dataset is P. Since most of the main memory M can be devoted to buckets, the number of buckets will be approximately $M/4$. The average count of a bucket will then be $4P/M$. In order that there be many buckets that are not frequent, we need

$$\frac{4P}{M} < s \text{ or } M > \frac{4p}{s}$$

An example illustrates some of the steps of the PCY algorithm.

Example 12

Given: Database D; minimum support = 2 and the following data.

TID	*Items*
1	1,3,4
2	2,3,5
3	1,2,3,5
4	2,5

Pass 1:

Step 1: Scan D along with counts. Also form possible pairs and hash them to the buckets. For example, {1,3}:2 means pair {1,3} hashes to bucket 2.

Itemset	*Sup*
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

```
for itemset in itemsets:
if supp(itemset) >= p*s
emit(itemset, null)
reduce(key, values):
emit(key, null)
```

The second map-reduce step:

```
  map(key, value):
// value is the candidate itemsets and a chunk
// of the full dataset
Count occurrences of itemsets in the chunk.
for itemset in itemsets:
    emit(itemset, supp(itemset))

  reduce(key, values):
    result = 0
    for value in values:
        result += value
    if result >= s:
        emit(key, result)
```

8.5.4 Toivonen's Algorithm

Toivonen's algorithm makes only one full pass over the database. The idea is to pick a random sample, use it to determine all association rules that probably hold in the whole database, and then to verify the results with the rest of the database. The algorithm thus produces exact association rules in one full pass over the database. In those rare cases where the sampling method does not produce all association rules, the missing rules can be found in a second pass.

The algorithm will give neither false negatives nor positives, but there is a small yet non-zero probability that it will fail to produce any answer at all. In that case, it needs to be repeated until it gives an answer. However, the average number of passes needed before it produces all and only the frequent itemsets is a small constant.

Toivonen's algorithm begins by selecting a small sample of the input dataset, and finding from it the candidate frequent itemsets. It uses the random sampling algorithm, except that it is essential that

the threshold be set to something less than its proportional value. That is, if the support threshold for the whole dataset is s, and the sample size is fraction p, then when looking for frequent itemsets in the sample, use a threshold such as $0.9ps$ or $0.7ps$. The smaller we make the threshold, the more main memory we need for computing all itemsets that are frequent in the sample, but the more likely we are to avoid the situation where the algorithm fails to provide an answer.

Having constructed the collection of frequent itemsets for the sample, we next construct the negative border. This is the collection of itemsets that are not frequent in the sample, but all of their immediate subsets (subsets constructed by deleting exactly one item) are frequent in the sample.

We shall define the concept of "negative border" before we explain the algorithm.

The negative border with respect to a frequent itemset, S, and set of items, I, is the minimal itemsets contained in PowerSet (I) and not in S. The basic idea is that the negative border of a set of frequent itemsets contains the closest itemsets that could also be frequent. Consider the case where a set X is not contained in the frequent itemsets. If all subsets of X are contained in the set of frequent itemsets, then X would be in the negative border. We illustrate this with the following example.

Example 14

Consider the set of items $I = \{A, B, C, D, E\}$ and let the combined frequent itemsets of size 1 to 3 be $S = \{\{A\}, \{B\}, \{C\}, \{D\}, \{AB\}, \{AC\}, \{BC\}, \{AD\}, \{CD\}\{ABC\}\}$

1. The negative border is $\{\{E\}, \{BD\}, \{ACD\}\}$.
2. The set $\{E\}$ is the only 1-itemset not contained in S.
3. $\{BD\}$ is the only 2-itemset not in S but whose 1-itemset subsets are.
4. $\{ACD\}$ is the only 3-itemset whose 2-itemset subsets are all in S.

The negative border is important since it is necessary to determine the support for those itemsets in the negative border to ensure that no large itemsets are missed from analyzing the sample data. Support for the negative border is determined when the remainder of the database is scanned. If we find that an itemset X in the negative border belongs in the set of all frequent itemsets, then there is a potential for a superset of X to also be frequent. If this happens, then a second pass over the database is needed to make sure that all frequent itemsets are found.

To complete Toivonen's algorithm, we make a pass through the entire dataset, counting all the itemsets that are frequent in the sample or are in the negative border. There are two possible outcomes.

Outcome 1: No member of the negative border is frequent in the whole dataset. In this case, we already have the correct set of frequent itemsets, which were found in the sample and also were found to be frequent in the whole.

> **Outcome 2:** Some member of the negative border is frequent in the whole. Then it is difficult to decide whether there exists more larger sets that may become frequent when we consider a more larger sample. Thus, the algorithm terminates without a result. The algorithm must be repeated with a new random sample.

It is easy to see that the Toivonen's algorithm never produces a false positive, since it only ouputs those itemsets that have been counted and found to be frequent in the whole. To prove that it never produces a false negative, we must show that when no member of the negative border is frequent in the whole, then there can be no itemset whatsoever, that is:

Frequent in the whole, but in neither the negative border nor the collection of frequent itemsets for the sample.

Suppose the above is not true. This means, there is a set S that is frequent in the whole, but not in the negative border and not frequent in the sample. Also, this round of Toivonen's algorithm produced an answer, which would certainly not include S among the frequent itemsets. By monotonicity, all subsets of S are also frequent in the whole.

Let T be a subset of S that is of the smallest possible size among all subsets of S that are not frequent in the sample. Surely T meets one of the conditions for being in the negative border: It is not frequent in the sample. It also meets the other condition for being in the negative border: Each of its immediate subsets is frequent in the sample. For if some immediate subset of T were not frequent in the sample, then there would be a subset of S that is smaller than T and not frequent in the sample, contradicting our selection of T as a subset of S that was not frequent in the sample, yet as small as any such set.

Now we see that T is both in the negative border and frequent in the whole dataset. Consequently, this round of Toivonen's algorithm did not produce an answer.

8.6 Counting Frequent Items in a Stream

We have seen in Chapter 6 that recently there has been much interest in data arriving in the form of continuous and infinite data streams, which arise in several application domains like high-speed networking, financial services, e-commerce and sensor networks.

We have seen that data streams possess distinct computational characteristics, such as unknown or unbounded length, possibly very fast arrival rate, inability to backtrack over previously arrived items (only one sequential pass over the data is permitted), and a lack of system control over the order in which the data arrive. As data streams are of unbounded length, it is intractable to store the entire data into main memory.

Finding frequent itemsets in data streams lends itself to many applications of Big Data. In many such applications, one is normally interested in the frequent itemsets in the recent period of time.

pairs, doing so wastes half the space. Thus, we use the single dimension representation of the triangular matrix. If fewer than one-third of the possible pairs actually occur in baskets, then it is more space-efficient to store counts of pairs as triples (i, j, c), where c is the count of the pair $\{i, j\}$ and $i < j$. An index structure such as a hash table allows us to find the triple for (i, j) efficiently.

- **Monotonicity of frequent itemsets:** An important property of itemsets is that if a set of items is frequent, then so are all its subsets. We exploit this property to eliminate the need to count certain itemsets by using its contrapositive: If an itemset is not frequent, then neither are its supersets.

- **The Apriori algorithm for pairs:** We can find all frequent pairs by making two passes over the baskets. On the first pass, we count the items themselves and then determine which items are frequent. On the second pass, we count only the pairs of items both of which are found frequent on the first pass. Monotonicity justifies our ignoring other pairs.

- **The PCY algorithm:** This algorithm improves on Apriori by creating a hash table on the first pass, using all main-memory space that is not needed to count the items. Pairs of items are hashed, and the hash-table buckets are used as integer counts of the number of times a pair has hashed to that bucket. Then, on the second pass, we only have to count pairs of frequent items that hashed to a frequent bucket (one whose count is at least the support threshold) on the first pass.

- **The multistage and multihash algorithm:** These are extensions to the PCY algorithm by inserting additional passes between the first and second pass of the PCY algorithm to hash pairs to other, independent hash tables. Alternatively, we can modify the first pass of the PCY algorithm to divide available main memory into several hash tables. On the second pass, we only have to count a pair of frequent items if they hashed to frequent buckets in all hash tables.

- **Randomized algorithms:** Instead of making passes through all the data, we may choose a random sample of the baskets, small enough that it is possible to store both the sample and the needed counts of itemsets in the main memory. While this method uses at most one pass through the whole dataset, it is subject to false positives (itemsets that are frequent in the sample but not the whole) and false negatives (itemsets that are frequent in the whole but not the sample).

- **The SON algorithm:** This algorithm divides the entire file of baskets into segments small enough that all frequent itemsets for the segment can be found in main memory. Candidate itemsets are those found frequent for at least one segment. A second pass allows us to count all the candidates and find the exact collection of frequent itemsets. This algorithm is especially appropriate for a MapReduce setting making it ideal for Big Data.

- **Toivonen's algorithm:** This algorithm improves on the random sampling algorithm by avoiding both false positives and negatives. To achieve this, it searches for frequent itemsets in a sample, but with the threshold lowered so there is little chance of missing an itemset that is frequent in the whole. Next, we examine the entire file of baskets, counting not only the itemsets that are frequent in

the sample, but also the negative border. If no member of the negative border is found frequent in the whole, then the answer is exact. But if a member of the negative border is found frequent, then the whole process has to repeat with another sample.

