Simulation of HPC Systems

Martin Quinson, A. Legrand, A. Degomme
and the whole SimGrid Team
(with the expertise our colleagues at UIUC and at BSC)

JLPC/PUF Summer School, June 13th 2014

(How) can we reach Exascale?

Peta-scale is already insanely challenging

- Hardware scale's increase seems endless
- As an answer, the software complexity increases (super-linearly)
- Do we have a chance to understand Exascale Systems ?
- Mont-Blanc's fun approach to Exascale relies on ARM+GPUs+Ethernet
- Need for application performance prediction and capacity planning

Motivation toward Simulation of MPI applications

1. Helping application developers
   - Non-intrusive tracing and repeatable execution
   - Classical debugging tools (gdb, valgrind) can be used
   - Save computing resources (may run on your laptop)
2. Helping application users (provides a baseline for comparison)
3. Capacity planning (can we save on components? what-if analysis)

Flourishing state of the Art

There are many different projects

- Dimemas (BSC, probably one of the earliest)
- PSINS (SDSC, used to rely on Dimemas)
- BigSim (UIUC): BigNetSim or BigFastSim
- LogGopSim (UIUC/ETHZ)
- SST (Sandia Nat. Lab.): Micro or Macro
- SimGrid (Inria, CNRS, U. Lorraine, UCSD, UH)
- ...

This tutorial aims at making you up to speed

- First get an overview of the challenges and existing solutions:
  why you don't want to develop your own simulator
- Then get our hands dirty through practical manipulations:
  how to use my pet project, so that you can then learn any other one

So, we are ready for a 3-hours tutorial!

Learning Objectives

- Understand the trade-offs when designing a simulator of HPC
- Learn how to use one such tool, to get the working concepts

Tutorial Agenda

- Introduction and Motivation
- Simulation of HPC Systems (DIY)
  - Observing the Application
  - Modeling Computations
  - Modeling Communications
  - MPI Operations
  - Recap and Illustration With Different Simulation Projects
- SMPI 101
- Practical Session

Agenda

- Introduction and Motivation
- Simulation of HPC Systems (DIY)
  - Observing the Application
  - Modeling Computations
  - Modeling Communications
  - MPI Operations
  - Recap and Illustration With Different Simulation Projects
- SMPI 101
- Practical Session

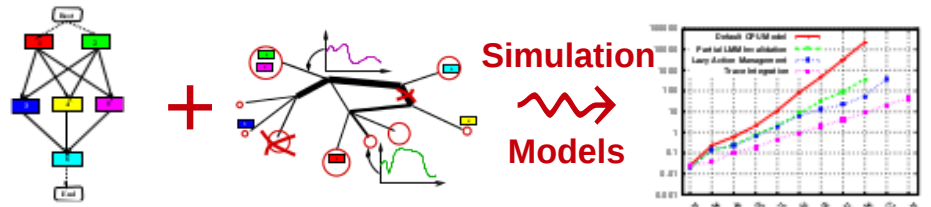
Section Objectives: What would it imply to build your own simulator?

- Understand the components of such a simulator
- Learn about the alternatives for each such component
- Compare the design objectives and internals of major existing projects

Simulation in a Nutshell

Fastest path from idea to data; Easiest way to study distributed apps

- Everything's centralized again: Central state and time; No heisenbug.



Idea or MPI code | Experimental Setup | Scientific Results

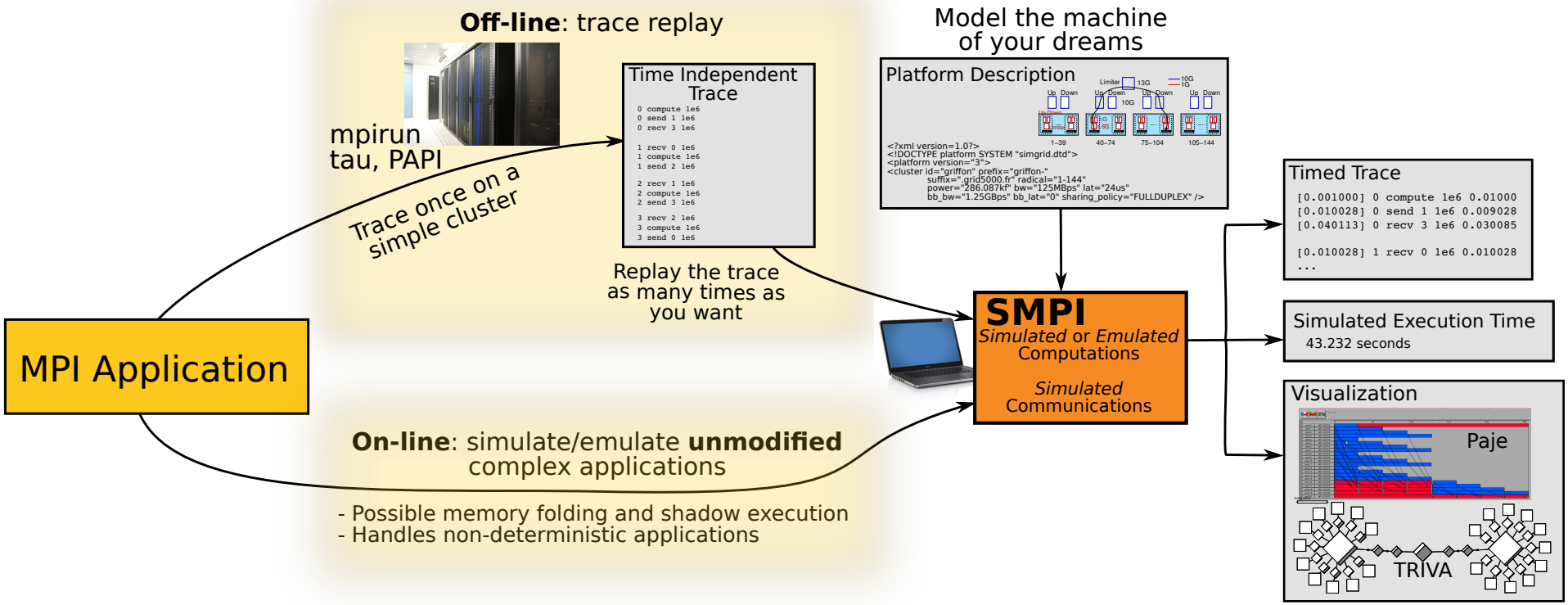
Common Model of MPI Applications

- Interleaving of *Sequential Execution Blocks* with MPI communications
- Many interferences ignored: SEB $\leftrightarrow$ MPI; SEB $\leftrightarrow$ SEB on other cores

Major Components of any Simulation-based Experiment

- An observation of your application: either a trace or the live application
- Models of your platform: CPU, network, any other relevant resource
- A configuration describing the experimental settings

Observing the Application



Offline Simulation

- Obtain a trace of your application
- Replay quickly and easily that trace
- Hard to extrapolate, adaptative apps?

Online Simulation

- Directly run your application
- Technically very challenging
- No limit (but the resources)

Most existing tools go for offline simulation

Challenges in Observing Applications Offline

Many contributions in the literature

- Reduce intrusiveness while capturing the traces to avoid heisenbugs
- Compact the traces (that can grow very quickly)
- Extrapolate the trace to new conditions

Was shown during the tutorial of Sanjay and Juan

- So we don't get into detail again

Challenges in Observing Applications Online (1/2)

HPC codes are resource hungry

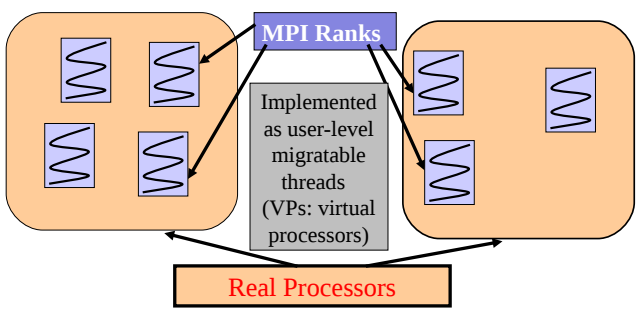
- It does not fit easily on single laptop or node
- Sometimes, host machine must be larger than studied machine
- Some tricks allow to cheat here
  - Memory folding to allocate once, and share between processes
  - Kernel sampling to reduce execution time

Challenges in Observing Applications Online (2/2)

Folding the application is difficult

- Global variables of distributed processes hard to fold into thread locals
  - Manual modification: works but burdensome
  - Source-to-Source: turn globals into arrays of locals
  - Compiler's pass: move globals into TLS area
    changes toolchain (no `icc`) $\rightsquigarrow$ alters SEBs (as any previous solution)
  - GOT injection: rewrite the ELF symbol table when switching contextes
    static variables are not part of the GOT unfortunately
  - `mmap` of `.data` and `.bss`: preserves SEBs but forces sequential exec
  - Run real processes, MPI interactions turned into external `mmap`. Perf?

