TakTuk-3
Large scale remote execution deployment

Guillaume Huard

AHA IMAG/INRIA Project and MOAIS INRIA Project,
ID Laboratory (Grenoble), supported by CNRS, INRIA, INPG, UJF

Laboratoire
Informatique et
Distribution

CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

INSTITUT NATIONAL
DE RECHERCHE EN
INFORMATIQUE ET
EN AUTOMATIQUE

Institut National
Polytechnique
de Grenoble

Outline

1. Motivations
   - Introduction
   - Needs
2. Deployment
   - Parallelization
   - Distribution
   - Optimal deployment
3. Dynamic deployment
   - Dynamic environment
   - Pipeline size evaluation
   - The new TakTuk engine
   - Using TakTuk

Outline

1. Motivations
   - Introduction
   - Needs
2. Deployment
   - Parallelization
   - Distribution
   - Optimal deployment
3. Dynamic deployment
   - Dynamic environment
   - Pipeline size evaluation
   - The new TakTuk engine
   - Using TakTuk

The number of processors in parallel machines

- gone from hundreds to thousands during last years
- still growing at a fast pace
- need for **tools to administrate and exploit** them

List for June 2005

R_{max} and R_{peak} values are in GFlops. For more details about other fields, please click on the button "Explanation of the Fields"

DETAILS EXPLANATION OF THE FIELDS

1-100 | 101-200 | 201-300 | 301-400 | 401-500

Rank	Site Country/Year	Computer / Processors Manufacturer	R_{max} R_{peak}
1	DOE/NNSA/LLNL United States/2005	*BlueGene/L* **eServer Blue Gene Solution** / 65536 IBM	136800 183500
2	IBM Thomas J. Watson Research Center United States/2005	*BGW* **eServer Blue Gene Solution** / 40960 IBM	91290 114688
3	NASA/Ames Research Center/NAS United States/2004	*Columbia* **SGI Altix 1.5 GHz, Voltaire Infiniband** / 10160 SGI	51870 60960
4	The Earth Simulator Center Japan/2002	**Earth-Simulator** / 5120 NEC	35860 40960
5	Barcelona Supercomputer Center Spain/2005	*MareNostrum* **JS20 Cluster, PPC 970, 2.2 GHz, Myrinet** / 4800 IBM	27910 42144
6	ASTRON/University Groningen Netherlands/2005	**eServer Blue Gene Solution** / 12288 IBM	27450 34406.4
7	Lawrence Livermore	*Thunder*	19940

Administration and development on large scale machines

Nodes administration:
launch the same command on each node, e.g.:

- `uptime` to grab statistics about the recent machine availability
- `ifconfig` to get mac addresses for `dhcpd` configuration
- `dig`, `ping`, ... for network issues diagnostic
- ...

Parallel applications development:
launch the same parallel program on all nodes (like `mpirun`), e.g.:

- slaves of a master/slave application
- all participants of a symmetric parallel application
- self organizing system (P2P), daemons (monitoring)
- ...

Deployment needs

Automatization and multiplexing:
User should not have to launch each remote execution by himself

- automatization of remote connections to each machine
- I/O multiplexing to the root node (gathering of result)

Administration and development are interactive tasks :
High efficiency is mandatory

- execution latency when tracking bugs
- quick system diagnostics to solve administration issues
- ...

Questions addressed in this talk

How to perform efficiently all the remote connections ?

- parallelisation ?
- distribution ?

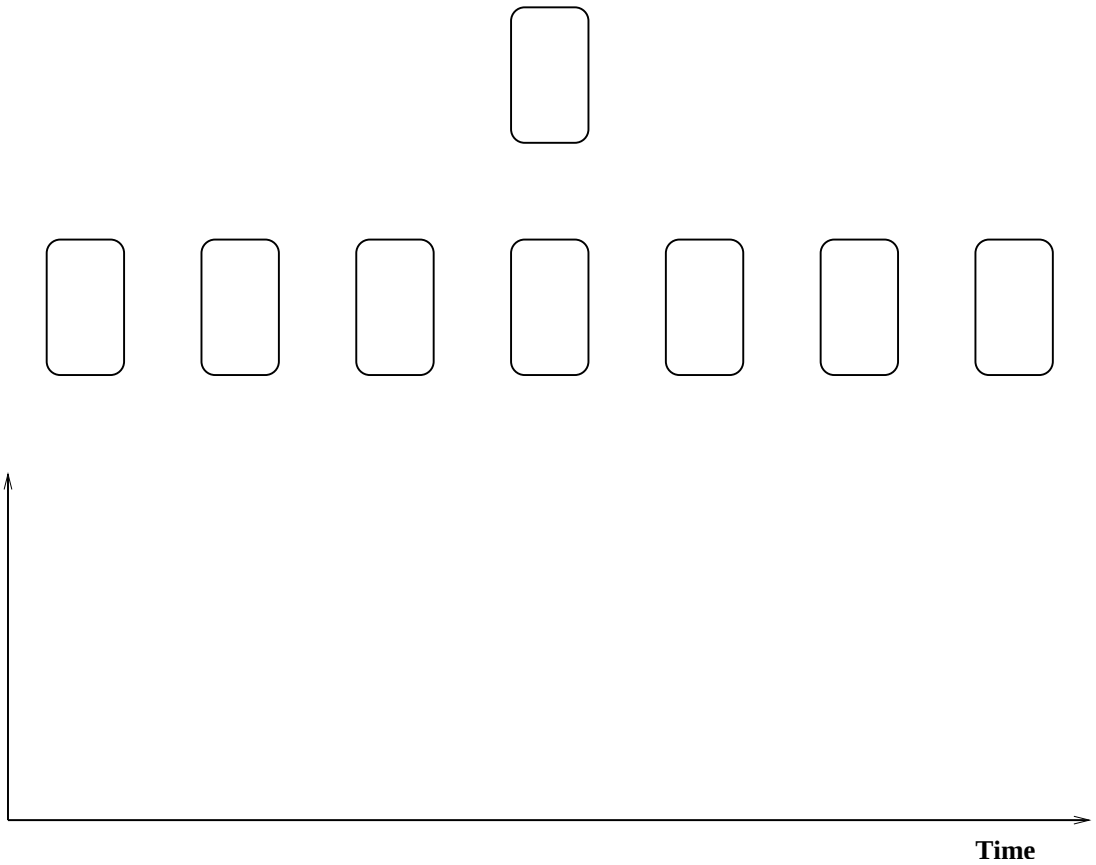
... to minimize execution latency.

Outline

1. Motivations
   - Introduction
   - Needs
2. Deployment
   - Parallelization
   - Distribution
   - Optimal deployment
3. Dynamic deployment
   - Dynamic environment
   - Pipeline size evaluation
   - The new TakTuk engine
   - Using TakTuk

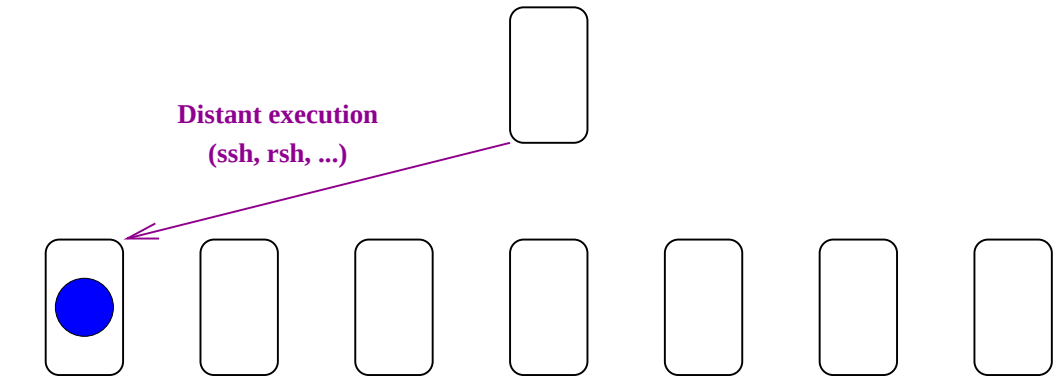
Example of trivial solution

```
Foreach i in hosts do ssh $i uptime
```



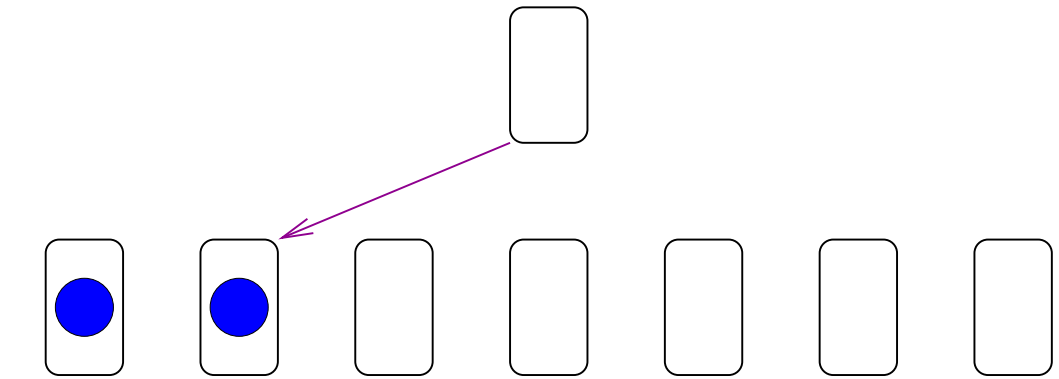
Example of trivial solution

```
Foreach i in hosts do ssh $i uptime
```



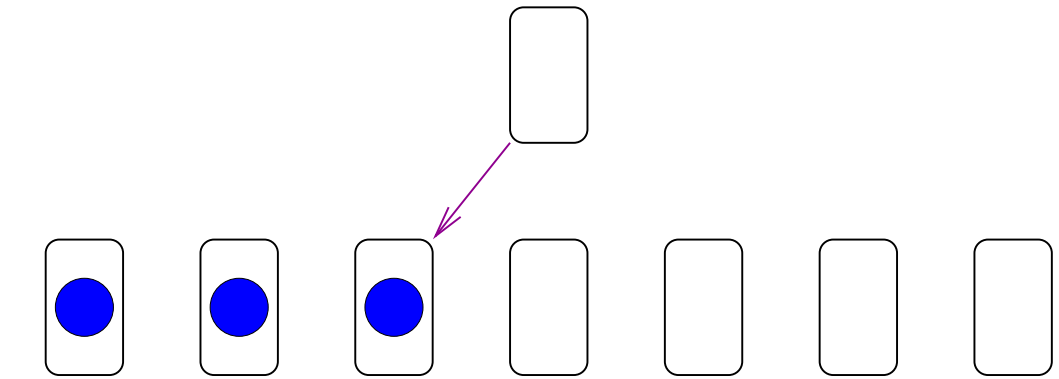
Example of trivial solution

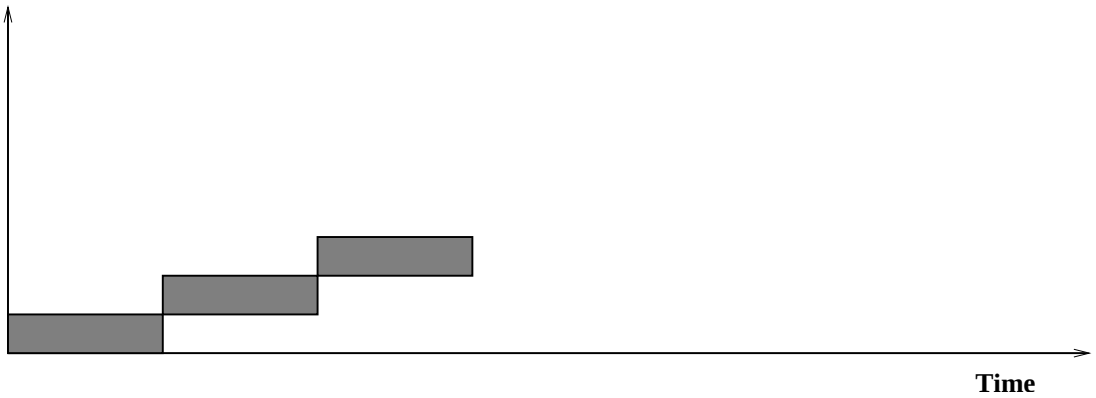
```
Foreach i in hosts do ssh $i uptime
```



Example of trivial solution

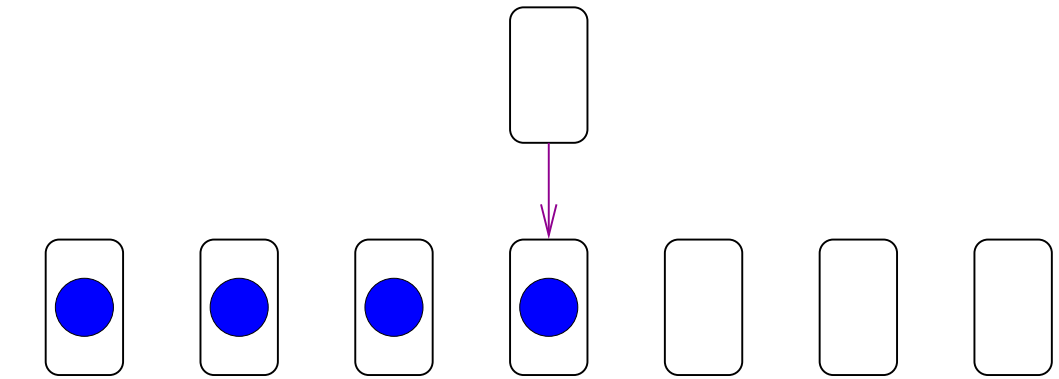
```
Foreach i in hosts do ssh $i uptime
```

