Universidad Zaragoza

Proyecto Fin de Carrera
Ingeniería en Informática

Diseño e implementación de un sistema de ejecución de trabajos distribuidos

David Ceresuela Palomera

Director: Javier Celaya

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Curso 2011/2012
Junio 2012

Resumen

A la hora de ejecutar trabajos en un entorno distribuido la aproximación clásica ha sido bien el uso de un *cluster* de ordenadores o bien el uso de la computación en malla o *grid*. Con la proliferación de entornos *cloud* durante estos últimos años y su facilidad de uso, parece que una nueva opción se abre para la ejecución de este tipo de trabajos.

La computación en *cloud* proporciona capacidad de cálculo, aplicaciones *software* y acceso, gestión y almacenamiento de datos sin requerir del usuario un conocimiento de los detalles internos de la infraestructura del *cloud*. Asimismo, frente a las dos aproximaciones clásicas de ejecución de trabajos, la computación en *cloud* ofrece la capacidad de añadir nodos para la ejecución de trabajos de forma más dinámica. Esta capacidad de aprovisionamiento de nodos de ejecución de manera tan dinámica supone todo un reto a la hora de administrar el *cloud*. Además de la administración, la puesta en marcha de una infraestructura de tipo *cloud* es también un proceso laborioso.

En vista de lo cual, en este proyecto se ha diseñado una solución capaz de automatizar tanto la puesta en marcha como la administración y el mantenimiento de un sistema de ejecución de trabajos distribuidos. Para ello se han estudiado entornos clásicos de ejecución de trabajos como Condor y Torque, entornos de ejecución de trabajos en *cloud* como AppScale y herramientas clásicas de administración de sistemas como Puppet y CFEngine. El objetivo principal de estas herramientas de adminisitración de sistemas es la gestión del nodo. En este proyecto se ha extendido la funcionalidad de una de estas herramientas – Puppet – añadiéndole la capacidad de gestión del *cloud*. Para verificar la solución diseñada se ha hecho uso del laboratorio 1.03b del Departamento de Informática e Ingeniería de Sistemas. Dicho laboratorio ofrece un entorno el el que simular, a pequeña escala, funcionalidades similares a las que obtendríamos en un *cloud* a gran escala.

Como resultado de este proyecto se presenta una extensión a la herramienta de configuración Puppet como solución para simplificar de manera notable la puesta en marcha de un sistema de ejecución de trabajos distribuidos así como la administración del mismo.

Índice general

Capítulo 1
Introducción

[Revisar]

La computación en la nube es un nuevo paradigma que pretende transformar la computación en un servicio. Durante estos últimos años la computación en la nube ha ido ganando importancia de manera progresiva ya que la posibilidad de usar la computación como un servicio permite a los usuarios de una aplicación acceder a ésta a través de un navegador web, una aplicación móvil o un cliente de escritorio mientras que la lógica de la aplicación y los datos se encuentran en servidores situados en una localización remota. Esta facilidad de acceso a la aplicación sin necesitar de un profundo conocimento de la infraestructura es la que, por ejemplo, brinda a las empresas la posibilidad de ofrecer servicios web sin tener que hacer una gran inversión inicial en infraestructura propia. Las aplicaciones alojadas en la nube tratan de proporcionar al usuario el mismo servicio y rendimiento que las aplicaciones instaladas localmente en su ordenador.

A lo largo de este proyecto se han usado tres ejemplos de infraestructuras distribuidas. La primera de ellas es la infraestructura de ejecución de trabajos. Este tipo de infraestructuras está especializada en la ejecución de grandes cargas de trabajo paralelizable e intensivo en computación. Son por lo tanto idóneas para ser usadas en la computación de altas prestaciones. Dentro de esta infraestructura distribuida los ejemplos más claros que podemos encontrar son Condor y Torque.