- **Frequent itemsets in streams:** We present overview of a few techniques that can be used to count frequent items in a stream. We also present a few techniques that use the concept of decaying windows for finding more recent frequent itemsets.

Exercises

1. Imagine there are 100 baskets, numbered 1, 2, …, 100, and 100 items, similarly numbered. Item i is in basket j if and only if i divides j evenly. For example, basket 24 is the set of items {1, 2, 3, 4, 6, 8, 12, 24}. Describe all the association rules that have 100% confidence.

2. Suppose we have transactions that satisfy the following assumptions:

- The support threshold s is 10,000.
- There are one million items, which are represented by the integers 0, 1, ..., 999999.
- There are N frequent items, that is, items that occur 10,000 times or more.
- There are one million pairs that occur 10,000 times or more.
- There are $2M$ pairs that occur exactly once. M of these pairs consist of two frequent items, the other M each have at least one non-frequent item.
- No other pairs occur at all.
- Integers are always represented by 4 bytes.

Suppose we run the Apriori algorithm to find frequent pairs and can choose on the second pass between the triangular-matrix method for counting candidate pairs (a triangular array count[i][j] that holds an integer count for each pair of items (i, j) where $i < j$) and a hash table of item-item-count triples. In the first case neglect the space needed to translate between original item numbers and numbers for the frequent items, and in the second case neglect the space needed for the hash table. Assume that item numbers and counts are always 4-byte integers.

As a function of N and M, what is the minimum number of bytes of main memory needed to execute the Apriori algorithm on this data?

3. If we use a triangular matrix to count pairs, and n, the number of items, is 20, what pair's count is in a [100]?

4. Let there be I items in a market-basket dataset of B baskets. Suppose that every basket contains exactly K items. As a function of I, B, and K:

(a) How much space does the triangular-matrix method take to store the counts of all pairs of items, assuming four bytes per array element?

(b) What is the largest possible number of pairs with a non-zero count?

(c) Under what circumstances can we be certain that the triples method will use less space than the triangular array?

5. Imagine that there are 1100 items, of which 100 are "big" and 1000 are "little". A basket is formed by adding each big item with probability 1/10, and each little item with probability 1/100. Assume the number of baskets is large enough that each itemset appears in a fraction of the baskets that equals its probability of being in any given basket. For example, every pair consisting of a big item and a little item appears in 1/1000 of the baskets. Let s be the support threshold, but expressed as a fraction of the total number of baskets rather than as an absolute number. Give, as a function of s ranging from 0 to 1, the number of frequent items on Pass 1 of the Apriori algorithm. Also, give the number of candidate pairs on the second pass.

6. Consider running the PCY algorithm on the data of Exercise 5, with 100,000 buckets on the first pass. Assume that the hash function used distributes the pairs to buckets in a conveniently random fashion. Specifically, the 499,500 little–little pairs are divided as evenly as possible (approximately 5 to a bucket). One of the 100,000 big–little pairs is in each bucket, and the 4950 big–big pairs each go into a different bucket.

(a) As a function of s, the ratio of the support threshold to the total number of baskets (as in Exercise 5), how many frequent buckets are there on the first pass?

(b) As a function of s, how many pairs must be counted on the second pass?

7. Here is a collection of 12 baskets. Each contains three of the six items 1 through 6. {1,2,3} {2,3,4} {3,4,5} {4,5,6} {1,3,5} {2,4,6} {1,3,4} {2,4,5} {3,5,6} {1,2,4} {2,3,5} {3,4,6}. Suppose the support threshold is 3. On the first pass of the PCY algorithm we use a hash table with 10 buckets, and the set $\{i, j\}$ is hashed to bucket $i \times j$ mod 10.

(a) By any method, compute the support for each item and each pair of items.

(b) Which pairs hash to which buckets?

(c) Which buckets are frequent?

(d) Which pairs are counted on the second pass of the PCY algorithm?

8. Suppose we run the Multistage algorithm on the data of Exercise 7, with the same support threshold of 3. The first pass is the same as in that exercise, and for the second pass, we hash pairs to nine buckets, using the hash function that hashes $\{i, j\}$ to bucket $i + j$ mod 8. Determine the counts of the buckets on the second pass. Does the second pass reduce the set of candidate pairs?

9. Suppose we run the Multihash algorithm on the data of Exercise 7. We shall use two hash tables with five buckets each. For one, the set $\{i, j\}$ is hashed to bucket $2i + 3j + 4$ mod 5, and for the other, the set is hashed to $i + 4j$ mod 5. Since these hash functions are not symmetric in i and j, order the items so that $i < j$ when evaluating each hash function. Determine the counts of each of the 10 buckets. How large does the support threshold have to be for the Multistage algorithm to eliminate more pairs than the PCY algorithm would, using the hash table and function described in Exercise 7?

10. During a run of Toivonen's algorithm with set of items {A,B,C,D,E,F,G,H} a sample is found to have the following maximal frequent itemsets: {A,B}, {A,C}, {A,D}, {B,C}, {E}, {F}. Compute the negative border. Then, identify in the list below the set that is NOT in the negative border.

are sometimes represented by more complicated data structures than the vectors of attributes. Good examples include text documents, images, or graphs. Determining the similarity (or differences) of two objects in such a situation is more complicated, but if a reasonable similarity (dissimilarity) measure exists, then a clustering analysis can still be performed. Such measures which also were discussed in Chapter 5 include the Jaccard distance, Cosine distance, Hamming distance, and Edit distance.

Example 1 **Clustering Problem: Books**

1. **Intuitively: Books are divided into categories, and customers prefer a few categories**

   Represent a book by a set of customers who bought it:

   Similar books have similar sets of customers, and vice-versa

2. **Space of all books:**

   Think of a space with one dimension for each customer

   Values in a dimension may be 0 or 1 only

   A book is a point in this space $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ iff the i^{th} customer bought the CD

   For an online bookseller Amazon, this dimension is tens of millions

3. **Task:** Find clusters of similar CDs

Example 2 **Clustering Problem: Documents**

Finding topics:

1. Represent a document by a vector $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ iff the i^{th} word (in some order) appears in the document
2. Documents with similar sets of words may be about the same topic
3. We have a choice when we think of documents as sets of words:
   - *Sets as vectors*: Measure similarity by the cosine distance
   - *Sets as sets*: Measure similarity by the Jaccard distance
   - *Sets as points*: Measure similarity by Euclidean distance

9.2.2 Clustering Applications

Some common application domains in which the clustering problem arises are as follows:

1. **Intermediate step for other fundamental data mining problems:** Since a clustering can be considered a form of data summarization, it often serves as a key intermediate step for many fundamental data mining problems, such as classification or outlier analysis. A compact summary of the data is often useful for different kinds of application-specific insights.
2. **Collaborative filtering:** Incollaborative filtering methods of clustering provides a summarization of like-minded users. The ratings provided by the different users for each other are used in order to perform the collaborative filtering. This can be used to provide recommendations in a variety of applications.
3. **Customer segmentation:** This application is quite similar to collaborative filtering, since it creates groups of similar customers in the data. The major difference from collaborative filtering is that instead of using rating information, arbitrary attributes about the objects may be used for clustering purposes.
4. **Data summarization:** Many clustering methods are closely related to dimensionality reduction methods. Such methods can be considered as a form of data summarization. Data summarization can be helpful in creating compact data representations that are easier to process and interpret in a wide variety of applications.
5. **Dynamic trend detection:** Many forms of dynamic and streaming algorithms can be used to perform trend detection in a wide variety of social networking applications. In such applications, the data is dynamically clustered in a streaming fashion and can be used in order to determine the important patterns of changes. Examples of such streaming data could be multi-dimensional data, text streams, streaming time-series data, and trajectory data. Key trends and events in the data can be discovered with the use of clustering methods.
6. **Multimedia data analysis:** A variety of different kinds of documents, such as images, audio, or video, fall in the general category of multimedia data. The determination of similar segments has numerous applications, such as the determination of similar snippets of music or similar photographs. In many cases, the data may be multi-modal and may contain different types. In such cases, the problem becomes even more challenging.
7. **Biological data analysis:** Biological data has become pervasive in the last few years, because of the success of the human genome effort and the increasing ability to collecting different kinds of gene expression data. Biological data is usually structured either as sequences or as networks. Clustering algorithms provide good ideas of the key trends in the data, as well as the unusual sequences.
8. **Social network analysis:** In these applications, the structure of a social network is used in order to determine the important communities in the underlying network. Community detection has important applications in social network analysis, because it provides an important understanding of the community structure in the network. Clustering also has applications to social network summarization, which is useful in a number of applications.

The above-mentioned list of applications represents a good cross-section of the wide diversity of problems that can be addressed with the clustering algorithms.

