Virtualization and High Availability

Leonello Servoli[1]

[1]INFN Sezione di Perugia

Workshop 2007 su Calcolo e Reti INFN

Outline

Outline

LHC, Grid and High Availability

When the new collider LHC will be switched on, large amounts of data will be produced every year and should be analyzed by thousands of physicists distributed over hundreds of Institutions.
Hence the HEP community adopted the concept of *computing GRID* as cornestone to build the Computing model for LHC experiments.

- Some nodes and services, either GRID middleware or experiment specific, are going to be critical in such complex infrastructure and it is necessary to have them working 24x7.
- Finding a reliable solution to make Highly Available such services is an important (and not easy) task.

What is High Availability?

A service that is supposed to work "always", means a small fraction of unavailability (99% $\sim$ 3,6 days of downtime/year)

- Lots of problems can cause a failure:
  - HW (hard disks, RAM, etc.),
  - SW,
  - operator errors...
- The complexity of these problems varies a lot, and the restore time goes between minutes to days.
  - Time to solve problems might rely on component availability, distributors and other issues.
  - People is not always available to solve problems (lunches, nights, weekends, holidays).

Where we are?

24x7 services expected at least at all Tier1s, and this should be a goal also for Tier2s.

- We won't be able to even approach that goal, unless we build redundancy at several layers:
  - system-level
  - application-level
  - geographically
- Unexpected outages are not the only concern: big centers need to routinely perform maintenance on several subsystems isolate SPoF
- INFN Tier-1: currently doing not too bad, but there's room for improvements.

  Feb 2007: 93% reliability/availability, 2nd place after Taiwan (i.e. before CERN, RAL, FZK, IN2P3, etc)

Work Area

- System-level HA. typically using clustering solutions like Linux-HA, or RH Cluster Suite.
  Some solutions may involve the adoption of virtualization technologies.
- Application-level HA. This may be a specialization of the above, and/or involve ad-hoc (per-application) configurations.
  Software should be designed with HA in mind (in the perfect world...).
- Geographic failover and/or load-balancing. May again overlap with some of the above.

INFN and HA

Several INFN sites have already deployed HA solutions, others are looking with interest to them. It's an area where sometimes we need to experiment a lot.
But it's not all geeky work: in the end *we need clear and reliable solutions.*
Note that there are several areas where we are still sorely missing HA configurations (example: a transparently redundant and load-balanced WMS)
The INFN HA group tries to collate experiences, with the goal of producing results, guidelines and HowTo's. Please join us if you are interested,
e-mailing ***ha@cnaf.infn.it***

Outline

Host redundancy

The classical approach: Host redundancy.

- This model is based on the replication of machines.
- Uses a technology like HeartBeat, Red Hat Cluster Suite, etc.
- Downtime is minimal as there is a second machine already running and ready to substitute the first.

But...

- Increments the costs as the number of machines gets duplicated (not only monetary costs, but also human resources to manage HW and SW).
- The resources are not fully exploited, i.e. the clones are unused until main machines stop working.

Outline

Brief explanation

Definition

Platform virtualization is the use of a specific software[a] to create different execution environments on a given machine (*host*), making it possible to execute another operating system (*guest*) on top of them, on an isolated, idependent and secure way.

[a]Called Virtual Machine Monitor (VMM) or Hypervisor

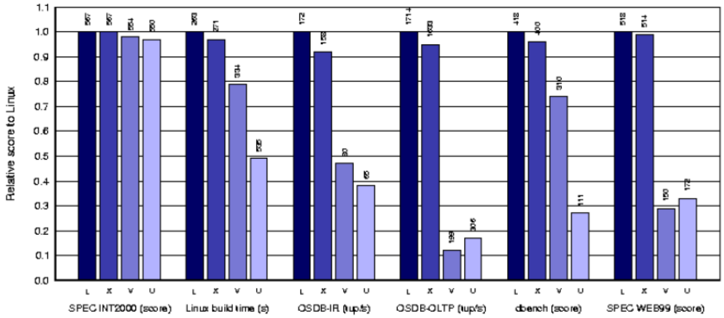
- Different types of virtualization and different products (Xen, Vmware, Qemu...).
- Virtualization, generally, introduces a non neglibile performance loss.
- Xen[1] implements paravirtualization, which produces a lower overhead.

[1]`http://xen.sourceforge.net`

What Virtual Machines are good for?

- More and more often regarded as a panacea for all evils. Realistically, they allow to: (cheaply) isolate services. This is good from many standpoints, last but not least security.
- Detach hardware and software configurations. More on this later.
- Schedule maintenance of key machines without service interruption. (e.g. through VM live migration)
- Easily set-up test environments.
- Spread HA solutions (multiple [virtual] hosts providing a reliable service)

Why Xen?

