



**HAL**  
open science

# Performance evaluation in database research: principles and experience

Ioana Manolescu, Stefan Manegold

► **To cite this version:**

Ioana Manolescu, Stefan Manegold. Performance evaluation in database research: principles and experience. 12th International Conference on Extending Database Technology, Mar 2009, Saint Petersburg, Russia. inria-00431423

**HAL Id: inria-00431423**

**<https://inria.hal.science/inria-00431423>**

Submitted on 12 Nov 2009

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Performance Evaluation in Database Research: Principles and Experiences

Ioana Manolescu<sup>1</sup>   Stefan Manegold<sup>2</sup>

<sup>1</sup>INRIA Saclay-Île-de-France, France, ioana.Manolescu@inria.fr

<sup>2</sup>CWI Amsterdam, The Netherlands, Stefan.Manegold@cwi.nl

EDBT 2009

## Disclaimer

- There is **no single way** how to do it **right**.
- There are **many ways** how to do it **wrong**.
- This is not a “mandatory” script.
- This is more a collection of **anecdotes** or **fairy tales** — not always to be taken literally, only, but all provide some **general rules** or **guidelines** *what (not) to do*.





## 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?

# Data sets & workloads

- Micro-benchmarks
  - Standard benchmarks
  - Real-life applications
- 
- No general simple rules, which to use when
  - But some guidelines for the choice...

## 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?

## Examples

- RDBMS, OODBMS, ORDMBS:  
TPC-{A,B,C,H,R,DS}, OO7, ...
- 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!

## Pros

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

## Cons

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

# Two types of experiments

## 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



# 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

# Choosing the software

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

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

# Metrics: What to measure?

- Basic
  - Throughput: queries per time
  - Evaluation time
    - wall-clock (“real”)
    - CPU (“user”)
    - I/O (“system”)
    - Server-side vs. client-side
  - Memory and/or storage usage / requirements
- Comparison
  - Scale-up
  - Speed-up
- Analysis
  - System events & interrupts
  - Hardware events

# Metrics: What to measure?

- 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

Q	server		client		run ... time (milliseconds)
	3rd user	3rd real	3rd real	4th real	
1	2830	3533	3534	3575	
16	550	618	707	1468	

# Metrics: What to measure?

- 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

Q	server		client		run ... time (milliseconds)
	3rd user	3rd real	3rd real	4th real	
1	2830	3533	3534	3575	
16	550	618	707	1468	

# Metrics: What to measure?

- 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

Q	server 3rd		client 3rd		result size	run ... time (milliseconds) output went to ...
	user file	real file	real file	4th real terminal		
1	2830	3533	3534	3575	1.3 KB	
16	550	618	707	1468	1.2 MB	

# Metrics: What to measure?

- 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

Q	server		client		result size	run ... time (milliseconds) output went to ...
	3rd user file	3rd real file	3rd real file	4th real terminal		
1	2830	3533	3534	3575	1.3 KB	
16	550	618	707	1468	1.2 MB	

Be aware *what* you measure!



# Metrics: How to measure?

Tools, functions and/or system calls to measure time: **Unix**

- `/usr/bin/time`, shell built-in time
  - Command line tool  $\Rightarrow$  works with any executable
  - Reports “real”, “user” & “sys” time (*milliseconds*)
  - Measures entire process incl. start-up
  - **Note: output format varies!**
- `gettimeofday()`
  - System function  $\Rightarrow$  requires source code
  - Reports timestamp (*microseconds*)

# Metrics: How to measure?

Tools, functions and/or system calls to measure time: **Windows**

- `TimeGetTime()`, `GetTickCount()`
  - System function  $\Rightarrow$  requires source code
  - Reports timestamp (*milliseconds*)
  - **Resolution can be as coarse as 10 milliseconds**
- `QueryPerformanceCounter()` / `QueryPerformanceFrequency()`
  - System function  $\Rightarrow$  requires source code
  - Reports timestamp (*ticks per seconds*)
  - **Resolution can be as fine as 1 microsecond**
- cf., <http://support.microsoft.com/kb/172338>

# Metrics: How to measure?

Use timings provided by the tested software (DBMS)

- IBM DB2
  - db2batch
- Microsoft SQLserver
  - GUI and system variables
- PostgreSQL

```
postgresql.conf
```

```
log_statement_stats = on  
log_min_duration_statement = 0  
log_duration = on
```

- MonetDB/XQuery & MonetDB/SQL
  - mclient -lxquery -t
  - mclient -lsql -t
  - (PROFILE|TRACE) select ...

# Metrics: How to measure?

```
mclient -lxquery -t -s'1+2'
```

```
3
```

```
Trans 11.626 msec
```

```
Shred 0.000 msec
```

```
Query 6.462 msec
```

```
Print 1.934 msec
```

```
Timer 21.201 msec
```

```
mclient -lsql -t PROFILE_select_1.sql
```

```
% . # table_name
```

```
% single_value # name
```

```
% tinyint # type
```

```
% 1 # length
```

```
[ 1 ]
```

```
#times real 62, user 0, system 0, 100
```

```
Timer 0.273 msec
```

“We run all experiments in warm memory.”

“We run all experiments in warm memory.”



# “hot” vs. “cold”

- Depends on what you want to show / measure / analyze
- No formal definition, but “common sense”

## Cold run

A cold run is a run of the query right after a DBMS is started and no (benchmark-relevant) data is preloaded into the system’s main memory, neither by the DBMS, nor in filesystem caches. Such a clean state can be achieved via a system reboot or by running an application that accesses sufficient (benchmark-irrelevant) data to flush filesystem caches, main memory, and CPU caches.

## Hot run

A hot run is a run of a query such that as much (query-relevant) data is available as close to the CPU as possible when the measured run starts. This can (e.g.) be achieved by running the query (at least) once before the actual measured run starts.

