



EEC 686/785

Modeling & Performance Evaluation of Computer Systems

Lecture 4

Wenbing Zhao

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

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



Outline

2

- Review of lecture 3
- Monitors
- Program execution monitors
- Accounting logs



Types of Work Load

- Test workload
 - Real workload
 - Synthetic workload
- Test workload used
 - Addition instruction
 - Instruction mixes
 - Kernels
 - Synthetic programs
 - Application benchmarks



The Art of Workload Selection

- Services exercised
 - SUT, CUT
 - Metrics and workload depend on the system
 - The requests at the service-interface level of the SUT should be used to specify or measure the workload
- Level of detail
 - Most frequent request
 - Frequency of request types
 - Time-stamped sequence of requests
 - Average resource demand
 - Distribution of resource demands



The Art of Workload Selection

- Timeliness
- Loading level
- Impact of other components
- Repeatability



Monitors

- **Monitor:** a tool used to observe the activities on a system
- Usage:
 - To improve software performance. Find frequently used segments of the software
 - To measure resource utilizations and to find the performance bottleneck
 - To tune the system
 - To characterize the workload
 - To find model parameters, to validate models, and to develop inputs for models



Monitor Terminology

- **Event:** a change in the system state
 - Examples: process context switching, beginning of seek on a disk, and arrival of a packet
- **Trace:** a log of events, usually including the time, type and other parameters associated with the event
- **Overhead:** artifact, *i.e.*, monitors may consume system resources and hence slightly perturb the system operation
- **Domain:** the set of activities observable by the monitor
 - Domain of accounting logs: CPU time, number of disks, paging I/O's, elapsed time for each user session, etc.



Monitor Terminology

- **Input rate:** the maximum frequency of events that a monitor can correctly observe
 - **Burst-mode rate:** the rate at which an event can occur for a short duration
 - **Sustained rate:** can tolerate for long durations
- **Resolution:** the coarseness of the information observed
 - For example, record time only in units of 16 milliseconds
- **Input width:** the number of bits of information recorded on a event



Monitor Classification

- **By implementation**

- Software monitor
- Hardware monitor
- Firmware monitor
- Hybrid monitor

- **By trigger mechanism**

- **Event-driven**: good for rare events
- **Timer-driven** (sampling monitor): good for frequent events



Monitor Classification

- **By result display ability**

- **Online monitors**: display the system state continuously
- **Batch monitors**: collect the data that can be analyzed later

- Example: a particular monitor may be classified as a hybrid-sampling-batch monitor



Software Monitors

- Used for operating systems and higher level software such as networks and databases
- Suitable only if the input rate is low
- Also used if overhead is not an issue
 - For example, in an instruction-tracing monitor, every single-user instruction executed may be interrupted => the software monitor can be used on if the time to execute the program is not important



Software Monitors

- Compared to hardware monitor
 - Lower input rates
 - Lower resolutions
 - Higher overhead
 - Higher input widths
 - Higher recording capacities
 - Easier to develop
 - Easier to modify



Issues in Software-Monitor Design

- **Activation mechanism:** how to trigger data collection routine
 - **Trap instruction:** software interrupt mechanism that transfer control to a data collection routine
 - **Trace mode:** the processor runs in trace mode – instruction execution is interrupted after every instruction, and the control is passed to a data collection routine
 - **Timer interrupt:** use service provided by OS to transfer control to data collection routine periodically



Issues in Software-Monitor Design

- **Buffer size:** record data in buffers initially and later write to disk or other secondary storage
 - Large => too much time per write
 - Small => too many write operations
 - Optimum = function of input rate, input width, and the emptying rate



Issues in Software-Monitor Design

- **Number of buffers:** buffers are usually organized in a ring so that the recording (buffer-emptying) process follows the monitoring (buffer-filling) process as closely as possible
 - Minimum two
 - More to allow for variation in the filling and emptying rate



Issues in Software-Monitor Design

- **Buffer overflow:** there is always a finite probability that all buffers become full
 - Overwrite: old info is lost
 - Block: new info is lost

 - In any case, record buffer overflows.
 - Similarly, counters can be stuck-at or zeroed on overflow



