**Data Intensive Computing:**

**Map-Reduce Programming**

* Focuses on class of applications that deal with a large amount of data.
* Computational science to social networking, produce large volumes of data that need to be efficiently stored, made accessible, indexed, and analyzed.
* Distributed computing is definitely of help in addressing these challenges .
* This chapter characterizes nature of data-intensive computing and presents an overview of the challenges of large volumes of data and how they are handled.
* MapReduce.
* Concerned with production, manipulation, and analysis of large-scale data in MB to PB or beyond.
* **Dataset** is used to identify a collection of information that is relevant to one or more applications.
* Datasets are maintained in repositories.
* **Metadata** are attached to datasets.
* Data-intensive computations occur in many application domains.
* **Computational science** - People conducting scientific simulations and experiments produce, analyze, and process huge volumes of data.
* **Bioinformatics** applications mine databases that may end up containing terabytes of data.
* **Earthquake simulators** process a massive amount of data, which is produced as a result of recording the vibrations of the Earth across the entire globe.

##### Characterizing data-intensive computations

* + 1. **Challenges ahead**

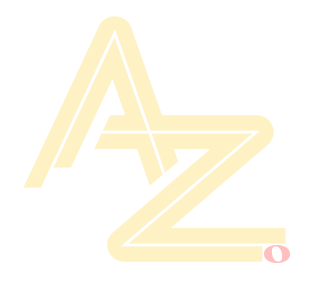
##### Historical perspective

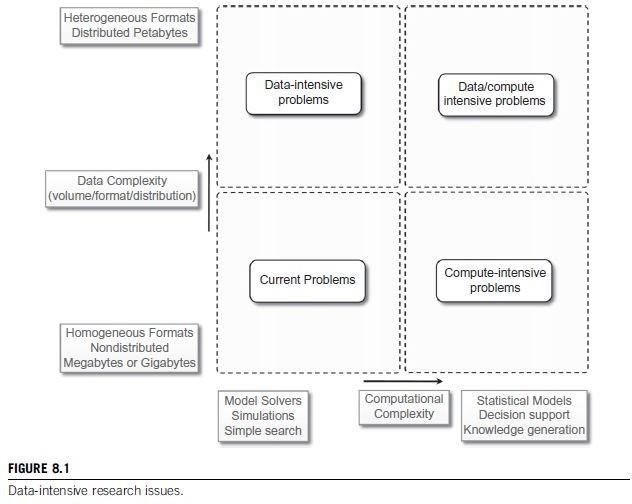
1 The early age: high-speed wide-area networking 2 Data grids

1. Data clouds and “Big Data”
2. Databases and data-intensive computing

##### Characterizing data-intensive computations

* Deals with huge volumes of data, also exhibit compute-intensive properties.
* Handle datasets on the scale of multiple terabytes and petabytes.
* Applications process data in multistep analytical pipelines, including transformation and fusion stages.





##### Challenges ahead

* Huge amount of data produced, analyzed, or stored imposes requirements on infrastructures and middleware that are least found in the traditional solutions.
* Moving terabytes of data.
* **Data partitioning, content replication** and **scalable algorithms** help in improving performance.

##### Challenges ahead

**Open challenges** in data-intensive computing given by Ian Gorton et al.

1. Scalable algorithms that can search and process massive datasets.
2. New metadata management technologies that can handle complex, heterogeneous, and distributed data sources.
3. Advances in high-performance computing platforms for accessing in-memory multi terabyte data structures.
4. High-performance, highly reliable, petascale distributed file systems.
5. Data signature-generation techniques for data reduction and rapid processing.
6. Software mobility to move computation where data is present.
7. Interconnected architectures that provide support for filtering multi gigabyte datastreams.
8. Software integration techniques that facilitate the combination of software modules running on different platforms.

##### Historical perspective

1. **The early age: high-speed wide-area networking**
   * In 1989, the first experiments in high-speed networking as a support for remote visualization of scientific data.
   * Two years later, TCP/IP-based distributed applications was demonstrated at Supercomputing 1991 (SC91).

##### Wide Area Large Data Object (WALDO) system:

* + - auto- matic generation of metadata;
    - automatic cataloguing of data and metadata;
    - facilitation of cooperative research by providing local and remote users access to data;
  + mechanisms to incorporate data into databases and other documents.
  + **Distributed Parallel Storage System (DPSS)** to support TerraVision, a terrain visualization application that lets users explore and navigate a tridimensional real landscape.
  + **Clipper project:** a collection of independent, architecturally consistent service components to support data-intensive computing.
  + Clipper’s main focus to develop collection of services for applications to build

##### on-demand, large-scale, high-performance, wide-area problem-solving

**environments.**

##### 8.1.3 Historical perspective

1. **Data grids**

A data grid provides services that help users discover, transfer, and manipulate large datasets stored in distributed repositories as well as create and manage copies of them.

two main functionalities: high-performance and reliable file transfer.

Data grids mostly provide storage and dataset management facilities as support for scientific experiments

Heterogeneity and security.

