Chapter 4 – Cloud Computing Applications and Paradigms

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.
- Clouds for science and engineering.
- High performance computing on a cloud.
- Legacy applications on a cloud.
- Social computing, digital content, and cloud computing.

Cloud applications

- 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. If an application can partition the workload in *n* segments and spawn *n* instances of itself, then the execution time could be reduced by a factor close to *n*.
- Cloud computing is also beneficial for the providers of computing cycles - it typically leads to a higher level of resource utilization.

Cloud applications (cont'd)

- Ideal applications for cloud computing:
  - Web services.
  - Database services.
  - Transaction-based service. The resource requirements of transaction-oriented services benefit from an elastic environment where resources are available when needed and where one pays only for the resources it consumes.
- 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.
  - When the workload cannot be arbitrarily partitioned.

Challenges for cloud application development

- Performance isolation - nearly impossible to reach in a real system, especially when the system is heavily loaded.
- Reliability - major concern; server failures expected when a large number of servers cooperate for the computations.
- Cloud infrastructure exhibits latency and bandwidth fluctuations which affect the application performance.
- Performance considerations limit the amount of *data logging*; the ability to identify the source of unexpected results and errors is helped by frequent logging.

Existing and new application opportunities

- Three broad categories of existing applications:
  - Processing pipelines.
  - Batch processing systems.
  - Web applications.
- 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.
  - Science and engineering could greatly benefit from cloud computing as many applications in these areas are compute-intensive and data-intensive.

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.
- 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.
- 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

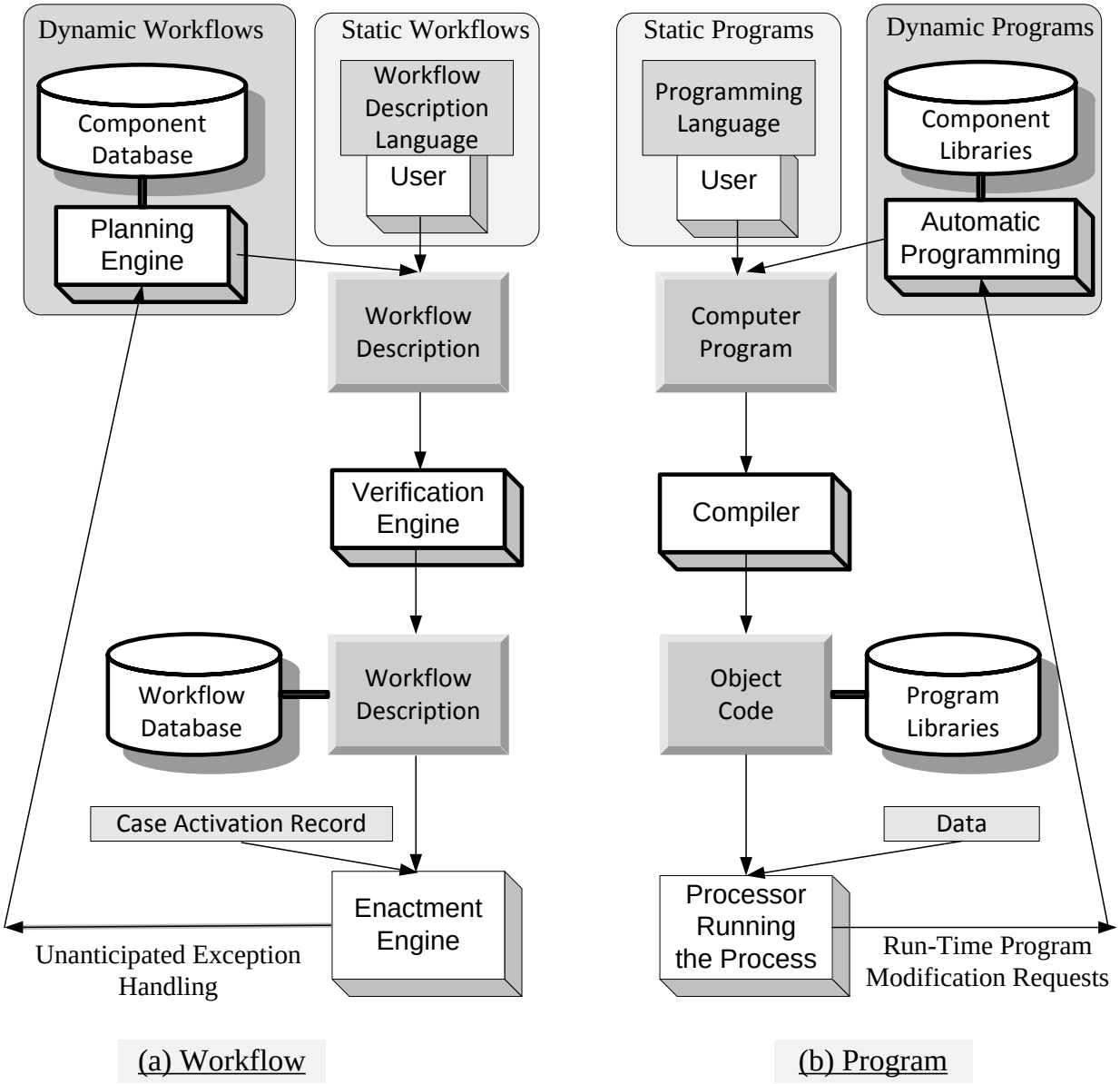
- Sites for online commerce.
- Sites with a periodic or temporary presence.
  - Conferences or other events.
  - Active during a particular season (e.g., the Holidays Season) or income tax reporting.
- Sites for promotional activities.
- Sites that ``sleep'' during the night and auto-scale during the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

Architectural styles for cloud applications

- Based on the client-server paradigm.
- Stateless servers - view a client request as an independent transaction and respond to it; the client is not required to first establish a connection to the server.
- Often clients and servers communicate using Remote Procedure Calls (RPCs).
- Simple Object Access Protocol (SOAP) - application protocol for web applications; message format based on the XML. 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.