Issues in Software-Monitor Design

- **Data compression or analysis:** process the data as it is observed
 - Reduces the space required
 - Increases the overhead
- **On/Off switch:** use conditional (IF ... THEN ...)
 - Reduces overhead
 - Helps during development and debugging



Issues in Software-Monitor Design

- **Language:** use the same language as the system
- **Priority**
 - Should not affect system operations => Low priority for the monitor
 - Timely recording => High priority



Issues in Software-Monitor Design

- **Abnormal-events monitor:** should observe normal as well as abnormal events. For example
 - System initialization
 - Device failure
 - Program failures
- Users prefer to monitor abnormal events
 - Abnormal events occur at a lower rate and impose less monitoring overhead
 - Abnormal events help the user take preventive action



Hardware Monitors

- Separate equipment attached to the system being monitored via probes
- No system resources consumed in monitoring
- Higher input rate
- Less chances of introducing bugs into the system operation

Hardware Monitors Components

- **Probes:** to observe signals at desirable points in system hardware
- **Counters:** incremented whenever a particular event occurs
- **Logic elements:** AND, OR, and other logic gates. Combinations are used to indicate events that may increment the counters
- **Comparators:** to compare counter or signal values with preset values
- **Mapping hardware:** consists of multiple comparators and counters. To allow histograms
- **Timer:** for time-stamping or sampling
- **Tape/disk:** to store the data
- **Probe-point libraries:** a list of points on the system where the probes can be attached

Software vs. Hardware Monitors

Criterion	Hardware Monitor	Software Monitor
Domain	Difficult to monitor operating system events	Difficult to monitor hardware events unless recognizable by an instruction
Input Rate	Sampling rate of 10^5 per second possible	Sampling rate limited by the process MIPS and overhead required
Time Resolution	10 ns is possible	General 10 to 16 ms
Expertise	Requires intimate knowledge of hardware	Requires intimate knowledge of software

Software vs. Hardware Monitors

Criterion	Hardware Monitor	Software Monitor
Recording Capacity	Limited by memory and secondary storage. Not a problem currently	Limited by overhead desired
Input Width	Can record several simultaneous	Can't record several simultaneous events unless there are multiple processes
Monitor Overhead	None	Overhead depends upon input rate and input width. Less than 5% adequate and more than 100% possible
Portability	General portable	Specific to an OS

Software vs. Hardware Monitors

Criterion	Hardware Monitor	Software Monitor
Availability	Monitoring continues even during system malfunction or failure	Can't monitor during system crash
Errors	Possible to connect the probes to wrong points	Once debugged, errors are rare
Cost	High	Medium



Firmware Monitors

- Implemented by modifying the processor microcode
- For applications that fall in between the software and hardware monitoring boundaries
- Similar to software monitors
- Tighter timer limitations
- Very limited data reduction, if any



Example Firmware Monitors

- Network interfaces microprogrammed to monitor all traffic on the network
- Address profiles of microcode (micro-PC histograms)



Hybrid Monitors

- Combination of software, hardware, or firmware
- Hardware data-gathering + software data-reduction
- Example: Diamond monitor by Hughes (1980)
 - Hardware part can observe all traffic on the system bus
 - Software part can read instruction addresses, processor modes, and system and user identifications
 - Two parts can communicate through device status and control registers



Distributed-System Monitors

- A distributed computer system consists of many hardware and software components
- The monitor for such a system must be distributed and consists of several components as well



Distributed-System Monitors

- **Observation:** gathers raw data
- **Collection:** collects raw data
- **Analysis:** analyzes the data gathered
- **Presentation:** produces reports, displays and alarms
- **Interpretation:** intelligent entity that can make meaningful interpretations of data
- **Console:** interface to control the system parameters and states
- **Management:** the entity that makes decision to set or change system parameters

Figure 7.1



Distributed-System Monitors

- There is a many-to-many relationship between successive layers

Figure 7.2



Two Special Monitors

- **Program execution monitors:** also known as program optimizers, or program execution analyzers
 - To observe application software to help improve the performance of programs
- **Accounting log**
 - Record resources used by the users and their processes. Designed for accounting and billing purposes, also good source of information about resource usage