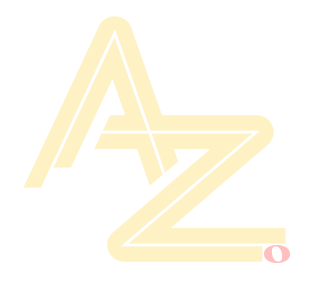
Data grids have their own characteristics and introduce new challenges:

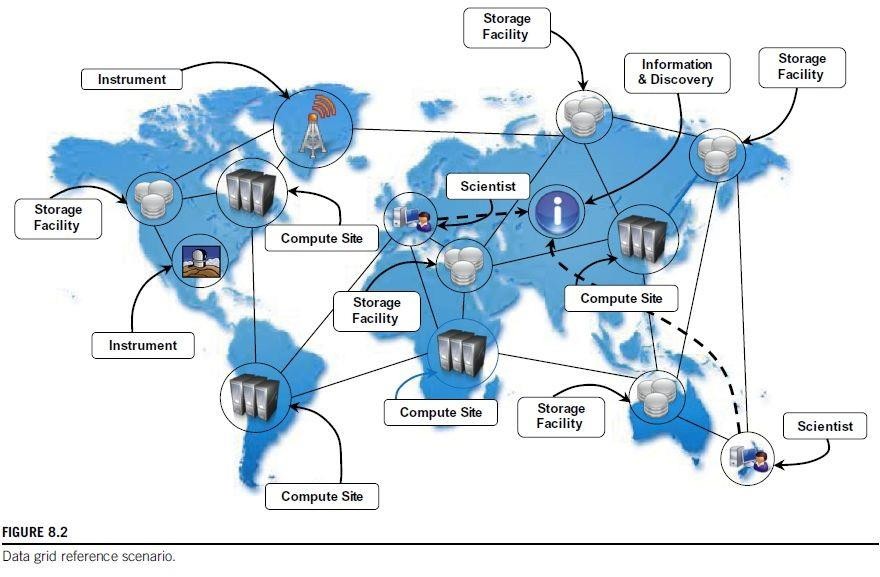
##### Massive datasets.

* 1. **Shared data collections.**

##### Unified namespace.

* 1. **Access restrictions.**





##### Data clouds and “Big Data”

* Big Data - characterizes the nature of data- intensive computations and identifies datasets that grow large they become complex to work using DBMS tools.
* The term Big Data applies to datasets of which the size is beyond the ability of commonly used software tools to capture, manage, and process within a tolerable elapsed time.

Cloud technologies support data-intensive computing in several ways:

1. By providing a large amount of compute instances on demand, which can be used to process and analyze large datasets in parallel.
2. By providing a storage system optimized for keeping large blobs of data and other distributed data store architectures.
3. By providing frameworks and programming APIs optimized for the processing and management of large amounts of data.

##### 3 Data clouds and “Big Data”

A datacloud is a combination of above 3 components.

Ex 1: MapReduce framework, which provides the best performance for leveraging Google File System on top of Google’s large computing infrastructure.

Ex 2: Hadoop system, the most mature, large, and open-source datacloud. It consists of Hadoop Distributed File System (HDFS) and Hadoop’s implementation of MapReduce.

Ex 3: Sector, which consists of Sector Distributed File System (SDFS) and compute service called Sphere, that allows users to execute arbitrary user-defined functions (UDFs) over the data managed by SDFS.

##### Databases and data-intensive computing

* Traditional distributed database using DBMS less computation power.
* Distributed databases are a collection of data stored at different sites of a computer network.
* A distributed database can be created by splitting and scattering the data of an existing database over different sites.
* These systems are very robust and provide distributed transaction processing, distributed query optimization, and efficient management of resources.

Concerns the development of applications that are mainly focused on processing large quantities of data.

##### Storage systems

* + - 1. High-performance distributed file systems and storage clouds
      2. NoSQL systems

##### Programming platforms

* + - 1. The MapReduce programming model.
      2. Variations and extensions of MapReduce.
      3. Alternatives to MapReduce.

#### Storage systems

* + - * DBMS constituted **de facto** storage support.
      * Due to the explosion of unstructured data - blogs, Web pages, software logs, and sensor readings, **the relational model** is not preferred solution.

Some factors contributing to change in data are:

1. Growing of popularity of Big Data.
2. Growing importance of data analytics in the business chain.
3. Presence of data in several forms, not only structured.
4. New approaches and technologies for computing.

##### 8.2.1 Storage systems

**1. High-performance distributed file systems and storage clouds.**

* Distributed file systems constitute primary support for data management.
* They provide interface to store information in the form of files
* later access them for read and write operations.

##### Lustre.

* + massively parallel distributed file system
  + covers small workgroup of clusters to a large-scale computing cluster.
  + The file system is used by several of the Top 500 supercomputing systems.
  + Designed to provide access to petabytes (PBs) of storage to serve thousands of clients with an I/O throughput of hundreds of gigabytes per second (GB/s).
  + The system is composed of **metadata server** that contains the metadata about the file system and collection of **object storage servers** that are in charge of providing storage.

##### 8.2.1 Storage systems

**1. High-performance distributed file systems and storage clouds.**

##### IBM General Parallel File System (GPFS).

* + GPFS is high-performance distributed file system.
  + provides support for RS/6000 supercomputer and Linux computing clusters.
  + Multiplatform distributed file system.
  + Concept of shared disks, in which a collection of disks is attached to the file system nodes by switching fabric.
  + The file system makes this infrastructure transparent to users and stripes large files over the disk array by replicating portions of the file to ensure high availability.
  + Eliminates a single point of failure.