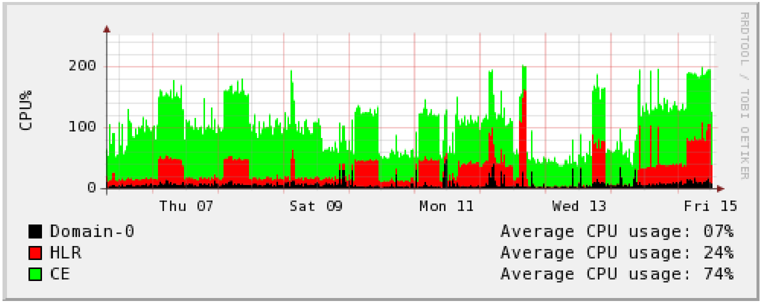


Native Linux (L), Xen/Linux (X), VMware Workstation 3.2 (V), User Mode Linux (U).

- Software OpenSource.
- Paravirtualization (multiple independent kernels on the same physical machine).
- Some experience already present.

Xen used in production services

- INFN-Torino has been running Grid services (CE + HLR) based on Xen for almost one year without any relevant problem and with low CPU overhead.

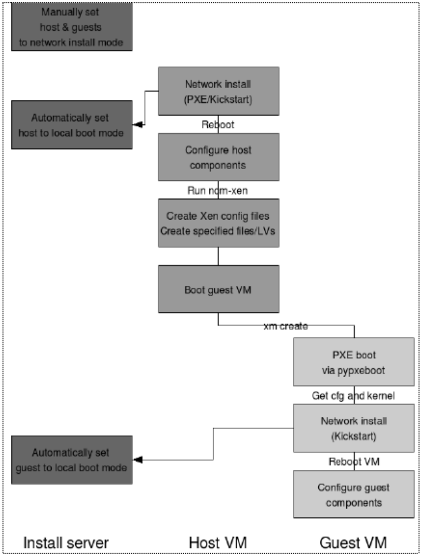


- INFN-Perugia has been running Phedex (a specific CMS server for data transfer) from the beginning without problems.

HA group current work on Xen

- Documentation!
- Testing VT, i.e. hardware-supported virtualization. This brings two main advantages:
  - It further reduces the already low performance penalty introduced by VM
  - It allows one to run unmodified kernels for a VM. This is a sensible improvement.
- Tier-1 specific configurations:
  - AFS works.
  - GPFS still requires some effort and testing. It has to work in both host and guest configurations.
  - Easy deployment of XEN-based VM, based on the Tier-1 production installation system (Quattor).

Xen and Quattor

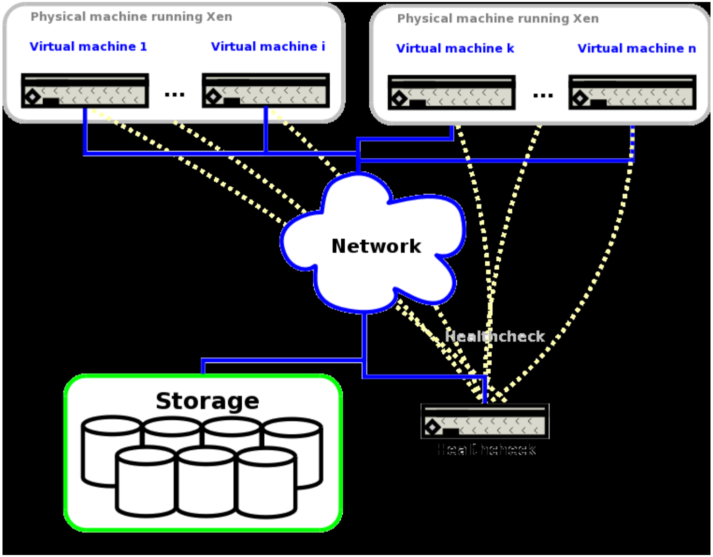

Manually set host & guests to network install mode
Network install (PXE/Kickstart)
Automatically set host to local boot mode
Reboot
Configure host components
Run ncm-xen
Create Xen config files Create specified files/LVs
Boot guest VM
xm create
PXE boot via pypxeboot
Get cfg and kernel
Network install (Kickstart)
Automatically set guest to local boot mode
Reboot VM
Configure guest components
Install server
Host VM
Guest VM

- Quattor is used at Tier1 to store VM configurations centrally to allow from-scratch installations of host and guests
- The creation of replicas is greatly facilitated
- There's now a Quattor component allowing all of this (developed by Trinity College, Dublin).

