

EEC 686/785

Modeling & Performance Evaluation of Computer Systems

Lecture 3

Wenbing Zhao

Department of Electrical and Computer Engineering
Cleveland State University
wenbing@ieee.org

(based on Dr. Raj jain's lecture notes)

Outline

- Review of lecture 2
- Part II: Measurement Techniques and Tools
 - Types of Workload
 - The Art of Workload Selection

Commonly Used Performance Metrics

- Response time, Reaction time
- Turnaround time
- Stretch factor
- Throughput
- Capacity (nominal, typical, knee)
- Efficiency
- Utilization
- Reliability, availability

Systematic Approach

- Define the system,
- List its services, metrics, parameters,
- Decide factors, evaluation technique, workload, experimental design,
- Analyze the data, and present results

Selecting Evaluation Techniques

- The life-cycle stage is the key
- Other considerations are:
 - Time available, tools available,
 - Accuracy required,
 - Trade-offs to be evaluated,
 - Cost,
 - Salability of results

Selecting Metrics

- Include:
 - Performance:
 - Time: responsiveness
 - Rate: throughput or productivity
 - Resource: utilization
 - Error rate, probability
 - Time to failure and duration
- Consider including:
 - Mean and variance
 - Individual and global
- Selection criteria:
 - Low-variability
 - Non-redundancy
 - completeness

Part II: Measurement Techniques and Tools

- *Measurements are not to provide numbers but insight*
– Ingrid Bucher
- What are the different types of workloads?
- Which workloads are commonly used by other analysts
- How are the appropriate workload types selected?
- How is the measured workload data summarized?
- How is the system performance monitored?
- How can the desired workload be placed on the system in a controlled manner?
- How are the results of the evaluation presented?

Types of Workloads

- **Test workload:** any workload used in performance studies.
 - Test workload can be real or synthetic
- **Real workload:** observed on a system being used for normal operations
 - It cannot be repeated => in general not suitable for use as a test workload

Types of Workloads

- **Synthetic workload:** a test workload that is similar to real workload and can be applied repeatedly in a controlled manner
 - Do not need real-world data files, which may be large and contain sensitive data
 - Easily modified without affecting operation
 - Easily ported to different systems due to its small size
 - May have built-in measurement capabilities

Type of Workloads

- The following types of test workloads have been used to compare computer processors and timesharing systems
 - Addition instruction
 - Instruction mixes
 - Kernels
 - Synthetic programs
 - Application benchmarks

Addition Instruction

- Processors were the most expensive and most used components of the system
- Addition was the most frequent instruction
- => The computer with faster addition instruction was considered to be the better performer

Instruction Mixes

- As the number of complexity of instructions supported by processors grew, the addition instruction was no longer sufficient
- **Instruction mix = instructions + usage frequency**
- Gibson mix: developed by Jack C. Gibson in 1959 for IBM 704 systems

1. Load and Store	31.2
2. Fixed-Point Add and Subtract	6.1
3. Compares	3.8
4. Branches	16.6
5. Floating Add and Subtract	6.9
6. Floating Multiply	3.8
7. Floating Divide	1.5
8. Fixed-point Multiply	0.6
9. Fixed-point Divide	0.2
10. Shifting	4.4
11. Logical, And, Or, etc.	1.6
12. Instructions Not Using Registers	5.3
13. Indexing	18.0
Total	100.0

Disadvantages of Instruction Mixes

- Complex classes of instructions not reflected in the mixes
- Instruction time varies with
 - Addressing modes
 - Cache hit rates
 - Pipeline efficiency
 - Interference from other devices during processor-memory access cycles
 - Parameter values
 - Frequency of zeros as a parameter
 - ...