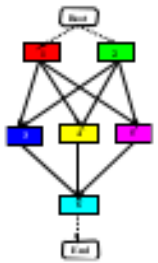
Architecture (in AMPI)



Approaches implemented

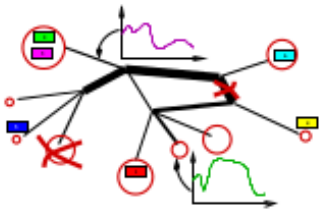
- AMPI: Source-to-source with Photran
  GOT injection; Compiler's pass for TLS
- SMPI: source-to-source (coccinelle, f2c)
  Recently implemented mmaping
- Full processes not implemented yet (?)

Observing the Application was the Easy Part



Idea or
MPI code

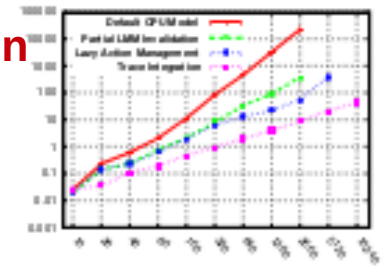
+



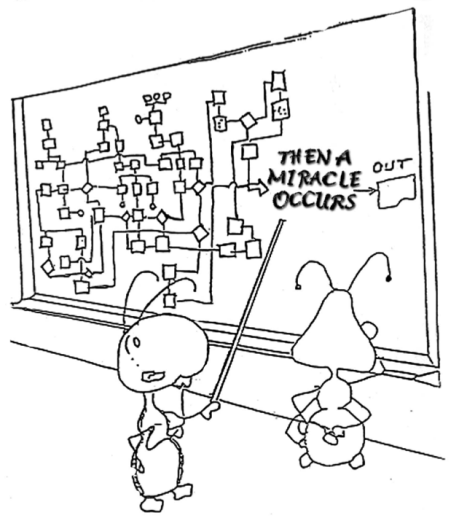
Experimental
Setup

Simulation

Models



Scientific Results



Good Models are very hard to come up with

- CPU and Network resources are mandatory. Disks welcomed.
- Usage model: Predict ending time of each task in isolation
  - On Network, both one hop models, and multi-hops paths
- Contention model: predicts how tasks interfere with each others
  - On Network, needs to take topology (and routing) into account
- Models of complex operations (MPI global communications)

(that's why you don't want to design your own tool :)

Fine-grain Simulation of CPU

Many Cycle-accurate Models and Simulators exist

- We could simulate entirely each core, each node, each site, etc.
- Most resources are modeled separately: cores, buses, networks, disks
- Popular belief: more details means more accurate simulation

we could combine these tools together!

Microscopic Modeling (only) is not an option

- Immensely slow: x1000 slowdown when host machine $\approx$ studied system
  - So folding a larger system into a smaller host machine is impossible
  - This approach is sensible, for other scientific workflows
- More details actually bring more chaos and less insight
  - Complex models are hard to instantiate and fragile (Flash project)
  - Phase effects: clean simulations lead to resonance effects [Floyd 91]
  - *A wealth of information creates a poverty of attention* [Simon 71]
- Mixing *macro* and *micro* models sounds appealing but difficult
  - As done in SST project and also by the BSC group

Simplistic CPU Model

How it works

- Computation load measured in Flops; CPU's power measured in Flops/s
- Timing is obtained by simply dividing one by the other
- Basically, this is just like reinjecting timing.

What is this Model Good for?

- Allows to see what you would get with a CPU twice faster
- Almost every projects does this (SimGrid, Dimemas, . . .)

Known Limits

- Hardware extrapolation to other kind of CPUs, w/ cache contention
  - Dimemas can adjust per SEB; PSINS extrapolates from hardware counters
  - SST mixes Micro (cycle accurate) and Macro models to that extend
- Multicore memory contention (could hack something but haphazard)
- Scalability extrapolation: what would happen with more nodes
  - BigSim can model the SEB perf as a polynomial of #processes
  - PSINS tries to fit a model from the SEB's parameters

Elaborate Analytic CPU Model

The Promise

- Get a bunch of hardware-level counters while benchmarking the SEBs
- Automatically build portable performance models out of it

The (many) Challenges

- Relating the hardware counters you see to the actual timing you get
- You need a performance model taking the counters as an input
  - PSINS has the convolver for that, but hard to get and understand it
  - Our preliminary results: encouraging for some kernels, deceiving for others
- How to obtain the hardware counters?
  - Measurements? SimGrid/Dimemas use PAPI on real runs (hard to extrapolate)
  - Cache simulation? PSINS goes this way
  - Code analysis? Maqao does it
- How generic and portable will the models be?
  - Things are very different e.g. on ARM

Conclusion on CPU modeling

Upcoming complexity is somehow depressing

- Multicores, Implicit Mem Accesses, OpenMP, Memory/ PCI contention
- Modern processors overclock themselves when only one core is used
- GPU, SOC systems, dedicated accelerators

Don't seek for a complete model

- KISS is better, and the advantage of more complex CPU models is unclear
- At least in the use-cases that we target (at our scale)
- We are not competing with cycle-accurate simulators
- So simply refine your simple models, only when the need is blatant

Much more insight can be injected into the Network Models

- Things are very complex too, but maybe less integrated by vendors
- We can work at the level of standard protocols (TCP, InfiniBand)

Agenda

- Introduction and Motivation
- Simulation of HPC Systems (DIY)
  - Observing the Application
  - Modeling Computations
  - Modeling Communications
  - MPI Operations
  - Recap and Illustration With Different Simulation Projects
- SMPI 101
- Practical Session

Components of a good model

- Point to point communications: latency, protocol switch
- Topology: shared memory $\neq$ remote, latency penalty for remote cabinets
- Contention

Fine Grain Network Simulation

Packet-level simulators run the full protocol stack

- Hopefully perfect, since "everything's taken into account"
- But complex models $\rightsquigarrow$ hard to instantiate and unstable

> Flores Lucio, Paredes-Farrera, Jammeh, Fleury, Reed. *Opnet modeler and ns-2: Comparing the accuracy of network simulators for packet-level analysis using a network testbed.* WSEAS Transactions on Computers 2, no. 3 (2003)

- Inherently slow, and parallelism won't save you here!
  BigSim proved that distribution is for size (memory) issues, but sequential is faster
- Sometimes wrongly implemented
- Not really helping to understand the macroscopic behavior

Same bias and drawbacks than cycle-accurate CPU simulation

- Perfectly fitted to study TCP variants or wireless algorithms
- Very bad choice to study MPI algorithms (IMHO)

Modeling Point to Point Networks

Basic Model: $Time = L + \frac{size}{B}$

- Resource work at given *rate* (B, in Mb/s); Uses have a given latency (L, in s)
- Very similar to the basic CPU model (simply adds latency)
- This somehow works for Multi-Hops Networks

Better Model of TCP Multi-Hops Networks

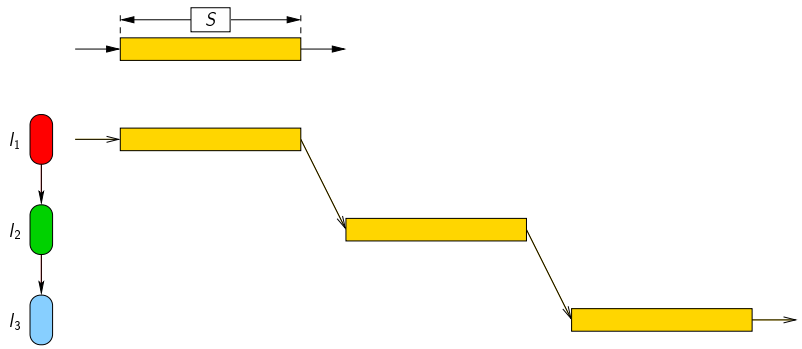
- Several models proposed in Networking Literature, such as [Krusoe 2000]

$$B = min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{2bp/3} + T_0 \times min(1, 3\sqrt{3bp/8}) \times p(1+32p^2)}\right)$$