Outline

The idea

Our first approach



Components

- Physical machines
- Virtual machines
- Storage
- Healthcheck

The idea

Explanation

- A VMM[2] like Xen is installed on every physical (real) machine (PM).
- All the machines that provide a service are going to be virtualized.
- The filesystem of the virtual machines (VM) is going to be loaded from a network-based storage.
  - The virtual machines (VM) can run on any physical machine (PM).
  - If a machine that hosts a VM goes down, we can boot the VM (exactly the same) on another one.

The idea is to make this architecture to work in an autonomous and operator-independent way.

[2]Virtual Machine Monitor

The idea

Consequences

With our solution:

- The software is separated from the hardware where it is going to run.
  - As a consequence of this, a VM can run on every PM.
  - We can use any operating system on the physical machines.
  - We can run old sofware on a new machine.
  - We can put PMs in a private network segment to have the maximum security.
- It is possible to create easily machines with testing purposes.
- The resources are fully exploited until something fails (i.e. few or no spare machines).

Outline

Software

The software related to the virtualization has been extensively tested

- Tested Xen (first version 2, now version 3).
  - Hardware: AMD and Intel.
  - Several Operating Systems, both hosts (Slackware, Gentoo, Debian, Scientific Linux, etc) and guests (Scientific Linux 3 and 4.)
- Stress tests to access filesystem under the VM.
- Installed some Grid components on VM (WN, CE, SE...).
- Installed non-Grid components on VM (mailserver, Phedex, etc.).

Storage

In our approach it is a key element and should be very performant from the I/O point of view.
Different solutions tested:

- Fibre Channel (the best solution so far).
- iSCSI: Good performance/price ratio.
- GNBD: Not so good as iSCSI (expecially on server CPU usage) but working.
- Ata over Ethernet (AoE) (now under testing).
  - New product.
  - Presents some problems if used directly with Xen (might use LVM).

Healthcheck

A component that:

- Controls the nodes and detects a failure (HW and/or SW) and
- acts automatically to try to solve the problems.

Possibilities:

Custom healthcheck

- Really complex to implement.
- Useful to focus the problems that might be present.
- Maybe not write the whole healthcheck, but some parts of it.

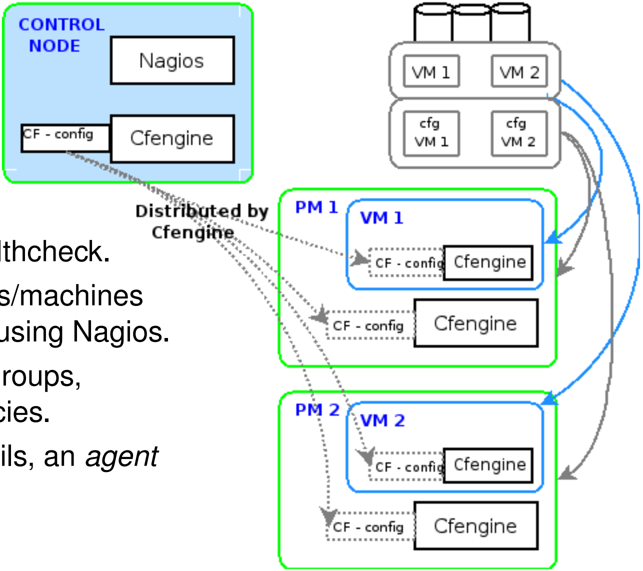
Existing tools that might be used

- Hearbeat
- OpenQRM
- Nagios
- Cfengine
- ...

Outline

Components

The Control Node.

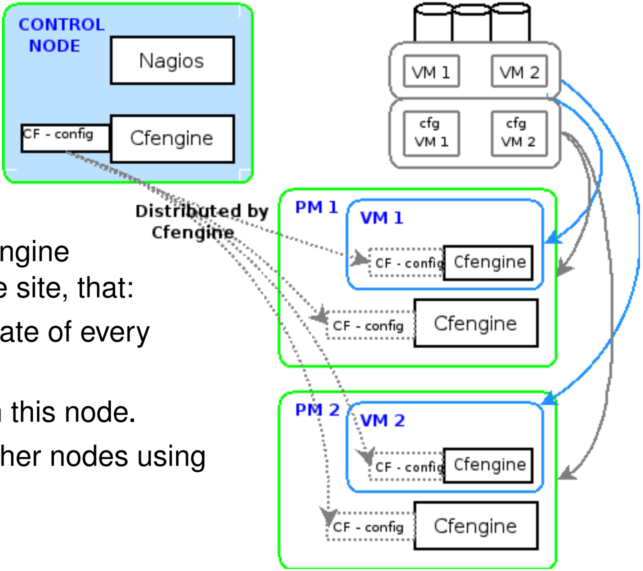


This node does the healthcheck.

- Checks the services/machines (both VM and PM) using Nagios.
- Different machine groups, different check policies.
- When something fails, an *agent* decides what to do.