Kernels

- Kernel = the most frequent function provided by processors
- Commonly used kernels: Sieve, Puzzle, Tree Searching, Ackerman's Function, Matrix Inversion, and Sorting
- Disadvantages: do not make use of any operating system services and I/O devices

Synthetic Programs

- To measure I/O performance exerciser loops were used
- An exerciser loop makes a specified number of service calls or I/O requests
 - Can be used to calculate the average CPU time and elapsed time for each service call
- The first exerciser loop was by Bucholz (1969) who called it a synthetic program

Synthetic Programs

- A sample exerciser

```

!Perform a number of read I/Os
DO 100 i=1,Num_Reads
  READ(1'i),Record
100 CONTINUE
!Do computation
DO 200 j=1,Num_Computes
  DO 200 i=1,500
200 Record(i)=1 + i + i*i + i*i*i
!Perform a number of write I/Os
DO 300 i=1,Num_Writes
  WRITE(2'i),Record
300 CONTINUE
500 CONTINUE
CALL Get_Time(CPU2,Elapsed2) !Get ending time

```

Advantages of Synthetic Programs

- Quickly developed and given to different vendors
- No real data files
- Easily modified and ported to different systems
- Have built-in measurement capabilities
- Measurement process is automated
- Repeated easily on successive versions of the operating systems

Disadvantages of Synthetic Programs

- Too small
- Do not make representative memory or disk references
- Mechanisms for page faults and disk cache may not be adequately exercised
- CPU-I/O overlap may not be representative
- Loops may create synchronizations
=> better or worse performance

Application Benchmarks

- If the computer systems to be compared are to be used for a particular application, a representative subset of functions for that application can be used
- Such benchmarks make use of almost all resources in the system, including processors, I/O devices, networks and databases
- Example: debit-credit benchmark

Popular Benchmarks

- **Benchmarking**: the process of performance comparison for two or more systems by measurements
- **Benchmark** = workload (except instruction mixes)
 - Kernels, synthetic programs, and application-level workload
- Alternatively definition of benchmark: set of programs taken from real workloads
 - It is rarely used

Sieve

- Based on Eratosthenes' sieve algorithm: find all prime numbers below a given number n
- Algorithm:
 - Write down all integers from 1 to n
 - Strike out all multiples of k , for $k = 2, 3, \dots, \sqrt{n}$

Sieve Example

- Write down all numbers from 1 to 20. mark all as prime:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
- Remove all multiples of 2 from the list of primes:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
- The next integer in the sequence is 3. Remove all multiples of 3:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
- $5 \geq \sqrt{20} \Rightarrow$ stop

Ackermann's Function

- To access the efficiency of the procedure-calling mechanisms. The function has two parameters and is defined recursively
- Ackermann($3, n$) evaluated for values of n from one to six
- Metrics
 - Average execution time per call
 - Number of instructions executed per call
 - Stack space per call

Ackermann's Function

- Verification: $\text{ackermann}(3, n) = 2^{n+3} - 3$
- Number of recursive calls in evaluating Ackermann($3, n$):
 $(512 \times 4^{n-1} - 15 \times 2^{n+3} + 9n + 37)/3$
 \Rightarrow Execution time per call
- Depth of the procedure calls = $2^{n+3} - 4$
 \Rightarrow stack space required doubles when $n \leftarrow n+1$

Ackermann Program in Simula

```

BEGIN
INTEGER n;           !Loop index;
INTEGER j;           !Function value;
INTEGER num_calls;  !Number of recursive calls;
INTEGER k;           !Contains 2**(n+3);
INTEGER k1;         !Contains 4**(n-1);
REAL t1,t2;         !CPU time values;

INTEGER PROCEDURE Ackermann(m,n); VALUE m,n; INTEGER m,n;
Ackermann := IF n=0 THEN n+1
              ELSE IF n=0 THEN Ackermann(m-1,1)
              ELSE Ackermann(m-1,Ackermann(m,n-1));
!Main Program;
k := 16; k1 := 1;   !Initialize k and k1 for n=1;
FOR n := 1 STEP 1 UNTIL 6 DO
BEGIN
t1 := CPUTIME;     !Beginning CPU time;
j := Ackermann(3,n); !Compute the function;
t2 := CPUTIME;     !Ending CPU time;
IF j <> k-3 THEN OUTTEXT("Wrong Value");
OUTTEXT("Net CPU Time for Ackermann(3,");
OUTINT(n,1); OUTTEXT(") is");
OUTREAL(t2-t1,7,15); OUTIMAGE;
Num_calls := (512*k1-15*k+9*n+37)/3;
OUTTEXT("CPU Time per call:");
OUTREAL((t2-t1)/num_calls,7,15);
OUTIMAGE;
k1 := 4*k1;       !Update k1 for the next n;
k := 2*k;        !Update k for the next n;
END
END