- Be aware and document what you do / choose

## “hot” vs. “cold”

- 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

Q	cold	hot	time (milliseconds)
1	2930	2830	



## “hot” vs. “cold”

- 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

Q	cold		hot		...	time (milliseconds)
	user		user			
1	2930		2830			

## “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

Q	cold		hot		... time (milliseconds)
	user	real	user	real	
1	2930	13243	2830	3534	

## “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

Q	cold		hot		... time (milliseconds)
	user	real	user	real	
1	2930	13243	2830	3534	

Be aware *what* you measure!

# Of apples and oranges

## 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.

# Of apples and oranges

## 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 ...

# Of apples and oranges

## 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** ...

# Of apples and oranges

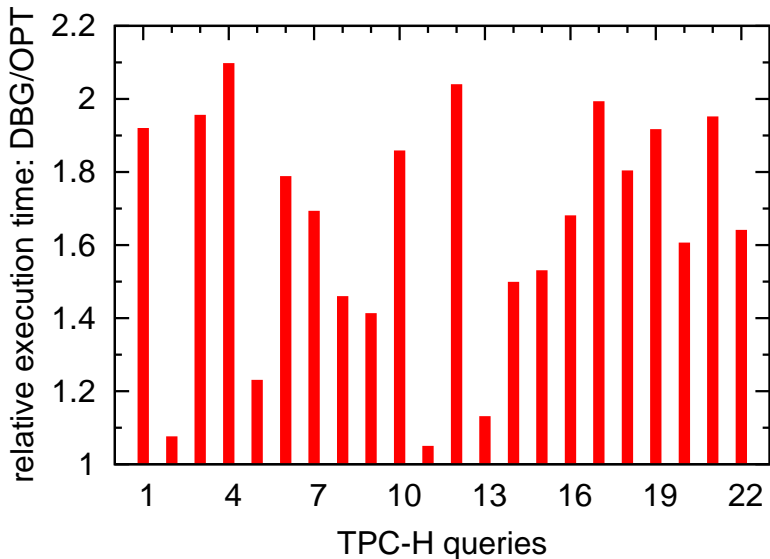
## DBG

```
configure --enable-debug --disable-optimize --enable-assert  
  
CFLAGS = "-g [-O0]"
```

## OPT

```
configure --disable-debug --enable-optimize --disable-assert  
  
CFLAGS = "  
-O6 -fomit-frame-pointer -finline-functions  
-malign-loops=4 -malign-jumps=4 -malign-functions=4  
-fexpensive-optimizations -funroll-all-loops -funroll-loops  
-frerun-cse-after-loop -frerun-loop-opt -DNDEBUG  
"
```

# Of apples and oranges





# Of apples and oranges

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

# Of apples and oranges

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

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

# Of apples and oranges

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

Our problem-specific, hand-tuned, prototype  $X$  outperforms an out-of-the-box installation of a full-fledged off-the-shelf system  $Y$ ; in  $X$ , we focus on pure query execution time, omitting the times for query parsing, translation, optimization and result printing;

# Of apples and oranges

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

Our problem-specific, hand-tuned, prototype  $X$  outperforms an out-of-the-box installation of a full-fledged off-the-shelf system  $Y$ ; in  $X$ , we focus on pure query execution time, omitting the times for query parsing, translation, optimization and result printing; we did not manage to do the same for  $Y$ .

# Of apples and oranges

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

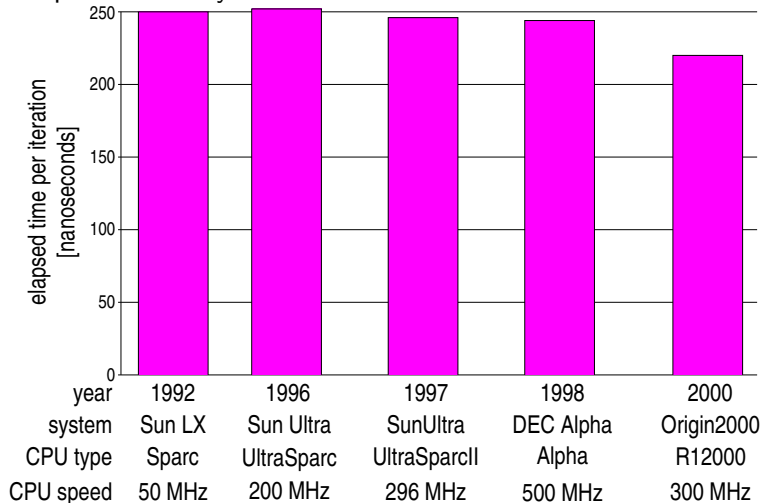
Our problem-specific, hand-tuned, prototype  $X$  outperforms an out-of-the-box installation of a full-fledged off-the-shelf system  $Y$ ; in  $X$ , we focus on pure query execution time, omitting the times for query parsing, translation, optimization and result printing; we did not manage to do the same for  $Y$ .

- “Absolutely fair” comparisons virtually impossible
- But:

Be at least aware of the 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`



# Do you know what happens?

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!??

# Do you know what happens?

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!**



# Do you know what happens?

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)

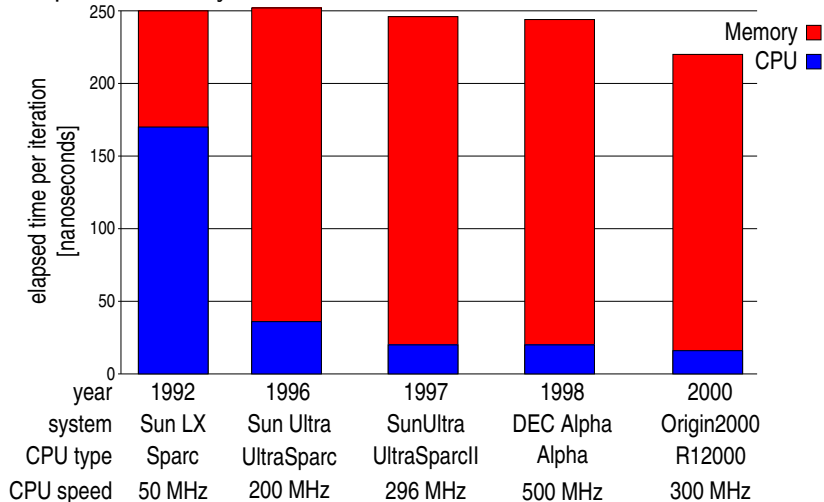
# Do you know what happens?