Components

The Control Node.

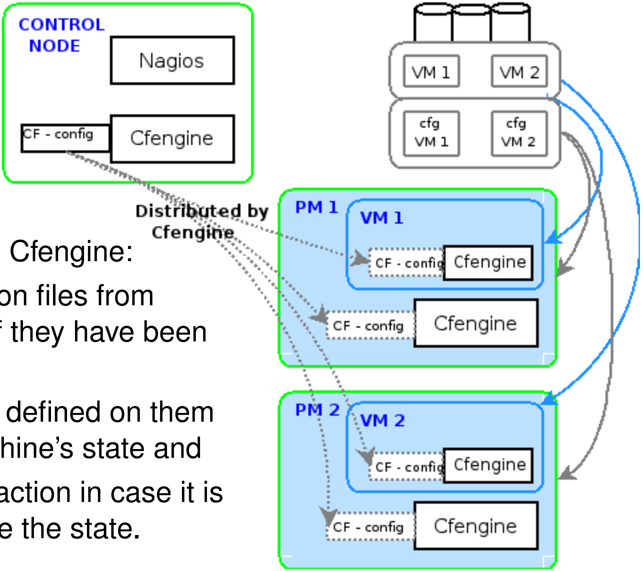


This node hosts the Cfengine configuration files for the site, that:

- define the "ideal" state of every machine.
- are modified only in this node.
- are distributed to other nodes using Cfengine's tools.

Components

PM 1, PM 2.
VM 1, VM 2.

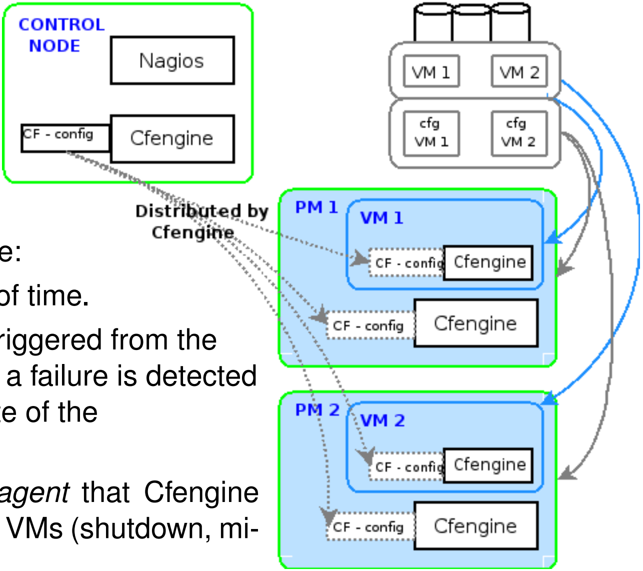


Each node has installed Cfengine:

- gets the configuration files from the "control node" if they have been modified,
- compares the state defined on them with the actual machine's state and
- performs a certain action in case it is necessary to update the state.

Components

PM 1, PM 2.
VM 1, VM 2.



These actions take place:

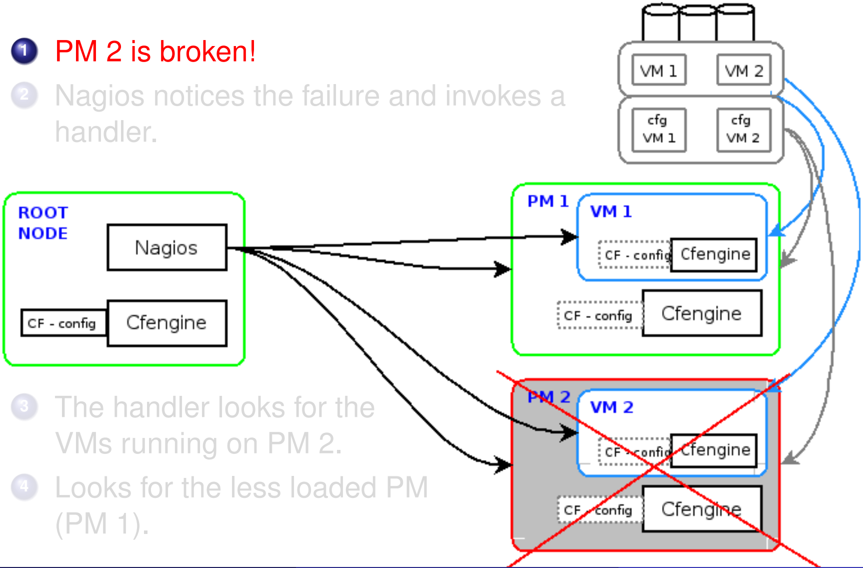
- At specific periods of time.
- When Cfengine is triggered from the control node (when a failure is detected or there is an update of the configuration).

