Performance Evaluation in Database Research: Principles and Experiences

Stefan Manegold

Stefan.Manegold@cwi.nl
Amsterdam, The Netherlands
http://www.cwi.nl/~manegold/

Planning & conducting experiments

- What do you plan to do / analyze / test / prove / show?
  - Which data / data sets should be used?
  - Which workload / queries should be run?
  - Which hardware & software should be used?
  - Metrics:
    - What to measure?
    - How to measure?
    - How to compare?
    - CSI: How to find out what is going on?

- Micro-benchmarks
- Standard benchmarks
- Real-life applications

Data sets & workloads

- Micro-benchmarks
- Standard benchmarks
- Real-life applications

Micro-benchmarks

Definition
- Specialized, stand-alone piece of software
- Isolating one particular piece of a larger system
- E.g., single DB operator (select, join, aggregation, etc.)

Pros
- Focused on problem at hand
- Controllable workload and data characteristics
  - Data sets (synthetic & real)
  - Data size / volume (scalability)
  - Value ranges and distribution
  - Correlation
  - Queries
  - Workload size (scalability)
- Allow broad parameter range(s)
- Useful for detailed, in-depth analysis
- Low setup threshold; easy to run

Cons
- Neglect larger picture
- Neglect contribution of local costs to global / total costs
- Neglect impact of micro-benchmark on real-life applications
- Neglect embedding in context / system at large
- Generalization of result difficult
- Application of insights in full systems / real-life applications not obvious
- Metrics not standardized
- Comparison?
Performance Evaluation: Principles & Experiences

**Examples**
- RDBMS, OODBMS, ORDMBS: TPC-(A,B,C,H,R,DS), O07, ...
- XML, XPath, XQuery, XUF, SQL/XML: MBench, XBench, XMach-1, XMark, X007, TPoX, ...
- Stream Processing: Linear Road, ...
- General Computing: SPEC, ...

**Pros**
- Mimic real-life scenarios
- Publicly available
- Well defined (in theory ...)
- Scalable data sets and workloads (if well designed ...)
- Metrics well defined (if well designed ...)
- Easily comparable (?)

**Cons**
- Often "outdated" (standardization takes (too?) long)
- Often compromises
- Often very large and complicated to run
- Limited dataset variation
- Limited workload variation
- Systems are often optimized for the benchmark(s), only!

**Real-life applications**

**Pros**
- There are so many of them
- Existing problems and challenges

**Cons**
- There are so many of them
- Proprietary datasets and workloads

**Choosing the hardware**

Choice mainly depends on your problem, knowledge, background, taste, etc.

What ever is required by / adequate for your problem

A laptop might not be the most suitable / representative database server...

**Choosing the software**

Which DBMS to use?

**Commercial**
- Require license
- "Free" versions with limited functionality and/or optimization capabilities?
- Limitations on publishing results
- No access to code
- Optimizers
- Analysis & Tuning Tools

**Open source**
- Freely available
- No limitations on publishing results
- Access to source code

Other choices depend on your problem, knowledge, background, taste, etc.

- Operating system
- Programming language
- Compiler
- Scripting languages
- System tools
- Visualization tools

**Analysis: "CSI"**

- Investigate (all?) details
- Analyze and understand behavior and characteristics
- Find out where the time goes and why!

**Publication**

- "Sell your story"
- Describe picture at large
- Highlight (some) important / interesting details
- Compare to others
### Metrics: What to measure?

- **Basic**
  - Throughput: queries per time
  - Evaluation time:
    - wall-clock time ("real")
    - user CPU time ("user")
    - system CPU time ("system")
  - Server-side vs. client-side
  - Memory and/or storage usage / requirements

- **Comparison**
  - Scale-up
  - Speed-up

- **Analysis**
  - System events & interrupts
  - Hardware events

### Performance Evaluation: Principles & Experiences

- Laptop: 1.5 GHz Pentium M (Dothan), 2 MB L2 cache, 2 GB RAM, 5400 RPM disk
- TPC-H (sf = 1)
- MonetDB/SQl v5.5.0/2.23.0
- measured 3rd & 4th of four consecutive runs

<table>
<thead>
<tr>
<th>run</th>
<th>server</th>
<th>client</th>
<th>time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2830</td>
<td>3533</td>
<td>3534</td>
</tr>
<tr>
<td>16</td>
<td>550</td>
<td>618</td>
<td>707</td>
</tr>
</tbody>
</table>

