BIG DATA AND CLOUD PROGRAMMING AND SOFTWARE ENVIRONMENTS

Prepared by: Dr. Faramarz Safi
Islamic Azad University, Najafabad Branch, Esfahan, Iran.
Gartner’s 2014 Hype Cycle on Cloud, Big Data, IoT, …

- **Plateau of Productivity**
- **Innovation Trigger**
- **Peak of Inflated Expectations**
- **Trough of Disillusionment**
- **Slope of Enlightenment**

- **Expectations**
  - Speech-to-Speech Translation
  - Autonomous Vehicles
  - Smart Advisors
  - Data Science
  - Prescriptive Analytics
  - Neurobusiness
  - Biochips
  - Affective Computing
  - Smart Robots
  - 3D Bioprinting Systems
  - Volumetric and Holographic Displays
  - Software-Defined Anything
  - Quantum Computing
  - Human Augmentation
  - Brain-Computer Interface
  - Connected Home
  - Virtual Personal Assistants
  - Digital Security
  - Bioacoustic Sensing
- **Internet of Things**
  - Natural-Language Question Answering
  - Wearable User Interfaces
  - Consumer 3D Printing
  - Cryptocurrencies
  - Complex-Event Processing
  - Big Data
  - In-Memory Database Management Systems
  - Content Analytics
  - Hybrid Cloud Computing
  - Gamification
  - Augmented Reality
  - Machine-to-Machine Communication Services
  - Mobile Health Monitoring
  - Enterprise 3D Printing
- **As of July 2014**

- **Plateau will be reached in:**
  - less than 2 years
  - 2 to 5 years
  - 5 to 10 years
  - more than 10 years
  - obsolete before plateau

- **time**
Cycle of Innovation in CLOUD, BIG DATA, Internet of Things (IoT), and SDN
Changes Driving Innovation in IT

**Internet of Everything (IoT)**
- Everything is becoming connected
  - More devices
  - More Apps
  - Shift to mobile access

**Software Defined Network (SDN)**
- Software interfaces for networking
  - Automation and self-service
  - Traffic Engineering
  - Network Function Virtualization

**Big Data / Analytics**
- Analytics become business critical
  - Customer behavior
  - Systems optimization
  - Diversity of sources and data types

**Cloud Platform Software**
- Public and Private Clouds
  - Open Cloud Platforms: OpenStack
  - App development and deployment
  - Scalable services
Images Used

- http://upload.wikimedia.org/wikipedia/commons/1/15/London_traffic-lights.jpg
- http://www.cccblog.org/wp-content/uploads/2012/08/bigdata_olympics_1h.jpg

Credits & References

All images used are found on the internet and believed to be in the public domain. In the event that there is still a problem or error with copyrighted material, the break of the copyright is unintentional and non-commercial and the material will be removed upon request.
### Some Data sizes

<table>
<thead>
<tr>
<th>Category</th>
<th>Data Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>~$40 \times 10^9$ Web pages</td>
<td>at ~300 kilobytes each = 10 Petabytes</td>
</tr>
<tr>
<td>Youtube</td>
<td>48 hours video uploaded per minute;</td>
</tr>
<tr>
<td></td>
<td>• in 2 months in 2010, uploaded more than total NBC ABC CBS</td>
</tr>
<tr>
<td></td>
<td>• ~2.5 petabytes per year uploaded?</td>
</tr>
<tr>
<td>LHC</td>
<td>15 petabytes per year</td>
</tr>
<tr>
<td>Radiology</td>
<td>69 petabytes per year</td>
</tr>
<tr>
<td>Square Kilometer Array Telescope</td>
<td>will be 100 terabits/second</td>
</tr>
<tr>
<td>Earth Observation</td>
<td>becoming ~4 petabytes per year</td>
</tr>
<tr>
<td>Earthquake Science</td>
<td>– few terabytes total today</td>
</tr>
<tr>
<td>PolarGrid</td>
<td>– 100’s terabytes/year</td>
</tr>
<tr>
<td>Exascale simulation data dumps</td>
<td>data dumps – terabytes/second</td>
</tr>
</tbody>
</table>
Application and Sources of Big Data analytics

- Smarter Healthcare
- Homeland Security
- Traffic Control
- Manufacturing
- Multi-channel sales
- Telecom
- Trading Analytics
- Search Quality
The FOUR V’s of Big Data

Volume

Scale of Data

- 40 zettabytes (43 trillion gigabytes) of data will be created by 2020, an increase of 300 times from 2005
- 6 billion people have cell phones

Velocity

Analysis of Streaming Data

- Modern cars have close to 100 sensors that monitor items such as fuel level and tire pressure

Variety

Different Forms of Data

- 38 billion pieces of content are shared on Facebook every month
- 400 million tweets are sent per day by about 200 million monthly active users

Veracity

Uncertainty of Data

- 27% of respondents in one survey were unsure of how much of their data was accurate

As of 2011, the global size of data in healthcare was estimated to be 150 exabytes (141 trillion gigabytes)

By 2014, it’s anticipated there will be 420 million wearable, wireless health monitors

By 2015:
- 4.4 million IT jobs will be created globally to support big data, with 1.3 million in the United States

1 in 3 business leaders don’t trust the information they use to make decisions

The New York Stock Exchange captures 1 Td of trade information during each trading session

By 2016, it is projected there will be 18.9 billion network connections
40 Zettabytes [43 Trillion Gigabytes] of data will be created by 2020, an increase of 300 times from 2005.

It's estimated that 2.5 Quintillion Bytes [2.3 Trillion Gigabytes] of data are created each day.

Volume Scale of Data

- Most companies in the U.S. have at least 100 Terabytes [100,000 Gigabytes] of data stored.

World Population: 7 Billion
The New York Stock Exchange captures 1 TB of trade information during each trading session.

Modern cars have close to 100 sensors that monitor items such as fuel level and tire pressure.

By 2016, it is projected there will be 18.9 billion network connections – almost 2.5 connections per person on earth.
As of 2011, the global size of data in healthcare was estimated to be **150 EXABYTES** [161 BILLION GIGABYTES].

By 2014, it’s anticipated there will be **420 MILLION WEARABLE, WIRELESS HEALTH MONITORS**.

30 BILLION PIECES OF CONTENT are shared on Facebook every month.

4 BILLION+ HOURS OF VIDEO are watched on YouTube each month.

400 MILLION TWEETS are sent per day by about 200 million monthly active users.
1 IN 3 BUSINESS LEADERS don’t trust the information they use to make decisions.

27% OF RESPONDENTS in one survey were unsure of how much of their data was inaccurate.

Veracity
Uncertainty of Data

Poor data quality costs the US economy around $3.1 TRILLION A YEAR.
What’s BigData

▪ **Data has always been Big.** The one aspect that differs now, if compared with the past, would be the sheer *scale and accessibility* of Data, which is the direct result of the super efficient speeds in which data can now be computed. Big Data is therefore an all-encompassing term for any collection of large *data sets* that were once difficult to process.

▪ Big data requires exceptional technologies to efficiently process large quantities of data within tolerable elapsed times.
Ten ways big data is different from small data

Goals
- **Small data** is usually gathered for a specific goal.
- **Big data** on the other hand may have a goal in mind when it’s first started, but things can evolve or take unexpected directions.

Location
- **Small data** is usually in one place, and often in a single computer file.
- **Big data** on the other hand can be in multiple files in multiple servers on computers in different geographic locations.

Data preparation
- **Small data** is usually prepared by the end user for their own purposes,
- **big data** is often prepared by one group of people, analyzed by a second group of people, and then used by a third group of people, and they may have different purposes, and they may have different disciplines.
Ten ways big data is different from small data

Data structure and content

• **Small data** is usually highly structured like an Excel spreadsheet, and it’s got rows and columns of data.

• **Big data** on the other hand can be unstructured, it can have many formats in files involved across disciplines, and may link to other resources.

Measurements

• **Small data** is typically measured with a single protocol using set units and it’s usually done at the same time.

• With **big data** on the other hand, because you can have people in very different places, in very different times, different organizations, and countries, you may be measuring things using different protocols, and you may have to do a fair amount of conversion to get things consistent.
Ten ways big data is different from small data

Longevity

- **Small data** is usually kept for a specific amount of time after the project is over because there's a clear ending point. In the academic world it's maybe five or seven years and then you can throw it away,
- **Big data** each data project, because it often comes at a great cost, gets continued into others, and so you have data in perpetuity, and things are going to stay there for a very long time. They may be added on to in terms of new data at the front, or contextual data of things that occurred beforehand, or additional variables, or linking up with different files. So it has a much longer and really uncertain lifespan compared to a small data set.