Also here there is an *agent* that Cfengine launches to manage the VMs (shutdown, migrate, etc).

An example

The Failure

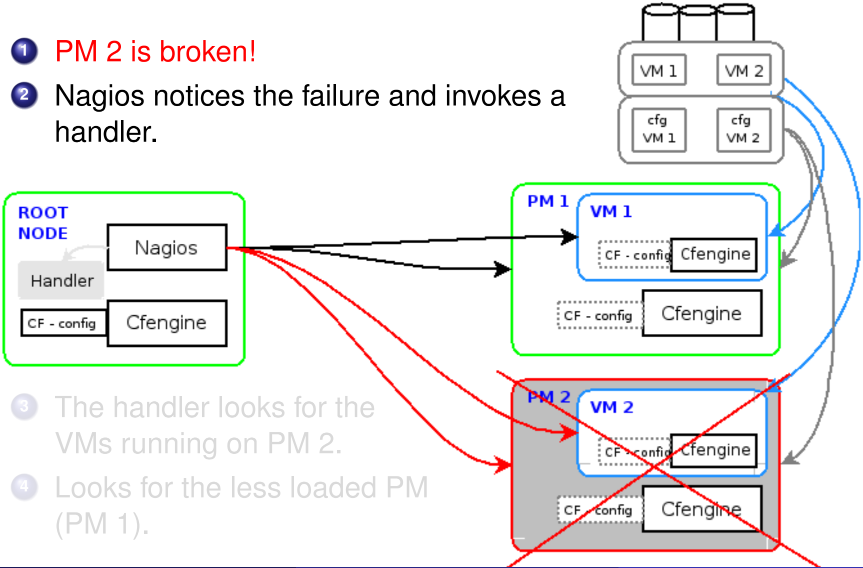
1. PM 2 is broken!
2. Nagios notices the failure and invokes a handler.
3. The handler looks for the VMs running on PM 2.
4. Looks for the less loaded PM (PM 1).



An example

The Failure

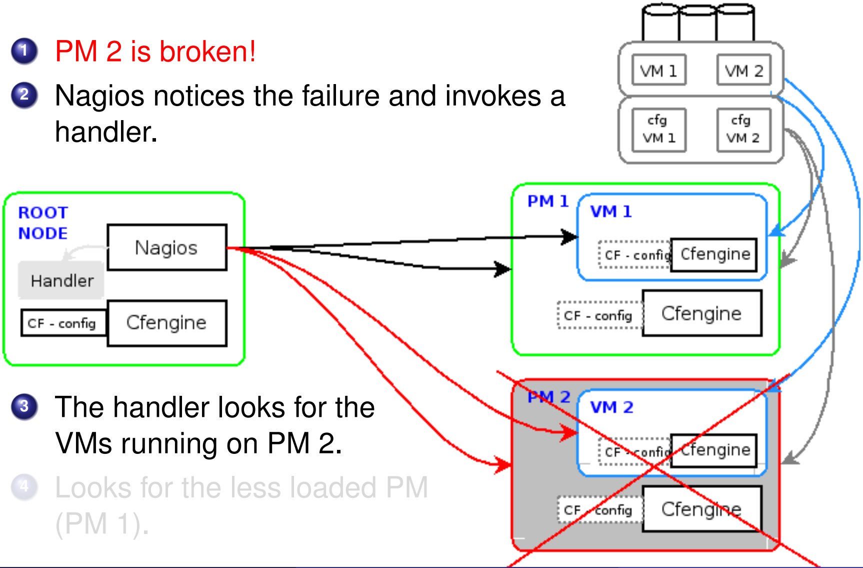
1. PM 2 is broken!
2. Nagios notices the failure and invokes a handler.
3. The handler looks for the VMs running on PM 2.
4. Looks for the less loaded PM (PM 1).



