Measuring and Characterizing HPC applications and CPU/GPU simulation

Georges Da Costa
IRIT, Toulouse University

GreenDays@Lyon, 2012

Plan

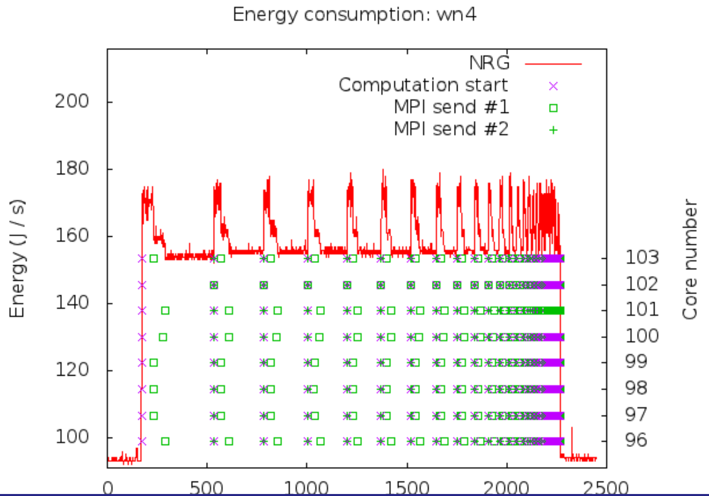
Context

To optimize a computing center:

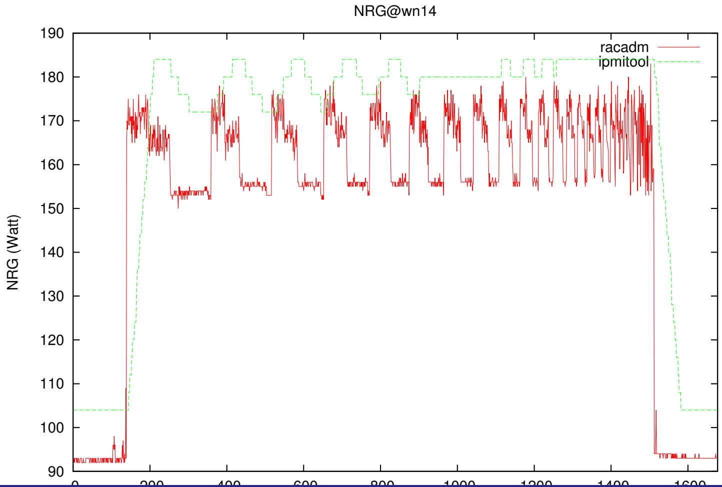
- Gather insight on running applications
- Choose how to act
  - depends on application
  - More precise : phase of application
- Act (change frequency, switch on/off parts of nodes,...)

Optimize : reduce energy consumption at the same performance

Application modification



Remark on monitoring: Choose the right tool



Ignorance is bliss, really?

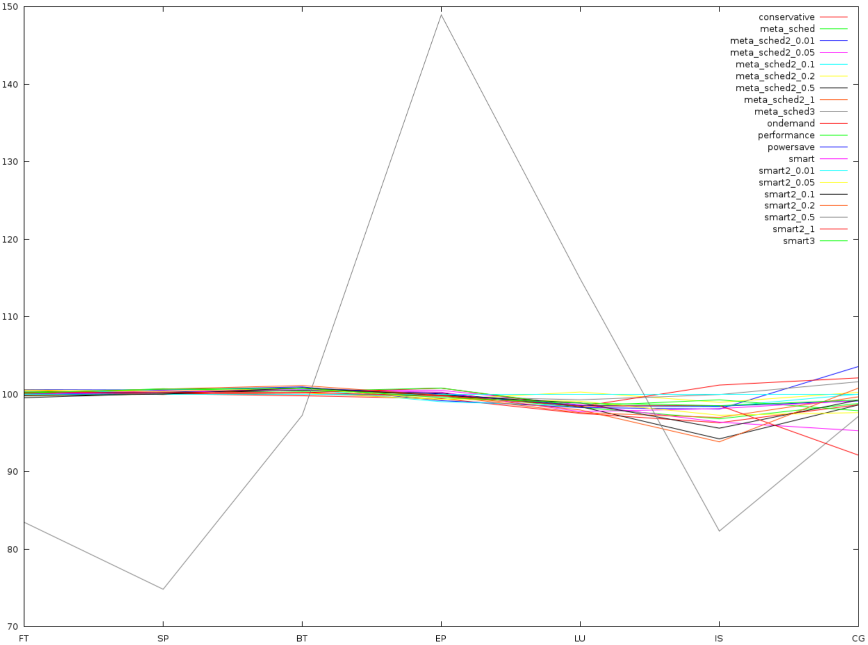
- *-AAS (PAAS, IAAS,...) leads to ignorance
- Ignorance leads to errors
- Errors lead to inefficiency