Workflows

- *Process description* - structure describing the tasks to be executed and the order of their execution. Resembles a flowchart.
- *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.
- The *life cycle of a workflow* - creation, definition, verification, and enactment; similar to the life cycle of a traditional program (creation, compilation, and execution).


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
Static Programs
Programming Language
User
Computer Program
Compiler
Object Code
Program Libraries
Data
Processor Running the Process
Run-Time Program Modification Requests
Dynamic Programs
Component Libraries
Automatic Programming
(a) Workflow
(b) Program

Safety and liveness

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

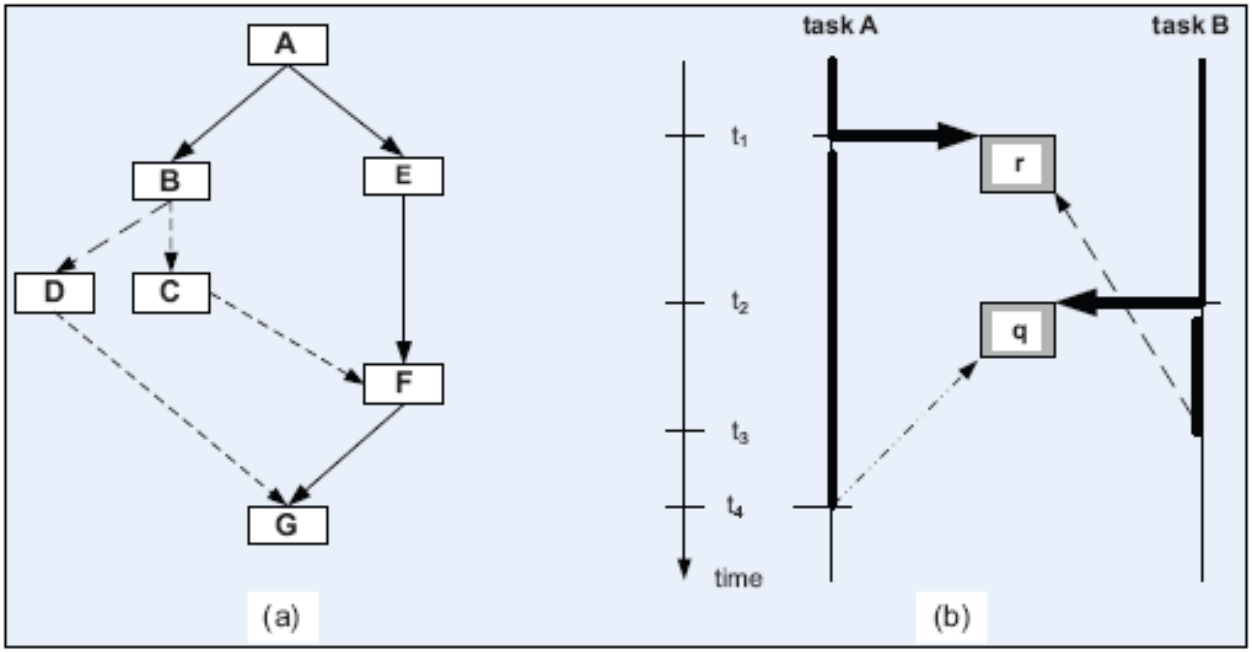


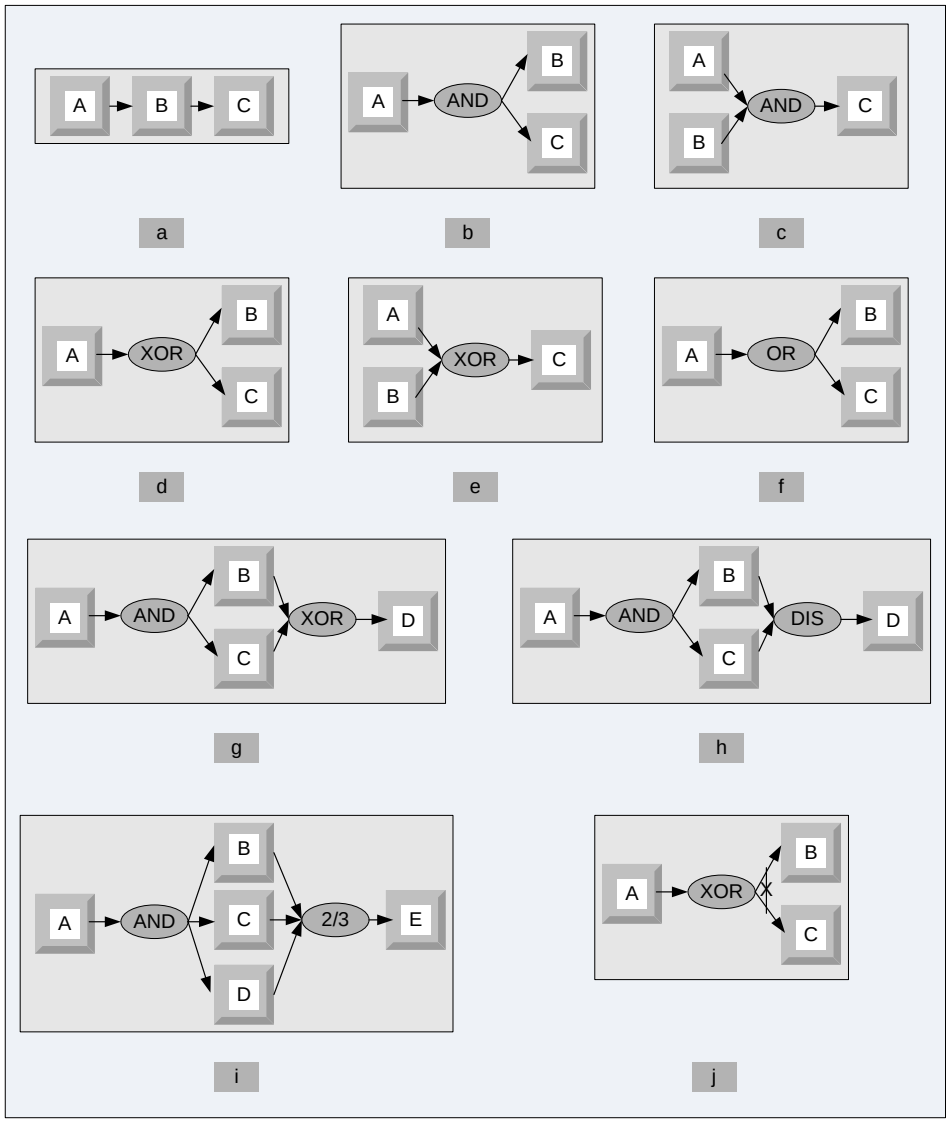