An example

The Failure

1. PM 2 is broken!
2. Nagios notices the failure and invokes a handler.

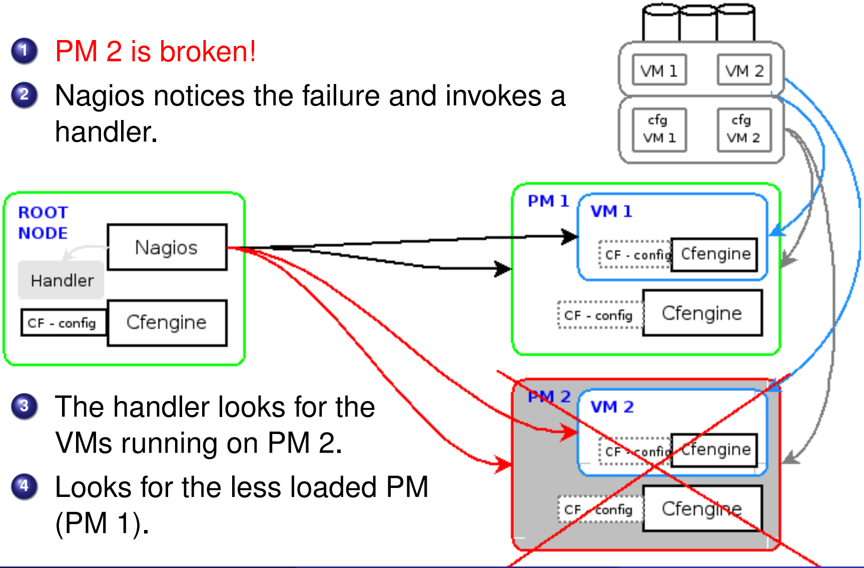


3. The handler looks for the VMs running on PM 2.
4. Looks for the less loaded PM (PM 1).

An example

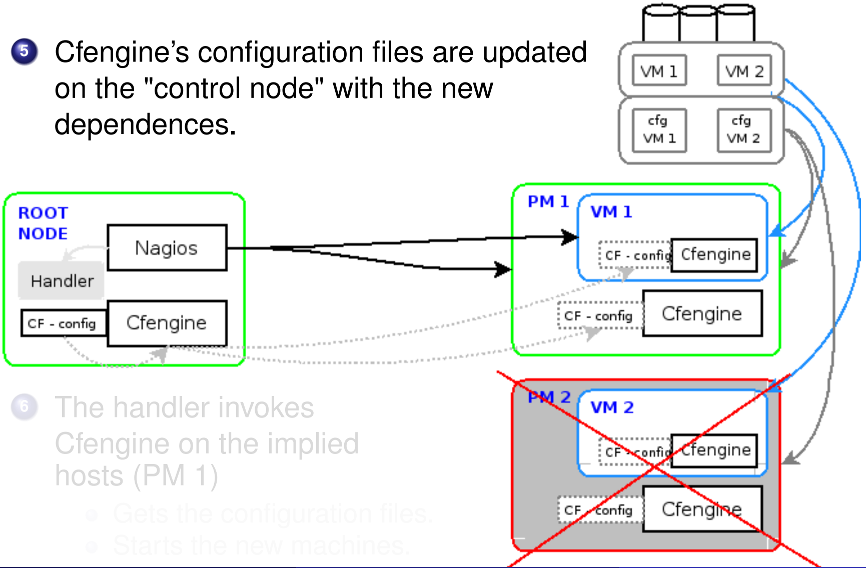
The Failure

1. PM 2 is broken!
2. Nagios notices the failure and invokes a handler.
3. The handler looks for the VMs running on PM 2.
4. Looks for the less loaded PM (PM 1).



An example
The Recover

5. Cfengine's configuration files are updated on the "control node" with the new dependences.

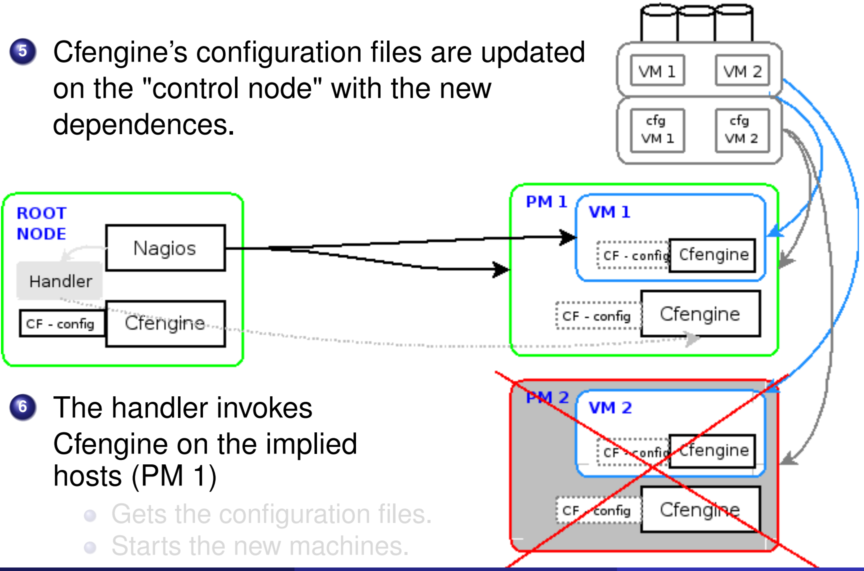


6. The handler invokes Cfengine on the implied hosts (PM 1)
   - Gets the configuration files.
   - Starts the new machines.

