Experimental Approaches in Computer Science

Dror Feitelson
Hebrew University

Lecture 3 – Measurements
Assume we start taking measurements of something:
Such measurements are limited by the resolution of our measurement apparatus: the smallest difference between measurements.

Example: measuring time in milliseconds.
Another problem is the **uncertainty** in the measurements: if we repeat them, we get somewhat different results.

This is caused by **random errors** that reflects the (im)precision of the measurement.
A third issue is the accuracy of the measurement: how far it is from the “real” value. This reflects the systematic error in the measurement.
Systematic errors are systematic – they always have the same effect

Example: when measuring the time to perform an action, the overhead of the measurement itself is added to the result

Part of the design should be to identify systematic errors and factor them out

A special problem is warmup or hysteresis-type errors, where the outcome depends on history; for example, the first measurement could be different from subsequent ones
Random errors are random – they may have different effects

Example: when measuring the time to perform an action, the result may depend on cache state, interrupts from the network, and competition from other processes.

This can be analyzed statistically.

In extreme cases, interference leads to outliers that should be ignored.
Model of random errors:
Assume the error is a combination of multiple effects, each contributing $\pm \varepsilon$
If we have $n$ such effects, the outcome will have a binomial distribution

$$p(X + k \varepsilon) = \left( \frac{n}{n+k} \right) \frac{1}{2^n}$$

For large $n$, this can be approximated by a normal distribution

Calculation of confidence intervals is based on assumption of a normal distribution
Summary:

- **Inaccuracy** due to systematic errors
- **Resolution:** the smallest difference we can measure
- **Uncertainty** due to random errors
- **Outlier** due to interference

The REAL value
Measuring Time
A computer (specifically, Intel PC) has three time sources:

Timer interrupts

Time stamp counter

Time server and NTP protocol
Timer interrupts:
Cause a CPU interrupt, and running the clock interrupt handler
Come at a certain frequency (e.g. 250 Hz) from an external timer
The system counts how many interrupts (called ticks or jiffies) have occurred
Reasonably accurate measure of wallclock time
Typical resolution of several milliseconds (maximal resolution is microseconds)
Time stamp counter:
A special register that counts the cycles since the machine was booted
Can be read by a special assembler instruction
Rate depends on the CPU clock rate
Can drift e.g. when temperature rises
Can change due to power considerations, especially on laptops (reduce speed to save energy)
Nanosecond (cycle) resolution, but need many cycles to take a reading
NTP (Network Time Protocol)

Most networks have a designated NTP server

The NTP server gets time from some standard source

All nodes in the network synchronize with the NTP server

Effect is to update the system time

Can lead to a jump in time

Jump can also be backwards

Alternative is to change the time gradually
Linux gettimeofday()
Upon each clock interrupt
Update notion of real time as per the external source
Synchronize with the time stamp counter
When called, read the current time stamp counter and extrapolate from the previous clock interrupt
Combines different timing sources
Report result in microsecond resolution (API)
Making a Measurement
The framework:
Measurement of some computer activity or operation
Done from user level
With no specialized tools
A simple measurement:

```c
    t1 = gettimeofday();
    operation being measured
    t2 = gettimeofday();
    print "time was ", t2 - t1, "\n";
```

Potential problems:

Inaccuracy due to measurement overhead
measurement overhead:

gettimeofday() measured operation gettimeofday()

measured value includes operation + one gettimeofday()

Thus one gettimeofday() should be subtracted
A simple measurement:

```c
    t1 = gettimeofday();
    operation being measured
    t2 = gettimeofday();
    print "time was ", t2 – t1, "\n";
```

Potential problems:
Inaccuracy due to measurement overhead
Operation is shorter than our measurement resolution (get \( t1 = t2 \) so \( \text{time} = 0 \))
Operation is short relative to random errors (so actually get a random time)
Resolution problem: printouts will be 0 or 1
Random errors: printouts will fluctuate wildly
But in both cases the average (for large enough N) should be good

Solution 1: iterate making multiple measurements

```c
for (i=0 ; i<N ; i++) {
    t1 = gettimeofday();
    operation being measured
    t2 = gettimeofday();
    print "time was ", t2 – t1, "\n";
}
```
Solution 1a: buffer the results to reduce measurement interference

```c
for (i=0 ; i<N ; i++) {
    t1 = gettimeofday();
    operation being measured
    t2 = gettimeofday();
    time[i] = t2 - t1;
}

print "average is ", avg(time[0..N-1]), "\n";
```

Important because printing is a heavy operation that affects cache state and may cause a context switch
Solution 2: invert loop and measurement

\[
t_1 = \text{gettimeofday}(); \\
\text{for } (i=0 ; i<N ; i++) \{ \\
    \text{operation being measured} \\
\} \\
\]
\[
t_2 = \text{gettimeofday}(); \\
\text{print } "\text{average was }", (t_2 - t_1)/N, "\n";
\]

Measurement of $t_2 - t_1$ will now be stable and meaningful.