Figure 35: (a) A process description which violates the liveness requirement; if task C is chosen after completion of B, the process will terminate after executing task G; if D is chosen, then F will never be instantiated because it requires the completion of both C and E. The process will never terminate 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 occur if the following sequence of events occur: 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 patterns - the temporal relationship among the tasks of a process
  - 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.
  - 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; then wait for the remaining branches to finish without taking any action until all of them have terminated. Next, resets itself.

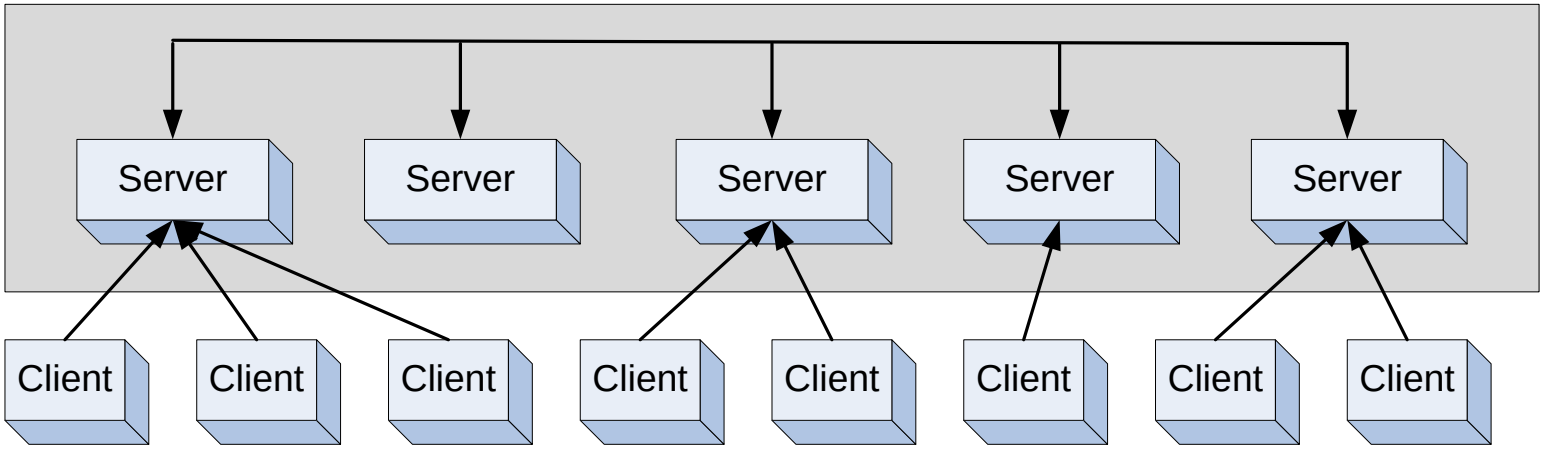
Basic workflow patterns (cont'd)

- 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.