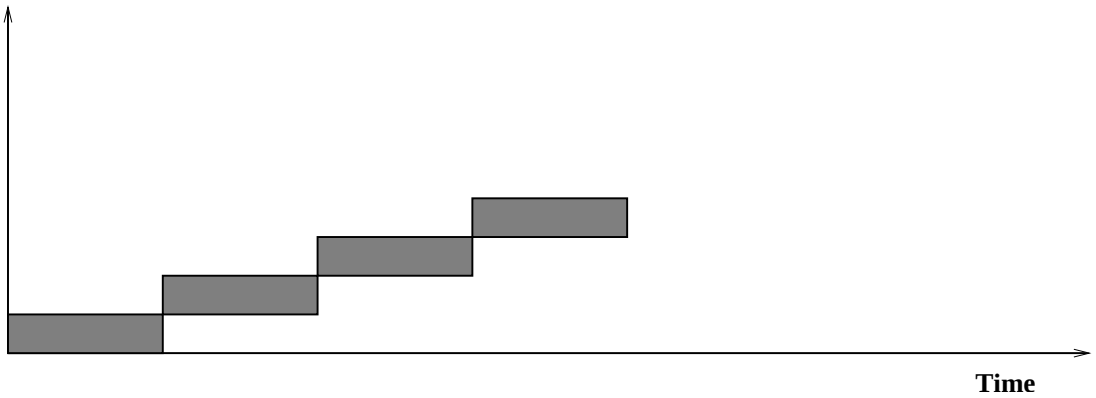




Example of trivial solution

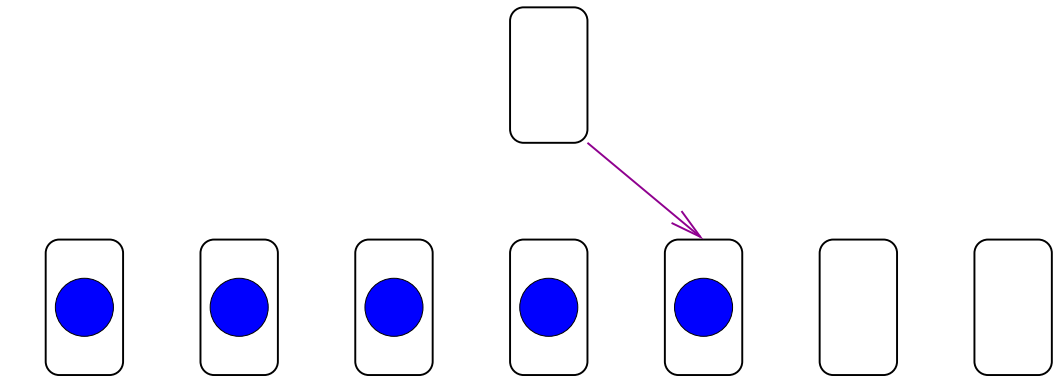
```
Foreach i in hosts do ssh $i uptime
```

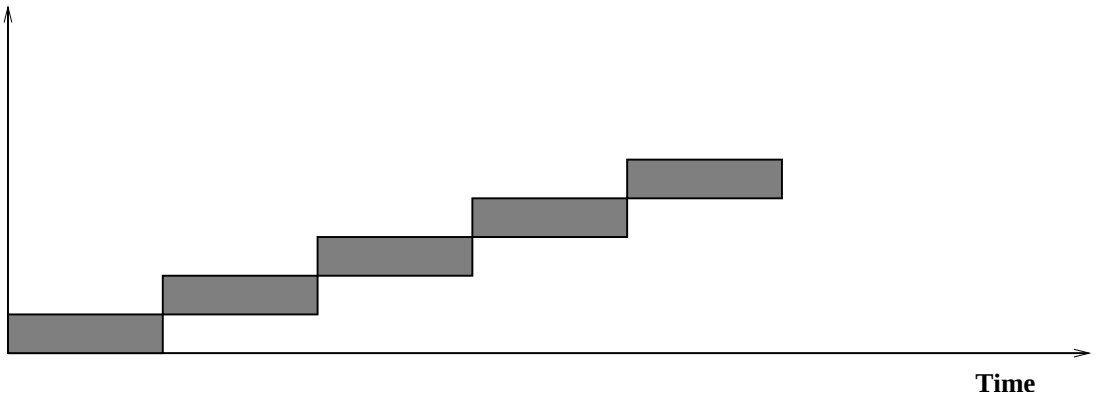




Example of trivial solution

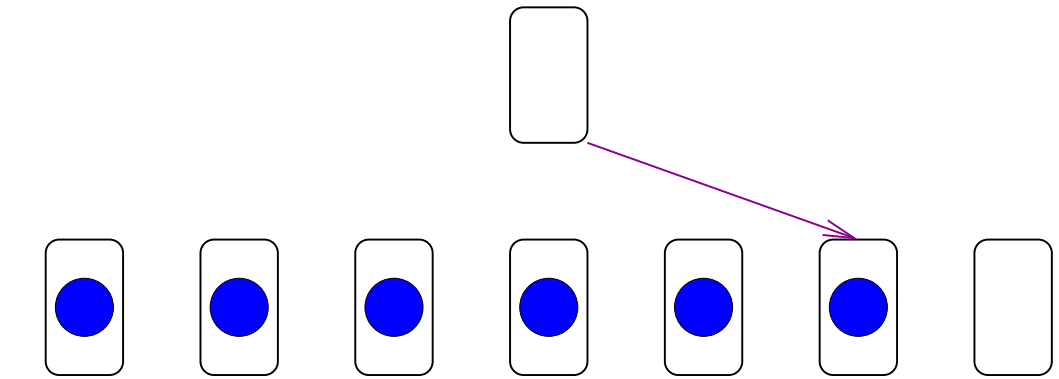
```
Foreach i in hosts do ssh $i uptime
```

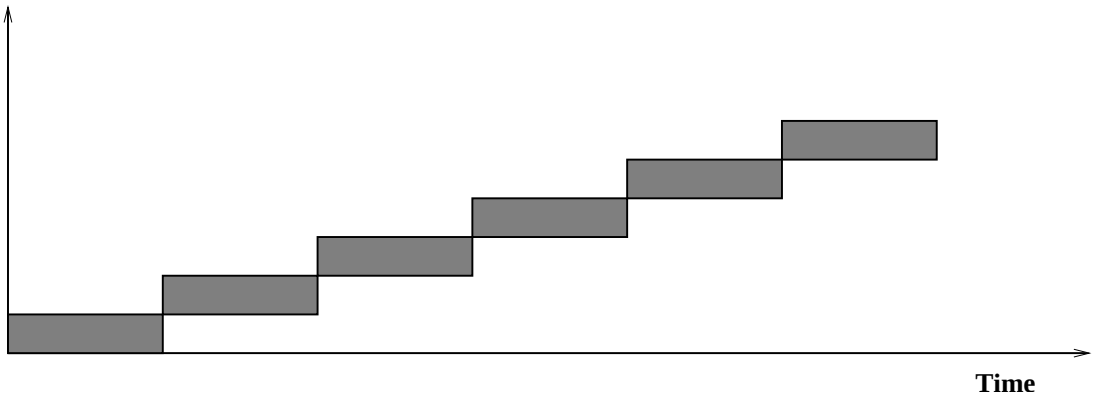




Example of trivial solution

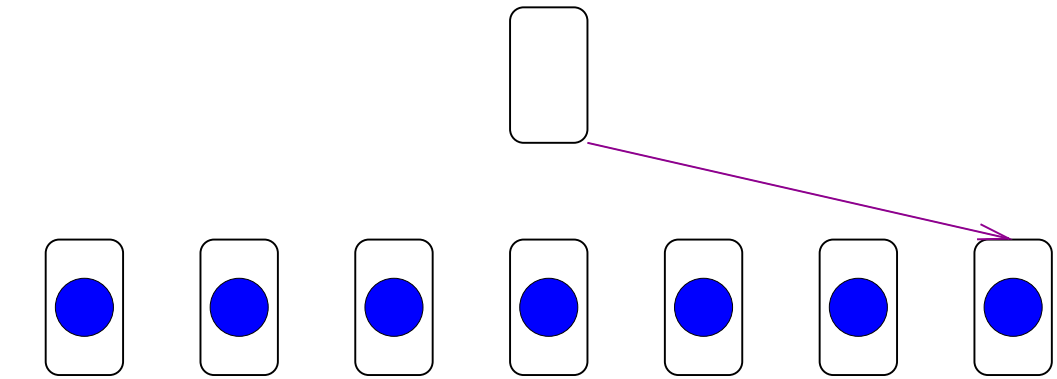
```
Foreach i in hosts do ssh $i uptime
```

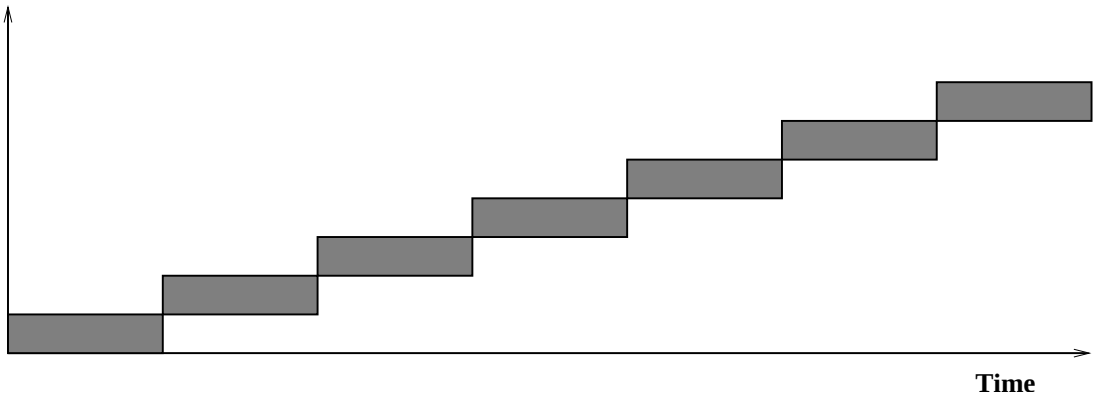




Example of trivial solution

```
Foreach i in hosts do ssh $i uptime
```





ssh takes about:
$100ms$
Execution time:
$n \times 100ms$
For 1000 nodes:
$1mn40$

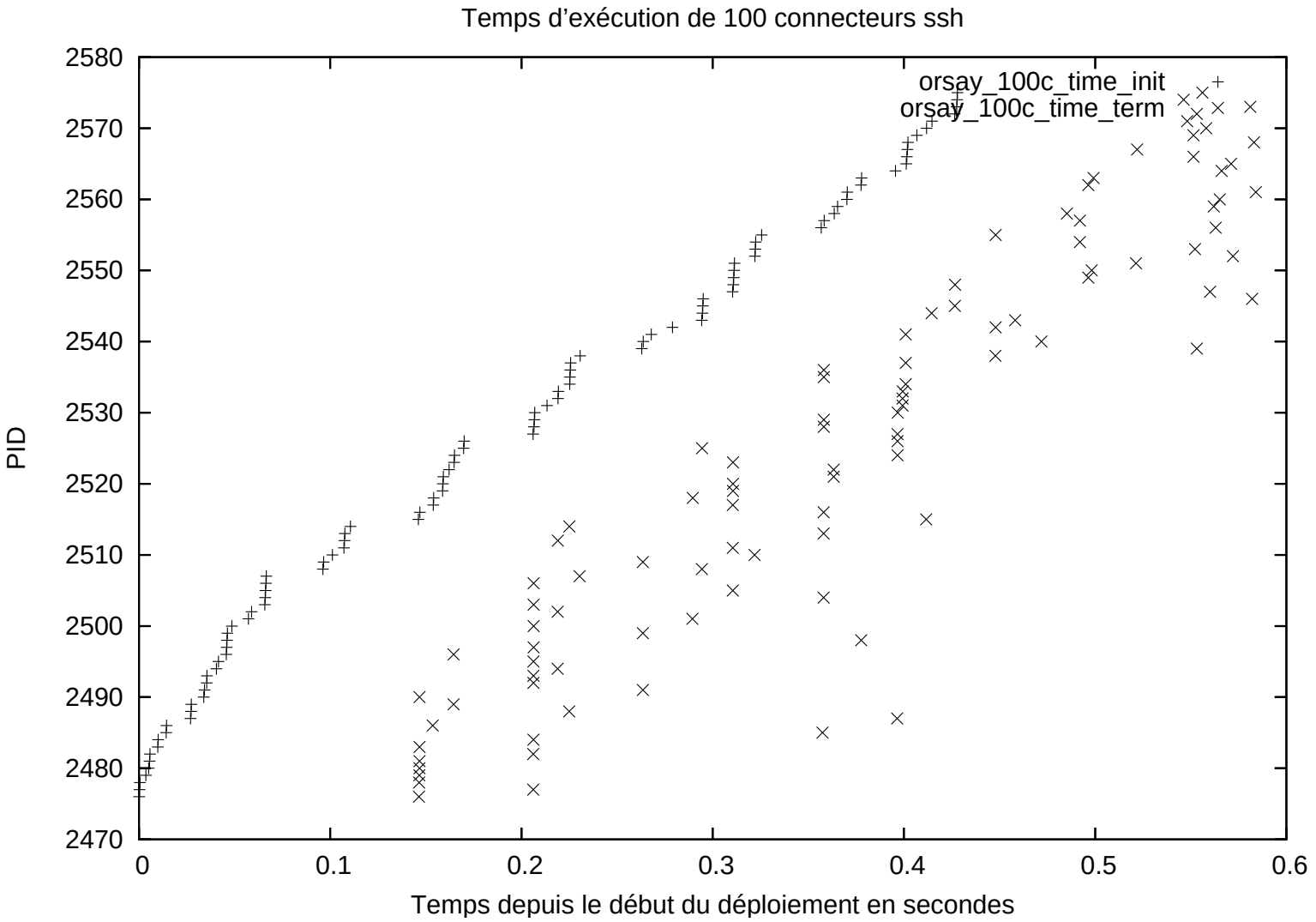
Optimization seems simple

The deployment is embarrassingly parallel

- just create one process (or thread) for each `ssh`
- all the connections will be initiated in parallel

