Getting a first grip on doing large computations at CWI

Nicolas Höning
Centrum Wiskunde & Informatica – CWI
(Centre for Mathematics and Computer Science)
Intelligent Systems Group

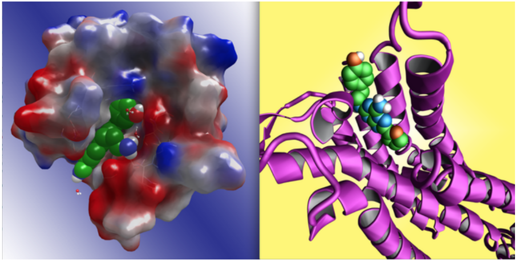
Scope: embarrassingly parallel computation problems

- No effort is required to separate the problem into a number of parallel tasks
- Results are independent, e.g. in
  - searching through large data sets
  - 3D graphics rendering
  - simulating independent scenarios
  - repeating computations with differing randomisation seeds (Monte Carlo Sampling)
  - etc.

Scope: state of the art

So this is happening right now:

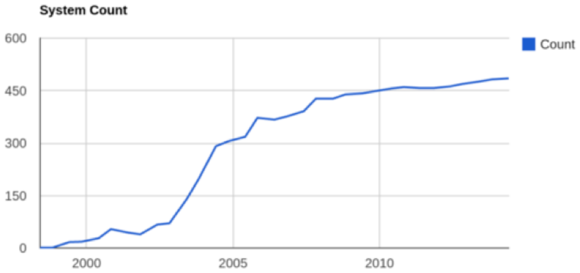
"4,829$-per-hour supercomputer (50,000 cores) built on Amazon cloud to fuel cancer research"



Source: ArsTechnica

Scope: state of the art

Linux is where it's at:



With 97 percent of the world's fastest supercomputers running Linux, the open-source operating system has eliminated almost all its rivals.

Source: ZDNet

Scope: Everything increases

- Embarrassment:
  - Lots of CPUs in your computer
  - Lots of CPUs in clouds

Scope: Everything increases

- Embarrassment:
  - Lots of CPUs in your computer
  - Lots of CPUs in clouds
- Expectations:
  - Big data
  - Complex problems
  - etc.

Scope: Everything increases

- Embarrassment:
  - Lots of CPUs in your computer
  - Lots of CPUs in clouds
- Expectations:
  - Big data
  - Complex problems
  - etc.
- Possibilities:
  - Many people write many tools
  - You will invest time

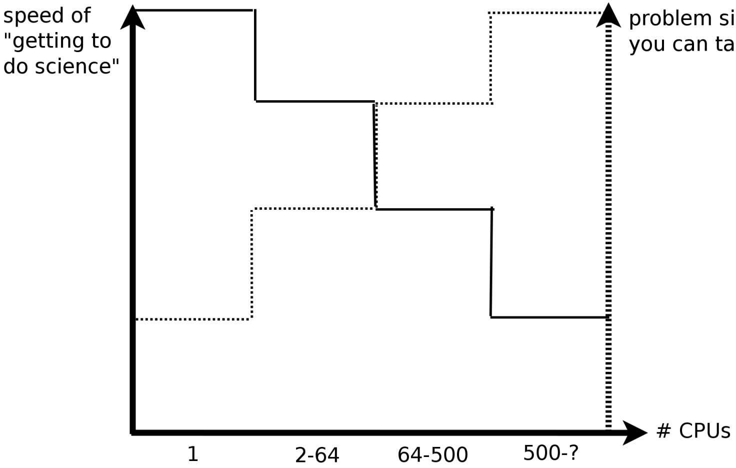
Outline

1. Computing resources: How to use many CPUs for many more subtasks?
2. Workflow management: How to generate and keep track of all those subtasks?

Make use of computing resources: Choices

- *Dynamic allocation* of tasks. Requires one process to assign tasks (to workers).
- A parallelisation tool can *be agnostic to the programming language* you are using, or embed in a language. I am interested in the former.