Focus :
How to optimize a computing center while knowing nothing?

Know your enemy

- What we know
  - HPC applications
  - Goal: Save energy
  - No impact on performance (SLA,...)
- Name your weapon (constraints)
  - Minimum impact of monitoring
  - Closed application, no source
  - Even full OS freedom (Grid'5000, VMs)

Is it so important?



Plan

Who's Who

- Which application is running?
  - Ask the developer, but
    - Depends on library
    - Can cheat (if accounting is related to it)
    - Computers work for us, not the opposite
  - Application is not important, its behavior is!
    - Two different apps can have the same impact
    - System changes can have the same impact

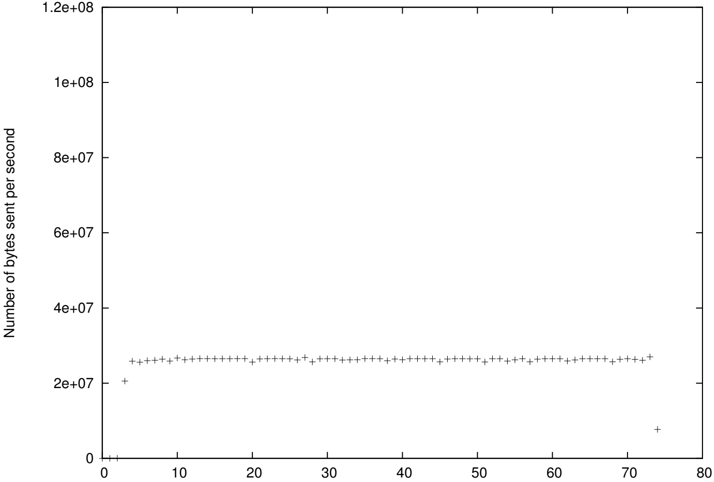
We need Run-Time Behavioral Detection

BigBrother is watching

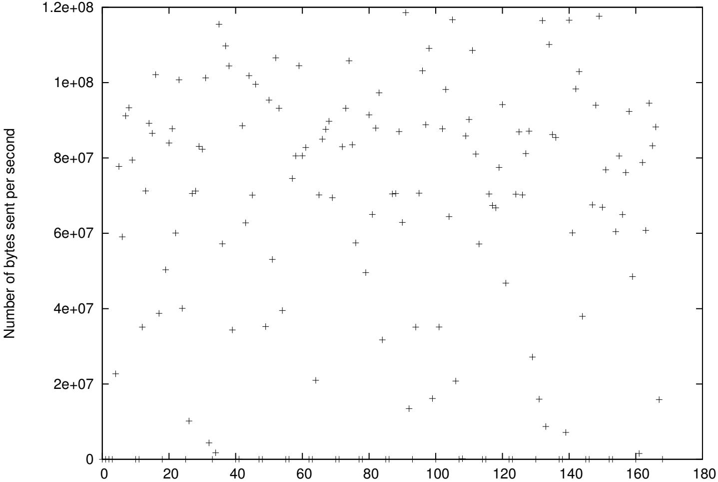
Run-time detection

- Behavioral pattern
- Extract information
  - Fine grained : performance counters, system values
  - Coarse grained : network, disk, power consumption
- Classical remark: impact of the monitoring infrastructure

