WALT on Grid’5000 (with NBFS)

or how to pack all my activities into a single project ;)

Etienne Dublé (LIG / CNRS)
Workshop Axes, May 2021
Motivation

Sample use case

WalT & fast OS prototyping

OS deployment rework

Related news
Types of Computer Problems
By how much debugging them makes your brain stop working

None       Some       A Lot

Normal Problems

Networking Problems

Before noon, odd-numbered packets were laggy, but after noon, even-numbered ones are! It's the opposite of yesterday!
Are you sure you're okay?
I'm fine and I believe in ghosts now!
A word about Grid’5000

G5K: What is it?

- Infrastructure as a Service
- 8 sites in France
- Accessible to academic community
### A word about WALT

<table>
<thead>
<tr>
<th>WALT: What is it?</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Software to build your own versatile platform</td>
</tr>
<tr>
<td>• AFAIK deployed in 4 labs (France, Turkey) and at Schneider Electric</td>
</tr>
<tr>
<td>• Open source project</td>
</tr>
</tbody>
</table>
Why WALT on G5K?

- Why WALT on G5K?

1. "I want to run my WALT experiment uncoded on a larger testbed"
2. WALT could bring some useful features on top of G5K
   - fast OS prototyping
   - seamless multi-site experiments
3. It could be an entrypoint for new users who want to try WALT

Limits:
- Features are limited to what both platforms can do (simplified G5K resources selection, wired network only, no physical access to devices, etc.)
Why WALT on G5K?

- Why WALT on G5K?
  1. "I want to run my WALT experiment unmodified on a larger testbed"
Why WALT on G5K?

1. "I want to run my WALT experiment unmodified on a larger testbed"
2. WALT could bring some useful features on top of G5K
   - fast OS prototyping
   - seamless multi-site experiments
Why WALT on G5K?

1. "I want to run my WALT experiment unmodified on a larger testbed"
2. WALT could bring some useful features on top of G5K
   - fast OS prototyping
   - seamless multi-site experiments
3. It could be an entrypoint for new users who want to try WALT
Why WALT on G5K?

1. "I want to run my WALT experiment unmodified on a larger testbed"
2. WALT could bring some useful features on top of G5K
   - fast OS prototyping
   - seamless multi-site experiments
3. It could be an entrypoint for new users who want to try WALT

Limits:
- Features are limited to what both platforms can do (simplified G5K resources selection, wired network only, no physical access to devices, etc.)
Agenda

Motivation

Sample use case

WaIT & fast OS prototyping

OS deployment rework

Related news
Sample use case
Sample use case

$ ssh grenoble.g5k

WaiT client

Nodes

G5K Grenoble

Frontend

Nodes

G5K Lille

Frontend
Sample use case

$ ssh grenoble.g5k

WaIT client
Sample use case

$ ssh grenoble.g5k
  pip3 install walt-client[g5k]
Sample use case

$ ssh greno ble.g5k
> pip3 install walt-client[g5k]
Sample use case
Sample use case

> walt g5k deploy

WalT client

Nodes

G5K Grenoble

Frontend

Nodes

G5K Lille

Frontend
Sample use case

> walt g5k deploy
which resources?
Sample use case

> walt g5k deploy
which resources? [...]

Walt client

Nodes

G5K Grenoble
Frontend

Nodes

G5K Lille
Frontend
Sample use case

- Walt g5k deploy
  - which resources? [...]

- WaltT client

Diagram:
- WaltT Server
- Nodes
- G5K Grenoble
- Frontend
- G5K Lille
- Frontend
Sample use case

> walt g5k deploy
which resources? [...]

Walt client

Walt Server

Nodes

G5K Grenoble

Frontend

Nodes

G5K Lille

Frontend
Sample use case

> walt g5k deploy
which resources? [...]
Sample use case

> walt g5k deploy

which resources? [...]

WalT client

WalT Server

Nodes

G5K Grenoble

Frontend

G5K global VLAN

G5K Lille

Frontend

Nodes
Sample use case

```
> walt g5k deploy
done.
```

![Diagram showing network components and deployment process]
Sample use case

> `walt image shell ...`

WaltT client

Nodes

WaltT Server

Nodes

G5K Grenoble

Frontend

G5K global VLAN

G5K Lille

Frontend
Sample use case
Sample use case

> `walt image [clone|publish|...]`

**Remote access**

**Nodes**

**Walt Server**

**Walt Image pc-regular**

**Walt Image pc-realtime**

**Walt client**

**G5K Grenoble**

**G5K Lille**

**Frontend**

**G5K global VLAN**
Sample use case
Sample use case

```
walt node boot <nodes> <image>
walt node boot <nodes> <image>
```
Sample use case

```bash
> walt node boot <nodes> <image>
```