##### 8.2.1 Storage systems

1.**High-performance distributed file systems and storage clouds.**

##### Google File System (GFS)

storage infrastructure supports execution of distributed applications in Google’s computing cloud.

GFS is designed with the following assumptions:

1. System is built on top of commodity hardware that often fails.
2. System stores large files; multi-GB files are common and should be treated efficiently, and small files must be supported.
3. Workloads has two kinds of reads: large streaming reads and small random reads.
4. Workloads also have large, sequential writes that append data to files.
5. High-sustained bandwidth is more important than low latency.

Architecture organized into a **single master**, which contains the metadata of file system, and collection of **chunk servers**, which provide storage space..

##### 8.2.1 Storage systems

1.**High-performance distributed file systems and storage clouds.**

##### Sector.

Storage cloud that supports execution of data-intensive applications defined according to **Sphere framework.**

Architecture is composed of four nodes:

* + security server,
  + master nodes,
  + slave nodes, and
  + client machines.

**Security server** maintains all the information about access control policies for user and files,

**Master servers** coordinate and serve the I/O requests of **clients**, which interact with **Slave nodes** to access files.

#### 8.2.1 Storage systems

##### High-performance distributed file systems and storage clouds.

1. **Amazon Simple Storage Service (S3)**.

* Amazon S3 is the online storage service provided by Amazon.
* System offers flat storage space organized into buckets, which are attached to an Amazon Web Services (AWS) account.
* Each bucket can store multiple objects, identified by a unique key.

#### 8.2.1 Storage systems

##### NoSQL systems

* + **Not Only SQL (NoSQL)** identify set of UNIX shell scripts and commands to operate on text files containing the actual data.
  + NoSQL cannot be considered a relational database, it is a collection of scripts that allow users to manage database tasks by using text files as information stores.

Some prominent implementations that support data-intensive applications:

##### Apache CouchDB and MongoDB.

1. **Amazon Dynamo.**

##### Google Bigtable.

1. **Apache Cassandra.**

##### Hadoop HBase.

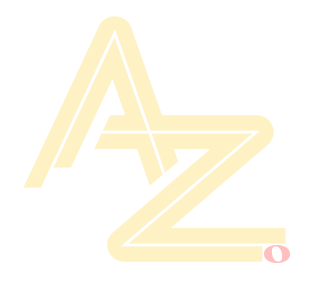
1. **NoSQL systems**

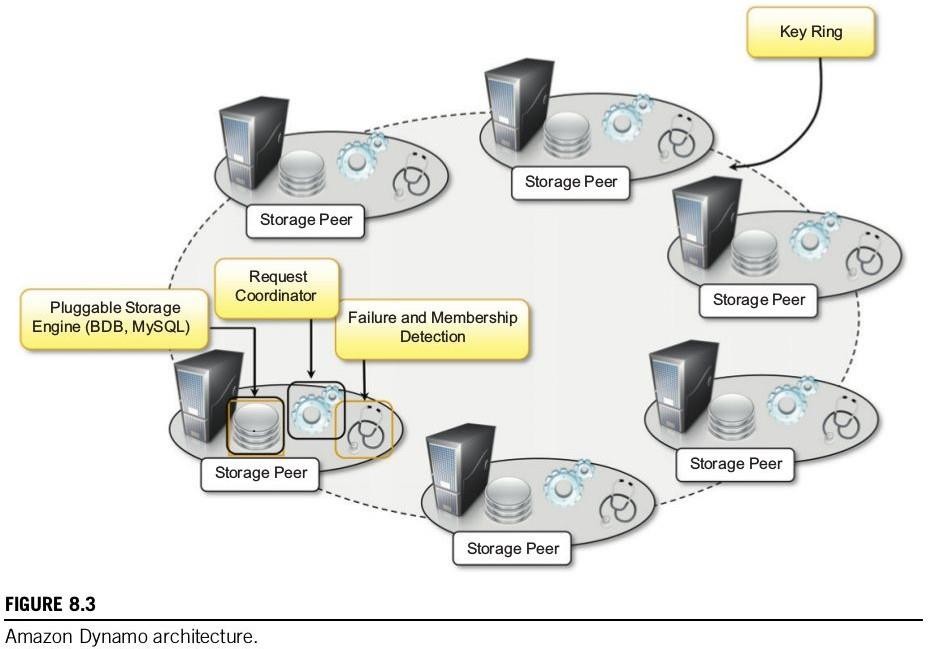
##### Apache CouchDB and MongoDB.

* + - Both provide a schema-less store whereby the primary objects are documents organized into a collection of key-value fields.
    - The value of each field can be of type string, integer, float, date, or an array of values.
    - RESTful interface and represent data in JSON format.
    - querying and indexing data by using the MapReduce programming model.
    - JavaScript as a base language.
    - support large files as documents.

##### 2. NoSQL systems

* 1. **Amazon Dynamo.**
     + provide an incrementally scalable and highly available storage system.
     + provides a simplified interface based on get/put semantics, where objects are stored and retrieved with a unique identifier (key).