9.2.3 Clustering Strategies

Distinction in clustering approaches, that is between hierarchical and partitional approaches, is highlighted below:

1. **Hierarchical techniques** produce a nested arrangement of partitions, with a single cluster at the top consisting of all data points and singleton clusters of individual points at the bottom. Each intermediate level can be viewed as combining (splitting) two clusters from the next lower (next higher) level. Hierarchical clustering techniques which start with one cluster of all the points and then keep progressively splitting the clusters till singleton clusters are reached are called "divisive" clustering. On the other hand, approaches that start with singleton clusters and go on merging close clusters at every step until they reach one cluster consisting of the entire dataset are called "agglomerative" methods. While most hierarchical algorithms just join two clusters or split a cluster into two sub-clusters at every step, there exist hierarchical algorithms that can join more than two clusters in one step or split a cluster into more than two sub-clusters.
2. **Partitional techniques** create one-level (un-nested) partitioning of the data points. If K is the desired number of clusters, then partitional approaches typically find all K clusters in one step. The important issue is we need to have predefined value of K, the number of clusters we propose to identify in the dataset.

   Of course, a hierarchical approach can be used to generate a flat partition of K clusters, and likewise, the repeated application of a partitional scheme can provide a hierarchical clustering.

There are also other important distinctions between clustering algorithms as discussed below:

1. Does a clustering algorithm use all attributes simultaneously (polythetic) or use only one attribute at a time (monothetic) to compute the distance?
2. Does a clustering technique use one object at a time (incremental) or does the algorithm consider all objects at once (non-incremental)?
3. Does the clustering method allow a point to belong to multiple clusters (overlapping) or does it insist that each object can belong to one cluster only (non-overlapping)? Overlapping clusters are not to be confused with fuzzy clusters, as in fuzzy clusters objects actually belong to multiple classes with varying levels of membership.

Algorithms for clustering big data can also be distinguished by

1. Whether the dataset is treated as a Euclidean space, and if the algorithm can work for any arbitrary distance measure. In a Euclidean space where data is represented as a vector of real numbers,

the notion of a Centroid which can be used to summarize a collection of data points is very natural - the mean value of the points. In a non-Euclidean space, for example, images or documents where data is a set of words or a group of pixels, there is no notion of a Centroid, and we are forced to develop another way to summarize clusters.

2. Whether the algorithm is based on the assumption that data will fit in main memory, or whether data must reside in secondary memory, primarily. Algorithms for large amounts of data often must take shortcuts, since it is infeasible to look at all pairs of points. It is also necessary to summarize the clusters in main memory itself as is common with most big data algorithms.

9.2.4 Curse of Dimensionality

It was Richard Bellman who apparently originated the phrase "the curse of dimensionality" in a book on control theory. In current times any problem in analyzing data that occurs due to a very a large number of attributes that have to be considered is attributed to the, "curse of dimensionality".

Why should large number of dimensions pose problems? Firstly a fixed number of data points become increasingly "sparse" as the dimensionality increase. To visualize this, consider 100 points distributed with a uniform random distribution in the interval [–0, 1]. If this interval is broken into 10 cells. Then it is highly likely that all cells will contain some points. But now if we keep the number of points the same, but (project) distribute the points over the unit square that is 2-D and fix the unit of discretization to be 0.1 for each dimension, then we have 100 2-D cells, and it is quite likely that some cells will be empty. For 100 points and three dimensions, most of the 1000 cells will be empty since there are far more points than the cells. Thus our data is "lost in space" as we go to higher dimensions.

We face one more problem with high dimensions in the clustering scenario. This concerns the effect of increasing dimensionality on distance or similarity. The usefulness of a Clustering technique is critically dependent on the measure of distance or similarity used. Further for clustering to be meaningful, we require that objects within clusters should be closer to each other than to objects in other clusters. One way of analyzing the existence of clusters in a dataset is to plot the histogram of the pairwise distances of all points in a dataset (or of a sample of points in a large dataset) If clusters exist in the data, then they will show up in the graph as two peaks; one representing the distance between the points in clusters, and the other representing the average distance between the points. If only one peak is present or if the two peaks are close to each other, then clustering via distance-based approaches will likely be difficult.

Studies have also shown that for certain data distributions, the relative difference of the distances of the closest and farthest data points of an independently selected point goes to 0 as the dimensionality increases, that is,

$$\lim \frac{(\max \text{dist} - \min \text{dist})}{\min \text{dist}} = 0 \text{ as } d \to \infty$$

Thus, it is often said, "in high-dimensional spaces, distances between points become relatively uniform". In such cases, the notion of the nearest neighbor of a point is meaningless. To understand this in a more geometrical way, consider a hyper-sphere whose center is the selected point and whose radius is the distance to the nearest data point. Then, if the relative difference between the distance to nearest and farthest neighbors is small, expanding the radius of the sphere "slightly" will include many more points.

9.3 Hierarchical Clustering

Hierarchical clustering builds a cluster hierarchy or, in other words, a tree of clusters, also known as a dendrogram. Every cluster node contains child clusters; sibling clusters partition the points covered by their common parent. Such an approach allows exploring data on different levels of granularity. Hierarchical clustering methods are categorized into agglomerative (bottom–up) and divisive (top–down).

The advantages of hierarchical clustering include: embedded flexibility regarding the level of granularity; ease of handling of any forms of similarity or distance; can be extended to be applicable with any attribute types. But these algorithms also suffer from the following drawbacks: ambiguity in termination criteria; the fact that most hierarchical algorithms cannot disturb an earlier intermediate cluster even for improvement of cluster quality.

Hierarchical clustering can also be differentiated based on whether we are dealing with Euclidean or non-Euclidean space.

9.3.1 Hierarchical Clustering in Euclidean Space

These methods construct the clusters by recursively partitioning the instances in either a top-down or bottom-up fashion. These methods can be subdivided as following:

1. **Agglomerative hierarchical clustering:** Each object initially represents a cluster of its own. Then clusters are successively merged until the desired clustering is obtained.
2. **Divisive hierarchical clustering:** All objects initially belong to one cluster. Then the cluster is divided into sub-clusters, which are successively divided into their own sub-clusters. This process continues until the desired cluster structure is obtained.

The result of the hierarchical methods is a dendrogram, representing the nested grouping of objects and similarity levels at which groupings change. A clustering of the data objects is obtained by cutting the dendrogram at the desired similarity level. Figure 9.1 shows a simple example of hierarchical clustering. The merging or division of clusters is to be performed according to some similarity measure, chosen so as to optimize some error criterion (such as a sum of squares).

The hierarchical clustering methods can be further divided according to the manner in which inter-cluster distances for merging are calculated.

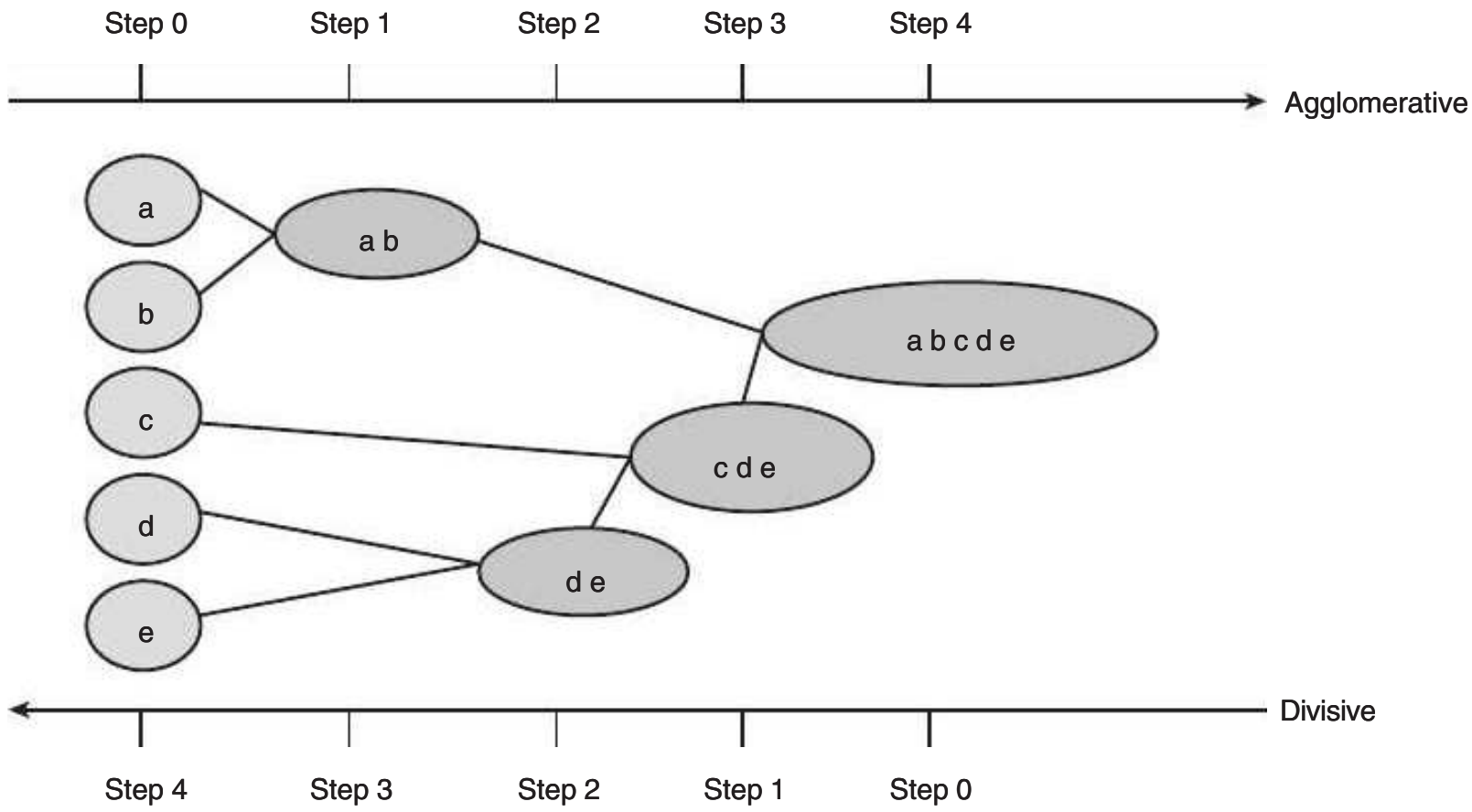


Figure 9.1 Hierarchical clustering.