La segunda infraestructura es la de servicios web en tres capas. Este tipo de infraestructura tiene tres niveles claramente diferenciados: balanceo o distribución de carga, servidor web y base de datos. El balanceador de carga es el encargado de distribuir las peticiones web a los servidores web que se encuentran en el segundo nivel de la infraestructura. Éstos procesarán las peticiones web y para responder a los clientes puede que tengan que consultar o modificar ciertos datos. Los datos de la aplicación se encuentran en el tercer nivel de la estructura, y por consiguiente, cada vez que uno de los elementos del segundo nivel necesite leer información de la base de datos o modificarla, accederá a este nivel. En esta infraestructura no hay un ejemplo de uso que destaque sobre los demás, pero es tan común que cualquier página web profesional de hoy en día se sustenta en una infraestructura similar a ésta.

La tercera y última es AppScale, una implementación *open source* del App Engine de Google. App Engine permite alojar aplicaciones web en la infraestructura que Google posee. Además de presentar esta funcionalidad AppScale también ofrece las APIs de EC2, MapReduce y Neptune. La API de EC2 añade a las aplicaciones la capacidad de interactuar con máquinas alojadas en Amazon EC2. La API de MapReduce permite escribir aplicaciones que hagan uso del *framework* (o paradigma?) MapReduce. La última API, Neptune, añade a App Engine la capacidad de usar los nodos de la infraestructura para ejecutar trabajos. Los trabajos más representativos que

puede ejecutar son: de entrada, de salida y MPI. El trabajo de entrada sirve para subir ficheros (generalmente el código que se ejecutará) a la infraestructura, el de salida para traer ficheros (generalmente los resultados obtenidos después de la ejecución) y el de MPI para ejecutar un trabajo MPI.

La infraestructura necesaria para dar soporte a todas estas APIs ya no es tan sencilla como en los dos casos anteriores. De hecho, las anteriores infraestructuras estarían contenidas en ésta. Hay dos maneras de definir la infraestructura de AppScale. En la primera de ellas se define un despliegue por defecto, en el que un nodo toma el rol de *controller* y el resto de nodos toman el rol de *servers*. El nodo *controller* es el que carga con la responsabilidad de la coordinación y los nodos *servers* son los que llevan cabo la mayor parte del trabajo. La segunda manera de definir la infraestructura es hacerlo a través de un despliegue personalizado. En este despliegue podemos definir con más precisión los roles que queremos que desempeñe un nodo. Entre todos los posibles roles que AppScale ofrece, los más interesantes desde nuestro punto de vista son: *master*, *appengine*, *database*, *login* y *open*.

A lo largo de los últimos años las herramientas de administración de sistemas (o herramientas de gestión de configuración) también han experimentado un considerable avance: con entornos cada vez más heterogéneos y complejos la administración de sistemas complejos de manera manual ya no es una opción. Entre todo el conjunto de herramientas destacan de manera especial Puppet y CFEngine. Puppet es una herramienta de gestión de configuración basada en un lenguaje declarativo: el usuario especifica qué estado debe alcanzarse y Puppet se encarga de hacerlo. CFEngine, también con un lenguaje declarativo, permite al usuario un control más detallado de cómo se hacen las cosas, mejorando el rendimiento a costa de perder abstracciones de más alto nivel.

Sin embargo, estas herramientas de gestión de la configuración carecen de la funcionalidad requerida para administrar infraestructuras distribuidas. Son capaces de asegurar que cada uno de los nodos se comporta de acuerdo a la configuración que le ha sido asignada pero no son capaces de administrar una infraestructura distribuida como una entidad propia. Si tomamos la administración de un *cloud* como la administración de las máquinas virtuales que forman los nodos del mismo nos damos cuenta de que la administración es puramente *software*. Únicamente tenemos que asegurarnos de que para cada nodo de la infraestructura distribuida hay una máquina virtual que está cumpliendo con su función.

1.1 Contexto del proyecto

Para la realización de este proyecto de fin de carrera se ha hecho uso del laboratorio 1.03b de investigación que el Departamento de Informática e Ingeniería de Sistemas posee en la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza. Los ordenadores que forman este laboratorio poseen procesadores con soporte de virtualización, lo que permite la creación de diversas máquinas virtuales. La creación de los distintos tipos de cloud que representan cada una de las infraestructuras distribuidas se ha llevado a cabo a través de máquinas virtuales alojadas en distintos ordenadores del laboratorio.