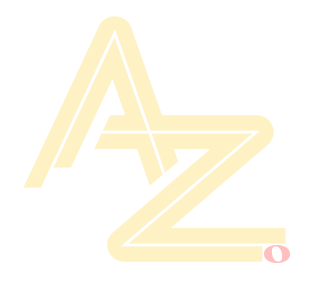
## Technologies for data-intensive computing

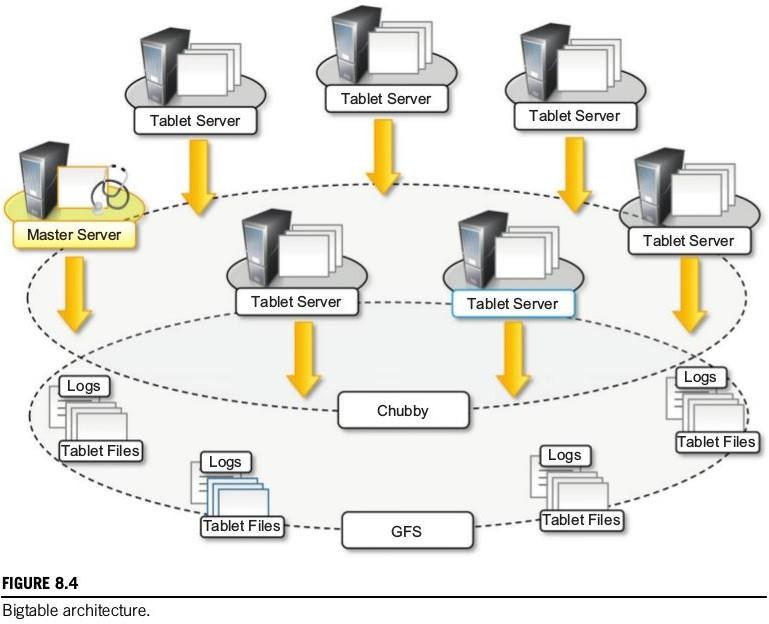
##### Storage systems

**2. NoSQL systems**

##### Google Bigtable.

* + - provides storage support: from **throughput-oriented batch-processing jobs** to **latency-sensitive serving** of data.
    - design goals: wide applicability, scalability, high performance, and high availability.
    - Bigtable organizes the data storage in tables of which the rows are distributed over the distributed file system supporting the middleware, i.e Google File System.
    - A table is organized into rows and columns;
    - columns can be grouped in column family, which allow for specific optimization for better access control, storage and indexing of data.





##### 2. NoSQL systems

* 1. **Apache Cassandra.**
     + designed to avoid a single point of failure and offer highly reliable service.
     + developed by Facebook; now it is part of Apache incubator initiative.
     + provides storage support for Web applications such as Facebook, Digg, Twitter.
     + concept: tables implemented as a distributed multidimensional map indexed by a key.
     + value corresponding to key is structured object and constitutes the row of a table.
     + Cassandra organizes the row of a table into columns, and sets of columns can be grouped into column families.
     + APIs provided by the system to access and manipulate data: insertion, retrieval, and deletion.
     + The insertion is performed at the row level; retrieval and deletion operate at

column level.

##### 2. NoSQL systems

* 1. **Hadoop HBase.**
     + inspiration from Google Bigtable.
     + offer real-time read/write operations for tables with billions of rows and millions of columns.
     + architecture and logic model of HBase is very similar to Google Bigtable, and the entire system is backed by the Hadoop Distributed File System (HDFS).
     1. **Programming platforms**

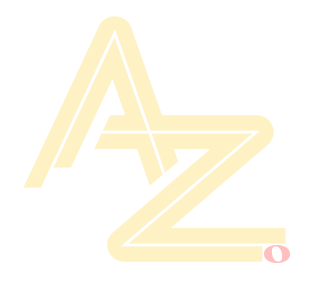
focus on the processing of data and move into the runtime system the management of transfers, thus making the data always available where needed.

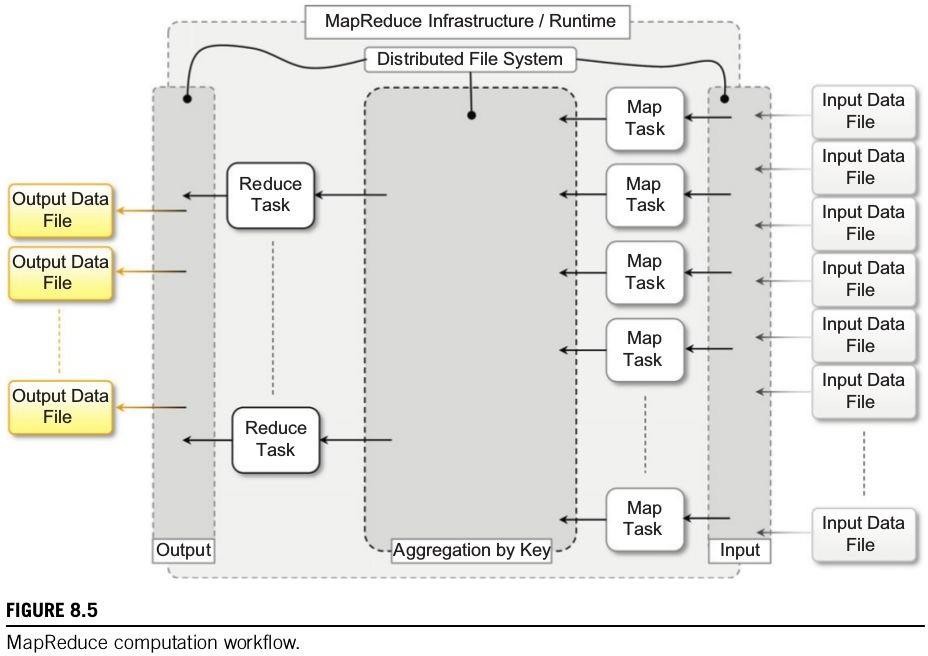
approach followed by the MapReduce programming platform.