  - T_0: retransmission timeout; RTT: round-trip t; W_{max} max window size
  - p: loss rate; b: #packages acknowledged per ACK (hard to instanciate)
- Keep It Instanciable, Silly: use $\beta' = min(\beta, \frac{W_{max}}{RTT})$ *(TCP windowing)*

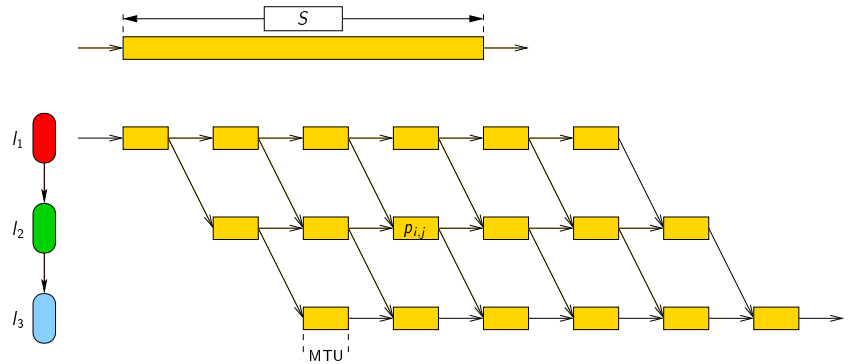
Taking the Network Topology into Account

Store & Forward



- Sounds Natural:
  cf. time to go from city to city
- But Plainly Wrong:
  Data not stored on routers

Wormhole



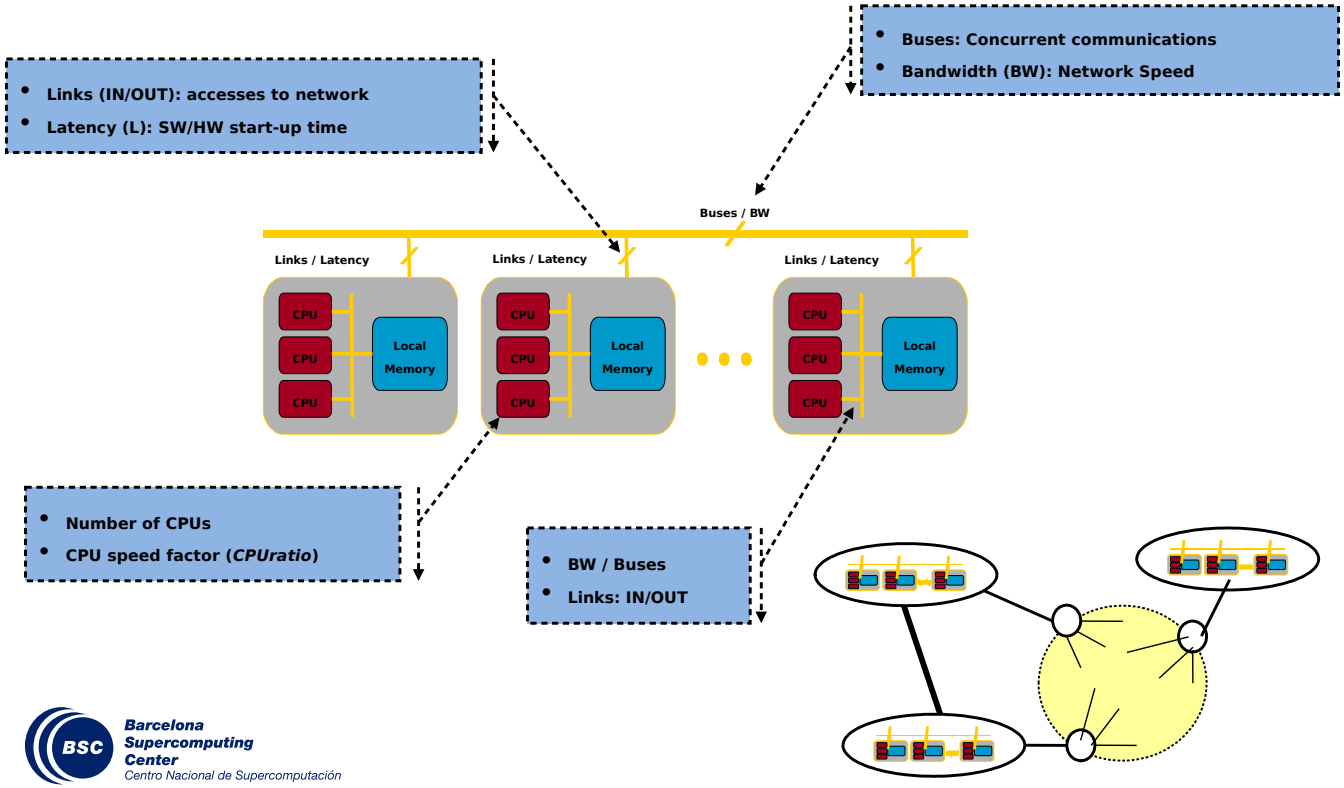
- Appealing: (& widely used ☹)
  Remember networking class?
- Really inaccurate:
  TCP congestion, etc

What's in between these two approaches?

Packet-level Simulators

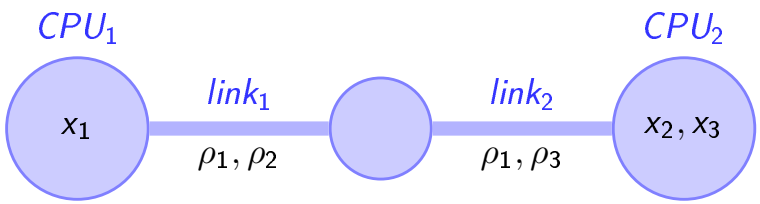
- ☺: Realism commonly accepted; ☹: Slooooooow
- No usable models of HPC networks in generic tools (NS2/3)

Exclusive Resource Usage



- In Dimemas, resources are allocated exclusively with more than one token
- Nicely models buses' backplane: up to N flows get through, others do wait
- Then a delay-model computes the time of each communication
- Applied at each models (memory, networks), with no overlap between both
- Similar mechanism in BigFastSim (?)

Analytic Network Models



$$x_1 \leqslant Power\_CPU_1 \quad (1a)$$
$$x_2 + x_3 \leqslant Power\_CPU_2 \quad (1b)$$
$$\rho_1 + \rho_2 \leqslant Power\_link_1 \quad (1c)$$
$$\rho_1 + \rho_3 \leqslant Power\_link_2 \quad (1d)$$

Computing the sharing between flows

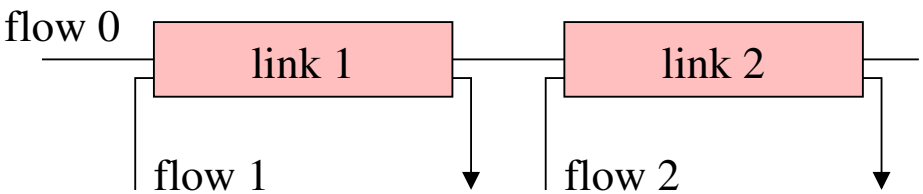
- Objective function: maximize $\min_{f \in \mathcal{F}}(\rho_f)$ [Massoulié & Roberts 2003]
- Equilibrium: increasing any ρ_f decreases a ρ'_f (with $\rho_f > \rho'_f$)
- (actually, that's a simplification of SimGrid's real objective function)

Efficient Algorithm

1. Search for the bottleneck link l so that: $\frac{C_l}{n_l} = min\left\{\frac{C_k}{n_k},\ k \in \mathcal{L}\right\}$
2. This determines any flow f on this link: $\rho_f = \frac{C_l}{n_l}$
3. Update all n_l and C_l to remove these flows; Loop until all ρ_f are fixed

Max-Min Fairness

Homogeneous Linear Network



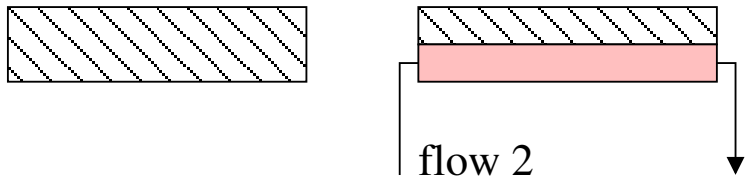
$C_1 = C \qquad n_1 = 2$
$C_2 = C \qquad n_2 = 2$

$\rho_0 =$
$\rho_1 =$
$\rho_2 =$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1.