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)
- Need to dissect CPU & memory access costs
- Use hardware performance counters to analyze cache-hits, -misses & memory accesses
- VTune, oprofile, perfctr, perfmon2, PAPI, PCL, etc.

# Find out what happens!

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



# Find out what happens!

Use info provided by the tested software (DBMS)

- IBM DB2
  - db2expln
- Microsoft SQLserver
  - GUI and system variables
- MySQL, PostgreSQL
  - **EXPLAIN** select ...
- MonetDB/SQL
  - (**PLAN|EXPLAIN|TRACE**) select ...

# Find out what happens!

## 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' + 'kcachegrind'`
  - Reports call tree, times, instructions executed and cache misses
  - Thread-aware
  - Does not require (re-)compilation
  - Simulation-based  $\Rightarrow$  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!

TPC-H Q1 ( $sf = 1$ ) (AMD AthlonMP @ 1533 GHz, 1 GB RAM)

cum.	excl.	calls	ins.	IPC	function
11.9	11.9	846M	6	0.64	ut_fold_ulint_pair
20.4	8.5	0.15M	27K	0.71	ut_fold_binary
26.2	5.8	77M	37	0.85	memcpy
<b>29.3</b>	<b>3.1</b>	<b>23M</b>	<b>64</b>	<b>0.88</b>	<b>Item_sum_sum::update_field</b>
32.3	3.0	6M	247	0.83	row_search_for_mysql
<b>35.2</b>	<b>2.9</b>	<b>17M</b>	<b>79</b>	<b>0.70</b>	<b>Item_sum_avg::update_field</b>
37.8	2.6	108M	11	0.60	rec_get_bit_field_l
40.3	2.5	6M	213	0.61	row_sel_store_mysql_rec
42.7	2.4	48M	25	0.52	rec_get_nth_field
45.1	2.4	60	19M	0.69	ha_print_info
47.5	2.4	5.9M	195	1.08	end_update
49.6	2.1	11M	89	0.98	field_conv
51.6	2.0	5.9M	16	0.77	Field_float::val_real
53.4	1.8	5.9M	14	1.07	Item_field::val
54.9	1.5	42M	17	0.51	row_sel_field_store_in_mysql..
56.3	1.4	36M	18	0.76	buf_frame_align
<b>57.6</b>	<b>1.3</b>	<b>17M</b>	<b>38</b>	<b>0.80</b>	<b>Item_func_mul::val</b>
59.0	1.4	25M	25	0.62	pthread_mutex_unlock
60.2	1.2	206M	2	0.75	hash_get_nth_cell
61.4	1.2	25M	21	0.65	mutex_test_and_set
62.4	1.0	102M	4	0.62	rec_get_lbyte_offs_flag
63.4	1.0	53M	9	0.58	rec_l_get_field_start_offs
64.3	0.9	42M	11	0.65	rec_get_nth_field_extern_bit
<b>65.3</b>	<b>1.0</b>	<b>11M</b>	<b>38</b>	<b>0.80</b>	<b>Item_func_minus::val</b>
<b>65.8</b>	<b>0.5</b>	<b>5.9M</b>	<b>38</b>	<b>0.80</b>	<b>Item_func_plus::val</b>

MySQL gprof trace

SF=1		SF=0.001		tot	res	(BW = MB/s)	
ms	BW	us	BW	MB	size	MIL statement	
127	352	150	305	45	5.9M	s0 := select(Lshipdate).mark	
134	505	113	608	68	5.9M	s1 := join(s0,Lreturnflag)	
134	506	113	608	68	5.9M	s2 := join(s0,Llinestatus)	
235	483	129	887	114	5.9M	s3 := join(s0,Lextprice)	
233	488	130	881	114	5.9M	s4 := join(s0,Ldiscount)	
232	489	127	901	114	5.9M	s5 := join(s0,Ltax)	
134	507	104	660	68	5.9M	s6 := join(s0,Lquantity)	
290	155	324	141	45	5.9M	s7 := group(s1)	
329	136	368	124	45	5.9M	s8 := group(s7,s2)	
0	0	0	0	0	4	s9 := unique(s8.mirror)	
206	440	60	1527	91	5.9M	r0 := [+](1.0,s5)	
210	432	51	1796	91	5.9M	r1 := [-](1.0,s4)	
274	498	83	1655	137	5.9M	r2 := [*](s3,r1)	
274	499	84	1653	137	5.9M	r3 := [*](s12,r0)	
165	271	121	378	45	4	r4 := {sum}(r3,s8,s9)	
165	271	125	366	45	4	r5 := {sum}(r2,s8,s9)	
163	275	128	357	45	4	r6 := {sum}(s3,s8,s9)	
163	275	128	357	45	4	r7 := {sum}(s4,s8,s9)	
144	151	107	214	22	4	r8 := {sum}(s6,s8,s9)	
112	196	145	157	22	4	r9 := {count}(s7,s8,s9)	
3724		2327		<b>TOTAL</b>			

MonetDB/MIL trace



# Graphical presentation of results

We all know

A picture is worth a thousand words

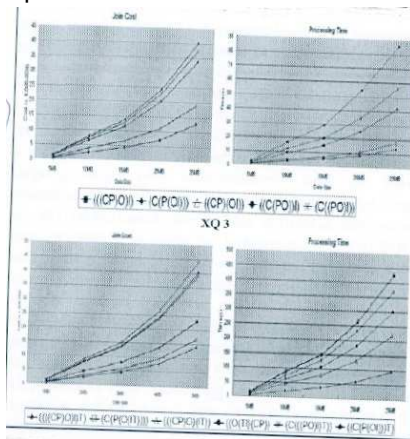


# Graphical presentation of results

We all know

A picture is worth a thousand words

Er, maybe not all pictures...

