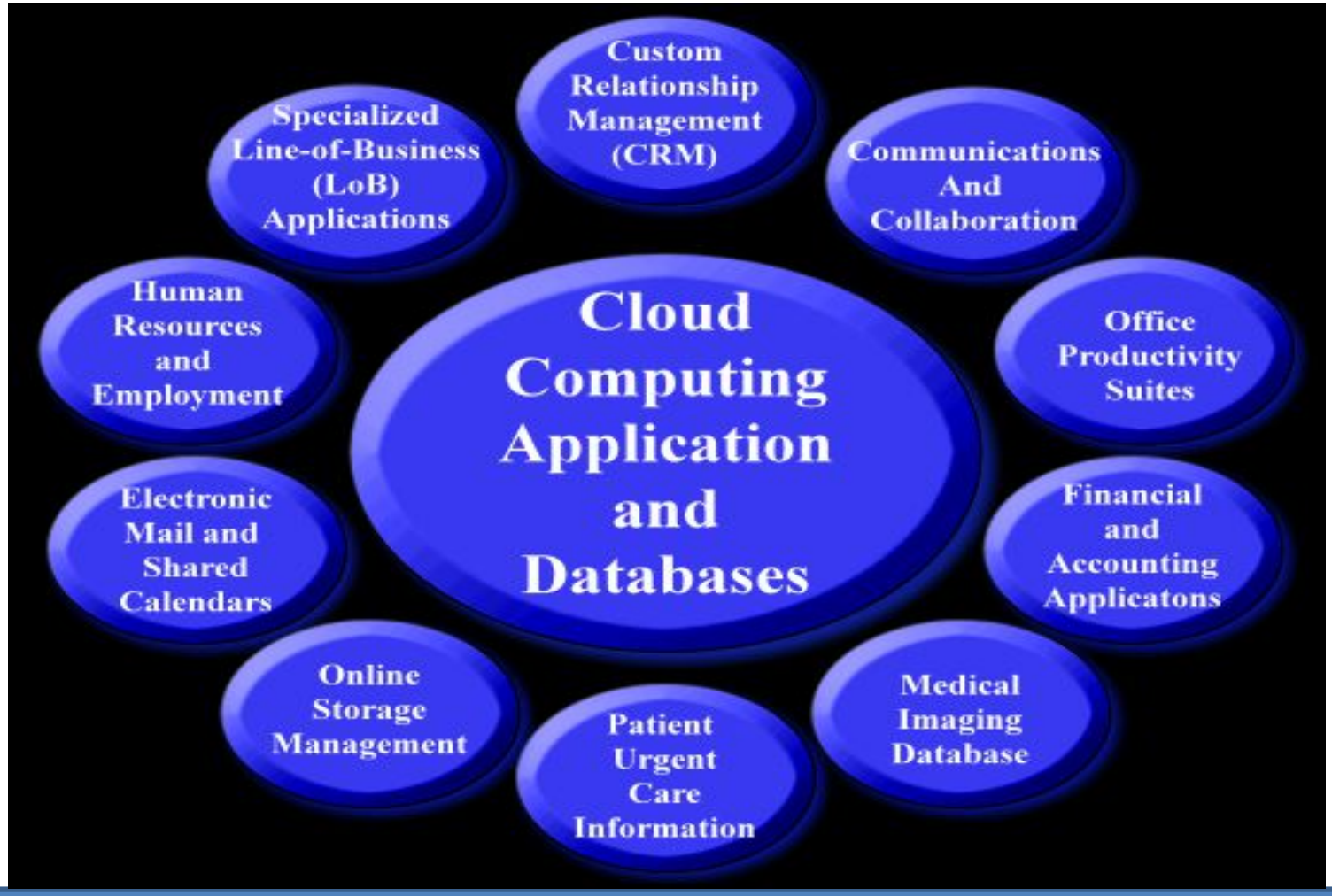
Cloud Computing: Applications and Paradigms



CHAPTER 03

Contents

- Challenges for cloud computing.
- Existing cloud applications and new opportunities.
- Architectural styles for cloud applications.
- Workflows - coordination of multiple activities.
- Coordination based on a state machine model.
- The MapReduce programming model.
- A case study: the GrepTheWeb application.

Challenges for cloud computing

- Performance isolation - nearly impossible to reach in a real system, especially when the system is heavily loaded. The performance of virtual machines fluctuates based on the load, the infrastructure services, and the environment, including the other users.
- Reliability - major concern; server failures expected when a large number of servers cooperate for the computations.

Challenges for cloud computing

- Security isolation is also challenging on multitenant systems.
- Cloud infrastructure exhibits internode latency and bandwidth fluctuations which affect the application performance.
- Performance considerations limit the amount of *data logging*; but frequent logging is required to identify the source of unexpected results and errors.

Challenges for cloud computing

- Techniques to preserve the logs for a postmortem analysis.
- Choosing an optimal instance (in terms of performance isolation, reliability, and security) from those offered by the cloud infrastructure.
- Efficiency, consistency, and communication scalability are major concerns for an application developer.

Challenges for cloud computing

- Data storage plays a critical role in the performance of any data-intensive application; the organization of the storage, the storage location, and the storage bandwidth must be carefully analyzed to lead to optimal application performance.
- software licensing.

Cloud computing is very attractive to the users

- Economic reasons.
  - low infrastructure investment.
  - low cost - customers are only billed for resources used.
- Convenience and performance.
  - Application developers enjoy the advantages of a just-in-time infrastructure; they are free to design an application without being concerned with the system where the application will run.
  - The execution time of compute-intensive and data-intensive applications can, potentially, be reduced through parallelization.

Cloud computing is very attractive to the users

- Convenience and performance.
  - Application developers enjoy the advantages of a just-in-time infrastructure; they are free to design an application without being concerned with the system where the application will run.
  - The execution time of compute-intensive and data-intensive applications can, potentially, be reduced through parallelization. If an application can partition the workload in **n segments** and **spawn n instances** of itself, the execution time could be reduced by a factor close to **n.**

Cloud computing is very attractive to the users

- Cloud elasticity allows an application to absorb the additional workload without any effort from application developers.
- Cloud computing is also beneficial for the providers of computing cycles - it typically leads to a higher level of resource utilization.
- The main attraction of cloud computing is the ability to use as many servers as necessary to optimally respond to the cost and the timing constraints of an application.

Note

- The appeal of cloud computing is its focus on enterprise applications.
- This clearly differentiates it from the grid computing effort, which was largely focused on scientific and engineering applications.
- The other major advantage of the cloud computing approach over grid computing is the concentration of resources in large data centers in a single administrative domain.

Note

- The *arbitrarily divisible load-sharing model* is common to many applications, and these are precisely the applications suitable for cloud computing.
- Web services, database services, and transaction-based services are ideal applications for cloud computing.
- Cloud applications benefits from an elastic environment in which resources are available when needed and users pay only for the resources they consume.

Ideal Applications for Cloud computing

- Web services
- database services
- Transaction-based services:
  - the cost/performance of such applications/services benefits from an elastic environment in which resources are available when needed and users pay only for the resources they consume.

Applications unlikely to perform well on a cloud

- Applications with a complex workflow and multiple dependencies, as is often the case in high-performance computing.
- Applications which require intensive communication among concurrent instances.
- Applications for which the workload cannot be arbitrarily partitioned

Existing cloud applications

- Three broad categories of existing applications:
    - Processing pipelines.
    - Batch processing systems.
    - Web applications.

Processing pipelines

- Indexing large datasets created by web crawler engines.
- Data mining - searching large collections of records to locate items of interests.
- Image processing .
  - Image conversion, e.g., enlarge an image or create thumbnails.
  - Compress or encrypt images.

