Motivations
Deployment
TakTuk

TakTuk, Adaptive Deployment of Remote Executions

Benot Claudel, Guillaume Huard and Olivier Richard

MOAIS, MESCAL and SARDES INRIA Projects
CNRS LIG Laboratory
Grenoble University, France
The goal is to execute:

```
Foreach host in list_of_hosts do
  ssh $host some_command
```

Where:

- Remote executions are independent: **parallel remote executions**
- Either "root" or any of "list_of_hosts" can execute the ssh: **deployment of remote execution tasks**
Why parallel remote executions?

Nodes administration:
run the same command on all nodes of a platform
  - uptime to grab statistics about the recent machine availability
  - dig, ping, ifconfig ... for network issues diagnostic
  - ...

Parallel applications execution:
run the same executable on all nodes (mpirun does that)
  - Slaves of a master/slave application
  - All participants of a symmetric parallel application
  - Self organizing system (P2P), daemons (monitoring)
  - ...

very frequent during applications development
Needs

**Transparency:**
User should not have to worry about execution mechanics
- Automatization of remote connections to all the machines
- I/O multiplexing to the root node (gathering of result)
- ... whatever the target platform (cluster, grid, ...)

**Efficiency:**
Administration and applications development are interactive tasks
- Time to deploy is critical
- Management of all the nodes has to scale
- ...
Questions addressed in this talk

How to perform efficiently all the remote connections?
- Parallelization?
- Distribution?

How to address heterogeneity?
- Adaptivity?
- Assume some configuration on nodes?
Motivations

- Problem
- Needs

Deployment

- Parallelization
- Distribution
- Optimal

TakTuk

- Engine
- Performance
- Perspectives
Example of trivial solution

Foreach i in hosts do ssh $i uptime

\[\text{\begin{array}{l}
\text{\hspace{1cm}}
\end{array}}\]
Example of trivial solution

Foreach i in hosts do ssh $i uptime

Distant execution (ssh, rsh, ...)

Time

Benot Claudel, Guillaume Huard and Olivier Richard
TakTuk, Adaptive Deployment of Remote Executions
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

```plaintext
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
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
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: 20ms
Execution time: $n \times 20ms + 80ms$
For 1000 nodes: $\approx 20s$
Gain is about $100/20 = 5$
(constant factor)
Motivations
Deployment
TakTuk
Parallelization
Distribution
Optimal

Experiment with 100 connections run in parallel

