Coarse-Grained Simulation for Resource Management of Distributed Systems

Millian Poquet

2022-04-27

Study distributed systems and applications

Many distributed systems in use today or tomorrow (HPC, Clouds, Edge, Fog. . .)

Resource management for many issues (energy, fault tolerance, scheduling, scalability, heterogeneity. . .)

Methodological experimental approaches

- Direct experimentation (real applications on real platforms)
- Simulation (application prototypes on platforms models)
- Something in between (emulation, partial simulation. . .)

Building simulators from scratch is risky

How useful is a simulator whose results cannot be trusted?

- Models validated?
- Implementation tested?
- Model instantiation evaluated?

Doing it thoroughly may take (dozens of) years!

Using a validated simulation framework helps a lot

- Thoroughly validated models
- Thoroughly tested implementation
- Model instantiation responsibility is still on you

Promising simulation framework for resource management?

Convenient API but bad models (PeerSim, GridSim, CloudSim. . .)

- No hope to observe complex phenomena

Packet-level network simulators (NS-3, INSEE. . .)

- Fine granularity → does not scale for concurrent jobs / large systems
- Usable for special cases — *e.g.*, interference-free placements [PML15]
- No model for other resources (CPUs, storages. . .)

Flow-level versatile simulator (SimGrid)

- Tunable granularity, scales
- Models for main types of resources (network, CPUs, storages)
- Power consumption models based on resource usage

Table of Contents

Overview

Simulation framework around distributed platforms and applications

Main use cases

- Prototype systems or algorithms
- Evaluate various platform topologies/configurations
- Study existing distributed app (create digital twin)

Key features

- Sound/accurate models: theoretically and experimentally evaluated
- Scalable: fast models and implementations
- Usable: LGPL, linux/mac/windows, C++ Python and Java

Overview (2)

Numbers

- Exists since early 2001, development still very active
- $\approx$ 200k lines of C/C++ code
- $\approx$ 32k commits
- Used in at least 532 scientific articles

Community

- 4 main developers
- Many power users (current/previous PhD. students...)
- Get help easily (documentation, mattermost, mailing list...)
- Your contributions can be merged

Architecture

How to build your simulator?

- Use one of the SimGrid interfaces
- Link the SimGrid library with your code

Available interfaces

- **S4U** write your own simulator (actors, messages), C++ C or Python
- MSG older brother of S4U, C or Java
- MC verify properties on your application *model* (model is code)
- SMPI `smpicc`/`smpirun` on your real MPI app
- **RSG** emulate distributed memory apps (S4U-like API)
- **Batsim** study resource management (higher-level)

Platform and network models

Platform = graph of hosts and links

Hosts: computational resources

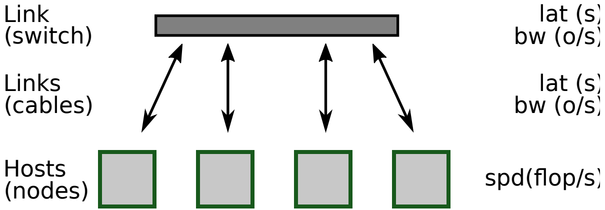
- Speed (FLOP per second)

Links: network resources
(cables, switches, routers...)

- Latency (seconds)
- Bandwidth (bytes per second)

Several network models available

- Fast flow-level: slow start, TCP congestion, cross-traffic
- Constant time: a bit faster (unrealistic)
- Packet-level: NS-3 binding



Actors, computations and communications

Actors

- One of the simulation *actors* — AKA agent, thread, process...
- Executes user-given code on a Host
- User-given code may contain SimGrid calls

Main SimGrid calls

- Compute x flops on current host
- Send x bytes to an actor/host/mailbox
- Yield (just interrupt control flow)

S4U simulator example (Python)

```python
from simgrid import Actor, Engine, Host, this_actor

def sleeper():
    this_actor.info("Sleeper started")
    this_actor.sleep_for(1)
    this_actor.info("I'm done. See you!")

def master():
    this_actor.execute(64)
    actor = Actor.create("sleeper", Host.current(), sleeper)
    this_actor.info("Join sleeper (timeout 2)")
    actor.join(2)

if __name__ == '__main__':
    e = Engine(sys.argv)
    e.load_platform(sys.argv[1])
    Actor.create("master", Host.by_name("Tremblay"), master)
    e.run()
```