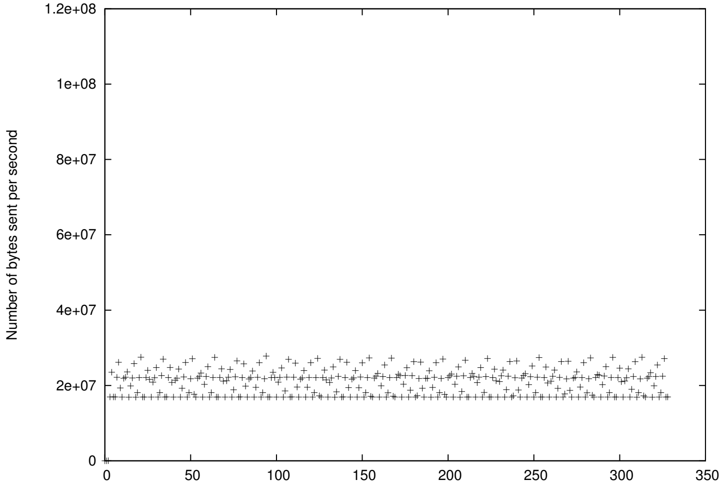
Finding patterns, NPB example, CG



Finding patterns, NPB example, FT



Finding patterns, NPB example, SP



Model creation

First create a model

- Run and monitor reference applications
- Cluster subset of characteristic
- Choose the most best subset
  - The most discriminating
  - The one with less impact
  - The one with best stability

Using a clear metric reduces bias

Use your creation

Simple to use the model once it is created:

- Step 1: Measure some characteristics
- Step 2: Compare to reference
- Step 3: Categorize application (or phase)

Low impact method:

- Low computing cost
- Network and power characteristics have low monitoring costs

Plan

Configuration

Usecase : Nas Parallel Benchmark

- Different type of workload
- Representative of HPC applications
- Seven benchmarks
  - Embarrassingly parallel
  - Communication intensive

Monitoring infrastructure

- Performance Counters
  - Standard Linux *perfcounters* (>2.6.31)
  - 1 measure/second/core/perfcounter
- Network IO
  - Inbound and outbound packets and bytes
  - 1 measure/second/host
- Disk IO
  - Read and write
  - 1 measure/second/host

Post-measure processing

- Instantaneous measures fluctuate widely
- Need for low cost post-processing
- Small window processing
  - To react to application phase
  - Low memory/processing cost of processing (ie no FFT)
- Simple statistical processing:
  - Mean, median, standard deviation, decile

Characteristic choice

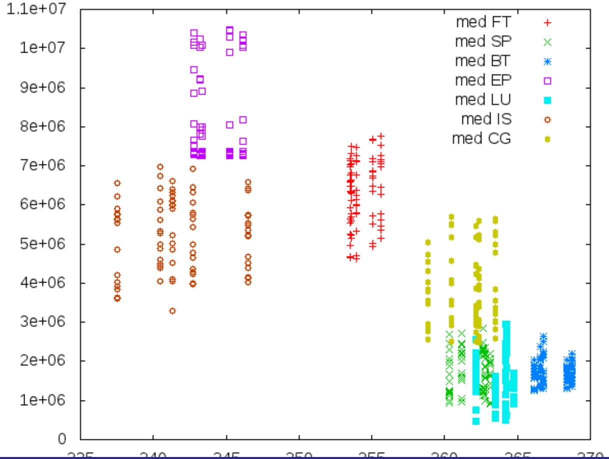
Characteristics relevance depends heavily on applications

For HPC application

- Disk is of no use
  - Disk IO only at beginning and end
  - Can change on low memory condition (swaping)
- Perfcounters are expected to be relevant
- Network and power also

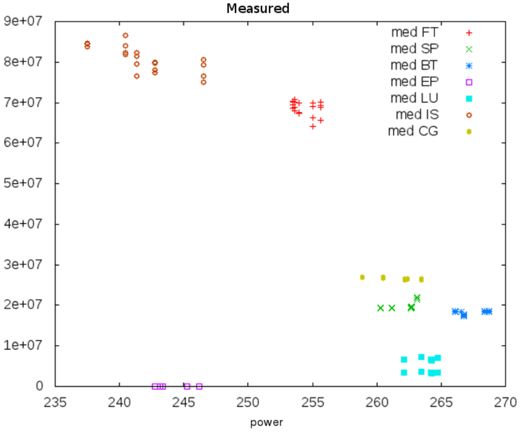
Perfcounters: unexpectedly bad

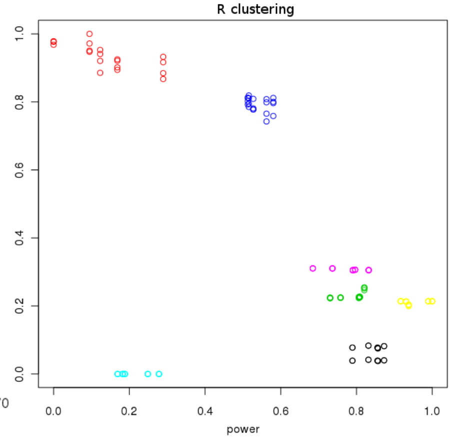
One of the best: **HW_BRANCH_MISS** (with power on x-axis)