Analysis

- With **small data** it's usually possible to analyze all of the data at once in a single procedure from a single computer file.
- With **big data** however, because things are so enormous and they're spread across lots of different files and servers, you may have to go through extraction, reviewing, reduction, normalization, transformation, and other steps and deal with one part of the data at a time to make it more manageable, and then eventually aggregate your results.
Ten ways big data is different from small data

Stakes

- On small data, if things go wrong the costs are limited, it's not an enormous problem,
- but with big data, projects can cost hundreds of millions of dollars, and losing the data or corrupting the data can doom the project, possibly even the researcher's career or the organization's existence.

reproducibility

- Small data sets can usually be reproduced in their entirety if something goes wrong in the process.
- Big data sets on the other hand, because they come in so many forms and from different directions, it may not be possible to start over again if something's gone wrong. Usually the best you can hope to do is to at least identify which parts of the data project are problematic and keep those in mind as you work around them.
Ten ways big data is different from small data

Introspection
• what this means is that the **data describes itself in an important way**. With **small data**, the ideal for instance is what's called a triple that's used in several programming languages where you say, first off, the object that is being measured.
Data mining -> Big Data mining?
THE EVOLUTION OF BUSINESS INTELLIGENCE

1990’s

BI Reporting /OLAP & Data warehouse
Business Objects, SAS, Informatica, Cognos other SQL Reporting Tools

2000’s

Interactive Business Intelligence & In-memory RDBMS
QlikView, Tableau, HANA

2010’s

Big Data: Batch Processing & Distributed Data Store
Hadoop/Spark; HBase/Cassandra

Big Data: Real Time & Single View
Graph Databases, <key-value>, Column/Document oriented

Speed
Scale
Speed
Scale
Mind-Map and Classification of Big Data

- **Data Sources**
  - Web & Social
  - Machine
  - Sensing
  - Transactions
  - IoT

- **Content Format**
  - Structured
  - Semi-structured
  - Unstructured

- **Data Stores**
  - Document-oriented
  - Column-oriented
  - Graph-based
  - Key-value

- **Data Staging**
  - Cleaning
  - Normalization
  - Transform

- **Data processing**
  - Batch
  - Real time
(Big) Data will be in the Cloud
The Growth of Cloud-Based Big Data

- Data Analytics as a Service
- BI as a Service
- Comm as a Service
- Logs as a Service
- Data as a Service
- Dev aaS
- Integration aaS

Growth Rate (CAGR)

Maturity in Market (Years)

Size = Revenue in 2017 ($B)

Source: http://jameskaskade.com/
Cloud Technology
Percentage of Cloud-Based Data Traffic

35% 2013

70% 2020

The Digital Future

Mobile Economy 2014
Cloud computing is going to absorb your big data workloads, too

- **Cloudera** is offering a tool called **Director** that makes it easier to manage a fully functional Cloudera Hadoop cluster on the Amazon Web Services cloud.

- The **Hortonworks Data Platform** is now fully certified with **Microsoft Azure**, meaning users can run the same software and get the same experience as in their own data center.

- Further — presumably as a result of its longstanding partnership with Hortonworks — **Microsoft**'s HDInsight Hadoop service now supports the Storm stream-processing framework.

- **Rackspace** launched a big data version of its bare-metal cloud service, promising deployment of a Hadoop-Spark cluster in just a few clicks.

- **Tableau** announced a beta version of a connector for **Amazon Elastic MapReduce** (among others, including Spark SQL), which will let users pull data from the cloud service for analysis in Tableau.
Companies like Google, Microsoft and IBM continue to release new machine learning features on top of their cloud platforms and inside their cloud applications, making them look like more appealing places to store data, as well.

Teradata is running a Hadoop cloud offering as well as a cloud-based offering of its flagship data warehouse system.

Oracle — Oracle! — announced a platform-as-a-service offering. It even hired Peter Magnusson, one of the key engineers behind Google App Engine, to help lead its development. Laugh if you will, but Oracle seriously suggesting its customers run their databases in the cloud (including “extreme performance” versions) says a lot about how times have changed.
Drivers for big data on cloud adoption

- Cost reduction
  - Managing cloud-based big data is cost-effective, scalable, and fast to build.
- Rapid provisioning/time to market
  - Faster provisioning is important for big data applications because the value of data reduces quickly as time goes by.
- Flexibility/scalability
  - Big data analysis, especially in the life sciences industry, requires huge compute power for a brief amount of time. For this type of analysis, servers need to be provisioned in minutes.
Cloud enables BigData

- Some of the first adopters of big data in cloud computing are users that deployed Hadoop clusters in highly scalable and elastic clouds: IBM, Azure, AWS
- Cloud computing democratizes big data – any enterprise can now work with unstructured data at a huge scale.
- Analytics-as-a-service (AaaS) models for cloud-based big data analytics
Relationship between Cloud and BIG DATA

- Data sources
  - Storage
  - Web

- Analytics/Reports
  - Query Engine e.g., Hive, Mahout

- Programming model for processing large data sets with a parallel, distributed algorithm on a cluster like MapReduce.

- Distributed fault tolerant database for large unstructured data sets like NOSQL.

- Hadoop Distributed File System (HDFS)

- Distributed configuration and synchronization service

- Decision making

- Data visualization

- APIs
Types of Cloud-based Tools Used in Big-Data

Where processing is **hosted**?
Distributed Servers / Cloud (e.g. Amazon EC2)

Where data is **stored**?
Distributed Storage (e.g. Amazon S3)

What is the **programming model**?
Distributed Processing (e.g. MapReduce)

How data is **stored & indexed**?
High-performance schema-free databases (e.g. MongoDB)

What operations are performed on data?
Analytic / Semantic Processing
BIG DATA Processing Engines & Map-Reduce

- Data Sources
  - Web & Social
  - Machine
  - Sensing
  - Transactions
  - IoT

- Content Format
  - Structured
  - Semi-structured
  - Unstructured

- Data Stores
  - Document-oriented
  - Column-oriented
  - Graph based
  - Key-value

- Data Staging
  - Cleaning
  - Normalization
  - Transform

- Data processing
  - Batch
  - Real time
BIG DATA Processing Engines & Map-Reduce

Diagram showing various data processing engines including Big Data, GreenPlum, and Hadoop with associated numbers.
BIG DATA Processing Engines & Map-Reduce
Current International Standards

ISO/IEC DIS 17788  Information technology - Cloud computing - Overview and vocabulary
ISO/IEC DIS 17789  Information technology - Cloud computing - Reference architecture
ISO/IEC NP 19086  Information technology - Cloud computing - Service level agreement (SLA)

ITU

Y.3501: Cloud computing framework and high-level requirements
Y.3510: Cloud computing infrastructure requirements
Y.3511: Cloud computing - Framework of inter-cloud for network and infrastructure
Y.3520: Cloud computing framework for end to end resource management
Y.3503: Requirement of Desktop as a Service
No Standards Yet

ISO
ISO/IEC JTC 1 Big Data Study Group (SG on BD)

ITU-T SG 13, “Requirements and capabilities for cloud computing based big data” (Y.Bigdata-reqt)

W3C
Big Data Community Group

OASIS
Big Data Technical Committees (AMQP, KVDB, MQTT, XMILE)
**Call for Papers**

**Big Data Computing on Clouds**

Special Issue in IEEE Transactions on Cloud Computing

- Cloud Architecture for Big Data
- Resource scheduling and SLA for Big Data on Cloud
- Storage and computation management in Cloud for Big Data
- Large-scale data intensive workflow in support of Big Data processing on Cloud
- Multiple source data processing and integration on Cloud
- Virtualisation and visualisation of Big Data on Cloud
- Fault tolerance and reliability for Big Data processing on Cloud
- MapReduce with Cloud for Big Data processing
- Distributed file storage system with Cloud for Big Data
- Inter-cloud technology for Big Data
- Security, privacy and trust in Big Data processing on Cloud
- Green, energy-efficient models and sustainability issues in Cloud for Big Data processing
- Cloud infrastructure for social networking with Big Data
- User friendly Cloud access for Big Data processing
- Innovative Cloud data centre networking for Big Data
- Wireless and mobility support in Cloud data centre for Big Data
FEATURES OF CLOUD AND GRID PLATFORMS

• This chapter is devoted to programming real cloud platforms. MapReduce, BigTable, Twister, Dryad, DryadLINQ, Hadoop, Sawzall, and Pig Latin are introduced and assessed.

• We use concrete examples to explain the implementation and application requirements in the cloud. We review core service models and access technologies.

• Cloud services provided by Google App Engine (GAE), Amazon Web Service (AWS), and Microsoft Windows Azure are illustrated by example applications.

• In particular, we illustrate how to programming the GAE, AWS EC2, S3, and EBS. We review the open-source Eucalyptus, Nimbus, and OpenNebula and the startup Manjrasoft Aneka system for cloud computing.