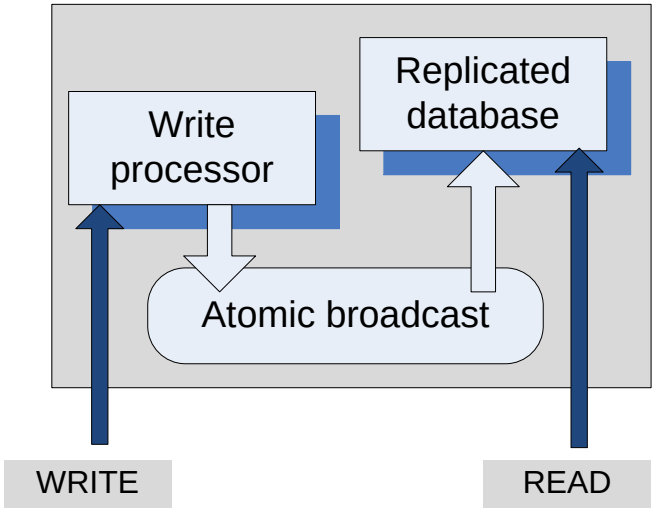

A
B
C
a
A
AND
B
C
b
A
B
AND
C
c
A
XOR
B
C
d
A
B
XOR
C
e
A
OR
B
C
f
A
AND
B
C
XOR
D
g
A
AND
B
C
DIS
D
h
A
AND
B
C
2/3
E
D
i
A
XOR
B
C
j

Coordination - ZooKeeper

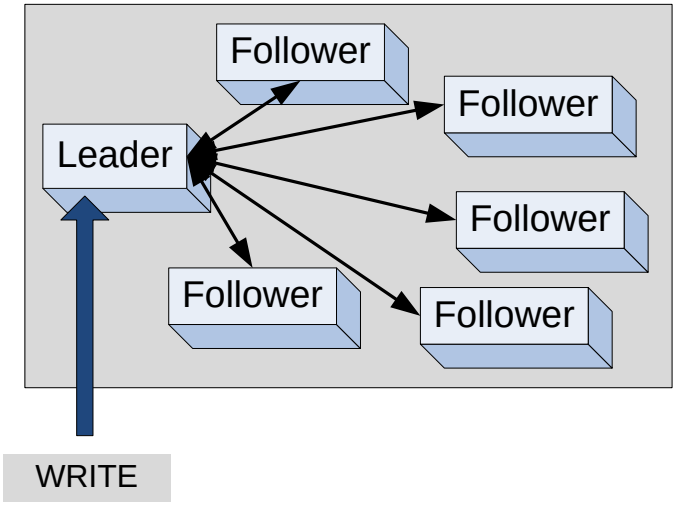
- Cloud elasticity → distribute computations and data across multiple systems; coordination among these systems is a critical function in a distributed environment.
- ZooKeeper
  - Distributed coordination service for large-scale distributed systems.
  - High throughput and low latency service.
  - Implements a version of the Paxos consensus algorithm.
  - Open-source software written in Java with bindings for Java and C.
  - The servers in the pack communicate and elect a leader.
  - A database is replicated on each server; consistency of the replicas is maintained.
  - A client connect to a single server, synchronizes its clock with the server, and sends requests, receives responses and watch events through a TCP connection.


Server
Server
Server
Server
Server
Client
Client
Client
Client
Client
Client
Client
Client

(a)


Write processor
Replicated database
Atomic broadcast
WRITE
READ

(b)


Follower
Follower
Leader
Follower
Follower
Follower
WRITE

(c)

Zookeeper communication

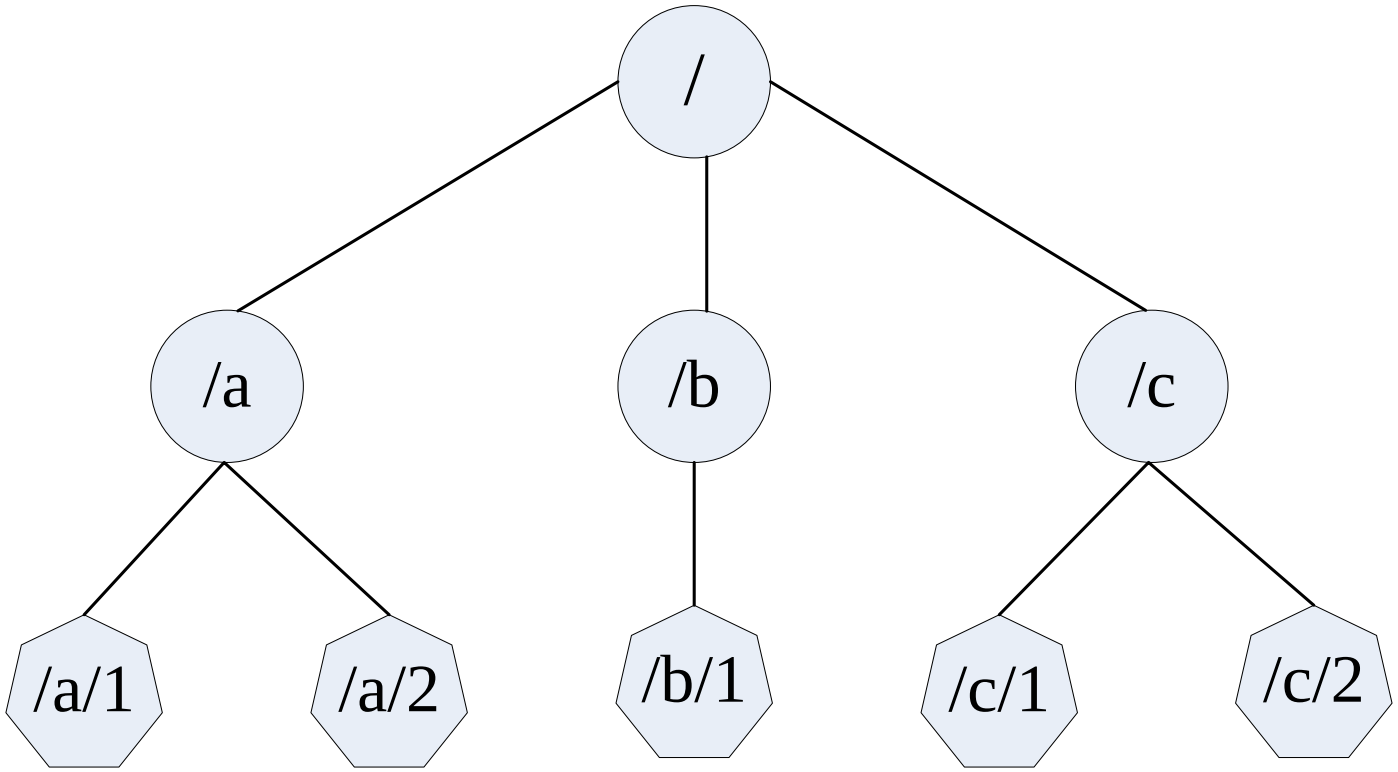
- Messaging layer → responsible for the election of a new leader when the current leader fails.