### Metrics: How to measure?

#### Planning Presentation
- Repeatability
- Summary

**Benchmarks HW SW Metrics How to run Compare CSI**

- **Tools, functions and/or system calls to measure time:**
  - **Unix**
    - `/user/bin/time`, shell built-in time
    - Reports `real`, "user" & "sys" time (milliseconds)
    - Measures entire process incl. start-up
    - Note: output format varies!
  - `gettimeofday()`:
    - System function requires source code
    - Reports timestamp (microseconds)

- **Windows**
  - `QueryPerformanceFrequency()`
  - `QueryPerformanceCounter()`
  - `TimeGetTime()`

- **Commands**
  - `db2batch`
  - `db2listproc`
  - `db2profile`

**System events & interrupts**

- `MCD箪aryCounter` (with `mclient --interactive --timer=(clock,performance)`)
- `user.s0`

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```

### Metrics: How to run

**Example**

```c
message := user.s0
mclient --interactive --timer=(clock,performance) message
```
### How to run experiments

#### “We run all experiments in warm memory.”

Laptop: 1.5 GHz Pentium M (Dothan), 2 MB L2 cache, 2 GB RAM, 5400 RPM disk

- TPC-H (sf = 1)
- MonetDB/SQL v5.5.0/2.23.0
- measured last of three consecutive runs

<table>
<thead>
<tr>
<th>Q</th>
<th>cold</th>
<th>hot</th>
<th>time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2930</td>
<td>2830</td>
<td></td>
</tr>
</tbody>
</table>

#### “hot” vs. “cold” & user vs. real time

- Laptop: 1.5 GHz Pentium M (Dothan), 2 MB L2 cache, 2 GB RAM, 5400 RPM disk
- TPC-H (sf = 1)
- MonetDB/SQL v5.5.0/2.23.0
- measured last of three consecutive runs

<table>
<thead>
<tr>
<th>Q</th>
<th>cold</th>
<th>hot</th>
<th>user</th>
<th>real</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2930</td>
<td>2830</td>
<td>3534</td>
<td>13243</td>
</tr>
</tbody>
</table>

Be aware what you measure!

### Of apples and oranges

- Laptop: 1.5 GHz Pentium M (Dothan), 2 MB L2 cache, 2 GB RAM, 5400 RPM disk
- TPC-H (sf = 1)
- MonetDB/SQL v5.5.0/2.23.0
- measured last of three consecutive runs

#### Two colleagues A & B each implemented one version of an algorithm, A the “old” version and B the improved “new” version

- They ran identical experiments on identical machines, each for his code.
- Though both agreed that B’s new code should be significantly better, results were consistently worse.

#### Once upon a time at CWI ...

- Two colleagues A & B each implemented one version of an algorithm, A the “old” version and B the improved “new” version
- They ran identical experiments on identical machines, each for his code.
- Though both agreed that B’s new code should be significantly better, results were consistently worse.
- They tested, profiled, analyzed, argued, wondered, fought for several days ...

Be aware and document what you do / choose.
DeBuG
configure --enable-debug --disable-optimize --enable-assert
CFLAGS = "-g [-0] ..."

OPTimized
configure --enable-debug --disable-optimize --disable-assert
CFLAGS = "-O3 -fomit-frame-pointer -pipe ..."

in case of doubt, check:
msql-server5 --version

S. Manegold (CWI)

DeBuG
configure --enable-debug --disable-optimize --enable-assert
CFLAGS = "-g [-0] ...

OPTimize
configure --enable-debug --disable-optimize --disable-assert
CFLAGS = "-O3 -fomit-frame-pointer -pipe ...

in case of doubt, check:
msql-server5 --version

S. Manegold (CWI)

Once upon a time at CWI ...
- Two colleagues A & B each implemented one version of an algorithm, A the "old" version and B the improved "new" version
- They ran identical experiments on identical machines, each for his code.
- Though both agreed that B's new code should be significantly better, results were consistently worse.
- They tested, profiled, analyzed, argued, wondered, fought for several days ...
- ... and eventually found out that A had compiled with optimization enabled, while B had not ...

Our problem-specific, hand-tuned, prototype outperforms an out-of-the-box installation of a full-fledged off-the-shelf system Y;

Our problem-specific, hand-tuned, prototype X outperforms an out-of-the-box installation of a full-fledged off-the-shelf system Y;

Our problem-specific, hand-tuned, prototype X outperforms an out-of-the-box installation of a full-fledged off-the-shelf system Y;

- Compiler optimization ⇒ up to factor 2 performance difference
- DBMS configuration and tuning ⇒ factor x performance difference (2 ≤ x ≤ 10?)
  - "Self-*" still research
  - Default settings often too "conservative"
  - Do you know all systems you use/compare equally well?