• In four tables, we cover the capabilities, traditional features, data features, and features for programmers and runtime systems to use. The entries in these tables are source references for anyone who wants to program the cloud efficiently.
6.1.1 Cloud Capabilities and Platform Features

- Commercial clouds need broad capabilities, as summarized in Table 6.1. These capabilities offer cost-effective utility computing with the elasticity to scale up and down in power.
- However, as well as this key distinguishing feature, commercial clouds offer a growing number of additional capabilities commonly termed “Platform as a Service” (PaaS).
- For Azure, current platform features include Azure Table, queues, blobs, Database SQL, and web and Worker roles. Amazon is often viewed as offering “just” Infrastructure as a Service (IaaS), but it continues to add platform features including SimpleDB (similar to Azure Table), queues, notification, monitoring, content delivery network, relational database, and MapReduce (Hadoop).
- Google does not currently offer a broad-based cloud service, but the Google App Engine (GAE) offers a powerful web application development environment.
6.1.1 Cloud Capabilities and Platform Features

- Table 6.2 lists some low-level infrastructure features. Table 6.3 lists traditional programming environments for parallel and distributed systems that need to be supported in Cloud environments.
- They can be supplied as part of system (Cloud Platform) or user environment.
- Table 6.4 presents features emphasized by clouds and by some grids. Note that some of the features in Table 6.4 have only recently been offered in a major way.
- In particular, these features are not offered on academic cloud infrastructures such as Eucalyptus, Nimbus, OpenNebula, or Sector/Sphere (although Sector is a data parallel file system or DPFS classified in Table 6.4).
- We will cover these emerging cloud programming environments in Section 6.5.
### 6.1.1 Cloud Capabilities and Platform Features

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical or virtual computing platform</td>
<td>The cloud environment consists of some physical or virtual platforms. Virtual platforms have unique capabilities to provide isolated environments for different applications and users.</td>
</tr>
<tr>
<td>Massive data storage service, distributed file system</td>
<td>With large data sets, cloud data storage services provide large disk capacity and the service interfaces that allow users to put and get data. The distributed file system offers massive data storage service. It can provide similar interfaces as local file systems.</td>
</tr>
<tr>
<td>Massive database storage service</td>
<td>Some distributed file systems are sufficient to provide the underlying storage service application developers need to save data in a more semantic way. Just like DBMS in the traditional software stack, massive database storage services are needed in the cloud.</td>
</tr>
<tr>
<td>Massive data processing method and programming model</td>
<td>Cloud infrastructure provides thousands of computing nodes for even a very simple application. Programmers need to be able to harness the power of these machines without considering tedious infrastructure management issues such as handling network failure or scaling the running code to use all the computing facilities provided by the platforms.</td>
</tr>
<tr>
<td>Workflow and data query language support</td>
<td>The programming model offers abstraction of the cloud infrastructure. Similar to the SQL language used for database systems, in cloud computing, providers have built some workflow language as well as data query language to support better application logic.</td>
</tr>
<tr>
<td>Programming interface and service deployment</td>
<td>Web interfaces or special APIs are required for cloud applications: J2EE, PHP, ASP, or Rails. Cloud applications can use Ajax technologies to improve the user experience while using web browsers to access the functions provided. Each cloud provider opens its programming interface for accessing the data stored in massive storage.</td>
</tr>
<tr>
<td>Runtime support</td>
<td>Runtime support is transparent to users and their applications. Support includes distributed monitoring services, a distributed task scheduler, as well as distributed locking and other services. They are critical in running cloud applications.</td>
</tr>
<tr>
<td>Support services</td>
<td>Important support services include data and computing services. For example, clouds offer rich data services and interesting data parallel execution models like MapReduce.</td>
</tr>
</tbody>
</table>
### 6.1.1 Cloud Capabilities and Platform Features

<table>
<thead>
<tr>
<th><strong>Table 6.2</strong> Infrastructure Cloud Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accounting:</strong> Includes economies; clearly an active area for commercial clouds</td>
</tr>
<tr>
<td><strong>Appliances:</strong> Preconfigured virtual machine (VM) image supporting multifaceted tasks such as message-passing interface (MPI) clusters</td>
</tr>
<tr>
<td><strong>Authentication and authorization:</strong> Could need single sign-on to multiple systems</td>
</tr>
<tr>
<td><strong>Data transport:</strong> Transports data between job components both between and within grids and clouds; exploits custom storage patterns as in BitTorrent</td>
</tr>
<tr>
<td><strong>Operating systems:</strong> Apple, Android, Linux, Windows</td>
</tr>
<tr>
<td><strong>Program library:</strong> Stores images and other program material</td>
</tr>
<tr>
<td><strong>Registry:</strong> Information resource for system (system version of metadata management)</td>
</tr>
<tr>
<td><strong>Security:</strong> Security features other than basic authentication and authorization; includes higher level concepts such as trust</td>
</tr>
<tr>
<td><strong>Scheduling:</strong> Basic staple of Condor, Platform, Oracle Grid Engine, etc.; clouds have this implicitly as is especially clear with Azure Worker Role</td>
</tr>
<tr>
<td><strong>Gang scheduling:</strong> Assigns multiple (data-parallel) tasks in a scalable fashion; note that this is provided automatically by MapReduce</td>
</tr>
<tr>
<td><strong>Software as a Service (SaaS):</strong> Shared between clouds and grids, and can be supported without special attention; Note use of services and corresponding service oriented architectures are very successful and are used in clouds very similarly to previous distributed systems.</td>
</tr>
<tr>
<td><strong>Virtualization:</strong> Basic feature of clouds supporting “elastic” feature highlighted by Berkeley as characteristic of what defines a (public) cloud; includes virtual networking as in ViNe from University of Florida</td>
</tr>
</tbody>
</table>
### 6.1.1 Cloud Capabilities and Platform Features

**Table 6.3 Traditional Features in Cluster, Grid, and Parallel Computing Environments**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cluster management</strong></td>
<td>ROCKS and packages offering a range of tools to make it easy to bring up clusters</td>
</tr>
<tr>
<td><strong>Data management</strong></td>
<td>Included metadata support such as RDF triple stores (Semantic web success and can be built on MapReduce as in SHARD); SQL and NOSQL included in</td>
</tr>
<tr>
<td><strong>Grid programming environment</strong></td>
<td>Varies from link-together services as in Open Grid Services Architecture (OGSA) to GridRPC (Ninf, GridSolve) and SAGA</td>
</tr>
<tr>
<td><strong>OpenMP/thrading</strong></td>
<td>Can include parallel compilers such as Cilk; roughly shared memory technologies. Even transactional memory and fine-grained data flow come here</td>
</tr>
<tr>
<td><strong>Portals</strong></td>
<td>Can be called (science) gateways and see an interesting change in technology from portlets to HUBzero and now in the cloud: Azure Web Roles and GAE</td>
</tr>
<tr>
<td><strong>Scalable parallel computing environments</strong></td>
<td>MPI and associated higher level concepts including ill-fated HP FORTRAN, PGAS (not successful but not disgraced), HPCS languages (X-10, Fortress, Chapel), patterns (including Berkeley dwarves), and functional languages such as F# for distributed memory</td>
</tr>
<tr>
<td><strong>Virtual organizations</strong></td>
<td>From specialized grid solutions to popular Web 2.0 capabilities such as Facebook</td>
</tr>
<tr>
<td><strong>Workflow</strong></td>
<td>Supports workflows that link job components either within or between grids and clouds; relate to LIMS Laboratory Information Management Systems.</td>
</tr>
</tbody>
</table>
### 6.1.1 Cloud Capabilities and Platform Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Blob</strong></td>
<td>Basic storage concept typified by Azure Blob and Amazon S3</td>
</tr>
<tr>
<td><strong>DPFS</strong></td>
<td>Support of file systems such as Google (MapReduce), HDFS (Hadoop), and Cosmos (Dryad) with compute-data affinity optimized for data processing</td>
</tr>
<tr>
<td><strong>Fault tolerance</strong></td>
<td>As reviewed in [1] this was largely ignored in grids, but is a major feature of clouds</td>
</tr>
<tr>
<td><strong>MapReduce</strong></td>
<td>Support MapReduce programming model including Hadoop on Linux, Dryad on Windows HPCS, and Twister on Windows and Linux. Include new associated languages such as Sawzall, Pregel, Pig Latin, and LINQ</td>
</tr>
<tr>
<td><strong>Monitoring</strong></td>
<td>Many grid solutions such as Inca. Can be based on publish-subscribe</td>
</tr>
<tr>
<td><strong>Notification</strong></td>
<td>Basic function of publish-subscribe systems</td>
</tr>
<tr>
<td><strong>Programming model</strong></td>
<td>Cloud programming models are built with other platform features and are related to familiar web and grid models</td>
</tr>
<tr>
<td><strong>Queues</strong></td>
<td>Queuing system possibly based on publish-subscribe</td>
</tr>
<tr>
<td><strong>Scalable synchronization</strong></td>
<td>Apache Zookeeper or Google Chubby. Supports distributed locks and used by BigTable. Not clear if (effectively) used in Azure Table or Amazon SimpleDB</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Relational database</td>
</tr>
<tr>
<td><strong>Table</strong></td>
<td>Support of table data structures modeled on Apache Hbase or Amazon SimpleDB/Azure Table. Part of NOSQL movement</td>
</tr>
<tr>
<td><strong>Web role</strong></td>
<td>Used in Azure to describe important link to user and can be supported otherwise with a portal framework. This is the main purpose of GAE</td>
</tr>
<tr>
<td><strong>Worker role</strong></td>
<td>Implicitly used in both Amazon and grids but was first introduced as a high-level construct by Azure</td>
</tr>
</tbody>
</table>
Traditional Features Common to Grids and Clouds

Workflow

• As introduced before, workflow has spawned many projects in the United States and Europe.
• Pegasus, Taverna, and Kepler are popular, but no choice has gained wide acceptance. There are commercial systems such as Pipeline Pilot, AVS (dated), and the LIMS environments.
• A recent entry is Trident [2] from Microsoft Research which is built on top of Windows Workflow Foundation.
• If Trident runs on Azure or just any old Windows machine, it will run workflow proxy services on external (Linux) environments.
• Workflow links multiple cloud and non-cloud services in real applications on demand.
Traditional Features Common to Grids and Clouds

Data Transport

- The cost (in time and money) of data transport in (and to a lesser extent, out of) commercial clouds is often discussed as a difficulty in using clouds. If commercial clouds become an important component of the national cyberinfrastructure, we can expect that high-bandwidth links will be made available between clouds and TeraGrid.