- Messaging protocols use:
  - Packets - sequence of bytes sent through a FIFO channel.
  - Proposals - units of agreement.
  - Messages - sequence of bytes atomically broadcast to all servers.
- A message is included into a proposal and it is agreed upon before it is delivered.
- Proposals are agreed upon by exchanging packets with a quorum of servers, as required by the Paxos algorithm.

Zookeeper communication (cont'd)

- Messaging layer guarantees:
  - Reliable delivery: if a message **m** is delivered to one server, it will be eventually delivered to all servers.
  - Total order: if message **m** is delivered before message **n** to one server, it will be delivered before **n** to all servers.
  - Causal order: if message **n** is sent after **m** has been delivered by the sender of **n**, then **m** must be ordered before **n**.

Shared hierarchical namespace similar to a file system; znodes instead of inodes



ZooKeeper service guarantees

- Atomicity - a transaction either completes or fails.
- Sequential consistency of updates - updates are applied strictly in the order they are received.
- Single system image for the clients - a client receives the same response regardless of the server it connects to.
- Persistence of updates - once applied, an update persists until it is overwritten by a client.
- Reliability - the system is guaranteed to function correctly as long as the majority of servers function correctly.

Zookeeper API

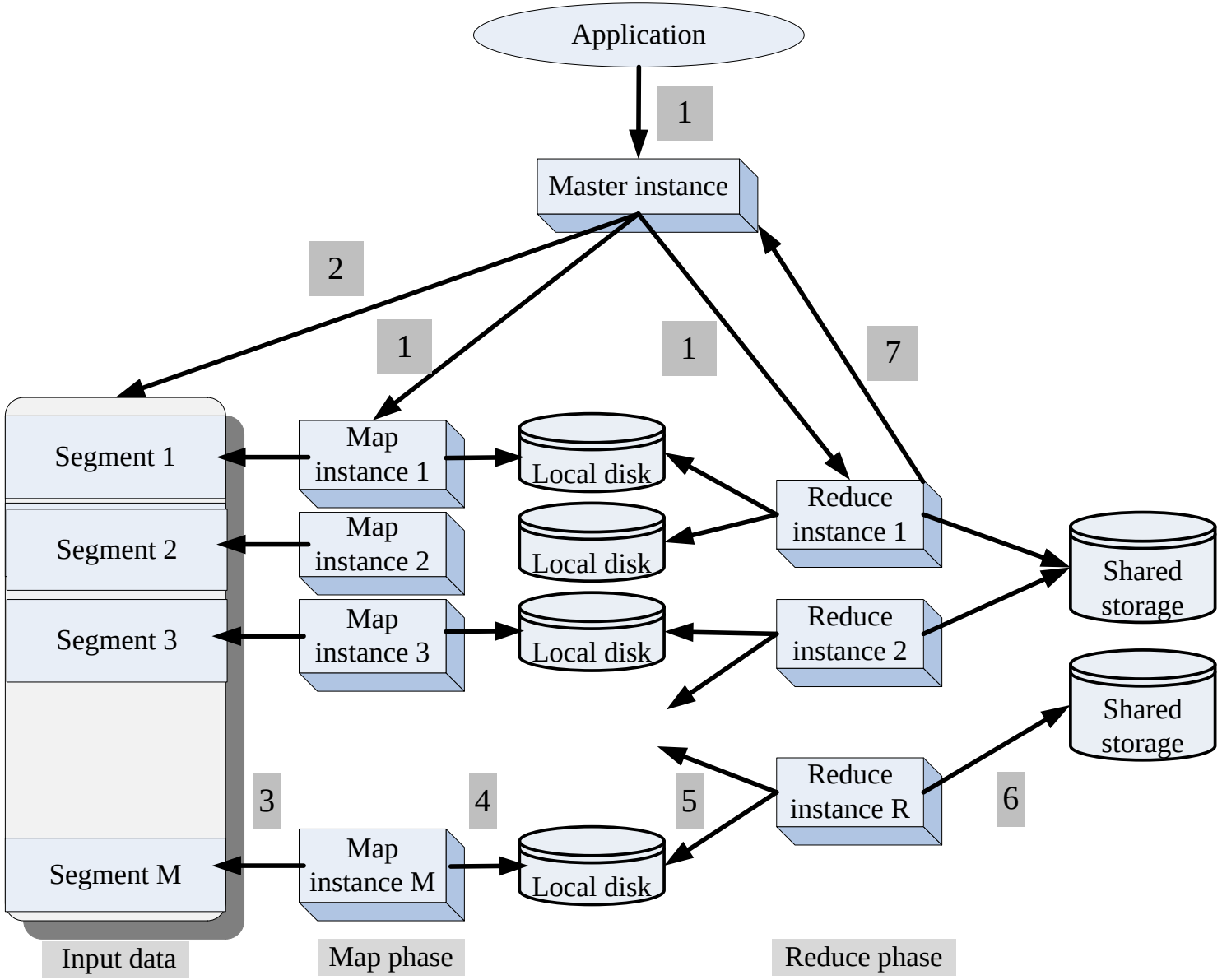
- The API is simple - consists of seven operations:
  - Create - add a node at a given location on the tree.
  - Delete - delete a node.
  - Get data - read data from a node.
  - Set data - write data to a node.
  - Get children - retrieve a list of the children of the node.
  - Synch - wait for the data to propagate.