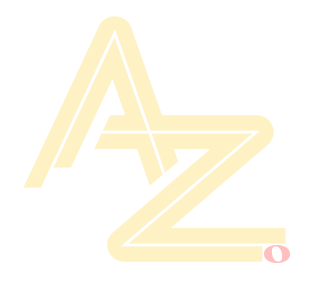
### 8.2.2 Programming platforms

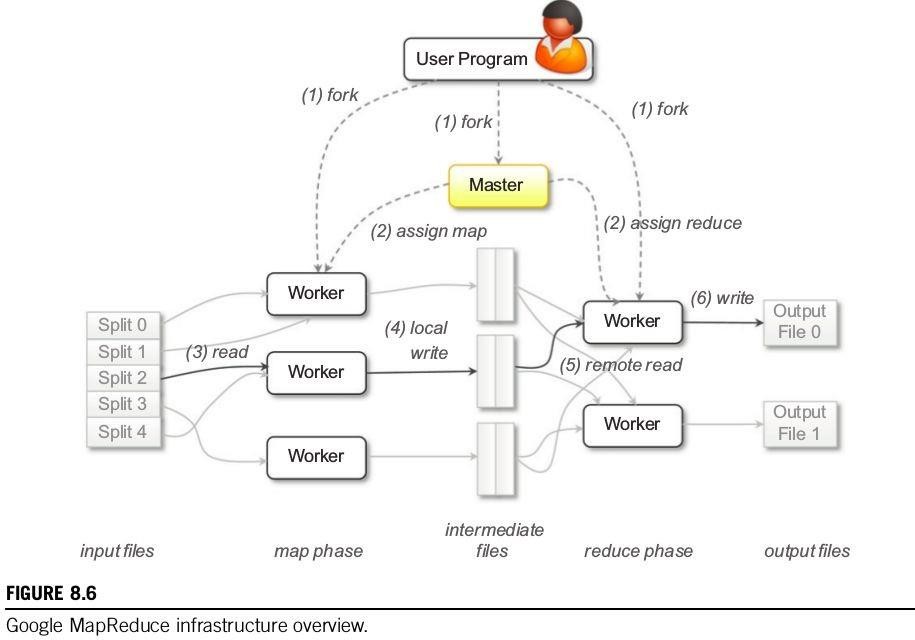
##### The MapReduce programming model.

* + the computational logic of an application in two simple functions: map and reduce.
  + Data transfer and management are completely handled by the distributed storage infrastructure (i.e., the Google File System)
  + map function reads a key-value pair and produces a list of key-value pairs of different types.
  + reduce function reads a pair composed of a key and a list of values and produces a list of values of the same type.
  + (k1,v1,k2,kv2) hints how these two functions are connected and are executed.









### 8.2.2 Programming platforms

##### The MapReduce programming model.

* + **The master process** is in charge of controlling the execution of map and reduce tasks, partitioning, and reorganizing the intermediate output
  + **The worker processes** are used to host the execution of map and reduce tasks and provide basic I/O faciities

##### Variations and extensions of MapReduce.

1. **Hadoop.**

##### Pig.

1. **Hive.**

##### Map-Reduce-Merge.

1. **Twister.**

##### Hadoop.

* + collection of software projects for reliable and scalable distributed computing.
  + Hadoop Distributed File System (HDFS) and Hadoop MapReduce.
  + Same as Google File System & Google MapReduce.

##### Pig.

* + analysis of large datasets.
  + Pig consists of a high-level language for data analysis programs, coupled with infrastructure.
  + compiler for high-level language that produces sequence of MapReduce jobs.

##### 2. Variations and extensions of MapReduce.

1. **Hive.**
   * data warehouse infrastructure on top of Hadoop MapReduce.
   * tools for easy data summarization, ad hoc queries, and analysis of large datasets
   * major advantages is ability to scale out, since it is based on the Hadoop framework.

##### Map-Reduce-Merge.

* + a third phase — Merge phase—that allows efficiently merging data already partitioned and sorted.
  + simplifies the management of heterogeneous related datasets.
  + provides abstraction to express the common relational algebra operators & several join algorithms.

##### 2. Variations and extensions of MapReduce.

1. **Twister.**
   * iterative executions of MapReduce jobs.
   * Twister proposes the following extensions:
2. Configure Map
3. Configure Reduce
4. While Condition Holds True Do
   1. Run MapReduce
   2. Apply Combine Operation to Result
   3. Update Condition
5. Close
   * additional phase called combine, run at the end of MapReduce job, that aggregates output together.

