CLOUD COMPUTING THEORY AND PRACTICE

Presentation

Yuan Ma

ECE 428 - Cloud Computing
Department of Electrical and Computer Engineering
University of Michigan-Dearborn

May 16, 2014

Chapter 2
Parallel and Distributed Systems

Cloud computing is intimately tied to parallel and distributed computing

Cloud applications

- Cloud applications are based on the client-server paradigm with relatively simple software, a *thin client*, running on the user's machine while the computations are carried out on the cloud
- Many cloud applications are data-intensive and use a number of instances that run concurrently

Communication protocols

Communication protocols support coordination of distributed processes and transport information through noisy and unreliable communication channels that may lose messages or deliver duplicate, distorted, or out-of-order messages

Checkpoint

- Checkpoints are taken periodically in anticipation of the need to restart a software process when one or more systems fail
- The concept of consistent cuts and distributed snapshots are at the heart of *checkpoint-restart* procedures for long-lasting computations.

Monitor

- Monitors are system components that collect state information from the individual systems
- Security and reliability can only be implemented using information provided by specialized monitors

Parallel Computing

- Parallel computing allows us to solve large problems by splitting them into smaller ones and solving them concurrently
- Parallel hardware and software systems allow us to solve problems demanding more resources than those provided by a single system and, at the same time, to reduce the time required to obtain a solution

The speed-up measures the effectiveness of parallelization.
Definition of *speed-up* of the parallel computation is

$$S(N) = \frac{T(1)}{T(N)}$$

$T(1)$ — the execution time of the sequential computation
$T(N)$ — the execution time when N parallel computations are carried out

Amadahl's Law gives the potential speed-up of a parallel computation; it states that the portion of the computation that cannot be parallelized determines the overall speedup.

Amdahl's Law

If α is the fraction of running time a sequential program spends on nonparallelizable segments of the computation, then

$$S = \frac{1}{\alpha}$$

Amdahl's law applies to a *fixed problem size*. In this case the amount of work assigned to each one of the parallel processes decreases when the number of processes increases, and this affects the efficiency of the parallel execution.

When the problem size is allowed to change, Gustafsons Law gives the *scaled speed-up* with N parallel processes as

$$S(N) = N - \alpha N - 1$$

- Amdahls Law and the *scaled speed-up* assume that all processes are assigned the same amount of work.
- The scaled speed-up assumes that the amount of work assigned to each process is the same, regardless of the problem size. Then, to maintain the same execution time, the number of parallel processes must increase with the problem size.
- The scaled speed-up captures the essence of efficiency, namely that the limitations of the sequential part of a code can be balanced by increasing the problem size.

- Coordination of concurrent computations could be quite challenging and involves overhead.
  - *barrier synchronization*
- Concurrent execution could e very challenging
  - the presence of *deadlocks* is is a potential for concurrent execution of multiple processes or threads
  - *Coffman conditions* must hold simultaneously for a *deadlock* to occur
- Other potential problems related to concurrency like
  - *Livelock*
  - *Priority inversion*

Concurrent processes/tasks can communicate using messages or shared memory.

- Multicore processors sometimes use shared memory, but the shared memory is seldom used in modern supercomputers because shared-memory systems are not scalable.
- Message passing is the communication method used exclusively in large-scale distributed systems, and our discussion is restricted to this communication paradigm.

fine-grain parallelism

In fine-grain parallelism, relatively small blocks of the code can be executed in parallel without the need to communicate or synchronize with other threads or processes

coarse-grain parallelism

- In coarse-grain parallelism, large blocks of code can be executed in parallel
- The speedup of applications displaying fine-grain parallelism is considerably lower than that of coarse-grained applications

Data parallelism

Data parallelismis based on partitioning the data into several blocks and running multiple copies of the same program concurrently, each running on a different data block

- Bit-level parallelism
- Instruction-level parallelism
- Data parallelism or loop parallelism
  - The program loops can be processed in parallel
- Task parallelism
  - The problem can be decomposed into tasks that can be carried out concurrently

Classification of computer architectures based on the number of *concurrent control/instruction* and *data streams*

- Single Instruction, Single Data (SISD)
- Single Instruction, Multiple Data (SIMD)
- Multiple Instructions, Multiple Data (MIMD)

Distributed systems

- A *distributed system* is a collection of autonomous computers that are connected through a network and distribution software called *middleware*, which enables computers to coordinate their activities and to share the resources of the system.
- A distributed systems users perceive the system as a single integrated computing facility.