1. **Single-link clustering:** Here the distance between the two clusters is taken as the shortest distance from any member of one cluster to any member of the other cluster. If the data consist of similarities, the similarity between a pair of clusters is considered to be equal to the greatest similarity from any member of one cluster to any member of the other cluster.
2. **Complete-link clustering:** Here the distance between the two clusters is the longest distance from any member of one cluster to any member of the other cluster.
3. **Average-link clustering:** Here the distance between two clusters is the average of all distances computed between every pair of two points one from each cluster.
4. **Centroid link clustering:** Here the distance between the clusters is computed as the distance between the two mean data points (average point) of the clusters. This average point of a cluster is called its **Centroid**. At each step of the clustering process we combine the two clusters that have the smallest Centroid distance. The notion of a Centroid is relevant for Euclidean space only, since all the data points have attributes with real values. Figure 9.2 shows this process.

Another decision point for hierarchical clustering would be the stopping criteria. Some choices are: stop merging (or splitting) the clusters when

1. Some pre-determined number of clusters is reached. We know beforehand that we need to find a fixed number of clusters "*K*" and we stop when we have *K* clusters.

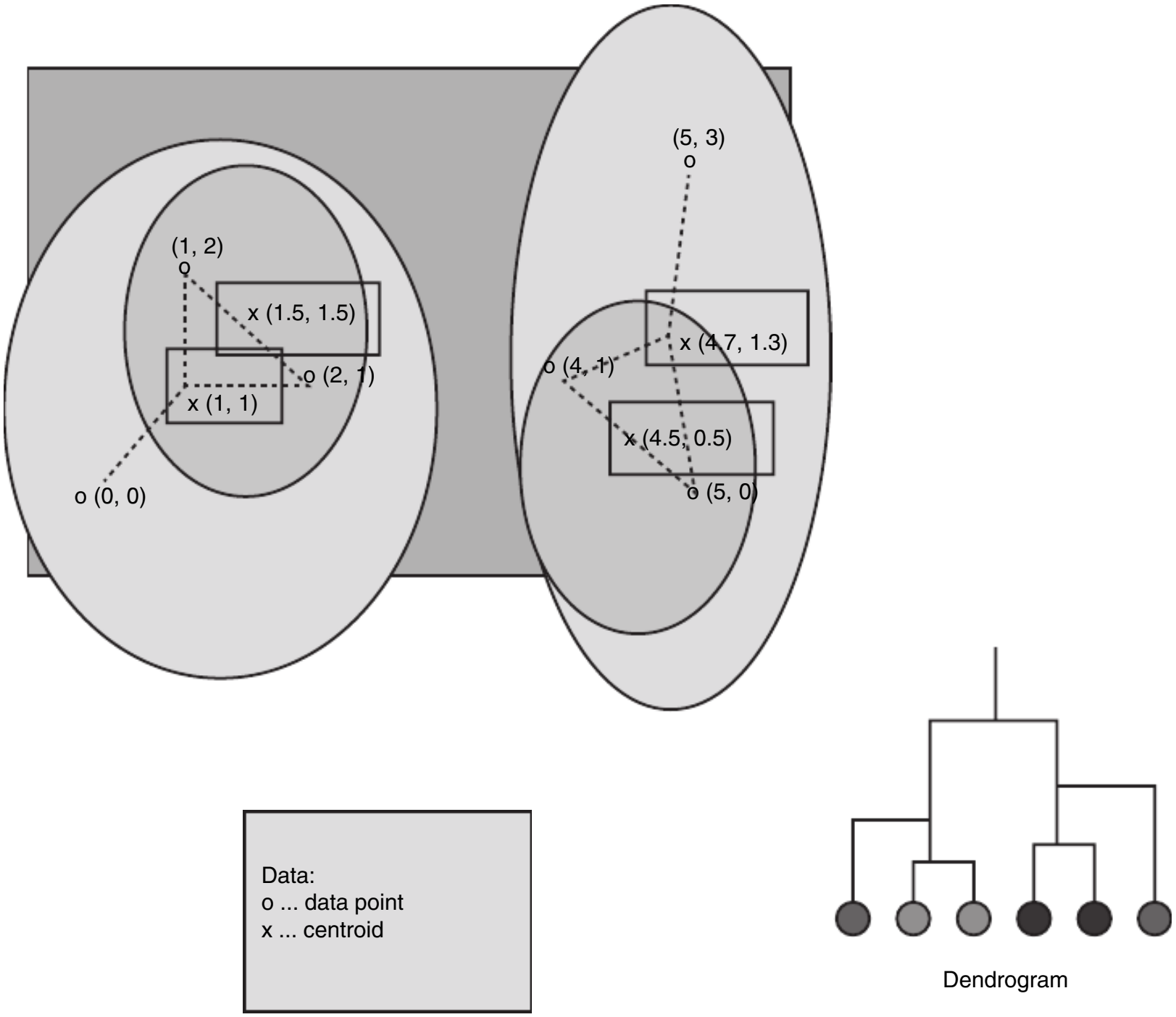


Figure 9.2 Illustrating hierarchical clustering.

2. If across a series of merges or splits, very little change occurs to the clustering, it means clustering has reached some stable structure.
3. If the maximum distance between any two points in a cluster becomes greater than a pre-specified value or threshold we can stop further steps.
4. Combination of the above conditions.

The main disadvantages of the hierarchical methods are as follows:

1. Inability to scale well – The time complexity of hierarchical algorithms is at least $O(m2)$ (where m is the total number of instances), which is non-linear with the number of objects.

Clustering a large number of objects using a hierarchical algorithm is also characterized by huge I/O costs.

2. Hierarchical methods can never undo what was done previously. Namely there is no back-tracking capability.

9.3.2 Hierarchical Clustering in Non-Euclidean Space

When the space is non-Euclidean, as is the case when data points are documents or images etc., we need to use some distance measure, such as Jaccard, cosine, or edit distance. We have seen in Chapter 5 how to compute such distances between two such data points. We need some mechanism for finding inter-cluster distance using Centroids. A problem arises when we need to represent a cluster, because we cannot replace a collection of points by their Centroid.

Example 3

Suppose we are using Jaccard distances and at some intermediate stage we want to merge two documents with Jaccard distance within a threshold value. However, we cannot find a document that represents their average, which could be used as its Centroid. Given that we cannot perform the average operation on points in a cluster when the space is non-Euclidean, our only choice is to pick one of the points of the cluster itself as a representative or a prototype of the cluster. Ideally, this point is similar to most points of the cluster, so it in some sense the "center" point.

This representative point is called the "Clustroid" of the cluster. We can select the Clustroid in various ways. Common choices include selecting as the Clustroid the point that minimizes:

1. The sum of the distances to the other points in the cluster.
2. The maximum distance to another point in the cluster.
3. The sum of the squares of the distances to the other points in the cluster.

9.4 Partitioning Methods

Partition-based clustering techniques create a flat single-level partitioning of the data points. If K is the desired number of clusters we want to discover in the data set, partition-based approaches find all K clusters in one step. Partitioning methods start with a random initial partition and keep reallocating the data points to different clusters in an iterative manner till the final partition is attained. These methods typically require that the number of clusters K will be predefined by the user. To achieve optimal clusters in partitioned-based clustering, a brute force approach will need to check all possible partitions of

the data set and compare their clustering characteristics to come up with the ideal clustering. Because this is not computationally feasible, certain greedy heuristics are used and an iterative optimization algorithm is used.

The simplest and most popular algorithm in this class is the *K*-means algorithm. But its memory requirements dictate they can only be used on small datasets. For big datasets we discuss a variant of *K*-means called the BFR algorithm.