Max-Min Fairness

Homogeneous Linear Network



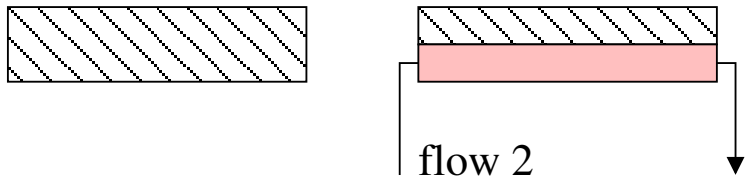
$C_1 = 0 \qquad n_1 = 0$
$C_2 = C/2 \qquad n_2 = 1$

$\rho_0 = C/2$
$\rho_1 = C/2$
$\rho_2 =$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1.
- This sets ρ_0 and ρ_1. Remove flows 0 and 1; Update links' capacity and uses

Max-Min Fairness

Homogeneous Linear Network



$$C_1 = 0 \qquad n_1 = 0$$
$$C_2 = 0 \qquad n_2 = 0$$

$$\rho_0 = C/2$$
$$\rho_1 = C/2$$
$$\rho_2 = C/2$$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1.
- This sets ρ_0 and ρ_1. Remove flows 0 and 1; Update links' capacity and uses
- Link 2 sets $\rho_1 = C/2$.
- We are done computing the bandwidths ρ_i

SimGrid Implementation is **efficient**

- Dedicated LMM solver with Lazy updates, Trace integration, and Cache locality

Flow-level Models Facts

Several sharing methods are possible, many have been evaluated in SimGrid

Pros

- rather flexible (add linear limiters whenever you need one)
- account for network topology
- account for many non-trivial phenomena
  e.g., RTT-unfairness of TCP and even reverse-traffic interference to some extent

Cons

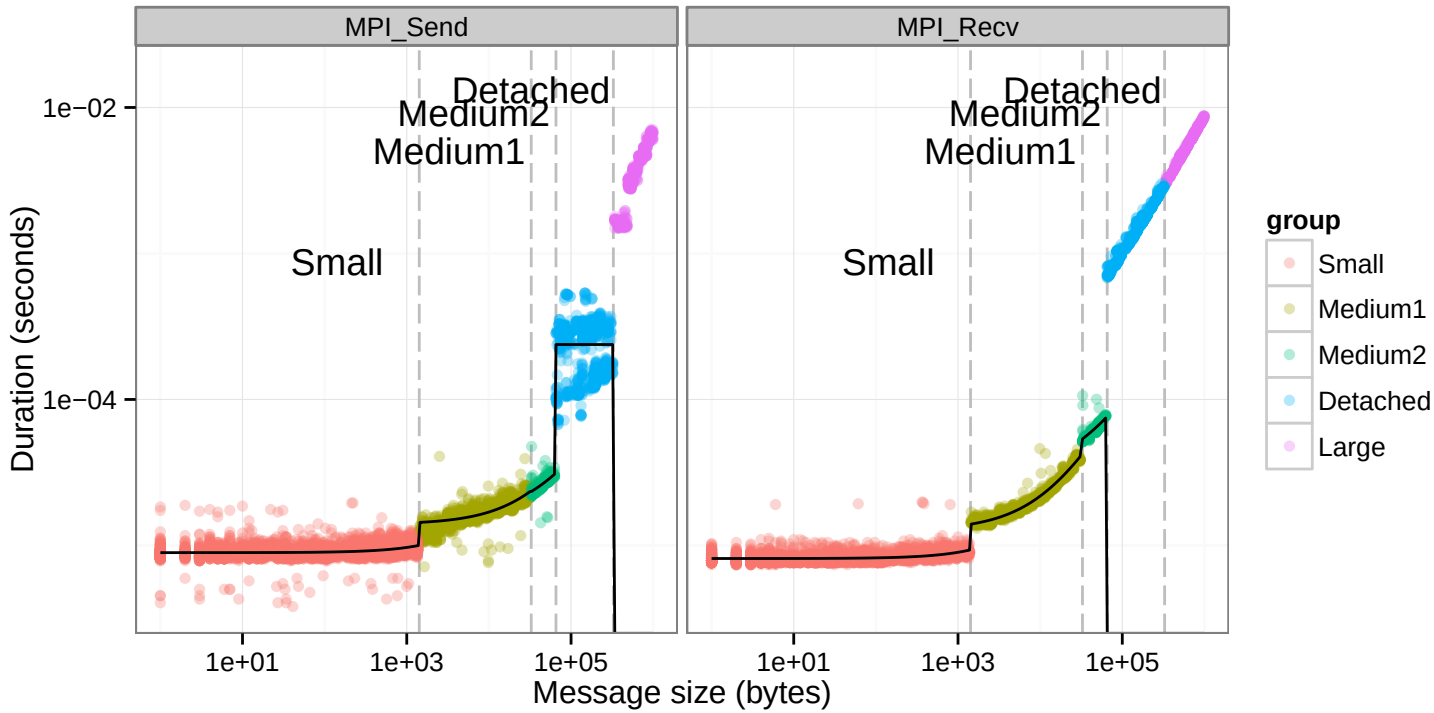
- ignores protocol oscillations, TCP slow start
- ignores all transient phases
- does not model well very unstable situations
- does not model computation/communication overlap

Conclusion

- Common belief: this cannot scale, so often ruled out
- Yet, when correctly implemented and optimized, it's a strong alternative
- Captures contention if TCP is in steady state (when $size > 1Mb$)

MPI Point-to-Point Communication on Ethernet

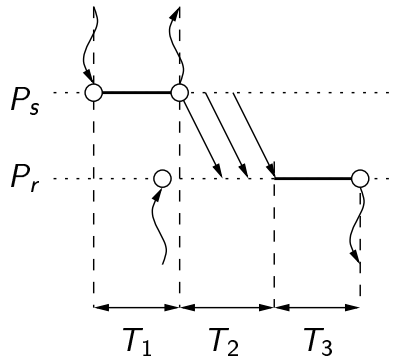
Randomized measurements (OpenMPI/TCP/Eth1GB) since we are not interested in peak performance but in performance characterization



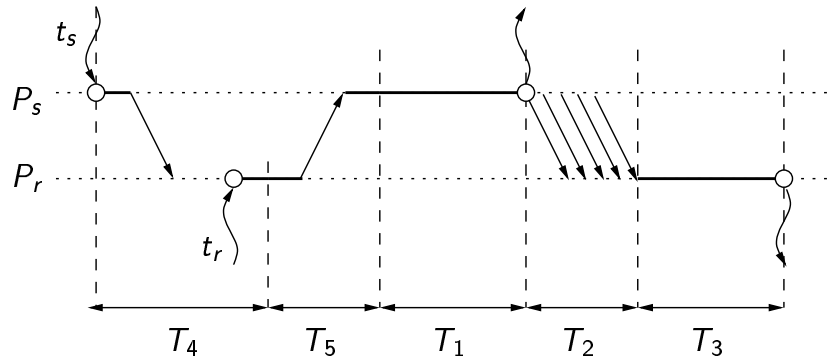
- There is a quite important variability
- There are at least 4 different modes, each is piece-wise linear and discontinuous

LogGPS in a Nutshell

- LogP model initially designed for complexity analysis and algorithm design
- Many variations account for protocol switch through continuous linear functions



Asynchronous mode ($k \leqslant S$)