- The special structure of cloud data with blocks (in Azure blobs) and tables could allow high-performance parallel algorithms, but initially, simple HTTP mechanisms are used to transport data [3–5] on academic systems/TeraGrid and commercial clouds.
Traditional Features Common to Grids and Clouds

- **Security, Privacy, and Availability**
  The following techniques are related to security, privacy, and availability requirements for developing a healthy and dependable cloud programming environment.
  - Use virtual clustering to achieve dynamic resource provisioning with minimum overhead cost.
  - Use stable and persistent data storage with fast queries for information retrieval.
  - Use special APIs for authenticating users and sending e-mail using commercial accounts.
  - Cloud resources are accessed with security protocols such as HTTPS and SSL.
  - Fine-grained access control is desired to protect data integrity and deter intruders or hackers.
  - Shared data sets are protected from malicious alteration, deletion, or copyright violations.
  - Features are included for availability enhancement and disaster recovery with life migration of VMs.
  - Use a reputation system to protect data centers. This system only authorizes trusted clients and stops pirates.
Data Features and Databases

In the following paragraphs, we review interesting programming features related to the program library, blobs, drives, DPFS, tables, and various types of databases including SQL, NOSQL, and non-relational databases and special queuing services.

Program Library

- Many efforts have been made to design a VM image library to manage images used in academic and commercial clouds. The basic cloud environments described in this chapter also include many management features allowing convenient deployment and configuring of images (i.e., they support IaaS).

Blobs and Drives

- The basic storage concept in clouds is blobs for Azure and S3 for Amazon. These can be organized (approximately, as in directories) by containers in Azure. In addition to a service interface for blobs and S3, one can attach “directly” to compute instances as Azure drives and the Elastic Block Store for Amazon. This concept is similar to shared file systems such as Lustre used in TeraGrid. The cloud storage is intrinsically fault-tolerant while that on TeraGrid needs backup storage. However, the architecture ideas are similar between clouds and TeraGrid, and the Simple Cloud File Storage API [6] could become important here.
Data Features and Databases

DPFS:

- This covers the support of file systems such as Google File System (MapReduce), HDFS (Hadoop), and Cosmos (Dryad) with compute-data affinity optimized for data processing. It could be possible to link DPFS to basic blob and drive-based architecture, but it’s simpler to use DPFS as an application-centric storage model with compute-data affinity and blobs and drives as the repository-centric view.

- In general, data transport will be needed to link these two data views. It seems important to consider this carefully, as DPFS file systems are precisely designed for efficient execution of data-intensive applications. However, the importance of DPFS for linkage with Amazon and Azure is not clear, as these clouds do not currently offer fine-grained support for compute-data affinity. We note here that Azure Affinity Groups are one interesting capability.

- We expect that initially blobs, drives, tables, and queues will be the areas where academic systems will most usefully provide a platform similar to Azure (and Amazon). Note the HDFS (Apache) and Sector (UIC) projects in this area.
Data Features and Databases

SQL and Relational Databases:

• Both Amazon and Azure clouds offer relational databases and it is straightforward for academic systems to offer a similar capability unless there are issues of huge scale where, in fact, approaches based on tables and/or MapReduce might be more appropriate [8].

• As one early user, we are developing on FutureGrid a new private cloud computing model for the Observational Medical Outcomes Partnership (OMOP) for patient-related medical data which uses Oracle and SAS where FutureGrid is adding Hadoop for scaling to many different analysis methods.

• Note that databases can be used to illustrate two approaches to deploying capabilities.
  • Traditionally, one would add database software to that found on computer disks. This software is executed, providing your database instance. However, on Azure and Amazon, the database is installed on a separate VM independent from your job (worker roles in Azure). This implements “SQL as a Service.”
  • It may have some performance issues from the messaging interface, but the “aaS” deployment clearly simplifies one’s system. For N platform features, one only needs N services, whereas number of possible images with alternative approaches is a prohibitive 2^N.
Data Features and Databases

Table and NOSQL Non-relational Databases

- A substantial number of important developments have occurred regarding simplified database structures—termed “NOSQL” [9,10]—typically emphasizing distribution and scalability.
- These are present in the three major clouds:
- Tables are clearly important in science as illustrated by the VOTable standard in astronomy [14] and the popularity of Excel. However, there does not appear to be substantial experience in using tables outside clouds.
- There are, of course, many important uses of non-relational databases, especially in terms of triple stores for metadata storage and access. Recently, there has been interests in building scalable RDF triple stores based on MapReduce and tables or the Hadoop File System [8,15], with early success reported on very large stores.
Data Features and Databases

Table and NOSQL Non-relational Databases

The current cloud tables fall into two groups:

- Azure Table and Amazon SimpleDB are quite similar [16] and support lightweight storage for “document stores,” while BigTable aims to manage large distributed data sets without size limitations.

- All these tables are schema-free (each record can have different properties), although BigTable has a schema for column (property) families. It seems likely that tables will grow in importance for scientific computing, and academic systems could support this using two Apache projects:
Data Features and Databases

Queuing Services:
- Both Amazon and Azure offer similar scalable, robust queuing services that are used to communicate between the components of an application.
- The messages are short (less than 8 KB) and have a Representational State Transfer (REST) service interface with “deliver at least once” semantics.
- They are controlled by timeouts for posting the length of time allowed for a client to process.
- One can build a similar approach (on the typically smaller and less challenging academic environments), basing it on publish-subscribe systems such as ActiveMQ [20] or Narada Brokering [21,22] with which we have substantial experience.
Programming and Runtime Support

Programming and runtime support are desired to facilitate parallel programming and provide runtime support of important functions in today’s grids and clouds. Various MapReduce systems are reviewed in this section.

MapReduce

- There has been substantial interest in “data parallel” languages largely aimed at loosely coupled computations which execute over different data samples.
- The language and runtime generate and provide efficient execution of “many task” problems that are well known as successful grid applications.
- However, MapReduce, summarized in Table 6.5, has several advantages over traditional implementations for many task problems, as it supports dynamic execution, strong fault tolerance, and an easy-to-use high-level interface.
- The major open source/commercial MapReduce implementations are Hadoop [23] and Dryad [24–27] with execution possible with or without VMs.
Programming and Runtime Support

• **MapReduce**
  - Hadoop is currently offered by Amazon, and we expect Dryad to be available on Azure. A prototype Azure MapReduce was built at Indiana University, which we will discuss shortly.
  - On FutureGrid, we already intend to support Hadoop, Dryad, and other MapReduce approaches, including Twister [29] support for iterative computations seen in many data-mining and linear algebra applications.
  - Note that this approach has some similarities with Cloudera [35] which offers a variety of Hadoop distributions including Amazon and Linux.
  - MapReduce is closer to broad deployment than other cloud platform features, as there is quite a bit of experience with Hadoop and Dryad outside clouds.
## Programming and Runtime Support

### Table 6.5 Comparison of MapReduce Type Systems

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Programming Model</strong></td>
<td>MapReduce</td>
<td>MapReduce</td>
<td>DAG execution, extensible to MapReduce and other patterns</td>
<td>Iterative MapReduce</td>
<td>Currently just MapReduce; will extend to Iterative MapReduce</td>
</tr>
<tr>
<td><strong>Data Handling</strong></td>
<td>GFS (Google File System)</td>
<td>HDFS (Hadoop Distributed File System)</td>
<td>Shared directories and local disks</td>
<td>Local disks and data management tools</td>
<td>Azure blob storage</td>
</tr>
<tr>
<td><strong>Scheduling</strong></td>
<td>Data locality</td>
<td>Data locality; rack-aware, dynamic task scheduling using global queue</td>
<td>Data locality; network topology optimized at runtime; static task partitions</td>
<td>Data locality; static task partitions</td>
<td>Dynamic task scheduling through global queue</td>
</tr>
<tr>
<td><strong>Failure Handling</strong></td>
<td>Reexecution of failed tasks; duplicated execution of slow tasks</td>
<td>Reexecution of failed tasks; duplicated execution of slow tasks</td>
<td>Reexecution of failed tasks; duplicated execution of slow tasks</td>
<td>Reexecution of iterations</td>
<td>Reexecution of failed tasks; duplicated execution of slow tasks</td>
</tr>
<tr>
<td><strong>HLL Support</strong></td>
<td>Sawzall [31]</td>
<td>Pig Latin [32,33]</td>
<td>DryadLINQ [27]</td>
<td>Pregel [34] has related features</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Environment</strong></td>
<td>Linux cluster</td>
<td>Linux clusters, Amazon Elastic MapReduce on EC2</td>
<td>Windows HPCS cluster</td>
<td>Linux cluster, EC2</td>
<td>Windows Azure, Azure Local Development Fabric</td>
</tr>
<tr>
<td><strong>Intermediate Data Transfer</strong></td>
<td>File</td>
<td>File, HTTP</td>
<td>File, TCP pipes, shared-memory FIFOs</td>
<td>Publish-subscribe messaging</td>
<td>Files, TCP</td>
</tr>
</tbody>
</table>


Programming and Runtime Support

Cloud Programming Models

• In many ways, most of the previous sections describe programming model features, but these are “macroscopic” constructs and do not address, for example, the coding (language and libraries).