```
<table>
<thead>
<tr>
<th>Process number</th>
<th>Execution time (in secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0.1</td>
</tr>
<tr>
<td>20</td>
<td>0.2</td>
</tr>
<tr>
<td>30</td>
<td>0.3</td>
</tr>
<tr>
<td>40</td>
<td>0.4</td>
</tr>
<tr>
<td>50</td>
<td>0.5</td>
</tr>
<tr>
<td>60</td>
<td>0.6</td>
</tr>
<tr>
<td>70</td>
<td>0.7</td>
</tr>
<tr>
<td>80</td>
<td>0.8</td>
</tr>
<tr>
<td>90</td>
<td>0.9</td>
</tr>
<tr>
<td>100</td>
<td>1.0</td>
</tr>
</tbody>
</table>
```

ssh
rsh

Benot Claudel, Guillaume Huard and Olivier Richard
How to further optimize the deployment?

Issues

- Cost of **local parallelization** still linear
- The initiating machine is a critical resource: maximal number of processes, number of opened fds, ...

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
while remaining hosts do
    choose i in hosts
    ssh $i taktuk_engine(part(hosts))
taktuk_exec uptime
while remaining hosts do
    choose i in hosts
    ssh $i taktuk_engine(part(hosts))
    taktuk_exec uptime
while remaining hosts do
  choose i in hosts
  ssh $i taktuk_engine(part(hosts))
taktuk_exec uptime
Work distribution

while remaining hosts do
  choose i in hosts
  ssh $i taktuk_engine(part(hosts))
taktuk_exec uptime

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

Obviously not, for small node counts.
A connector (ssh) can be abstracted by 2 parts
- Non overlapping part (protocol computation)
- Overlapping part (wait)

- Non overlapping
- Overlapping

Similar to the postal model
- Homogeneous case known in the literature: **ASAP** is an **optimal** schedule
- Polynomially computable by a greedy algorithm
Optimal deployment
Optimal deployment

Root

Time

Benot Claudel, Guillaume Huard and Olivier Richard

TakTuk, Adaptive Deployment of Remote Executions
Optimal deployment
Optimal deployment
Optimal deployment

Node A

Root

C
B
A

Time
Optimal deployment

Node A

lost time

Time
Optimal deployment
Optimal deployment

Node B

Node A

Root

C
B
A

Time
Optimal deployment

Execution time:
inverse of a generalized fibonacci sequence
For 1000 nodes: 0.54s without overhead of the engine
Outline

1 Motivations
   - Problem
   - Needs

2 Deployment
   - Parallelization
   - Distribution
   - Optimal

3 TakTuk
   - Engine
   - Performance
   - Perspectives
Dynamic environment

The performance of nodes and network vary

- Heterogeneous architectures in different clusters
- Load due to OS or hanged processes (zombies, infinite loop)
- External contention (network, centralized services)
- Cache effects, swap, other users, ...

The nodes cannot be considered as homogeneous

Optimal postal solution does not hold anymore
Dynamic deployment proposal

Combine dynamically local parallelization and distribution:

- Try to do things ASAP
- Nodes initiate a **batch of parallel connections**

![Diagram showing concurrent calls, pipeline size, and time]

Ideally, 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
TakTuk engine

Perl implementation of the dynamic deployment proposal
- Architecture independent (tested on x86, PPC, IA-64)
- I/O and commands status multiplexing to the root
- Configurable mechanics, output templates

Suitable to both administration and applications deployment
- Interactive (shell-like) mode
- Nodes logical numbering and multicast communication layer
- Distribution using adaptive work-stealing algorithm
- Insensitive to nodes failures and heterogeneity
TakTuk architecture

Remote TakTuk father

Commands and options parsing

Core events server

I/O management

Active messages Server

Local command

Remote TakTuk instance

Connection initializer

Scheduler

Tasks pool

Work received

Work sent

Connector

Remote TakTuk instance

Remote TakTuk instance

Remote TakTuk instance

Benot Claudel, Guillaume Huard and Olivier Richard

TakTuk, Adaptive Deployment of Remote Executions
Compared to flat deployment

pdsh (clone of dsh included in IBM cluster toolsuite):

- Local parallelization only
- Low overhead, insensitive to node failures

Using rsh
(typical use in homogeneous clusters)
Compared to flat deployment

*pdsh* (clone of *dsh* included in IBM cluster toolsuite):

- Local parallelization only
- Low overhead, insensitive to node failures

Using *ssh* (mandatory in *grids*)
Compared to distributed deployment

gexec (part of ganglia cluster toolsuite):
- Distribution using n-ary tree to contact gexec daemons
- Not adaptive, cannot handle connections failure or loss

Uses ssh authentication without data encryption
No installation required with TakTuk

TakTuk can propagate itself without installation on remote node:

- Establishes connection and remotely execute a Perl interpreter
- Fetch itself through the connection

Self-propagation induces low overhead
TakTuk is insensitive to failures

Nodes unresponsiveness or connections loss do not hinder TakTuk

- Incriminated node is ignored when deploying
- ssh timeouts can be overridden for more responsiveness

Gexec does not handle faults
## Features comparison

<table>
<thead>
<tr>
<th></th>
<th>TakTuk</th>
<th>pdsh/dsh</th>
<th>gexec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deployment capabilities</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No remote installation required</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>New connector plugin</td>
<td>Immediate</td>
<td>Simple</td>
<td>No</td>
</tr>
<tr>
<td>Can mix several connectors</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Insensitive to nodes failures</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Distributed deployment</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Compiled engine</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Application support</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nodes logical numbering</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Integrated communication layer</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Files transfer capabilities</td>
<td>Yes</td>
<td>pdcp</td>
<td>PCP</td>
</tr>
</tbody>
</table>
Projects using TakTuk

- KAAPI: Parallel and Distributed programming environment
  - karun uses TakTuk to run KAAPI applications
  - KAAPI/TakTuk won ERCIM Grid@Works Plugtest for 3 consecutive years
  - Deployment of real application on 4000+ processor cores taken from 2 distinct grids

- OAR: Grid’5000 Batch Scheduler OAR uses TakTuk as a monitoring tool (test for nodes connectivity)
Conclusion and Perspectives

TakTuk

- Performs scalable parallel remote executions
- Adapts to heterogeneity
- Portable and highly configurable

Upcoming works

- Reactive and accurate adaptation of parallelization window
- Rewrite of `mpirun` using TakTuk (for some MPI distribution)
- Efficient data broadcasting using TakTuk to initiate transfers
Get TakTuk

Several ways to get TakTuk

- visit http://taktuk.gforge.inria.fr
- type 'TakTuk' in google
- 'apt-get install taktuk' under Debian/Ubuntu GNU/Linux

Acknowledgements

- Cyrille Martin (seminal work on TakTuk)
- Lucas Nussbaum (Debian maintainer)

Thanks for your attention
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

```
ssh Host1 perl
```

Host1
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

Ack informs that the remote node is ready

Host1
**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

The TakTuk code is sent
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

The Perl interpreter
now executes TakTuk

Host1
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

Deployed TakTuk can also deploy
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 > 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 ]

Host 1 Host 2 Host m
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 ]

Benot Claudel, Guillaume Huard and Olivier Richard

TakTuk, Adaptive Deployment of Remote Executions
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`

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

- TakTuk command

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