Rendez-vous mode ($k > S$)

$$T_1 = o + kO_s \qquad T_4 = \max(L + o, t_r - t_s) + o$$

$$T_2 = \begin{cases} L + kg & \text{if } k < \boxed{s} \\ L + sg + (k - s)G & \text{otherwise} \end{cases}$$

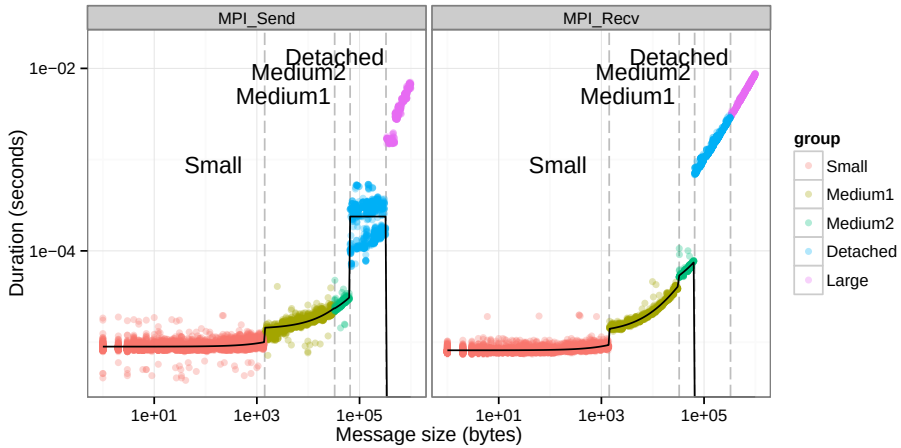
$$T_3 = o + kO_r \qquad T_5 = 2o + L$$

Routine	Condition	Cost
`MPI_Send`	$k \leqslant S$	T_1
	$k > S$	$T_4 + T_5 + T_1$
`MPI_Recv`	$k \leqslant S$	$\max(T_1 + T_2 - (t_r - t_s), 0) + T_3$
	$k > S$	$\max(o + L - (t_r - t_s), 0) + o +$ $T_5 + T_1 + T_2 + T_3$
`MPI_Isend`		o
`MPI_Irecv`		o

- May reflect the operation of specialized HPC networks from the early 1990s...
- Ignores many factors: contention, topology, complex protocol stack, ...
- So? What's the best? Fluid or LogP? None! They are complementary!

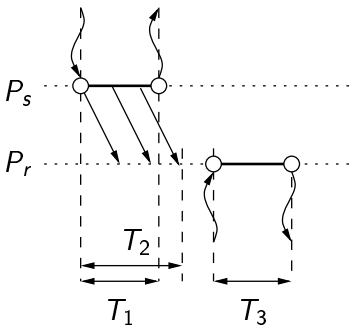
SimGrid Network Model

Measurements

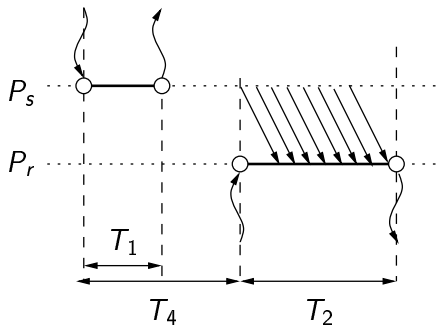


Hybrid Model

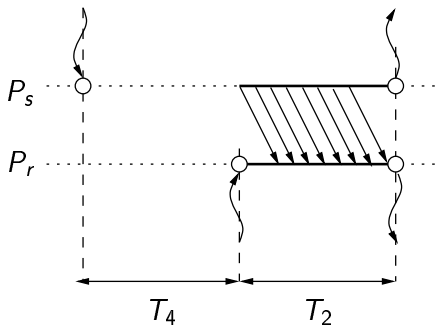
Asynchronous ($k \leqslant S_a$)



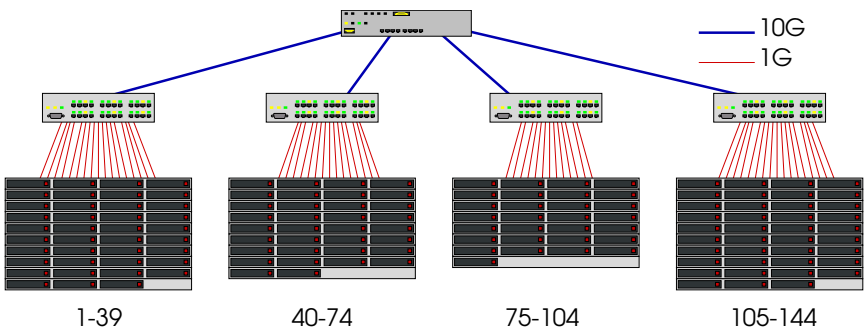
Detached ($S_a < k \leqslant S_d$)

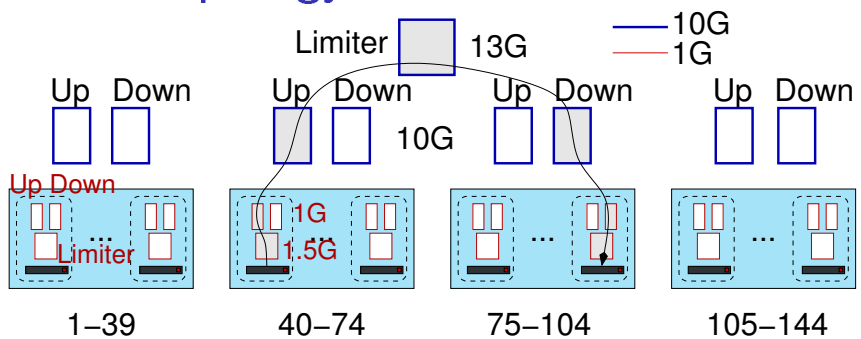


Synchronous ($k > S_d$)



Fluid model: account for contention and network topology

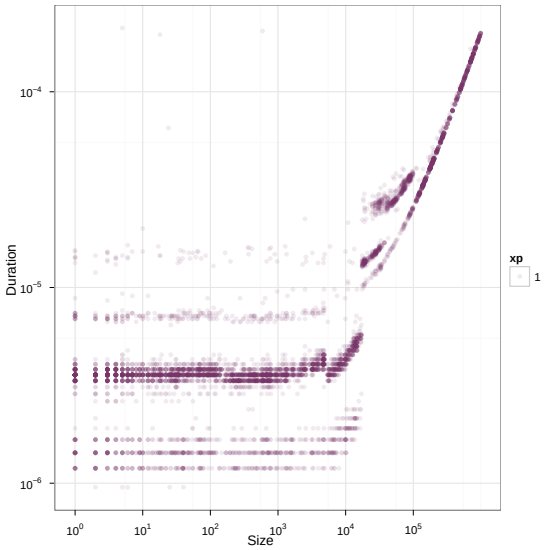


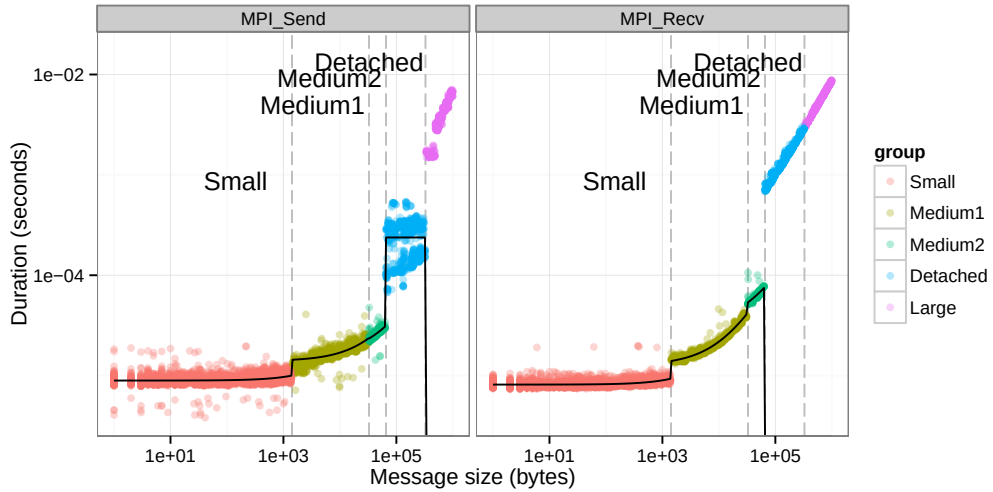


MPI Point-to-Point Communication on IB

IB have supposedly simpler and more predictable performance

- It should be clean and stable, with less intelligence in the protocol
- Indeed, it's faster and cleaner than TCP, but *IB is not that different*





Surprisingly, Modeling InfiniBand is complex wrt Bandwidth Sharing!

