CS 147: Computer Systems Performance Analysis
Measurement Tools
Overview

Monitors
  Types of Monitors
  Design Issues

Tools and Methods
  Instrumentation
  Tracing Packages
  System Metrics
A monitor is a tool used to observe system activity
- Proper use of monitors is key to performance analysis
- Also useful for other system observation purposes
Classifications of Monitors

- Hardware vs. software
- Event-driven vs. sampling
- On-line vs. batch
Hardware vs. Software Monitors

- Hardware monitors used primarily by hardware designers
  - Requires substantial knowledge of hardware details
  - VLSI limits monitoring possibilities
- Software monitors used (mostly) by everyone else
  - Exception: power measurement
Event-Driven vs. Sampling Monitors

- Event-driven monitors notice every time a particular type of event occurs
  - Ideal for rare events
  - Require low per-invocation overheads
- Sampling monitors check system state periodically
  - Good for frequent events
  - Can afford higher overheads
Online vs. Batch Monitors

- Online monitors can display their information continuously
  - Or at least frequently
- Batch monitors save it for later
  - Usually have separate analysis procedures
Issues in Monitor Design

- Activation mechanism
- Buffer issues
- Data compression/analysis
- Enabling/disabling monitors
- Priority issues
- Distributed monitoring
- Abnormal events monitoring
Activation Mechanism

When do you collect the data?

- When an interesting event occurs, trap to data collection routine
- Analyze every step taken by system
- Go to data collection routine when timer expires
Buffer Issues

- Buffer size
  - Big enough to avoid frequent disk writes
  - Small enough to make disk writes cheap

- Number of buffers
  - At least two, typically
  - One to fill up, one to record

- Buffer overflow
  - Overwrite old data you haven’t recorded
  - Or lose new data you don’t have room for
  - In either case, count what’s lost

- Sometimes can wait for buffer to empty
Data Compression or Analysis

- Data can be literally compressed
- Or can be reduced to a summary form
- Both methods save space for holding data
- At cost of extra overhead in gathering it
- Sometimes can use idle time to compress
  - But maybe better spent dumping data to disk
- Space may be limit on what you can gather
Enabling/Disabling Monitors

- Most system monitors have some overhead
- Need to turn them off if high performance required
  - Unless overhead is trivial
  - Or if primary system purpose is gathering data
    - As with many research systems
Monitor Priority

- How high a priority for monitor’s operations?
- Trade off performance impact against timely & complete data gathering
- Not always simple question
Monitoring Abnormal Events

- Often, knowing about failures and errors more important than knowing about normal operation
- Sometimes requires special attention
  - System may not be operating very well at time of failure!
Monitoring distributed system is similar to designing one

Must deal with

- Distributed state
- Unsynchronized clocks
- Partial failures
Viewing a Distributed Monitor in Layers

**Management**
- Make system changes, as necessary

**Console**
- Control overall system

**Interpretation**
- Decide what results mean

**Presentation**
- Present your results

**Analysis**
- Analyze what you’ve stored

**Collection**
- Store what you’ve seen for later

**Observation**
- Watch what happens
Observation Layer

- Layer that actually gathers data
- *Implicit spying*—watching what other sites do without disturbing the activity
- *Explicit instrumentation*—inserting code to monitor activities
- *Probing*—making feeler requests into system to discover what’s happening
Collection Layer

- Data can be collected at one or several points in distributed system
- How does data get from observer to collector (if not collocated)?
  - Advertising—observers send it out, collectors listen and grab it
  - Soliciting—collectors ask observers to send it
- Clock issues can be key
- In distributed system, may be more feasible to analyze on the fly
- Can sometimes dedicate one (or more) machines to analysis
- But often requires gathering all data to one point
OK, so how do I actually measure a piece of software?
What practical tools and methods are available to me?
How do I get my project done?
Tools For Software Measurement

- Code instrumentation
- Tracing packages
- System-provided metrics and utilities
- Profiling
Code Instrumentation

- Adding monitoring code to system under study
- Basically, just add code that does what you want
Advantages and Disadvantages of Code Instrumentation