En este laboratorio se ha comprobado la validez de la extensión introducida en la herramienta de gestión de configuraciones Puppet para administración de infraestructuras distribuidas que se ha desarrollado a lo largo de este proyecto de fin de carrera.

1.2 Objetivos

El objetivo de este proyecto es proporcionar una herramienta que facilite la puesta en marcha de infraestructuras distribuidas y su posterior mantenimiento. Las tareas principales en las que se puede dividir este proyecto son:

1. Análisis de las herramientas de administración de virtualización *hardware*.
2. Estudio de algunas de las infraestructuras distribuidas existentes profundizando en la parte relativa a la ejecución de trabajos distribuidos.
3. Investigación de las herramientas de gestión de configuración existentes más relevantes y elección de aquella que mayor facilidad de integración y uso proporcione.
4. Extensión de la herramienta de gestión de configuración para que soporte la puesta en marcha y el mantenimiento de un sistema de ejecución de trabajos distribuidos.

1.3 Tecnología utilizada

Para la elaboración de este proyecto se ha hecho uso de las siguientes tecnologías:

- KVM, QEMU, libvirt y virsh para el soporte y la gestión de las máquinas virtuales.
- Ruby como lenguaje de programación para la extensión de Puppet.
- Shell como lenguaje de programación de los *scripts* de configuración de las máquinas virtuales.
- Sistema operativo Debian para las máquinas del laboratorio y Ubuntu para las máquinas virtuales.
- LaTeX [6] para la redacción de esta memoria.

1.4 Organización de la memoria

El resto de este documento queda organizado de la siguiente manera:

Capítulo 2 Metodología seguida durante la realización del proyecto.

Capítulo 3 Extensión de Puppet para gestión de infraestructuras distribuidas.

Capítulo 4 Uso de la extensión de Puppet para gestión de infraestructura AppScale.

Capítulo 5 Uso de la extensión de Puppet para gestión de infraestructura web de tres niveles.

Capítulo 6 Uso de la extensión de Puppet para gestión de infraestructura de trabajos distribuidos.

Capítulo 7 Conclusiones.

1.5 Agradecimientos

Agradecimientos

Capítulo 2
Metodología

[Revisar]

En este capítulo se aborda la metodología seguida a la hora de realizar el proyecto.

2.1 Elección de la herramienta de virtualización *hardware*

A la hora de hacer una virtualización *hardware* hay varias opciones entre las que elegir. Las más ampliamente usadas son Xen y KVM. La principal diferencia entre ambas es que Xen ofrece paravirtualización y KVM ofrece virtualización nativa. La paravirtualización presenta a las máquinas virtuales una interfaz que es similar, pero no idéntica, al hardware de la máquina física en la que se aloja. Todas las llamadas con privilegios deben ser capturadas y traducidas a llamadas al hipervisor. El sistema operativo de la máquina virtual debe ser modificado para hacer estas capturas.

[Figuras, figuras, figuras]

La virtualización nativa permite hacer una virtualización *hardware* completa de manera eficiente. No requiere de ninguna modificación en el sistema operativo de la máquina virtual, pero necesita de un procesador con soporte para virtualización. KVM está incluido como un módulo del núcleo de Linux desde su versión 2.6.20, así que viene incluido por defecto en el sistema operativo de las máquinas del laboratorio.

Como los ordenadores del laboratorio poseen procesadores con extensiones de soporte para virtualización se eligió KVM para dar soporte a las máquinas virtuales. Esto significa que podemos usar cualquier sistema operativo, sin hacer ninguna modificación, para las máquinas virtuales.