Make use of computing resources: The dilemma



Make use of computing resources: your local network, via Gnu parallel

- `https://www.gnu.org/software/parallel/`
- repeat any Unix command on separate CPUs
- communicates per SSH, runs jobs in threads
- very mature and rich
- target audience: system admins
- **Short demo**

```
parallel --gnu touch {}.tmp ::: 1 2 3 4 5
```

Make use of computing resources: your local network, via FJD

- https://github.com/nhoening/fjd
- communicates per SSH, runs workers in Unix screens who pick up jobs
- Assumes shared home directory
- Advantages:
  1. light-weight
  2. config files for parameterisation
  3. inspecting workers in progress possible
  4. you can re-sort job queue on the fly
- suited for long-running jobs (fix costs)
- **Short demo**

```
fjd --exe 'touch $1.tmp' --parameters 1,2,3,4,5
fjd --exe "mktemp XXX.tmp" --repeat 5
```

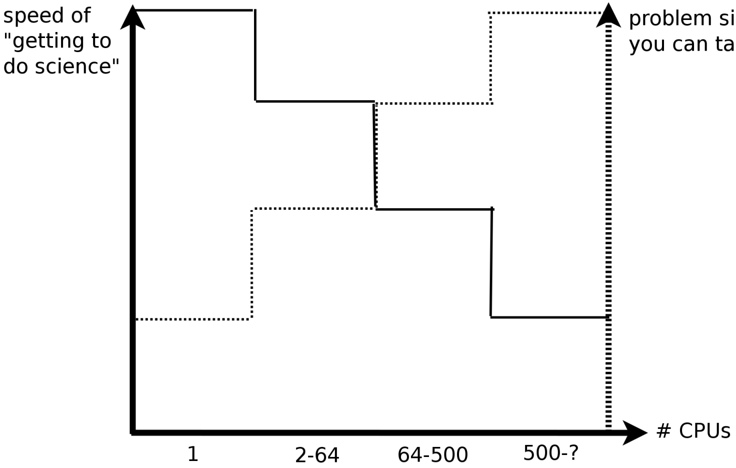
Make use of computing resources: Surfsara LISA cluster

- `https://surfsara.nl/systems/lisa`
- Computers with 12+ cores (all in all: 8960)
- Uses PBS (Torque) scheduling (born in 1980s), describe what you want in a job file
- Can use message passing (MPI) between nodes
- CWI/NWO has an agreement with SurfSara
- **Short look around LISA**

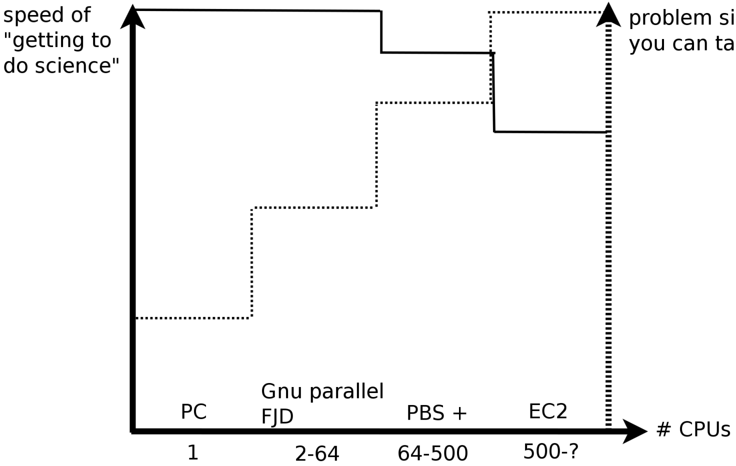
Make use of computing resources: The “cloud”

- Commercial cloud servers (e.g. Amazon EC2)
- Faster response time than PBS
- Almost no constraints on number of computers, more on your budget
- Skills you need here are also *very* useful for industry jobs
- Use some protocol to distribute tasks between cores, via MPI or AMQP, e.g. RabbitMQ
- Much control possible, e.g. with Docker