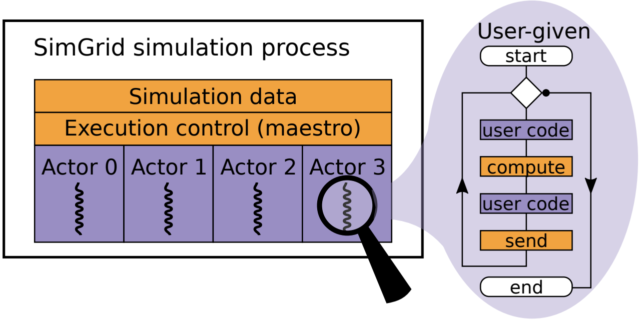
Actor execution model

Main points

- mutual exclusion on actors
- *maestro* dictates who run (deterministic)
- SG calls ≈ syscalls
  - interruption points inside user-given functions

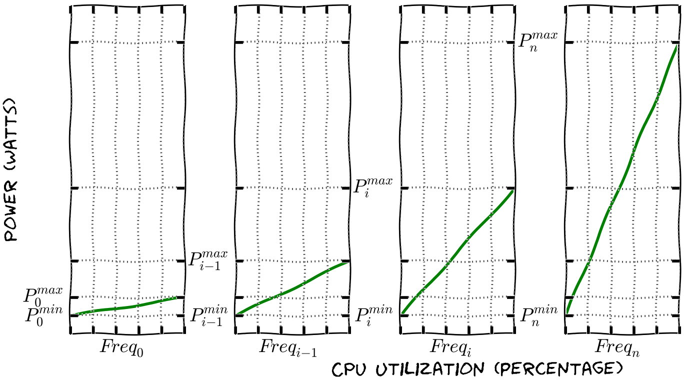
Various implementations

- pthread: *easy* debug, slow
- asm: blazing fast
- ucontext, boost context...



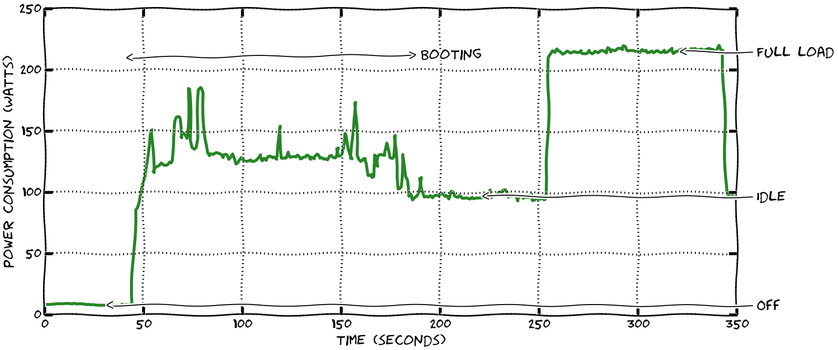
Energy model (DVFS)

- Resources have *power states* (DVFS)
- SimGrid: Manually switch pstates, which change the flop rate
- For one pstate, consumption = linear function of CPU use (+ idle jump)



Energy model (ON/OFF)

ON $\leftrightarrow$ OFF takes time (seconds) and energy (Joules)



- Not easy for the noise: everybody wants something specific
- SimGrid provides basic mechanisms, you have to help yourself
- Switching ON/OFF is instantaneous

Table of Contents

Overview

Resource management simulator built on top of SimGrid

Main use cases

- Analyze and compare online resource management algorithms
- Workload/platform dimensioning

Key features

- Prototype scheduling algorithms in any programming language
- Or use real schedulers (done on OAR and K8s, prototypes for flux/slurm)
- Several job models (tunable level of realism) without deep SimGrid knowledge

Overview (2)