+ Usually most direct way to gather data
+ Complete flexibility in where to insert monitoring code
+ Strong control over costs of monitoring
+ Resulting measurements always available
  – Requires access to source
  – Requires strong knowledge of design and details of code
  – Requires recompilation to change monitoring facility
  – If overdone, strong potential to affect performance

− Requires access to source
− Requires strong knowledge of design and details of code
− Requires recompilation to change monitoring facility
− If overdone, strong potential to affect performance
Typical Types of Instrumentation

- Counters
  - Cheap and fast
  - Low level of detail

- Logs
  - More detail
  - More costly
  - Require occasional dumping or digesting

- Timers
  - To determine elapsed time for operations
  - Typically using OS-provided system calls
Counters

- Useful only if number of times an event occurs is of interest
- Can be used to accumulate totals
- In modern systems, make them wide enough to not overflow (64-bit is good)
Counter Examples

- Number of times a network protocol transmits packets
- Number of times programs are swapped out due to exceeding time slices
- Number of incoming requests to Web server
Logs

- Can log arbitrarily complex data about an event
- But more complex data takes more space
- Typically, log data into reserved buffer
- When full, ask that buffer be written to disk
  - Often want second buffer to gather data while awaiting disk write
Designing a Log Entry

- What form should a log entry take?
  - Binary is compact but fragile
  - Text is human-readable, robust, bulky
  - Always consider ease of parsing

- Easy to post-format for printing
  - Useful for system debugging
  - Make sure no important information is lost in compacting log entry

- *Always* include a version stamp

- Also collect metadata (machine collected on, configuration, etc.)
Many OSes provide system calls that start and stop timers
  - Allows measuring how long things took

- Usually, only elapsed time measurable
  - Not necessarily time spent running particular process

- Care required to capture real meaning of timings
Tracing Packages

- Allow dynamic monitoring of code that doesn’t have built-in monitors
- Basically, augment code to call monitoring routines when desired
- Akin to debuggers
- Typically allow counters and some forms of logging
Advantages and Disadvantages of Tracing Packages

+ Allow pretty arbitrary insertion of monitoring code
+ Don’t need recompilation to instrument code
+ Tremendous flexibility at measurement time
+ No instrumentation overhead when you’re not using it
  – Somewhat higher overheads than building instrumentation into code
  – Usually requires access to source for effective use
  – Usually requires deep understanding of code internals
  – Only produces data when special package used
  – Usually specific to particular systems

− Somewhat higher overheads than building instrumentation into code
− Usually requires access to source for effective use
− Usually requires deep understanding of code internals
− Only produces data when special package used
− Usually specific to particular systems
How Do Tracing Packages Work?

Much like debuggers:

- Attach to running programs
- Use commands in tracing packages to associate data gathering with particular points in the programs
- Replace normal code at that point in program with calls to data-gathering code
Many operating systems provide users access to some metrics

Most operating systems also keep some form of accounting logs

Lots of information can be gathered this way
What a Typical System Provides

- Timing tools
- Process-state tools
- System-state tools
- OS accounting logs
- Logs for important system programs
Timing Tools

- Tools that time execution of a process
- Several different times often provided
- E.g., Unix time command gives system, user, and elapsed time
- Some components of times provided may depend on other system activities
  - Just calling time on a command may not tell the whole story
Many systems have ways for users to learn state of their processes

Typically provide information about

- Time spent running process so far
- Process size (virtual/real)
- Status (running, waiting for I/O, etc.)
- Priority
- I/O history
Using Process-State Tools

- Typically can’t monitor process state continuously
  - Updates not provided every time things change
- Can get snapshots on demand
  - Most useful for sampling monitors
System-State Tools

- Many systems allow some users to examine internal state
  - Virtual memory statistics
  - Length of various queues
  - I/O rates
- May be available only to privileged users
- Typically, understanding state requires substantial expertise
- Often useful only for specific purposes
Many operating systems maintain logs of significant events
Based on either event-driven or sampling monitors
Examples:
  - Logins
  - Quota violations
  - Program executions
  - Device failures
Often, non-OS systems programs keep logs
  - Mail software
  - Web servers

Usually only useful for monitoring those programs
But sometimes can provide indirect information
  - E.g., notice of failure to open connection to name server may indicate network failure