```

12 September 2005

Debit-Credit Benchmark

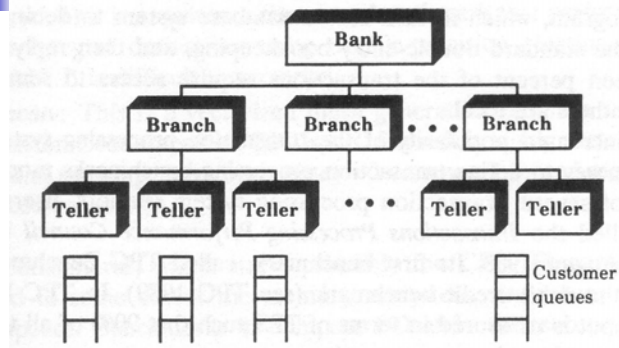
- A *de facto* standard for transaction processing systems
- First recorded in Anonymous et al (1975)
- In 1973, a retail bank wanted to put its 1000 branches, 10,000 tellers, and 10,000,000 accounts online with a peak load of 100 TPS
- Each TPS requires 10 branches, 100 tellers, and 100,000 accounts

12 September 2005

EEC686/785

Wenbing Zhao

Debit-Credit Benchmark



12 September 2005

EEC686/785

Wenbing Zhao

Debit-Credit Benchmark

- **Metric:** price/performance ratio
- **Cost** = total expenses for a 5-year period on purchase, installation, and maintenance of the hardware and software in the machine room
- Cost does not include expenditures for terminals, communications, application development, or operations

12 September 2005

EEC686/785

Wenbing Zhao

Debit-Credit Benchmark

- **Performance:** throughput in terms of TPS such that 95% of all transactions provide one second or less response time
- **Response time:** measured as the time interval between the arrival of the last bit from the communications line and the sending of the first bit to the communications line

Pseudo-Code for Debit-Credit Benchmark

Begin-Transaction

Read message from terminal	(100 bytes)
Rewrite account	(100 bytes, random)
Write history	(50 bytes, sequential)
Rewrite teller	(100 bytes, random)
Rewrite branch	(100 bytes, random)
Write message to the terminal	(200 bytes)

Commit-Transaction

Pseudo-Code for Debit-Credit Benchmark

- Four record types: account, teller, branch, and history
- 15% of the transactions require remote access
- Transactions processing performance council (TPC) was formed in August 1988
- TPC Benchmark A is a variant of the debit-credit
- Metrics: TPS such that 90% of all transactions provide two seconds or less response time

SPEC Benchmark Suite

- The Systems Performance Evaluation Corporation (SPEC) is a nonprofit corporation formed by leading computer vendors to develop a standard set of benchmarks
- Release 1.0 of the SPEC benchmark suite consists of 10 benchmarks drawn from various engineering and scientific applications
 - GCC, Espresso, Spice 2g6, Deduc, NASA7, LI, Eqntott, Matrix300, Fpppp, Tomcatv
- These benchmarks are designed to compare CPU speeds

SPEC Benchmark Suite

- SPEC Web site: <http://www.spec.org/benchmarks.html>
- Latest SPEC benchmarks cover much broader areas:
 - CPU
 - Graphics/Applications
 - High Performance Computing (HPC) and MPI
 - Java client/server
 - Mail servers
 - Network file systems
 - Web servers
- The benchmark programs are available commercially