Processing pipelines

- Video transcoding from one video format to another, e.g., from AVI to MPEG.
- Document processing.
  - Convert large collections of documents from one format to another, e.g., from Word to PDF.
  - Encrypt documents.
  - Use Optical Character Recognition to produce digital images of documents.

Batch processing applications

- Generation of daily, weekly, monthly, and annual activity reports for retail, manufacturing, other economical sectors.
- Processing, aggregation, and summaries of daily transactions for financial institutions, insurance companies, and healthcare organizations.
- Inventory management for large corporations.
- Processing billing and payroll records.
- Management of the software development, e.g., nightly updates of software repositories.
- Automatic testing and verification of software and hardware systems.

Web access

- Web Sites for online commerce.
- Web Sites with a periodic or temporary presence.
  - Conferences or other events.
  - Active during a particular season (e.g., the Holidays Season) or income tax reporting.
- Web Sites for promotional activities.
- Web Sites that ``sleep'' during the night and auto-scale during the day.

Potentially new applications

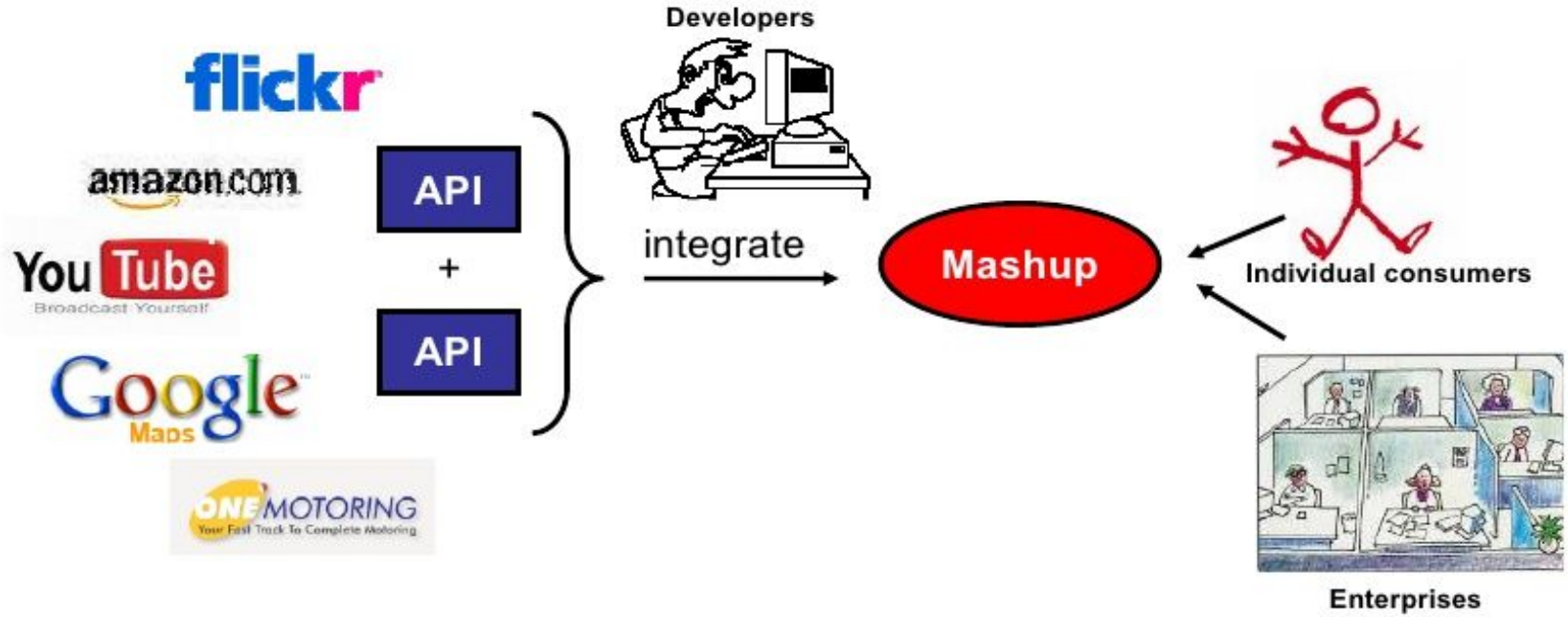
- Batch processing for decision support systems and business analytics.
- Mobile interactive applications which process large volumes of data from different types of sensors and services that combine more than one data source (e.g., mashups)
- Science and engineering could greatly benefit from cloud computing because many applications in these areas are compute-intensive and data-intensive. (Matlab and Mathematica - Maths S/W)

Mashups

- A *mashup* is an application that combines data, presentation, or functionality from two or more sources to create a service.
- A **mashup** in web development, is a web page, or web **application, that uses** content from more than one source to create a single new service displayed in a single graphical interface

The Mashup Ecosystem

- A "**mashup** is a web application that combines **data** from more than one source into a single integrated tool"



Architectural styles for cloud applications

- Based on the client-server paradigm: The vast majority of cloud applications take advantage of request/response communication between clients and stateless servers.
- Stateless servers - view a client request as an independent transaction and respond to it; they do not keep track of which clients are accessing them.
- The application clients and the servers running on the cloud communicate using Remote Procedure Calls (RPCs).

RPC

- Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details.

Architectural styles for cloud applications

- Simple Object Access Protocol (SOAP) - application protocol for web applications; It is an XML-based messaging protocol for exchanging information among heterogeneous systems. Uses TCP or UDP transport protocols.
- Representational State Transfer (REST) - software architecture for distributed hypermedia systems. Supports client communication with stateless servers, it is platform independent, language independent, supports data caching, and can be used in the presence of firewalls.

Architectural styles for cloud applications

- REST uses HTTP to support all four Create/Read/Update/Delete (CRUD) operations.
- REST is a much easier-to-use alternative to RPC, CORBA, or Web Services such as SOAP or WSDL.
- Web Services Description Language (WSDL) (see www.w3.org/TR/wsdl) was introduced in 2001 as an XML-based grammar to describe communication between endpoints of a networked application.

Workflows: Coordination of multiple activities

- Many cloud applications require the completion of multiple interdependent tasks.
- The description of a complex activity involving such an ensemble of tasks is known as a *workflow.*
- **workflow** consists of an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services, or process information.
- Task is the central concept in workflow modeling; a task is a unit of work to be performed on the cloud, and it is characterized by several attributes.

Attributes

- Name: A string of characters uniquely identifying the task.
- Description: A natural language description of the task.
- Actions: Modifications of the environment caused by the execution of the task.
- Preconditions: Boolean expressions that must be true before the action(s) of the task can take place.
- Post-conditions: Boolean expressions that must be true after the action(s) of the task take place.

Attributes

- Exceptions: Provide information on how to handle abnormal events.
- Case : an instance of a process description.
- State of a case at time t - defined in terms of tasks already completed at that time.
- Events - cause transitions between states.

