Operating Systems, Concurrency and Time

performance analysis

Johan Lukkien
Questions

• What is software performance?
• Which performance *metrics* are relevant?
• How can these metrics be
  – computed
  – predicted
• What is the relation between these metrics and
  – the execution architecture
  – the system software (OS)
  – the application software organization
    • the mapping on the platform (platform = hardware + OS)
• Which hazards and common pitfalls exist?
• What controls are required for managing performance of an application, and what is actually available from an OS?
Questions

• How can performance be improved?
  – what are bottlenecks and how to find them?
  – which heuristics help?

• Which role does concurrency play?
  – what are sources of concurrency?
  – what are positive and negative effects?

• Are there structured methods to arrive at a realization with sufficient performance, or to deal with performance at all?
  – how to take performance into design and validation?
  – what would be the ideal development process, and how is it done now?

• How to analyze performance?
  – which experiments and which tools?
  – when to evaluate?
  – how to judge?
Performance and metrics

• Performance of a system (or software) $S$ refers to properties that relate an execution of $S$ to a quality. Considered qualities for now are time related.

• Typical (high-level, system-level) metrics:
  – latency: the time that elapses from the first stimulus of a task to the beginning of the observed response
  – computation (execution) time: the actual time spent on a task
  – turnaround: time that elapses from start to completion of a task
  – throughput: number of complete task executions per time unit
    • effective throughput: measured for the system at hand
    • characteristic: under fully loaded conditions
    • determined over a fixed time window $W$: $\frac{\text{#executions in } W}{W}$
  – jitter: worst case spread
    • e.g. latency: (maximum) difference in completion times of adjacent task executions
Measurements: what to record?

- **typically**: (serial) traces of timestamped events, events counts
- **derive**: the mentioned metrics, durations, utilizations
- **derive overview**: what is this program or system doing?
Measurements: where from?

• Performance Measurements comprise:
  – details collected by the hardware
    • e.g. Event Counters, maintained by hardware components
  – details collected by the OS
    • e.g. Event Tracing for Windows, ETW
  – details collected by the user
    • instrumented code (e.g. by compiler, or by programmer)
    • explicit timing measurements (e.g. using OS primitives for timing fragments)
Measurements: which tools?

- Properties of tools
  - resolution: (max) frequency of measurements
  - accuracy: deviation of the real value
  - granularity: (min) code block that can be measured

- Examples:
  - WMI: Windows Management Instrumentation
  - RDTSC instruction (ASM): returns #instructions since processor start
    - read time stamp counter (e.g. before and after call)
    - mind scheduling anomalies while using this
  - Windows Performance Recorder / Windows Performance Analyzer
    - records and displays information from Event Tracing for Windows

<table>
<thead>
<tr>
<th>Method</th>
<th>Typical Resolution</th>
<th>Typical Accuracy</th>
<th>Granularity</th>
<th>Difficulty of Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop-watch</td>
<td>0.01 sec</td>
<td>0.5 sec</td>
<td>program</td>
<td>easy</td>
</tr>
<tr>
<td>date</td>
<td>0.02 sec</td>
<td>0.2 sec</td>
<td>program</td>
<td>easy</td>
</tr>
<tr>
<td>time</td>
<td>0.02 sec</td>
<td>0.2 sec</td>
<td>program</td>
<td>easy</td>
</tr>
<tr>
<td>prof and gprof</td>
<td>10 msec</td>
<td>20 msec</td>
<td>subroutines</td>
<td>moderate</td>
</tr>
<tr>
<td>clock()</td>
<td>15-30 msec</td>
<td>15-30 msec</td>
<td>statement</td>
<td>moderate</td>
</tr>
<tr>
<td>software analyzers</td>
<td>10 μsec</td>
<td>20 μsec</td>
<td>subroutine</td>
<td>moderate</td>
</tr>
<tr>
<td>timer/counter chips</td>
<td>0.5-4 μsec</td>
<td>1-8 μsec</td>
<td>statement</td>
<td>very hard</td>
</tr>
<tr>
<td>logic or bus analyzer</td>
<td>50 nsec</td>
<td>half μsec</td>
<td>statement</td>
<td>hard</td>
</tr>
</tbody>
</table>

Table by David Stewart, dr. Dobb collected mainly for Unix.
Examples

static __inline__ unsigned long long rdtsc(void)
{
    unsigned hi, lo;
    __asm__ __volatile__ ("rdtsc": "=a"(lo), "=d"(hi));
    return ((unsigned long long)lo)|( ((unsigned long long)hi)<<32 );
}

RDTSC: return 64bit counter

WPA/WPR view
Measurements: what do you want to know?