• Both the GAE and Manjrasoft Aneka environments represent programming models; both are applied to clouds, but are really not specific to this architecture.

• Iterative MapReduce is an interesting programming model that offers portability between cloud, HPC and cluster environments.
Programming and Runtime Support

SaaS:

• Services are used in a similar fashion in commercial clouds and most modern distributed systems.

• We expect users to package their programs wherever possible, so no special support is needed to enable SaaS.

• We already discussed in Section 6.1.3 why “Systems software as a service” was an interesting idea in the context of a database service. We desire a SaaS environment that provides many useful tools to develop cloud applications over large data sets.

• In addition to the technical features, such as MapReduce, BigTable, EC2, S3, Hadoop, AWS, GAE, and WebSphere2, we need protection features that may help us to achieve scalability, security, privacy, and availability.
PARALLEL AND DISTRIBUTED PROGRAMMING PARADIGMS

- We define a parallel and distributed program as a parallel program running on a set of computing engines or a distributed computing system.
- The term carries the notion of two fundamental terms in computer science:
  - **distributed computing system and parallel computing**.
- A distributed computing system is a set of computational engines connected by a network to achieve a common goal of running a job or an application. A computer cluster or network of workstations is an example of a distributed computing system.
- **Parallel computing** is the simultaneous use of more than one computational engine (not necessarily connected via a network) to run a job or an application. For instance, parallel computing may use either a distributed or a non-distributed computing system such as a multiprocessor platform.
PARALLEL AND DISTRIBUTED PROGRAMMING PARADIGMS

• Running a parallel program on a distributed computing system (parallel and distributed programming) has several advantages for both users and distributed computing systems.
• From the users’ perspective, it decreases application response time.
• From the distributed computing systems’ standpoint, it increases throughput and resource utilization.
• Running a parallel program on a distributed computing system; however, could be a very complicated process. Data flow of running a typical parallel program on a distributed system is further explained in this chapter.
PARALLEL AND DISTRIBUTED PROGRAMMING PARADIGMS
Parallel Computing and Programming Paradigms

- Consider a distributed computing system consisting of a set of networked nodes or workers. The system issues for running a typical parallel program in either a parallel or a distributed manner would include the following [36–39]:

**Partitioning:** This is applicable to both computation and data as follows:

- **Computation partitioning:** This splits a given job or a program into smaller tasks. Partitioning greatly depends on correctly identifying portions of the job or program that can be performed concurrently. In other words, upon identifying parallelism in the structure of the program, it can be divided into parts to be run on different workers. Different parts may process different data or a copy of the same data.

- **Data partitioning:** This splits the input or intermediate data into smaller pieces. Similarly, upon identification of parallelism in the input data, it can also be divided into pieces to be processed on different workers. Data pieces may be processed by different parts of a program or a copy of the same program.

**Mapping:** This assigns the either smaller parts of a program or the smaller pieces of data to underlying resources. This process aims to appropriately assign such parts or pieces to be run simultaneously on different workers and is usually handled by resource allocators in the system.
PARALLEL AND DISTRIBUTED PROGRAMMING PARADIGMS
Parallel Computing and Programming Paradigms

Synchronization: Because different workers may perform different tasks, synchronization and coordination among workers is necessary so that race conditions are prevented and data dependency among different workers is properly managed. Multiple accesses to a shared resource by different workers may raise race conditions, whereas data dependency happens when a worker needs the processed data of other workers.

Communication: Because data dependency is one of the main reasons for communication among workers, communication is always triggered when the intermediate data is sent to workers.

Scheduling: For a job or program, when the number of computation parts (tasks) or data pieces is more than the number of available workers, a scheduler selects a sequence of tasks or data pieces to be assigned to the workers. It is worth noting that the resource allocator performs the actual mapping of the computation or data pieces to workers, while the scheduler only picks the next part from the queue of unassigned tasks based on a set of rules called the scheduling policy. For multiple jobs or programs, a scheduler selects a sequence of jobs or programs to be run on the distributed computing system. In this case, scheduling is also necessary when system resources are not sufficient to simultaneously run multiple jobs or programs.
6.2.2 MapReduce, Twister, and Iterative MapReduce

- MapReduce, as introduced in Section 6.1.4, is a software framework which supports parallel and distributed computing on large data sets [27,37,45,46]. This software framework abstracts the data flow of running a parallel program on a distributed computing system by providing users with two interfaces in the form of two functions: Map and Reduce. Users can override these two functions to interact with and manipulate the data flow of running their programs.

- Figure 6.1 illustrates the logical data flow from the Map to the Reduce function in MapReduce frameworks. In this framework, the “value” part of the data, (key, value), is the actual data, and the “key” part is only used by the MapReduce controller to control the data flow [37].
6.2.2 MapReduce, Twister, and Iterative MapReduce

FIGURE 6.1
MapReduce framework: Input data flows through the Map and Reduce functions to generate the output result under the control flow using MapReduce software library. Special user interfaces are used to access the Map and Reduce resources.
MapReduce, Twister, and Iterative MapReduce
Formal Definition of MapReduce

- The MapReduce software framework provides an abstraction layer with the data flow and flow of control to users, and hides the implementation of all data flow steps such as data partitioning, mapping, synchronization, communication, and scheduling. Here, although the data flow in such frameworks is predefined, the abstraction layer provides two well-defined interfaces in the form of two functions: Map and Reduce [47].
- These two main functions can be overridden by the user to achieve specific objectives.
- Figure 6.1 shows the MapReduce framework with data flow and control flow. Therefore, the user overrides the Map and Reduce functions first and then invokes the provided MapReduce (Spec, & Results) function from the library to start the flow of data. The MapReduce function, MapReduce (Spec, & Results), takes an important parameter which is a specification object, the Spec.
- This object is first initialized inside the user’s program, and then the user writes code to fill it with the names of input and output files, as well as other optional tuning parameters.
- This object is also filled with the name of the Map and Reduce functions to identify these user defined functions to the MapReduce library.
MapReduce, Twister, and Iterative MapReduce

Formal Definition of MapReduce

• The overall structure of a user’s program containing the Map, Reduce, and the Main functions is given below.

• The Map and Reduce are two major subroutines. They will be called to implement the desired function performed in the main program.

```java
Map Function ..... 
{ 
    ........
} 
Reduce Function ..... 
{ 
    ........
} 
Main Function ..... 
{ 
    Initialize Spec object 
    .......
    MapReduce (Spec, & Results) 
}
```
MapReduce, Twister, and Iterative MapReduce

MapReduce Logical Data Flow

• The input data to both the Map and the Reduce functions has a particular structure. This also pertains for the output data.
• The input data to the Map function is in the form of a (key, value) pair.
• For example, the key is the line offset within the input file and the value is the content of the line.
• The output data from the Map function is structured as (key, value) pairs called intermediate (key, value) pairs.
• In other words, the user-defined Map function processes each input (key, value) pair and produces a number of (zero, one, or more) intermediate (key, value) pairs.
• Here, the goal is to process all input (key, value) pairs to the Map function in parallel (Figure 6.2).
• In turn, the Reduce function receives the intermediate (key, value) pairs in the form of a group of intermediate values associated with one intermediate key, (key, [set of values]).
• In fact, the MapReduce framework forms these groups by first sorting the intermediate (key, value) pairs and then grouping values with the same key. It should be noted that the data is sorted to simplify the grouping process.
• The Reduce function processes each (key, [set of values]) group and produces a set of (key, value) pairs as output.
MapReduce, Twister, and Iterative MapReduce
MapReduce Logical Data Flow

**Figure 6.2**
MapReduce logical data flow in 5 processing stages over successive (key, value) pairs.
MapReduce, Twister, and Iterative MapReduce

MapReduce Logical Data Flow

To clarify the data flow in a sample MapReduce application, one of the well-known MapReduce problems, namely word count, to count the number of occurrences of each word in a collection of documents is presented here. Figure 6.3 demonstrates the data flow of the word-count problem for a simple input file containing only two lines as follows: (1) “most people ignore most poetry” and (2) “most poetry ignores most people.” In this case, the Map function simultaneously produces a number of intermediate (key, value) pairs for each line of content so that each word is the intermediate key with 1 as its intermediate value; for example, (ignore, 1). Then the MapReduce library collects all the generated intermediate (key, value) pairs and sorts them to group the 1’s for identical words; for example, (people, [1,1]). Groups are then sent to the Reduce function in parallel so that it can sum up the 1 values for each word and generate the actual number of occurrence for each word in the file; for example, (people, 2).
MapReduce, Twister, and Iterative MapReduce

MapReduce Logical Data Flow

MapReduce logical data flow in 5 processing stages over successive (key, value) pairs.