LMbench

- LMbench is a portable performance analysis toolkit for UNIX systems
 - It measures a system's ability to transfer data between processor, cache, memory, network, and disk
- Developed by Larry McVoy's and Carl Staelin
- LMbench Web site: <http://www.bitmover.com/lmbench/>
- First published in Usenix conference in 1996 (**won best paper award**)
- Latest version of LMbench tool can be downloaded (for free) at <http://www.bitmover.com/lmbench/lmbench3.tar.gz>

LMbench

- Bandwidth benchmarks
 - Cached file read
 - Memory copy
 - Memory read
 - Memory write
 - Pipe
 - TCP

LMbench

- Latency benchmarks
 - Context switching
 - Networking: connection establishment, pipe, TCP, UDP, and RPC
 - File system creates and deletes
 - Process creation
 - Signal handling
 - System call overhead
 - Memory read latency
- Misc. such as processor clock rate calculation

Other Benchmarks

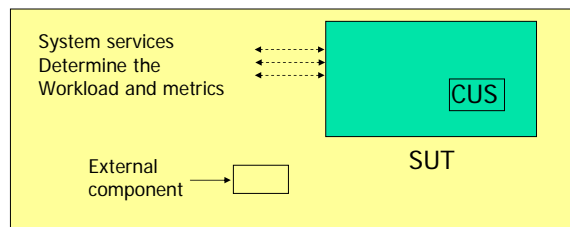
- Whetstone
- U.S. Steel
- LINPACK
- Dhrystone
- Doduc
- TOP
- Lawrence Livermore Loops
- Digital Review Labs
- Abingdon Cross Image-Processing Benchmarks

The Art of Workload Selection

- Services exercised
- Level of detail
- Loading level
- Impact of other components
- Timeliness

Services Exercised

- **SUT = System Under Test:** the complete set of components that are being purchased or being designed
- **CUS = Component Under Study**



Services Exercised

- Do not confuse SUT with CUS
- **Metrics depend upon SUT:** they should reflect the performance of services provided at the system level rather than at the component level
 - MIPS is ok for two CPUs but not for two timesharing systems

Services Exercised

- **Workload depends upon the system.**

Examples:

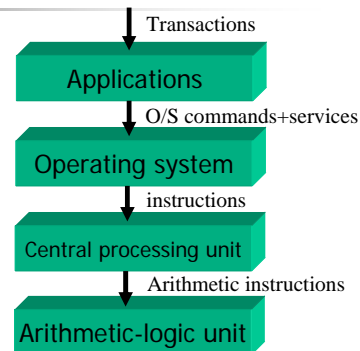
- CPU: instructions
- System: transactions
- Transactions not good for CPU and vice versa
- Two systems identical except for CPU
 - Comparing systems: use transactions
 - Comparing CPUs: use instructions
- Multiple services: exercise as complete a set of services as possible

Services Exercised

- The requests at the service-interface level of the SUT should be used to specify or measure the workload
- One should carefully distinguish between the SUT and CUS

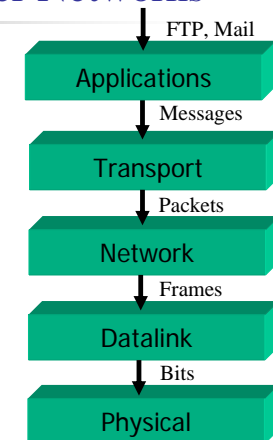
Timesharing Systems

- Applications => application benchmark
- Operation System => synthetic program
- Central Processing Unit => instruction mixes
- Arithmetic Logic Unit => addition instruction



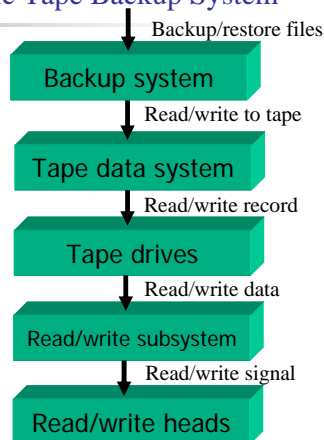
Example 5.1: Computer Networks

- Application layer:
 - Frequency of various types of apps and their associated characteristics
- Transport layer:
 - Frequency, sizes, and other characteristics of various messages
- Network layer:
 - Source-destination matrix, distance between s and d, characteristics of packets transmitted
- Datalink layer:
 - Characteristics of frames transmitted
- Physical layer:
 - Frequency of various bit patterns