But need to subtract loop overhead (found by measuring an empty loop).
Problem: compiler optimizations

Compiler may optimize away an empty loop
Even if it has an operation in it like \( j++ \)
If \( j \) is not subsequently used, why update it?
Even if it is, it's value will grow just like that of the loop index

Could work if the operation is on a global variable, because cross-procedural optimization is harder

Still, need to check

e.g. verify that measured overhead depends on number of iterations
Solution 2a: reduce effect of overhead by unrolling

```c
    t1 = gettimeofday();
    for (i=0 ; i<N/3 ; i++) {
        operation being measured
        operation being measured
        operation being measured
    }
    t2 = gettimeofday();
    print "average was ", (t2 – t1)/N, "\n";
```

Need to decide number of repetitions
In all iterative measurements, need to define $N$

$N$ should be big enough to pass resolution limit and average out random errors

$N$ should not be too big so as to reduce risk of large interferences from other activities in the system

Underlying assumption that measurement is short

Therefore completed within a single scheduling quantum without interference

If this is not the case, need to account for time spent doing other things

But what if we're unlucky?
Example: time to write 100 bytes at the beginning of a file

outliers are not common, but do happen
Solution 3: double loop

```c
for (r=0 ; r<REP ; r++) {
    t1 = gettimeofday();
    for (i=0 ; i<N ; i++) {
        operation being measured
    }
}
```

```c
t2 = gettimeofday();
print "average was ", (t2 – t1)/N, "\n";
```

Catch outliers in which interference has occurred
Outliers may also exhibit periodic behavior indicative of some system activity that needs to be understood.

Example is writing successive blocks of 100 bytes to a file.
Biggest problem:
Systematic errors that don't make sense

Need to look at the data
notice that something is wrong
identify it precisely
remove it from the analysis

Hard to do if sensitive to exact conditions
example code:

```c
for (i=0 ; i<=10 ; i++) {
    gettimeofday( &ts[i], 0 );
}
for (i=1 ; i<=10 ; i++) {
    printf("delta=%d\n",
            ts[i].tv_usec - ts[i-1].tv_usec);
}

gmtimeofday( &ts[0], 0 );
gmtimeofday( &ts[1], 0 );
...
gmtimeofday( &ts[10], 0 );

for (i=1 ; i<=10 ; i++) {
    printf("delta=%d\n",
            ts[i].tv_usec - ts[i-1].tv_usec);
}
```
Possible output:

<table>
<thead>
<tr>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

But also:

<table>
<thead>
<tr>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
<th>delta</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

delta=2  delta=103
delta=1  delta=1
delta=1  delta=33
delta=1  delta=1
delta=1  delta=1
delta=1  delta=1
delta=1  delta=1
delta=1  delta=1
delta=1  delta=1
delta=1  delta=1

so we seem to have some random interference
But if we repeat this 10 times and average, a typical result is:

- avg delta = 1.20
- avg delta = 1.00
- avg delta = 0.70
- avg delta = 0.90
- avg delta = 1.00
- avg delta = 1.00
- avg delta = 0.90
- avg delta = 1.00
- avg delta = 1.10
- avg delta = 1.00

(Note: typical is not always...)

So there is actually some systematic problem at the beginning of the second batch of measurements.
Sensitivity:

This only happens when output is directed to the terminal; all is well if it is directed to a file.

This is related to printing the results of the first batch just before starting the second batch; all is well if all printing is done at the end.
Calculating Confidence Intervals
When we perform multiple measurements of the same thing, we can calculate confidence intervals.

Assume measurements are samples from a (normal) distribution.

Characterize the distribution's dispersion.

Find the range that includes the desired mass of the probability density (e.g. 90%).
Assume a set of measurements come from a normal distribution (real value + random error).

This set has an average, which is an estimate of the real value.

If we repeat this with different samples, we will get a slightly different average.

So multiple samples from the base distribution induce a single sample from the distribution of averages.
multiple sets of samples induce multiple samples from the distribution of averages

The distribution of averages is narrower than the base distribution

So it gives a tighter estimate of the real value
Assumption: the averages reflect a true value plus some random noise.

Thus the averages are distributed around the true value.
Assumption: the averages reflect a true value plus some random noise

Thus the averages are distributed around the true value

Given the distribution, we can find the range $h$ that is expected to contain 90% of the averages
This also works the other way: for 90% of the averages, the true value is within $h$.

So the range average $\pm h$ has probability 0.9 to include the true value.
Let $\mu$ denote the real mean of the base distribution

Let $\bar{x}$ denote the average of $n$ samples

If the base distribution is normal, then the averages have a $t$ distribution

Let $\alpha$ denote the acceptable uncertainty (implying that the level of confidence is $1-\alpha$)

Define the half-width to be

$$h = t_{n-1,1-\alpha/2} \frac{S}{\bar{x}}$$

Then

$$p(|\bar{x} - \mu| < h) = 1 - \alpha$$
The half width:

\[ h = t_{n-1, 1-\alpha/2} S_{\bar{x}} \]

t\(_{n-1, 1-\alpha/2}\) comes from tables
n-1 degrees of freedom
n is the number of samples
for large n approximated by the normal
\( Z \sim \mathcal{N}(0, 1) \)
\( S_{\bar{x}} \) is the standard deviation of the averages
assuming the base samples are independent, this can be calculated as \( s / \sqrt{n} \) (where s is the standard deviation of the base samples)

with more samples, the distribution of the averages becomes narrower
The confidence interval:

\[ p(\left| \bar{x} - \mu \right| < h) = 1 - \alpha \]

with a certainty of \(1 - \alpha\), the distance between a sample of the average \(\bar{x}\) and the true mean \(\mu\) is less than \(h\)

If we repeat this many times, and each time we draw a segment of \(\pm h\) around \(\bar{x}\), then in \(1 - \alpha\) of the cases this segment will include \(\mu\)
Assumptions:
The base samples come from a normal distribution

    If not, but have a finite variance, the averages will still be normal, but this requires a larger $n$

Base samples are independent

    If not, maybe using larger batches will reduce the correlation between them
Note: first clean the data, then compute confidence intervals

Remove outliers that indicate interference

Mechanized approach: remove top and bottom measurements

Better approach: look at the data!

Usually, outliers are only bigger (interference cannot reduce the measured time)

Remove history/warmup effects