- Most people ignore most poetry
- Most poetry ignores most people

**Map**
- (most, 1)
- (people, 1)
- (ignore, 1)
- (ignore, 1)
- (most, 1)
- (most, 1)
- (poetry, 1)
- (ignore, 1)
- (most, 1)

**Intermediate (key, val) pairs**
- (ignore, 1)
- (ignores, 1)
- (most, 1)
- (most, 1)
- (most, 1)
- (most, 1)
- (people, 1)
- (people, 1)

**Unique keys**
- (ignore, 1)
- (ignores, 1)
- (most, 4)
- (people, 2)
- (poetry, 2)

**Reduce**
- (ignore, 1)
- (ignores, 1)
- (most, 1)
- (people, 1)
- (poetry, 1)

**FIGURE 6.3**
The data flow of a word-count problem using the MapReduce functions (Map, Sort, Group and Reduce) in a cascade operations.
MapReduce, Twister, and Iterative MapReduce

Formal Notation of MapReduce Data Flow

• The Map function is applied in parallel to every input (key, value) pair, and produces new set of intermediate (key, value) pairs [37] as follows:

\[(key_1, val_1) \xrightarrow{\text{Map Function}} \text{List}(key_2, val_2)\]

• Then the MapReduce library collects all the produced intermediate (key, value) pairs from all input (key, value) pairs, and sorts them based on the “key” part. It then groups the values of all occurrences of the same key.

• Finally, the Reduce function is applied in parallel to each group producing the collection of values as output as illustrated here:

\[(key_2, \text{List}(val_2)) \xrightarrow{\text{Reduce Function}} \text{List}(val_2)\]
MapReduce, Twister, and Iterative MapReduce
Strategy to Solve MapReduce Problems

- As mentioned earlier, after grouping all the intermediate data, the values of all occurrences of the same key are sorted and grouped together. As a result, after grouping, each key becomes unique in all intermediate data. Therefore, finding unique keys is the starting point to solving a typical MapReduce problem. Then the intermediate (key, value) pairs as the output of the Map function will be automatically found.
- The following three examples explain how to define keys and values in such problems:
  - Problem 1: Counting the number of occurrences of each word in a collection of documents.
    - Solution: unique “key”: each word, intermediate “value”: number of occurrences
  - Problem 2: Counting the number of occurrences of words having the same size, or the same number of letters, in a collection of documents
    - Solution: unique “key”: each word, intermediate “value”: size of the word
  - Problem 3: Counting the number of occurrences of anagrams in a collection of documents. Anagrams are words with the same set of letters but in a different order (e.g., the words “listen” and “silent”).
    - Solution: unique “key”: alphabetically sorted sequence of letters for each word (e.g., “eilnst”), intermediate “value”: number of occurrences.
MapReduce, Twister, and Iterative MapReduce
MapReduce Actual Data and Control Flow

• The main responsibility of the MapReduce framework is to efficiently run a user’s program on a distributed computing system. Therefore, the MapReduce framework meticulously handles all partitioning, mapping, synchronization, communication, and scheduling details of such data flows [48,49].
• We summarize this in the following distinct steps:
  1. Data partitioning The MapReduce library splits the input data (files), already stored in GFS, into M pieces that also correspond to the number of map tasks.
  2. Computation partitioning This is implicitly handled (in the MapReduce framework) by obliging users to write their programs in the form of the Map and Reduce functions. Therefore, the MapReduce library only generates copies of a user program (e.g., by a fork system call) containing the Map and the Reduce functions, distributes them, and starts them up on a number of available computation engines.
  3. Determining the master and workers The MapReduce architecture is based on a master worker model. Therefore, one of the copies of the user program becomes the master and the rest become workers. The master picks idle workers, and assigns the map and reduce tasks to them.
  A map/reduce worker is typically a computation engine such as a cluster node to run map/reduce tasks by executing Map/Reduce functions. Steps 4–7 describe the map workers.
  4. Reading the input data (data distribution) Each map worker reads its corresponding portion of the input data, namely the input data split, and sends it to its Map function. Although a map worker may run more than one Map function, which means it has been assigned more than one input data split, each worker is usually assigned one input split only.
  5. Map function Each Map function receives the input data split as a set of (key, value) pairs to process and produce the intermediated (key, value) pairs.
MapReduce, Twister, and Iterative MapReduce

MapReduce Actual Data and Control Flow

6. Combiner function This is an optional local function within the map worker which applies to intermediate (key, value) pairs. The user can invoke the Combiner function inside the user program. The Combiner function runs the same code written by users for the Reduce function as its functionality is identical to it. The Combiner function merges the local data of each map worker before sending it over the network to effectively reduce its communication costs.

As mentioned in our discussion of logical data flow, the MapReduce framework sorts and groups the data before it is processed by the Reduce function. Similarly, the MapReduce framework will also sort and group the local data on each map worker if the user invokes the Combiner function.

7. Partitioning function As mentioned in our discussion of the MapReduce data flow, the intermediate (key, value) pairs with identical keys are grouped together because all values inside each group should be processed by only one Reduce function to generate the final result.

However, in real implementations, since there are M map and R reduce tasks, intermediate (key, value) pairs with the same key might be produced by different map tasks, although they should be grouped and processed together by one Reduce function only.

Therefore, the intermediate (key, value) pairs produced by each map worker are partitioned into R regions, equal to the number of reduce tasks, by the Partitioning function to guarantee that all (key, value) pairs with identical keys are stored in the same region. As a result, since reduce worker $i$ reads the data of region $i$ of all map workers, all (key, value) pairs with the same key will be gathered by reduce worker $i$ accordingly (see Figure 6.4).

To implement this technique, a Partitioning function could simply be a hash function (e.g., Hash(key) mod R) that forwards the data into particular regions. It is also worth noting that the locations of the buffered data in these R partitions are sent to the master for later forwarding of data to the reduce workers.
MapReduce, Twister, and Iterative MapReduce

MapReduce Actual Data and Control Flow

**FIGURE 6.4**

Use of MapReduce *partitioning* function to link the Map and Reduce workers.
MapReduce, Twister, and Iterative MapReduce

MapReduce Actual Data and Control Flow

- Figure 6.5 shows the data flow implementation of all data flow steps. The following are two networking steps:
- 8. Synchronization MapReduce applies a simple synchronization policy to coordinate map workers with reduce workers, in which the communication between them starts when all map tasks finish.
- 9. Communication Reduce worker i, already notified of the location of region i of all map workers, uses a remote procedure call to read the data from the respective region of all map workers.
- Since all reduce workers read the data from all map workers, all-to-all communication among all map and reduce workers, which incurs network congestion, occurs in the network. This issue is one of the major bottlenecks in increasing the performance of such systems [50–52].
- A data transfer module was proposed to schedule data transfers independently [55].
MapReduce, Twister, and Iterative MapReduce

MapReduce Actual Data and Control Flow

• Steps 10 and 11 correspond to the reduce worker domain:

  10. Sorting and Grouping When the process of reading the input data is finalized by a reduce worker, the data is initially buffered in the local disk of the reduce worker. Then the reduce worker groups intermediate (key, value) pairs by sorting the data based on their keys, followed by grouping all occurrences of identical keys. Note that the buffered data is sorted and grouped because the number of unique keys produced by a map worker may be more than R regions in which more than one key exists in each region of a map worker (see Figure 6.4).

  11. Reduce function The reduce worker iterates over the grouped (key, value) pairs, and for each unique key, it sends the key and corresponding values to the Reduce function. Then this function processes its input data and stores the output results in predetermined files in the user’s program.

• To better clarify the interrelated data control and control flow in the MapReduce framework, Figure 6.6 shows the exact order of processing control in such a system contrasting with dataflow in Figure 6.5.
MapReduce, Twister, and Iterative MapReduce

MapReduce Actual Data and Control Flow

**Figure 6.5**

Data flow implementation of many functions in the Map workers and in the Reduce workers through multiple sequences of partitioning, combining, synchronization and communication, sorting and grouping, and reduce operations.
MapReduce, Twister, and Iterative MapReduce
MapReduce Actual Data and Control Flow

**FIGURE 6.6**
Control flow implementation of the MapReduce functionalities in Map workers and Reduce workers (running user programs) from input files to the output files under the control of the master user program.

(Courtesy of Yahoo! Pig Tutorial [54])
MapReduce, Twister, and Iterative MapReduce
Compute-Data Affinity

• The MapReduce software framework was first proposed and implemented by Google. The first implementation was coded in C. The implementation takes advantage of GFS [53] as the underlying layer. MapReduce could perfectly adapt itself to GFS. GFS is a distributed file system where files are divided into fixed-size blocks (chunks) and blocks are distributed and stored on cluster nodes.

• As stated earlier, the MapReduce library splits the input data (files) into fixed-size blocks, and ideally performs the Map function in parallel on each block. In this case, as GFS has already stored files as a set of blocks, the MapReduce framework just needs to send a copy of the user’s program containing the Map function to the nodes’ already stored data blocks. This is the notion of sending computation toward data rather than sending data toward computation. Note that the default GFS block size is 64 MB which is identical to that of the MapReduce framework.
MapReduce, Twister, and Iterative MapReduce

Twister and Iterative MapReduce

• It is important to understand the performance of different runtimes and, in particular, to compare MPI and MapReduce [43,44,55,56].