Make use of computing resources: The dilemma

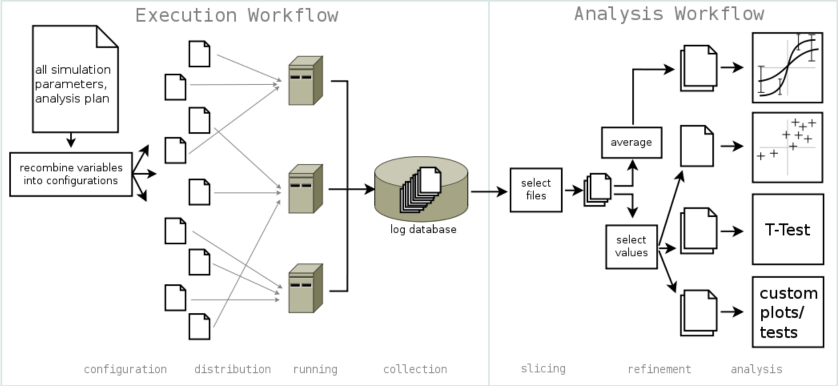


Make use of computing resources: Getting better



Hard challenge: switch effortlessly back and forth

Workflow management



Let's support iterative development + portability

Workflow management: Sumatra

- https://pythonhosted.org/Sumatra/
- “Scientific Notebook”
- “Automated tracking of scientific computations”

-
  ```
  python main.py default.param
  ```

  becomes

  ```
  smt run --executable=python
          --main=main.py default.param
  ```
- Links code, parameters and result files by watching folders (using version control systems, e.g. git)
- automatic work history, viewable in a browser
- Can use parallel computation with an MPI layer (can also run on PBS)

Workflow management

You should use version control for your code, by the way:

`https://scm.cwi.nl/`

Workflow management: StoSim

- `https://homepages.cwi.nl/~nicolas/stosim`
- Only tracks log files
- Very easy to get started
  1. few dependencies
  2. built-in support for FJD and PBS(+FJD) → switch effortlessly
  3. Can make plots and T-tests for you
  4. `stosim --run --plots --ttests`
- You can make (incremental) snapshots of code and results
- **Short demo**

Misc: SSH config

- configure SSH in `~/.ssh/config` [1]
- host shortcuts
- SSH keys
- connection sharing

[1] http://blogs.perl.org/users/smylers/2011/08/ssh-productivity-tips.html

Misc: my ~/.ssh/config file

```
ControlMaster auto
ControlPath /tmp/ssh_mux_%h_%p_%r
ControlPersist 4h

Host cwi
HostName ssh.cwi.nl
User nicolas
IdentityFile ~/.ssh/id_cwi
```

Misc: run Unix commands in background

- Appending an ampersand
- CTRL-Z and bg/fg
- Unix screens
- nohup

Misc: get LISA account

- `https://surfsara.nl/systems/shared/fom-ncf`
- Basically, fill in forms and email copies over to them
- normally, projects begin March 1
- People are helpful there. Call 020 800 1400 or write to hic@surfsara.nl

Extra: ask the PBS system about its queue

```bash
#!/bin/bash
job=$1

idle=`showq | grep "IDLE JOBS" -n | cut -d: -f1`
jobline=`showq | grep -n $job | cut -d: -f1`
place=`expr $jobline - $idle - 2`

echo "Idle Jobs section starts at line $idle"
echo "Job $job at line $jobline"
echo "Place in queue: $place"
```

Extra: use all CPUs on PBS nodes with Gnu Parallel

1. write PBS files to request nodes
2. wait until nodes are started

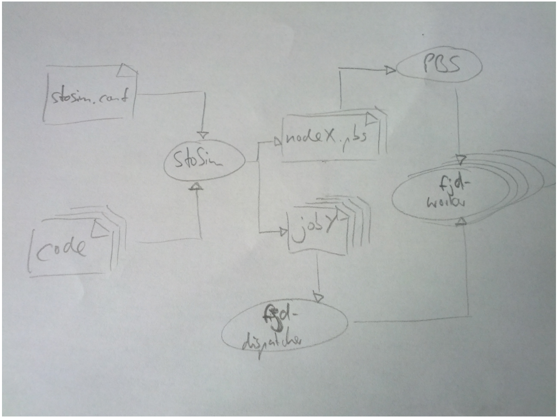
3.
   ```
   cat argfile |
        parallel --slf $PBS_NODEFILE your_command
   ```

Extra: use all CPUs on PBS nodes with FJD

A brute force benchmark for a problem, evaluating > 600K problem configurations on a PBS computation cluster:

`https://github.com/nhoening/fjd/blob/master/fjd/example/runbrute.py`

Extra: use all CPUs on PBS nodes with StoSim (+ FJD)



Simply put "scheduler:pbs" in the stosim.conf file (see also docs for additional information you can add about your requirements on LISA)

The End

Thanks for coming