Fairness of Time Constraints

Martin Mareš
mj@ucw.cz

Charles University in Prague
Faculty of Math and Physics
Department of Applied Mathematics

Introduction – resource limits and fairness

Many programming contests employ automatic evaluation:

- Programs are run on batches of test data.
- Correctness of output is checked.
- Resource limits are enforced, e.g.:
  - execution time
  - memory consumption
  - communication complexity, ...

To ensure fairness, we want repeatable results. Problems:

- Non-deterministic programs
- Precision of measuring resource consumption, *especially execution time*

Measuring execution time

We still use timing techniques of the 1980's.
However, the PC's have radically changed since that time.

Sources of errors:

- **Multi-processing OS** – context switches
- **Cache hierarchy** – aliased, flushed on context switches
- **Multiple cores** – movement between cores
- **Non-uniform memory** – memory speed varies
- **Power management** – unexpected changes of CPU speed
- **System management mode** – even the OS loses control

Problems with accuracy still resurface (e.g., [Merry 2010]).

Test setup – IOI 2009 tasks

Testing three tasks from IOI 2009 (Plovdiv, Bulgaria):

- **Raisins**
  - batch task, DP, small inputs
  - model solution in time $\mathcal{O}(n^5)$, memory $\mathcal{O}(n^4)$
- **Mecho**
  - batch task, large inputs
  - model solution in time $\mathcal{O}(n \log n)$, memory $\mathcal{O}(n)$
- **Regions**
  - interactive task, data structure
  - judge (not adaptive, so we can run it as a batch task, too)
  - about 10^5 queries/test.
  - model solution: time $\mathcal{O}(n^{1/2})$ per query, memory $\mathcal{O}(n)$

We used real submissions from the contest and real test data (or a subset of).

Test setup – machines

Machines used for our tests:

- **Camellia** – AMD Athlon64 X2, 2 cores at 1 GHz, 2 GB RAM
- **Turing** – Intel Core2 Quad, 4 cores at 2 GHz, 4 GB RAM
- **Corbu** – Intel Core i7, 4 cores at 2.66 GHz, 6 GB RAM
- **Arcikam** – AMD Opteron 2218, 2 CPUs with 2 cores each, 2.5 GHz, 8 GB RAM in 2 NUMA nodes

All machines run Debian Linux 5.0 (Lenny) with kernel 2.6.37.

We used our contest environment (Moe), only the sandbox is interesting.