• The two major sources of parallel overhead are load imbalance and communication (which is equivalent to synchronization overhead as communication synchronizes parallel units [threads or processes] in Categories 2 and 6 of Table 6.10).

• The communication overhead in MapReduce can be quite high, for two reasons:

  • MapReduce reads and writes via files, whereas MPI transfers information directly between nodes over the network.
  • MPI does not transfer all data from node to node, but just the amount needed to update information. We can call the MPI flow δ flow and the MapReduce flow full data flow.

The same phenomenon is seen in all “classic parallel” loosely synchronous applications which typically exhibit an iteration structure over compute phases followed by communication phases.
MapReduce, Twister, and Iterative MapReduce

Twister and Iterative MapReduce

We can address the performance issues with two important changes:

• 1. Stream information between steps without writing intermediate steps to disk.
• 2. Use long-running threads or processors to communicate the $\delta$ (between iterations) flow.

These changes will lead to major performance increases at the cost of poorer fault tolerance and ease to support dynamic changes such as the number of available nodes.

This concept [42] has been investigated in several projects [34,57–59] while the direct idea of using MPI for MapReduce applications is investigated in [44].
MapReduce, Twister, and Iterative MapReduce
Twister and Iterative MapReduce

The Twister programming paradigm and its implementation architecture at runtime are illustrated in Figure 6.7(a, b).

In Example 6.1, we summarize Twister [60] whose performance results for K means are shown in Figure 6.8 [55,56], where Twister is much faster than traditional MapReduce. Twister distinguishes the static data which is never reloaded from the dynamic $\delta$ flow that is communicated.
MapReduce, Twister, and Iterative MapReduce

Twister and Iterative MapReduce

(a) Twister for iterative MapReduce programming

(b) Architecture of Twister at runtime

FIGURE 6.7
Twister: An iterative MapReduce programming paradigm for repeated MapReduce executions.
Example 6.1 Performance of K Means Clustering in MPI, Twister, Hadoop, and DryadLINQ

- Example 6.1 Performance of K Means Clustering in MPI, Twister, Hadoop, and DryadLINQ. The MapReduce approach leads to fault tolerance and flexible scheduling, but for some applications the performance degradation compared to MPI is serious, as illustrated in Figure 6.8 for a simple parallel K means clustering algorithm.

- Hadoop and DryadLINQ are more than a factor of 10 slower than MPI for the largest data set, and perform even more poorly for smaller data sets.

- One could use many communication mechanisms in Iterative MapReduce, but Twister chose a publish-subscribe network using a distributed set of brokers, as described in Section 5.2 with similar performance achieved with ActiveMQ and NaradaBrokering.

- The Map-Reduce pair is iteratively executed in long-running threads.

- We compare in Figure 6.9. the different thread and process structures of 4 parallel programming paradigms: namely Hadoop, Dryad, Twister (also called MapReduce++), and MPI.

- Note that Dryad can use pipes and avoids costly disk writing according to the original papers [26,27].
Example 6.1 Performance of K Means Clustering in MPI, Twister, Hadoop, and DryadLINQ
Example 6.1 Performance of K Means Clustering in MPI, Twister, Hadoop, and DryadLINQ

**FIGURE 6.9**
Thread and process structure of four parallel programming paradigms at runtimes.
Example 6.2 Performance of Hadoop and Twister on ClueWeb Data Set over 256 Processor Cores

Important research areas for Iterative MapReduce include fault tolerance and scalable approaches to communication.

Figure 6.10 shows [55] that iterative algorithms are found in information retrieval.

This figure shows the famous Page Rank algorithm (with a kernel of iterative matrix vector multiplication) run on the public ClueWeb data sets, and independent of size, Twister is about 20 times faster than Hadoop as revealed by the gap between the top and lower curves in Figure 6.10.
Example 6.2 Performance of Hadoop and Twister on ClueWeb Data Set over 256 Processor Cores

**Figure 6.10**
Performance of Hadoop and Twister on ClueWeb data set using 256 processing cores.
Hadoop Library from Apache

- Hadoop is an open source implementation of MapReduce coded and released in Java (rather than C) by Apache. The Hadoop implementation of MapReduce uses the Hadoop Distributed File System (HDFS) as its underlying layer rather than GFS.
- The Hadoop core is divided into two fundamental layers: the MapReduce engine and HDFS. The MapReduce engine is the computation engine running on top of HDFS as its data storage manager.
- The following two sections cover the details of these two fundamental layers.
Hadoop Library from Apache

- **HDFS**: HDFS is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.

- **HDFS Architecture**: HDFS has a master/slave architecture containing a single NameNode as the master and a number of DataNodes as workers (slaves).

- To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes).

- The mapping of blocks to DataNodes is determined by the NameNode. The NameNode (master) also manages the file system’s metadata and namespace.

- In such systems, the namespace is the area maintaining the metadata, and metadata refers to all the information stored by a file system that is needed for overall management of all files.

- For example, NameNode in the metadata stores all information regarding the location of input splits/blocks in all DataNodes. Each DataNode, usually one per node in a cluster, manages the storage attached to the node. Each DataNode is responsible for storing and retrieving its file blocks [61].
Hadoop Library from Apache

**HDFS Fault Tolerance:** One of the main aspects of HDFS is its fault tolerance characteristic. Since Hadoop is designed to be deployed on low-cost hardware by default, a hardware failure in this system is considered to be common rather than an exception. Therefore, Hadoop considers the following issues to fulfill reliability requirements of the file system [64]:

- **Block replication:** To reliably store data in HDFS, file blocks are replicated in this system. In other words, HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster. The replication factor is set by the user and is three by default.

- **Replica placement:** The placement of replicas is another factor to fulfill the desired fault tolerance in HDFS. Although storing replicas on different nodes (DataNodes) located in different racks across the whole cluster provides more reliability, it is sometimes ignored as the cost of communication between two nodes in different racks is relatively high in comparison with that of different nodes located in the same rack. Therefore, sometimes HDFS compromises its reliability to achieve lower communication costs. For example, for the default replication factor of three, HDFS stores one replica in the same node the original data is stored, one replica on a different node but in the same rack, and one replica on a different node in a different rack to provide three copies of the data [65].

- **Heartbeat and Blockreport messages:** Heartbeats and Blockreports are periodic messages sent to the NameNode by each DataNode in a cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly, while each Blockreport contains a list of all blocks on a DataNode [65]. The NameNode receives such messages because it is the sole decision maker of all replicas in the system.
Hadoop Library from Apache

- **HDFS High-Throughput Access to Large Data Sets (Files):** Because HDFS is primarily designed for batch processing rather than interactive processing, data access throughput in HDFS is more important than latency. Also, because applications run on HDFS typically have large data sets, individual files are broken into large blocks (e.g., 64MB) to allow HDFS to decrease the amount of metadata storage required per file.

- **This provides two advantages:** The list of blocks per file will shrink as the size of individual blocks increases, and by keeping large amounts of data sequentially within a block, HDFS provides fast streaming reads of data.
Hadoop Library from Apache

- **HDFS Operation**: The control flow of HDFS operations such as write and read can properly highlight roles of the NameNode and DataNodes in the managing operations.

- In this section, the control flow of the main operations of HDFS on files is further described to manifest the interaction between the user, the NameNode, and the DataNodes in such systems [63].

- **Reading a file**: To read a file in HDFS, a user sends an “open” request to the NameNode to get the location of file blocks.

- For each file block, the NameNode returns the address of a set of DataNodes containing replica information for the requested file.

- The number of addresses depends on the number of block replicas. Upon receiving such information, the user calls the read function to connect to the closest DataNode containing the first block of the file.

- After the first block is streamed from the respective DataNode to the user, the established connection is terminated and the same process is repeated for all blocks of the requested file until the whole file is streamed to the user.
Hadoop Library from Apache

- **Writing to a file**: To write a file in HDFS, a user sends a “create” request to the NameNode to create a new file in the file system namespace. If the file does not exist, the NameNode notifies the user and allows him to start writing data to the file by calling the write function.

- The first block of the file is written to an internal queue termed the data queue while a data streamer monitors its writing into a DataNode.

- Since each file block needs to be replicated by a predefined factor, the data streamer first sends a request to the NameNode to get a list of suitable DataNodes to store replicas of the first block.

- The steamer then stores the block in the first allocated DataNode. Afterward, the block is forwarded to the second DataNode by the first DataNode.

- The process continues until all allocated DataNodes receive a replica of the first block from the previous DataNode. Once this replication process is finalized, the same process starts for the second block and continues until all blocks of the file are stored and replicated on the file system.
Hadoop Library from Apache
Architecture of MapReduce in Hadoop

• The topmost layer of Hadoop is the MapReduce engine that manages the data flow and control flow of MapReduce jobs over distributed computing systems.

• Figure 6.11 shows the MapReduce engine architecture cooperating with HDFS. Similar to HDFS, the MapReduce engine also has a master/slave architecture consisting of a single JobTracker as the master and a number of TaskTrackers as the slaves (workers).

• The JobTracker manages the MapReduce job over a cluster and is responsible for monitoring jobs and assigning tasks to TaskTrackers. The TaskTracker manages the execution of the map and/or reduce tasks on a single computation node in the cluster.