- Compiler optimization ⇒ up to factor 2 performance difference
- DBMS configuration and tuning ⇒ factor x performance difference (2 ≤ x ≤ 10?)
  - "Self-*" still research
  - Default settings often too "conservative"
  - Do you know all systems you use/compare equally well?

- Compiler optimization ⇒ up to factor 2 performance difference
- DBMS configuration and tuning ⇒ factor x performance difference (2 ≤ x ≤ 10?)
  - "Self-*" still research
  - Default settings often too "conservative"
  - Do you know all systems you use/compare equally well?

Be at least aware of the crucial factors and their impact,

"Absolutely fair" comparisons virtually impossible

But:
Be at least aware of the crucial factors and their impact, and document accurately and completely what you do.

Do you know what happens?

Simple In-Memory Scan: SELECT MAX(column) FROM table
Simple In-Memory Scan: SELECT MAX(column) FROM table

- No disk-I/O involved
- Up to 10x improvement in CPU clock-speed
- Yet hardly any performance improvement!??

Research: Always question what you see!

Standard profiling (e.g., `gcc -gp` + `gprof`) does not reveal more (in this case)

Use hardware performance counters to analyze cache-hits, -misses & memory accesses

VTune, oprofile, perfctr, perfmon2, PAPI, PCL, etc.

Use profiling and monitoring tools

- `gcc -gp` + `gprof`
  - Reports call tree, time per function and time per line
  - Requires re-compilation and static linking
- `valgrind --tool=callgrind + l cachegrind`
  - Reports call tree, times, instructions executed and cache misses
  - Thread-aware
  - Does not require (re-)compilation
  - Simulation-based slows down execution up to a factor 100
- Hardware performance counters
  - To analyze cache-hits, -misses & memory accesses
  - VTune, oprofile, perfctr, perfmon2, PAPI, PCL, etc.
- System monitors
  - ps, top, iostat, ...

Find out what happens!

Use info provided by the tested software (DBMS)
- IBM DB2
  - db2expin
- Microsoft SQLServer
  - GUI and system variables
- MySQL, PostgreSQL
  - EXPLAIN select ...
- MonetDB/SQL
  - (PLAN | EXPLAIN | TRACE) select ...
Guidelines for preparing good graphic charts

Maximize information: try to make the graph self-sufficient
- Use keywords in place of symbols to avoid a join in the reader’s brain
- Use informative axis labels: prefer “Average I/Os per query” to “Average 1/0s” to “I/0s”
- Include units in the labels: prefer “CPU time (ms)” to “CPU time”

Minimize ink: present as much information as possible with as little ink as possible
Prefer the chart that gives the most information out of the same data

Use commonly accepted practice: present what people expect
- Usually axes begin at 0, the factor is plotted on x, the result on y
- Usually scales are linear, increase from left to right, divisions are equal
- Use exceptions as necessary

http://www.edwardtufte.com/tufte/books_vdqi
Common presentation mistakes

Presenting many result variables on a single chart
Commonly done to fit into available page count :-((

![Graph showing throughput and utilization](image)

Huh?

Using symbols in place of text

![Graphs showing response time and arrival rate](image)

Human brain is a poor processor
Humans get frustrated by computing joins

Changing the graphical layout of a given curve from one figure to another

![Graphs comparing execution times](image)

What do you mean “my graphs are not legible”?

Pictorial games

MINE is better than YOURS!

![Graphs comparing MINE and YOURS](image)

Plot random quantities without confidence intervals

![Graphs showing random quantities](image)

Overlapping confidence intervals sometimes mean the two quantities are statistically indifferent

Manipulating cell size in histograms

![Histograms with different cell sizes](image)

Rule of thumb: each cell should have at least five points
Not sufficient to uniquely determine what one should do.
"We use a machine with 3.4 GHz."
### Specifying hardware environments

- **CPU:** Vendor, model, generation, clockspeed, cache size(s)
  - 1.5 GHz Pentium M (Dothan), 32 KB L1 cache, 2 MB L2 cache
- **Main memory:** size
  - 2 GB RAM
- **Disk (system):** size & speed
  - 120 GB Laptop ATA disk @ 5400RPM
  - 1 TB striped RAID-0 system (5x 200 GB S-ATA disk @ 7200 RPM
- **Network (interconnection):** type, speed & topology
  - 1 GB shared Ethernet