9.4.1 K-Means Algorithm

The *K*-means algorithm discovers *K* (non-overlapping) clusters by finding *K* centroids ("central" points) and then assigning each point to the cluster associated with its nearest centroid. A cluster centroid is typically the mean or median of the points in its cluster and "nearness" is defined by a distance or similarity function. In the ideal case, the Centroids are chosen to minimize the total "error", where the error for each point is given by a function that measures the dispersion of a point from its cluster Centroid, for example, the squared distance.

The algorithm starts with an initial set of cluster Centroids chosen at random or according to some heuristic procedure. In each iteration, each data point is assigned to its nearest cluster Centroid. Nearness is measured using the Euclidean distance measure. Then the cluster Centroids are re-computed. The Centroid of each cluster is calculated as the mean value of all the data points that are assigned to that cluster.

Several termination conditions are possible. For example, the search may stop when the error that is computed at every iteration does not reduce because of reassignment of the Centroids. This indicates that the present partition is locally optimal. Other stopping criteria can be used also such as stopping the algorithms after a pre-defined number of iterations. The procedure for the *K*-means algorithm is depicted in Fig. 9.3.

Algorithm

Input: *S* (data points), *K* (number of clusters)

Output: *K* clusters

1. Choose initial *K* cluster Centroids randomly.
2. **while** termination condition is not satisfied **do**
   - **(a)** Assign data points to the closest Centroid.
   - **(b)** Recompute Centroids based on current cluster points.

   end while

One such algorithm is the *K*-medoids or partition around medoids (PAM). Each cluster is represented by the central object in the cluster, rather than by the mean that may not even belong to the cluster. The *K*-medoids method is more robust than the *K*-means algorithm in the presence of noise and outliers because a medoid is less influenced by outliers or other extreme values than a mean. However, its processing is more costly than the *K*-means method. Both methods require the user to specify *K*, the number of clusters.

9.4.2 K-Means For Large-Scale Data

We shall now examine an extension of *K*-means that is designed to deal with datasets so large that they cannot fit in main memory. This algorithm does not output the clustering per se but just determines the cluster Centroids. If we require the final clustering of points, another pass is made through the dataset, assigning each point to its nearest Centroid which is then returned as the result.

This algorithm, called the BFR algorithm, for its authors (Bradley, Fayyad and Reina), assumes the data points are from an n-dimensional Euclidean space. Therefore, we can use the Centroids to represent clusters, as they are being formed. The BFR algorithm also assumes that the quality of a cluster can be measured by the variance of the points within a cluster; the variance of a cluster is the average of the square of the distance of a point in the cluster from the Centroid of the cluster.

This algorithm for reasons of optimality does not record the Centroid and variance, but rather stores the following $(2n + 1)$ summary statistics for each cluster:

1. N, the number of points in the cluster.
2. For each dimension i, the sum of the i^{th} coordinates of the points in the cluster, denoted SUM_i.
3. For each dimension i, the sum of the squares of the i^{th} coordinates of the points in the cluster, denoted as SUMSQ_i.

The reasoning behind using these parameters comes from the fact that these parameters are easy to compute when we merge two clusters. We need to just add the corresponding values from the two clusters. Similarly we can compute the Centroid and variance also very easily from these values as:

1. The i^{th} coordinate of the Centroid is $\dfrac{\text{SUM}_i}{N}$.
2. The variance in the i^{th} dimension is $\dfrac{\text{SUMSQ}_i}{N} - \left(\dfrac{\text{SUM}_i}{N}\right)^2$.
3. The standard deviation in the i^{th} dimension is the square root of the variance in that dimension.

Initially, the BFR algorithm selects k points, either randomly or using some preprocessing methods to make better choices. In the next step the data file containing the points of the dataset are in chunks. These chunks could be from data stored in a distributed file system or there may be one monolithic

huge file which is then divided into chunks of the appropriate size. Each chunk consists of just so many points as can be processed in the main memory. Further some amount of main memory is also required to store the summaries of the k clusters and other data, so the entire memory is not available to store a chunk.

The data stored in the main-memory other than the chunk from the input consists of three types of objects:

1. **The discard set:** The points already assigned to a cluster. These points do not appear in main memory. They are represented only by the summary statistics for their cluster.
2. **The compressed set:** There are several groups of points that are sufficiently close to each other for us to believe they belong in the same cluster, but at present they are not close to any current Centroid. In this case we cannot assign a cluster to these points as we cannot ascertain to which cluster they belong. Each such group is represented by its summary statistics, just like the clusters are, and the points themselves do not appear in main memory.
3. **The retained set:** These points are not close to any other points; they are "outliers." They will eventually be assigned to the nearest cluster, but for the moment we have to retain each such point in main memory.

These sets will change as we bring in successive chunks of data into the main memory. Figure 9.4 indicates the state of the data after a few chunks of data have been processed by the BFR algorithm.

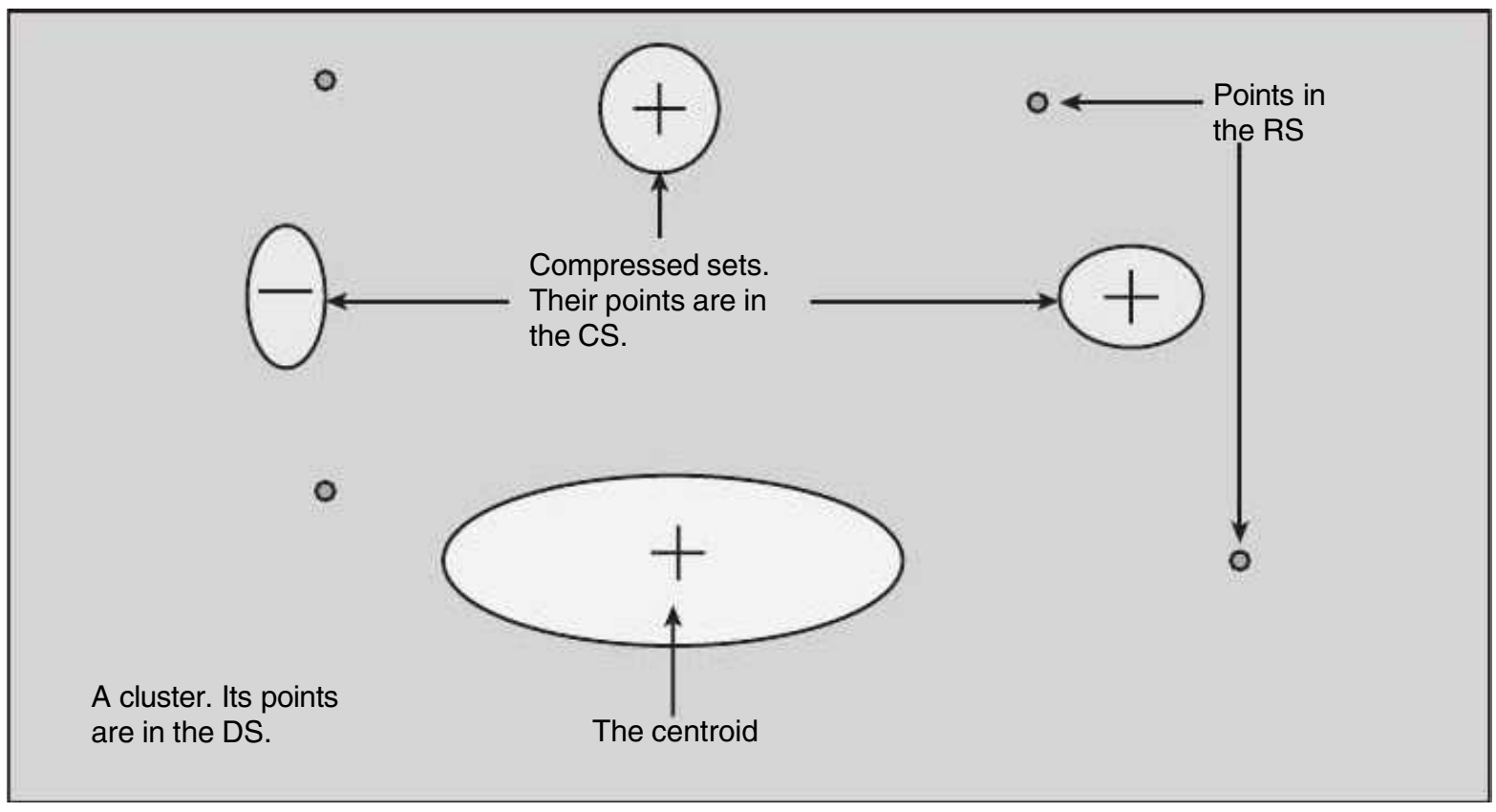


Figure 9.4 The three sets of points identified by BFR.