Error distribution for batch tasks – type 1

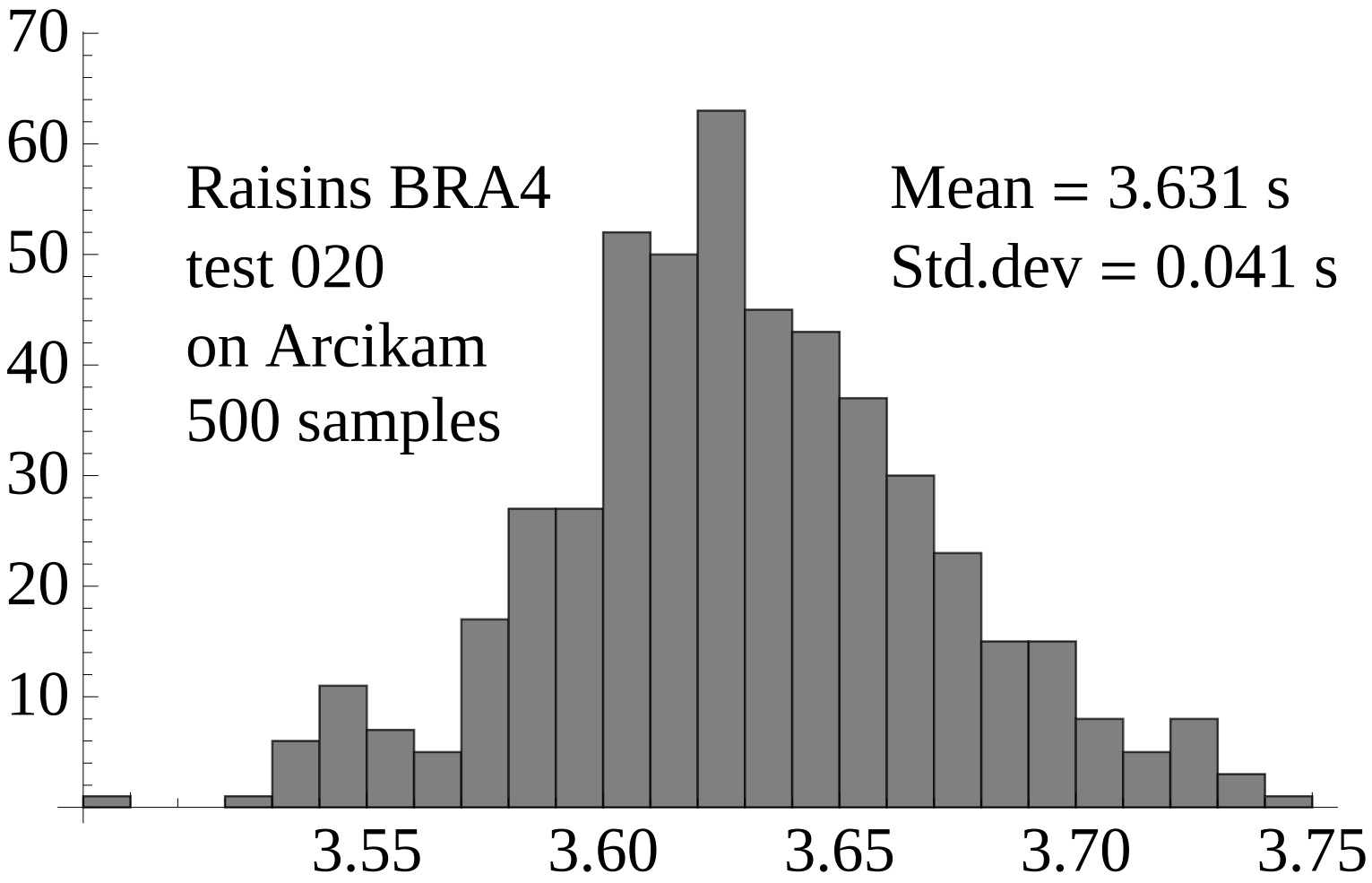


Figure: Histogram of time [s] spent on a single test, type 1

Error distribution for batch tasks – type 2

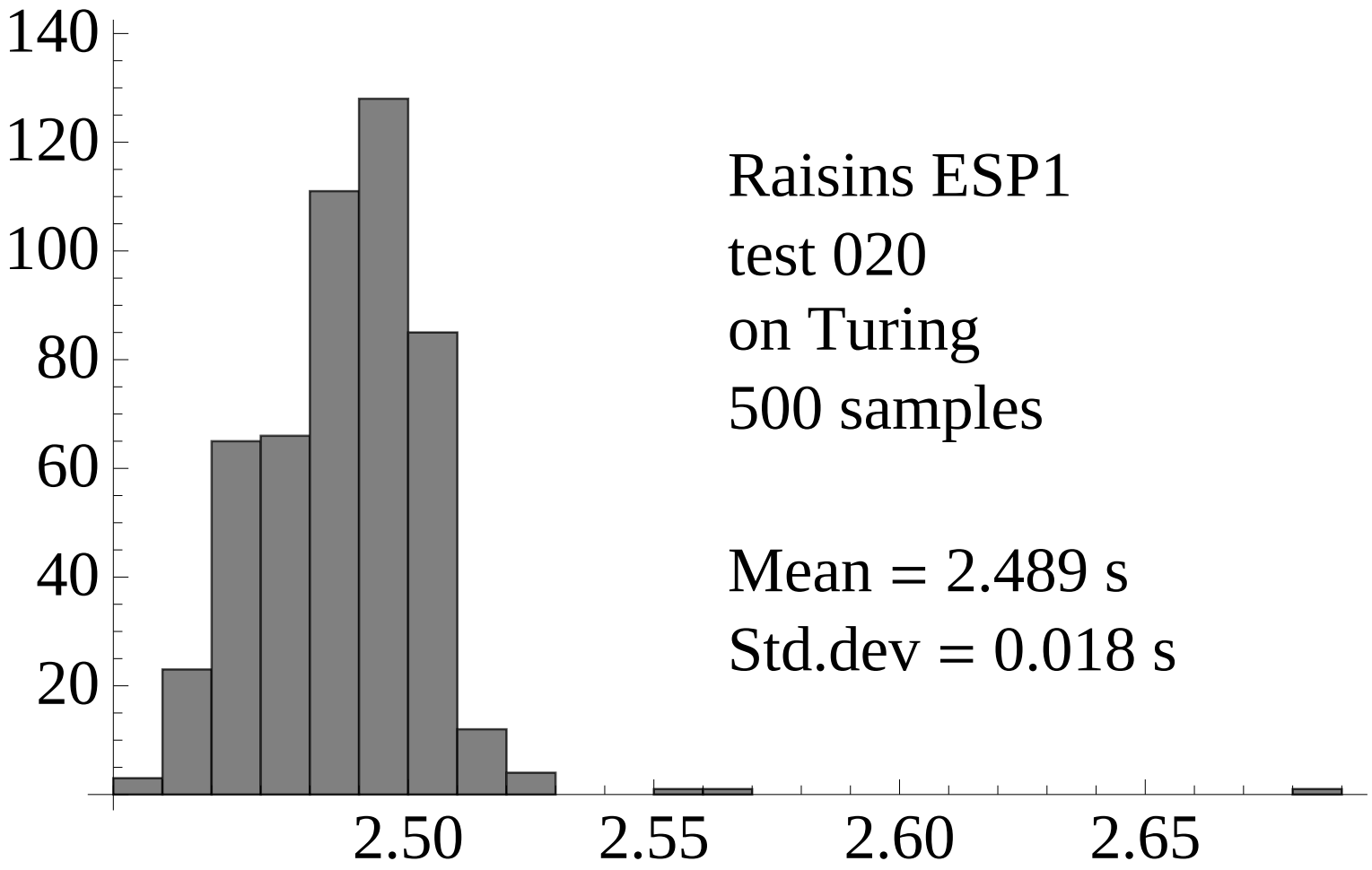


Figure: Histogram of time [s] spent on a single test, type 2

Error distribution – is it Gaussian?

Gaussian (normal) distribution expected, is it so?

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
Arcikam	0.694	0.938	0.998	1.000	1.000	1.000