Types of Tasks

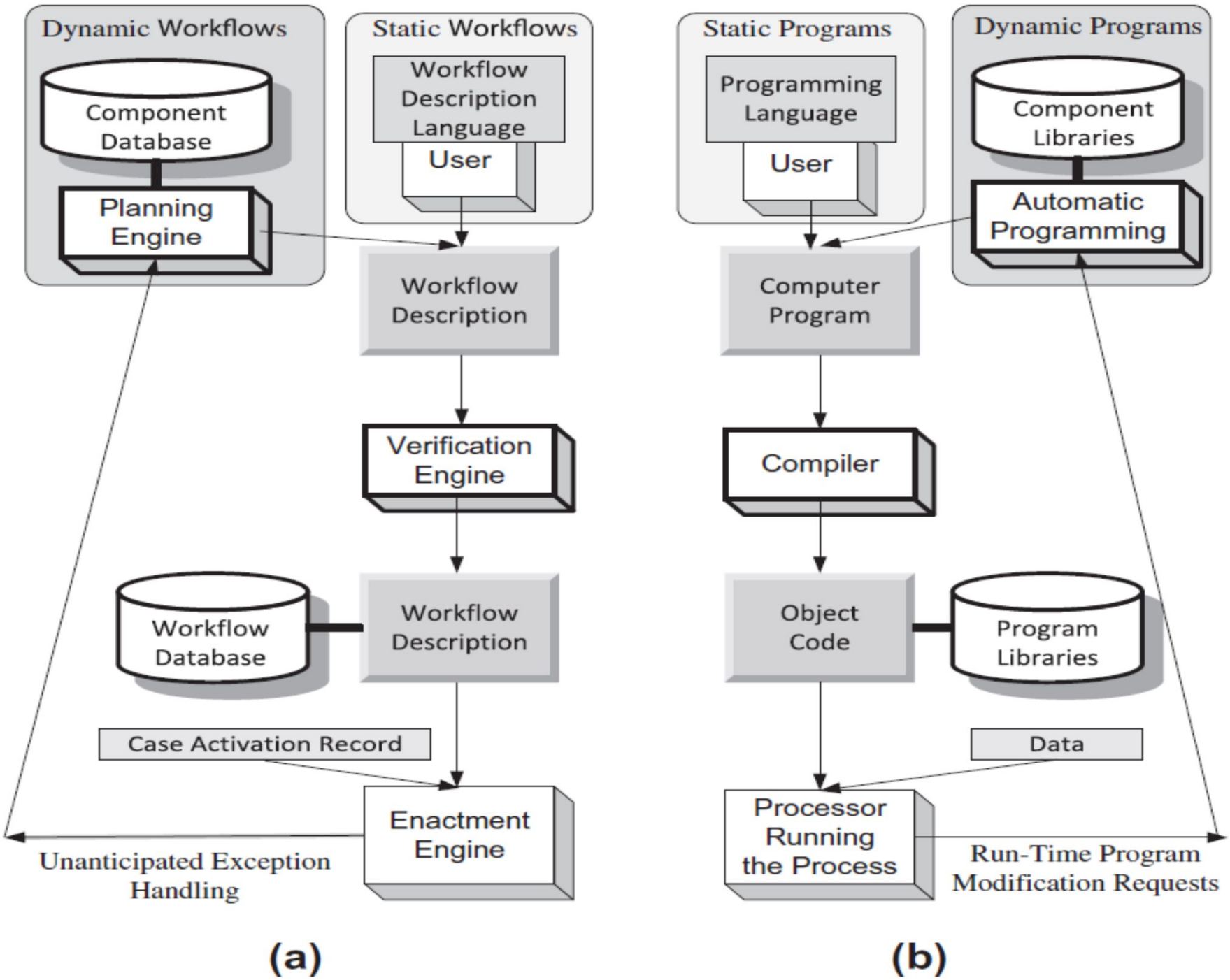
- A *composite task* is a structure describing a subset of tasks and the order of their execution.
- A *primitive task* is one that cannot be decomposed into simpler tasks.
- A *routing task* is a special-purpose task connecting two tasks in a workflow description.
- *predecessor* task is one that has just completed its execution.
- *Successor task* is the one that to be initiated next.
- A *fork routing task* triggers execution of several successor tasks

Types of Tasks

- A *join routing task* waits for completion of its predecessor tasks.
  - The successor is enabled after all predecessors end.
  - The successor is enabled after k out of $n > k$ predecessors end.
  - Iterative: The tasks between the fork and the join are executed repeatedly.

Process description

- A process description, also called a workflow schema, is a structure describing the tasks or activities to be executed and the order of their execution.
- A process description contains one start symbol and one end symbol. A process description can be provided in a workflow definition language (WFDL), supporting constructs for choice, concurrent execution, the fork, join constructs, and iterative execution.
- Workflow description resembles a flowchart, a concept from programming.


Dynamic Workflows
Component Database
Planning Engine
Static Workflows
Workflow Description Language
User
Workflow Description
Verification Engine
Workflow Database
Workflow Description
Case Activation Record
Enactment Engine
Unanticipated Exception Handling
(a)
Static Programs
Programming Language
User
Dynamic Programs
Component Libraries
Automatic Programming
Computer Program
Compiler
Object Code
Program Libraries
Data
Processor Running the Process
Run-Time Program Modification Requests
(b)

A parallel between workflows and programs

- (a) The life cycle of a workflow. (b) The life cycle of a computer program.
- The workflow definition/specification is analogous to writing a program. Planning is analogous to automatic program generation.
- Verification corresponds to syntactic verification of a program.
- Workflow enactment mirrors the execution of a program.
- A static workflow corresponds to a static program and a dynamic workflow to a dynamic program.

A parallel between workflows and programs

- The life cycle of a workflow - creation, definition, verification, and enactment; similar to the life cycle of a traditional program (creation, compilation, and execution).
- *The enactment engine* executes workflow.

Safety and liveness

- Desirable properties of workflows.
- Safety → nothing “bad” ever happens.
- Liveness → something “good” will eventually happen.