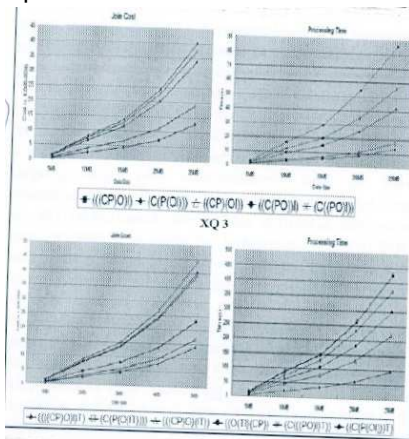


# Graphical presentation of results

We all know

A picture is worth a thousand words

Er, maybe not all pictures...



(Borrowed from T.Grust's slides at VLDB 2007 panel)

# Guidelines for preparing good graphic charts

Require minimum effort from the reader

# Guidelines for preparing good graphic charts

Require minimum effort from the reader

- Not the minimum effort **from you**

# Guidelines for preparing good graphic charts

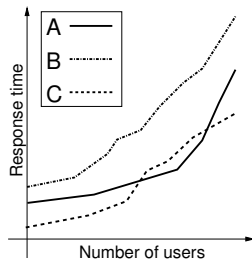
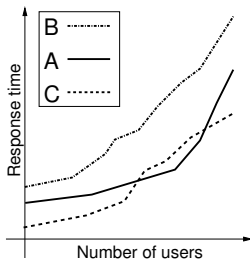
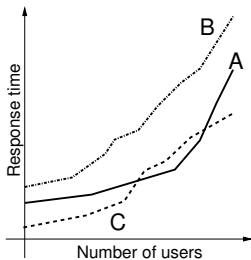
## Require minimum effort from the reader

- Not the minimum effort **from you**
- Try to be honest: how would you like to see it?

# Guidelines for preparing good graphic charts

## Require minimum effort from the reader

- Not the minimum effort **from you**
- Try to be honest: how would you like to see it?



# 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 I/Os" to "I/Os"
- Include units in the labels: prefer "CPU time (ms)" to "CPU time"

# 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 I/Os" to "I/Os"
- Include units in the labels: prefer "CPU time (ms)" to "CPU time"

**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



# Guidelines for preparing good graphic charts

**Minimize ink:** present as much information as possible with as little ink as possible

# Guidelines for preparing good graphic charts

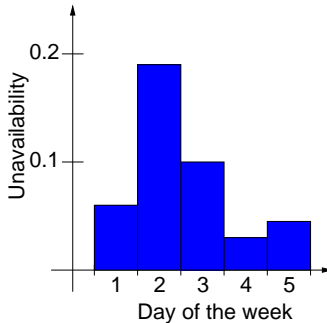
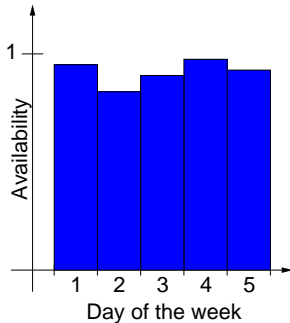
**Minimize ink:** present as much information as possible with as little ink as possible

Prefer the chart that gives the most information out of the same data

# Guidelines for preparing good graphic charts

**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



Edward Tufte: “The Visual Display of Quantitative Information”

[http://www.edwardtufte.com/tufte/books\\_vdqi](http://www.edwardtufte.com/tufte/books_vdqi)

# Common presentation mistakes

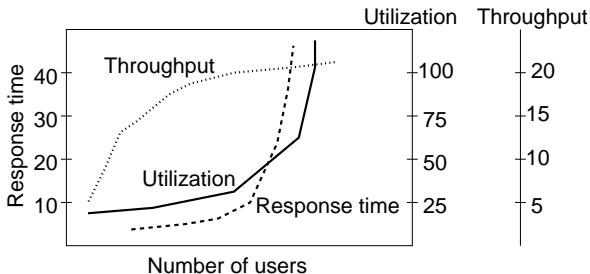
Presenting too many alternatives on a single chart

Rules of thumb, to override with good reason:

- A line chart should be limited to 6 curves
- A column chart or bar should be limited to 10 bars
- A pie chart should be limited to 8 components
- Each cell in a histogram should have at least five data points

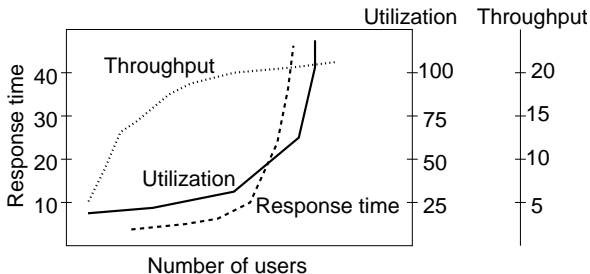
# Common presentation mistakes

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



# Common presentation mistakes

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



Huh?