Turing	0.740	0.982	0.994	0.996	0.998	0.998
Gaussian	0.683	0.955	0.997	0.999	0.999	1.000

Table: Probability of values within k-times standard deviation

Type 1 is close to Gaussian (further tests confirm that), Type 2 has a much longer tail.

Error source – absolute or relative?

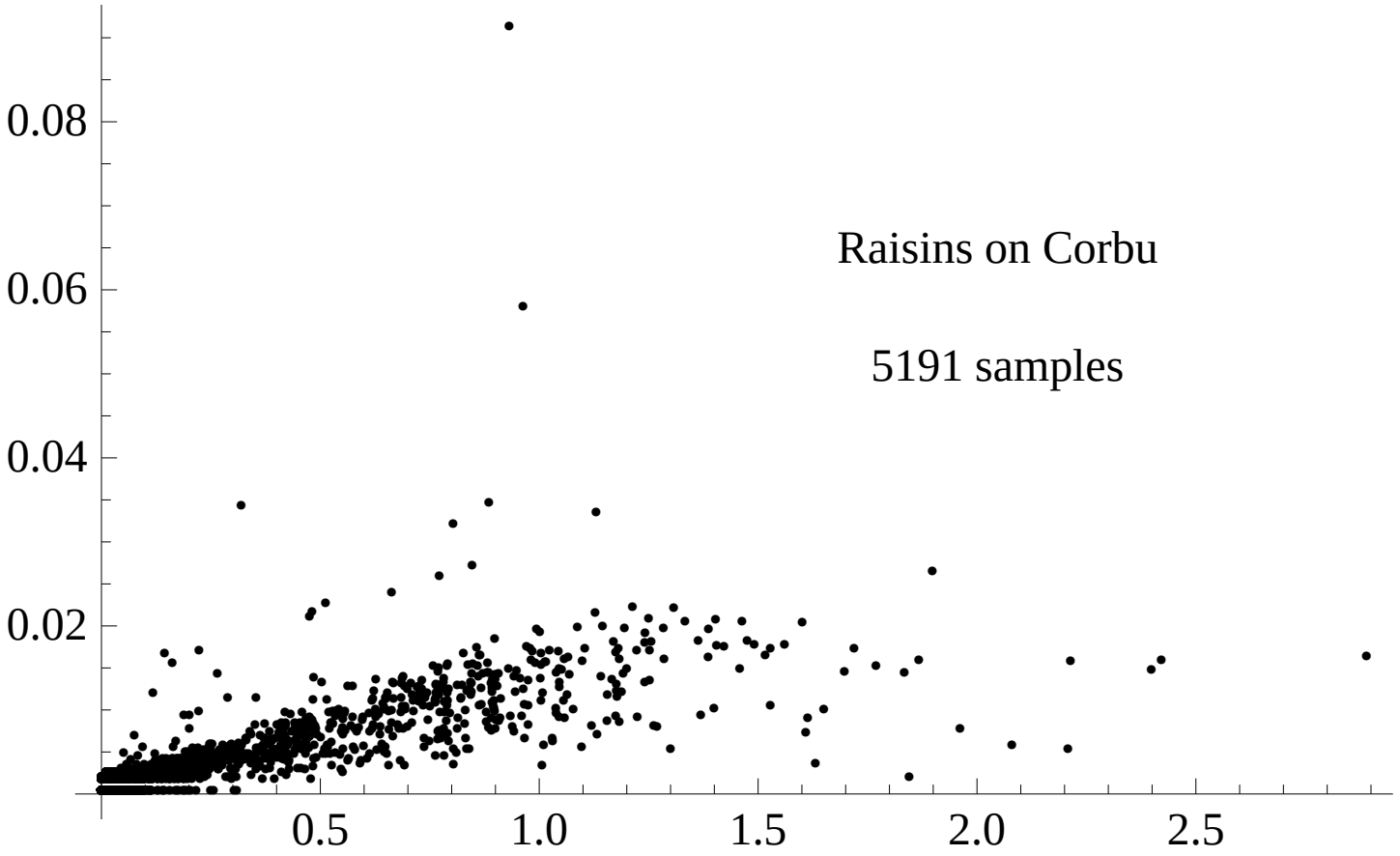


Figure: Dependence between mean and standard deviation [both s]

Indeed, linear regression gives a good fit with small variance.
(Power management can contribute an additive term.)