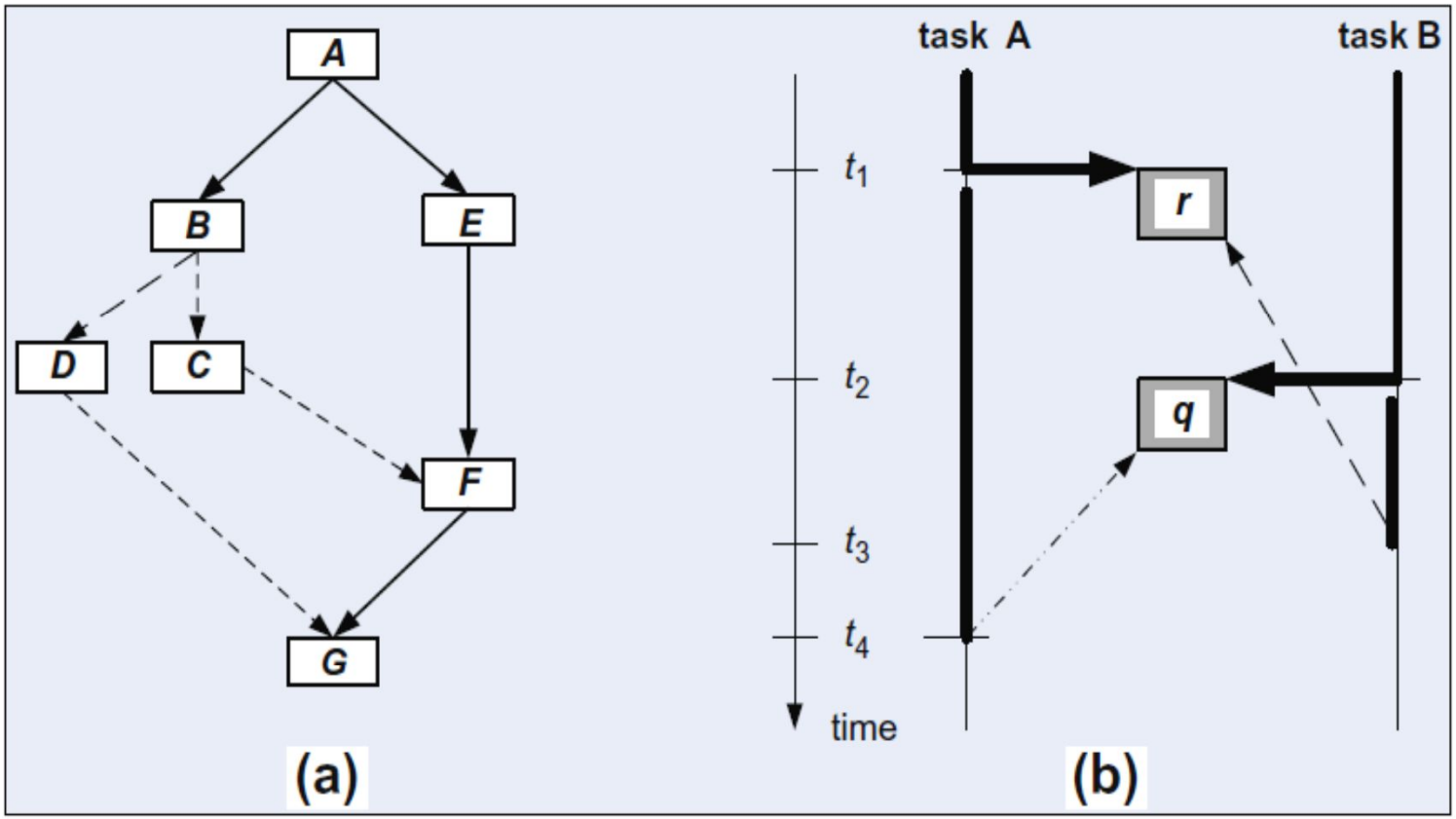


FIGURE 4.2

(a) A process description that violates the liveness requirement. If task *C* is chosen after completion of *B*, the process will terminate after executing task *G*; if *D* is chosen, then *F* will never be instantiated, because it requires the completion of both *C* and *E*. The process will never terminate, because *G* requires completion of both *D* and *F*. (b) Tasks *A* and *B* need exclusive access to two resources *r* and *q*, and a deadlock may take place if the following sequence of events occurs. At time t_1 task *A* acquires *r*, at time t_2 task *B* acquires *q* and continues to run; then at time t_3 task *B* attempts to acquire *r* and it blocks because *r* is under the control of *A*. Task *A* continues to run and at time t_4 attempts to acquire *q* and it blocks because *q* is under the control of *B*.

Basic workflow patterns

- Workflow pattern refers to the temporal relationship among the tasks of a process.

Basic workflow patterns

- Sequence - several tasks have to be scheduled one after the completion of the other.
- AND split - both tasks B and C are activated when task A terminates.
- Synchronization - task C can only start after tasks A and B terminate.
- XOR split - after completion of task A, either B or C can be activated.
- XOR merge - task C is enabled when either A or B terminate.

Basic workflow patterns

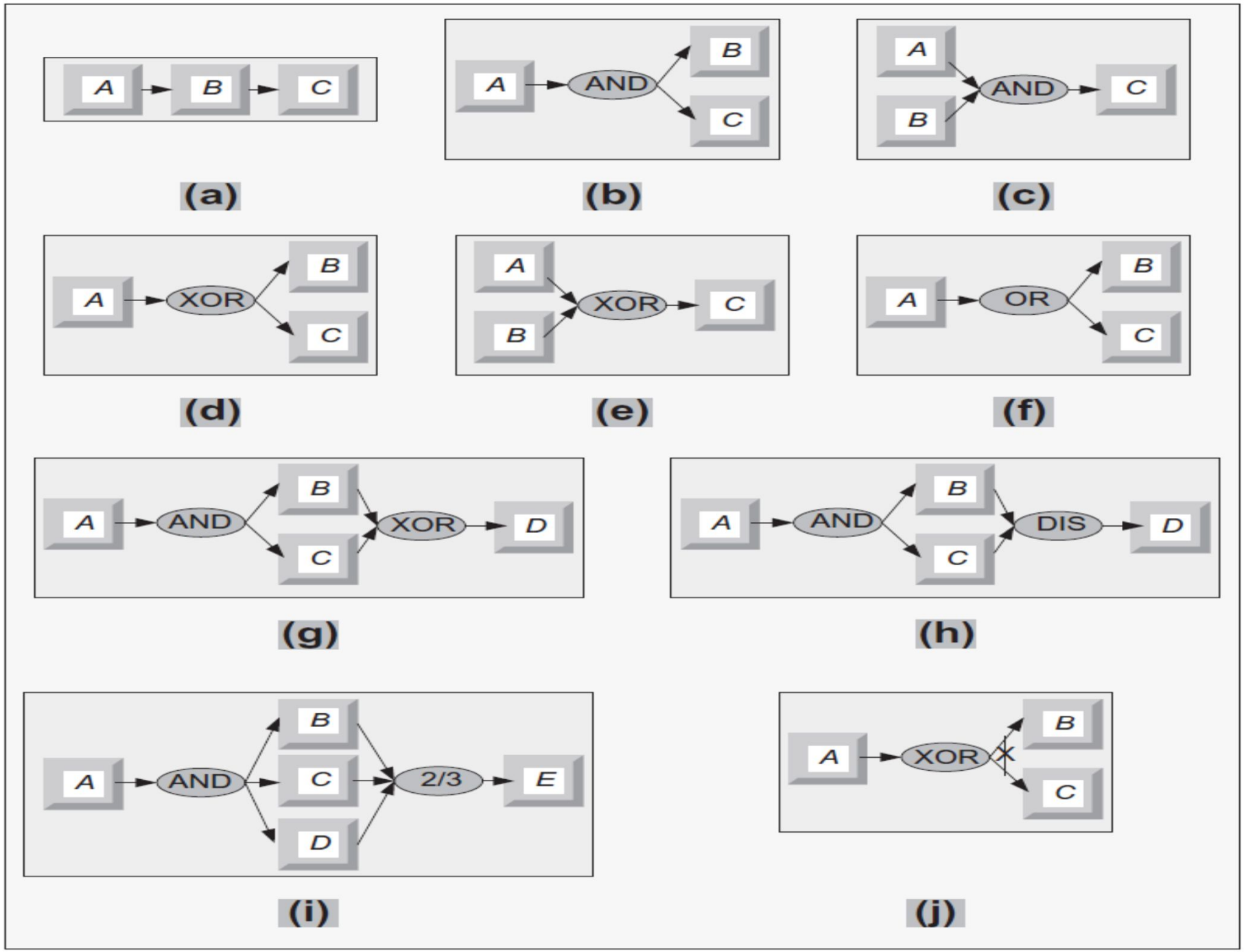
- OR split - after completion of task A one could activate either B, C, or both.
- Multiple Merge - once task A terminates, B and C execute concurrently; when the first of them, say B, terminates, then D is activated; then, when C terminates, D is activated again.
- Discriminator – wait for a number of incoming branches to complete before activating the subsequent activity.

Basic workflow patterns

- N out of M join - barrier synchronization. Assuming that M tasks run concurrently, N (N<M) of them have to reach the barrier before the next task is enabled. In our example, any two out of the three tasks A, B, and C have to finish before E is enabled.
- Deferred Choice - similar to the XOR split but the choice is not made explicitly; the run-time environment decides what branch to take.

Basic workflow patterns

a) *Sequence*.
b) *AND split*.
c) *Synchronization*.
d) *XOR split*.
e) *XOR merge*.
f) *OR split*.
g) *Multiple merge*.
h) *Discriminator*.
i) *N out of M join*.
j) *Deferred choice*.


A
B
C
(a)
A
AND
B
C
(b)
A
B
AND
C
(c)
A
XOR
B
C
(d)
A
B
XOR
C
(e)
A
OR
B
C
(f)
A
AND
B
C
XOR
D
(g)
A
AND
B
C
DIS
D
(h)
A
AND
B
C
D
2/3
E
(i)
A
XOR
B
C
(j)

The MapReduce programming model

- MapReduce is based on a idea of parallel processing for data-intensive applications supporting arbitrarily divisible load sharing.
- In first phase, split the data into blocks, assign each block to an instance or process, and run these instances in parallel to complete the computations assigned to them.
- In second phase: Merge the partial results produced by individual instances.
- It is based on same program, multiple data (SPMD).
- The master instance partitions the data and gathers the partial results.