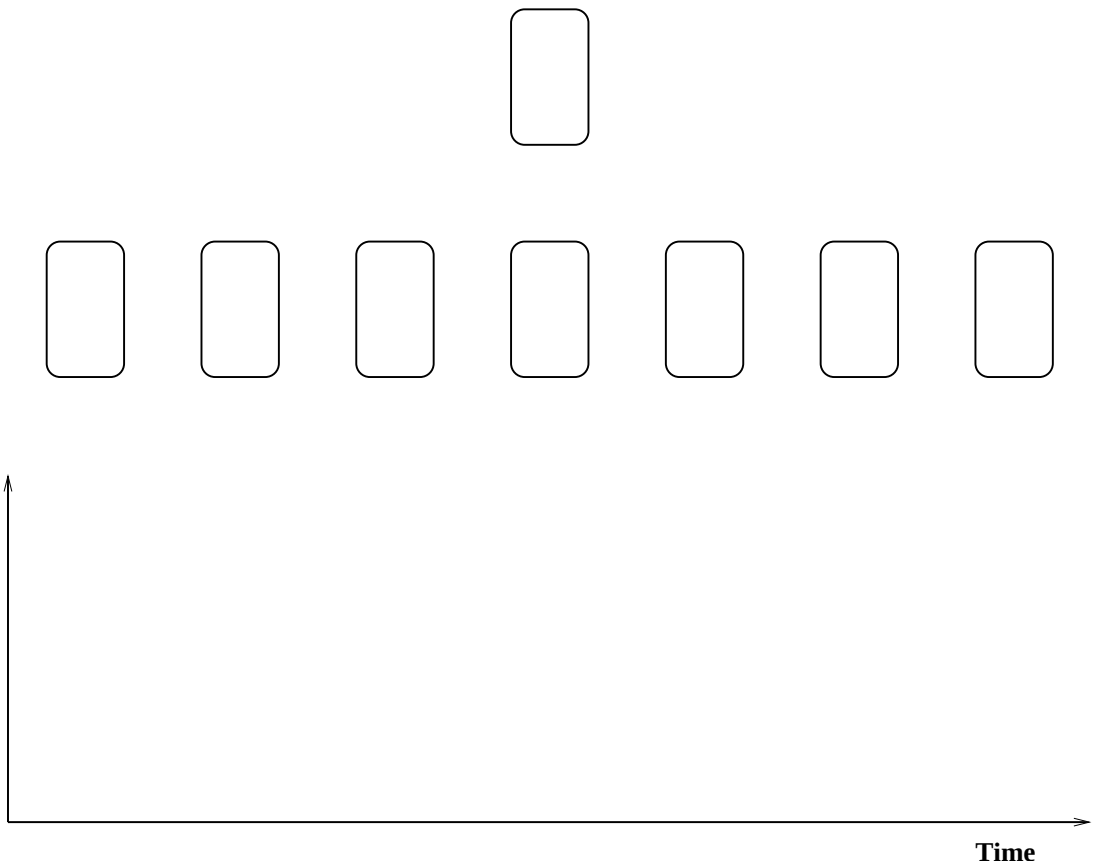
..but reality is more complex than this

Experiment with 100 connectors launched in parallel



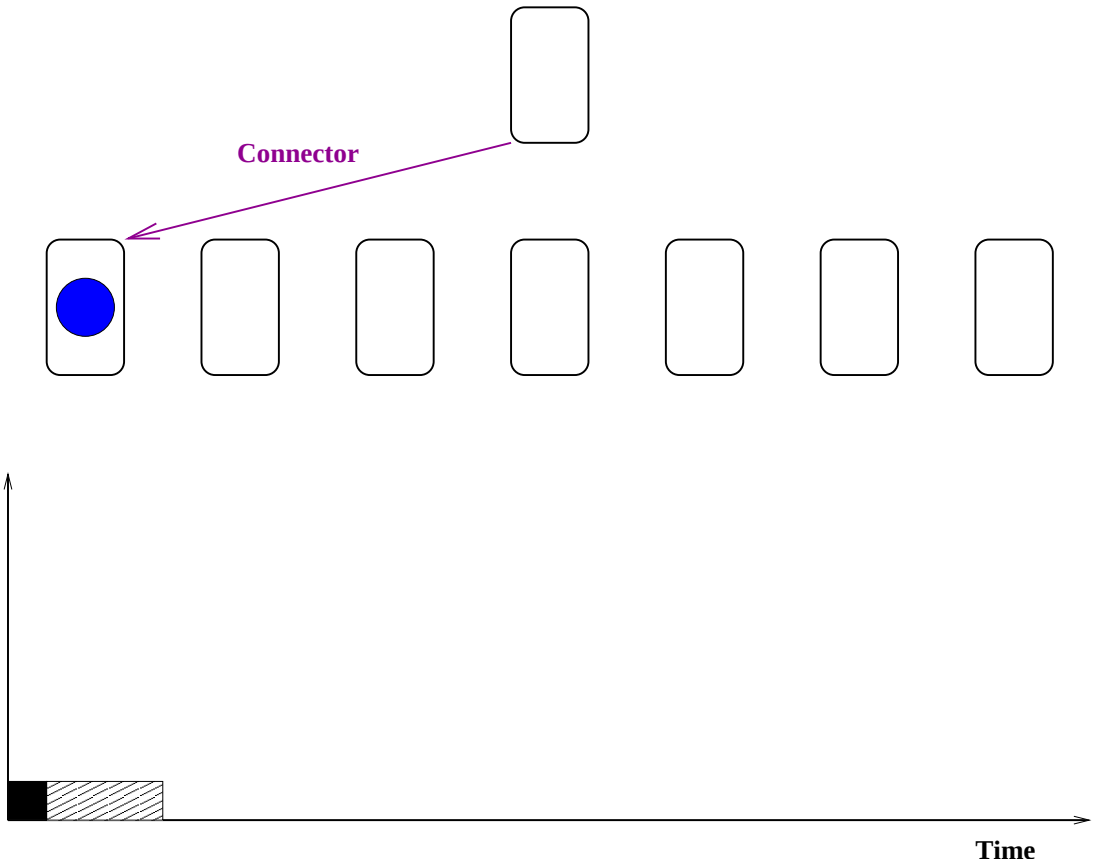
Local parallelization naturally pipelined by the scheduler

```
Foreach i in hosts do fork ssh $i uptime
```



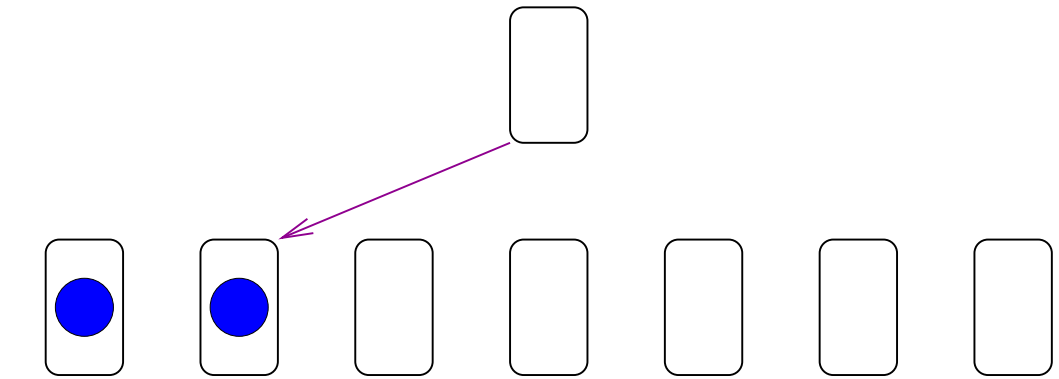
Local parallelization naturally pipelined by the scheduler

```
Foreach i in hosts do fork ssh $i uptime
```



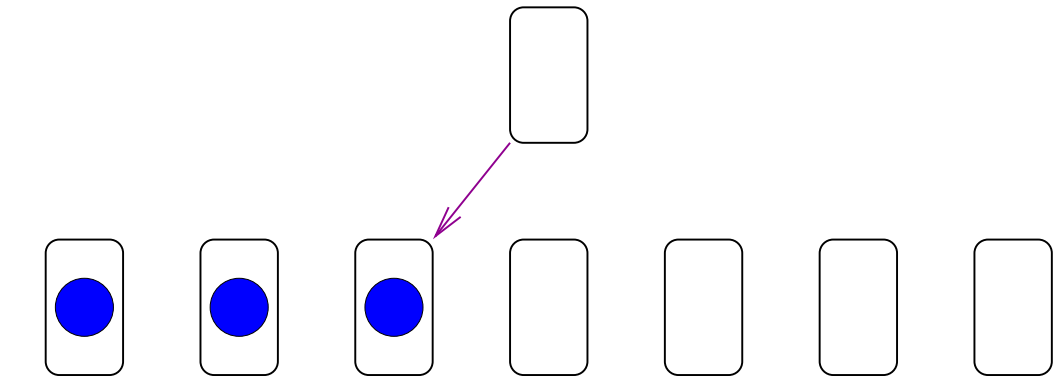
Local parallelization naturally pipelined by the scheduler

```
Foreach i in hosts do fork ssh $i uptime
```



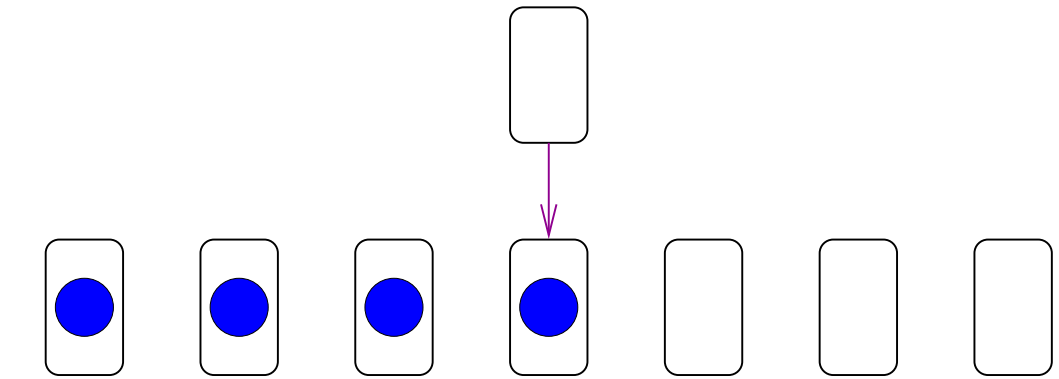
Local parallelization naturally pipelined by the scheduler

```
Foreach i in hosts do fork ssh $i uptime
```



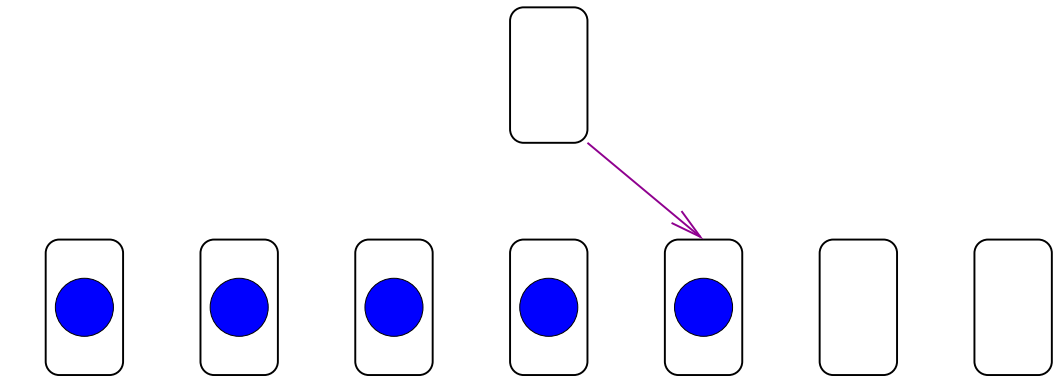
Local parallelization naturally pipelined by the scheduler

```
Foreach i in hosts do fork ssh $i uptime
```



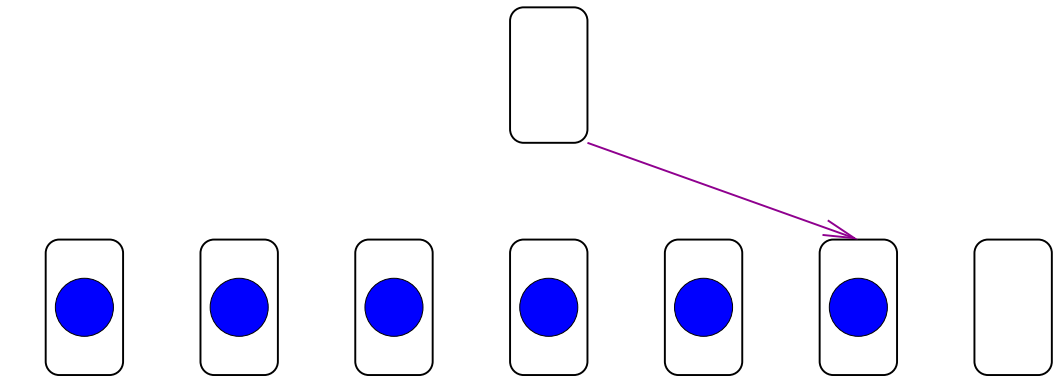
Local parallelization naturally pipelined by the scheduler

```
Foreach i in hosts do fork ssh $i uptime
```



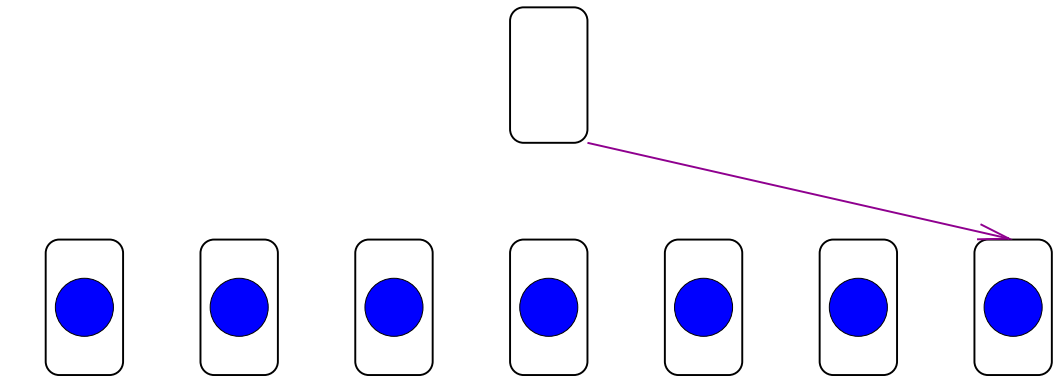
Local parallelization naturally pipelined by the scheduler

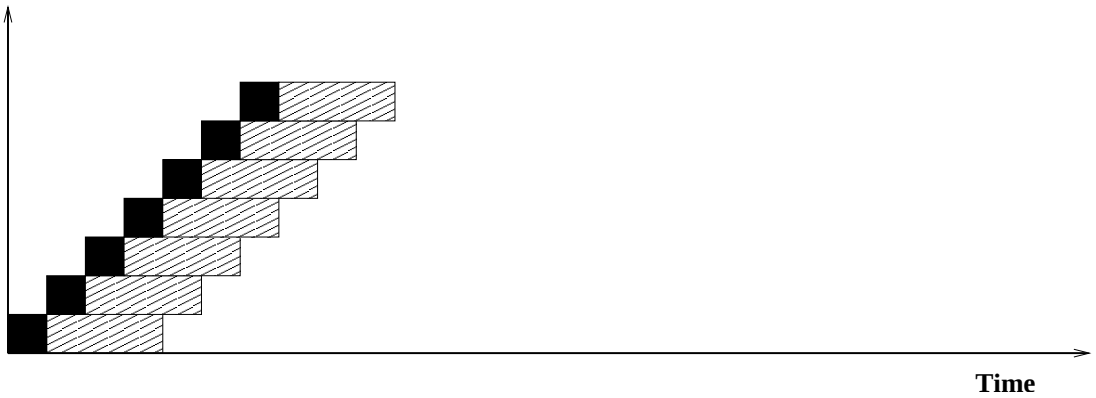
```
Foreach i in hosts do fork ssh $i uptime
```



Local parallelization naturally pipelined by the scheduler

```
Foreach i in hosts do fork ssh $i uptime
```





ssh pipeline shift:
$5ms$
Execution time:
$n \times 5ms + 95ms$
For 1000 nodes:
$5,1s$
Gain is about
$100/5 = 20$
(constant factor)

How to further optimize the deployment ?

Issues

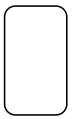
- the cost of local parallelization is still linear
- the initiating machine is a critical resource: authorized number of processes, number of opened file descriptors, ...

