Understanding Networked Applications:
A First Course

Chapter 17

by
David G. Messerschmitt

Summary

- Performance and quality
- Concurrency
- Scalability
- Operating systems

2

What is performance

- Characterizing non-functional application attributes
- Quantitative metrics
  - Latency, speed, etc
  - Parameterized by load, users, etc
- Performance of application as a whole usually not meaningful
  - Subdivide into tasks

3

Tasks

- Collection of actions that are not meaningfully separated
  - Only completed task is of interest; partial results not useful
  - How does this relate to atomicity in transactions?
- For repetitive tasks, users or operators often care about
  - Task completion time (sec)
  - Task throughput (tasks/sec)

4

Important questions

- For any application
- What are the tasks?
  - Which is important: completion time, throughput, or both, or something else?
  - What are the quantitative objectives?

5

Example: Web surfing

- Task:
  - User: Click on URL, retrieve page, display
  - Server: HTTP request received, HTML page formed as a message, message transmitted
- What performance metric(s) are important to:
  - User?
  - Server?

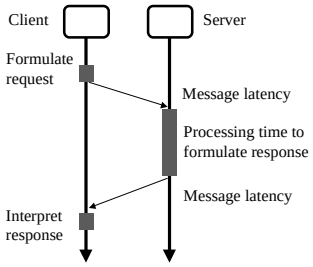
6

Examples

- What are the tasks and their appropriate performance metrics for the following applications:
  - Schwab stock trading system?
  - Stock ticker?
  - ATM system?
  - Travel reservation system?
  - Remote conferencing?

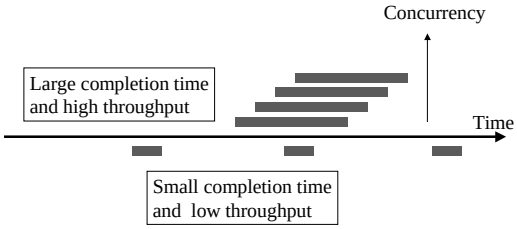
7

Typical components of task completion time



8

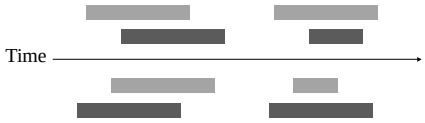
Completion time and throughput are unrelated



9

Concurrency

- Two tasks are concurrent if they overlap in time
- Four cases:



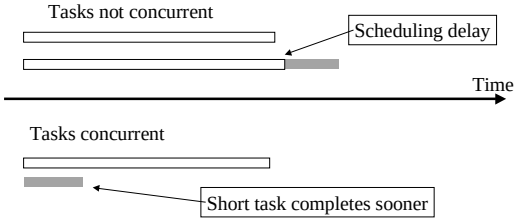
10

Concurrency may be….

- …a way to satisfy intrinsic application requirements
  - e.g. a number of users may generate concurrent tasks
- …a way to improve fairness
- …a way to increase task throughput
  - e.g. assign tasks to different hosts to get more done
- …a way to reduce task completion time
  - e.g. divide a task into subtasks and assign those subtasks to different hosts