##### Alternatives to MapReduce.

* 1. **Sphere.**

##### All-Pairs.

* 1. **DryadLINQ.**

##### Sphere.

* + Sector Distributed File System (SDFS).
  + implements the **stream processing model** (**Single Program, Multiple Data**)
  + developers to express the computation in terms of **user-defined functions (UDFs)**
  + built on top of Sector’s API.
  + execution of UDFs is achieved through **Sphere Process Engines (SPEs)**,
  + Sphere **client** sends a request for processing to the **master** node,
  + returns the list of available slaves,
  + client will choose the slaves on which to execute Sphere processes.

##### 3. Alternatives to MapReduce.

1. **All-Pairs.**

implements All-pairs function

##### All-pairs(A:set; B:set; F:function) -> M:matrix

The All-pairs function can be easily solved by the following algorithm:

1. For each $i in A
2. For each $j in B
3. Submit job F $i $j

Execution of a distributed application is developed in four stages:

1. model the system;
2. distribute the data;
3. dispatch batch jobs; and
4. clean up the system.

Ex1: field of biometrics, similarity matrices are composed by comparison of several images.

##### 3. Alternatives to MapReduce.

Ex2: applications and algorithms in data mining.

##### 3. Alternatives to MapReduce.

1. **DryadLINQ.**
   * Microsoft Research project
   * investigates programming models for writing parallel and distributed programs to scale from a small cluster to a large datacenter.
   * Developers express distributed applications as set of sequential programs connected by channels.
   * Computation is expressed as directed acyclic graph - nodes are sequential programs and vertices are channels connecting programs.
   * Superset of the MapReduce model.

Aneka provides an implementation of MapReduce introduced by **Google** and implemented by **Hadoop**.

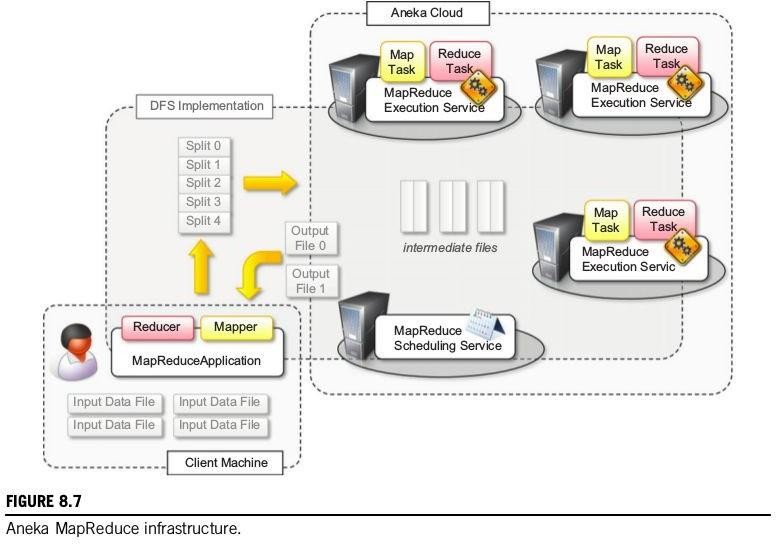
* + 1. **Introducing the MapReduce programming model**
       1. Programming abstractions
       2. Runtime support
       3. Distributed file system support

#### Example application

1. Parsing Aneka logs
2. Mapper design and implementation 3 Reducer design and implementation 4 Driver program

5 Running the application

#### 8.3.1 Introducing the MapReduce programming model

defines the abstractions and runtime support for developing MapReduce applications on top of Aneka.

#### 8.3.1 Introducing the MapReduce programming model

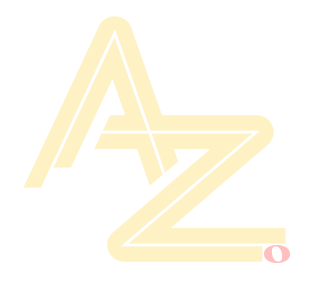
The runtime support is composed of **three** main elements:

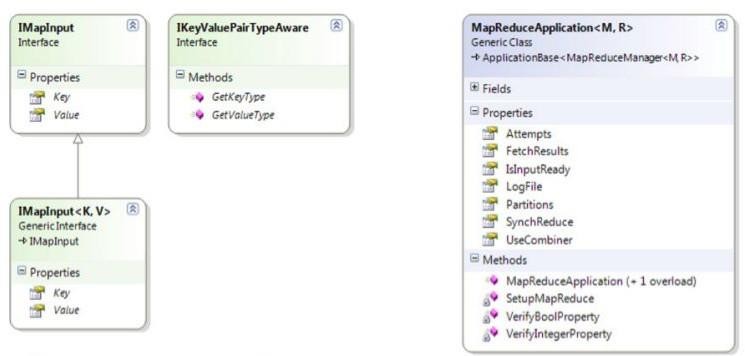
1. **MapReduce Scheduling Service**, which plays the role of the master process in the Google and Hadoop implementation.
2. **MapReduce Execution Service**, which plays the role of the worker process in the Google and Hadoop implementation.
3. A specialized **distributed file system** that is used to move data files.
   * Client components, MapReduce Application, are used to submit the execution of a MapReduce job, upload data files, and monitor it.
   * Local data files are automatically uploaded to Aneka, and output files are automatically downloaded to client machine.