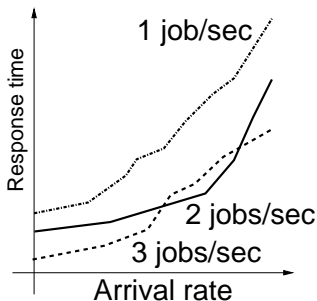
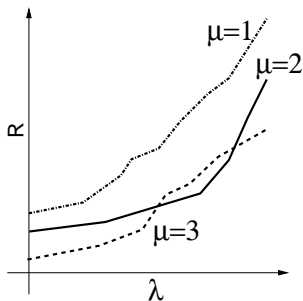
# Common presentation mistakes

Using symbols in place of text



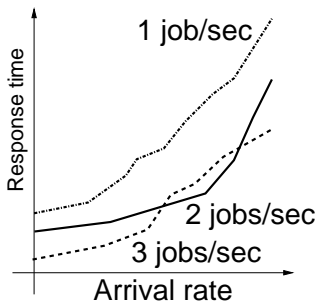
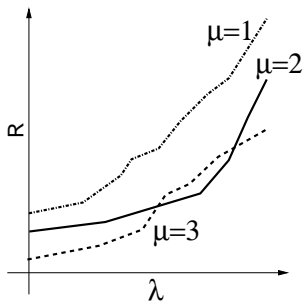
# Common presentation mistakes

Using symbols in place of text



# Common presentation mistakes

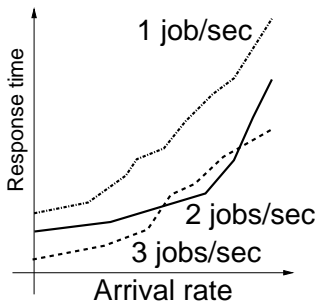
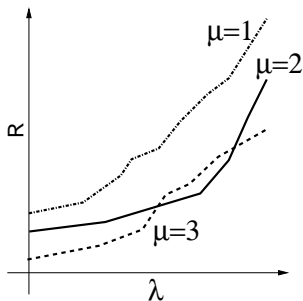
Using symbols in place of text



Human brain is a **poor join processor**

# Common presentation mistakes

Using symbols in place of text



Human brain is a **poor join processor**

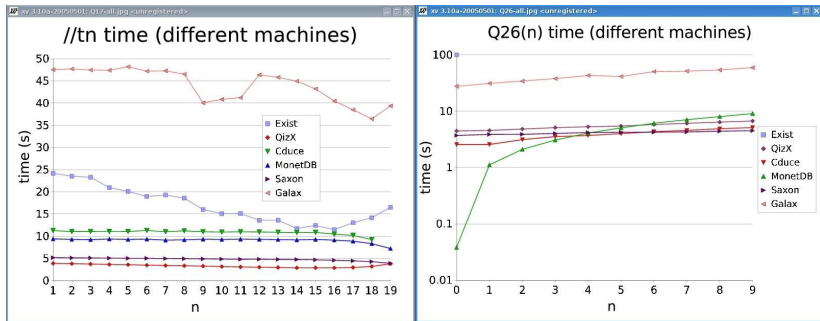
Humans **get frustrated** by computing joins

# Common presentation mistakes

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

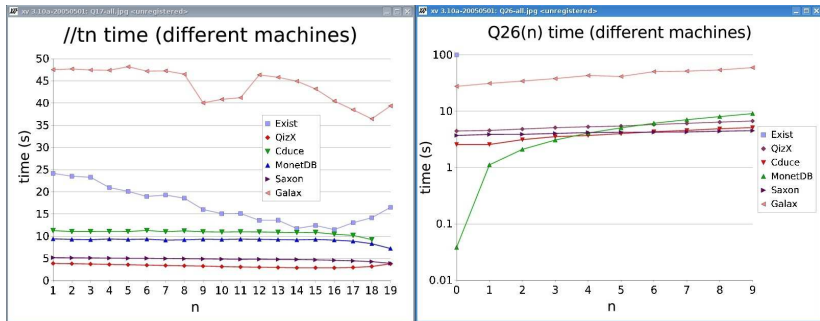
# Common presentation mistakes

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



# Common presentation mistakes

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

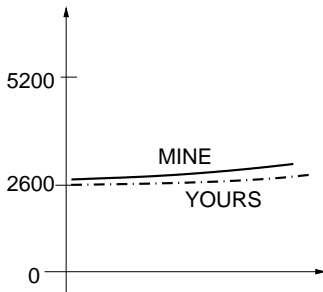
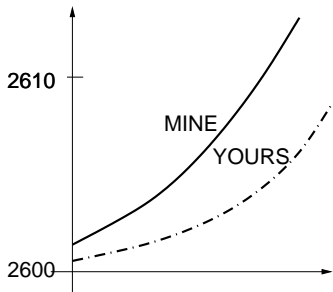


What do you mean “my graphs are not legible”?

MINE is better than YOURS!

# Pictorial games

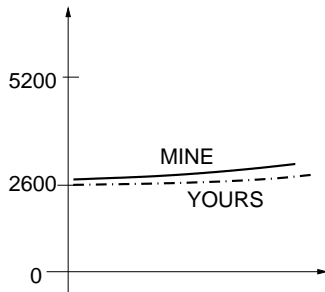
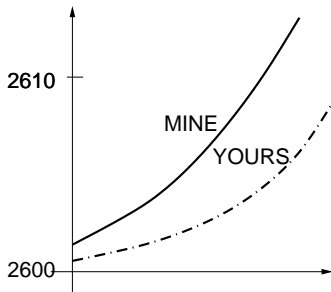
MINE is better than YOURS!





# Pictorial games

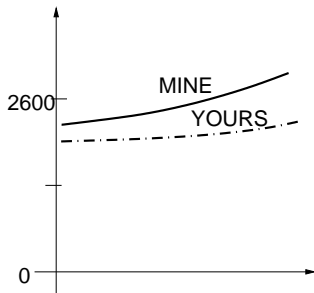
MINE is better than YOURS!