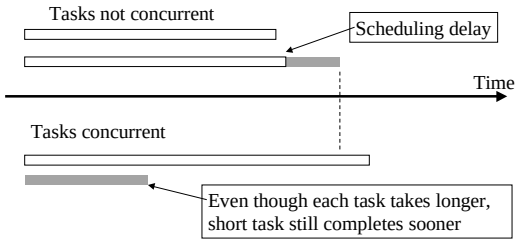
11

Concurrency for fairness



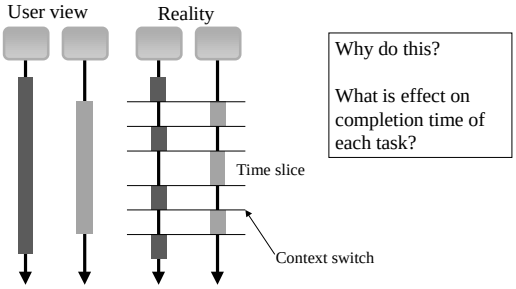
12

Concurrency may contribute to fairness on a single host



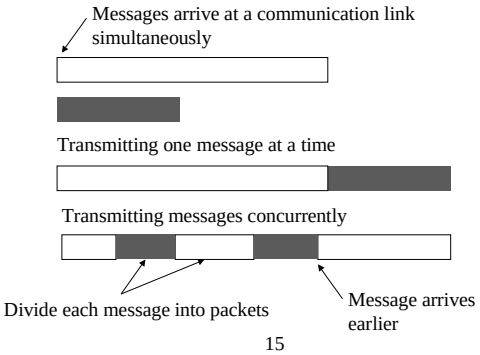
13

Concurrency on a single host



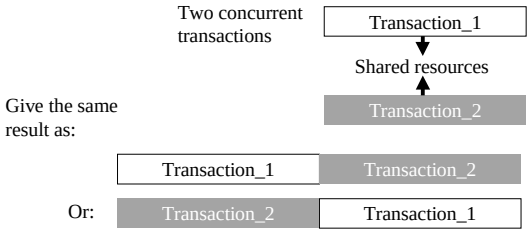
14

Why networks use packets



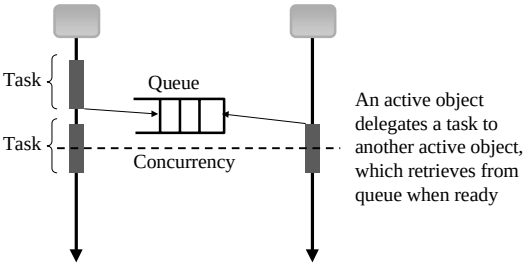
15

Isolation of transactions



16

Messages among active objects



17

Understanding Networked Applications:
A First Course

Scalability

by
David G. Messerschmitt

Definition

- An application is scalable if its performance parameters can be improved as necessary
  - Adding but not replacing equipment is acceptable
  - Re-configuration but not re-programming is acceptable
  - Normally equipment cost should increase no more than linearly with performance

19

Basic technique

- Concurrent tasks are a key to scalability
- Some obstacles:
  - Concurrent tasks aren't there
  - Dependency among tasks
  - Communication overhead
  - Variations in task loads result in congestion

20

Some problems to try to avoid

- Blocking
  - Processing blocked waiting for external events
  - Make sure processor can work on something else
- Duplication
  - In creating concurrent tasks, duplication of effort is created

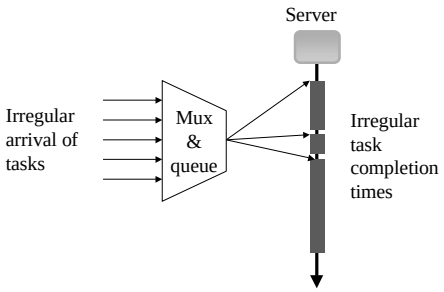
21

Problems to avoid (con't)

- Load balancing
  - Workload not equal among processors
  - Some processors not fully utilized
  - More difficult if concurrent tasks are dependent
- Congestion
  - Workload fluctuates
  - Makes load balancing more difficult because utilization can't be predicted

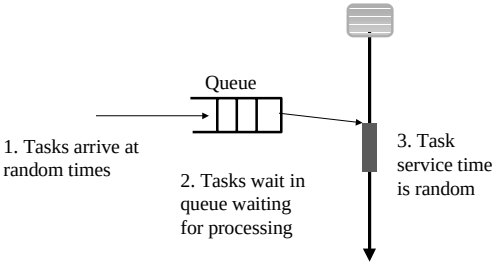
22

Source of congestion



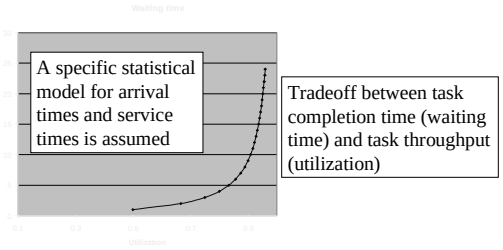
23

Congestion model



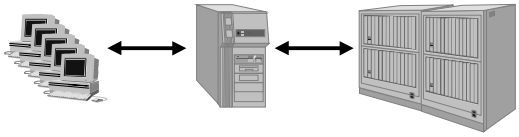
24

Statistical waiting time



25

Example: Web server



How do users know the server is reaching full utilization?

When a single Web server/processor is exhausted, what do we do?

26

Understanding Networked Applications:
A First Course

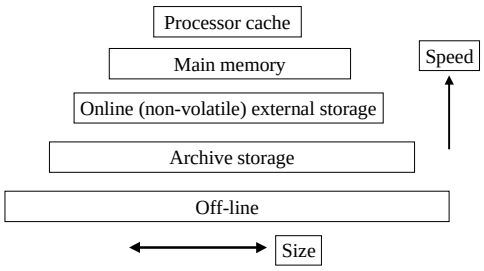
Operating systems

Some OS goals

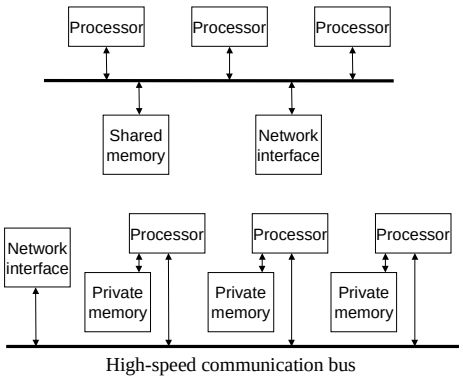
- Abstraction: hide hardware details
- Manage concurrency
  - Multitasking (time-slicing) concurrency
  - Process and thread
  - Inter-process communicaton
- Manage resources
  - Memory, storage, processing, network access