Tuning the scheduler – experiment setup

We want to test:

- **K** – default settings
- **C** – real-time process with high priority
- **R** – . . . and pinning to a fixed core
- **X** – **C** with no syscall interception
- **T** – **K** with an older kernel (2.6.32)

Our experiment:

- 5 well-behaved submissions
- 3 large test cases
- each instance run 50 times (alternating instances)
- each instance normalized by dividing by reference time: take **K**, drop outliers (2 at each side), use mean of the rest

Tuning the scheduler – results

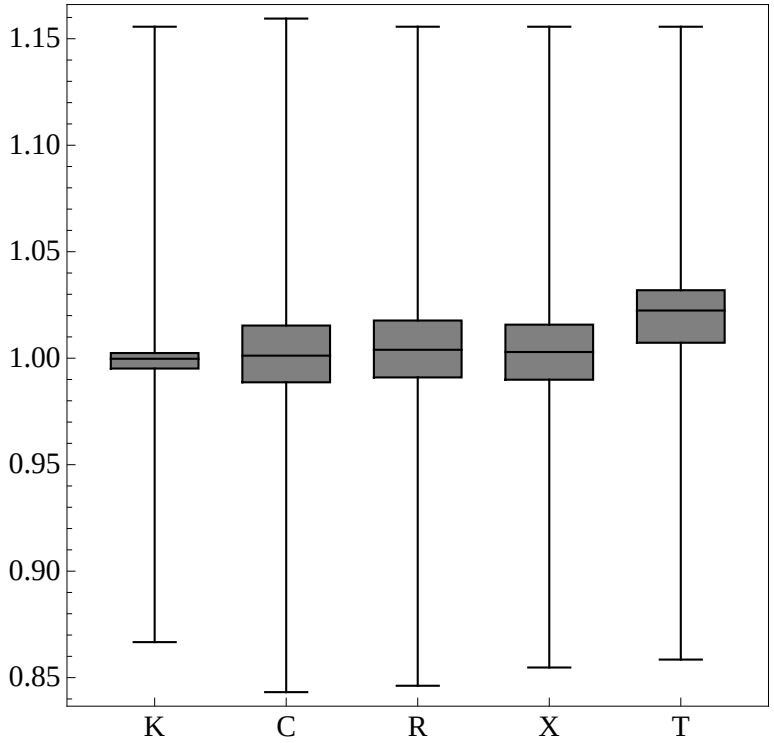


Figure: Raisins on Corbu: quantiles

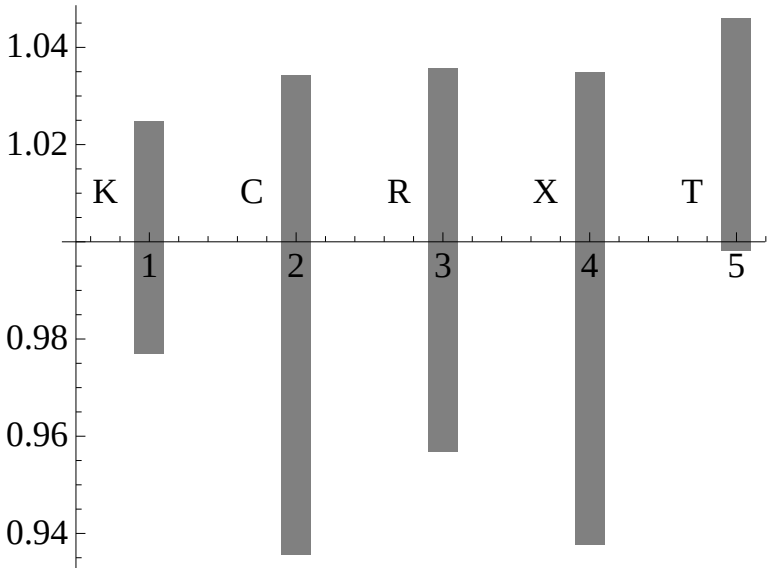


Figure: Raisins on Corbu: mean and standard deviation

Cheat sheet: **K** base, **C** real-time, **R** . . . with pinning, **X** no syscall checks, **T** older kernel.