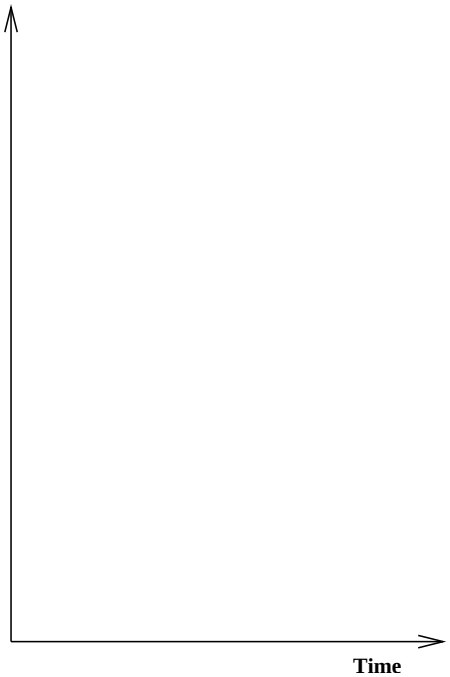
But we can make use of distribution

- remote execution of the deployment engine itself
- distant node take part of the deployment process
- the deployment engine has to multiplex and redirect I/Os

Work distribution

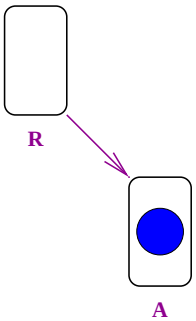
```
while remaining hosts do
      choose i in hosts
      ssh $i taktuk(part(hosts))
exec uptime
```

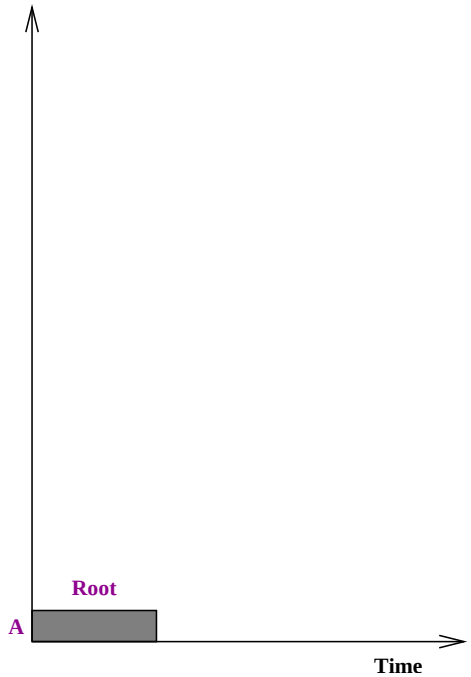




Work distribution

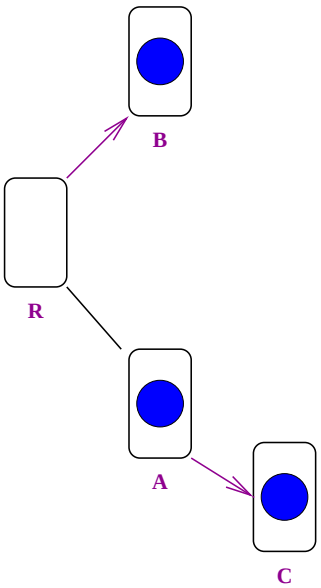
```
while remaining hosts do
      choose i in hosts
      ssh $i taktuk(part(hosts))
exec uptime
```

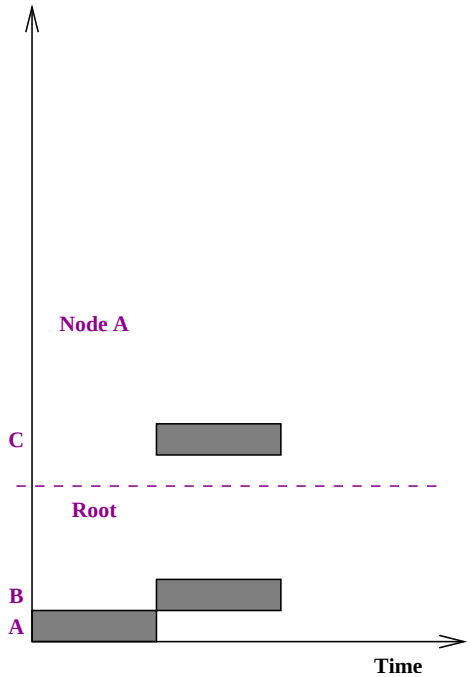




Work distribution

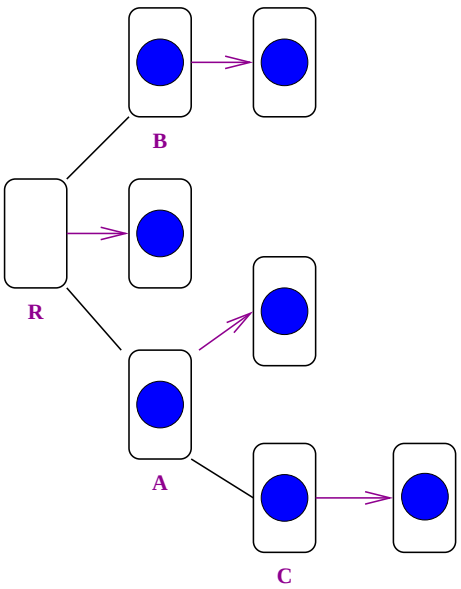
```
while remaining hosts do
      choose i in hosts
      ssh $i taktuk(part(hosts))
exec uptime
```

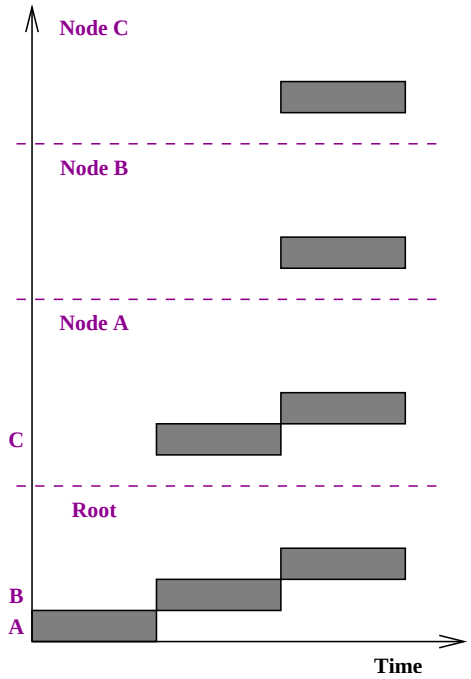




Work distribution

```
while remaining hosts do
      choose i in hosts
      ssh $i taktuk(part(hosts))
exec uptime
```

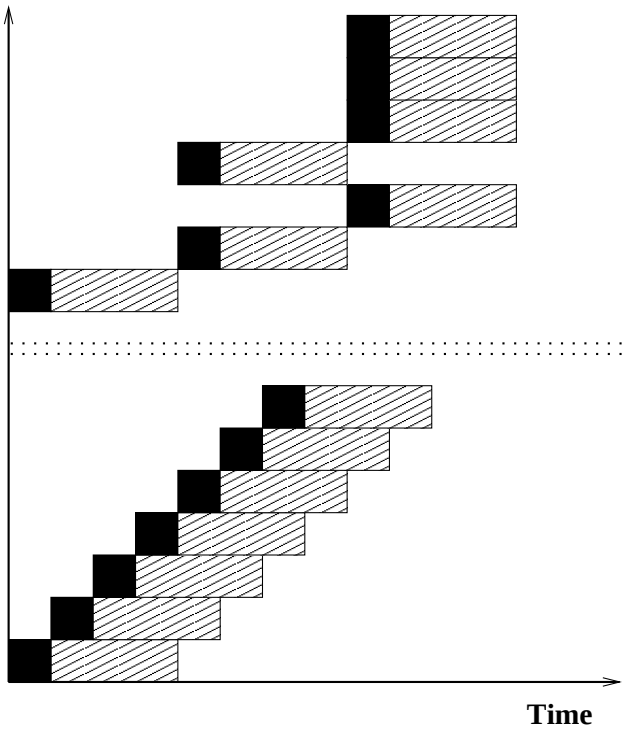


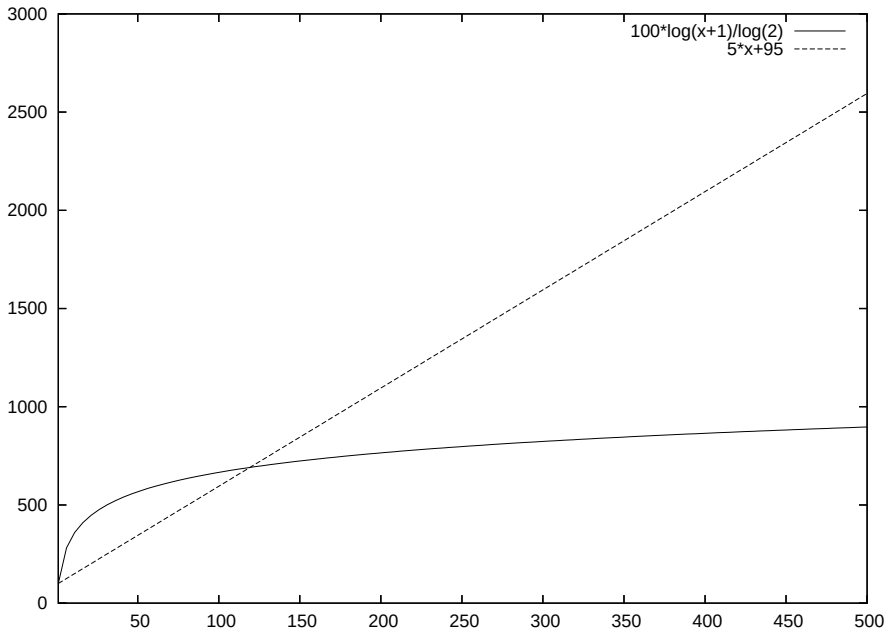


Deployment using a binomial tree
Execution time: $\log_2(n) \times 100ms$
For 1000 nodes: $1s$
without overhead of the engine
Gain:
Logarithmic factor

Is work distribution optimal ?

Obviously not, for small node counts

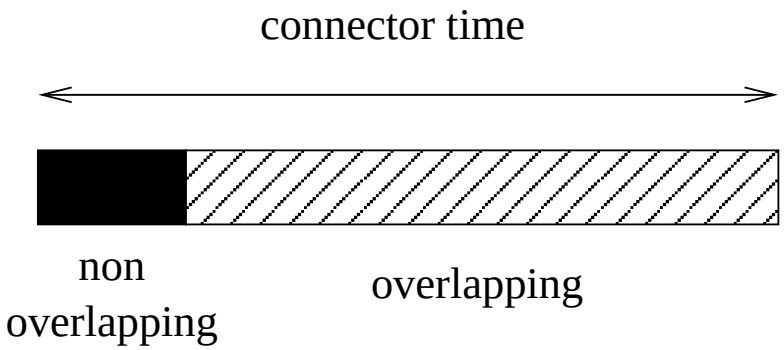




Connector model

A connector (`ssh`) can be abstracted by 2 parts

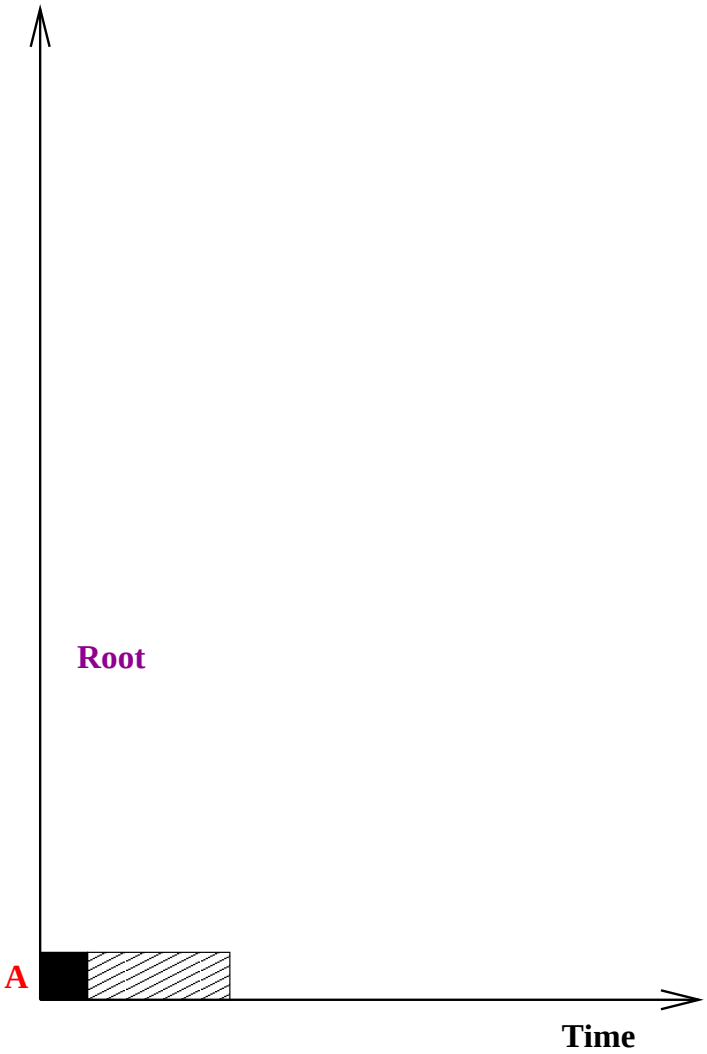
- a non overlapping part (protocol computation)
- an overlapping part (wait)



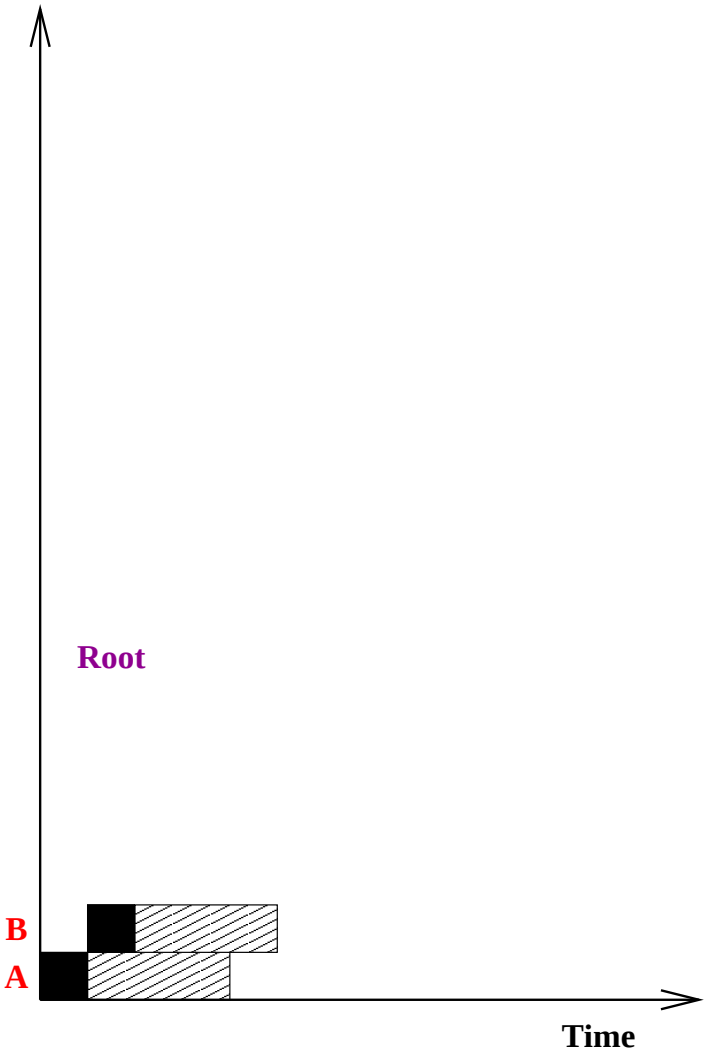
Similar to the postal model

- optimal schedule in the literature: ASAP
- polynomially computable by a greedy algorithm