Characteristics

- Its components are autonomous, scheduling and other resource management and security policies are implemented by each system.
- There are multiple points of control and multiple points of failure, and the resources may not be accessible at all times.
- Distributed systems can be scaled by adding additional resources and can be designed to maintain availability even at low levels of hardware/software/network reliability

Properties of middleware

The middleware should support a set of desirable properties of a distributed system:

- *Access transparency*
  - Local and remote information objects are accessed using identical operations.
- *Location transparency*
  - Information objects are accessed without knowledge of their location.
- *Concurrency transparency*
  - Several processes run concurrently using shared information objects without interference among them.

Properties of middleware cont.

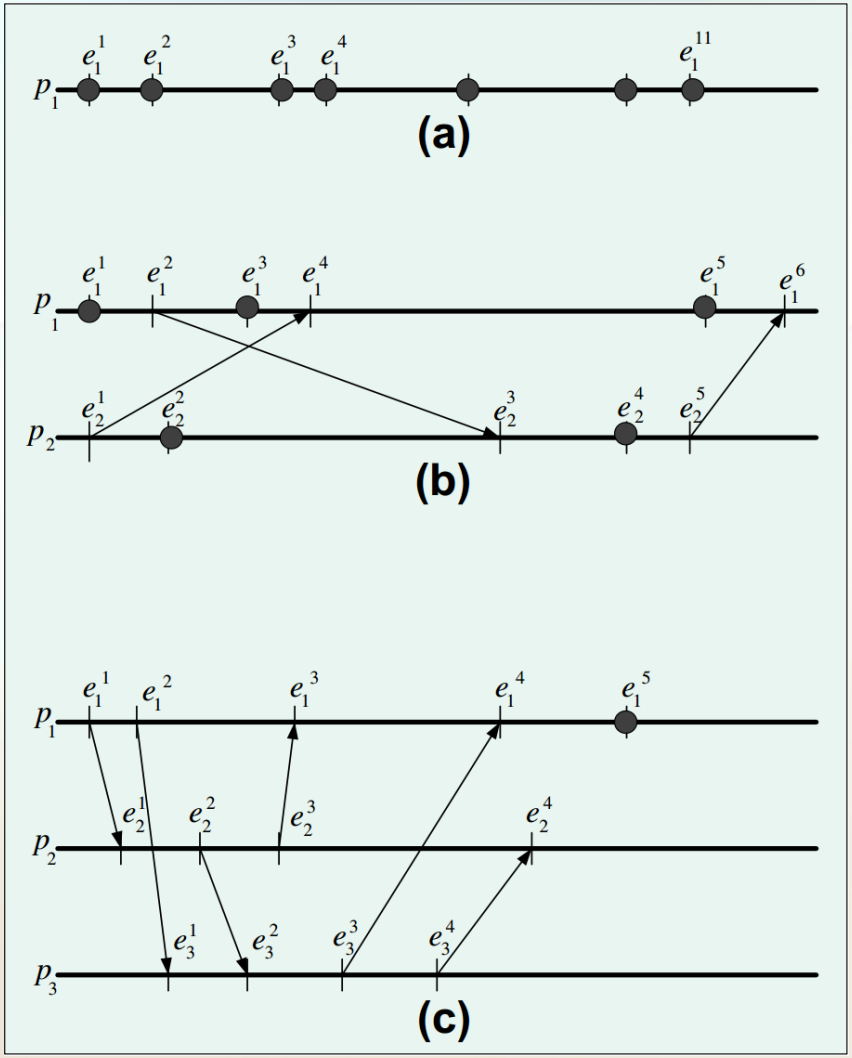
- *Replication transparency*
  - Multiple instances of information objects are used to increase reliability without the knowledge of users or applications.
- *Failure transparency*
  - The concealment of faults.
- *Migration transparency*
  - The information objects in the system are moved without affecting the operation performed on them.
- *Performance transparency*
  - The system can be reconfigured based on the load and quality of service requirements.
- *Scaling transparency*
  - The system and the applications can scale without a change in the system structure and without affecting the applications.