Parallel grading is tricky – 4-core machine (Corbu)

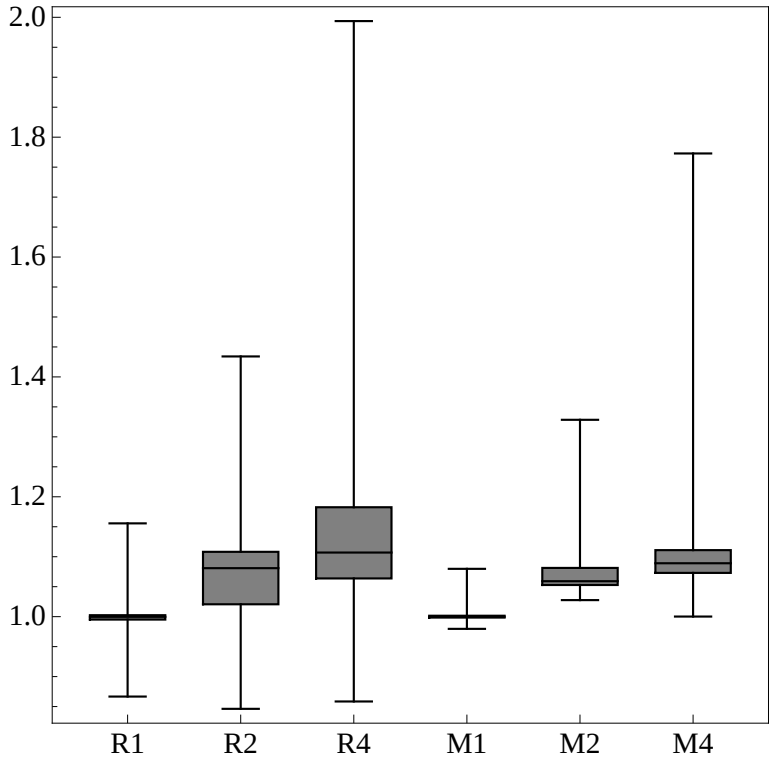


Figure: Parallel grading: quantiles

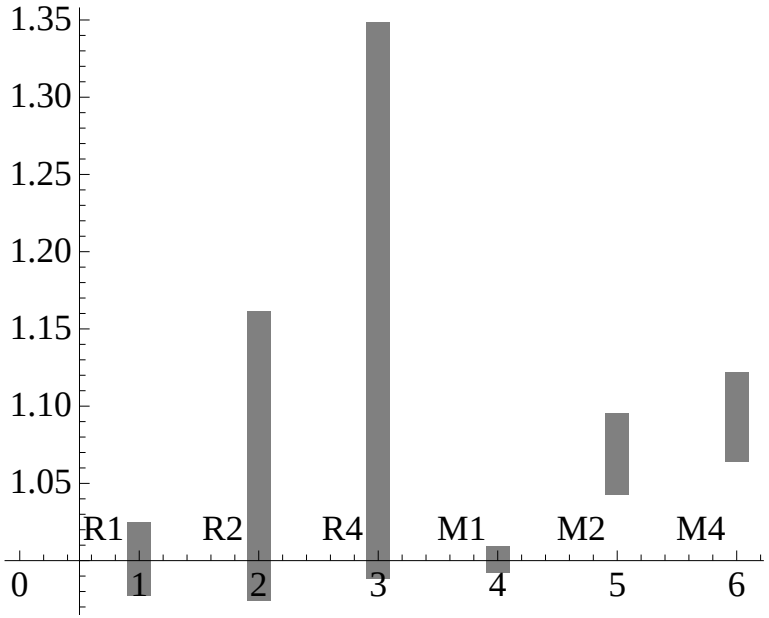


Figure: Parallel grading: mean and deviation

R is Raisins, **M** is Mecho, normalized against **R1**/**M1**.

Parallel grading is tricky – NUMA (Arcikam)