Diagram showing a network boot process involving Docker, WaitT Server, WaitT Image, WaitT client, Nodes, Remote access, and Network boot.
Motivation

Sample use case

WalT & fast OS prototyping

OS deployment rework

Related news
WalT allows fast OS prototyping:

- Modify an OS image very easily...
- WalT node boot <nodes> <image>
- Des are usually booted in less than 1 minute (on a local testbed with Raspberry Pi boards)
- Reproducibility is ensured at each node reboot
- Des are stateless
- Changes made with respect to image files are discarded on reboot
WalT allows fast OS prototyping:

- You can **modify an OS image** very easily...
  - `walt image shell <image>` (among other commands)
  - ... and save your changes in a few seconds
    (relies on docker image layered approach)
WalT allows fast OS prototyping:

- You can **modify an OS image** very easily...
  - `walt image shell <image>` (among other commands)
  - ... and save your changes in a few seconds
    (relies on docker image layered approach)

- You can **deploy an OS image** on nodes quickly:
  - `walt node boot <nodes> <image>`
  - Nodes are usually booted in less than 1 min
    (on a local testbed with Raspberry Pi boards)
WalT allows fast OS prototyping:

- You can **modify an OS image** very easily...
  - `walt image shell <image>` (among other commands)
  - ... and save your changes in a few seconds
    (relies on docker image layered approach)
- You can **deploy an OS image** on nodes quickly:
  - `walt node boot <nodes> <image>`
  - Nodes are usually booted in less than 1 min
    (on a local testbed with Raspberry Pi boards)
- **Reproducibility** is ensured at each node reboot
  - Nodes are stateless
  - Changes made wrt. image files are discarded on reboot
Agenda

Motivation

Sample use case

WalT & fast OS prototyping

OS deployment rework

Related news
How is OS deployment handled by WalT?
How is OS deployment handled by WalT?

- As a regular \textit{network boot} procedure
How is OS deployment handled by WalT?

- As a regular network boot procedure
WalT node deployment

How is OS deployment handled by WalT?

- As a regular network boot procedure

About network boot

- *OS image remains on walt server only*
  (it is never transferred as a whole to the node)
How is OS deployment handled by WaiT?

- As a regular network boot procedure

### About network boot

- *OS image remains on walt server only*  
  (it is never transferred as a whole to the node)
- Server exposes content of OS image as an **NFS network share**
How is OS deployment handled by WalT?

- As a regular **network boot** procedure

### About network boot

- *OS image remains on walt server only*  
  (it is never transferred as a whole to the node)
- Server exposes content of OS image as an **NFS network share**
- When booted, the whole OS of node is seated on this NFS share
WalT node deployment

How is OS deployment handled by WalT?

- As a regular **network boot** procedure
- Steps, for a light debian OS:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node: Hardware reboot</td>
<td>≈2min10</td>
</tr>
<tr>
<td>2</td>
<td>Node: Network bootloader</td>
<td>fast</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>≈2min30</td>
</tr>
</tbody>
</table>
WalT node deployment

How is OS deployment handled by WalT?

- As a regular **network boot** procedure
- Steps, for a light debian OS:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node: Hardware reboot</td>
<td>≈2min10</td>
<td>≈2min10</td>
</tr>
<tr>
<td>2</td>
<td>Node: Network bootloader</td>
<td>fast</td>
<td>≈4min</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈35s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>≈2min30</td>
<td>≈6min50</td>
</tr>
</tbody>
</table>

- Delay B: walt node and walt server are at a different G5K site.
How is OS deployment handled by WalT?

- As a regular network boot procedure
- Steps, for a light debian OS:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node: Hardware reboot</td>
<td>≈2min10</td>
<td>≈2min10</td>
</tr>
<tr>
<td>2</td>
<td>Node: Network bootloader</td>
<td>fast</td>
<td>≈4min</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈35s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>≈2min30</td>
<td>≈6min50</td>
</tr>
</tbody>
</table>

- Delay B: waltnode and walt server are at a different G5K site.
Walt node deployment – avoiding TFTP

- Goal: Improve the slow bootloader step in case B
• Goal: Improve the slow bootloader step in case B
• Diagnosis:
  • Kernel and initrd (≈30MB) downloaded from server site to node site
Goal: Improve the slow bootloader step in case B

Diagnosis:
- Kernel and initrd (≈30MB) downloaded from server site to node site
- iPXE bootloader uses TFTP protocol for this
• Goal: Improve the slow bootloader step in case B