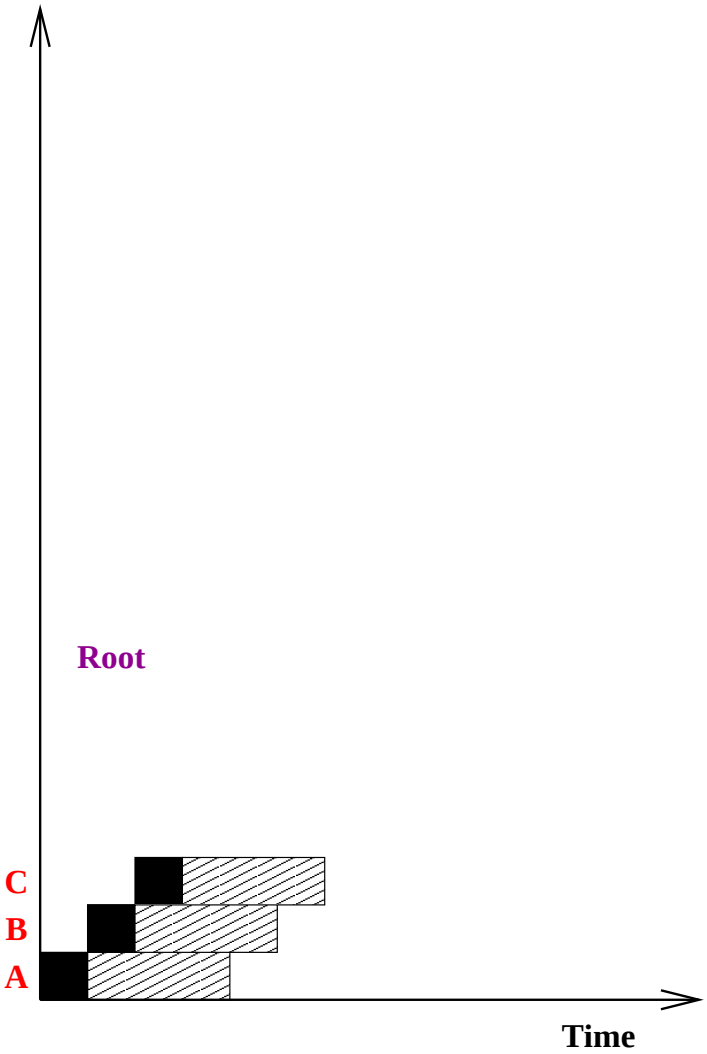
Optimal deployment



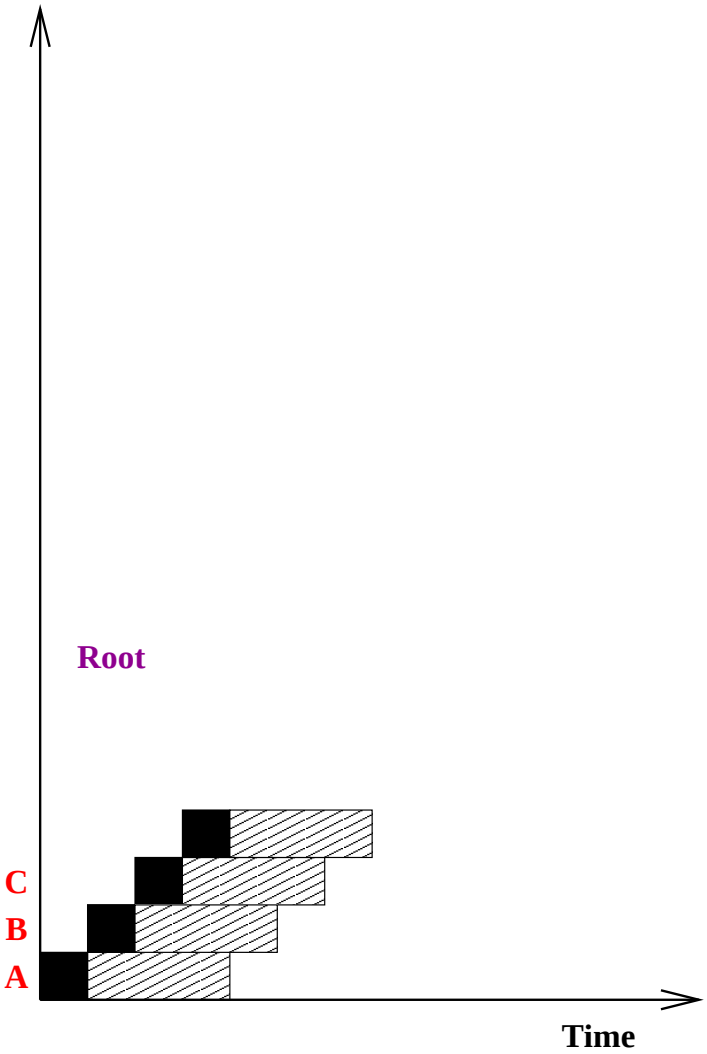
Optimal deployment



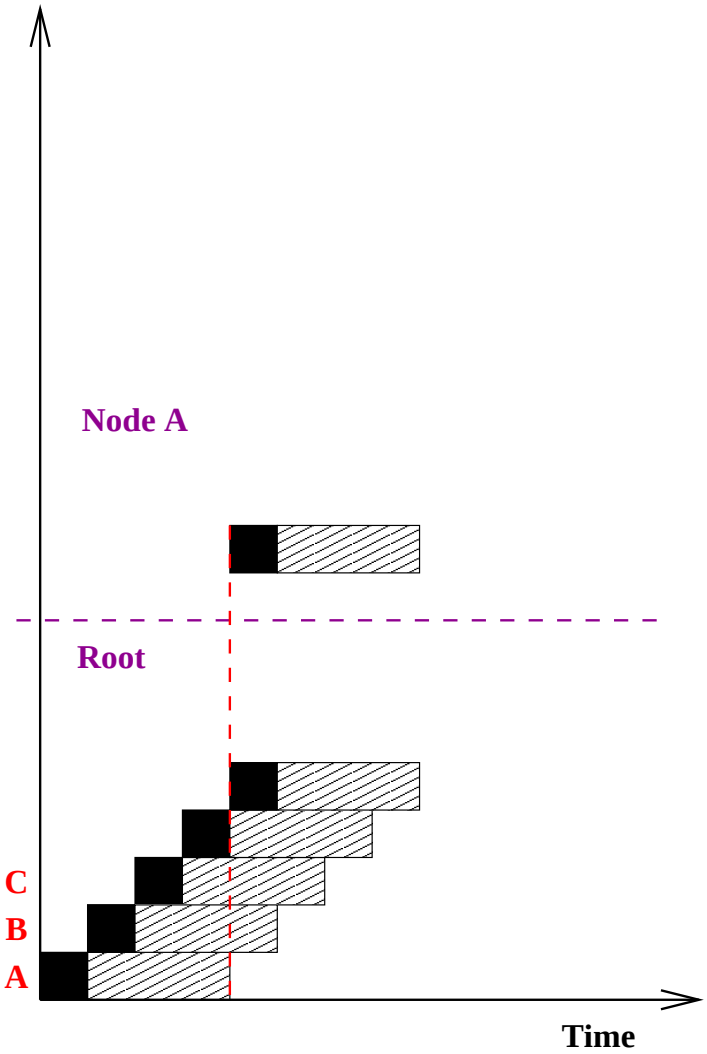
Optimal deployment



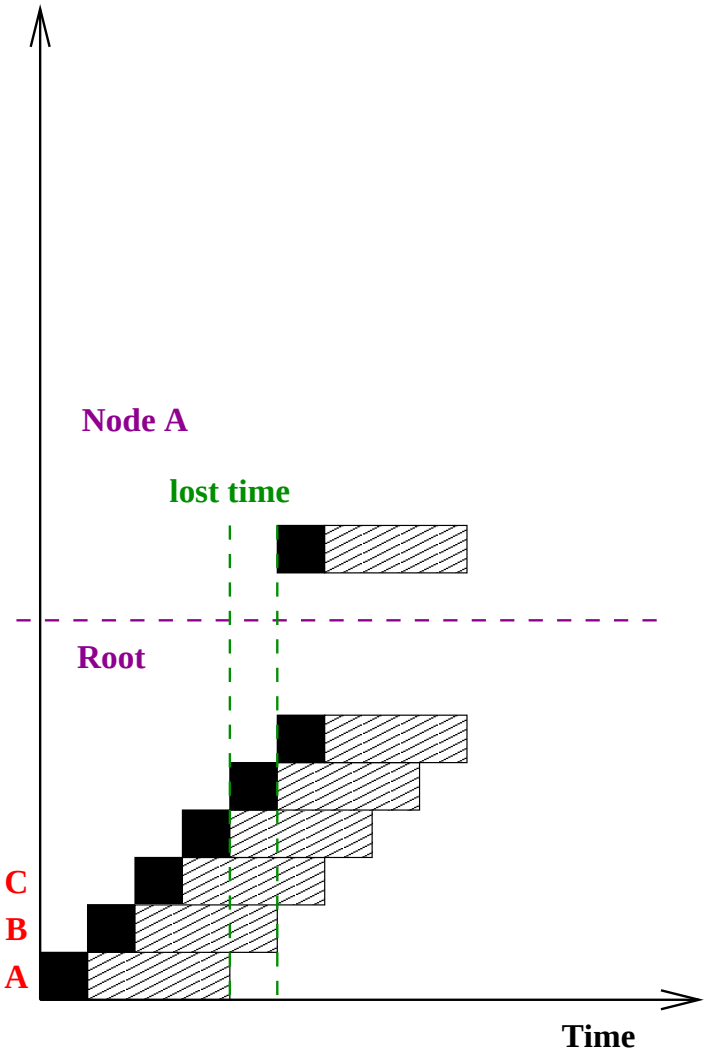
Optimal deployment



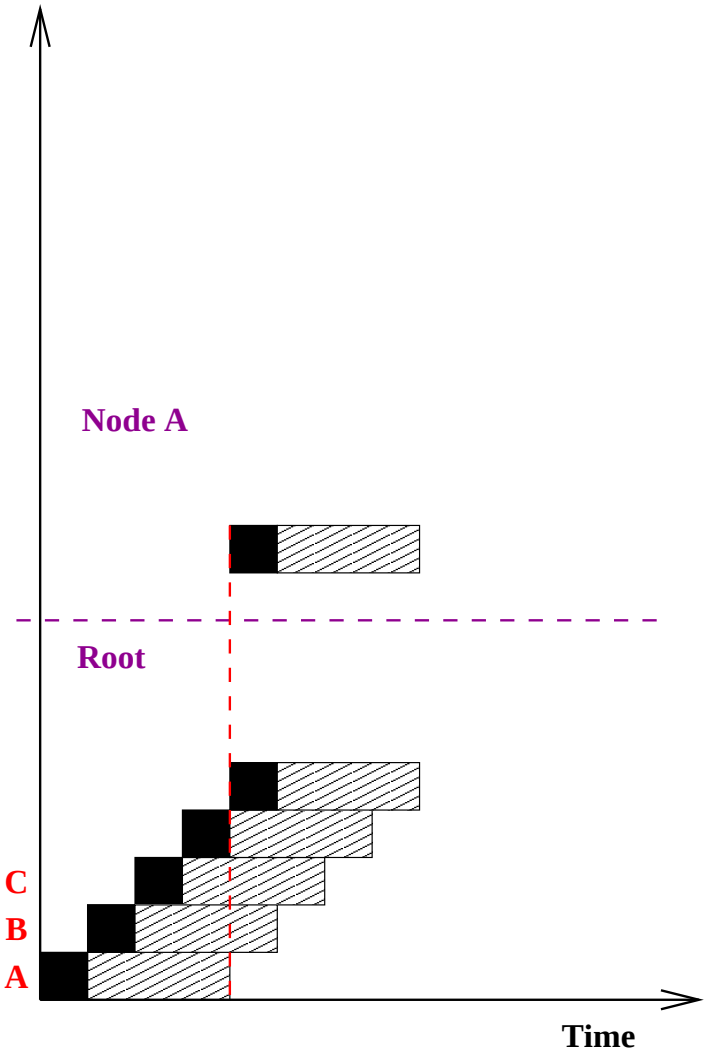
Optimal deployment



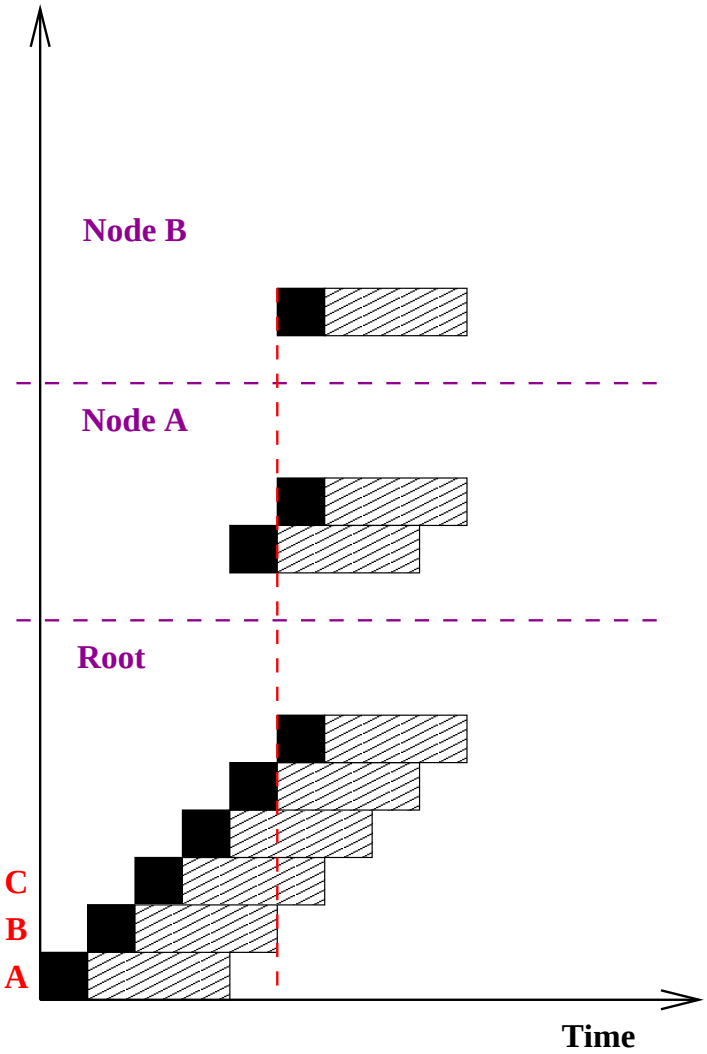
Optimal deployment



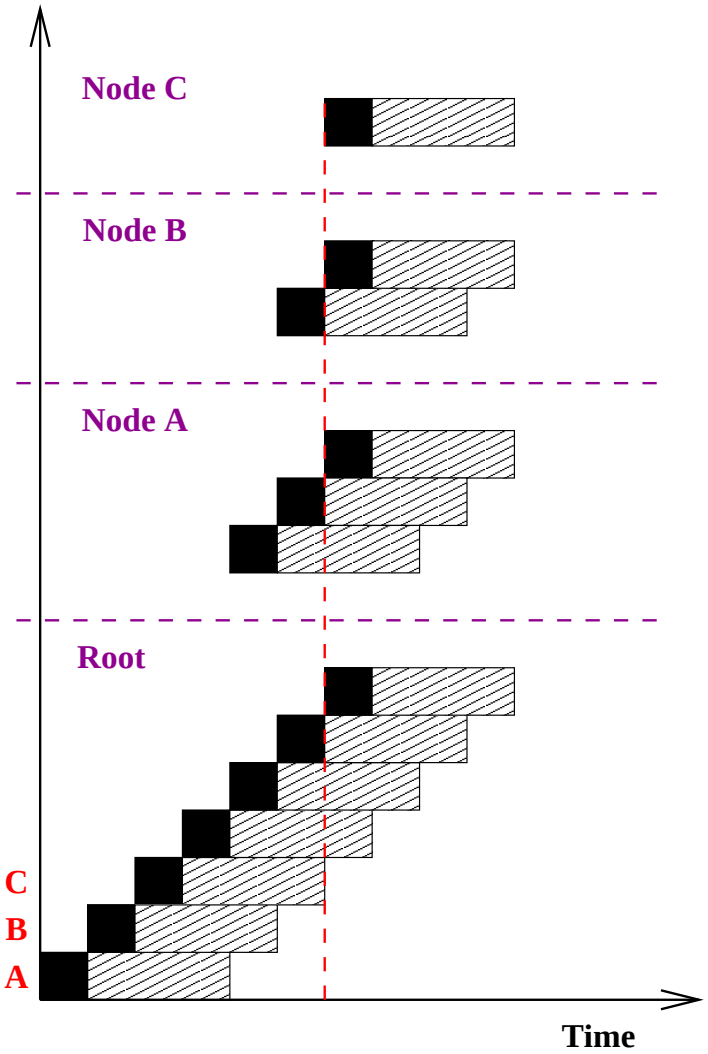
Optimal deployment



Optimal deployment



Optimal deployment



Execution time:
inverse of a
generalized
fibonacci sequence
For 1000 nodes:
$0,36s$
without overhead
of the engine

Outline

1. Motivations
   - Introduction
   - Needs
2. Deployment
   - Parallelization
   - Distribution
   - Optimal deployment
3. Dynamic deployment
   - Dynamic environment
   - Pipeline size evaluation
   - The new TakTuk engine
   - Using TakTuk

Dynamic environment

The performance of a node in a cluster vary

- local hanged processes (zombies, infinite loop)
- external contention (network, centralized services)
- cache effects, swap
- other users
- heat, cosmic rays
- ...

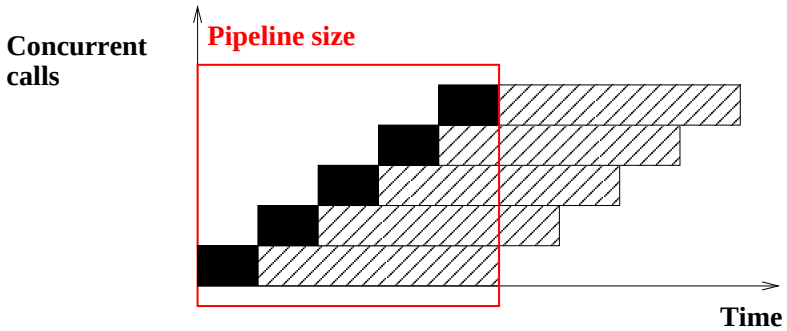
The nodes cannot be considered as homogeneous

Optimal postal solution does not hold anymore

Dynamic deployment

Combine dynamically local parallelization and distribution :

- try to do things ASAP