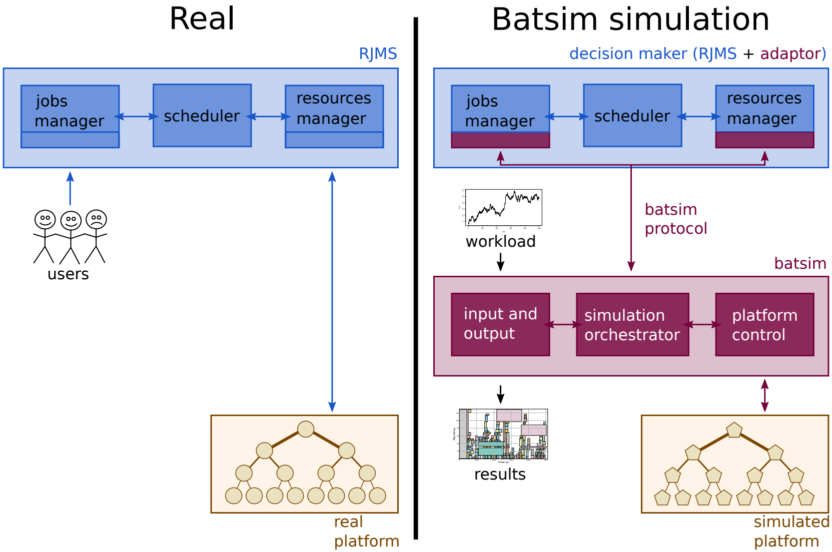
Numbers

- Exists since 2015
- $\approx$ 9k lines of C++ code
- $\approx$ 2k commits

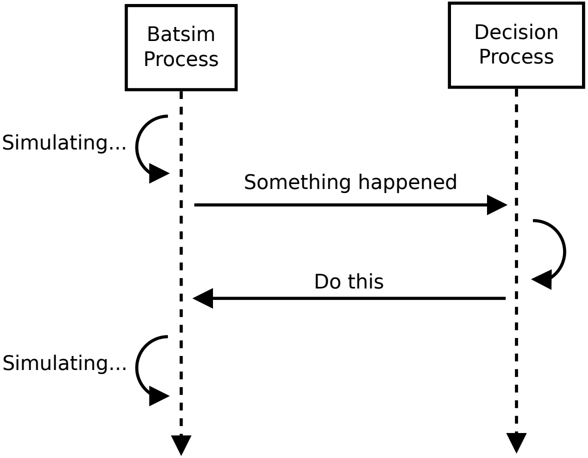
Community

- 1-2 main developers at the same time
- Get help easily (documentation, mattermost, mailing list)
- Users are mostly from scientific labs (international), companies

Architecture



Protocol



Classical scheduling events

- Job submitted
- Job finished

Resource management decisions

- Execute job j on $M = \{1, 2\}$
- Shutdown $M = \{3, ..., 5\}$

Simulation/monitoring control

- Call scheduler at $t = 120$
- How much energy used?
- How much data moved?

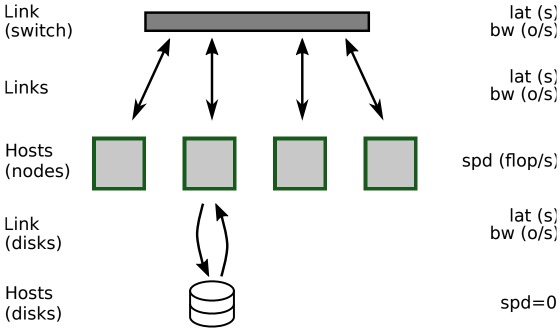
Platform

SimGrid platform + some sugar

RJMS internals on *master* host

Disks modeled as speed=0 hosts

- Enables parallel task use



Jobs and profiles

Jobs : scheduler view

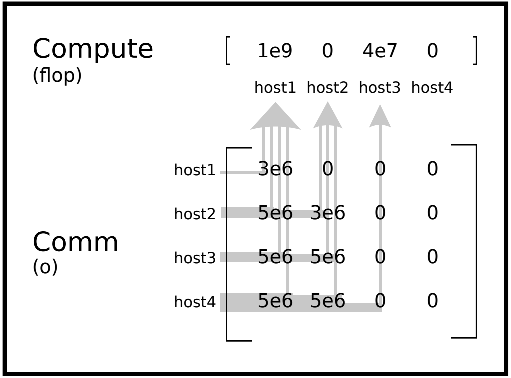
- User resource request
- (Walltime)
- Simulation profile

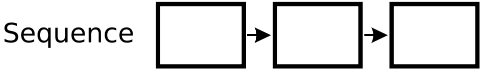
Profiles : simulator view

- How to simulate the app?

Profile types

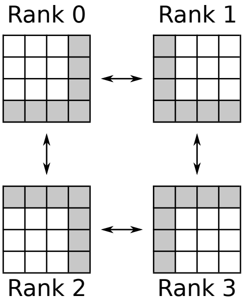
- Fixed length
- Parallel task
- Trace replay (MPI...)
- Composition (seq., parallel)
- Convenient shortcuts
  - IO transfers (alone)
  - IO transfers (along task)





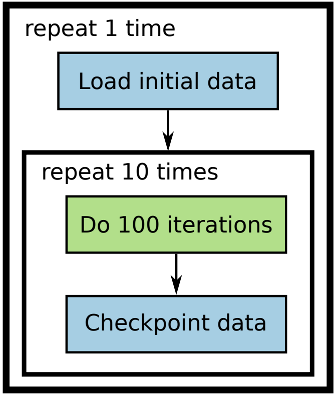
Application model example: Stencil with checkpoints

1. Loads data from parallel filesystem
2. Iteration: local computations, exchange data with neighbors
3. Every 100 iterations: dump checkpoint on parallel file system
4. Stop after 1000 iterations.