Network and power

Best combination: Median number of packets and Mean Power





Plan

Energy efficient dvfs

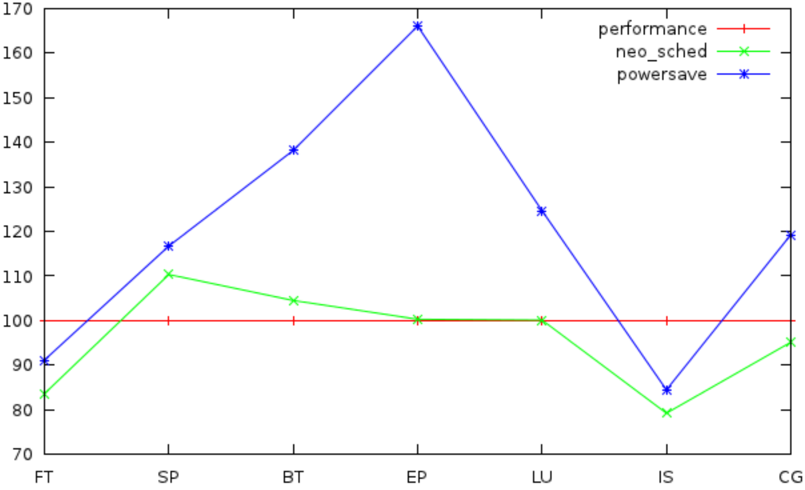
Using HPC application detection

- Categorize application (or phase)
- Apply a rule-based algorithm
- Change processor speed to min or max

Can take into account several objectives depending on rules

- Energy only
- Energy with taking into account performance
- ...

With more control comes more efficiency



Conclusion on application profiling

- Application characterization is possible with no impact!
- It leads to optimize resources usages and reduce energy (J)

Still much to do

- Improve statistical post-processing
- Improve reactivity (reduce window)
- Create a kernel governor

Plan

GPU are efficient for *some* algorithms

- Fast
- Energy-efficient

The system uses 7,168 NVIDIA Tesla M2050 GPUs and 14,336 CPUs; it would require more than 50,000 CPUs and twice as much floor space to deliver the same performance using CPUs alone (Nvidia)

With only CPU : 12 megawatts
Hybrid version : 4.04 megawatts

Runtime vs Placement

Where to run a task ? Two possibilities:

- Runtime
  - StarPU
  - + Low-impact, reactive
  - - Can be far from optimal
- Placement
  - - Need a-priori information
  - - Cannot adapt
  - + Can be optimal

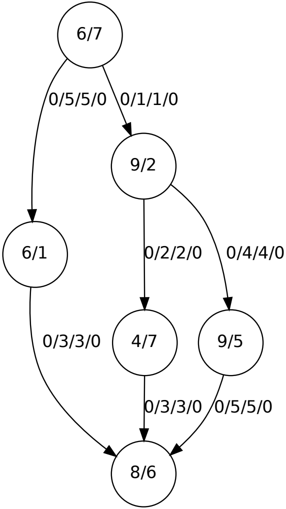
Hydrasim

Hydrasim: Tool to evaluate placement algorithm

- Realistic hardware
  - Number of processors and GPU
  - Energy and performance of hardware
  - Several policies (more later)
- Output
  - Makespan
  - Energy (for CPU, GPU and both)

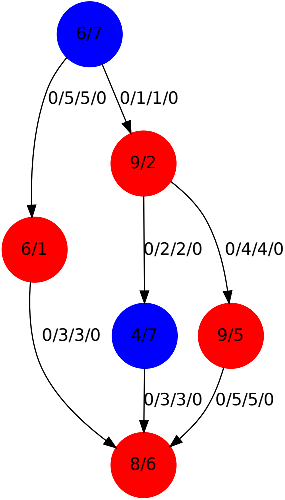
Task Model

- DAG of tasks
- Tagged *dot* file
- Tasks: time on CPU and GPU
- Communications: time of
  - CPU→CPU
  - CPU→GPU
  - GPU→CPU
  - GPU→GPU