An example

The Recover

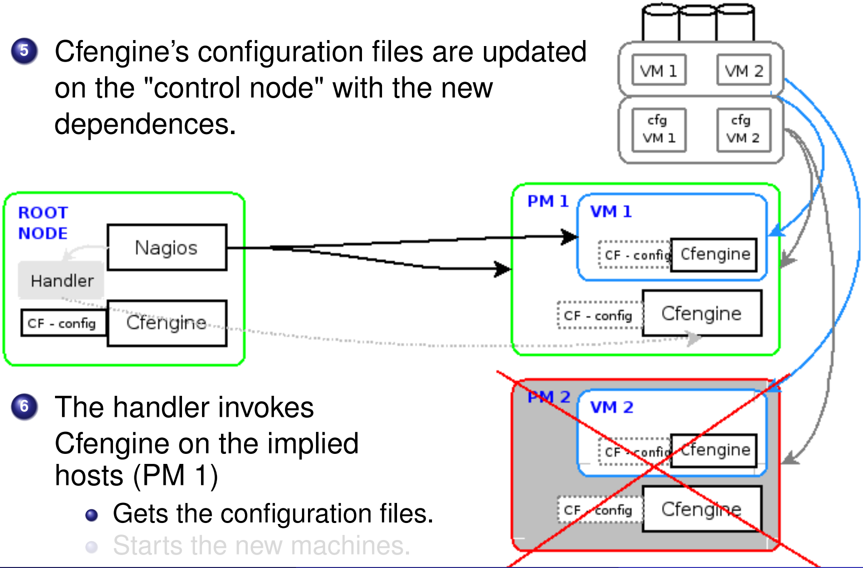
5. Cfengine's configuration files are updated on the "control node" with the new dependences.



6. The handler invokes Cfengine on the implied hosts (PM 1)
   - Gets the configuration files.
   - Starts the new machines.

An example
The Recover

5. Cfengine's configuration files are updated on the "control node" with the new dependences.

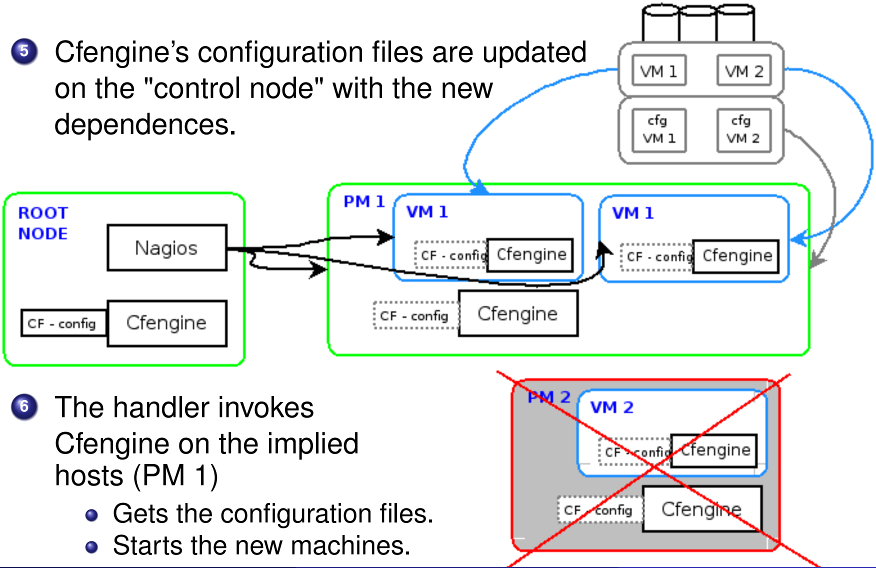


6. The handler invokes Cfengine on the implied hosts (PM 1)
   - Gets the configuration files.
   - Starts the new machines.

An example

The Recover

5. Cfengine's configuration files are updated on the "control node" with the new dependences.



6. The handler invokes Cfengine on the implied hosts (PM 1)
   - Gets the configuration files.
   - Starts the new machines.

Outline

Tests

We have created a small virtualized farm to test the prototype:

- Computing Element, Storage Element, and some Worker Nodes.

We have functionally tested the prototype:

- Stress of the prototype
  - Adding new machines.
  - Migration of the machines.
  - Forced failures.
- Tested the performace with job submission.
  - $\sim$ 200 jobs submitted in 6 hours.
  - Duration between 1 and 20 minutes.
  - Forced shutdown of the Computing Element every $\sim$ 10 minutes.
  - No jobs were undelivered!
- Now testing different combinations for the filesystems (part local and part remote, etc.).

Tests

We have created a small virtualized farm to test the prototype:

- Computing Element, Storage Element, and some Worker Nodes.