process

- A *process* is a program in execution, and a *thread* is a lightweight process. A thread of execution is the smallest unit of processing that can be scheduled by an operation.
- A process is characterized by its *status*; the state is the ensemble of information we need to restart a process after it was suspended.
- An *event* is a change of state of a process. The events affecting the state of process p_1 are numbered sequentially as $e_i^1, e_i^2, e_i^3, \cdots$, as shown in the *space-time diagram* in Figure 2.1(a). A process p_i is in state σ_i^j immediately after the occurrence of event e_i^j and remains in that status until the occurrence of the next event, e_i^{j+1}.

process group

A *process group* is a collection of cooperating processes; these processes work in concert and communicate with one another to reach a common goal.


p_1
e_1^1 e_1^2 e_1^3 e_1^4 e_1^{11}
(a)
p_1
e_1^1 e_1^2 e_1^3 e_1^4 e_1^5 e_1^6
p_2
e_2^1 e_2^2 e_2^3 e_2^4 e_2^5
(b)
p_1
e_1^1 e_1^2 e_1^3 e_1^4 e_1^5
p_2
e_2^1 e_2^2 e_2^3 e_2^4
p_3
e_3^1 e_3^2 e_3^3 e_3^4
(c)

communication channel

- A *communication channel* provides the means for processes or threads to communicate with one another and coordinate their actions by exchanging messages.
- Without loss of generality, we assume that communication among processes is done only by means of *send(m)* and *receive(m)* communication events, where *m* is a message.
- We use the term *message* for a structured unit of information, which can be interpreted only in a semantic context by the sender and the receiver.
- The *state of a communication channel* is defined as follows:
  - Given two procssses p_i and p_j, the state of the channel, $\xi_{i,j}$, from p_i to p_j consists of messages sent by p_i but not yer received by p_j.

Global state of a process group
Two critical components

- These two abstractions allow us to concentrate on critical properties of distributed systems without the need to discuss the detailed physical properties of the entities involved.
- The model presented is based on the two assumptions:
  - The channel is a unidirectional bit pipe of infinite bandwidth and zero latency, but unreliable.
  - Messages sent through a channel may be lost or distorted or the channel may fail, losing its ability to deliver messages
  - The time a process needs to traverse a set of states is of no concern and that processes may fail or be aborted

Protocol

A protocol is a finite set of messages exchanged among processes to help them coordinate their actions.

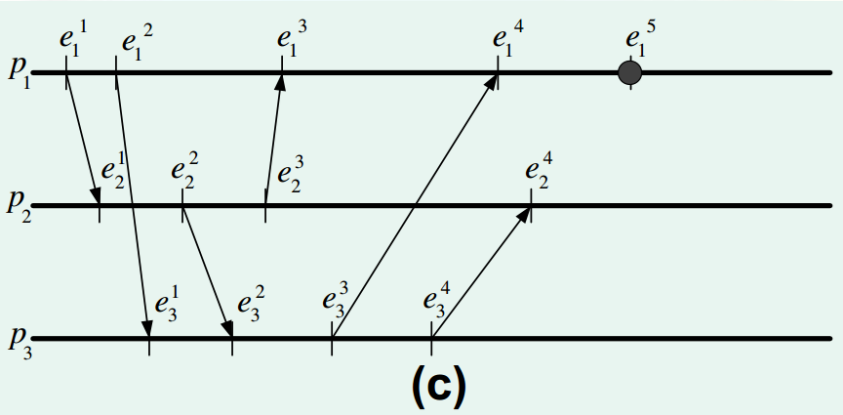


Figure above illustrates the case when communication events are dominant in the local history of processes, p_1, p_2, and p_3.

- In this case only e_1^5 is a local event; all others are communication events.
- The particular protocol requires processes p_2 and p_3 to send messages to the other processes in response to a message from process p_1.

Global state of a process group

- The informal definition of the state of a single process can be extended to collections of communicating processes.
- The *global state of a distributed system* consisting of several processes and communication channels is the union of the states of the individual processes and channels

Some symbols

- h_i^j — the history of process p_i up to and including its j-th event, e_i^j
- σ_i^j — the local process p_i following event e_i^j.