9.4.2.1 Processing a Memory Load of Points

We shall now describe how one chunk of data points are processed. We assume that the main memory currently contains the summary statistics for the K clusters and also for zero or more groups of points that are in the compressed set. The main memory also holds the current set of points in the retained set. We then perform the following steps:

1. For all points (xi, x2,…, xn) that are "sufficiently close" (based on distance threshold) to the Centroid of a cluster, add the point to that cluster. The point then is added to the discard set. We add 1 to the value N in the summary statistics for that cluster indicating that this cluster has grown by one point. We also add X_j to SUM_i and add X_j^2 to SUMSQ_i for that cluster.
2. If this is the last chunk of data, merge each group from the compressed set and each point of the retained set into its nearest cluster. We have seen earlier that it is very simple and easy to merge clusters and groups using their summary statistics. Just add the counts N, and add corresponding components of the SUM and SUMSQ vectors. The algorithm ends at this point.
3. Otherwise (this was not the last chunk), use any main-memory clustering algorithm to cluster the remaining points from this chunk, along with all points in the current retained set. Set a threshold on the distance values that can occur in the cluster, so we do not merge points unless they are reasonably close.
4. Those points that remain isolated as clusters of size 1 (i.e., they are not near any other point) become the new retained set. Clusters of more than one point become groups in the compressed set and are replaced by their summary statistics.
5. Further we can consider merging groups in the compressed set. Use some threshold to decide whether groups are close enough; the following section outlines a method to do this. If they can be merged, then it is easy to combine their summary statistics, as in (2) above.

9.4.2.2 How to Decide Whether a Point is near a Cluster or Not?

Intuitively, each cluster has a threshold limit in each dimension which indicates the maximum allowed difference between values in that dimension. Since we have only the summary statistics to work with, the appropriate statistic is the standard deviation in that dimension. Recall that we can compute the standard deviations from the summary statistics; the standard deviation is the square root of the variance. We want to include a point if its distance from the cluster Centroid is not too many standard deviations in any dimension.

Thus, the first thing to do with a point p that we are considering for inclusion in a cluster is to normalize p relative to the Centroid and the standard deviations of the cluster say p'. The normalized distance of p from the centroid is the absolute distance of p' from the origin. This distance is sometimes called the Mahalanobis distance.

Suppose initially we assign A1, B1, and C1 as the center of each cluster, respectively. Use the *K*-means algorithm to show only (a) the three cluster centers after the first round of execution and (b) the final three clusters.

2. Given a 1-D dataset {1,5,8,10,2}, use the agglomerative clustering algorithms with Euclidean distance to establish a hierarchical grouping relationship. Draw the dendrogram.

3. Both *K*-means and *K*-medoids algorithms can perform effective clustering. Illustrate the strength and weakness of *K*-means in comparison with the *K*-medoids algorithm. The goal of *K*-medoid algorithm is the same as *K*-means: minimize the total sum of the distances from each data point to its cluster center. Construct a simple 1-D example where *K*-median gives an output that is different from the result retuned by the *K*-means algorithm. (Starting with the same initial clustering in both cases.)

4. Suppose a cluster of 3-D points has standard deviations of 2, 3, and 5, in the three dimensions, in that order. Compute the Mahalanobis distance between the origin (0, 0, 0) and the point (1, –3, 4).

5. For the 2-D dataset (2, 2), (3, 4), (4, 8), (4, 10), (5, 2), (6, 8), (7, 10), (9, 3), (10, 5), (11, 4), (12, 3), (12, 6), we can make three clusters:

- Compute the representation of the cluster as in the BFR algorithm. That is, compute N, SUM, and SUMSQ.
- Compute the variance and standard deviation of each cluster in each of the two dimensions.

6. Execute the BDMO algorithm with $p = 3$ on the following 1-D, Euclidean data: 1, 45, 80, 24, 56, 71, 17, 40, 66, 32, 48, 96, 9, 41, 75, 11, 58, 93, 28, 39, 77. The clustering algorithms is k-means with $k = 3$. Only the centroid of a cluster, along with its count, is needed to represent a cluster.

7. Using your clusters from Exercise 6, produce the best centroids in response to a query asking for a clustering of the last 10 points.

8. In certain clustering algorithms, such as CURE, we need to pick a representative set of points in a supposed cluster, and these points should be as far away from each other as possible. That is, begin with the two furthest points, and at each step add the point whose minimum distance to any of the previously selected points is maximum.

Suppose you are given the following points in 2-D Euclidean space:

$x = (0,0)$; $y = (10,10)$; $a = (1,6)$; $b = (3,7)$; $c = (4,3)$; $d = (7,7)$; $e = (8,2)$; $f = (9,5)$.

Obviously, x and y are furthest apart, so start with these. You must add five more points, which we shall refer to as the first, second,..., fifth points in what follows. The distance measure is the normal Euclidean distance. Identify the order in which the five points will be added.

Programming Assignments

1. Implement the BFR (modified *K*-mean) algorithm on large set of 3-d points using MapReduce.
2. Implement the CURE algorithm using MapReduce on the same dataset used in Problem 1.

References

1. B. Babcock, M. Datar, R. Motwani, L. O'Callaghan (2003). Maintaining Variance and k-medians over Data Stream Windows. *Proc. ACM Symp. on Principles of Database Systems*, pp. 234–243.
2. P.S. Bradley, U.M. Fayyad, C. Reina (1998). Scaling Clustering Algorithms to Large Databases. *Proc. Knowledge Discovery and Data Mining*, pp. 9– 15.
3. V. Ganti, R. Ramakrishnan, J. Gehrke et al. (1999). Clustering Large Datasets in Arbitrary Metric Spaces. *Proc. Intl. Conf. on Data Engineering*, pp. 502–511.
4. H. Garcia-Molina, J.D. Ullman, J. Widom (2009). *Database Systems: The Complete Book,* Second Edition. Prentice-Hall, Upper Saddle River, NJ.
5. S. Guha, R. Rastogi, K. Shim (1998). CURE: An Efficient Clustering Algorithm for Large Databases. *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pp. 73–84.
6. L. Rokach, O. Maimon (2005). Clustering Methods. In: O. Maimon, L. Rokach (Eds.), *Data Mining and Knowledge Discovery Handbook*. Springer, New York, pp. 321–352.
7. C. C. Aggarwal, C. Reddy (2013). *Data Clustering: Algorithms and Applications*. CRC Press.
8. M. Steinbach, L. Ertoz, V. Kumar (2003). Challenges of Clustering High Dimensional Data. In: L. T. Wille (Ed.), *New Vistas in Statistical Physics – Applications in Econophysics, Bioinformatics, and Pattern Recognition*. Springer-Verlag.

10 Recommendation Systems

LEARNING OBJECTIVES

After reading this chapter, you will be able to:

- Learn the use of Recommender system.
- Understand various models of Recommender system.
- Learn collaborative filtering approach for Recommender system.
- Learn content based approach for Recommender system.
- Understand the methods to improve prediction function.

10.1 Introduction

10.1.1 What is the Use of Recommender System?

For business, the recommender system can increase sales. The customer service can be personalized and thereby gain customer trust and loyalty. It increases the knowledge about the customers. The recommender system can also give opportunities to persuade the customers and decide on the discount offers.

For customers, the recommender system can help to narrow down their choices, find things of their interests, make navigation through the list easy and to discover new things.

10.1.2 Where Do We See Recommendations?

Recommendations are commonly seen in e-commerce systems, LinkedIn, friend recommendation on Facebook, song recommendation at FM, news recommendations at Forbes.com, etc.

10.1.3 What Does Recommender System Do?

It takes the user model consisting of ratings, preferences, demographics, etc. and items with its descriptions as input, finds relevant score which is used for ranking, and finally recommends items that are relevant to the user. By estimating the relevance, information overload can be reduced. But the relevance can be context-dependent.

We further look at algorithms for two very interesting problems in social networks. The "SimRank" algorithm provides a way to discover similarities among the nodes of a graph. We also explore triangle counting as a way to measure the connectedness of a community.

11.2 Applications of Social Network Mining

Applications of social network mining encompass numerous fields. A few popular ones are enumerated here:

1. Viral marketing is an application of social network mining that explores how individuals can influence the buying behavior of others. Viral marketing aims to optimize the positive word-of-mouth effect among customers. Social network mining can identify strong communities and influential nodes. It can choose to spend more money marketing to an individual if that person has many social connections.
2. Similarly in the e-commerce domain, the grouping together of customers with similar buying profiles enables more personalized recommendation engines. Community discovery in mobile ad-hoc networks can enable efficient message routing and posting.
3. Social network analysis is used extensively in a wide range of applications, which include data aggregation and mining, network propagation modeling, network modeling and sampling, user attribute and behavior analysis, community-maintained resource support, location-based interaction analysis, social sharing and filtering, recommender systems, etc.
4. Many businesses use social network analysis to support activities such as customer interaction and analysis, information system development analysis, targeted marketing, etc.

11.3 Social Networks as a Graph

When the objective is to mine a social network for patterns, a natural way to represent a social network is by a graph. The graph is typically very large, with nodes corresponding to the objects and the edges corresponding to the links representing relationships or interactions between objects. One node indicates one person or a group of persons.

As an example, consider a set of individuals on a social networking site like LinkedIn. LinkedIn allows users to create profiles and "connections" with each other in an online social network which may represent real-world professional relationships. Thus, finding a person on a LinkedIn network is like finding a path in a graph representing the social network. Similarly, a company may want to find the most influential individual in a social network to perform targeted advertising. The hope is that more influential a person, more people can be influenced to buy their goods. This means identifying nodes with very high out-degree in the graph representing the social network.