• Diagnosis:
  • Kernel and initrd (≈30MB) downloaded from server site to node site
  • iPXE bootloader uses **TFTP protocol** for this
  • Transfers need ≈**30 000 round-trips** between sites to complete!
- Goal: Improve the slow bootloader step in case B
- Diagnosis:
  - Kernel and initrd ($\approx$30MB) downloaded from server site to node site
  - iPXE bootloader uses TFTP protocol for this
  - Transfers need $\approx$30,000 round-trips between sites to complete!
- Solution: let iPXE use HTTP instead of TFTP.
WalT node deployment – avoiding TFTP

- New numbers:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node: Hardware reboot</td>
<td>≈2min10</td>
<td>≈2min10</td>
</tr>
<tr>
<td>2</td>
<td>Node: Network bootloader</td>
<td>fast</td>
<td>≈35s</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈35s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td><strong>≈2min30</strong></td>
<td><strong>≈3min25</strong></td>
</tr>
</tbody>
</table>

Result: this delay was reduced from ≈4min to ≈35s.

Analysis: better, but such a transfer between sites should be <1s.

(Bootloader network driver is basic and sub optimal)
WalT node deployment – avoiding TFTP

- New numbers:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node: Hardware reboot</td>
<td>≈2min10</td>
<td>≈2min10</td>
</tr>
<tr>
<td>2</td>
<td>Node: Network bootloader</td>
<td>fast</td>
<td>≈35s</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈35s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td><strong>≈2min30</strong></td>
<td><strong>≈3min25</strong></td>
</tr>
</tbody>
</table>

- Result: *this delay was reduced from ≈4min to ≈35s.*
### WalT node deployment – avoiding TFTP

- **New numbers:**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node: Hardware reboot</td>
<td>≈2min10</td>
<td>≈2min10</td>
</tr>
<tr>
<td>2</td>
<td>Node: Network bootloader</td>
<td>fast</td>
<td>≈35s</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈35s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>≈2min30</td>
<td>≈3min25</td>
</tr>
</tbody>
</table>

- **Result:** *this delay was reduced from ≈4min to ≈35s.*

- **Analysis:** better, but such a transfer between sites should be <1s. (Bootloader network driver is basic and suboptimal)
WalT node deployment – avoiding TFTP

- New numbers:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node: Hardware reboot</td>
<td>≈2min10</td>
<td>≈2min10</td>
</tr>
<tr>
<td>2</td>
<td>Node: Network bootloader</td>
<td>fast</td>
<td>≈35s</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈35s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>≈2min30</td>
<td>≈3min25</td>
</tr>
</tbody>
</table>

- Result: **this delay was reduced from ≈4min to ≈35s.**

- Analysis: better, but such a transfer between sites should be <1s.
  (Bootloader network driver is basic and suboptimal)
Goal: Avoid hardware reboot

Idea:

Kexec technique can be used to directly boot a new kernel (bypassing hardware reboot)

Solution:

- Implement this kexec-boot in `/bin/walt-reboot`
- Mimic network bootloader behaviour: download and boot the kernel of the new image (not the current one!)
Goal: Avoid hardware reboot

Idea: **Kexec** technique can be used to directly boot a new kernel (bypassing hardware reboot)
WalT node deployment – avoiding hardware reboot

- **Goal:** Avoid hardware reboot
- **Idea:** Kexec technique can be used to directly boot a new kernel (bypassing hardware reboot)
- **Solution:**
  - Implement this kexec-boot in `/bin/walt-reboot`
    - Mimic network bootloader behaviour: download and boot the kernel of the new image (not the current one!)
WalT node deployment – avoiding hardware reboot

- **Goal:** Avoid hardware reboot
- **Idea:** Kexec technique can be used to directly boot a new kernel (bypassing hardware reboot)
- **Solution:**
  - Implement this kexec-boot in `/bin/walt-reboot`
  - Mimic network bootloader behaviour:
    - download and boot the kernel of the new image
    - *(not the current one!)*
We now have different steps:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node (walt-reboot): get kernel &amp; initrd</td>
<td>fast</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>2</td>
<td>Node (walt-reboot): kexec new kernel</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈35s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>≈20s</td>
<td>≈40s</td>
</tr>
</tbody>
</table>

Results:
- Hardware reboot delay completely eliminated.
- Kernel & initrd now transferred by walt-reboot (was iPXE).
- Acceptable delay for the whole procedure.
WalT node deployment – avoiding hardware reboot