Elasticity and load distribution

- Elasticity → ability to use as many servers as necessary to optimally respond to cost and timing constraints of an application.
- How to divide the load
  - Transaction processing systems → a front-end distributes the incoming transactions to a number of back-end systems. As the workload increases new back-end systems are added to the pool.
  - For data-intensive batch applications two types of divisible workloads are possible:
    - modularly divisible → the workload partitioning is defined a priori.
    - arbitrarily divisible → the workload can be partitioned into an arbitrarily large number of smaller workloads of equal, or very close size.
- Many applications in physics, biology, and other areas of computational science and engineering obey the arbitrarily divisible load sharing model.

MapReduce philosophy

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

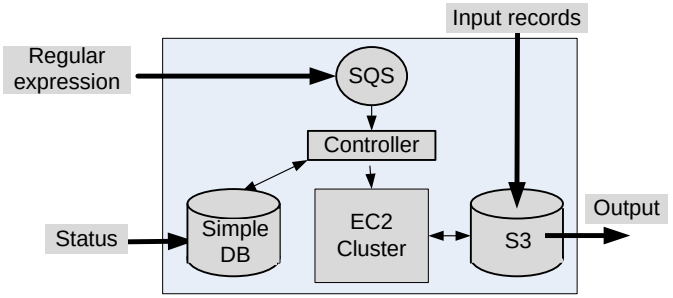
Case study: GrepTheWeb

- The application illustrates the means to
  - create an on-demand infrastructure.
  - run it on a massively distributed system in a manner that allows it to run in parallel and scale up and down, based on the number of users and the problem size.
- GrepTheWeb
  - Performs a search of a very large set of records to identify records that satisfy a regular expression.
  - It is analogous to the Unix *grep* command.
  - The source is a collection of document URLs produced by the Alexa Web Search, a software system that crawls the web every night.
  - Uses message passing to trigger the activities of multiple controller threads which launch the application, initiate processing, shutdown the system, and create billing records.

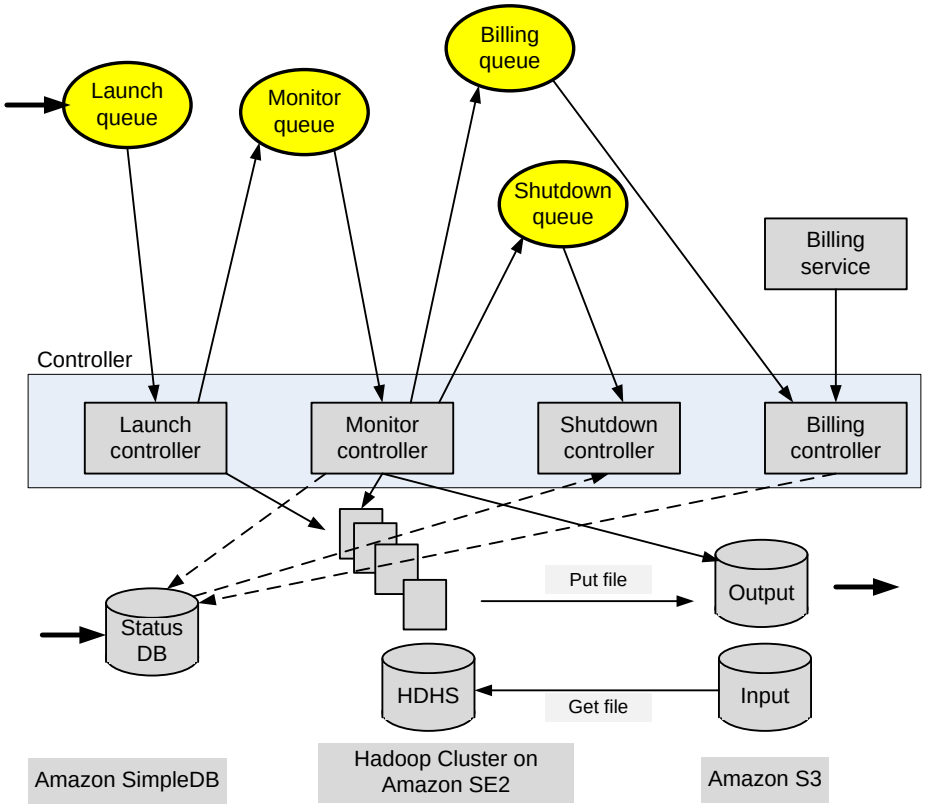
(a) The simplified workflow showing the inputs:

- the regular expression.
- the input records generated by the web crawler.
- the user commands to report the current status and to terminate the processing.