2.2 Análisis de las infraestructuras de ejecución de trabajos distribuidos

2.3 Elección de la herramienta de gestión de configuración

Dentro de las herramientas de gestión de configuración, se estudió el uso de CFEngine y Puppet. CFEngine es una herramienta de configuración con un lenguaje declarativo en el que se especifican acciones a realizar para las clases. Está programado en el lenguaje C y ofrece un buen rendimiento y un control detallado de cómo se hacen las cosas. Puppet es una herramienta con un lenguaje declarativo en el que se especifica cuál debe ser el estado de los elementos a configurar. A diferencia de CFEngine, Puppet posee un nivel mayor de abstracción que permite un mejor

modelado de los recursos de un sistema. Por ejemplo, Puppet proporciona tipos para modelar usuarios, grupos, archivos, paquetes y servicios. Está programado en el lenguaje Ruby.

A la hora de realizar una extensión de la funcionalidad, se pensó que sería más fácil hacerla en la herramienta que más abstracción proporcionase y en el lenguaje en el que más fácil fuera modelar esta extensión. Por estas razones se eligió Puppet como la herramienta de gestión de configuración sobre la que hacer la extensión.

2.4 Extensión de la herramientas de configuración elegida

Una vez escogida la herramienta sobre la que se haría la extensión, quedaba por determinar cómo realizarla.

La primera opción que se barajó fue la de usar *Faces* de Puppet. *Faces* es una API para crear subcomandos y acciones dentro de Puppet. Analizada a fondo, esta API no proporcionaba una ventaja muy superior a la ejecución de comandos desde la consola del sistema operativo, y no interesaba crear una abstracción que facilitara el trabajo para posteriormente estar usando continuamente la línea de comandos.

La segunda opción que se barajó fue la de la creación de un tipo y un proveedor para ese tipo. Esta opción sí que presenta una ventaja considerable: podemos usar el tipo para modelar la infraestructura distribuida y podemos usar el proveedor para indicar como iniciar y mantener esa infraestructura. Esta aproximación se acerca más al modelo que usa Puppet, ya que definimos la infraestructura como si fuera un recurso más de los que posee Puppet. Así pues, esta es la aproximación que se tomó para realizar la extensión.

Capítulo 3
Extensión de Puppet para el modelado de una infraestructura distribuida

Extensión de Puppet.

3.1 Modelado del recurso *cloud*

Gramática simple

3.2 Funcionamiento del proveedor

Explicación de lo que hace el proveedor y los problemas que soluciona

Capítulo 4
Uso de la extensión aplicado a la infraestructura AppScale

Extensión de Puppet aplicada a AppScale.

4.1 Metamanifiesto

4.2 Fichero de roles

Capítulo 5
Uso de la extensión aplicado a la infraestructura web de tres niveles

[Revisar]

Una típica arquitectura de servicios web consta de al menos tres niveles: balanceo de carga, servidores web y base de datos. Cada uno de estos niveles está compuesto por al menos un elemento clave: el balanceador de carga, el servidor web y el servidor de base de datos, respectivamente. El balanceador de carga es el punto de entrada al sistema y el que se encarga, como su nombre indica, de repartir las peticiones de los clientes a los distintos servidores web. Los servidores web se encargan de servir las páginas web a los clientes y para ello, dependiendo de las peticiones que hagan los clientes, podrán leer o almacenar información en la base de datos. Para manipular dicha información los servidores web tendrán que comunicarse con el servidor de base de datos, que es el que hará efectiva la lectura y modificación de la información. La arquitectura puede complicarse añadiendo más niveles y añadiendo más elementos a los niveles existentes.

[Figura de la arquitectura]

Para demostrar la validez del modelo desarrollado, además de AppScale también se puede controlar una infraestructura de servicios web. En el ejemplo se ha validado una arquitectura que consta de un balanceador de carga, dos servidores web y un servidor de bases de datos. Como balanceador de carga se ha usado nginx y como servidor de base de datos se ha usado MySQL. La creación de la página web se ha hecho usando el framework Sinatra sobre el servidor web WEBrick. Todos ellos corren sobre máquinas con sistema operativo Ubuntu.

