Performance Measurement

Introduction

Performance Measurement

- Measure the computer system's hardware or software states
- May involve monitoring the system while it is being subjected to a particular workload

Purposes

- System understanding
- Model development (including refinement of the model)Model verification and validation
- System performance evaluation

"The purpose of measurement is insight, not numbers" –Hamming

Methodology

- Understand the objective of the measurement studies
- Identify the performance measures or data needed to achieve the objective
- Identify the input parameters and their characteristics (e.g. workload characterization)
- Select appropriate measurement strategies or tools
- Design experiments Run the experiments and collect data
- Analyze, interpret, and present the results

Performance Metrics

- Should be relevant to the objective of performance measurement
- Characteristics of a good performance metric
  - Linearity
  - Reliability
  - Repeatability
  - ease of measurement
  - Consistency
  - independence

Performance Metrics

- For a software, may be interested in
  - path characteristics: the number of times each significant path is executed to compute the loop repetitions, and the execution probabilities for conditional path
  - software resource usage: the number of times each resource is requested, and the corresponding elapsed time
  - processing overhead: the amount of service the software require from each of the key system resources (such as CPU, disk and network)

Measurement Strategies: Where?

- Internal vs. External
  - internal: integrate the measurement codes into the software to directly detect events and record performance metrics
  - external: conduct the measurement independently of the execution of the software

Measurement Strategies: When?

- Event-driven
  - whenever the selected event or events occur, record the information (which is needed to calculate the performance metrics)
  - e.g., page fault, read/write to the disks, cache hit
  - system overhead is incurred only when the event occurs; therefore, this strategy is suitable for low-frequency events

Measurement Strategies

- Tracing
  - rather than counting the occurrence of an event (as in event-driven strategy), portion of the system states is recorded to uniquely identify the event
  - e.g., addresses that causes page faults are recorded
  - provide more detailed information
  - nevertheless, require substantial storage space and additional processing capacity

Measurement Strategies

- Sampling
  - at fixed time intervals, record the system states which are used to determine the performance metrics
  - provide a statistical summary of overall behavior of the system (where the event-driven and tracing strategies give exact number of an event's occurrence)
  - not suitable for low-frequency events
  - system overhead is a function of the length of time intervals or the sampling frequency(instead of the number of events occurred)

Performance Monitors

- A measurement tool to observe activities or events of a system and measure the perform
- Fundamental concepts
  - input rate: maximum frequency of events that can be correctly observed
  - resolution: the coarseness of information observed
  - overhead: how much is monitoring costing you?
  - perturbence: how much does the system behavior change due to monitoring?
- Types:
  - Hardware monitors
  - Software monitors

Hardware Monitors

- Used to measure the state of hardware components of the system, such as registers, memory locations, and I/O channels
- Separate piece of equipment attached to circuitry via electronic probes

Hardware Monitors

- Advantages:
  - high input rate
  - low overhead
  - generally portable: independent of the operating system
- Disadvantages:
  - no access to high-level information
  - little support for abstractions

Software Monitors

- Used to monitor operating system and higher level software, e.g., transport service, database
- Event-driven
  - instructions are executed (to collect data) each time an event occurs, e.g., start or end of a given service routine
- Sampling
  - timer interrupt is used; control is transferred to data collection routine at pre-specified interval

Examples of Software Monitors

- HTTPerf (Event-driven)
  - tool for measuring web server performance
  - act as a client and generate HTTP requests to the server
  - evaluate different metrics such as response time, connection time, and the number of errors
  - allow generation of different workloads
- Gprof (Sampling)
  - Profiling tool: provide an overall view of the execution behavior of a program
  - estimate how much time is spent in each module or component

Examples of Software Monitors

- Other monitors available on Unix
  - iostat: I/O subsystem
  - netstat: network subsystem
  - vmstat: virtual memory subsystem
  - trace/truss: track the system calls made by a process
  - tcpdump: record events from network
  - top: display information about the top processes running on the CPU

Writing Your Own Monitor

- Three reasons for instrumenting software (or why the provided tools may not be enough)
  - convenience: no tool generates one report with exactly those data that you need
  - data granularity (i.e., resolution): standard measurement tools rarely match your requirements
  - control: rarely need all the data all the time

Writing Your Own Monitor

- Design considerations
  - define the events to be measured, e.g., the start of a function
  - choose the granularity of the measurements
  - can you dynamically select the events to be recorded?
    - volume of information collected
    - impact of measurement on running system (i.e., the overhead)