The MapReduce programming model

- *MapReduce* is a programming model is conceived for processing and generating large data sets on computing clusters.
- *In MapReduce model, a set of input <key, value> pairs is transformed into a set of output <key, value> pairs.*

The MapReduce programming model

(1) An application starts a *master instance* and *M* worker instances for the *Map* phase and, later, *R* worker instances for the *Reduce* phase.

(2) The *master* partitions the input data in *M* segments.

(3) Each *Map instance* reads its input data segment and processes the data.

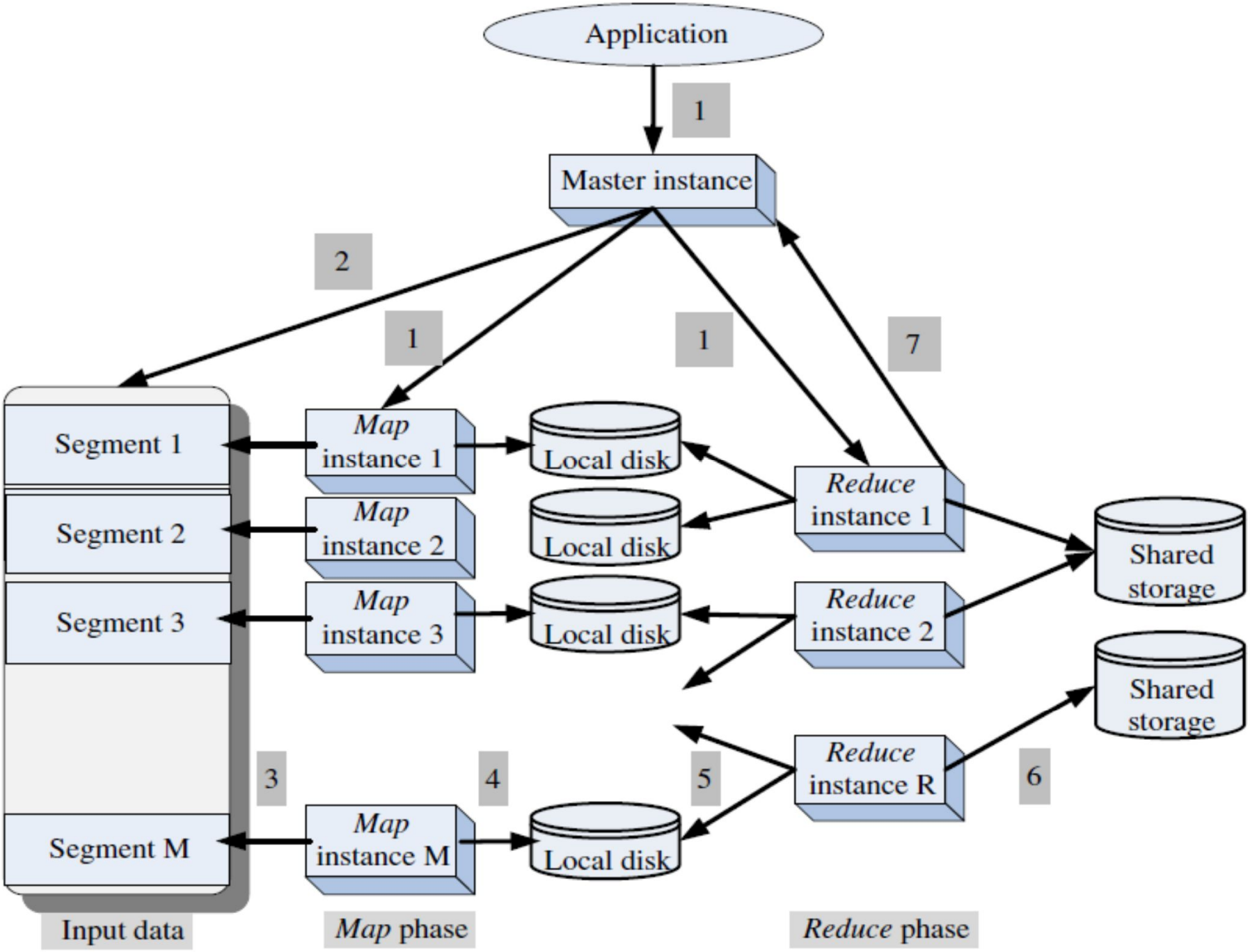
(4) The results of the processing are stored on the local disks of the servers where the *Map instances* run.

The MapReduce programming model

(5) When all *Map instances* have finished processing their data, the *R Reduce instances* read the results of the first phase and merge the partial results.

(6) The final results are written by the *Reduce instances* to a shared storage server.

(7) The *master instance* monitors the *Reduce instances* and, when all of them report task completion, the application is terminated.


Application
1
Master instance
2
1
1
7
Segment 1
Segment 2
Segment 3
Segment M
Map instance 1
Map instance 2
Map instance 3
Map instance M
Local disk
Local disk
Local disk
Local disk
Reduce instance 1
Reduce instance 2
Reduce instance R
Shared storage
Shared storage
3
4
5
6
Input data
Map phase
Reduce phase