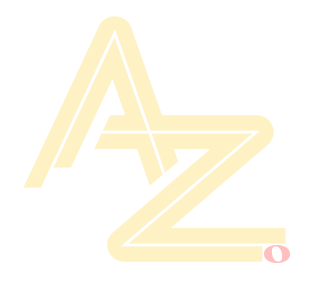
#### 8.3.1 Introducing the MapReduce programming model

##### 1 Programming abstractions

* Aneka executes any piece of user code within distributed application.
* task creation is responsibility of the infrastructure once the user has defined the map and reduce functions.









#### 8.3.1 Introducing the MapReduce programming model

##### Programming abstractions

**Listing 8.1** definition of the Mapper<K,V> class and of the related types that developers should be aware of for implementing the map function.

**Listing 8.2** implementation of the Mapper<K,V> component for Word Counter sample.

**Listing 8.3** definition of Reducer<K,V> class. The implementation of a specific reducer requires specializing the generic class and overriding the abstract method: Reduce(IReduceInputEnumerator<V> input).

**Listing 8.4** implement the reducer function for word-counter example.

**Listing 8.5** interface of MapReduceApplication<M,R>.

**Listing 8.6** displays collection of methods that are of interest in this class for execution of MapReduce jobs.

**Listing 8.7** MapReduce application for running the word-counter example defined by the previous WordCounterMapper and WordCounterReducer classes.

#### 8.3.1 Introducing the MapReduce programming model

##### Runtime support

comprises the collection of services that deal with scheduling and executing MapReduce tasks.

These are

the MapReduce Scheduling Service and the MapReduce Execution Service.

**Job and Task Scheduling.** responsibility of the MapReduce Scheduling Service. covers the same role as the master process in the Google MapReduce implementation.

The architecture of the Scheduling Service is organized into two major components:

the MapReduceSchedulerService and the MapReduceScheduler.

# 8.3 Aneka MapReduce programming

* 1. **Aneka MapReduce programming**

#### Introducing the MapReduce programming model

##### Runtime support

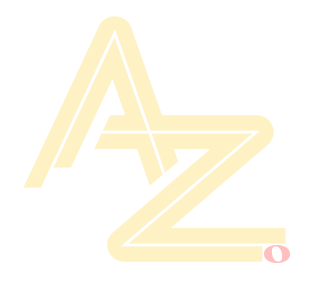
**Task Execution.**

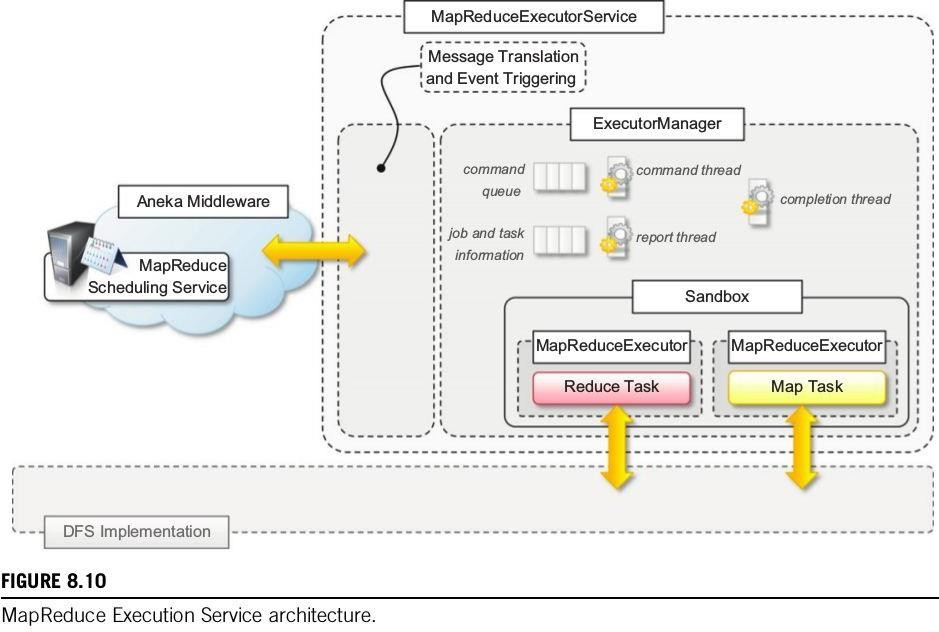
Controlled by MapReduce Execution Service.

This component plays role of the worker process in the Google MapReduce implementation.

There are three major components that coordinate together for executing tasks:

1. MapReduce- SchedulerService,
2. ExecutorManager, and
3. MapReduceExecutor.





#### 8.3.1 Introducing the MapReduce programming model

##### Distributed file system support

Aneka supports, the MapReduce model.

