COMP 190-088: Systems Performance Analysis

Measurement Techniques

Jasleen Kaur

Department of Computer Science
The University of North Carolina at Chapel Hill

Spring 2005

Events and Metrics

- Event-count metrics
  - Simple counts of number of times an event occurs
  - eg, number of page faults, number of disk I/Os
- Secondary-event metrics
  - Record the value of secondary parameters when events occur
  - eg, average number of messages queued in a send buffer
    - Triggering events: message-enque and message-deque
    - Measure: queue size, total number of queue operations
- Measurement Strategies:
  - Event-driven measurements
  - Sampling

Event-driven Measurements

- Record information whenever pre-selected events occur
  - Eg, modify page-fault handling routine to count number of page faults
- Tracing:
  - Record a subset of system state that uniquely identifies the event
  - Requires large amount of storage
- ☺:
  - Overhead of recording information incurred only when events occur
- ☹:
  - Time between measurements is unpredictable

Appropriate for low-frequency events

Sampling

- Records subset of system state at fixed time intervals
- Produces only a statistical summary of overall system behavior
- ☺:
  - Overhead independent of number of times an event occurs
    - Tradeoff between sampling frequency and desired resolution
- ☹:
  - Not every event occurrence gets recorded
  - Each run likely to produce different results
    - Samples taken asynchronously to program execution

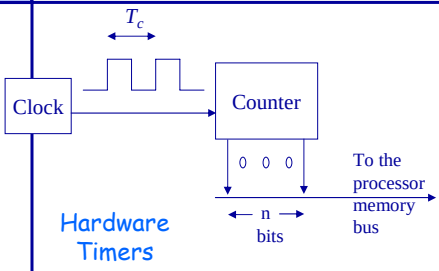
Fundamental Measurement Techniques: Issues

- **Interval Timers**
- Program Profiling
- Tracing

Interval Timers

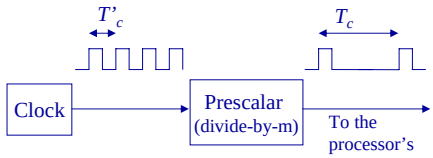
- Count the number of clock pulses that occur between two predefined events
- Uses:
  - Measure the execution time of a code section
  - Provide the time basis for sampling tools
- $T_e = T_c * (x_2 - x_1)$
  - T_e : measured time
  - T_c : clock period
- Can be implemented in hardware or software

Hardware and Software Timers



Hardware Timers

- Count the number of pulses it receives at its clock input
- Programs read memory location mapped to the counter by system manufacturer
- Timer is typically reset to 0 when system is powered up



Software Timers

- Program-readable counter not directly implemented by a clock
- Hardware clock used to generate an interrupt at regular intervals
  - Clock source divided by m through a pre-scaling counter to derive interrupt
  - Counter incremented by interrupt-service routine
- Some systems allow application to reset to 0

Timer Roll-over

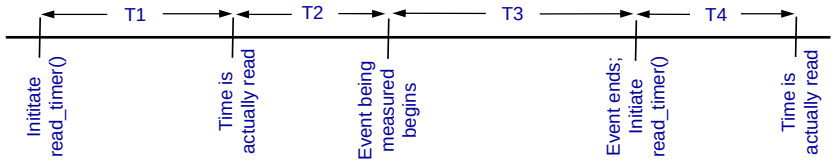
- Longest measurable interval: $(2^n - 1) * Tc$
- If count exceeds this value, (x_2-x_1) is incorrect measure
  - Use $2^n + (x_2-x_1)$ instead

Tc	Counter width, n				
	16	24	32	48	64
10 ns	655 us	168 ms	42.9 s	32.6 days	58.5 centuries
100 ns	6.55 ms	1.68 s	7.16 min	326 days	585 centuries
1 us	65.5 ms	16.8 s	1.19 h	9.15 years	5,850 centuries
10 us	655 ms	2.8 min	11.9 h	89.3 years	58,500 centuries
100 us	6.55 s	28 min	4.97 days	893 years	585,000 centuries
1 ms	1.09 min	4.66 h	49.7 days	89.3 centuries	5,850,000 centuries

A few bits can help a lot!

Timer Overhead

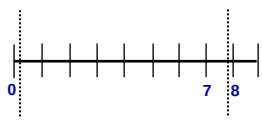
```
t_start = read_timer()
<event being timed>
t_end = read_timer()
elapsed_time = (t_end – t_start)*t_cycle
```



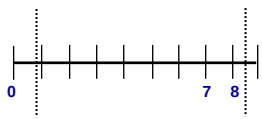
- Timer overhead: Tovhd
  - Event time = Te = T3
  - Measured time = T2 + T3 + T1 = Te + Tovhd
  - If Tovhd << Te, it can be ignored, else should be measured and subtracted
- ☹:
  - Variations in Tovhd can be comparable to those in Te
  - Te/Tovhd > 100

Quantization Errors

- Resolution: smallest change that can be detected and displayed by an interval timer
  - Typically, equal to a single clock tick, Tc
- Introduces a random quantization error into all measurements
  - Unpredictable and random



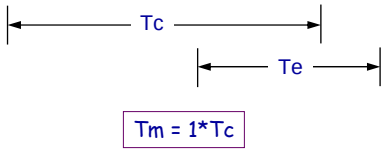
Timer reports: Tm = 7*Tc



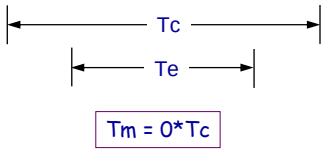
Timer reports: Tm = 8*Tc

- Actual event time is within n*Tc < Te < (n+1)*Tc
  - If Te is smaller than Tc, impossible to directly measure Te
- Tradeoff between resolution and roll-over frequency