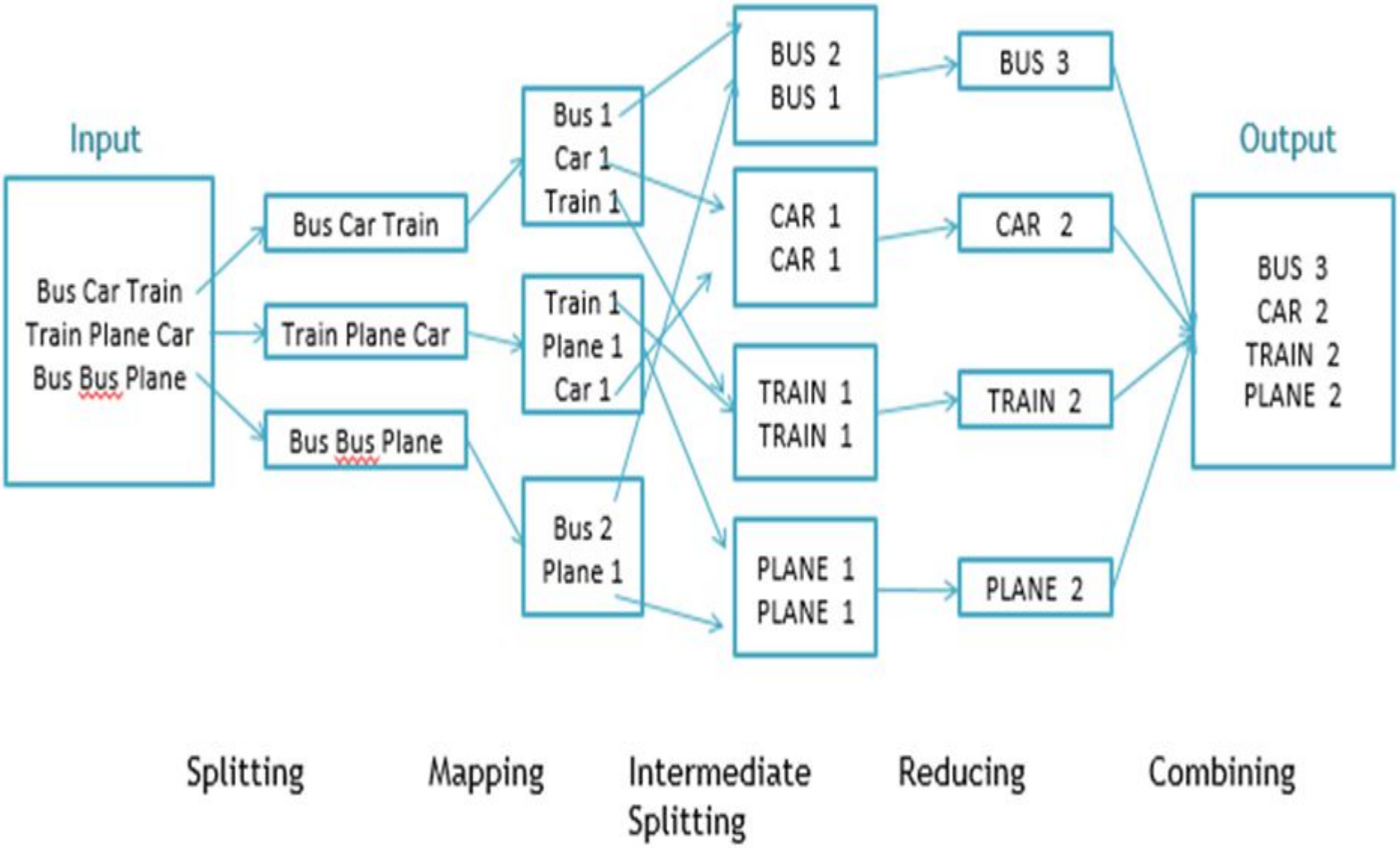
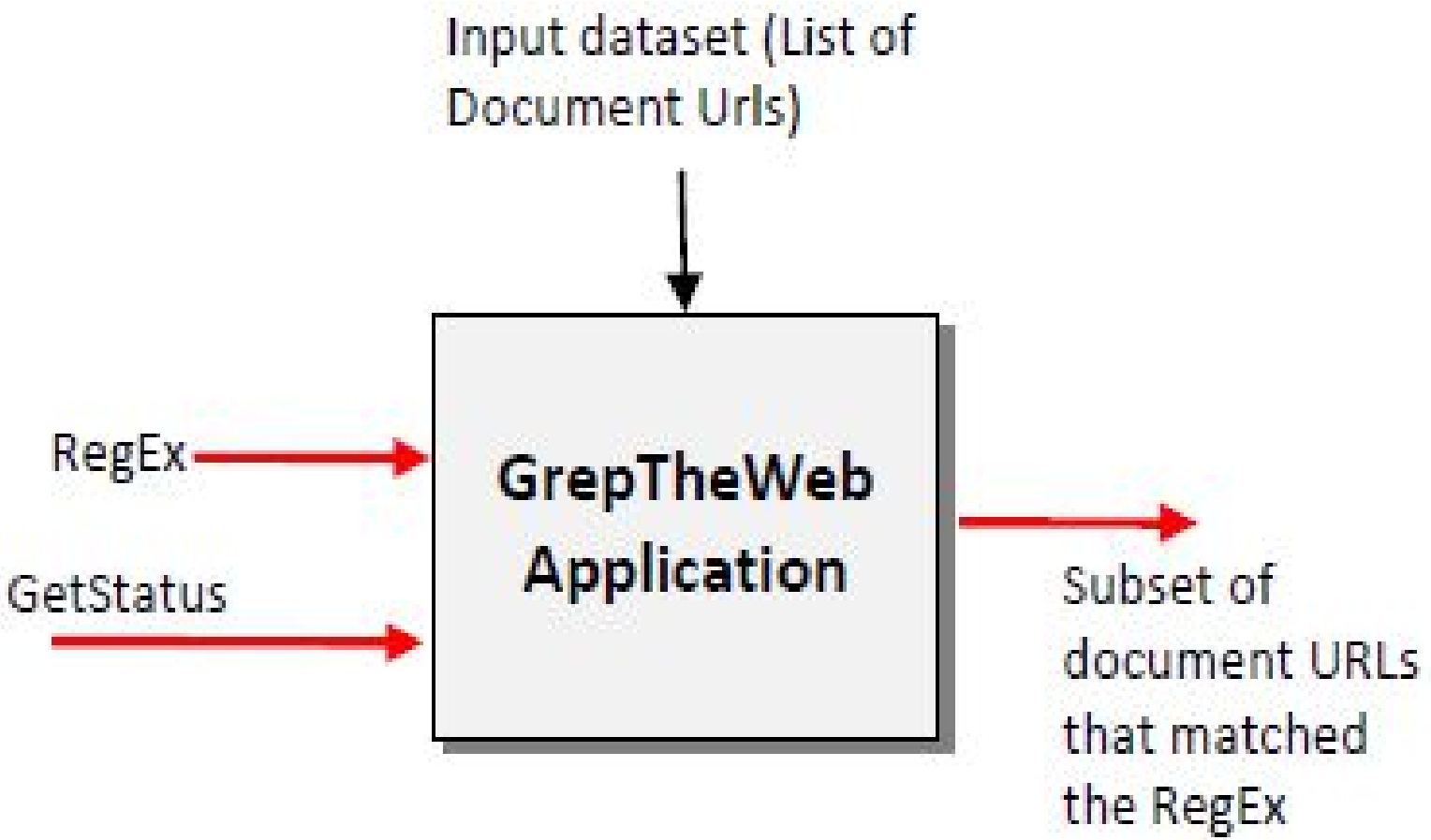


Fig. WorkFlow of MapReducing

Case study: GrepTheWeb (Now in production at Amazon)

- Used illustrate the power and appeal of cloud computing .
- The application allows a user to define a regular expression (Search pattern) and search the Web for records that match it.
- The GrepTheWeb can “grep” (a popular Unix command-line utility to search patterns) the actual Web documents.

Case study: GrepTheWeb



Case study: GrepTheWeb

- The source of this search is a collection of document URLs produced by the *Alexa Web Search*, a software system that crawls the Web every night.
- The inputs to the applications are a regular expression and the large data set produced by the Web-crawling software (web bots); the output is the set of records that satisfy the expression.
- The user is able to interact with the application and get the current status.

Case study: GrepTheWeb

- Runs on a massively distributed system and allows it to run in parallel and scale up and down, based on the number of users and the problem size.
- Uses message passing to trigger the activities of multiple controller threads which launch the application, initiate processing, shutdown the system, and create billing records.

Case study: GrepTheWeb

- *GrepTheWeb* uses *Hadoop MapReduce (Apache)*, an open-source software package that splits a large data set into chunks, distributes them across multiple systems, launches the processing, and, when the processing is complete, aggregates the outputs from different systems into a final result.

Case study: GrepTheWeb

- The application uses the *Hadoop MapReduce* software and four Amazon services: *EC2*, *Simple DB*, *S3*, and *SQS*.
- The simplified workflow (a) shows the two inputs, the regular expression and the input records generated by the Web crawler. A third type of input is the user commands to report the current status and to terminate the processing.