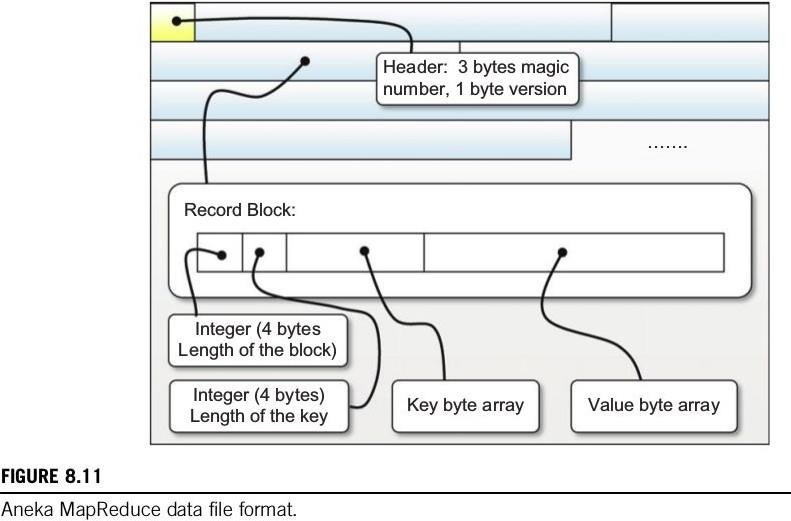
guarantee high availability and better efficiency by means of replication and distribution.

interfacing with different storage implementations and it maintains the same flexibility for the integration.

The level of integration requires to perform the following tasks:

* + - Retrieving the location of files and file chunks
    - Accessing a file by means of a stream

#### 8.3.1 Introducing the MapReduce programming model

**3 Distributed file system support**

#### Introducing the MapReduce programming model

##### 3 Distributed file system support

**Listing 8.8** shows the interface of the SeqReader and SeqWriter classes.

**Listing 8.9** shows a practical use of the SeqReader class by implementing the callback used in the word-counter example.

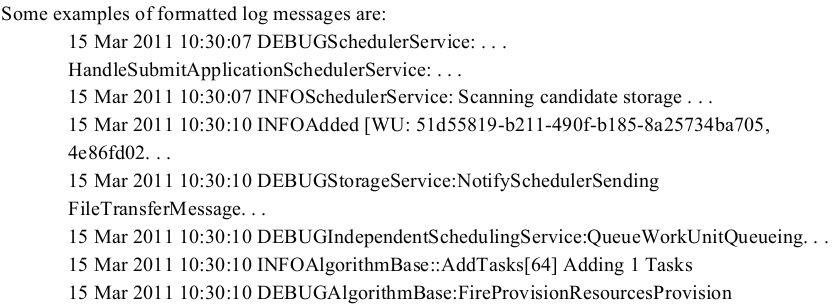
#### xample application

* + - 1. Parsing Aneka logs
      2. Mapper design and implementation 3 Reducer design and implementation 4 Driver program

5 Running the application

##### 1 Parsing Aneka logs

Aneka components (daemons, container instances, and services) produce a lot of information that is stored in the form of log files.

The entire framework leverages the log4net library for collecting and storing the log information.

##### Parsing Aneka logs

Possible information that we might want to extract from such logs is the following:

* + The distribution of log messages according to the level
  + The distribution of log messages according to the components The structure of the map and reduce functions will be the following:

map: (long; string) => (string; long) reduce: (long; string) => (string; long)

##### apper design and implementation

The operation performed by the map function is a very simple text extraction that identifies the level of the logging and the name of the component entering the information in the log.

Once this information is extracted, a key-value pair (string, long) is emitted by the function.

**Listing 8.10** shows the implementation of the Mapper class for the log parsing task.

##### Reducer design and implementation

Add all the values that are associated to the same key and emit a key-value pair with the total sum.

**Listing 8.11** , the operation to perform is very simple and actually is the same for both of the two different key-value pairs extracted from the log lines.

##### Driver program

**LogParsingMapper** and **LogParsingReducer** constitute the core functionality of the MapReduce job, which only requires to be properly configured in the main program in order to process and produce text tiles.

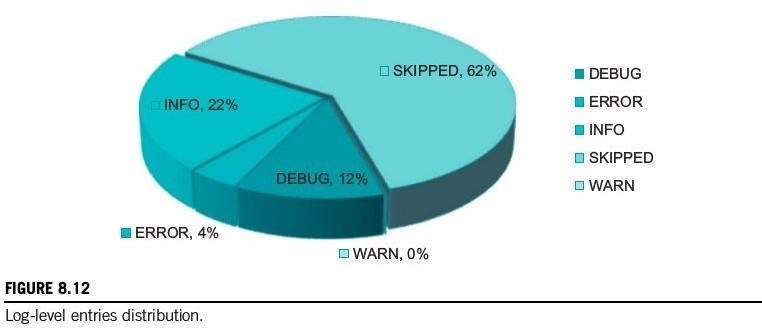
**Listing 8.12** shows the implementation of the driver program. With respect to the previous examples, there are three

things to be noted:

* + The configuration of the MapReduce job
  + The post-processing of the result files
  + The management of errors

##### Running the application

Aneka produces a considerable amount of logging information.

By continuing to run an Aneka Cloud for a few days, it is quite easy to collect enough data to mine for our sample application.

# 8.3 Aneka MapReduce programming

##### 5 Running the application