Statistical Measures of Short Intervals



Tm = 1*Tc



Tm = 0*Tc

- Measurement is a Bernoulli experiment
  - Outcome is 1*Tc with probability p
  - Outcome is 0*Tc with probability 1-p
- Repeating experiment n times yield a binomial distribution
  - As long as the n measurements are **independent**
- If number of "1" outcomes is m, then m/n ~ Te/Tc
  - Te = m*Tc/n

Fundamental Measurement Techniques: Issues

- Interval Timers
- **Program Profiling**
- Tracing

Program Profiling

- Profile:
  - Measurement of the time spent by system in certain states
- Useful in identifying bottlenecks
  - In program code
  - In accessing system resources
- Examples of profiling techniques
  - PC Sampling
  - Basic-block Counting

PC Sampling

- Executing program is sampled at fixed points in time
  - External periodic signal interrupts program at fixed intervals
  - State information recorded by interrupt-servicing routine
    - Check return-address stack to find address of instruction
    - Use compiler's symbol-table information to map instruction address to a subroutine identifier
    - Increment counter for that subroutine
- Accuracy of statistical inference can be improved by:
  - Collecting more samples
    - Sampling for longer durations
    - Increasing the sampling rate
  - Ensuring interrupts occur asynchronously with sampled events
    - Ensures independence of random samples

Basic-block Counting

- Basic block:
  - Sequence of instructions with no branches into or out of the sequence
    - If first instruction is executed, all of the rest are as well
- Program can be profiled by inserting additional instructions at the beginning of every basic block
  - Profiling resolution is in terms of basic-blocks (not subroutines)
- ☺:
  - Yields an **exact** execution frequency histogram
  - Repeatable
- ☹:
  - Can incur significant run-time overhead
    - Access array of counters for current block's counter
    - Basic blocks typically have 3-20 instructions: 100% overhead!
  - Additional memory required to store counter array
  - Both of the above may alter program behavior

Fundamental Measurement Techniques: Issues

- Interval Timers
- Program Profiling
- **Tracing**

Event Tracing

- Profiling ignores the time-ordering of events
  - Use program trace instead
- Trace:
  - A dynamic list of events generated by the executing program
    - Instructions executed
    - Memory addresses accessed
    - Disk blocks referenced
  - Can be analyzed to characterize overall program behavior
  - Typically, used as input to drive a simulator

Trace Generation

- Source-code modification
  - Add tracing statements to the source code
  - ☺: can reduce trace volume by tracing only desired events
  - ☹: time-consuming, error-prone
- Software Exceptions
  - Processor mode that forces a software exception just before each instruction execution
    - T-bit in DEC's VAX, Motorola 68000
  - ☹: slowed down programs by up to 1000 times
- Emulation
  - Emulation of a system on a completely different system
    - Eg, Java Virtual Machine
  - Modify emulator to output trace of instructions
  - ☹: emulation is slower than direct execution
- Compiler modification
  - Compiler adds extra instructions at entry point of each basic clock

```
sum_x = 0.0
trace(1);
for (i=1; I <=n; i++)
trace(2);
        {
        sum_x += x[i];
        trace(3);
        }
mean = sum_x / n;
trace(4);
```

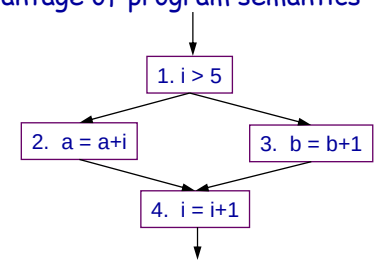
Trace Compression

- Tracing overheads:
  - Execution time slow down
  - Tremendous volume of data produced in a short time (~GB/min)
    - Large disk space needed
    - Disk I/O interferes with program execution
- Remedies for reducing amount of data:
  - Online trace consumption
    - Lack of repeatability
  - Data compression
    - Additional time needed to compress and decompress
  - Trace sampling
    - Not clear what is the right sampling technique, frequency, …
  - Abstract execution

Abstract Execution

- Trace compression that takes advantage of program semantics

```
1.  if ( i < 5 )
2.      then a = a + i;
3.      else b = b+1;
4.  i = i + 1;
```



- Two-step tracing process:
  - Trace-generation:
    - Program analysis to identify a subset of trace that can reproduce the full trace
  - Trace-consumption:
    - Execution of trace-generation routines to retrieve full trace
- ☺:
  - Reduces trace size by 10 to 100 times
  - Slows down program being traced by 2-10 times
    - Comparable to other tracing techniques

Measuring System Load Using an Idle Loop

- Estimate:
  - Number of processes running on the system
- Use:
  - A program that counts up from zero
- Idea:
  - Run for a fixed amount of time on unloaded system
    - Let value of count be n
  - Run on loaded system
    - If one other process, count should be n/2
    - If m other processes, count should be n/(m+1)

Perturbations Due to Measurements

- Measurements impact system performance
  - Code can alter the spatial and temporal patterns of memory access
    - Cache may be flushed more often
    - Trace instructions could increase cache-hit rate
    - Cache performance impacted in an **unpredictable** manner
  - Increase in code size can result in larger overall context-switches
    - Can significantly alter program's paging behavior
- Trends:
  - Higher is the granularity of measurement, more is the perturbation
  - Perturbations are non-linear and non-additive
    - Doubling the amount of instrumentation need not double the impact on performance

Impact of perturbations on conclusions should be analyzed

Course Outline

- Selection of metrics
- Performance Evaluation Methodologies
- Workload selection
- Measurements tools
- **Analysis and visualization of measured data**
- System Modeling
- Simulations
- Case studies
- Distributed monitoring infrastructures
- PA in the Research and Industrial communities