• Each TaskTracker node has a number of simultaneous execution slots, each executing either a map or a reduce task. Slots are defined as the number of simultaneous threads supported by CPUs of the TaskTracker node.

• For example, a TaskTracker node with N CPUs, each supporting M threads, has M * N simultaneous execution slots [66].

• It is worth noting that each data block is processed by one map task running on a single slot. Therefore, there is a one-to-one correspondence between map tasks in a TaskTracker and data blocks in the respective DataNode.
Hadoop Library from Apache
Architecture of MapReduce in Hadoop

FIGURE 6.11
HDFS and MapReduce architecture in Hadoop where boxes with different shadings refer to different functional nodes applied to different blocks of data.
Hadoop Library from Apache
Running a Job in Hadoop

• Three components contribute in running a job in this system: a user node, a JobTracker, and several TaskTrackers.
• The data flow starts by calling the runJob(conf) function inside a user program running on the user node, in which conf is an object containing some tuning parameters for the MapReduce framework and HDFS.
• The runJob(conf) function and conf are comparable to the MapReduce(Spec, &Results) function and Spec in the first implementation of MapReduce by Google. Figure 6.12 depicts the data flow of running a MapReduce job in Hadoop [63].
• Job Submission Each job is submitted from a user node to the JobTracker node that might be situated in a different node within the cluster through the following procedure:
  • A user node asks for a new job ID from the JobTracker and computes input file splits.
  • The user node copies some resources, such as the job’s JAR file, configuration file, and computed input splits, to the JobTracker’s file system.
  • The user node submits the job to the JobTracker by calling the submitJob() function.
Hadoop Library from Apache

Running a Job in Hadoop

• **Task assignment:** The JobTracker creates one map task for each computed input split by the user node and assigns the map tasks to the execution slots of the TaskTrackers. The JobTracker considers the localization of the data when assigning the map tasks to the TaskTrackers. The JobTracker also creates reduce tasks and assigns them to the TaskTrackers. The number of reduce tasks is predetermined by the user, and there is no locality consideration in assigning them.

• **Task execution:** The control flow to execute a task (either map or reduce) starts inside the TaskTracker by copying the job JAR file to its file system. Instructions inside the job JAR file are executed after launching a Java Virtual Machine (JVM) to run its map or reduce task.

• **Task running check:** A task running check is performed by receiving periodic heartbeat messages to the JobTracker from the TaskTrackers. Each heartbeat notifies the JobTracker that the sending TaskTracker is alive, and whether the sending TaskTracker is ready to run a new task.
Hadoop Library from Apache

Running a Job in Hadoop

**FIGURE 6.12**

Data flow in running a MapReduce job at various task trackers using the Hadoop library.
Dryad and DryadLINQ from Microsoft

Dryad

• Two runtime software environments are reviewed in this section for parallel and distributed computing, namely the Dryad and DryadLINQ, both developed by Microsoft.
• Dryad is more flexible than MapReduce as the data flow of its applications is not dictated/predetermined and can be easily defined by users.
• To achieve such flexibility, a Dryad program or job is defined by a directed acyclic graph (DAG) where vertices are computation engines and edges are communication channels between vertices. Therefore, users or application developers can easily specify arbitrary DAGs to specify data flows in jobs.
• Given a DAG, Dryad assigns the computational vertices to the underlying computation engines (cluster nodes) and controls the data flow through edges (communication between cluster nodes).
• Data partitioning, scheduling, mapping, synchronization, communication, and fault tolerance are major implementation details hidden by Dryad to facilitate its programming environment.
• Because the data flow of a job is arbitrary for this system, only the control flow of the runtime environment is further explained here.
• As shown in Figure 6.13(a), the two main components handling the control flow of Dryad are the job manager and the name server.
Dryad and DryadLINQ from Microsoft

Dryad

- In Dryad, the distributed job is represented as a DAG where each vertex is a program and edges represent data channels. Thus, the whole job will be constructed by the application programmer who defines the processing procedures as well as the flow of data.
- This logical computation graph will be automatically mapped onto the physical nodes by the Dryad runtime.
- A Dryad job is controlled by the job manager, which is responsible for deploying the program to the multiple nodes in the cluster.
- It runs either within the computing cluster or as a process in the user’s workstation which can access the cluster. The job manager has the code to construct the DAG as well as the library to schedule the work running on top of the available resources.
- Data transfer is done via channels without involving the job manager. Thus, the job manager should not be the performance bottleneck.
Dryad and DryadLINQ from Microsoft

Dryad

- In summary, the job manager
- 1. Constructs a job’s communication graph (data flow graph) using the application-specific program provided by the user.
- 2. Collects the information required to map the data flow graph to the underlying resources (computation engine) from the name server.

- The cluster has a name server which is used to enumerate all the available computing resources in the cluster.
- Thus, the job manager can contact the name server to get the topology of the whole cluster and make scheduling decisions.
- A processing daemon runs in each computing node in the cluster.
- The binary of the program will be sent to the corresponding processing node directly from the job manager.
- The daemon can be viewed as a proxy so that the job manager can communicate with the remote vertices and monitor the state of the computation.
- By gathering this information, the name server provides the job manager with a perfect view of the underlying resources and network topology.
Dryad and DryadLINQ from Microsoft

Dryad

- Therefore, the job manager is able to:
- 1. Map the data flow graph to the underlying resources.
- 2. Schedule all necessary communications and synchronization across the respective resources.
- It also considers data and computation locality when mapping the data flow graph to the underlying resources [26].
- When the data flow graph is mapped on a set of computation engines, a light daemon runs on each cluster node to run the assigned tasks.
- Each task is defined by the user using an application-specific program.
- During runtime, the job manager communicates with each daemon to monitor the state of the computation of the node and its communication with its preceding and succeeding nodes.
- At runtime, the channels are used to transport the structured items between vertices which represent the processing programs.
- There are several types of communication mechanisms for implementing channels such as shared memory, TCP sockets, or even distributed file systems.
- The execution of a Dryad job can be considered a 2D distributed set of pipes.
Dryad and DryadLINQ from Microsoft

Dryad

- As there are many nodes in the cluster, the job manager can choose another node to re-execute the corresponding job assigned to the failed node.

- In case of an edge failure, the vertex that created the channel will be re-executed and a new channel will be created and touch the corresponding nodes again.

- Dryad provides other mechanisms in addition to the runtime graph refinements which are used for improving execution performance.

- As a general framework, Dryad can be used in many situations, including scripting language support, map-reduce programming, and SQL service integration.
Dryad and DryadLINQ from Microsoft

Dryad

![Diagram showing Dryad control and data flow](image-url)
Dryad and DryadLINQ from Microsoft

Dryad

FIGURE 6.13
Dryad framework and its job structure, control and data flow.

(Courtesy of Isard, et al., ACM SIGOPS Operating Systems Review, 2007 [26])
Dryad and DryadLINQ from Microsoft

DryadLINQ from Microsoft

- DryadLINQ is built on top of Microsoft’s Dryad execution framework (see http://research.microsoft.com/en-us/projects/DryadLINQ/).
- Dryad can perform acyclic task scheduling and run on large-scale servers.
- The goal of DryadLINQ is to make large-scale, distributed cluster computing available to ordinary programmers.
- Actually, DryadLINQ, as the name implies, combines two important components:
  - the Dryad distributed execution engine and .NET Language Integrated Query (LINQ).
  - LINQ is particularly for users familiar with a database programming model.
- Figure 6.14 shows the flow of execution with DryadLINQ.
Dryad and DryadLINQ from Microsoft

**FIGURE 6.14**
LINQ-expression execution in DryadLINQ.

(Courtesy of Yu, et al. [27])
The execution is divided into nine steps as follows:

1. A .NET user application runs, and creates a DryadLINQ expression object. Because of LINQ’s deferred evaluation, the actual execution of the expression has not occurred.

2. The application calls ToDryadTable triggering a data-parallel execution. The expression object is handed to DryadLINQ.

3. DryadLINQ compiles the LINQ expression into a distributed Dryad execution plan. The expression is decomposed into subexpressions, each to be run in a separate Dryad vertex. Code and static data for the remote Dryad vertices are generated, followed by the serialization code for the required data types.

4. DryadLINQ invokes a custom Dryad job manager which is used to manage and monitor the execution flow of the corresponding task.

5. The job manager creates the job graph using the plan created in step 3. It schedules and spawns the vertices as resources become available.
Dryad and DryadLINQ from Microsoft

DryadLINQ from Microsoft

- 6. Each Dryad vertex executes a vertex-specific program.
- 7. When the Dryad job completes successfully it writes the data to the out table(s).
- 8. The job manager process terminates, and it returns control back to DryadLINQ. DryadLINQ creates the local DryadTable objects encapsulating the output of the execution. The DryadTable objects here might be the input to the next phase.
- 9. Control returns to the user application. The iterator interface over a DryadTable allows the user to read its contents as .NET objects.

- Not all programs go through all nine steps. Some programs may go through fewer steps. Based on the preceding description, DryadLINQ enables users to integrate their current programming language (C#) with a compiler and a runtime execution engine. The following example shows how to write a histogram in DryadLINQ.