Distributed Ports Builds in OpenBSD

Marc Espie (espie@openbsd.org)

october 2010, sunday 10

Note

- The actual slides are available from
- http://www.openbsd.org/papers/

In the beginning

- there was nothing
- and then Nikolay Sturm wrote the old dpb in 2004

- no multi-core machines
- distribute builds on sparcs
- very slow and reliable, retries things every time

Meanwhile

- pkg_add happened
- if you weren’t there this morning, too bad...

- generic framework for dealing with dependencies
- and multi-core laptops

Accidents

- lots of static information in the ports tree
- we check a lot of stuff without building
- introspection (make dump-vars)
- ... used for sqlports

New dpb

- happened more or less last winter
- start building as soon as we can
- Practical design
  - old dpb was annoying to set up
  - because it takes so long to start

New dpb: fast as hell

- as soon as you start it, it builds stuff
- dependencies are computed "as another job"
- uses partial information to figure out ports it can build

Separate ports in categories

- to-build: stuff to be built eventually
- queue: stuff that CAN be built (dependencies accounted for)
- packages: stuff that was built

As soon as queue fills up, we start building.

- Keep queue happy
- if a core is free, we want to build stuff

Maximal parallelism

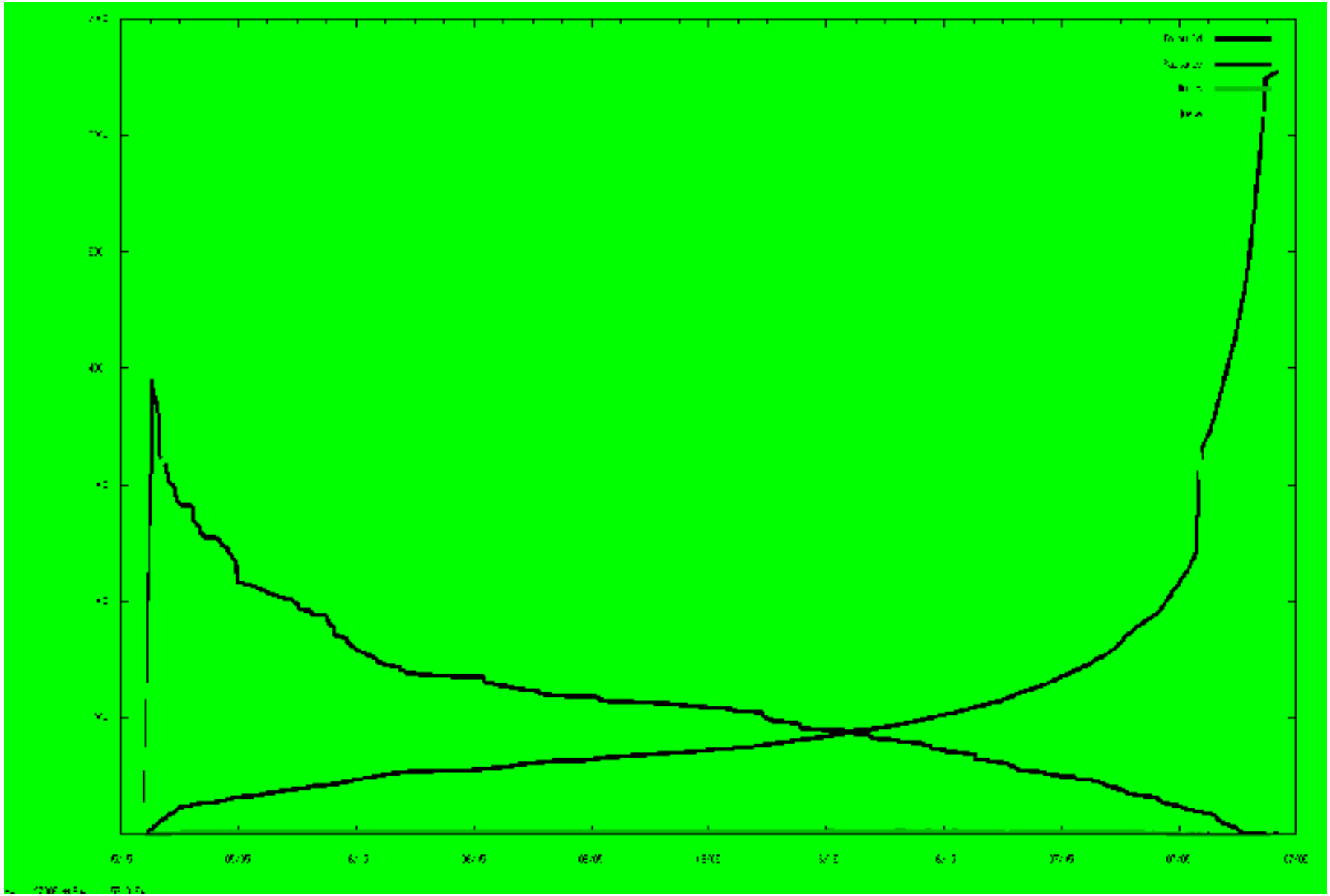
- Avoid core starvation
  - do most interesting dependencies first
- ... but
  - OpenOffice will fuck you up !
  - Other cores are twiddling their thumbs
  - OpenOffice takes an extra 6 hours to build

Feedback from previous builds

- Stuff that took long to build will take long to build
- So use build times to direct queues
  - Largest ports build first
- Dependencies matter
  - Weigh each port according to stuff that depends on it
  - Simply sum everything

- Good enough
  - Supernatural insights

Examples



CS Majors

- ... go away
- NP-complete problem
- but the ports tree is special
  - enough ports with no dependencies to keep us busy
  - enough small ports to fill the gaps