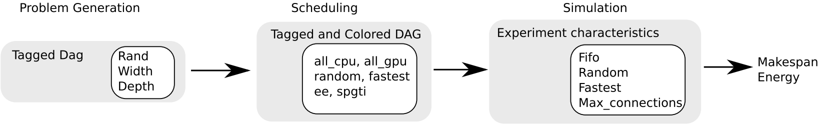


Allocation Model

- Colored *dot* file
- Blue processor
- Red GPU



Complete performance evaluation environment



Starting point depends on the problem

Simulator zoom

Simulator needs

- Number of CPU/GPU
- Characteristics of CPU/GPU
- Runtime policy to choose jobs
  - Fifo, Random, Fastest
  - Max_connections
- Runtime policy for the bus

Comparison of schedulers

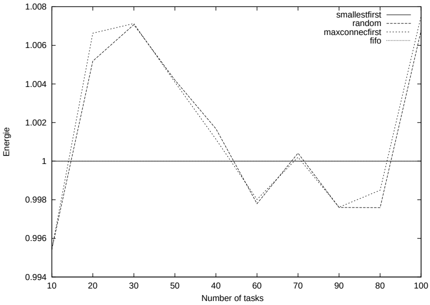
Let's take several schedulers

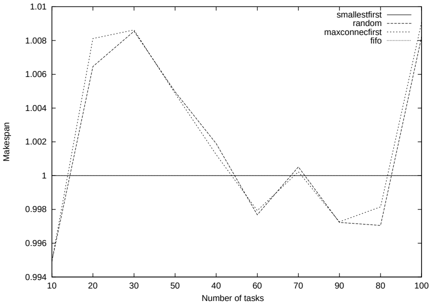
- All_CPU
- All_GPU
- Fastest
- EE (most energy efficient)
- Rand
- Spgti

Environment

- Applications
  - Generated using a random strategy
  - 10 to 100 tasks
- Hardware
  - CPU Intel Xeon E5540 and GPU Nvidia Tesla C1060.
  - 1 to 16 CPU and 1 to 16 GPU

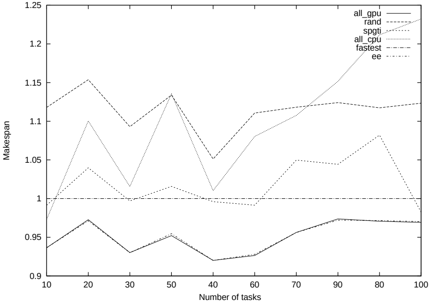
Direct results

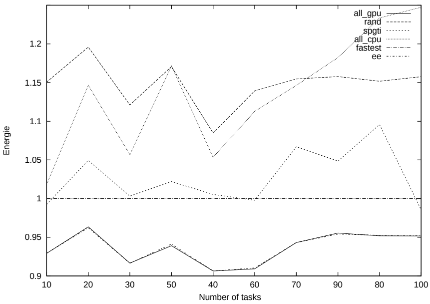




Energy and Makespan are not impacted by runtime

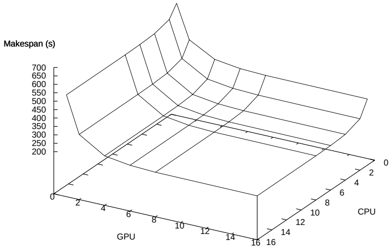
Direct results (cont)

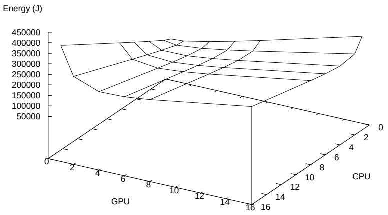




They are by scheduler

Optimize resource number





Conclusion

Using Hydrasim you can

- Test several hardware configuration
  - For free !
- Obtain the optimal number of resources
- Compare algorithms

We are interested in

- Use-cases
- Feedback on the simulator

http://hydrasim.sourceforge.net/

C++/Discrete event simulator/GPL