(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



(a)



(b)

Clouds for science and engineering

- The generic problems in virtually all areas of science are:
  - Collection of experimental data.
  - Management of very large volumes of data.
  - Building and execution of models.
  - Integration of data and literature.
  - Documentation of the experiments.
  - Sharing the data with others; data preservation for a long periods of time.

- All these activities require “big” data storage and systems capable to deliver abundant computing cycles.

  Computing clouds are able to provide such resources and support collaborative environments.

Online data discovery

- Phases of data discovery in large scientific data sets:
  - recognition of the information problem.
  - generation of search queries using one or more search engines.
  - evaluation of the search results.
  - evaluation of the web documents.
  - comparing information from different sources.
- Large scientific data sets:
  - biomedical and genomic data from the National Center for Biotechnology Information (NCBI).
  - astrophysics data from NASA.
  - atmospheric data from the National Oceanic and Atmospheric Administration (NOAA) and the National Center for Atmospheric Research (NCAR).

High performance computing on a cloud

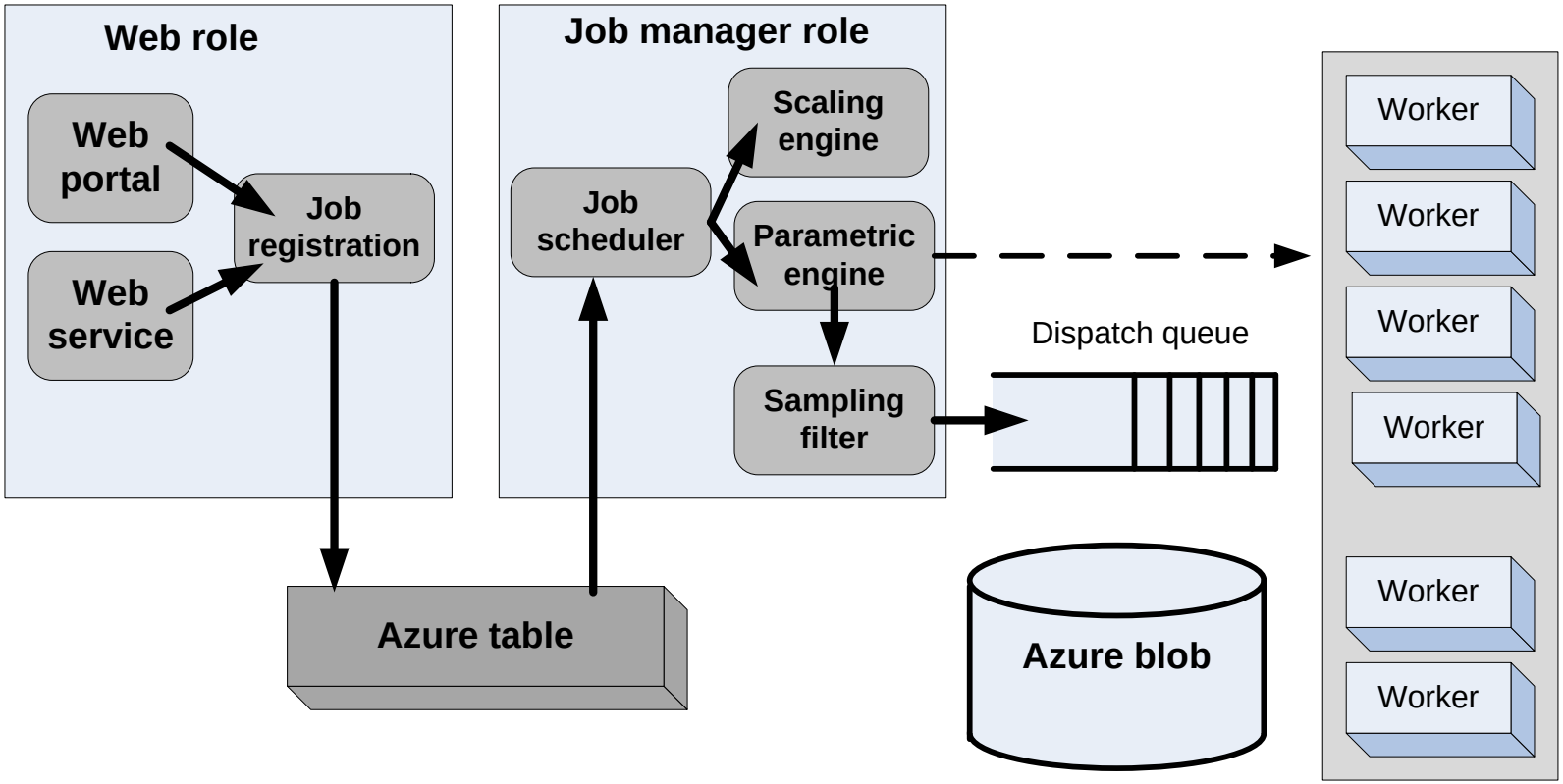
- Comparative benchmark of *EC2* and three supercomputers at the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory. NERSC has some 3,000 researchers and involves 400 projects based on some 600 codes.
- Conclusion – communication-intensive applications are affected by the increased latency and lower bandwidth of the cloud. The low latency and high bandwidth of the interconnection network of a supercomputer cannot be matched by a cloud.