A system consisting of n processes, $p_1, p_2, \ldots, p_i, \ldots, p_n$ with σ_i^j the local state of process pi; then the global state of the system is ann-tuple of local states; then the global state of the system is an n-touple of local states

$$\sum^{(j_1, j_2, \ldots, j_n)} = (\sigma_1^{j_1}, \sigma_2^{j_2}, \cdots, \sigma_i^{j_i}, \cdots, \sigma_n^{j_n}) \tag{1}$$

Global state of a process group

- The state of the channels does not appear explicitly in this definition of the global state because the state of the channels is encoded as part of the local state of the processes communicating through the channels.
- The global states of a distributed computation withnprocesses form an n-dimensional lattice. The elements of this lattice are global states described in Eq. 1.

- Fig. **??**(a) shows the lattice of global states of the distributed computation in Fig. **??**(b).
- This is a two-dimensional lattice because we have two processes, p_1 and p_2.
- The lattice of global states for the distributed computation in Fig **??**(c) is a three-dimensional lattice, and the computation consists of three concurrent processes, p_1, p_2, and p_3.

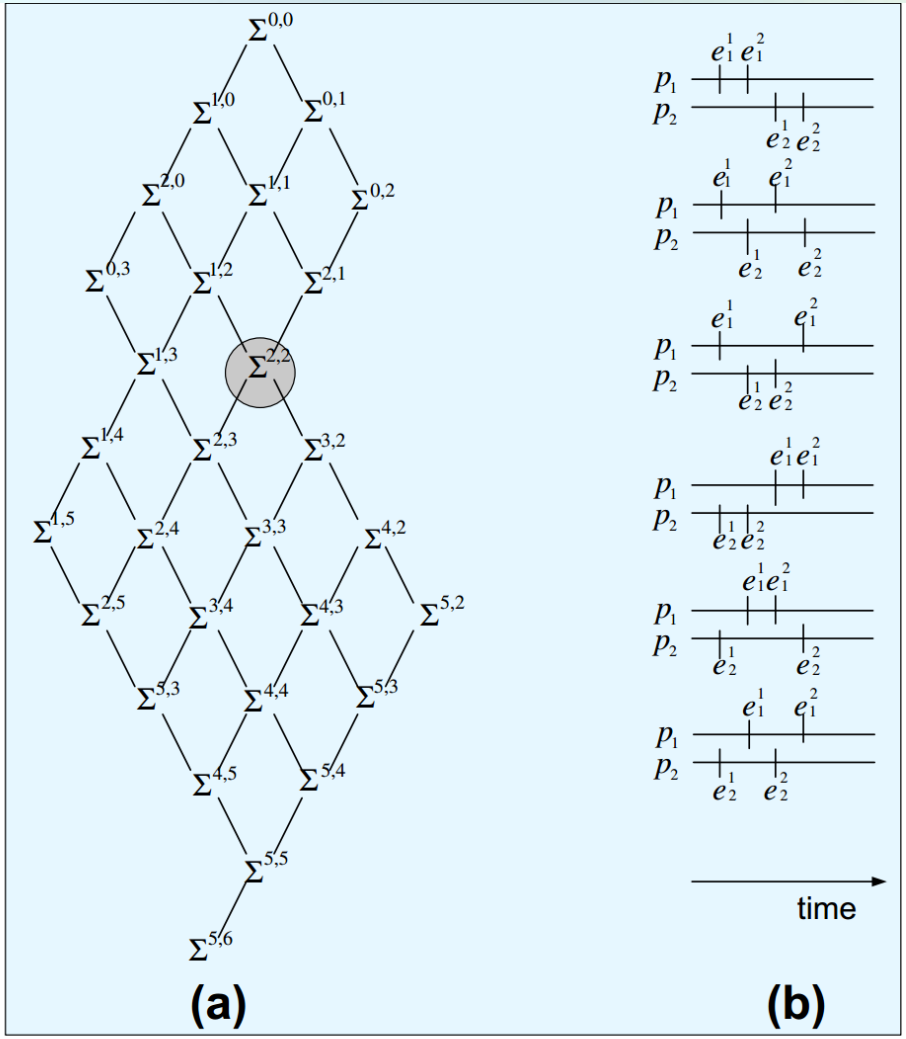


FIGURE 2.2

(a) The lattice of the global states of two processes with the space-time diagrams in Figure 2.2(b). Only the first two events for each thread are shown in Figure 2.2(b). (b) The six possible sequences of events leading to the state $\Sigma^{(2,2)}$.