- We now have different steps:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node (walt-reboot): get kernel &amp; initrd</td>
<td>fast</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>2</td>
<td>Node (walt-reboot): kexec new kernel</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈35s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>≈20s</td>
<td>≈40s</td>
</tr>
</tbody>
</table>

- Results:
  - hardware reboot delay completely eliminated.
WalT node deployment – avoiding hardware reboot

- We now have different steps:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node (walt-reboot): get kernel &amp; initrd</td>
<td>fast</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>2</td>
<td>Node (walt-reboot): kexec new kernel</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈35s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>≈20s</td>
<td>≈40s</td>
</tr>
</tbody>
</table>

- Results:
  - **hardware reboot delay completely eliminated.**
  - **kernel & initrd** now transfered by **walt-reboot** (was iPXE)
WalT node deployment – avoiding hardware reboot

- We now have different steps:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node (walt-reboot): get kernel &amp; initrd</td>
<td>fast</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>2</td>
<td>Node (walt-reboot): kexec new kernel</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>(\approx 15\text{s})</td>
<td>(\approx 35\text{s})</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>(\approx 20\text{s})</td>
<td>(\approx 40\text{s})</td>
</tr>
</tbody>
</table>

- Results:
  - hardware reboot delay completely eliminated.
  - kernel & initrd now transfered by \texttt{walt-reboot} (was iPXE)
  - Acceptable delay for the whole procedure.
WalT node deployment – avoiding hardware reboot

- We now have different steps:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node (wal-reboot): get kernel &amp; initrd</td>
<td>fast</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>2</td>
<td>Node (wal-reboot): kexec new kernel</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈35s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>≈20s</td>
<td>≈40s</td>
</tr>
</tbody>
</table>

- Results:
  - **hardware reboot delay completely eliminated.**
  - **kernel & initrd** now transferred by **wal-reboot** (was iPXE)
  - Acceptable delay for the whole procedure.
Goal: Reduce OS boot delay in case B

Diagnosis: NFS is slow when client-server latency is high.

Solution: Use NBFS instead.

About NBFS

An experimental network file system I am working on.

Specialized for network booting.

Uses speculation: good performance even when latency is high.

Improves responsiveness after bootup (e.g., reduced ssh login delay).

A research paper is being written with R. Lachaize, F. Rousseau, A. Duda.
Goal: Reduce OS boot delay in case B

Diagnosis: **NFS is slow when client-server latency is high.**
Goal: Reduce OS boot delay in case B

Diagnosis: **NFS is slow when client-server latency is high.**

Solution: use **NBFS** instead.
Goal: Reduce OS boot delay in case B

Diagnosis: **NFS is slow when client-server latency is high.**

Solution: use **NBFS** instead.

---

**About NBFS**

- An experimental *network filesystem* I am working on.
- Specialized for *network booting*.
- Uses *speculation*: good performance even when *latency* is high.
- Improves responsiveness after bootup too (e.g. reduced ssh login delay).
- A research paper is being written with R.Lachaize, F.Rousseau, A.Duda.
WalT node deployment – avoiding NFS

- New numbers with NBFS instead of NFS:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node (walt-reboot): get kernel &amp; initrd</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>2</td>
<td>Node (walt-reboot): kexec new kernel</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>≈15s</td>
<td>≈15s</td>
</tr>
<tr>
<td></td>
<td><strong>Total procedure</strong></td>
<td>≈20s</td>
<td>≈20s</td>
</tr>
</tbody>
</table>

Result: similar delay for remote and local bootup.
WalT node deployment – avoiding NFS

- New numbers with NBFS instead of NFS:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Delay A</th>
<th>Delay B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server: Route requests to new image</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>1</td>
<td>Node (walt-reboot): get kernel &amp; initrd</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>2</td>
<td>Node (walt-reboot): kexec new kernel</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>3</td>
<td>Node: Wait for OS boot</td>
<td>$\approx 15s$</td>
<td>$\approx 15s$</td>
</tr>
</tbody>
</table>

| Total procedure | $\approx 20s$ | $\approx 20s$ |

- Result: similar delay for remote and local bootup.
Agenda

Motivation

Sample use case

WaIT & fast OS prototyping

OS deployment rework

Related news
Related news

- **NBFS** is still **experimental**
- **Walt-on-G5K** feature planning:
  - available with **WALT version 8** (end of June)
  - **NBFS** will **not** be included
  - we are discussing with G5K team for improvements / documentation etc.
- **G5K** team is working on improving **kadeploy** with **kexec** too (near future)
More info:

WalT-on-G5K demo: https://vu.fr/walt-on-g5k
WalT website: https://vu.fr/walt
Questions, WalT training requests: etienne.duble@imag.fr