Profile example

- Bundle 100 iterations in 1 parallel task



Application model example: Stencil with checkpoints (code)

```
{ "initial_load": {
    "type": "parallel_homogeneous_pfs",
    "bytes_to_read": 67108864,
    "bytes_to_write": 0,
    "storage": "pfs" },
  "100_iterations": {
    "type": "parallel",
    "cpu": [   1e9,     1e9,     1e9,     1e9],
    "com": [     0, 819200, 819200,      0,
            819200,      0,      0, 819200,
            819200,      0,      0, 819200,
                 0, 819200, 819200,      0] },
  "checkpoint": {
    "type": "parallel_homogeneous_pfs",
    "bytes_to_read": 0,
    "bytes_to_write": 67108864,
    "storage": "pfs" },
  "iterations_and_checkpoints": {
    "type": "composed",
    "repeat": 10,
    "seq": ["100_iterations", "checkpoint"] },
  "imaginary_stencil": {
    "type": "composed",
    "repeat": 1,
    "seq": ["initial_load", "iterations_and_checkpoints"] }
}
```

Ecosystem and Usage

Ecosystem

- Set of scheduling algorithms (C++, Python, Rust, D, Perl...)
- Tools to generate platforms and workloads
- (Interactive) tools to visualize/analyze Batsim results
- Tools to help experiments (environment control, execution...)

Already used to study

- Online scheduling heuristics
- Energy/temperature management
- Use of Machine Learning in scheduling
- Big data / HPC convergence (best effort Spark jobs within HPC cluster) with distributed file system (HDFS)
- Evolving jobs with parallel file system + burst buffers
- Impact of user behaviors
- Fault tolerance

Table of Contents

Profile evaluation from Batsim initial paper[1]

Experiment

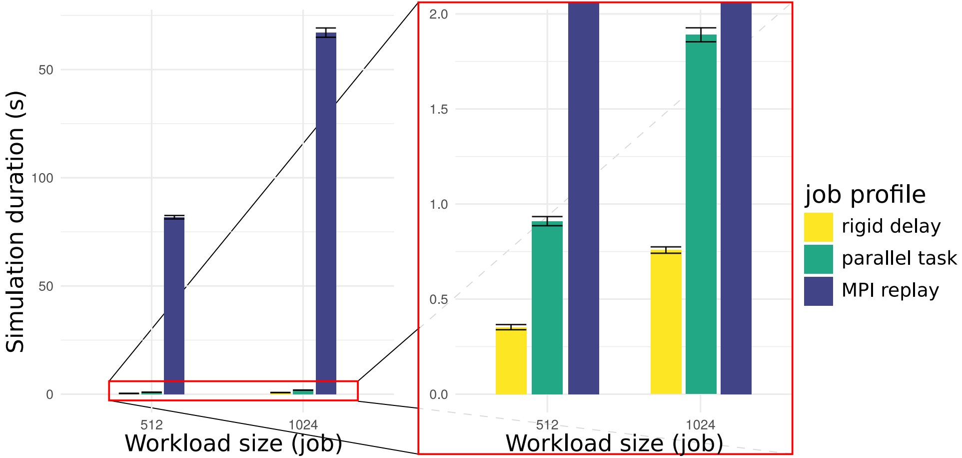
- Execute workloads with Batsim and on Grid'5000 (OAR)
- Same scheduler implementation (conservative backfilling)
- 9 synthetic workloads (4h each)
- Apps from NAS Parallel Benchmarks (IS, FT, LU), various sizes/classes
- Job profiles generated from app instrumentation
- Compare Gantt charts & scheduling objectives

Conclusions

- Real $\approx$ simulated for all profiles (delay, ptask, MPI replay)
- **Observed no interference** (network capacity $>$ workload needs)

[1] Pierre-François Dutot et al. "Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator". In: *Job Scheduling Strategies for Parallel Processing*. 2015.

Performance per profile type (2 synthetic workloads)



Reproduce repo. https://gitlab.inria.fr/adfaure/ptask_tit_eval

Profile types comparison

What performance/accuracy trade-off?

Rigid delay

- Very fast
- Context-free
- Rarely useful for apps (dynamic injection)

Parallel task

- Fast enough!
- Coarse-grained interf.
- Versatile & convenient
- Not validated yet

MPI trace replay

- Much slower
- Fine-grained interf.
- MPI only
- Validated predictions [CGS15]

Profile types comparison

What performance/accuracy trade-off?