- doesn’t work for non-bulk
  - starvation will happen

Multiple machines

- Treat them as equal
- and stuff breaks
- OpenOffice fucks you up, again
- don’t build big stuff on the old slow laptop

Physicist approach

- Annotate each core with a "speed factor"
- Approximation of processor speed (many cores)
- Slower machines should build smaller packages
- Again: directed by previous builds
  - sort packages according to build times
  - biggest stuff go to the biggest machine, and so on.
  - simple dynamic problem, that’s optimum...
  - ... if everything is known

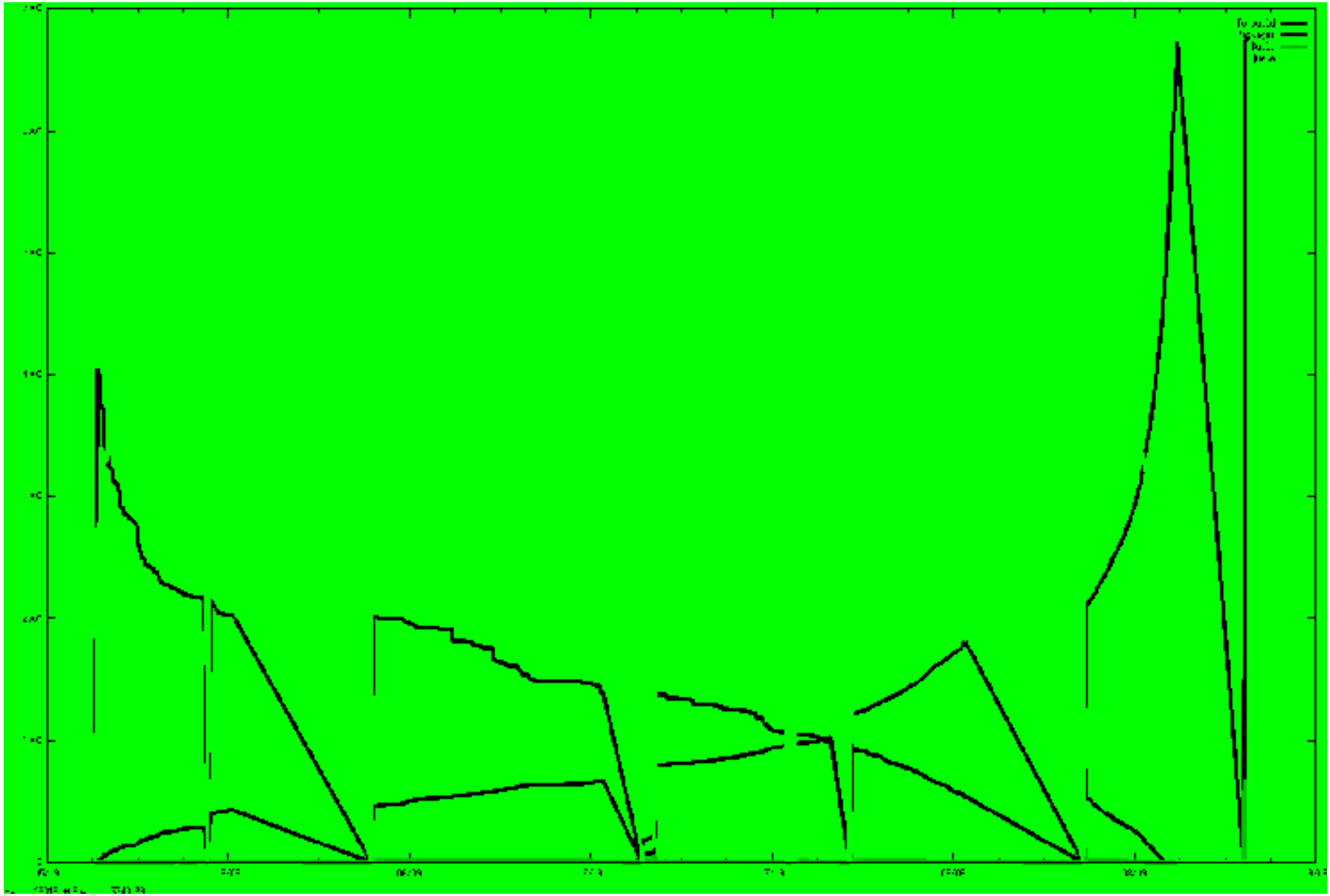
In practice

- Good enough even if everything is not known
- too slow: sorting repeatedly thru 3000 packages in perl
- physics again: use bins
  - seperate queue according to build time
  - each bin is 4 times as big as the next one
  - fastest machine has access to everything
  - other boxes are limited to their current bin
  - good enough in practice

Everyday life

- Builds break
  - disk fills up
  - sparcs panic
  - keep going anyways

Not so nice example



Everyday life

We want locks

- but the ports tree has locks
- they’re local
- dpb doesn’t want to stop, it wants to do other stuff
- locks are needed for multi-packages and flavors
  - you don’t want to build another flavor concurrently

Side benefits

- several dpb can run at once
- errors are not fatal

Everyday life (cont.)

Lots of logs

- log by pkgpath
- log by pkgname
- log errors
- log stats
- log machine build characteristics (libraries)

Everyday life (cont.)

Use logs

- monitor build in terminal
- show % of completion
- based on previous build
- not perfect, so what ?
- show stuck jobs
- show process numbers

dpb itself

Internals

- perl again
- Cores, Jobs, and Tasks

Work in progress

- Bugs to fix
  - pkgpath problems
  - hard recoveries
    - dump-vars
- Simple features
  - ETA
  - Dependencies rebuilds
- Hard features
  - More specific monitoring
  - Previous rusage

Thank you

Questions ?