Program-Execution Monitor

- **Purpose**
 - **Tracing:** to find the execution path of a program, e.g., strace
 - **Timing:** to find the time spent in various modules of the program
 - **Tuning:** to find the most frequent or most time-consuming sections of the code so that the system performance can be improved
 - **Assertion checking:** to verify the relationships assumed among the variables of a program
 - **Coverage analysis:** to determine the adequacy of a test run



Program-Execution Monitor

- Criteria for program selection
 - **Time critical:** to find out where the time is being spent
 - **Frequency of use:** high frequency programs should be optimized first
 - **Percentage of resources:** the programs consuming the highest percentage of resources should be optimized first



Steps in Program Execution Monitoring

Figure 8.1



Issues in Designing a Program-Execution Monitor

35

- All issues in software monitor design apply
- Program execution specific
 1. **Measurement unit:** modules, subroutines, high-level language statements, or machine instruction
 2. **Measurement technique:** tracing (traps) or sampling
 3. **Instrumentation mechanism**
 - Instrumentation added before/during/after compilation or during run time
 - By augmenting the source code, the compiler-generated object code, the run-time environment, the operating system, or the hardware

18 September 2005

EEEC686/785

Wenbing Zhao



Issues in Designing a Program-Execution Monitor

36

- Program execution specific
 4. **Profile report:**
 - Show a frequency and time histogram
 - Summaries by modules, procedures, statements
 - Show self-time and inherited time
 - Allow limiting or expanding the detail (zoom-in or zoom-out)

18 September 2005

EEEC686/785

Wenbing Zhao



Techniques for Improving Program Performance

- Code optimization
 - Finding the dynamic frequency of execution of various program modules
 - Optimizing the code of most frequently used modules
- I/O optimization
 - Combining several I/Os into larger records
 - Changing the file-access method
 - Caching or prefetching data
- Paging optimization
 - Observing the page reference pattern
 - Reorganizing program segments to minimize paging activity



Techniques for Improving Program Performance

- Optimize the common case. The most frequently used path should also be the most efficient. A procedure should handle all cases correctly and common cases efficiently
- Arrange a series of IF statements so that the most likely value is tested first
- Arrange a series of AND conditions so that the condition most likely to file is tested first



Techniques for Improving Program Performance

- Arrange entries in a table so that the most frequently sought values are the first to be compared
- Structure the main line of the program so that it contains the most frequently path of the program. Errors and exceptions should be handled separately
- Question the necessity of each instruction in the main path (time-critical or most frequent) path of the code



Techniques for Improving Program Performance

- Trade memory space for processor time wherever possible. If a function is computed more than once, compute it once and store the result. Some functions with parameters can be replaced by a table of precomputed values
- Use hints. Keeping a fast but possible inaccurate hint along with a slow but robust algorithm may save time in most cases
- Cache the data that is accessed often. However, one must ensure that there is sufficiently locality before using caches



Techniques for Improving Program Performance

- Unroll short loops. Save the cost of modifying and testing loop indexes
- Replace searches by direct indexing, wherever possible. In some cases, this may require more space than minimum
- Use the same size for data fields that need to be compared or added together
- Use the full word widths of the computer to evaluate expressions. For example, use 32 bits on a 32-bit computer even if need only 31



Techniques for Improving Program Performance

- Align data fields on word boundaries, wherever possible
- Evaluate items only when needed, particularly if it is likely that it will not be needed
- Initialize data areas during run time only when used. Wherever possible, the values should be initialized at the compile time
- Use algebraic identities to simplify conditional expressions



Techniques for Improving Program Performance

- Replace threshold tests on monotone (continuously nondecreasing or continuously nonincreasing) functions by tests on their parameters, thereby, avoiding the evaluation of the function
- Evaluate variables not changing in a loop before entering the loop
- Combine two nearby loops if they use the same set of conditions
- Use shifts in place of multiplication and division by powers of two



Techniques for Improving Program Performance

- Keep the code simple. Similar programs are more efficient
- Block I/O request together to reduce the number of I/O operations
- Preload small disk files into tables in memory
- Use multiple buffers for files to allow prefetching
- Link the most frequently used procedures together to maximize the locality of reference



Techniques for Improving Program Performance

- Reference data in the order stored. Arrays stored by columns should be referenced by columns
- Store data elements, that are used concurrently together
- Store subroutines in sequence so that calling and called subroutines will be loaded together
- Align I/O buffers to page boundaries
- Open files, which are used together, in sequence so that buffers will be located together