A-ha

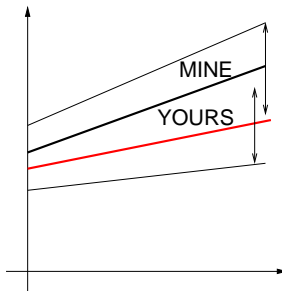
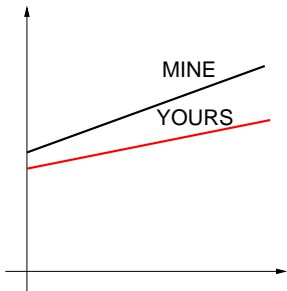
# Pictorial games

Recommended layout: let the useful height of the graph be  $\frac{3}{4}$ th of its useful width



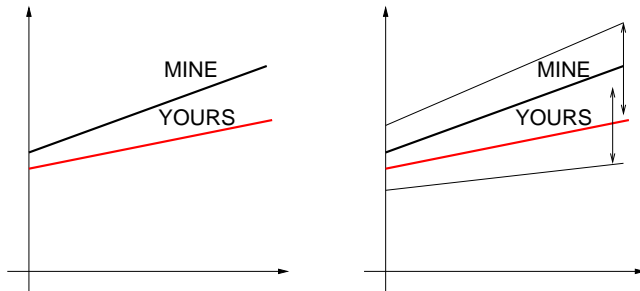
# Pictorial games

Plot random quantities without confidence intervals



# Pictorial games

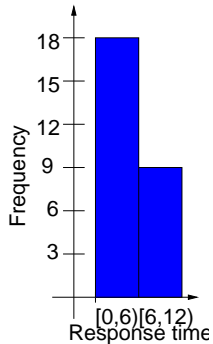
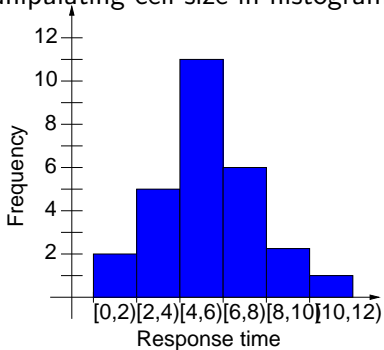
Plot random quantities without confidence intervals



Overlapping confidence intervals sometimes mean the two quantities are statistically indifferent

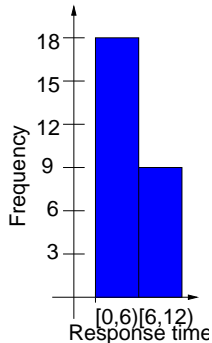
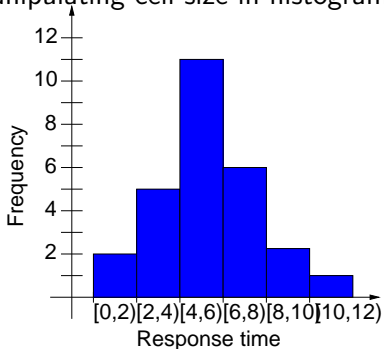
# Pictorial games

Manipulating cell size in histograms



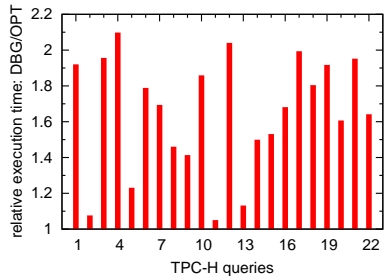
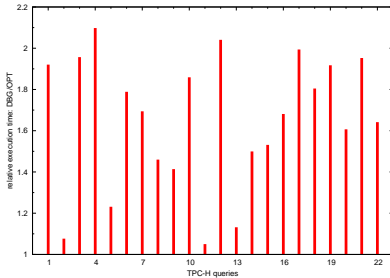
# Pictorial games

Manipulating cell size in histograms



Rule of thumb: each cell should have at least five points  
Not sufficient to uniquely determine what one should do.

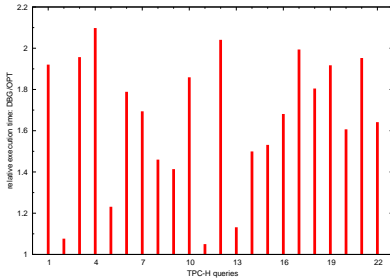
# Pictorial games: gnuplot & L<sup>A</sup>T<sub>E</sub>X



# Pictorial games: gnuplot & L<sup>A</sup>T<sub>E</sub>X

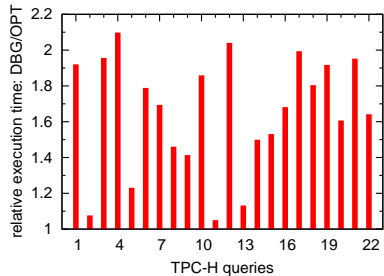
default:

set size ratio 0 1,1



better:

set size ratio 0 0.5,0.5

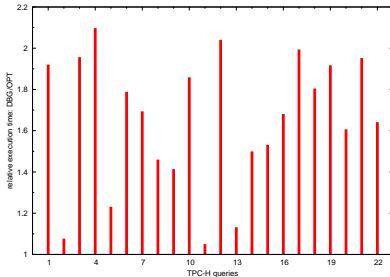




# Pictorial games: gnuplot & L<sup>A</sup>T<sub>E</sub>X

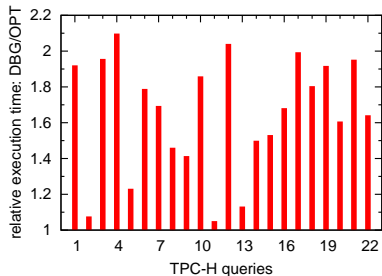
default:

set size ratio 0 1,1



better:

set size ratio 0 0.5,0.5



Rule of thumb for papers:

width of plot =  $x \backslash \text{textwidth}$

⇒ set size ratio 0  $x*1.5, y$

“We use a machine with 3.4 GHz.”

# Specifying hardware environments

“We use a machine with 3.4 GHz.”



3400x

?

“We use a machine with 3.4 GHz.”

⇒ Under-specified!