Finding communities or clustering social networks, all lead to the standard graph algorithms when the social network is modeled as a graph. This section will give an overview of how a social network can be modeled as a graph. A small discussion of the different types of graphs is also provided.

11.3.1 Social Networks

The web-based dictionary Webopedia defines a social network as "A social structure made of nodes that are generally individuals or organizations". A social network represents relationships and flows between people, groups, organizations, computers or other information/knowledge processing entities. The term "Social Network" was coined in 1954 by J. A. Barnes. Examples of social networks include Facebook, LinkedIn, Twitter, Reddit, etc.

The following are the typical characteristics of any social network:

1. In a social network scenario, the nodes are typically people. But there could be other entities like companies, documents, computers, etc.
2. A social network can be considered as a **heterogeneous** and **multi-relational** dataset represented by a graph. Both nodes and edges can have attributes. Objects may have class labels.
3. There is at least one relationship between entities of the network. For example, social networks like Facebook connect entities through a relationship called friends. In LinkedIn, one relationship is "endorse" where people can endorse other people for their skills.
4. In many social networks, this relationship need not be yes or no (binary) but can have a degree. That means in LinkedIn, we can have a degree of endorsement of a skill like say novice, expert, etc. Degree can also be a real number.
5. We assume that social networks exhibit the property of non-randomness, often called locality. Locality is the property of social networks that says nodes and edges of the graph tend to cluster in communities. This condition is the hardest to formalize, but the intuition is that the relationships tend to cluster. That is, if entity A is related to both B and C, then there is a higher probability than average that B and C are related. The idea is most relationships in the real worlds tend to cluster around a small set of individuals.

11.3.2 Social Network Graphs

A natural representation of social network data is to view its structure as a graph. The basic process is as follows. Entities are converted to nodes in the graph. The nodes in this graph can be of different types, for example, people, organizations and events. This model, thus, slightly deviates from the standard model of a graph in which the node set is one type. Edges in the graph correspond to the relations between entities. If there is a degree associated with the relationship, this degree is represented by labeling the edges. Edges can be one-directional, bi-directional and are not required to be binary.

Example 1

Figure 11.1 shows a small graph of the "followers" network of Twitter. The relationship between the edges is the "follows" relationship. Jack follows Kris and Pete shown by the direction of the edges. Jack and Mary follow each other shown by the bi-directional edges. Bob and Tim follow each other as do Bob and Kris, Eve and Tim. Pete follows Eve and Bob, Mary follows Pete and Bob follows Alex. Notice that the edges are not labeled, thus follows is a binary connection. Either a person follows somebody or does not.

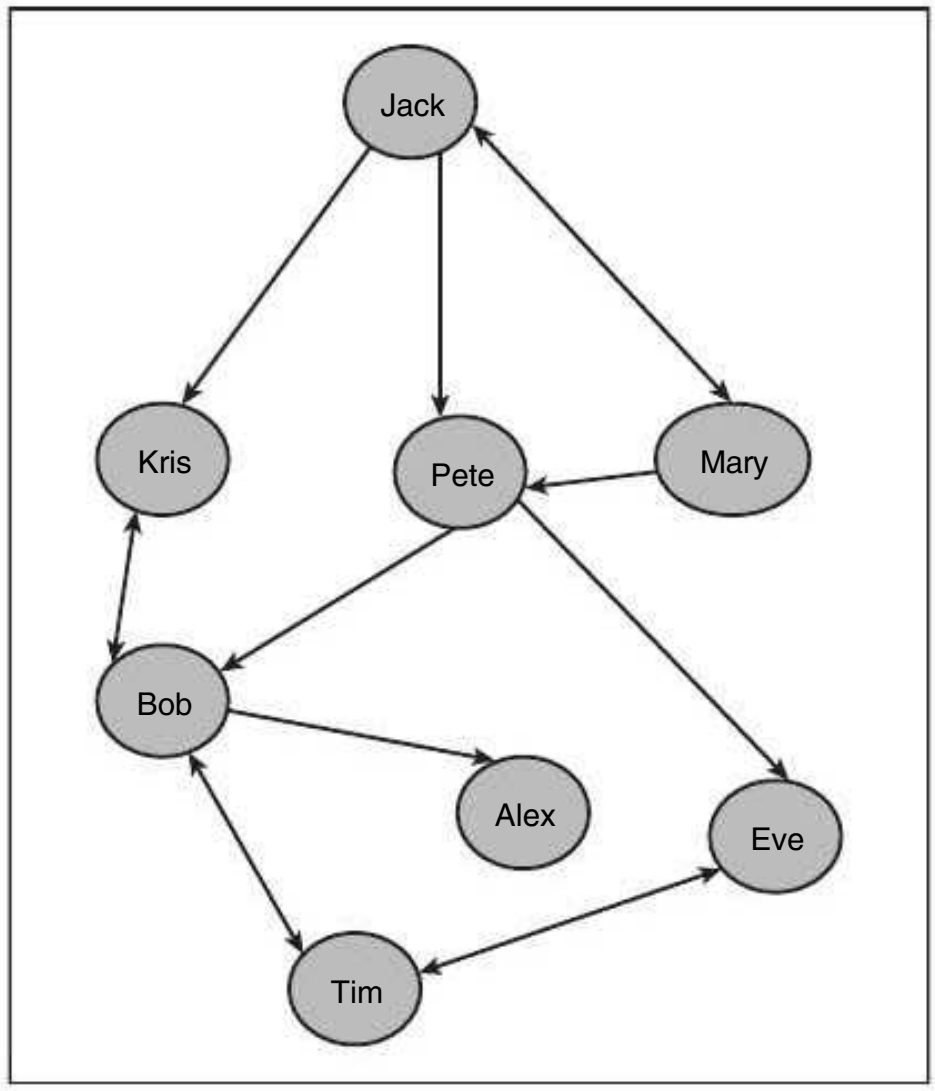


Figure 11.1 A social graph depicting follows relationship in Twitter.

Example 2

Consider LiveJournal which is a free on-line blogging community where users declare friendship to each other. LiveJournal also allows users to form a group which other members can then join. The graph depicted in Fig. 11.2 shows a portion of such a graph. Notice that the edges are undirected, indicating that the "friendship" relation is commutative.

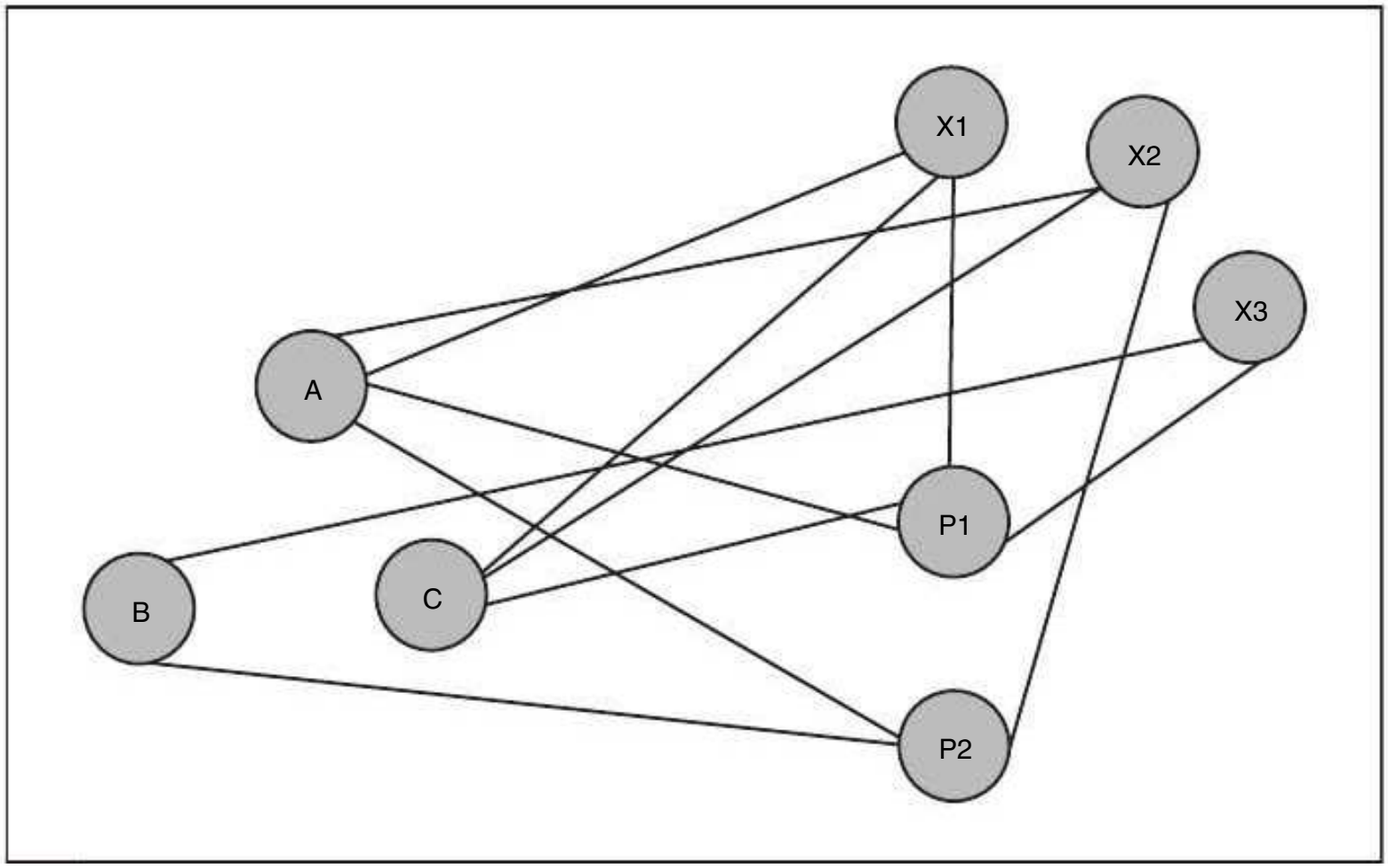


Figure 11.10 Sample social graph for SimRank.