Rigid delay

- Very fast
- Context-free
- Rarely useful for apps (dynamic injection)

Parallel task

- Fast enough!
- Coarse-grained interf.
- Versatile & convenient
- Not validated yet

MPI trace replay

- Much slower
- Fine-grained interf.
- MPI only
- Validated predictions [CGS15]

- Agregate MPI traces → huge accuracy drop, almost no performance gain :(

Profile types comparison

What performance/accuracy trade-off?

Rigid delay

- Very fast
- Context-free
- Rarely useful for apps (dynamic injection)

Parallel task

- Fast enough!
- Coarse-grained interf.
- Versatile & convenient
- Not validated yet

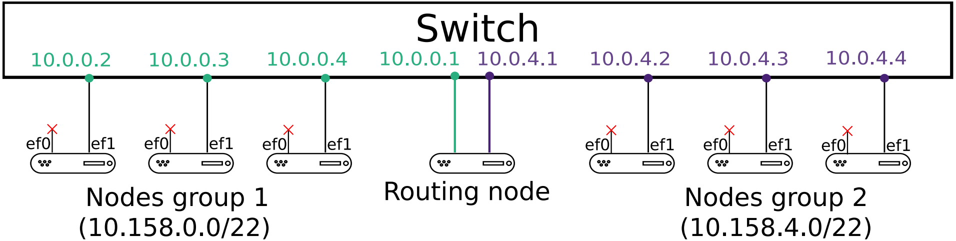
MPI trace replay

- Much slower
- Fine-grained interf.
- MPI only
- Validated predictions [CGS15]

- Agregate MPI traces → huge accuracy drop, almost no performance gain :(
- **Parallel tasks' accuracy needs to be evaluted**

Evaluate parallel tasks — platform setup

Platform network



Overdimensioned network

Need to create a contention point!

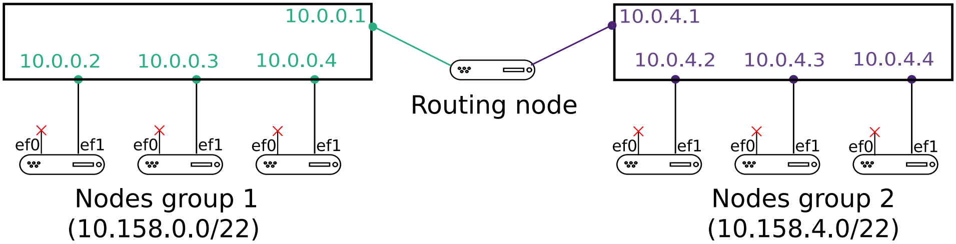
- Split switch into two groups (subnets)
- Inter-group comms via routing node

Grid'5000 platforms

- **Grisou** and **Paravance**
- Same homogeneous machines
- Different switch

Evaluate parallel tasks — platform setup

Reconfigured network



Overdimensioned network

Need to create a contention point!

- Split switch into two groups (subnets)
- Inter-group comms via routing node

Grid'5000 platforms

- **Grisou** and **Paravance**
- Same homogeneous machines
- Different switch

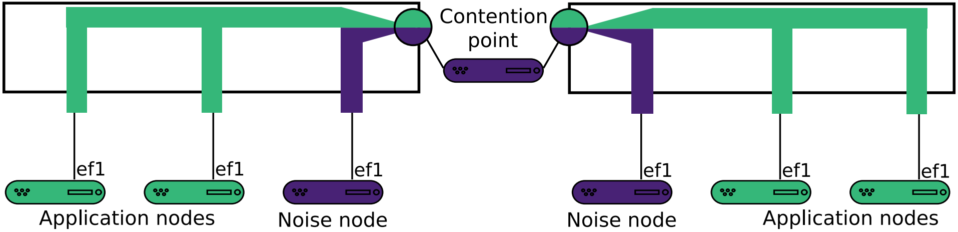
Evaluate parallel tasks — application and noise

Real application (matrix multiplication)

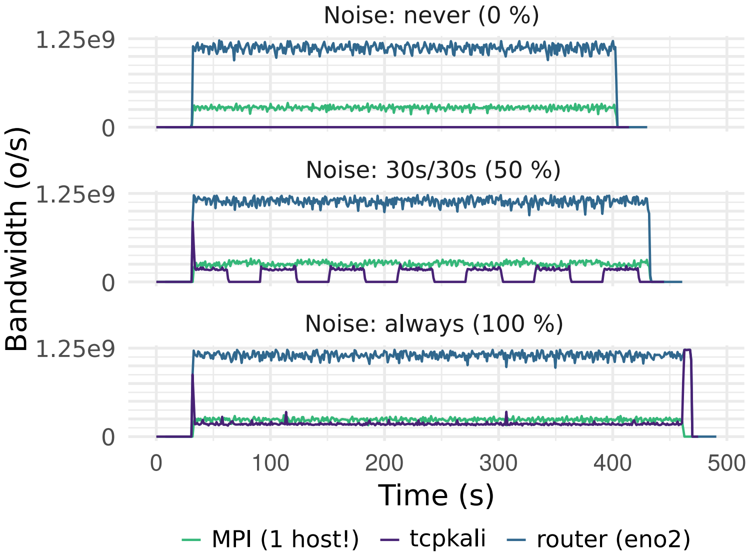
- Matches parallel tasks hypotheses
  - Short compute & comm phases
  - $\rightarrow$ Homogeneous progress
- 8 nodes per group (16 core / node)
- Parameters
  - Block size
  - Sync / Async broadcasts