# Specifying hardware environments

```
cat /proc/cpuinfo
```

```
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 13
model name    : Intel(R) Pentium(R) M processor 1.50GHz
stepping      : 6
cpu MHz       : 600.000
cache size    : 2048 KB
fdiv_bug      : no
hlt_bug       : no
f00f_bug      : no
coma_bug      : no
fpu           : yes
fpu_exception : yes
cpuid level   : 2
wp            : yes
flags         : fpu vme de pse tsc msr mce cx8 mtrr pge mca cmov pat clflush
               dts acpi mmx fxsr sse sse2 ss tm pbe up bts est tm2
bogomips     : 1196.56
clflush size  : 64
```

# Specifying hardware environments

```
cat /proc/cpuinfo
```

```
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 13
model name    : Intel(R) Pentium(R) M processor 1.50GHz ← !
stepping     : 6
cpu MHz      : 600.000 ← throttled down by speed stepping!
cache size   : 2048 KB
fdiv_bug     : no
hlt_bug      : no
f00f_bug     : no
coma_bug     : no
fpu          : yes
fpu_exception : yes
cpuid level   : 2
wp           : yes
flags        : fpu vme de pse tsc msr mce cx8 mtrr pge mca cmov pat clflush
              dts acpi mmx fxsr sse sse2 ss tm pbe up bts est tm2
bogomips     : 1196.56
clflush size : 64
```

# Specifying hardware environments

```
/sbin/lspci -v
```

```
00:00.0 Host bridge: Intel Corporation 82852/82855 GM/GME/PM/GMV Processor to I/O Controller (rev 02)
  Flags: bus master, fast devsel, latency 0
  Memory at <unassigned> (32-bit, prefetchable)
  Capabilities: <access denied>
  Kernel driver in use: agpgart-intel

...

01:08.0 Ethernet controller: Intel Corporation 82801DB PRO/100 VE (MOB) Ethernet Controller (rev 83)
  Subsystem: Benq Corporation Unknown device 5002
  Flags: bus master, medium devsel, latency 64, IRQ 10
  Memory at e0000000 (32-bit, non-prefetchable) [size=4K]
  I/O ports at c000 [size=64]
  Capabilities: <access denied>
  Kernel driver in use: e100
  Kernel modules: e100
```

```
/sbin/lspci -v | wc
```

```
151 lines
861 words
6663 characters
```

# Specifying hardware environments

```
/sbin/lspci -v
```

```
00:00.0 Host bridge: Intel Corporation 82852/82855 GM/GME/PM/GMV Processor to I/O Controller (rev 02)
  Flags: bus master, fast devsel, latency 0
  Memory at <unassigned> (32-bit, prefetchable)
  Capabilities: <access denied>
  Kernel driver in use: agpgart-intel

...

01:08.0 Ethernet controller: Intel Corporation 82801DB PRO/100 VE (MOB) Ethernet Controller (rev 83)
  Subsystem: Benq Corporation Unknown device 5002
  Flags: bus master, medium devsel, latency 64, IRQ 10
  Memory at e0000000 (32-bit, non-prefetchable) [size=4K]
  I/O ports at c000 [size=64]
  Capabilities: <access denied>
  Kernel driver in use: e100
  Kernel modules: e100
```

```
/sbin/lspci -v | wc
```

```
151 lines
861 words
6663 characters
```

⇒ Over-specified!



# 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 @ 5400 RPM
  - 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.

# 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 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 repeatable means:

- 1 Making experiments **portable** and **parameterizable**
- 2 Building a **test suite** and scripts
- 3 Writing **instructions**

# Making experiments portable

Try to use not-so-exotic hardware

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

# Making experiments portable

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...)



# Making experiments portable

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...)

You may omit using

Matlab as the driving platform for the experiments

# Making experiments portable

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...)

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

# Making experiments portable

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...)

## 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

# Making experiments portable

Try  
Try  
plo  
Cle  
rese

You  
Ma  
20-  
una



Code  
maintenance

# Making experiments portable

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...)

## 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)

4-years old library that is no longer distributed and you do no longer have (idem)

# Making experiments portable

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...)

## 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)

4-years old library that is no longer distributed and you do no longer have (idem)

`/usr/bin/time` to time execution, parse the output with `perl`,  
divide by zero

# Which abstract do you prefer?

## Abstract (Take 1)

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

# Which abstract do you prefer?

## Abstract (Take 1)

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

## Abstract (Take 2)

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.



# Which abstract do you prefer?

## Abstract (Take 1)

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

## Abstract (Take 2)

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.

There are obvious, undisputed exceptions

# Making experiments parameterizable

This is huge

# Making experiments parameterizable

This is **huge**

Parameters your code may depend on:

- credentials (OS, database, other)
- values of important environment variables (usually one or two)
- various paths and directories (see: environment variables)
- where the input comes from
- switches (pre-process, optimize, prune, materialize, plot ...)
- where the output goes

# Making experiments parameterizable

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

$$f_1 = v_1, f_2 = v_2, \dots, f_k = v_k$$

# Making experiments parameterizable

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

$$f_1 = v_1, f_2 = v_2, \dots, f_k = v_k$$

Many tricks. Very simple ones:

- `argc / argv`: specific to each class' `main`
- Configuration files
- Java Properties pattern
- + command-line arguments

# Making experiments parameterizable

## Configuration files

Omnipresent 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

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

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

# Making experiments parameterizable

## Java util.Properties

Flexible management of parameters for Java projects  
Defaults + overriding

How does it go:

- Properties extends Hashtable
- Properties is a map of (key, value) string pairs
  - `{ "dataDir", "./data" } { "doStore", "true" }`
- Methods:
  - `getProperty(String s)`
  - `setProperty(String s1, String s2)`
  - `load(InputStream is)`
  - `store(OutputStream os, String comments)`
  - `loadFromXML(...), storeToXML(...)`