### Specifying software environments

- **Product names, exact version numbers, and/or sources where obtained from**

### Making experiments repeatable

**Purpose:** another human equipped with the appropriate software and hardware can repeat your experiments.

- **Your supervisor / your students**
- **Your colleagues**
- **Yourself, 3 months later when you have a new idea**
- **Yourself, 3 years later when writing the thesis or answering requests for that journal version of your conference paper**
- **Future researchers (you get cited!)**

### Making experiments portable

**Try to use not-so-exotic hardware**

**Try to use free or commonly available tools (databases, compilers, plotters...)**

**Try to use not-so-exotic hardware**

**Try to use free or commonly available tools (databases, compilers, plotters...)**

**Clearly, scientific needs go first (joins on graphic cards; smart card research; energy consumption study...)**

### Summary

- **Planning & conducting experiments**
- **Presentation**
- **Repeatability**
  - Portable parameterizable experiments
  - Test suite
  - Documenting your experiment suite

---

This information is provided for educational purposes only. Always consult relevant documentation and consider the context of your specific use case.
You may omit using
Matlab as the driving platform for the experiments
20-years old software that only works on an old SUN and is now unavailable

You may omit using
Matlab as the driving platform for the experiments
20-years old software that only works on an old SUN and is now unavailable
- If you really love your code, you may even maintain it

Try to use free or commonly available tools (databases, compilers, plotters...)

Clearly, scientific needs go first (joins on graphic cards; smart card research; energy consumption study...)

We provide a new algorithm that on a Debian Linux machine with 4 GHz CPU, 60 GB disk, DMA, 2 GB main memory and our own brand of system libraries consistently outperforms the state of the art.

Try to use not-so-exotic hardware

Try to use free or commonly available tools (databases, compilers, plotters...)

Clearly, scientific needs go first (joins on graphic cards; smart card research; energy consumption study...)

We provide a new algorithm that consistently outperforms the state of the art.

You may omit using
Matlab as the driving platform for the experiments
20-years old software that only works on an old SUN and is now unavailable

You may omit using
Matlab as the driving platform for the experiments
20-years old software that only works on an old SUN and is now unavailable
- If you really love your code, you may even maintain it

Try to use free or commonly available tools (databases, compilers, plotters...)

Clearly, scientific needs go first (joins on graphic cards; smart card research; energy consumption study...)

We provide a new algorithm that on a Debian Linux machine with 4 GHz CPU, 60 GB disk, DMA, 2 GB main memory and our own brand of system libraries consistently outperforms the state of the art.

Try to use not-so-exotic hardware

Try to use free or commonly available tools (databases, compilers, plotters...)

Clearly, scientific needs go first (joins on graphic cards; smart card research; energy consumption study...)

We provide a new algorithm that consistently outperforms the state of the art.

You may omit using
Matlab as the driving platform for the experiments
20-years old software that only works on an old SUN and is now unavailable

You may omit using
Matlab as the driving platform for the experiments
20-years old software that only works on an old SUN and is now unavailable
- If you really love your code, you may even maintain it

Try to use free or commonly available tools (databases, compilers, plotters...)

Clearly, scientific needs go first (joins on graphic cards; smart card research; energy consumption study...)

We provide a new algorithm that on a Debian Linux machine with 4 GHz CPU, 60 GB disk, DMA, 2 GB main memory and our own brand of system libraries consistently outperforms the state of the art.

Try to use not-so-exotic hardware

Try to use free or commonly available tools (databases, compilers, plotters...)

Clearly, scientific needs go first (joins on graphic cards; smart card research; energy consumption study...)

We provide a new algorithm that consistently outperforms the state of the art.

You may omit using
Matlab as the driving platform for the experiments
20-years old software that only works on an old SUN and is now unavailable

You may omit using
Matlab as the driving platform for the experiments
20-years old software that only works on an old SUN and is now unavailable
- If you really love your code, you may even maintain it

Try to use free or commonly available tools (databases, compilers, plotters...)

Clearly, scientific needs go first (joins on graphic cards; smart card research; energy consumption study...)

We provide a new algorithm that on a Debian Linux machine with 4 GHz CPU, 60 GB disk, DMA, 2 GB main memory and our own brand of system libraries consistently outperforms the state of the art.

Try to use not-so-exotic hardware

Try to use free or commonly available tools (databases, compilers, plotters...)

Clearly, scientific needs go first (joins on graphic cards; smart card research; energy consumption study...)

We provide a new algorithm that on a Debian Linux machine with 4 GHz CPU, 60 GB disk, DMA, 2 GB main memory and our own brand of system libraries consistently outperforms the state of the art.

Try to use not-so-exotic hardware

Try to use free or commonly available tools (databases, compilers, plotters...)