simplified workflow

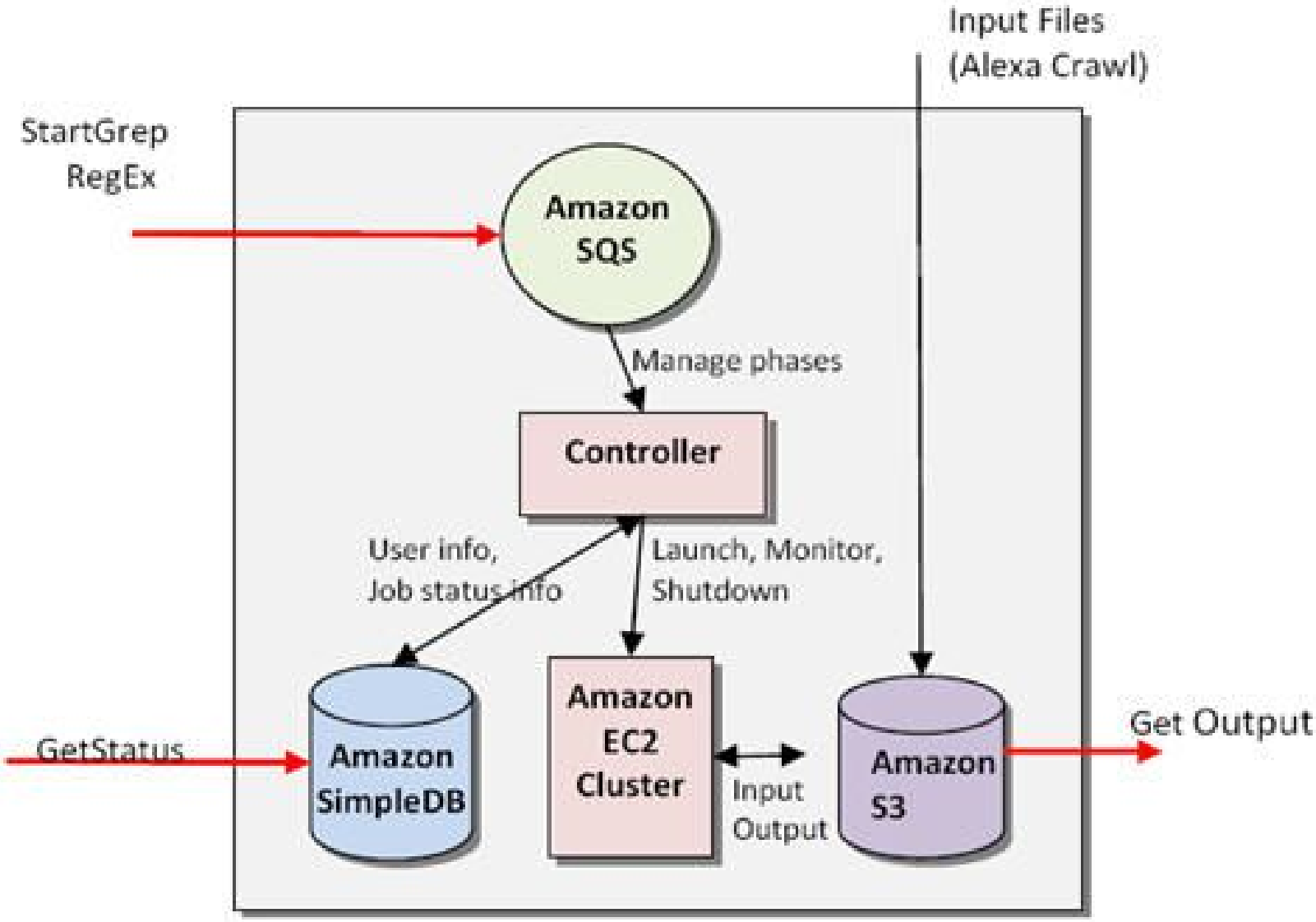
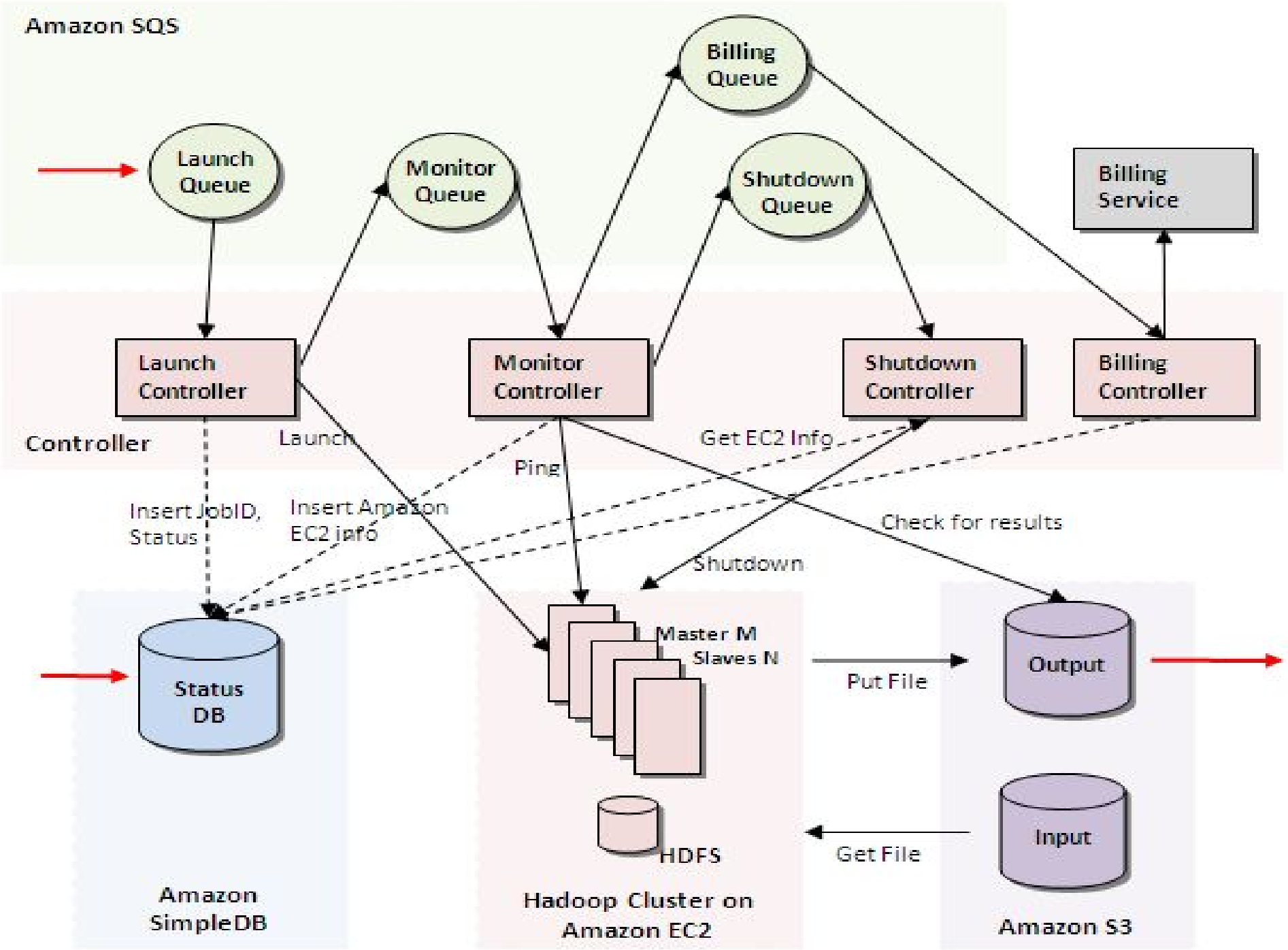
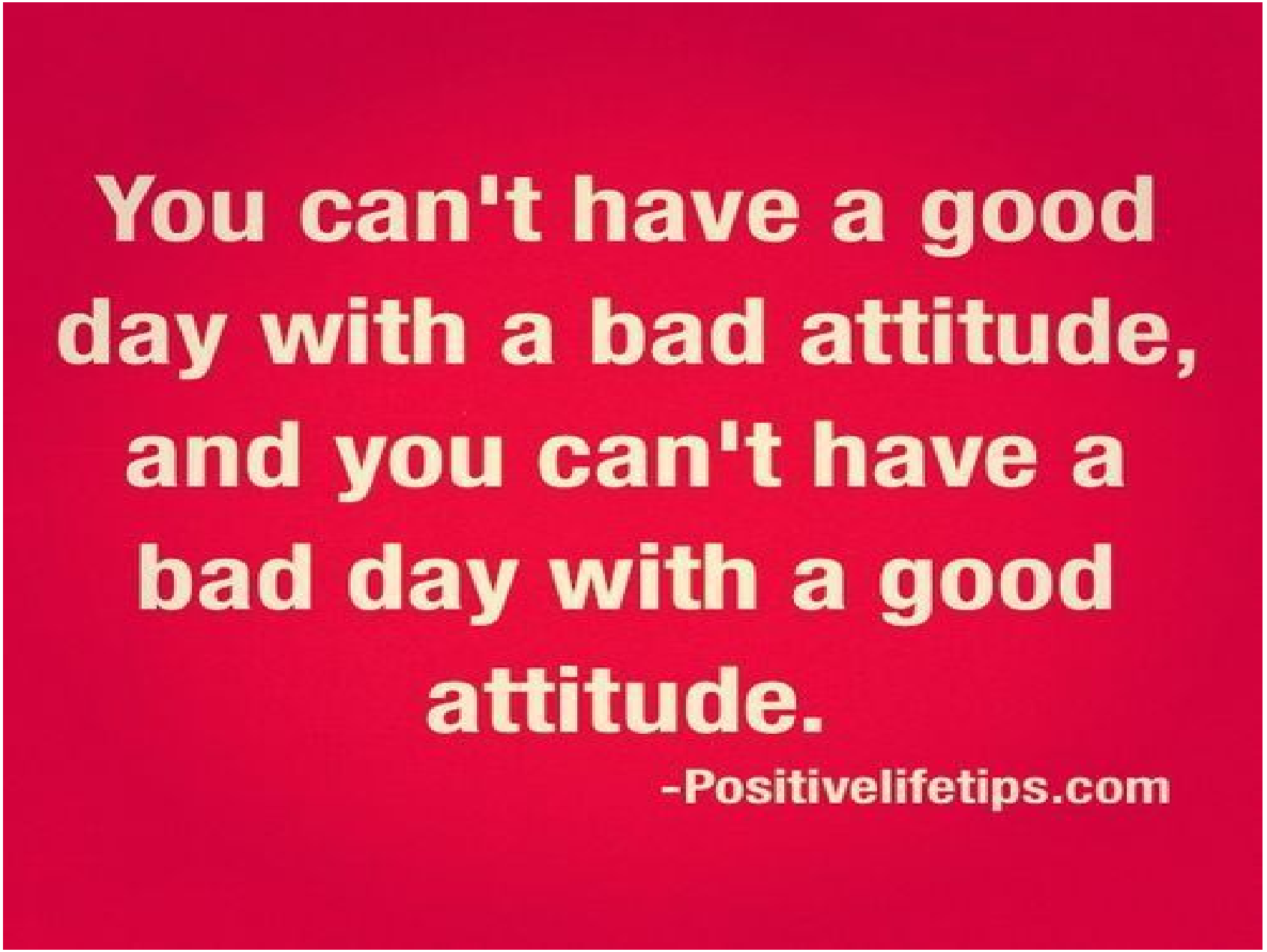