# Using java.util.Properties

## One possible usage

```
class Parameters{
    Properties prop;
    String[] [] defaults = {{"dataDir", "./data"},
                            {"doStore", "true"} };
    void init(){
        prop = new Properties();
        for (int i = 0; i < defaults.length; i ++){
            prop.put(defaults[i][0], defaults[i][1]);
        }
    }
    void set(String s, String v){ prop.put(s, v); }
    String get(String s){
        // error if prop is null!
        return prop.get(s);}
}
```



# Using `java.util.Properties`

When the code starts, it calls `Parameters.init()`, loading the defaults

The defaults may be overridden later from the code by calling `set`

The properties are accessible to all the code

The properties are stored in **one place**

Simple serialization/deserialization mechanisms may be used instead of constant defaults

## Better init method

```
class Parameters{
    Properties prop;
    ...
    void init(){
        prop = new Properties();
        for (int i = 0; i < defaults.length; i ++){
            prop.put(defaults[i][0], defaults[i][1]);
            Properties sysProps = System.getProperties();
            // copy sysProps into (over) prop! }
        }
    }
}
```

Call with:

```
java -DdataDir=./test -DdoStore=false pack.AnyClass
```

# Making your code 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

# Making your code 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.
```

# 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.

# 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.

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 generated graphs

You have:

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

# Automatically generated graphs

## You have:

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

## You need: graphs

Most frequently used solutions:

- Based on Gnuplot
- Based on Excel or OpenOffice clone

Other solutions: R; Matlab (remember portability)



# Automatically generating graphs with Gnuplot

- 1 Data file results-m1-n5.csv:

1	1234
2	2467
3	4623

# Automatically generating graphs with Gnuplot

- 1 Data file `results-m1-n5.csv`:

1	1234
2	2467
3	4623

- 2 Gnuplot command file `plot-m1-n5.gnu` for plotting this graph:

# Automatically generating graphs with Gnuplot

- 1 Data file results-m1-n5.csv:

1	1234
2	2467
3	4623

- 2 Gnuplot command file plot-m1-n5.gnu for plotting 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"
```

# Automatically generating graphs with Gnuplot

- 1 Data file results-m1-n5.csv:

1	1234
2	2467
3	4623

- 2 Gnuplot command file plot-m1-n5.gnu for plotting 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"
```

- 3 Call gnuplot plot-m1-n5.gnu

# Automatically producing graphs with Excel

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

A	B	C
1	Scale factor	Execution time
2	...	...
3	...	...

# Automatically producing graphs with Excel

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

A	B	C
1	Scale factor	Execution time
2	...	...
3	...	...

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

# Automatically producing graphs with Excel

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

A	B	C
1	Scale factor	Execution time
2	...	...
3	...	...

- 2 Insert in the area B2-C3 a **link** to the file `results-m1-n5.csv`
- 3 Create in the `.xls` file a graph out of the cells A1:B3, chose the layout, colors etc.

# Automatically producing graphs with Excel

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

A	B	C
1	Scale factor	Execution time
2	...	...
3	...	...

- 2 Insert in the area B2-C3 a **link** to the file `results-m1-n5.csv`
- 3 Create in the `.xls` file a graph out of the cells A1:B3, chose the layout, colors etc.
- 4 **When** the `.csv` file will be created, the graph is automatically filled in.



You may omit working like this:

In `avgs.out`, the first 15 lines correspond to `xyzT`, the next 15 lines correspond to `xYZT`, the next 15 lines correspond to `Xyzt`, the next 15 lines correspond to `xyZT`, the next 15 lines correspond to `XyzT`, the next 15 lines correspond to `XYZT`, and the next 15 lines correspond to `XyZT`. In each of these sets of 15, the numbers correspond to queries 1.1,1.2,1.3,1.4,2.1,2.2,2.3,2.4,3.1,3.2,3.3,3.4,4.1,4.2, and 4.3.

You may omit working like this:

In avgs.out, the first 15 lines correspond to xyzT, the next 15 lines correspond to xYZT, the next 15 lines correspond to Xyzt, the next 15 lines correspond to xyZT, the next 15 lines correspond to XyzT, the next 15 lines correspond to XYZT, and the next 15 lines correspond to XyZT. In each of these sets of 15, the numbers correspond to queries 1.1,1.2,1.3,1.4,2.1,2.2,2.3,2.4,3.1,3.2,3.3,3.4,4.1,4.2,and 4.3.

... either because you want to do clean work, or because you don't want this to happen:

# Why you should take care to generate your own graphs

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

a	b
1	13.666
2	15
3	12.3333
4	13

# Why you should take care to generate your own graphs

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

a	b
1	13.666
2	15
3	12.3333
4	13

Copy-paste into OpenOffice 2.3.0-6.11-fc8:

a	b
1	13666
2	15
3	123333
4	13

# Why you should take care to generate your own graphs

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

a	b
1	13.666
2	15
3	12.3333
4	13

Copy-paste into OpenOffice 2.3.0-6.11-fc8:

a	b
1	13666
2	15
3	123333
4	13

The graph doesn't look good :-)

# Why you should take care to generate your own graphs

File `avgs.out` contains average times over three runs: ( '.' decimals)

a	b
1	13.666
2	15
3	12.3333
4	13

Copy-paste into OpenOffice 2.3.0-6.11-fc8: (expecting ',' decimals)

a	b
1	13666
2	15
3	123333
4	13

The graph doesn't look good :-)

Hard to figure out when you have to produce by hand 20 such graphs and most of them look OK

# Documenting your experiment suite

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

Specify:

- 1 What the installation requires; how to install
- 2 For each experiment
  - 1 Extra installation if any
  - 2 Script to run
  - 3 Where to look for the graph

# Documenting your experiment suite

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

Specify:

- 1 What the installation requires; how to install
- 2 For each experiment
  - 1 Extra installation if any
  - 2 Script to run
  - 3 Where to look for the graph
  - 4 **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*.