- nodes initiate the proper number of parallel connections



  This number should match the pipeline size (local parallelization window)

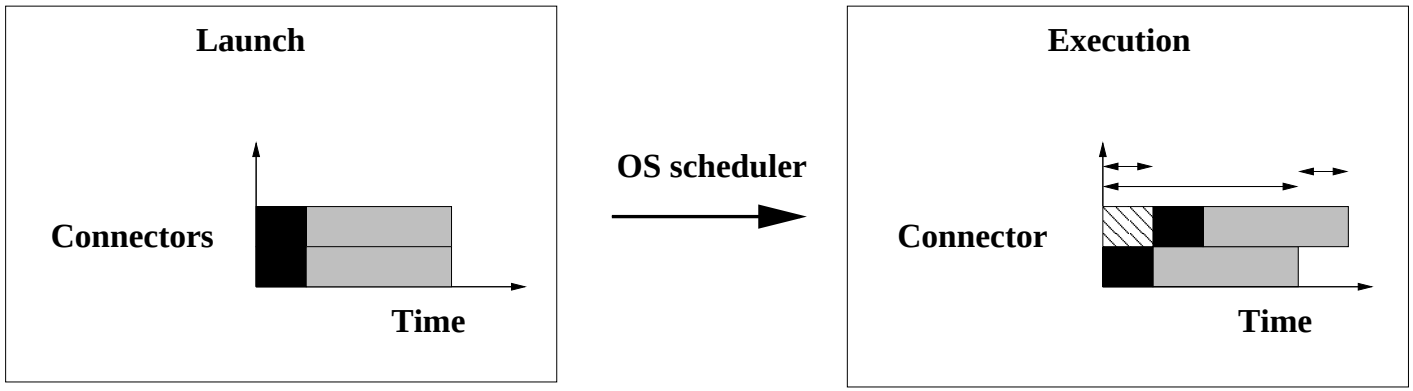
- idle nodes get remaining deployment tasks by work stealing

Need to evaluate the pipeline size for good work balance

Direct measure

We could directly measure the relevant data [J. Bourcier 2004]:

- connector execution time
- pipeline shift

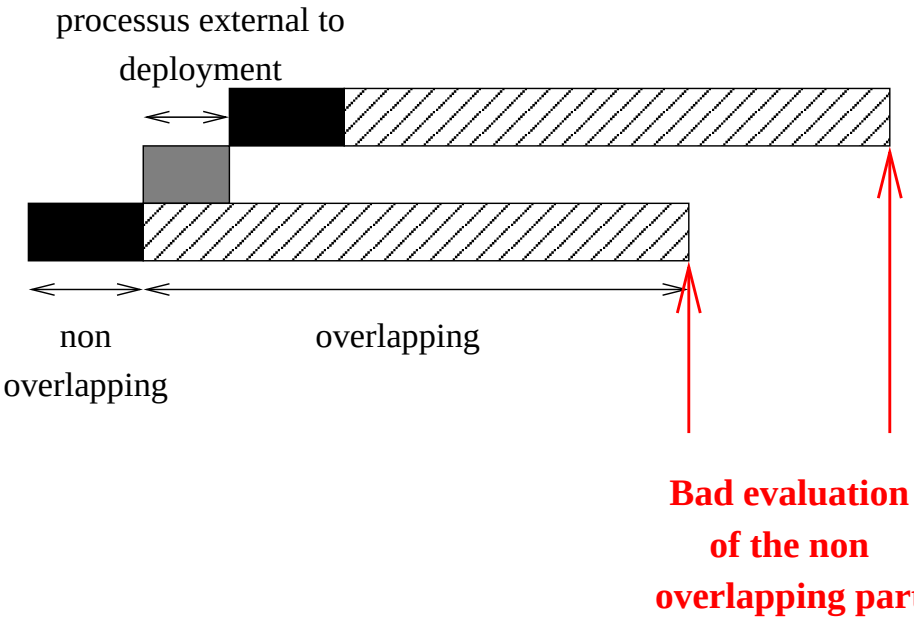


More reactive (100ms) than load average but :

- do not work on SMPs
- not sensible to other load sources in the system
- do not handle external contentions

Example of perturbation in the system

When a process external to the deployment is scheduled



The pipeline shift is not correctly computed

The size underestimated

Use UNIX statistics

Get UNIX process statistics [B. Claudel 2005]:

- Unloaded system
  - user time (non overlapping part)
  - real time (total connector length)

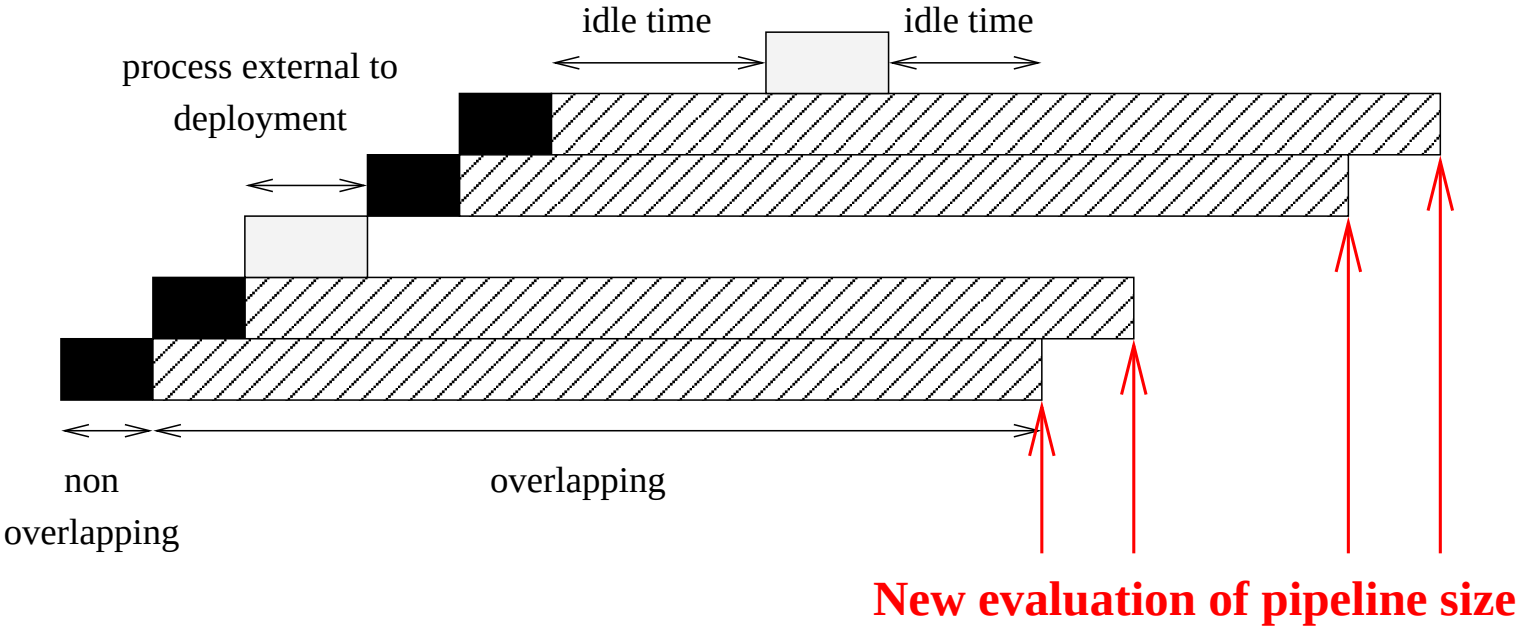
  Derive the ideal pipeline size : total time / user time

  Might be overestimated because of system load
- General case
  - system idle time

  Derive the best pipeline size : idle time / user time

Adaptability

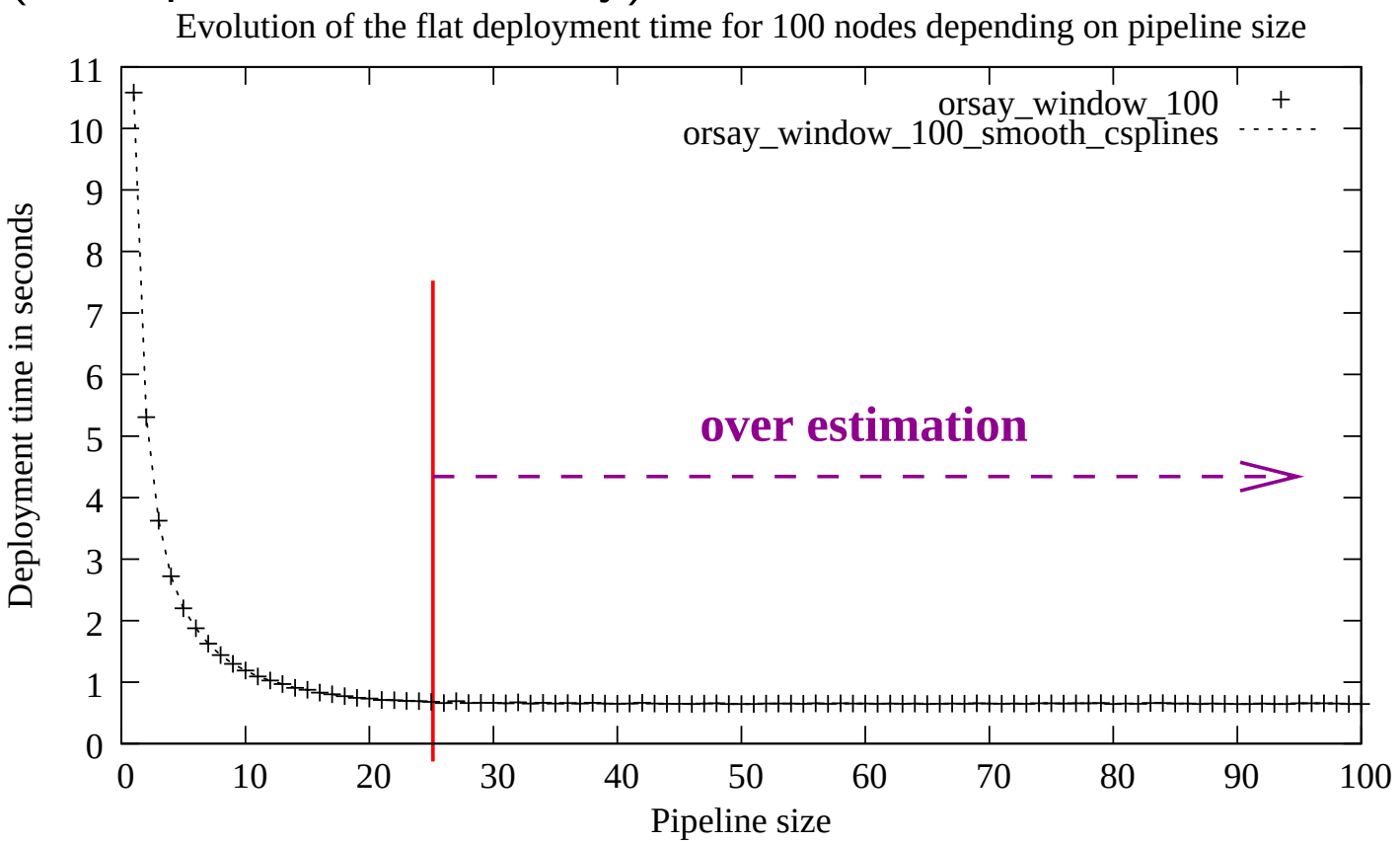
Recompute the evaluation at the end of each connector :