Noise

- High traffic generation via tcpkali
- 1 node per group
- Periodic ($T = 60\ s$)
  - 0 % noise : 60 s idle
  - 25 % noise : 15 s traffic $\rightarrow$ 45 s idle



Real runs behave as expected



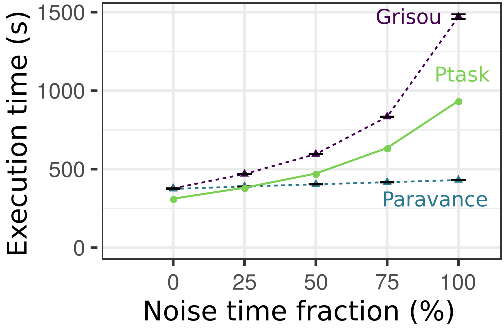
Ptask vs Reality

Results

- Parallel task: 0 % point seems fine
- Parallel task: consistent behavior
- Real: Grisou & Paravance are different

Questions

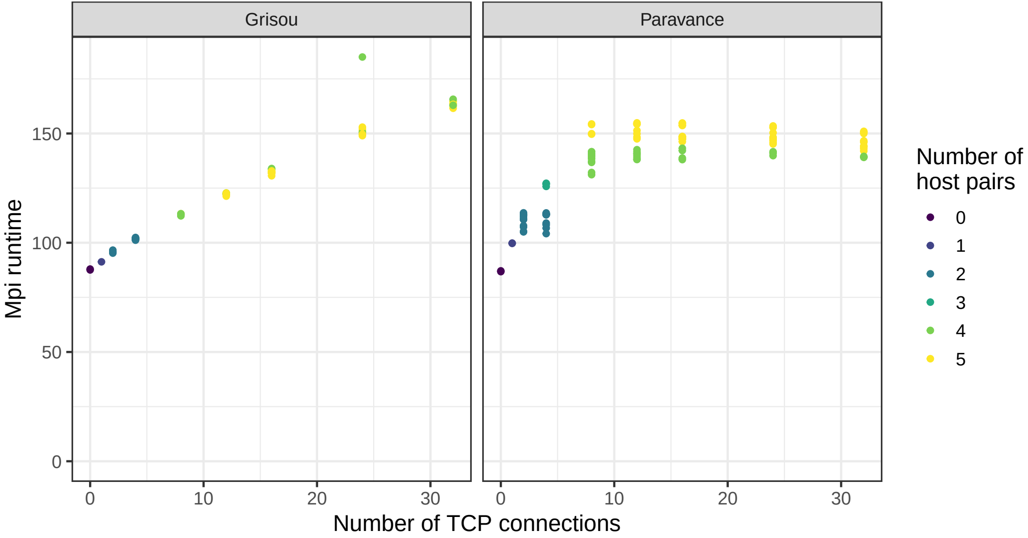
- How to calibrate the 100 % point?
- Why do Grisou & Paravance switches' behavior differs so much?



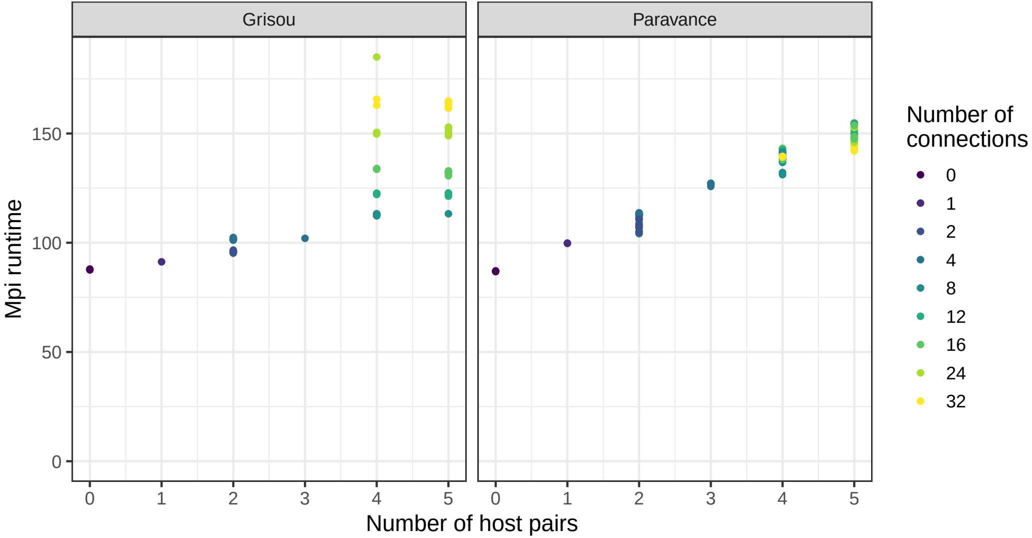
→ **Run another experiment with a more complex noise**