System	DGEMM Gflops	STREAM GB/s	Latency μs	Bndw GB/S	HPL Tflops	FFTE Gflops	PTRANS GB/s	RandAcc GUP/s
Carver	10.2	4.4	2.1	3.4	0.56	21.99	9.35	0.044
Frankl	8.4	2.3	7.8	1.6	0.47	14.24	2.63	0.061
Lawren	9.6	0.7	4.1	1.2	0.46	9.12	1.34	0.013
EC2	4.6	1.7	145	0.06	0.07	1.09	0.29	0.004

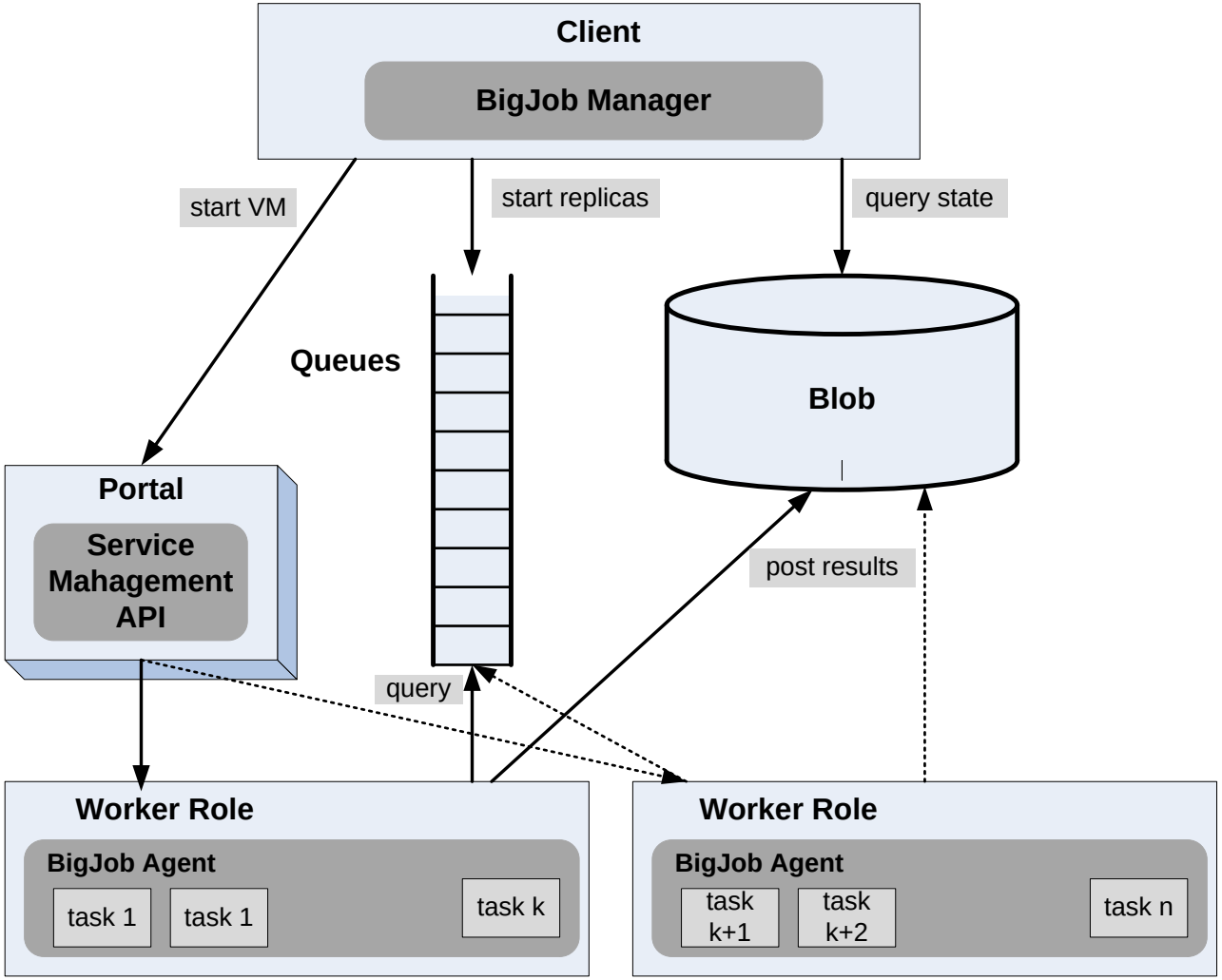
Legacy applications on the cloud

- Is it feasible to run legacy applications on a cloud?
- Cirrus - a general platform for executing legacy Windows applications on the cloud. A Cirrus job - a prologue, commands, and parameters. The prologue sets up the running environment; the commands are sequences of shell scripts including Azure-storage-related commands to transfer data between Azure blob storage and the instance.
- BLAST - a biology code which finds regions of local similarity between sequences; it compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches; used to infer functional and evolutionary relationships between sequences and identify members of gene families.
- AzureBLAST - a version of BLAST running on the Azure platform.

Cirrus


Web role
Web portal
Web service
Job registration
Job manager role
Scaling engine
Job scheduler
Parametric engine
Sampling filter
Dispatch queue
Worker
Worker
Worker
Worker
Worker
Worker
Azure table
Azure blob

Execution of loosely-coupled workloads using the Azure platform



Social computing and digital content

- Networks allowing researchers to share data and provide a virtual environment supporting remote execution of workflows are domain specific:
  - MyExperiment for biology.
  - nanoHub for nanoscience.
- Volunteer computing - a large population of users donate resources such as CPU cycles and storage space for a specific project:
  - Mersenne Prime Search
  - SETI@Home,
  - Folding@home,
  - Storage@Home
  - PlanetLab
- Berkeley Open Infrastructure for Network Computing (BOINC) → middleware for a distributed infrastructure suitable for different applications.