- Strictly fair share of IB buffers (in and out)
- Preliminary feelings: bandwidth is not fairly shared, but handling time is
- Counter-intuitive results, but results got confirmed (+ we have a candidate model)

Conclusion on Network Modeling

Analytic Models are possible

- TCP: Algorithmic model for synchronization + Equation-based for sharing
- IB: Still ongoing but encouraging (even with strange sharing)

Models are Getting Complex (but that's ok)

> *For today's complex simulations [from Computational Sciences], the computer program* **is** *the model. Questions such as* Does program X correctly implement model A?, *a question that made perfect sense in the 1960s, have become meaningless.* — Konrad Hinsen

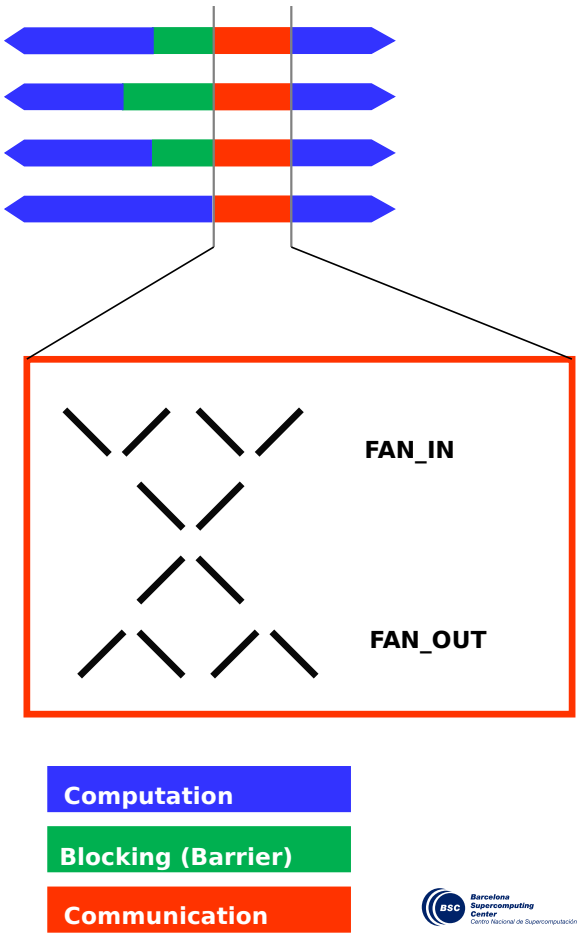
The runtime also induce protocol switches

- e.g. Eager mode vs. Rendez-vous mode
- Presented (SimGrid) Results are somehow specific to MPI
- MPI collective operations absolutely have to be modeled too

Analytic Collective Models (1/2)

Dimemas' Simple Models

- Regular and similar Algorithms:
  - Some Fan In, a middle operation, and some Fan Out
- To model a given collective algorithm, you specify
  - Amount of Fan In/Out and cost of each tree level
  - Cost of the middle operation
- Example of Scatter/Gather:
  $$\left\lceil \frac{\log N}{\log \text{fan}_{\text{in}}} \right\rceil \times \left(\text{latency} + \frac{\text{size}}{\text{bw}}\right) + \left\lceil \frac{\log N}{\log \text{fan}_{\text{out}}} \right\rceil \times \left(\text{latency} + \frac{\text{size}}{\text{bw}}\right)$$
- Cost of All2All: (no FAN in/out but similar)
  $$N(N-1) \times \left(\text{latency} + \frac{\text{size}}{\text{bw}}\right)$$
- Add a barrier before to nicely fit to the picture



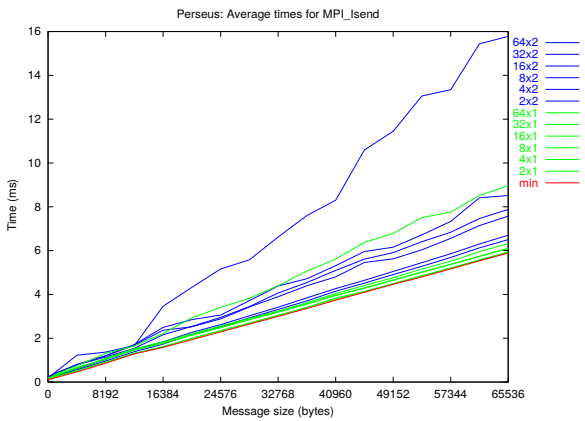
Analytic Collective Models (2/2)

Cons of Dimemas' Collective Models

- Models are simplistic compared to algorithms' sophistication, barrier is artificial
- Topology not taken into account, Contention through bus' tokens

Approach of [Grove, Coddington 2003]

- Don't model performance, benchmark and replay it
- On given cluster, benchmark every communicator size
- Also benchmark communicator geometries
- This gives the self-interference of collectives
- Could be extended to interference between collectives



Pros of Dimemas' Collective Models

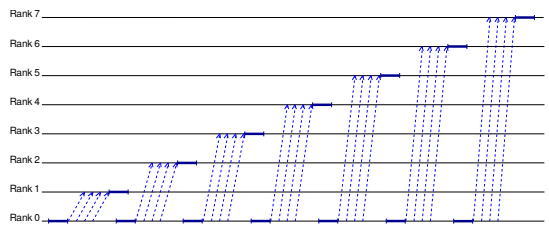
- You can easily extrapolate to other network characteristics and topology
- Easy to instanciate on a given platform

Collective Communications Through Trace Replay

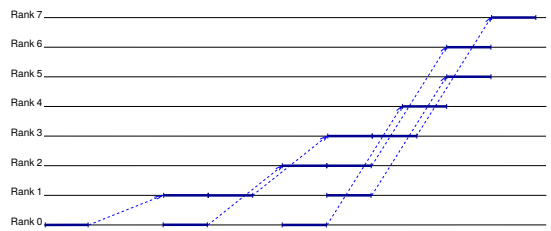
Improving the realism while enabling extrapolation

- Decompose any collective into a set of point-to-point comms
- Tracing is not trivial, as staying at PMPI level is not enough
- LogGOPSim: collectives are rewritten in a DSL called GOAL
- BigSim: traces are collected in Charm++, underneath

```
rank 0 {
   l1: calc 100 cpu 0
   l2: send 10b to 1 tag 0 cpu 0 nic 0
   l3: recv 10b from 1 tag 0 cpu 0 nic 0
   l2 requires l1
}
rank 1 {...
```



Linear Broadcast/Scatter Pattern.

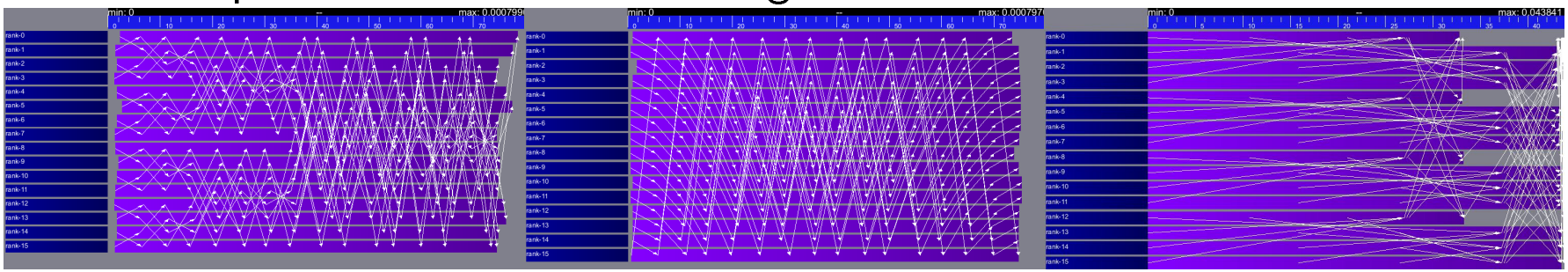


Binomial Tree Pattern.