Reacts dynamically to load variations

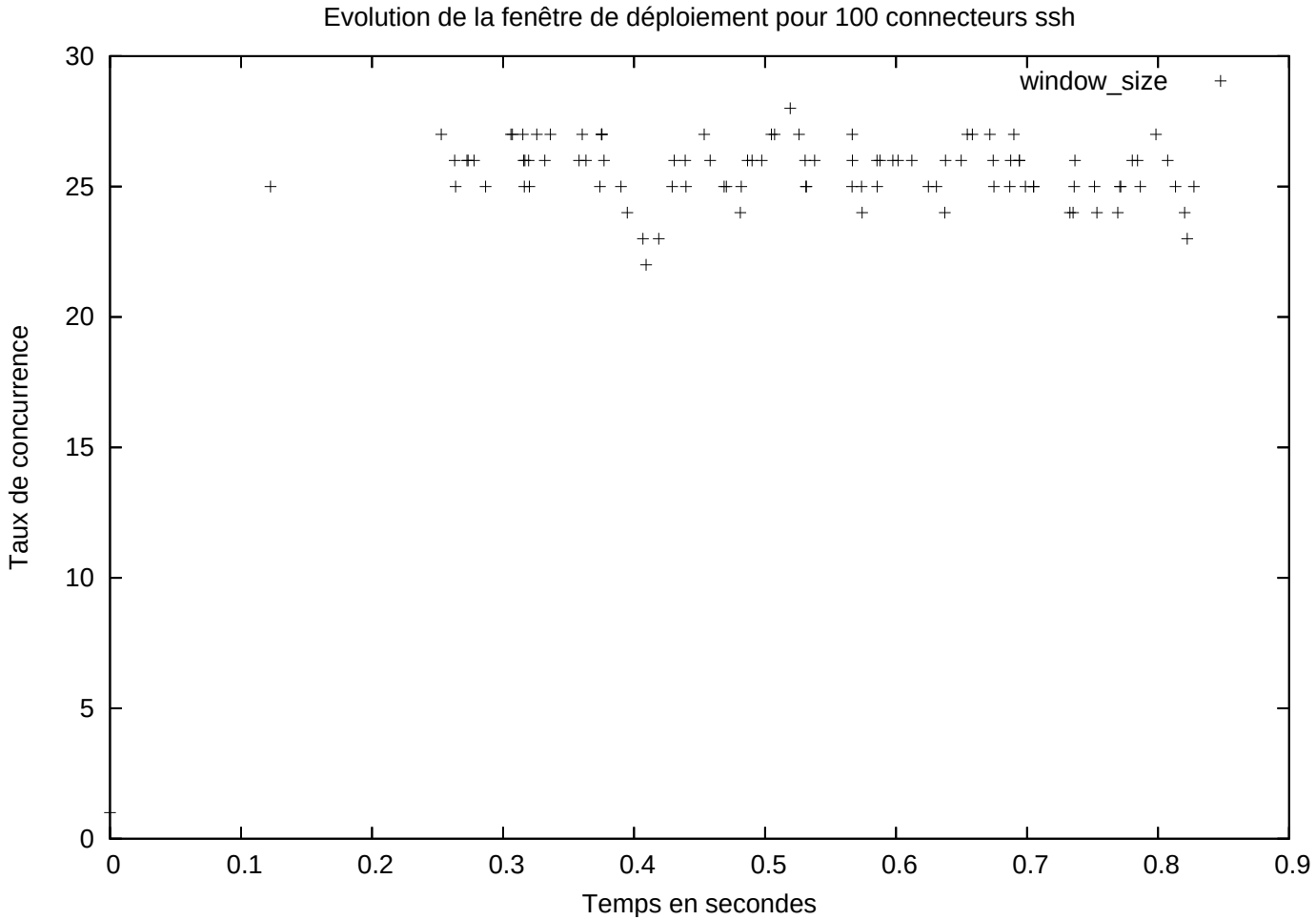
Static evaluation of pipeline size

Measure of the best window size on unloaded system (local parallelization only)



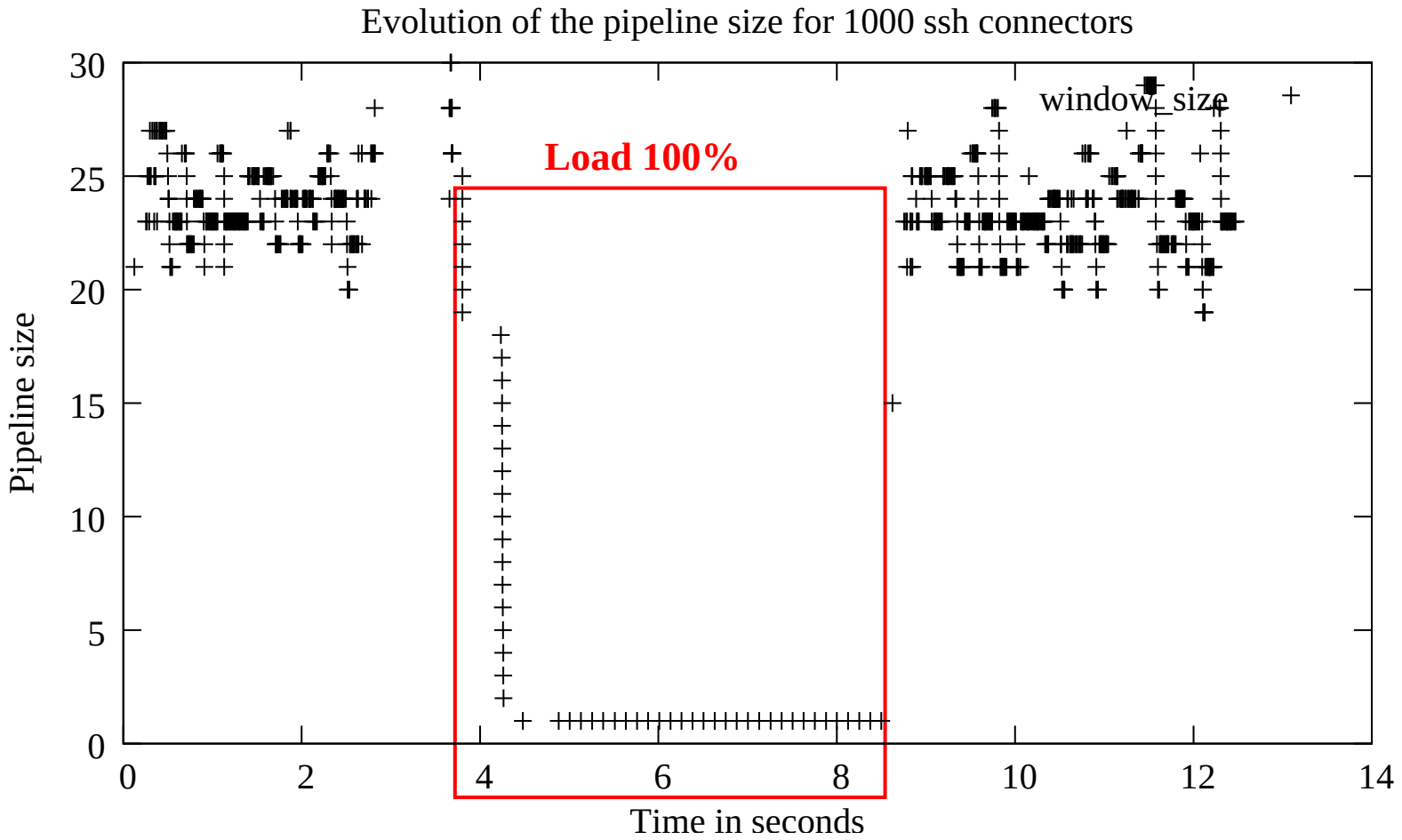
Dynamic evaluation of pipeline size

Using the general method on unloaded system



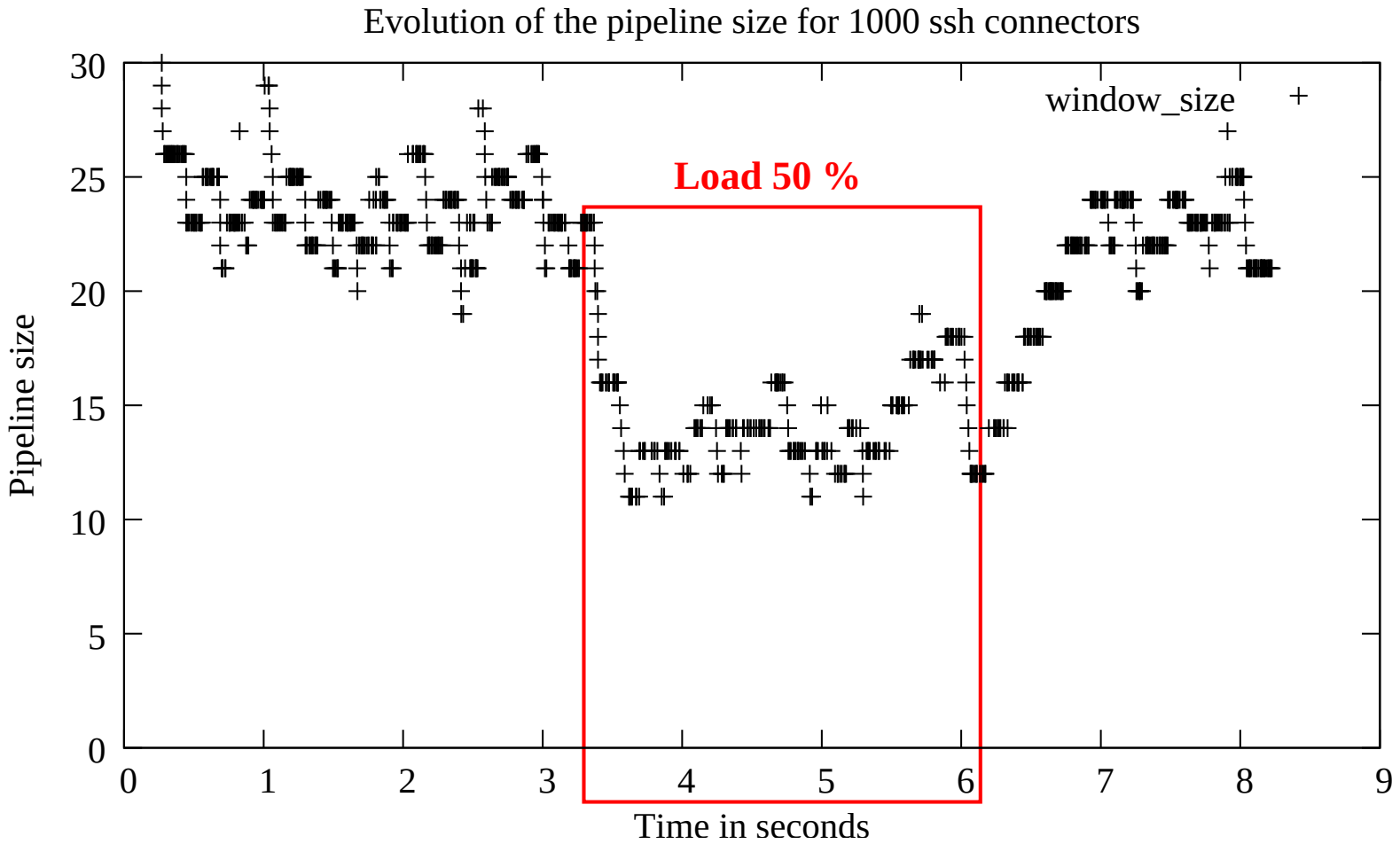
Dynamic evaluation of pipeline size

Loaded system (100%)



Dynamic evaluation of pipeline size

Loaded system (50%)



The old TakTuk engine

The TakTuk library [C. Martin 2003] does dynamic deployment:

- able to deploy itself on remote nodes
- communication layer between TakTuk instances
- I/O redirection
- combine local parallelization and distribution
- evaluates pipeline size by looking at the load average

Issues:

- synchronisation issues, needs deep debugging
- not highly configurable, lack of flexibility
- executable has to be installed on all remote nodes
- pipeline size evaluation not reactive (a few seconds !)
- do not take into account external contention

The new TakTuk engine

Completely rewritten in Perl by G. Huard, about to be released

- can deploy itself and spread its own code to remote nodes
- architecture independent (tested on x86, PPC, IA-64)
- nodes logical numbering and multicast communication layer
- I/O and commands status multiplexing to the root
- configurable mechanics (window, timeouts, ...), I/O templates
- distribution using adaptive work-stealing algorithm
- tested on larger scale, more stable than previous engine

Basic usage

Identical execution on some remote nodes

```
taktuk -m toto.nowhere.com -m tata.nowhere.com
  -m tutu.nowhere.com broadcast exec [hostname]
```

Will output something like

```
toto.nowhere.com-1: hostname (4164): output > toto.nowhere.com
toto.nowhere.com-1: hostname (4164): status > 0
tutu.nowhere.com-3: hostname (1468): output > tutu.nowhere.com
tutu.nowhere.com-3: hostname (1468): status > 0
tata.nowhere.com-2: hostname (3290): output > tata.nowhere.com
tata.nowhere.com-2: hostname (3290): status > 0
```

Basic usage

Do not necessarily require an installed TakTuk on each remote host

```
taktuk -s -m toto.nowhere.com -m tata.nowhere.com
  -m tutu.nowhere.com broadcast exec [hostname]
```