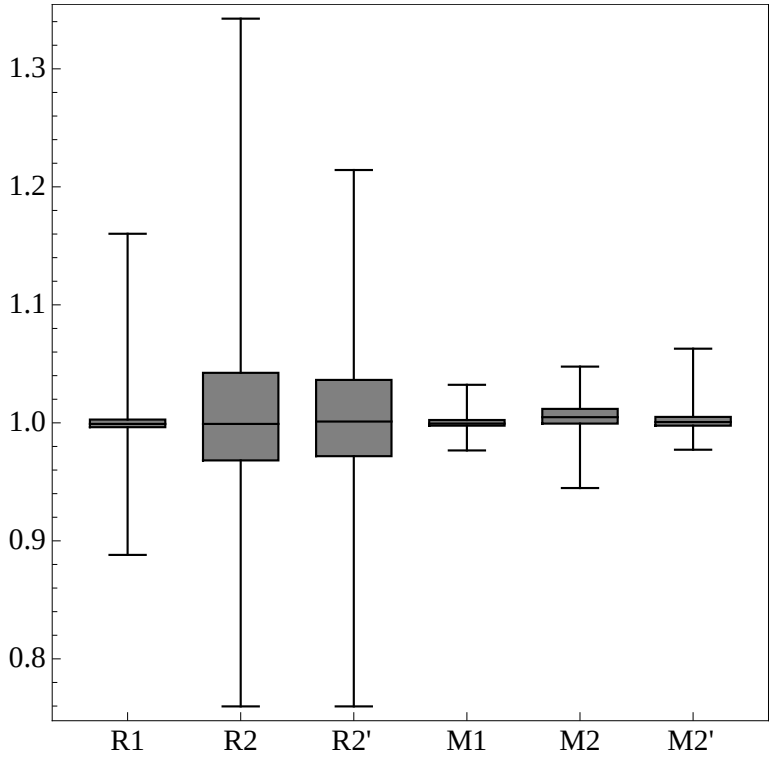


Figure: Parallel grading on NUMA: quantiles

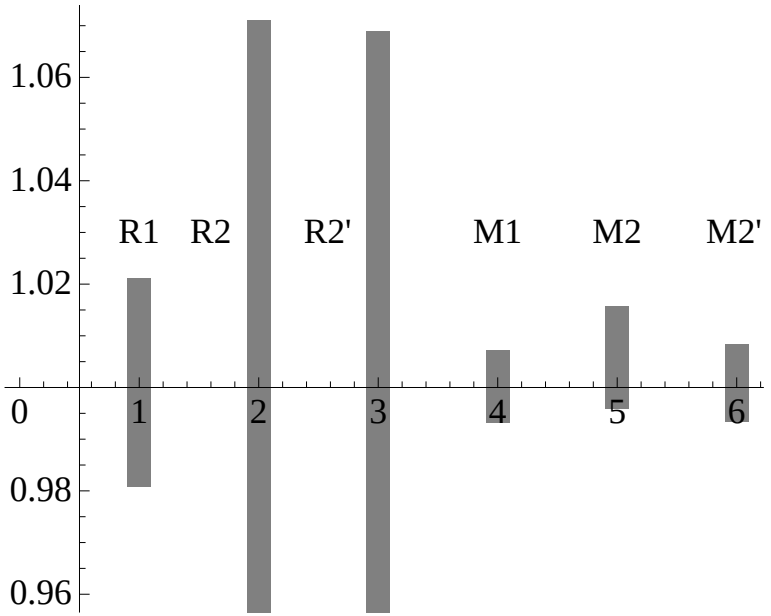


Figure: Parallel grading on NUMA: mean and deviation

R is Raisins, **M** is Mecho, **R2'**/**M2'** uses node pinning.

Batch tasks – conclusions

What we have learned for batch tasks:

- Variance of measured time is acceptably low.
- The scheduler works fairly well, tuning makes things worse.
- Sandbox overhead is reasonably low.
- Execution time must be regarded as a random variable. Repeat measurements *if near the threshold.*

Parallel grading:

- Acceptable variance for #graders $>$ #cores.
- Pinning improves the situation somewhat.

Interactive tasks – problems

What about interactive tasks?

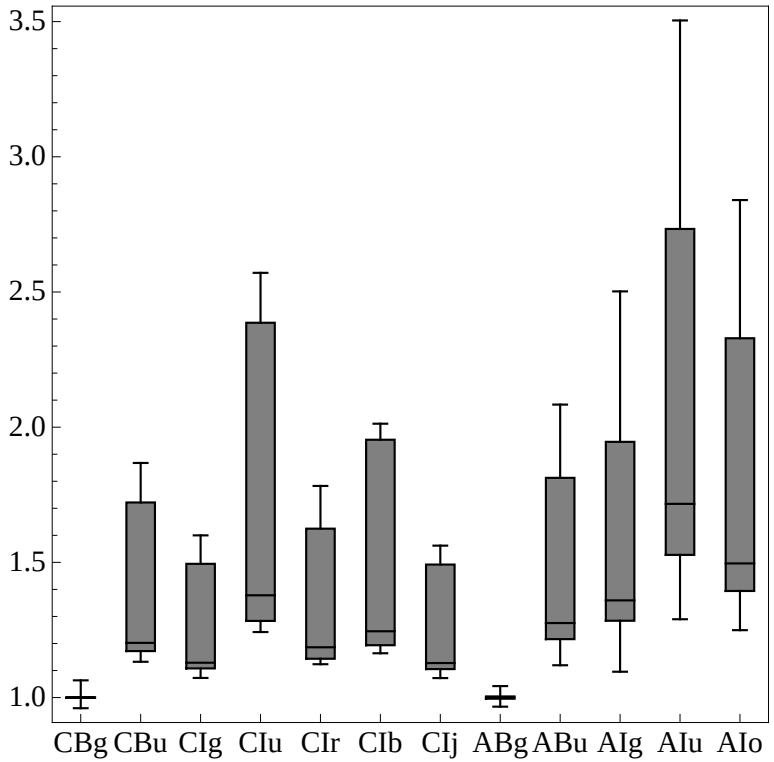
Regions – complex interaction between:

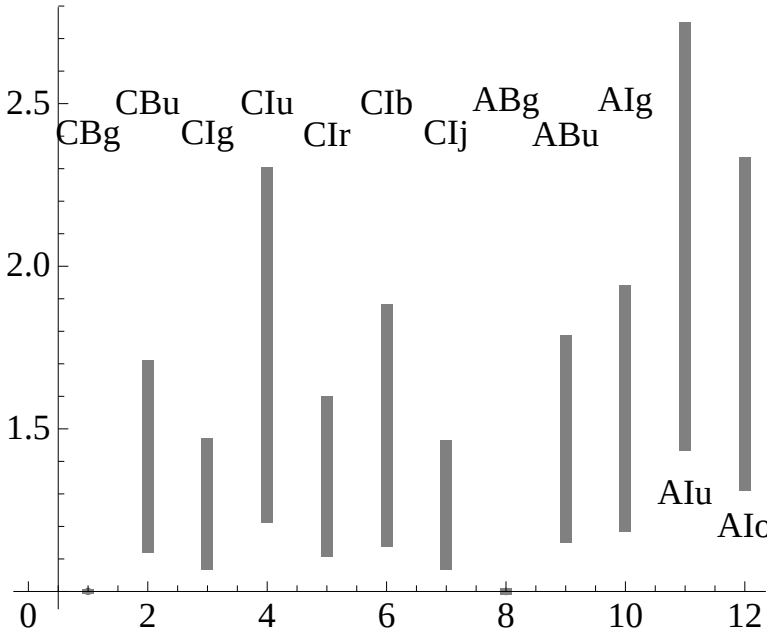
- contestant's solution
- judge
- sandbox

Each operation involves several context switches.

Bruce Merry [2010] compared performance under different sandboxes (also on Regions). Wanted: more insight.

Interactive tasks – experiments





C on Camellia, **A** on Arcikam, **B** is batch, **I** is interactive,
u base, **b** . . . all on 1 core, **r** . . . real-time, **g** no sandbox,
j is **g** pinned to 2 cores, **o** is **u** pinned to a NUMA node.