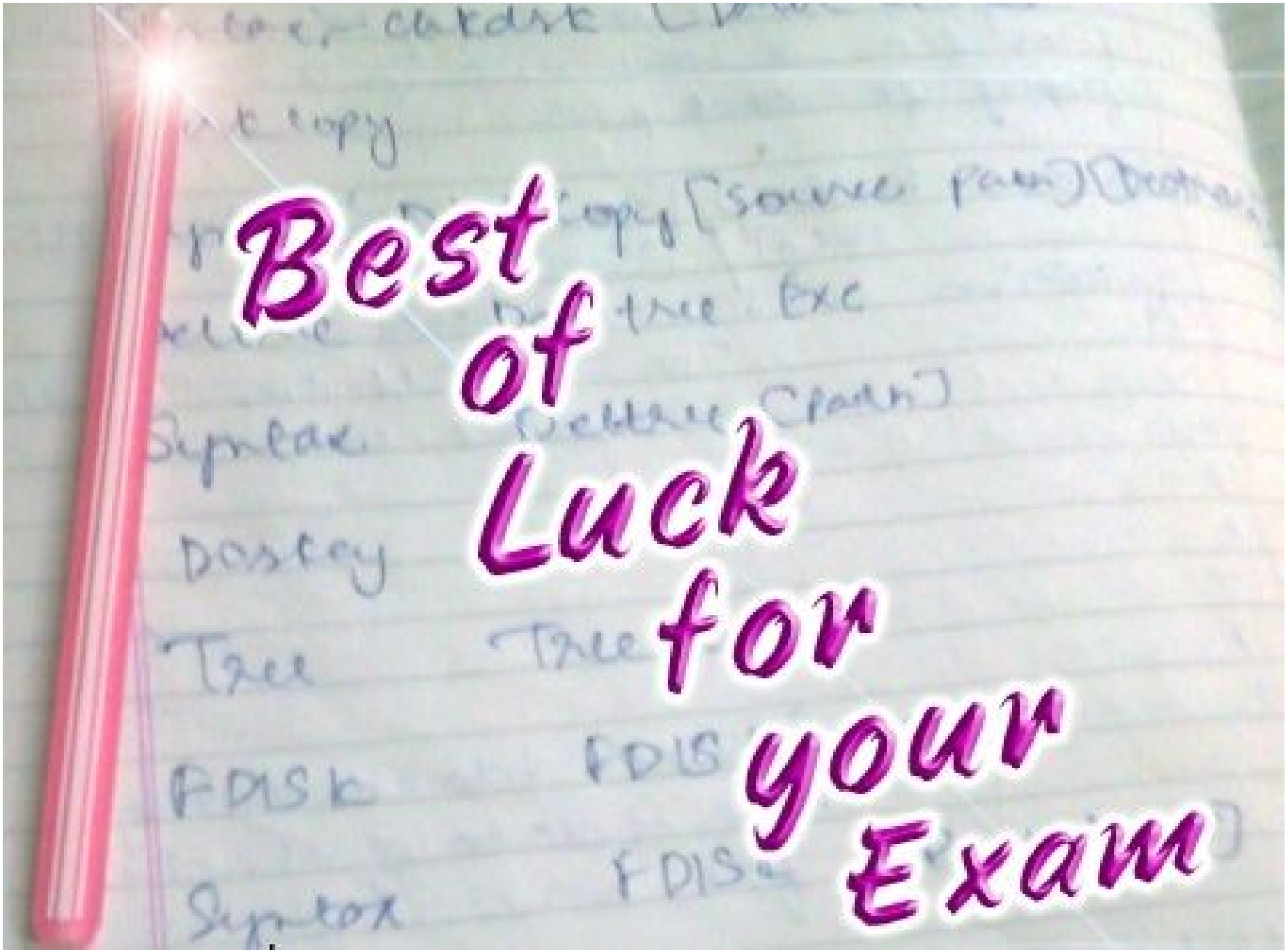
Figure 2: GrepTheWeb Architecture - Zoom Level 2

Case study: GrepTheWeb

- (b) The detailed workflow; the system is based on message passing between several queues; four controller threads periodically poll their associated input queues, retrieve messages, and carry out the required actions.


Amazon SQS
Billing Queue
Launch Queue
Monitor Queue
Shutdown Queue
Billing Service
Launch Controller
Monitor Controller
Shutdown Controller
Billing Controller
Controller
Launch
Get EC2 Info
Ping
Insert JobID, Status
Insert Amazon EC2 info
Check for results
Shutdown
Master M
Slaves N
Status DB
Output
Put File
HDFS
Input
Get File
Amazon SimpleDB
Hadoop Cluster on Amazon EC2
Amazon S3


You can't have a good day with a bad attitude, and you can't have a bad day with a good attitude.
-Positivelifetips.com


Best
of
Luck
for
your
Exam