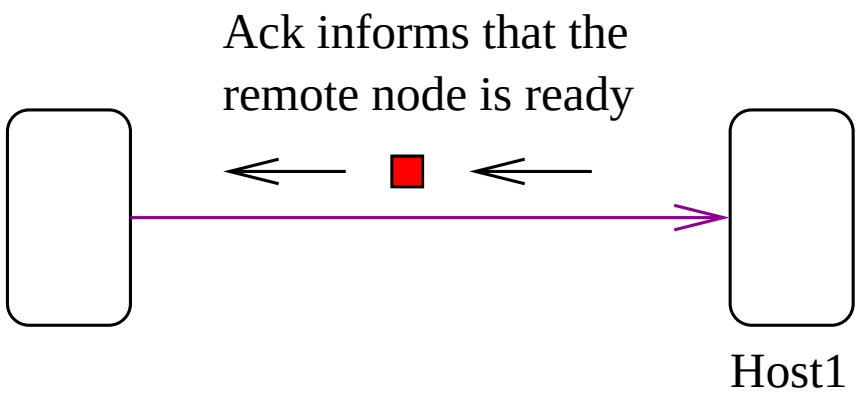
Effect of the `-s` switch

Basic usage

Do not necessarily require an installed TakTuk on each remote host

```
taktuk -s -m toto.nowhere.com -m tata.nowhere.com
  -m tutu.nowhere.com broadcast exec [hostname]
```

Effect of the `-s` switch

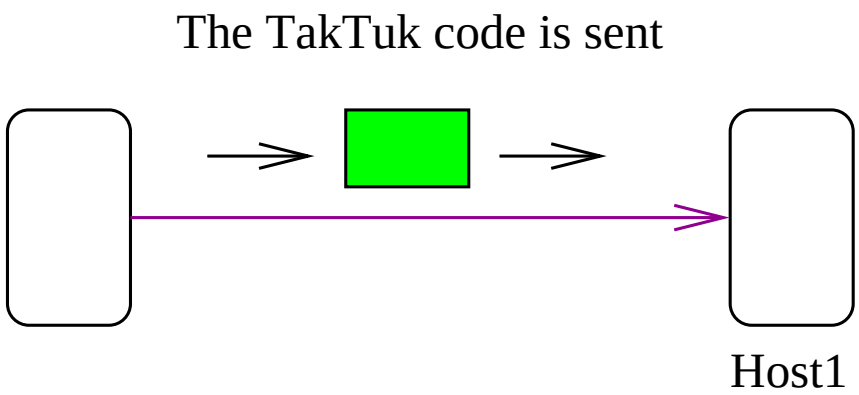


Basic usage

Do not necessarily require an installed TakTuk on each remote host

```
taktuk -s -m toto.nowhere.com -m tata.nowhere.com
  -m tutu.nowhere.com broadcast exec [hostname]
```

Effect of the `-s` switch

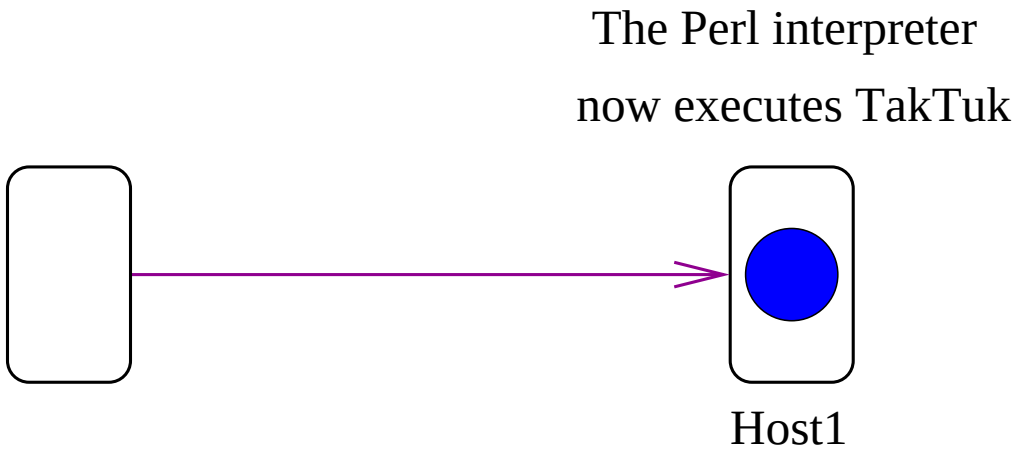


Basic usage

Do not necessarily require an installed TakTuk on each remote host

```
taktuk -s -m toto.nowhere.com -m tata.nowhere.com
  -m tutu.nowhere.com broadcast exec [hostname]
```

Effect of the `-s` switch

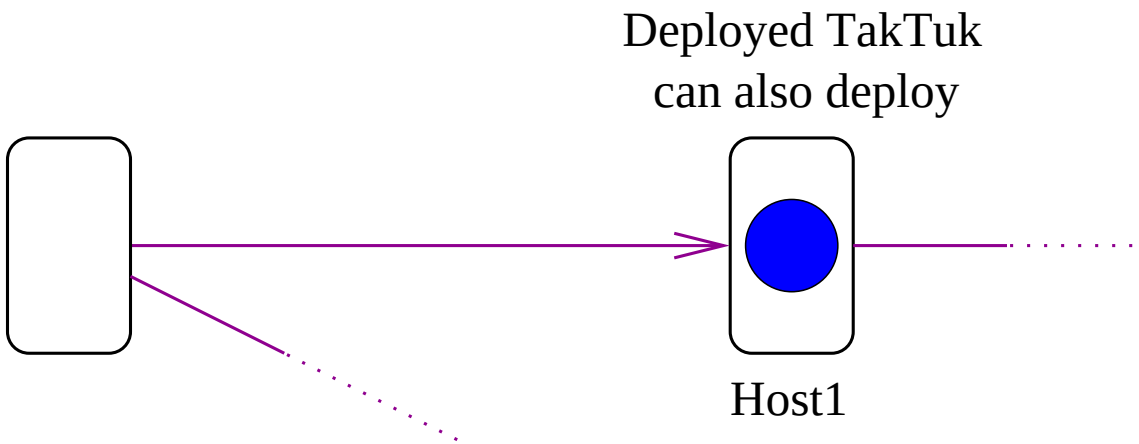


Basic usage

Do not necessarily require an installed TakTuk on each remote host

```
taktuk -s -m toto.nowhere.com -m tata.nowhere.com
  -m tutu.nowhere.com broadcast exec [hostname]
```

Effect of the `-s` switch



Basic usage

Commands can also be given:

- interactively
- on a per host basis

```
taktuk -m toto.nowhere.com -[ exec [hostname] -]
  -m tata.nowhere.com -[ exec [uptime] -]
  -m tutu.nowhere.com -[ exec
    [if [ \$RANDOM -gt 1000 ];then echo ok;fi] -]
```

And the set of remote nodes can be listed in a file

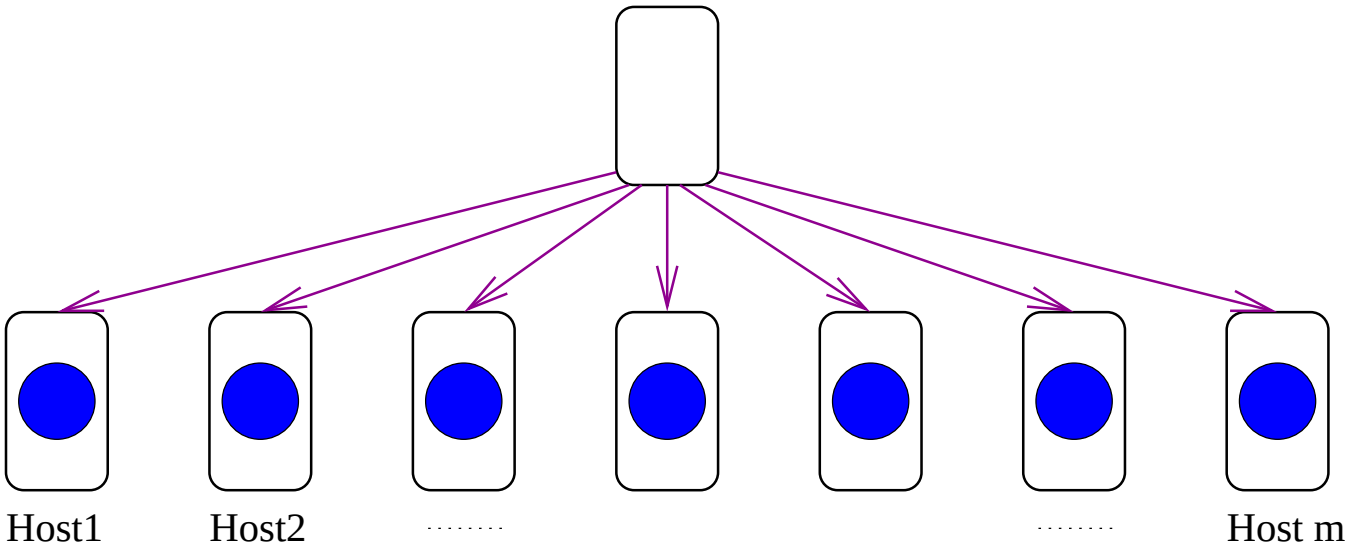
```
taktuk -f $OAR_NODE_FILE broadcast exec [hostname]
```

Flat tree topology

Deployment tree is constructed dynamically by work-stealing

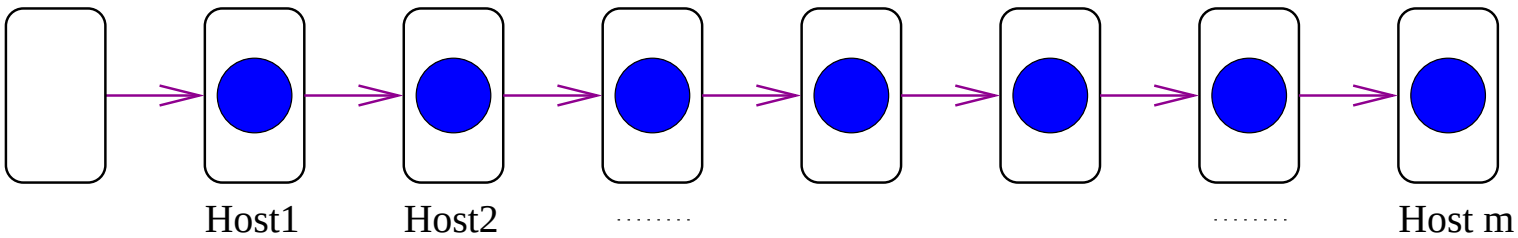
- it can be changed using TakTuk options
- by disabling work-stealing we get a flat tree

```
taktuk -d-1 -m host1 -m host2 ... -m hostm
  broadcast exec [hostname]
```



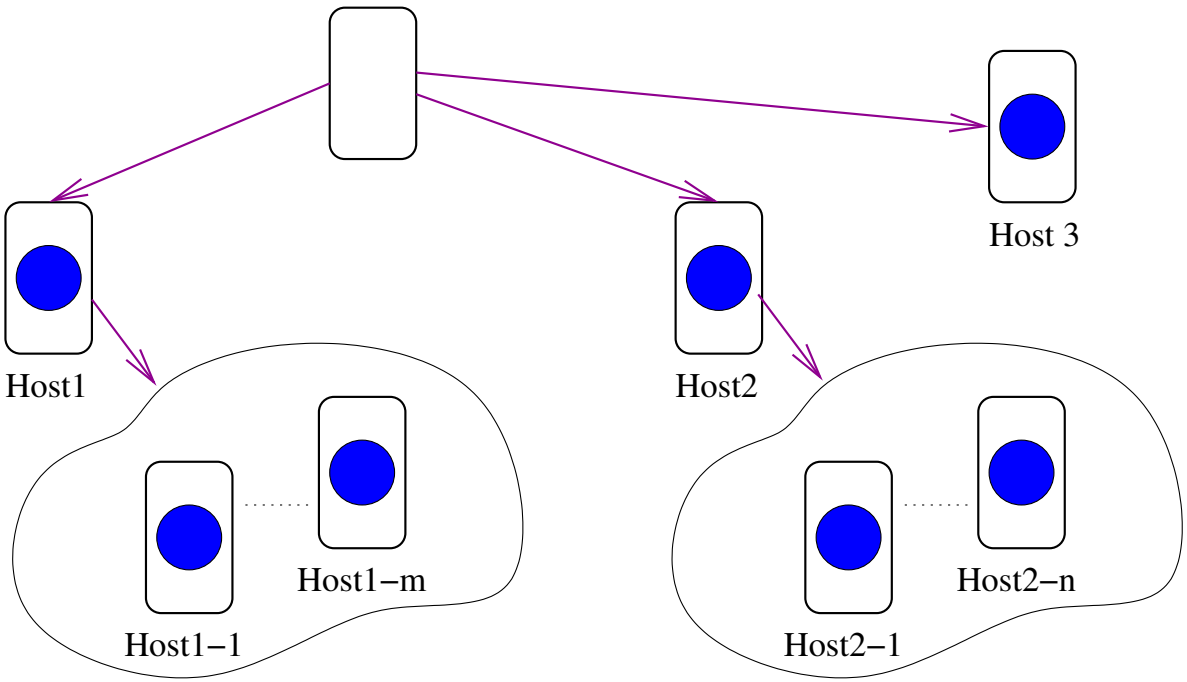
Chain topology

```
taktuk -m host1 -[
          -m host2 -[
            ... -[ -m hostm -] ...
          -]
        -]
  broadcast exec [hostname]
```



Mixed static/dynamic topology

```
taktuk -m host1 -[ -m host1-1 ... -m host1-m -]
       -m host2 -[ -m host2-1 ... -m host2-n -]
       -m host3 broadcast exec [hostname]
```



Communication layer

Locical numbering given using environment variables

- TAKTUK_RANK
- TAKTUK_COUNT

Communication between logical entities

- provided by taktuk Perl package or taktuk_perl command
- send/receive model
- multicast send
- receive can be timeouted

Communication example

- communicating script: communication.pl

```
if ($ENV{TAKTUK_RANK} == 1) {
    if ($ENV{TAKTUK_COUNT} > 1) {
        taktuk::send(to=>2, body=>"Salut a toi");
      }
  }
elsif ($ENV{'TAKTUK_RANK'} == 2) {
    my ($to, $from, $message) = taktuk::recv();
    print "Received $message from $from\n";
  }
```

- TakTuk command

```
taktuk -m host1 -m host2 broadcast taktuk_perl [],
  broadcast file_input [communication.pl]
```

Current TakTuk state

The TakTuk code is about to be released

- new development frozen, only bugfixes
- 3.0-perl-beta15 available for early testing

Gdx used to test TakTuk during the whole development

- direct tests (roughly up to 200 nodes)
- virtualized tests (above 500 instances)

Should be used in the next `OAR` distribution

Ongoing works

Quantitative experiments

- deployment time and scalability
- comparison with other tools (gexec, tentakel, cplant, ...)

Adaptive window for local deployment

- to handle external contention (centralized services: nfs, ldap)
- to adapt to local load (and distribute better)
- we already have solutions (J. Bourcier and B. Claudel)

Work-stealing priorities (based on nodes performance)

- appropriate model needed
- adaptation of the work stealing algorithm

Thanks for your attention

Tool developped at ID laboratory:

`http://www-id.imag.fr/Logiciels/TakTuk/`

`http://taktuk.gforge.inria.fr`

Any question ?