

T-106.290 Ohjelmistotekniikan laboriotyöt
2–5 ov

Kenneth Oksanen

`cessu@cs.hut.fi`

20. Feb 2004

Technology Evaluations

- “Shall we use Oracle, MS Access, Postgress, or MySQL for our customer database?”
- “Shall we write our FPS game in C, C++, Java, .NET, Perl, Python, Scheme, ...?”
- “Which data structure should we use to store IP routing information?”
- “How should we store pixel values in our photo manipulation program?”
- “Should we use TCP, UDP, XML-RPC, Jabber, Beep, ...?”
- Beyond this course: “Shall we invest in this company?”

- Given a technology, or a set of alternative technologies, produce and weigh the evidence for their suitability for a defined purpose.
- In our case, technology might mean:
 - A software system: RDBMS, compiler, subroutine library, ...
 - A data representation: protocol, schema, data structure, programming language, ...
 - An architecture: a set of above, an API, a programming paradigm, ...
 - An algorithm

- The “evidence” may be:
 - Quantitative: CPU and memory usage, cost, ...
 - Qualitative: list of features, ease of use, support, compatibility with or reuse of existing software, ...
 - Measurements of CPU, memory, disk, bandwidth and roundtrip usage,
 - asymptotic analysis of algorithms,
 - calculations/estimates of cost,
 - compatibility tests, ...
- Technology evaluations are often *situated*
- In this course, we shall emphasize measurements, but do not forget about the rest.

Experimental Algorithmics

- Typical problems with mathematical (worst/average/amortized) asymptotic analysis of algorithms:
 - difficult
 - must make many simplifying assumptions, such as
 - * count only number of comparisons in sorting algorithms
 - * ignore caching effects in accessing data from main memory
 - assumes unboundedly large inputs, does not tell everything relevant about small inputs
 - * Computing folklore says selection sort is fastest for $N < \sim 10$
 - Omits constant factors and terms
 - \Rightarrow almost all balanced trees have $O(\log N)$ search and update behavior

- Typical problems with experimental comparison of algorithms:
 - Very situated, depends on
 - * implementation details,
 - * programming language, compiler, libraries, operating system,
 - * hardware (CPU, motherboard, memory, disks, ...)
 - ⇒ Are the results applicable to any other environment?
 - Must have representative input
 - Must cope with measurement errors, other processes in the machine, and the cost of generating input and collecting results
 - Results may not be extrapolatable to larger inputs

- When comparing algorithms, one is usually interested in their most efficient implementation and environment achievable, for example:
 - C/C++/FORTRAN, with macros/inline functions, best compiler with highest optimization
- When comparing software systems, one is usually interested in their most typical use, for example:
 - most idiomatic programming style in a programming language,
 - standard-conforming SQL for a RDBMS, vendor-specific extensions only as additional information

Generating Input

Assume we are evaluating with index structures for integer-valued keys.

- Theoretical input, such as uniformly distributed integer values $\in [0 \dots 2^{32} - 1]$.
 - Not very natural
 - May miss worst cases, such as lack of balancing of a binary tree
- Real input, such as student book numbers from an access trace to WWW-Topi.
 - Limited feed of input
 - Applicability to other inputs, e.g. student book numbers $< 2^{16}$ and a 65536-entry array would suffice.

- Simulated input, for example:
 1. Pick an integer $\in [0 \dots 2^{32} - 1]$ according to some naturally occurring skewed distribution.
 2. Apply a random bijection $2^{32} \leftrightarrow 2^{32}$ to the integer
 - The skewedness of the distribution is a useful parameter for characterizing the input
 - Problem: generating good input has become CPU-intensive
 - Some useful routines will appear on the course's WWW-pages

Measuring

- Real time, system time and CPU time of a process can be queried from the operating system
 - `times(2)` or `times(3)` in UNIX
- Special instructions give some hardware-specific counter values
 - `rdtsc` reads clock cycle counter in x86
 - `mfspr` reads bus cycle counter /4 in in PowerPC.
- Some useful routines will appear on the course's WWW-pages

Measurement Errors

- Clock ticks not brief enough to give accurate results
- ⇒ Run many benchmark cases so that many (e.g. > 100) ticks pass, average over number of test runs.

- Competing processes steal CPU-time and pollute the cache
- ⇒
 - Choose a lightly loaded machine
 - Competing processes always *slow down* the process being measured ⇒ repeatedly repeat e.g. 20% of the worst measurements.

- Generating input interferes with measurements
- ⇒ Precompute and tabulate some input (say, 10^5 cases) before running the benchmark

Presenting Results

- Tables, histograms, 1-dimensional graphs, 2-dimensional surfaces
- Linear and logarithmic scales
- Giving confidence intervals may be justified
- Typical problem: too many parameters and measurements to show them all comfortably.

Discussing Results

“The purpose of computing is insight, not numbers.”

– R. W. Hamming

- Your report should not merely give numbers and beautiful pictures, it should also discuss the results:
 - Which technology/algorithm was found best/worst? Why?
 - What general trends in input affected the (relative) performance? Why?
 - Do your results question or support previous work, conventional wisdom, marketing materials, and mathematical analyses? Why?
 - Digest and chrystallize the results in a few sentences, included already in the abstract.

- Discuss also the reliability and generality of your results (be cautious!)
- What implementation details were found to affect performance?
Why?

Next Steps

- Begin to form groups
 - Topics should appear later this week, but do not enroll yet!
- ⇒ Follow the WWW-page
- Next weeks lecture:
 - Brief review of topics
 - An illustrative case study of experimenting an algorithm
 - The structure of a final report