- Noise always active
- 5 nodes per group for the noise
- Many ways to connect noisy nodes together (random graph generation)

Runtime vs Number of connections (real)



Runtime vs Number of pairs (real)



Grisou/Paravance difference explained

Grisou

- App performance correlated with number of TCP connections in noise
- Noise connection location has no effect

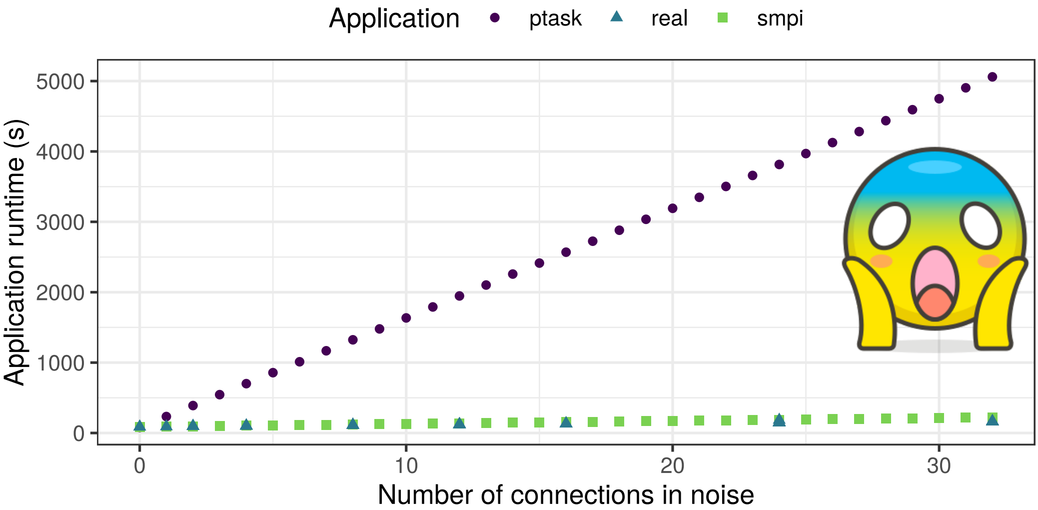
Paravance

- App performance correlated with number of different pairs of hosts in noise

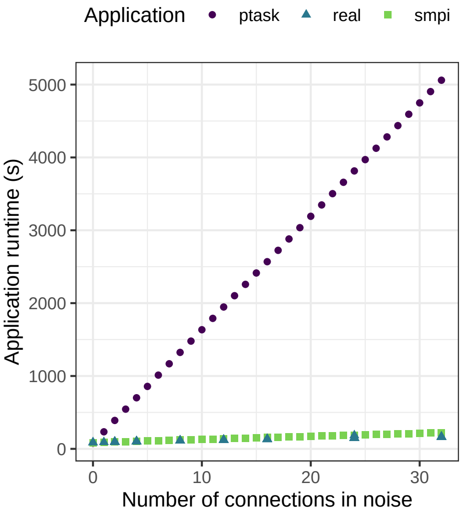
Conclusions

- Switches have a different sharing policy
- SimGrid: Fair sharing among TCP connections regardless of their source/destination
- $\rightarrow$ **Ignore Paravance for now**