28

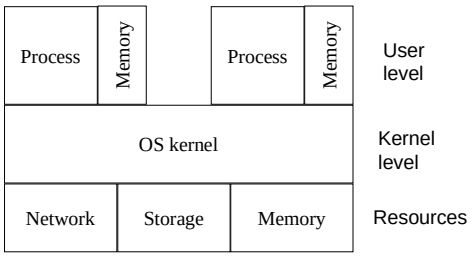
Memory/storage hierarchy



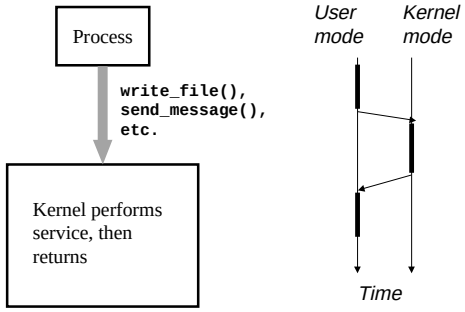
29



30

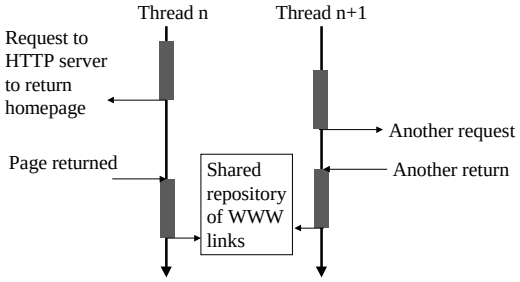


31



32

Threads



33

Understanding Networked Applications:
A First Course

Supplements

By
David G. Messerschmitt

Execution models for objects

- Active object
  - Independent center of activity
  - Works without methods being called
- Passive object
  - Works only in response to method invocations

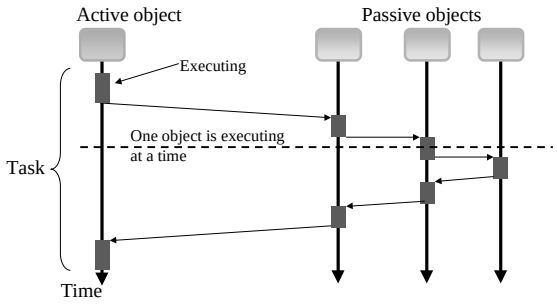
35

Two observations

- For anything to get done, there has to be at least one active object!
- Natural for each active object to work on one task at a time
  - If so there must be at least two active objects for concurrency
  - (although in principle an active object could time slice among tasks, this is much more complicated and not a good idea)

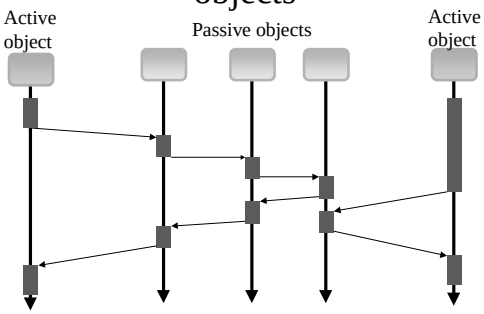
36

Natural for each active object to work on one task



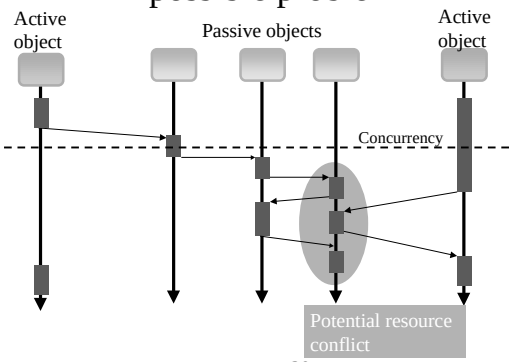
37

Concurrency with two active objects



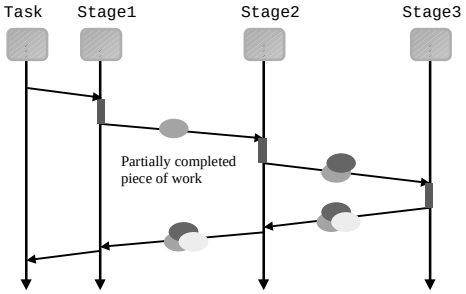
38

A possible problem



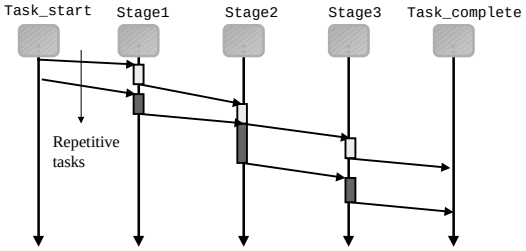
39

Without pipelining



40

With pipelining



41