Collectives' Code Scavenging

SimGrid's Approach

- SimGrid implements more than 120 algorithms for the 10 main MPI collectives



This code was . . . *integrated* (OpenMPI, MPICH, and StarMPI)

- Selection logic from OpenMPI, MPICH can be reproduced

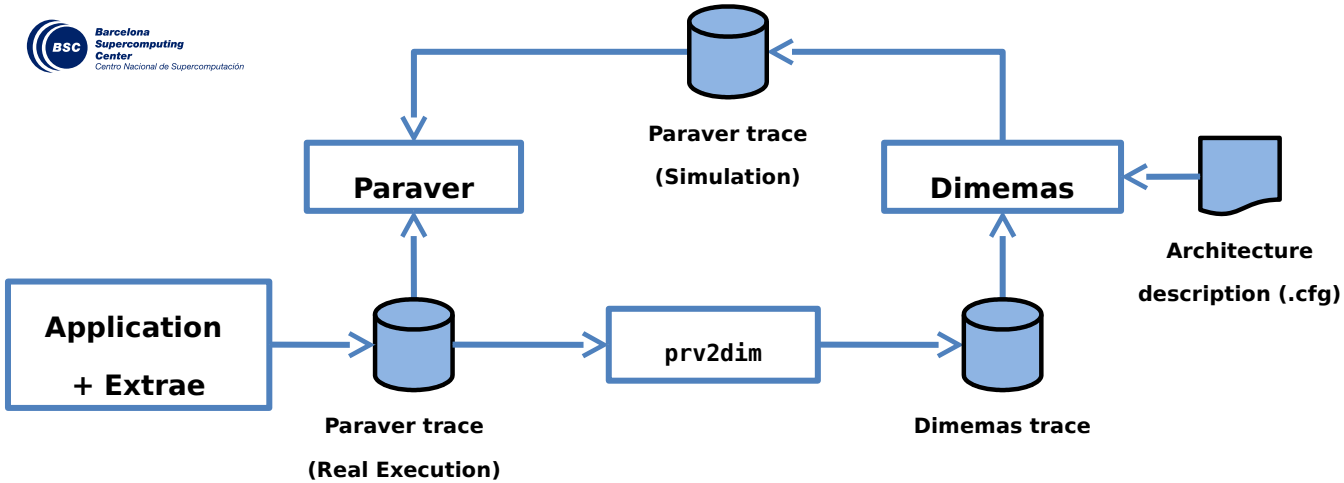
Future Work

- Expand this selection logic and autotuning possibilities to allow better selection
- See how all of this behaves on Multicore systems, with SMP-aware algorithms
- Implement MVAPICH2 Selector
- (In)validation on real platforms, with Infiniband, torus networks

Agenda

- Introduction and Motivation
- Simulation of HPC Systems (DIY)
  - Observing the Application
  - Modeling Computations
  - Modeling Communications
  - MPI Operations
  - Recap and Illustration With Different Simulation Projects
- SMPI 101
- Practical Session

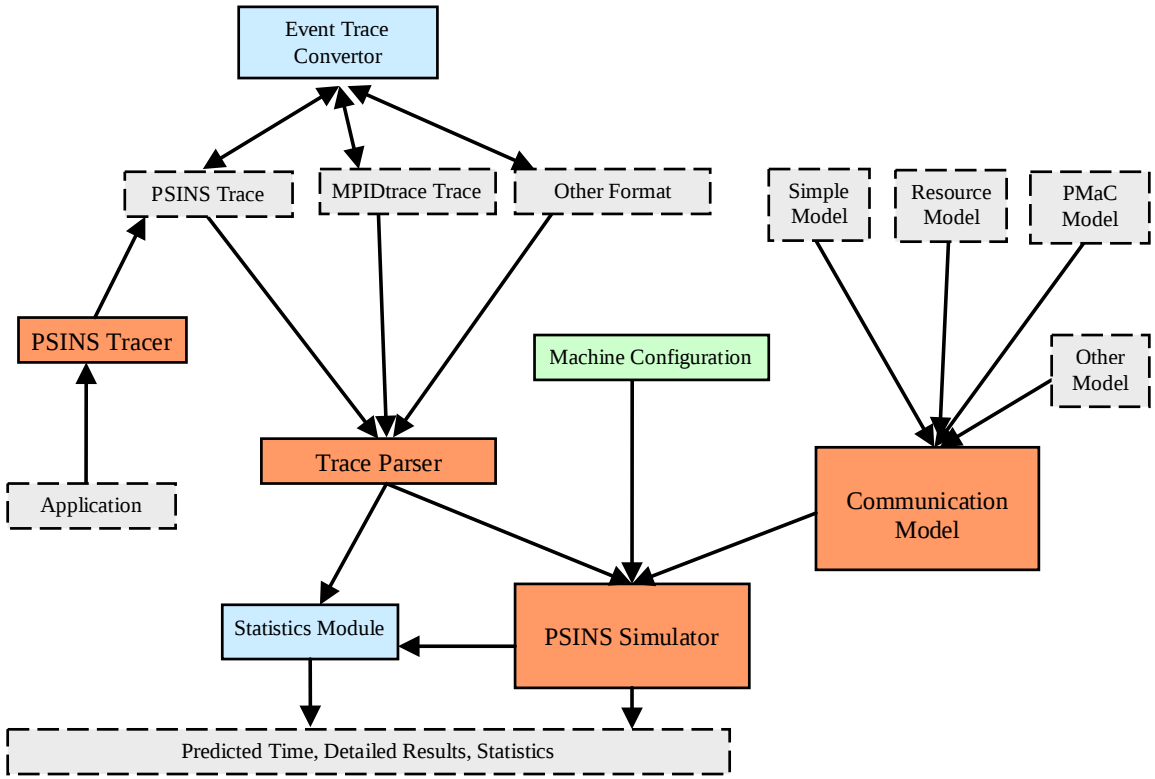
Dimemas (BSC)



Paraver trace What happens & When **Dimemas trace** Resource demands

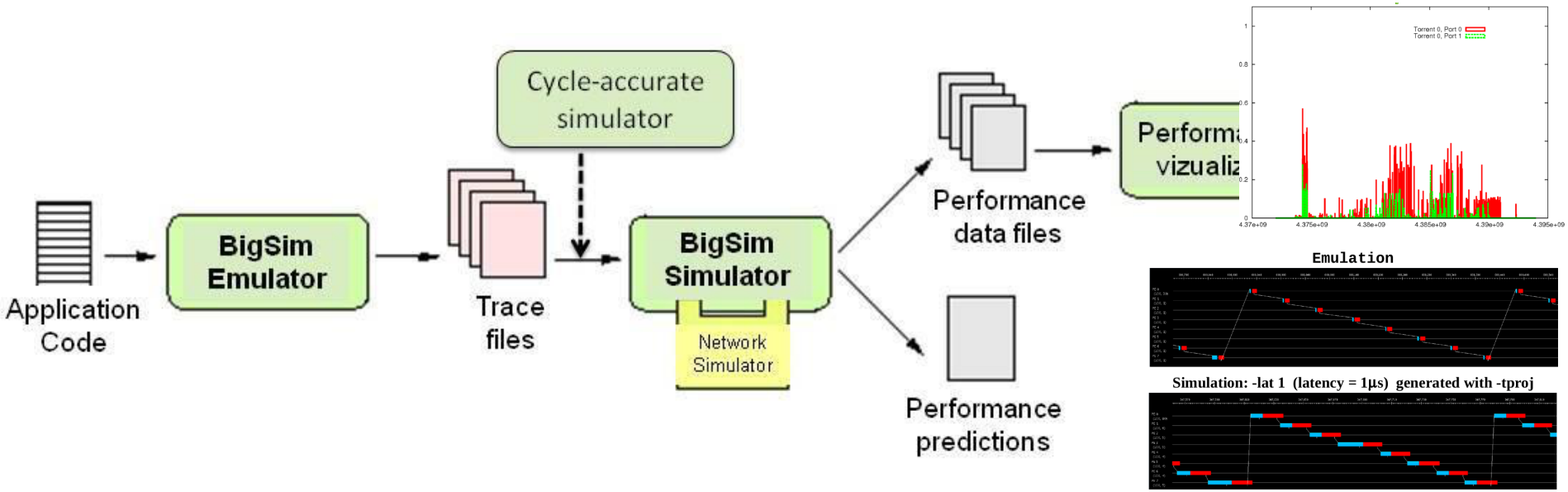
- Trace with Extrae, Explore with Paraver, Replay with Dimemas
- Simple lat/bw model, two-level network hierarchy, sharing through resource exclusion, analytic collectives
- Simple CPU model, SEB-specific speed factor
- Extended set of performance analysis tools (multispectral, clustering)
- Easy to modify, open source