We have functionally tested the prototype:

- Stress of the prototype
  - Adding new machines.
  - Migration of the machines.
  - Forced failures.
- Tested the performace with job submission.
  - $\sim$ 200 jobs submitted in 6 hours.
  - Duration between 1 and 20 minutes.
  - Forced shutdown of the Computing Element every $\sim$ 10 minutes.
  - No jobs were undelivered!
- Now testing different combinations for the filesystems (part local and part remote, etc.).

Tests

We have created a small virtualized farm to test the prototype:

- Computing Element, Storage Element, and some Worker Nodes.

We have functionally tested the prototype:

- Stress of the prototype
  - Adding new machines.
  - Migration of the machines.
  - Forced failures.
- Tested the performace with job submission.
  - $\sim$ 200 jobs submitted in 6 hours.
  - Duration between 1 and 20 minutes.
  - Forced shutdown of the Computing Element every $\sim$ 10 minutes.
  - No jobs were undelivered!
- Now testing different combinations for the filesystems (part local and part remote, etc.).

Tests

We have created a small virtualized farm to test the prototype:

- Computing Element, Storage Element, and some Worker Nodes.

We have functionally tested the prototype:

- Stress of the prototype
  - Adding new machines.
  - Migration of the machines.
  - Forced failures.
- Tested the performace with job submission.
  - $\sim$ 200 jobs submitted in 6 hours.
  - Duration between 1 and 20 minutes.
  - Forced shutdown of the Computing Element every $\sim$ 10 minutes.
  - No jobs were undelivered!
- Now testing different combinations for the filesystems (part local and part remote, etc.).

Some results

Nagios introduces a delay before deciding that a machine/service is down. We have tested the prototype with three different times:

- Non-critical: 1 min.
    - Best case (1 VM per PM): 1 minute, 30 seconds.
    - Worst case (5 VM per PM): 3 minutes, 30 seconds.
- Not-so-critical: 40 sec.
    - Best case (1 VM per PM): 1 minute, 10 seconds.
    - Worst case (5 VM per PM): 3 minutes.
- Critical: 20 sec.
    - Best case (1 VM per PM): 50 seconds.
    - Worst case (5 VM per PM): 2 minutes, 30 seconds.

Some results

Nagios introduces a delay before deciding that a machine/service is down. We have tested the prototype with three different times:

- Non-critical: 1 min.
  - Best case (1 VM per PM): 1 minute, 30 seconds.
  - Worst case (5 VM per PM): 3 minutes, 30 seconds.
- Not-so-critical: 40 sec.
  - Best case (1 VM per PM): 1 minute, 10 seconds.
  - Worst case (5 VM per PM): 3 minutes.
- Critical: 20 sec.
  - Best case (1 VM per PM): 50 seconds.
  - Worst case (5 VM per PM): 2 minutes, 30 seconds.

Some results

Nagios introduces a delay before deciding that a machine/service is down. We have tested the prototype with three different times:

- Non-critical: 1 min.
  - Best case (1 VM per PM): 1 minute, 30 seconds.
  - Worst case (5 VM per PM): 3 minutes, 30 seconds.
- Not-so-critical: 40 sec.
  - Best case (1 VM per PM): 1 minute, 10 seconds.
  - Worst case (5 VM per PM): 3 minutes.
- Critical: 20 sec.
  - Best case (1 VM per PM): 50 seconds.
  - Worst case (5 VM per PM): 2 minutes, 30 seconds.

Dynamic environments

- Tier-1: with O(20) VOs to support, it is very difficult to provide a computing environment suited to everybody.
    - I require gcc X.Y! No, I need gcc W.Z!
    - Please upgrade to SLC 15! No, I need Fedora Core 3! (admittedly a contrived example)
    - I run faster in 64-bit mode! No, my application won't run in 64-bit mode!
    - ... and so on.
- Our ideal scenario: clear separation of hardware and software configurations.
    - A VM manager (e.g. XEN) is running on all farm computers
    - Worker Node VM images are produced in advance, suited to experiment requirements
    - VM images are deployed on-demand, based e.g. on the submitting VO; jobs run on the appropriate VM. This requires strict integration with the batch system.

Dynamic environments

- It seems feasible, but it is not that easy. We believe this would substantially change (for the better) how a big farm is managed and provisioned.
- Currently started to interact with CERN, NIKHEF and Platform on these subjects.
  (There are possibly some hooks in Platform's new Virtual Machine Orchestrator (VMO), but this needs further investigation)

Future plans

- Packaging of the solution for an automatic deployment (apt...) and write the documentation
- Install other GRID services (other SE, WMS) and other non-GRID experiment related services on VM.
- Continue with the tests in course.
- Apply these ideas to "Dynamic Enviroments" and on-demand deployment.
- Working on persistence of data in the control node agent.
- Working on direct communication between the control node agent and the node's one.