Ptask vs Grisou — varying number of connections in noise



Ptask vs Grisou — varying number of connections in noise



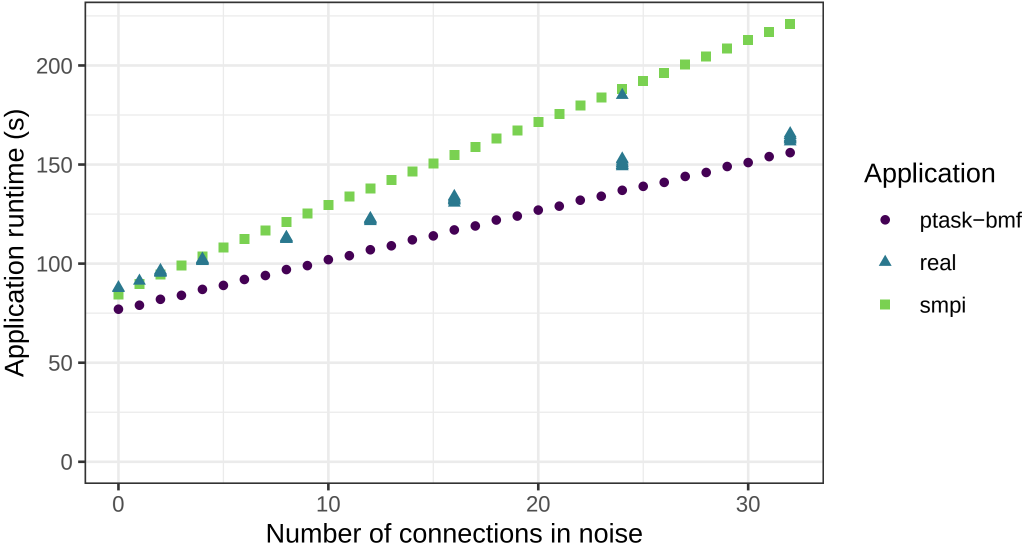
Houston, we have a problem!

- **Huge** overestimation when link saturated by many connections
- Number of connections inside ptasks **ignored** by `ptask_L07`
  - Bad sharing when Big vs Small ptasks
  - No fix in `ptask_L07` (recursive Max-Min Fairness)

→ **New model implementation**

- Bottleck Max Fairness [BR15]

Ptask-BMF vs Grisou — varying number of connections in noise



Take home message

This talk in a nutshell

- SimGrid: sound toolkit to build your simulator
- Batsim: study resource management, tunable profile granularity
- `ptask_bmf`: very promising coarse-grained model

Many questions around `ptask_bmf`

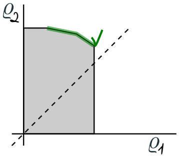
- BMF solution: existence but no uniqueness...
- Termination of fast/greedy solvers?
- Performance overhead?

Batsim

- Validation of **applications** models?
- Ongoing architecture overhaul
  - Single-process simulations
  - Flatbuffers serialization

Appendix

Max-min Fairness



The min function is not strictly increasing so a recursive optimization is needed

- Water-filling [BG87]
  - Allocate ϵ to each flow until a link is saturated ($\sum_i A_{i,j}\epsilon = C_j$)
  - Fix the saturated flows and repeat
- Recursive bottleneck identification
  - For each link j, $\epsilon_j = C_j / \sum A_{i,j}$, consider $\epsilon = \min_j \epsilon_j$
  - Fix the saturated flows, update link capacity, and repeat

Low complexity, gracefully extends to weighted version, exploits the fact that $A_{i,j} \geq 0$

Slide from Arnaud Legrand. https://gitlab.inria.fr/alegrand/slides_fair_sharing

Bottleneck Max Fairness

max-min fairness $\sim$ "bottleneck resources are fairly shared"

- **Axiom** : Every "flow" f has a bottleneck resource j s.t.
  - $\sum_i A_{i,j}\rho_i = C_j$ (the resource is saturated)
  - $A_{f,j}\rho_f = \max_i A_{i,j}\rho_j$ (f is active all the time)
    - $\rightsquigarrow$ Flows with the *same bottleneck* get the *same share*
  - Find $|\mathcal{F}|$ bottlenecks and solve $A'\rho = C'$

It is quite a reasonable choice for *streaming* and *parallel tasks*

Slide from Arnaud Legrand. https://gitlab.inria.fr/alegrand/slides_fair_sharing

References I

[PML15] Jose A Pascual, Jose Miguel-Alonso, and Jose A Lozano. "Locality-aware policies to improve job scheduling on 3D tori". In: *The Journal of Supercomputing* 71.3 (2015), pp. 966–994.

[Dut+15] Pierre-François Dutot et al. "Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator". In: *Job Scheduling Strategies for Parallel Processing*. 2015.

[CGS15] Henri Casanova, Anshul Gupta, and Frédéric Suter. "Toward more scalable off-line simulations of MPI applications". In: *Parallel Processing Letters* 25.03 (2015), p. 1541002.

[BR15] Thomas Bonald and James Roberts. "Multi-resource fairness: Objectives, algorithms and performance". In: *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. 2015, pp. 31–42.

References II

[BG87] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1987.