- The metrics for some task in varying (perhaps uncontrolled) circumstances
  - repeated measurements, possibly with different inputs
  - assume measurements are outcomes of a chance process

- The metrics for some task in function of some parameter
  - let inputs cover the parameter space systematically

- The metric of computation time (real work) of some task
  - this is – again – input dependent: need to cover the input space
  - use the minimum measurement as an estimation for each fixed input
  - need to understand platform operation (e.g. concurrency) for this purpose
    - add up contributions from different processors
Performance analysis cycle

PLATFORM

Architecture instances (= Machines)

OS

Network

Application

Mapping, configuration, instrumentation

Executable System

Execution and monitoring

Specifications History

Performance Measurements (event logs, traces with timing)

Analysis

Diagnostics, high level metrics

Adaptation

History
Restrict variations

• Usually, studied variations are limited
  – *application analysis*: fix the platform, vary application, mapping
  – *explore new hardware*: fix application, OS and network, vary hardware
  – *examine new OS*: fix hardware, application and network, vary OS

• In all cases: parameters / settings / mappings can be adjusted
  – e.g. *#processors*, binding, input to the application
Measurements: how to setup?

• Add manually, detailed, targeted measurements to the code
  – call a recording, timestamping function upon start/end of a task or event

• Instrument the code mechanically: typically a compiler option
  – sample the call stack regularly – fixed % overhead, trading accuracy for overhead
  – sample the current instruction – no need for changing the code. OS function, same tradeoff
  – add code to automatically record certain events

• Invoke OS-level tracing (if the OS supports)
• $t_2-t_1 + t_4-t_3$ must be $<< t_3-t_2$

• Alternatively,
  – determine these two overheads independently
  – or perform task sufficiently often to obtain good averaging
    • use variations in repetition to solve for the overhead

• Whatever is measured should be larger than the resolution of the used time tool
  – again, perform task sufficiently often

• The whole measurement must be repeated often enough to obtain insight in overheads caused by the OS interference
Measurement: anomalies

• OS behavior
  – moving tasks to different processors
  – interrupting execution
  – interfering applications

• *The uncertainty principle*: measuring leads to modified behavior

• Resolution of the timing tool may be too low

• Make sure the recording functions *do not delay*
  – log in memory, write to disk *afterwards*
  – System calls may behave unpredictably - mind implicit locks
Difficult points

• OS behavior (internal decisions) under changes in hardware and application
  – which aspects of mapping and configuration are relevant to manually control?
    • priority of processes/threads
    • binding of process to processor (affinity)
    • explicit memory allocation

• Effects of changes / upgrades in OS
  – what is relevant to consider, what do we need to look out for?

• Interpretation of measurements
  – what are relationships between ‘set-points’ that can be adapted, and outcomes of measurements?
  – what is the relation between measurements and metrics?
  – can we understand system behavior?

• Running the complete system
  – can we avoid this and obtain information from more limited setups?
  – can we compose partial measurements?
Statistics

• Input to analysis is a series of measurements
  – collect sufficient data, trying just a few times does not give insight

• Look at the data using some basic statistics
  – pdf (density function, histogram), boxplot (4 quartiles, min, max), average + standard deviation, CDF (cumulative distribution function)
    • rescale [min-max] → [0,1] to obtain probabilities

• Assume measurements are outcomes of a chance process
  – state hypotheses and perform statistical tests to find the distribution

• Parameterized
  – propose a model and fit (e.g. interpolation, extrapolation)
Uniform (2, 8)
Normal (4,1)
(clipped in [2,8])
Longer tail
Exercise P.1


- Download a trace file from http://www.win.tue.nl/~johanl/educ/OSRTWS/ORACLE17.06-11-2016.21-52-29.etl

- Load the trace file in the WPA analyzer.

- Answer the following questions
  - Examine the (precise) CPU usage. Which processes execute more or less periodically? What is their period, roughly? And how many threads are running?
  - On which processors do they run?
  - What is the utilization of the processors? Which of the four are actually hyperthreaded copies?
  - How many interrupts/sec occur?

- Do the same with a trace obtained from your own laptop running your application.