11.8 Counting Triangles in a Social Graph

One important analysis that is particularly useful in the social network context is identifying small communities and counting their occurrence. This basically amounts to finding small connected sub-graphs in the social graph. The most important such sub-graph is the triangle (3-clique). A triangle is the most commonly occurring pattern found in social network graphs. This could be due to two properties that are exhibited by most social graphs: "*homophily*" which is the tendency of individuals to associate and form groups with similar others, and "*transitivity*" which talks of the transitive nature of relationships in a social network. It says if A is associated (friend) to B and B is associated with C, then A and C will also become associates (friends).

The most basic structure in a graph with the above two properties is a triangle or a 3-clique. Thus, counting triangles is an important aspect in any social network analysis application. In this section we present a few motivating examples for counting triangles. We follow it up with an overview of existing triangle counting techniques. As we shall see triangle counting is a computationally expensive task. We give some thoughts on how these algorithms could be set up in a MapReduce framework.

11.8.1 Why Should We Count Triangles?

An important measure one is interested in a social graph is given a node "what is its clustering coefficient?". The clustering coefficient measures the degree to which a node's neighbors are themselves neighbors. A node's *clustering coefficient* is the ratio of the number of closed triplets in the node's neighborhood over the total number of triplets in the neighborhood.

There are two main motivating reasons for why clustering coefficients are important for graph analysis: A high clustering coefficient indicates very closely knit communities, that is a group of entities where most are of the form A associated with B and C implies B and C are also associated. Such communities are interesting for many applications like target marketing, social recommendations, etc. In a complementary way, a low clustering coefficient can signal a structure called as a "structural hole", a vertex that is well connected to many communities that are not otherwise connected to each other. Such vertices can act as bridges between communities and are useful for applications that need to identify influential nodes and also where information propagation is needed.

11.8.2 Triangle Counting Algorithms

The brute force method and obvious way to count triangles in a graph is simply checking every group of three vertices and check if they form a triangle or not. For implementing this algorithm, we need to be able to answer queries of the form "given a pair u; v of vertices, is (u, v) an edge?" This algorithm will take $O(n^3)$ time where n is the number of vertices in the graph. The major disadvantage of this method in addition to its high computation cost is that it takes the same amount of time to identify that a graph does not possess a triangle as it takes to find triangles.

A smarter approach is to first list all two-edge paths that are formed in the graph. Such paths of size 2 are called "wedges". Thus instead of attempting to find all vertex triples, we only attempt to check vertex triples in which we already know that two edges are present. For every edge (x, y) in the graph, check if (x, y, z) forms a triangle and if so add one to the count of triangles. This process has to be repeated for every vertex z distinct from vertices x and y in the graph.

The analysis of the running time is straightforward, as the algorithm runs in time which is $0\left(\sum_{v \in V} \text{degree}_v^2\right)$. It can be proved that for graphs of constant degree this is a linear time algorithm. But the existence of even one high degree vertex will make this algorithm quadratic. Since it is highly likely to come across such high degree nodes in massive social graphs, this algorithm too is not practical. Further, this algorithm counts each triangle $\{x, y, z\}$ six times (once each as $\{x, y, z\}$, $\{x, z, y\}$, $\{y, x, z\}$, $\{y, z, x\}$, $\{z, x, y\}$ and $\{z, y, x\}$).

One optimization is to count each triangle only once. We can use another trick for further making the algorithm efficient. The key idea to use is that "only the lowest-degree vertex of a triangle is responsible for counting it". Further, we also use an ordering of vertices which starts from the most likely vertices to form triangles to the ones with least probability.

The algorithm can be described as follows: Identify "Massive" vertices in a graph. Let the social graph have n vertices and m edges. We call a vertex massive if its degree is at least $\sqrt{m}$. If a triangle

has all its vertices as massive, then it is a massive triangle. The algorithm identifies massive triangles and non-massive triangles separately. We can easily see that the maximum number of massive vertices a graph can have is $2\sqrt{m}$. Now represent a graph as a list of its m edges.

1. Compute the degree of each vertex. Examine each edge and add 1 to the count of each of its two end vertices. The total time required is O(m).
2. Create an index on edges using a hash table, with its vertex pair as a key. So we can check whether an edge exists given a pair of vertices in constant O(1) time.
3. Create one more hash table for the edges key being a single vertex. Given a vertex, we can easily identify all vertices adjacent to it.
4. Number the vertices and order them in the ascending order of their degree. Lower degree vertices first followed by higher degree vertices. If two vertices have same degree then order them based on their number.

To find massive triangles we notice the following. There can be only O($\sqrt{m}$) massive vertices and so if we check all possible three subsets of these set of vertices, we have O($m^{3/2}$) possible massive triangles. To check the existence of all three edges we need only constant time because of the hash tables. Thus this implementation is only O($m^{3/2}$).

To find the non-massive triangles we adopt the following procedure:

1. Consider each edge (v_1, v_2). Ignore this edge when both v_1 and v_2 are massive.
2. Suppose v_1 is not massive and v_1 appears before v_2 in the vertex ordering. We enumerate all vertices adjacent to v_1 and call them u_1, u_2 , …, u_k, k will be less than $\sqrt{m}$. It is easy to find all the u's because of the second hash table which has single vertices as keys (O(1) time).
3. Now for each u_i we check if edge (u_i, v_2) exists and if v_1 appears before u_i in the list. We count this triangle. (Thus we avoid counting the same triangle several times.) This operation is benefitted because of the first hash table with key being edges.

Thus, the time to process all the nodes adjacent to v_1 is O($\sqrt{m}$). Since there are m edges, the total time spent counting other triangles is O($m^{3/2}$).

11.8.3 Counting Triangles Using MapReduce

For massive social graphs the methods described above will still be too expensive. This motivates us to find a parallel implementation of the triangle counting algorithms. We present a brief outline of how the MapReduce paradigm can be used for the triangle counting exercise. It is left to the reader to fill up the finer details of the MR implementation.

Let G be a social graph. We number the set of vertices V of a G as 1, 2, ..., n. Then we can construct a relation E on ($V \times V$) to represent edges. To avoid representing each edge twice, we declare that $E(A, B)$ is a tuple of this relation if an edge exists between A and B and as integers, we have $A < B$.

Index

About the Book

The goal of this book is to cover foundational techniques and tools required for *Big Data Analytics*. It focuses on concepts, principles, and techniques applicable to any technology environment and industry and establishes a baseline that can be enhanced further by additional real-world experience. This book aims to be a ready reckoner to either a novice or a professional working in the field.

Topics covered include Hadoop, MapReduce, Association Rules, Large-Scale Supervised Machine Learning, Data Streams, Clustering, NoSQL systems (Pig, Hive), and Applications including Recommendation Systems, Web and Security.

Precise Series

- Precise contents as per syllabi.
- Excellent presentation in a clear, logical and concise manner.
- Learning objectives at the beginning of each chapter.
- Focus on explanation of concepts.
- Sufficient examples for easy comprehension.
- Important points and formulas at the end of each chapter to quickly review the concepts.
- Optimum balance of qualitative and quantitative problem sets.

Highlights of the Book

- Fills the gap in the literature available on *Big Data Analytics*.
- Combines *Big Data Management and Analytics* in the same book.
- Provides overview of tools available for BDA: *NOSQL*, *MapReduce*, *Hadoop*.
- Covers many real-life examples.
- Provides links to several real-life datasets in the Appendix.
- Includes comprehensive *Lab Manual*.

READER LEVEL
Undergraduate/Graduate

SHELVING CATEGORY
Engineering

WILEY

Wiley India Pvt. Ltd.
4435-36/7, Ansari Road, Daryaganj
New Delhi-110 002
Customer Care +91 11 43630000
Fax +91 11 23275895
csupport@wiley.com
www.wileyindia.com

ISBN: 978-81-265-5865-0

9 788126 558650