Clearly, scientific needs go first (joins on graphic cards; smart card research; energy consumption study...)

We provide a new algorithm that on a Debian Linux machine with 4 GHz CPU, 60 GB disk, DMA, 2 GB main memory and our own brand of system libraries consistently outperforms the state of the art.

Try to use not-s
### Portability

#### Test Suite Documenting

- Building a test suite
- Making experiments parameterizable
- Making your code parameterizable

### Planning

- Presentation
- Repeatability
- Summary

### Portability

#### Test suite Documenting

- Making your code parameterizable
- Building a test suite

### Making experiments parameterizable

- Purpose: have a very simple mean to obtain a test for the values

\[ f_1 = v_1, f_2 = v_2, \ldots, f_k = v_k \]

- Many tricks. Very simple ones:
  - `argc / argv`: specific to each class’ main
  - Configuration files
  - `Java Properties` pattern
  - `+ command-line arguments`

### Making experiments parameterizable

- The bottom line: you will want to run it in different settings

  - With your or the competitor’s algorithm or special optimization
  - On your desktop or your laptop
  - With a local or remote MySQL server
  - Make it easy to produce a point
  - If it is very difficult to produce a new point, ask questions

**You may omit coding like this:**

The input data set files should be specified in source file `util.GlobalProperty.java`.

### Making your code parameterizable

- The input data set files should be specified in source file `util.GlobalProperty.java`

**Con:** the values are read when the process is created

**Pro:** human-readable even without running code

#### Configuration Files

- Omnispresent in large-scale software
  - Crucial if you hope for serious installations: see gnu software install procedure
  - Decide on a specific relative directory, fix the syntax
  - Report meaningful error if the configuration file is not found

### Building a test suite

- You already have:
  - Designs
  - Easy way to get any measure point

- You need:
  - Suited directory structure (e.g.: `source`, `bin`, `data`, `res`, `graphs`)
  - Control loops to generate the points needed for each graph, under `res/`, and possibly to produce graphs under `graphs`
    - Even Java can be used for the control loops, but...
    - It does pay off to know how to write a loop in shell/perl etc.

### Automatically generated graphs

- You have:
  - files containing numbers characterizing the parameter values and the results
  - basic shell skills

**You may omit coding like this:**

Change the value of the `delta` variable in `distribution.DistFreeNode.java` into 1.5, 15, 20 and so on.
Automatically generating graphs with Excel

Create an Excel file results-m1-n5.xls with the column labels:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scale factor</td>
<td>Execution time</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Insert in the area B2-C3 a link to the file results-m1-n5.csv

Create in the .xls file a graph out of the cells A1:B3, chose the layout, colors etc.

When the .csv file will be created, the graph is automatically filled in.

Automatically generating graphs with Gnuplot

Data file results-m1-n5.csv:

<table>
<thead>
<tr>
<th>1</th>
<th>1234</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2467</td>
</tr>
<tr>
<td>3</td>
<td>4623</td>
</tr>
</tbody>
</table>

Gnuplot command file plot-m1-n5.gnu to plot this graph:

- Set data style linespoints
- Set terminal postscript eps color
- Set output "results-m1-n5.eps"
- Set title "Execution time for various scale factors"
- Set xlabel "Scale factor"
- Set ylabel "Execution time (ms)"
- Plot "results-m1-n5.csv"

Call gnuplot plot-m1-n5.gnu
Why you should take care to generate your own graphs

File `avgs.out` contains average times over three runs:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.666</td>
<td>2 15</td>
</tr>
<tr>
<td>12.3333</td>
<td>4 13</td>
</tr>
</tbody>
</table>

Copy-paste into OpenOffice 2.3.0-6.11-fc8:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>13666</td>
<td>2 15</td>
</tr>
<tr>
<td>123333</td>
<td>4 13</td>
</tr>
</tbody>
</table>

The graph doesn’t look good :-(

Documenting your experiment suite

Very easy if experiments are already portable, parameterizable, and if graphs are automatically generated.

Specify:
- What the installation requires; how to install
- For each experiment:
  - Extra installation if any
  - Script to run
  - Where to look for the graph
- How long it takes

Summary & conclusions

- Good and repeatable performance evaluation and experimental assessment require no fancy magic but rather solid craftsmanship
- Proper planning helps to keep you from “getting lost” and ensure repeatability
- Repeatable experiments simplify your own work (and help others to understand it better)
- There is no single way how to do it right.
- There are many ways how to do it wrong.
- We provided some simple rules and guidelines what (not) to do.