Chapter 17

by
David G. Messerschmitt

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

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?

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?

Summary

• Performance and quality
• Concurrency
• Scalability
• Operating systems

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)
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?

Completion time and throughput are unrelated

Concurrency

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

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

Concurrency for fairness

Typical components of task completion time

Tasks not concurrent

Tasks concurrent
Concurrent may contribute to fairness on a single host

- Tasks not concurrent: Scheduling delay
- Tasks concurrent: Even though each task takes longer, short task still completes sooner

User view vs. Reality

- User view: Time slice
- Reality: Context switch

Why networks use packets

- Messages arrive at a communication link simultaneously
- Transmitting one message at a time
- Transmitting messages concurrently: Message arrives earlier
- Divide each message into packets

Isolation of transactions

- Two concurrent transactions
- Transaction_1
- Transaction_2
- Shared resources
- Transaction_1
- Transaction_2
- Transaction_2
- Transaction_1

Give the same result as:

- Or:

Messages among active objects

- An active object delegates a task to another active object, which retrieves from queue when ready

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

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

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

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

Source of congestion

![Diagram of source of congestion]

Congestion model

![Diagram of congestion model]

1. Tasks arrive at random times
2. Tasks wait in queue waiting for processing
3. Task service time is random
Statistical waiting time

A specific statistical model for arrival times and service times is assumed. Tradeoff between task completion time (waiting time) and task throughput (utilization).

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?

Some OS goals

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

Operating systems

Understanding Networked Applications: A First Course

Memory/storage hierarchy

Processor cache
Main memory
Online (non-volatile) external storage
Archive storage
Off-line

Size
Speed

Processor
Shared memory
Network interface

Processor
Private memory

Processor
Private memory

Processor
Private memory

Network interface

High-speed communication bus
Thread n
Request to HTTP server to return homepage
Page returned

Thread n+1
Shared repository of WWW links

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)
Natural for each active object to work on one task

Concurrency with two active objects

A possible problem

Without pipelining

With pipelining