5.1 Metamanifiesto

La sintaxis en el metamanifiesto es fundamentalmente similar a la utilizada en el ejemplo de AppScale: el único campo que cambia sustancialmente es el type, que pasa de tener valor `appscale` a tener valor `web`. El resto de campos se comportan como lo hacían en el ejemplo de AppScale: el campo file contiene el fichero de descripción de roles, el campo images contiene los discos duros de las máquinas virtuales, el campo domain contiene el fichero de descripción de la máquina virtual, el campo pool contiene el conjunto de máquinas físicas a usar y el campo ensure contiene el estado en el que queremos que quede el cloud.

```
cloud {'mycloud':
```

```
    type     => "web",
    file     => "/etc/puppet/modules/cloud/files/web-simple.yaml",
    images   => ["/var/tmp/dceresuela/lucid-web.img", "/var/tmp/dceresuela
        /lucid-db.img"],
    domain   => "/etc/puppet/modules/cloud/files/mycloud-template.xml",
    pool     => ["155.210.155.70"],
    ensure   => running,
}
```

5.2 Fichero de roles

El contenido del fichero de especificación de roles sí que poseerá valores distintos a los que tenía el ejemplo de AppScale ya que estamos describiendo una arquitectura distribuida distinta. Los nuevos roles que pueden desempeñar las máquinas son:

`balancer` La máquina que desempeñará el rol de balanceador de carga.

`server` La lista de máquinas que desempeñarán el rol de servidor web.

`database` La máquina que desempeñará el rol de servidor de base de datos.

Un ejemplo completo del fichero de especificación de roles tendría un contenido similar a éste:

```
---
:balancer: 155.210.155.175
:server:
- 155.210.155.73
- 155.210.155.178
:database: 155.210.155.177
```

Capítulo 6
Uso de la extensión aplicado a la infraestructura Torque

Extensión de Puppet aplicada a Torque.

6.1 Metamanifiesto

6.2 Fichero de roles

Capítulo 7
Conclusiones

Conclusiones.

Bibliografía

[1] Puppet labs: The leading open source data center automation solution. `http://www.puppetlabs.com/`.

[2] J. Arundel. *Puppet 2.7 Cookbook*. PACKT PUB, 2011.

[3] Navraj Chohan, Chris Bunch, and Sydney Pang. Appscale design and implementation. *Science*, pages 1–17, 2009.

[4] Navraj Chohan, Chris Bunch, Sydney Pang, Chandra Krintz, Nagy Mostafa, Sunil Soman, and Richard Wolski. Appscale: Scalable and open appengine application development and deployment. In Dimiter R. Avresky, Michel Diaz, Arndt Bode, Bruno Ciciani, and Eliezer Dekel, editors, *CloudComp*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 57–70. Springer, 2009.

[5] J. Loope. *Managing Infrastructure With Puppet*. O'Reilly Media, 2011.

[6] LaTeX project team. *LaTeX documentation*. `http://www.latex-project.org/guides/`.

[7] J. Turnbull. *Pulling Strings with Puppet: Configuration Management Made Easy*. Apress Series. Apress, 2008.

[8] J. Turnbull and J. McCune. *Pro Puppet*. Pro to Expert Series. Apress, 2011.

Apéndice A
Ruby

Antes de poder programar en el lenguaje de programación Ruby deberemos instalar una serie de paquetes. Además de lo necesario para programar en Ruby también instalaremos una serie de herramientas de ayuda.

A.1 Instalación

Empezaremos instalando Ruby, IRB (Interactive Ruby Shell) y RDoc, la herramienta de generación de documentación de Ruby:

```
_: apt-get install ruby irb rdoc
```

A continuación instalaremos RubyGems. Para ello, descarga el paquete más actual de rubyforge. Durante el resto de la instalación usaremos como ejemplo el paquete 1.8.10, pero los pasos son análogos para cualquier otra versión.