Example 5.2: Magnetic Tape Backup System

- A magnetic tape backup system consists of several tape data systems. Starting from a high level and moving down to lower levels



Example 5.2: Magnetic Tape Backup System

- Backup system
 - **Services:** backup files, backup changed files, restore files, list backed-up files
 - **Factors:** file-system size, batch or background process, incremental or full backups
 - **Metrics:** backup time, restore time
 - **Workload:** a computer system with files to be backed up. Vary frequency of backups

Example 5.2: Magnetic Tape Backup System

- Tape data system
 - **Services:** read/write to the tape, read tape label, autoloader tapes
 - **Factors:** type of tape drive
 - **Metrics:** speed, reliability, time between failures
 - **Workload:** a synthetic program generating representative tape I/O requests

Example 5.2: Magnetic Tape Backup System

- Tape drives
 - **Services:** read record, write record, rewind, find record, move to end of tape, move to beginning of tape
 - **Factors:** cartridge or reel tapes, drive size
 - **Metrics:** time for each type of service, for example, time to read record and to write record, speed (requests/time), noise, power dissipation
 - **Workload:** a synthetic program exerciser generating various types of requests in a representative manner

Example 5.2: Magnetic Tape Backup System

- Read/write subsystem
 - **Services:** read data, write data (as digital signals)
 - **Factors:** data-encoding techniques, implementation technology (CMOS, TTL, and so forth)
 - **Metrics:** coding density, I/O bandwidth (bits per second)
 - **Workload:** read/write data streams with varying patterns of bits

Example 5.2: Magnetic Tape Backup System

- Read/Write heads
 - **Services:** read signal, write signal (electrical signals)
 - **Factors:** composition, interhead spacing, gap sizing, number of heads in parallel
 - **Metrics:** magnetic field strength, hysteresis
 - **Workload:** read/write currents of various amplitudes, tapes moving at various speeds

Level of Detail

- This step is to determine the **level of details** in recording and thus reproducing the requests for the chosen services
- **Most frequent request**
 - Examples: addition instruction, debit-credit, kernels
 - Valid if one service is much more frequent than others
- **Frequency of request types**
 - Examples: instruction mixes
 - Context sensitivity = use set of services
 - History-sensitive mechanisms (caching) => context sensitivity

Level of Detail

- **Time-stamped sequence of requests**, also called a **trace**
 - May be too detailed
 - Not convenient for analytical modeling
 - May require exact reproduction of component behavior
- **Average resource demand**
 - Used for analytical modeling
 - Grouped similar services in classes

Level of Detail

- **Distribution of resource demands**
 - Used if variance is large
 - Used if the distribution impacts the performance
- Workload used in simulation and analytical modeling are called
 - Nonexecutable: used in analytical/simulation modeling
 - Executable workload: can be executed directly on a system

Representativeness

- The test workload and real workload should have the same
 - Elapsed time
 - Resource demands
 - Resource usage profile: sequence and the amounts in which different resources are used

Timeliness

- Users are a moving target
- New systems => new workloads
- Users tend to optimize the demand
 - Fast multiplication => higher frequency of multiplication instructions
- Important to monitor user behavior on an ongoing basis

Other Considerations in Workload Selection

- **Loading level:** a workload may exercise a system to its
 - Full capacity (best case)
 - Beyond its capacity (worst case)
 - At the load level observed in real workload (typical case)
 - For procurement purposes => typical
 - For design => best to worst, all cases
- **Impact of external components**
 - Do not use a workload that makes external component a bottleneck => all alternatives in the system give equally good performance
- **Repeatability:** results can be easily reproduced without too much variance