Techniques for Improving Program Performance

- Pass simple subroutine arguments by value rather than by reference, wherever possible
- Pass large arrays and other data structures to subroutines by reference rather than value
- Separate read-only code areas from read-write data areas to minimize the number of page-writes



Accounting Logs

■ Advantages

- Built into operating systems
 - E.g., in Linux, the syslog can be found in /var/log directory
- The data reflects real-system usage
- Use them before developing a new monitor



Accounting Logs

■ Disadvantages

- No analysis programs. Only utilities to convert them in a readable format are supplied
- Data may not be at the desired level of granularity. For example: distribution of I/O sizes
- Only charged resources may be recorded
- Low accuracy
- No system-level information, such as queue lengths or device utilizations
- Not possible to separate the effects of queueing from user service demands



Data in Accounting Logs

- Resource usage
 - CPU time
 - Elapsed time
 - Number and total size of disk I/O
 - Paging I/Os
 - Terminal I/Os
 - Network I/Os
- Granularity
 - System-wide resource usage
 - Resources used by each program
 - Resources used by each user session



Data in Accounting Logs

- Abnormal conditions such as
 - Security violations
 - Abnormal job terminations
 - System restarts
 - Down times
 - Devices errors



Typical Data Recorded by Accounting Utilities

- Name of the program
Program start time: date and time of the day
Program end time: date and time of the day
CPU time used by the program
- Number of disk writes
Total disk write bytes
Number of disk reads
Total disk read bytes



Typical Data Recorded by Accounting Utilities

- Number of terminal writes
Total terminal write bytes
Number of terminal reads
Total terminal read bytes
- Number of page-read I/Os to the paging device
Number of pages read from the paging device
Number of page faults



Derived Quantities

- **Per activation:** average resource consumption per activation of the program. useful for
 - Modeling program behavior
 - Constructing synthetic workloads
- **Percentage of total:** resource consumed by all activations of a particular program, expressed as a percentage of the resources consumed by all activations of all programs
 - Programs with a high percent have a high impact on the resource
 - These programs should be optimized
 - These programs should be included in test workloads



Derived Quantities

- **Per-second resource-consumption rates:** divide the resource consumption by the elapsed time
 - Indicate the intensity of resource usage (utilization)
 - Help calculate the number of users that can be supported
 - Help determine the number of resources required
 - Help find programs that can cause the resource to become bottleneck



Derived Quantities

- **Per-CPU-second resource consumption:** divide the resource consumption by the CPU time consumed
 - Less variable than per second consumption
 - Gives resource demand per instruction of the program
 - Represents the ratio of the resource demand to CPU demand
 - Programs with high disk I/O per CPU second are disk bound



Using Accounting Logs to Answer Commonly Asked Questions

- *Which programs provide the highest opportunity for better human interface?*
 - The programs with high percentage of total elapsed time (wall-clock time)
- *Which programs are good candidates for code optimization? Or, which programs should be included in a workload to be used in analyzing a new CPUs performance?*
 - The programs with high percentage of total CPU time



Using Accounting Logs to Answer Commonly Asked Questions 57

- *Which programs offer the highest opportunities for code restructuring to minimize the page faults?*
 - The programs with a high percentage of total page faults, a high percentage of total page-read operations, or a high percentage of total pages read
- *Which programs have a poor locality of references?*
 - The programs with high “page faults per-CPU-second”.

18 September 2005

EEEC686/785

Wenbing Zhao



Using Accounting Logs to Answer Commonly Asked Questions 58

- How many jobs can run simultaneously without undue performance degradation?
 - The CPU rate (CPU time per second) for CPU bound programs determines
 - The device with the highest utilization for non-CPU bound programs determines
- Which programs are I/O-bound?
 - The programs with high “disk I/Os per-CPU-second”

18 September 2005

EEEC686/785

Wenbing Zhao



Using Accounting Logs to Answer 59 Commonly Asked Questions

- *What is the average think time?*
 - Approximately = reciprocal of terminal reads per second
- *What workload characteristics should be used in a simulation or analytical model?*
 - To analyze a new device, the programs that use a high percentage of that resource should be included in the workload
 - To compare two different systems the overall average data per workload unit should be used