Interactive tasks – a surprising histogram

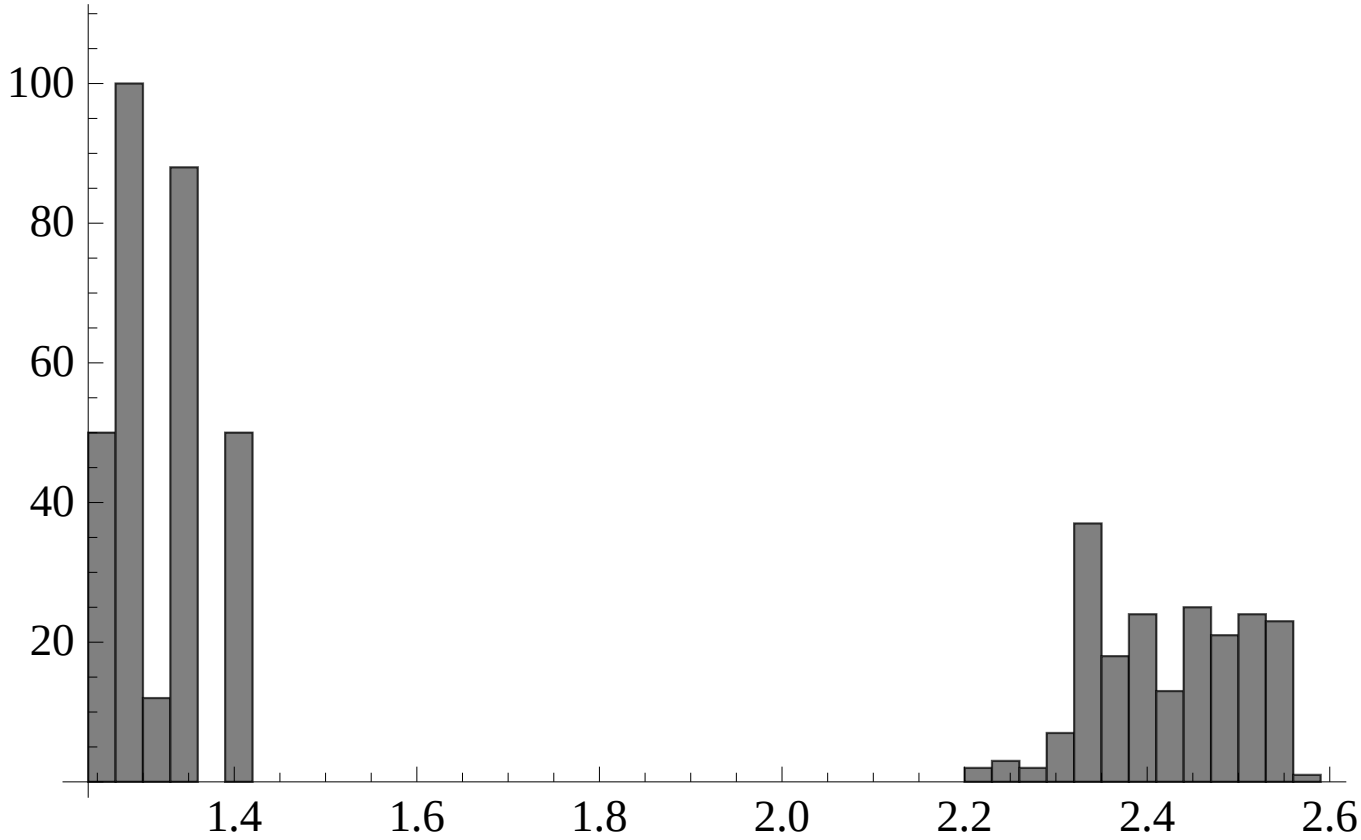


Figure: Histogram of the test **Clu**

Indeed, different test cases behave in a very different way.
So far, we do not have any theory explaining why.

Interactive tasks – different normalization

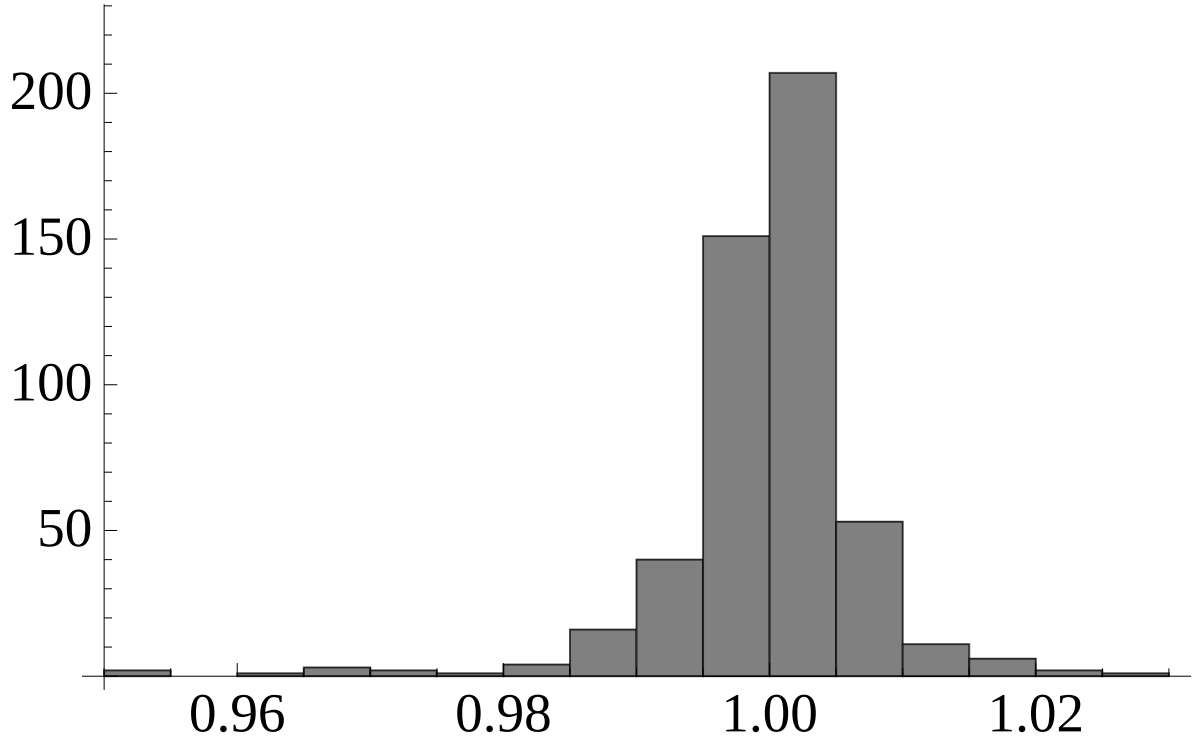


Figure: Test **Clu** normalized with respect to itself

Surprise again: Distribution similar to batch tasks.

Interactive tasks – conclusions

What we have learned:

- Compared to batch version of the problem, interactive version does not give realistic timing.
- The sandbox does not contribute much to the variance.
- Pinning to cores or to a NUMA node helps.
- When we do not seek comparison with the batch version, the interactive version can be considered fair.
- Results differ from [Merry 2010], very likely related to kernel improvements (2.6.30 vs. 2.6.37).
- Still, we recommend avoiding interaction of this type when the number of exchanges is large. Possible solutions: shared memory.

Good bye

Send comments and suggestions to

mj@ucw.cz