PSINS (UCSD)



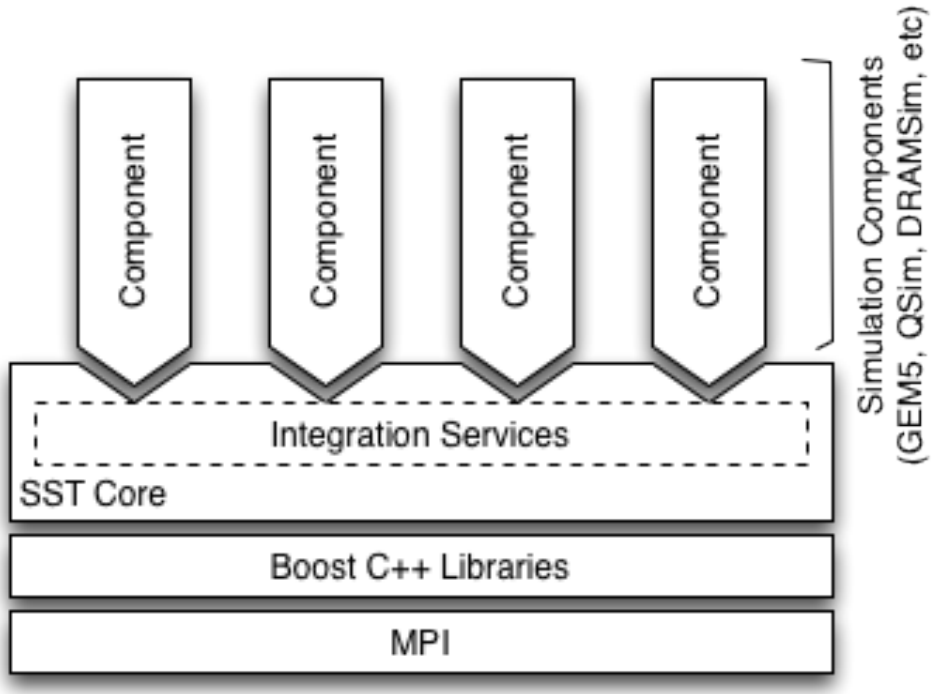
- Trace with on Dyninst, running several cache simulation at once.
- Used to rely on Dimemas. Now using a specific (but similar) simulator
- Fine-grain SEB performance prediction with the Convolver memory simulator
- PSINS is open, but Convolver not available $\rightsquigarrow$ difficult to use outside SDSC

BigSim (UIUC)



- Trace with AMPI/CHARM++
- Can do SEB extrapolation and trace projection (change the latency)
- Several ad-hoc simulators, running on CHARM++ itself
  - BigNetSim: Parallel/Sequential version (on top of CHARM++)
  - BigFastSim: Sequential version faster; both are slowly converging
- Simple delay model: $latency + \frac{size}{bw} + latencyPerPacket \times \frac{size}{packetSize}$
- Full packet-level simulator of IBM PERCS, Blue Waters, and many others
- Outputs: Projection traces and timings $\rightsquigarrow$ Gantt charts and Link stats

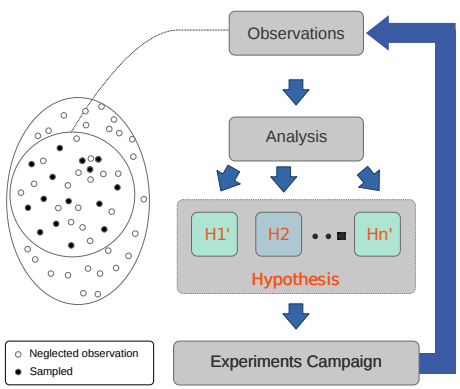
SST (Sandia National Labs)

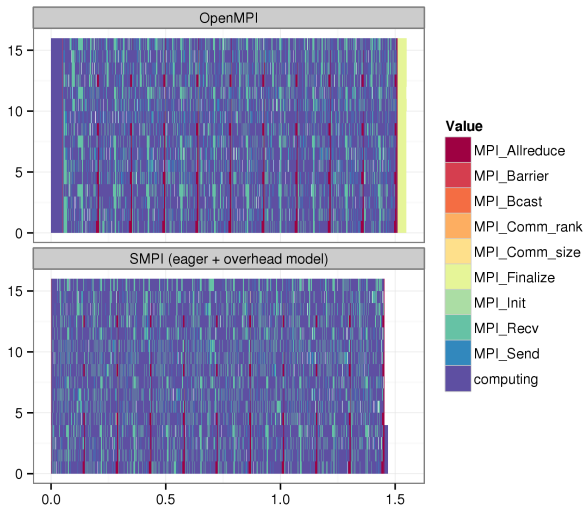


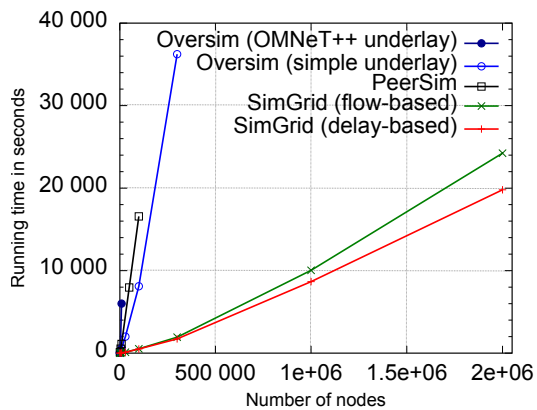
- Overall goal: speedup the co-design of next generation systems
- The Structural Simulation Toolkit: assembly of several interacting simulators
  - Some components developed internally, others are bindings to external projects
- Cycle-accurate simulators: Processor (Gem5), Memory (DRAMSim2), Network
- sst/macro glues things, allowing offline or online studies of C/C++ apps
- (the documentation is really impressive)

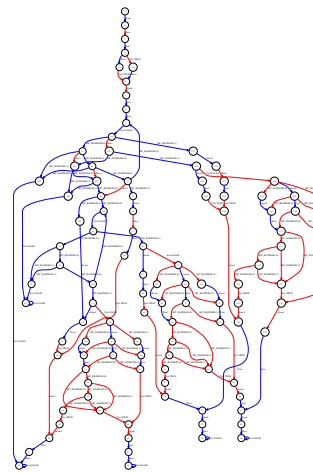
SimGrid (Inria, CNRS, various universities)

- Flow-level models (topology, contention, slow-start, cross-traffic)
  - We work on (in)validating our models since 10 years
- Scientific Instrument: grounded +100 papers, most are external to our group
- Versatile: Grid, P2P, HPC, Clouds, Volunteer Computing
- Sound: Validated, Scalable, Modular, Portable. Integrated in an Scosystem.
- Full-fledged model checker: verify safety, liveness (and more) on MPI apps









SMPI is the MPI flavor of SimGrid that we will use now

Agenda

- Introduction and Motivation
- Simulation of HPC Systems (DIY)
  - Observing the Application
  - Modeling Computations
  - Modeling Communications
  - MPI Operations
  - Recap and Illustration With Different Simulation Projects
- SMPI 101
- Practical Session

Section Objectives

- Learn to use the SMPI framework
- Preparation to the practical session that comes right afterward

Agenda

- Introduction and Motivation
- Simulation of HPC Systems (DIY)
  - Observing the Application
  - Modeling Computations
  - Modeling Communications
  - MPI Operations
  - Recap and Illustration With Different Simulation Projects
- SMPI 101
- Practical Session

Section Objectives

- Get up and running with the SMPI framework through practical questions