Descomprimimos el paquete:

```
_: tar xvf rubygems-1.8.10.tgz
```

E instalamos RubyGems:

```
_: cd rubygems-1.8.10
_: ruby setup.rb
```

Una vez instalado, comprobamos la versión:

```
_: gem --version
1.8.10
```

Y la actualizamos a la última versión disponible. Es posible que antes de hacer este paso haya que actualizar el sistema operativo.

```
_: gem update --system
_: gem --version
1.8.24
```

En caso de que necesites actualizar el sistema operativo, se hace de esta manera:

```
_: apt-get upgrade
```

A.2 Problemas

Puede que durante la ejecución de programas Ruby, o la instalación de *gems* te encuentres con el siguiente error: `'require': no such file to load - mkmf (LoadError)`. La manera de solucionarlo es instalando el paquete `ruby1.8-dev`:

```
_: apt-get install ruby1.8-dev
```

También puede aparecer el error `no such file to load - net/https (LoadError)`. Para solucionarlo, instala `libopenssl-ruby`:

```
_: apt-get install libopenssl-ruby
```

A.3 Versiones

Software	Versión
Ubuntu	10.04
Ruby	1.8.7
RubyGems	1.8.10 (o superior)

Apéndice B
Libvirt

B.1 Instalación

Instalaremos `libvirt-ruby` mediante apt-get:

```
_: apt-get install libvirt-ruby
```

B.2 Versiones

Software	Versión
Ubuntu	10.04
Ruby	1.8.7
libvirt-ruby	0.0.7-1

Apéndice C
RabbitMQ

RabbitMQ es un *middleware* de mensajería que implementa el estándar AMQP (Advanced Message Queuing Protocol).

C.1 Instalación de RabbitMQ y Erlang

Empezaremos añadiendo la línea

```
deb http://www.rabbitmq.com/debian/ testing main
```

a nuestro fichero `/etc/apt/sources.list` de la siguiente manera:

```
_: echo "deb http://www.rabbitmq.com/debian/ testing main" >> \
/etc/apt/sources.list
```

Para evitar avisos acerca de paquetes que no han sido firmados, podemos añadir la clave pública de RabbitMQ a nuestra lista de claves:

```
_: wget http://www.rabbitmq.com/rabbitmq-signing-key-public.asc
_: apt-key add rabbitmq-signing-key-public.asc
```

Actualizamos el sistema con los nuevos cambios:

```
_: apt-get update
```

E instalamos el paquete de la manera habitual:

```
_: apt-get install rabbitmq-server
```

La instalación incluirá automáticamente los paquetes de Erlang necesarios para ejecutar RabbitMQ.

C.2 Instalación de los *plugins* para el soporte de AMQP y STOMP

RabbitMQ no basta para proporcionar el servicio de paso de mensajes que MCollective requiere. Para que sea capaz de proporcionarlo necesitamos instalar unos *plugins*. Para ello, vamos al fichero `/usr/lib/rabbitmq/lib/rabbitmq_server-2.7.1/plugins` y comprobamos si existen `amqp_client-2.7.1.ez` y `rabbitmq_stomp-2.7.1.ez`. Si no existen, los instalamos:

```
_: wget -q http://www.rabbitmq.com/releases/plugins/v2.7.1/\
amqp_client-2.7.1.ez
_: wget -q http://www.rabbitmq.com/releases/plugins/v2.7.1/\
rabbitmq_stomp-2.7.1.ez
```

Nota: Los nombres de los *plugins* pueden variar dependiendo de la versión instalada.

Activamos los *plugins*:

```
_: rabbitmq-plugins list
_: rabbitmq-plugins enable amqp_client           # El nombre puede cambiar
_: rabbitmq-plugins enable rabbitmq_stomp        # El nombre puede cambiar
```

y reiniciamos el servidor:

```
_: /usr/sbin/service rabbitmq-server restart
```

C.3 Configuración de la cuenta de MCollective

Añadimos el usuario `mcollective` y la contraseña `mcollective`:

```
_: rabbitmqctl add_user mcollective mcollective
```

Establecemos los permisos necesarios:

```
_: rabbitmqctl set_permissions -p / mcollective "^amq.gen-.*" ".*" ".*"
```

Y por seguridad borramos la cuenta de invitado:

```
_: rabbitmqctl delete_user guest
```

C.4 Comprobación de la instalación

Para comprobar que hemos instalado correctamente RabbitMQ podemos hacer lo siguiente:

```
_: invoke-rc.d rabbitmq-server start
Starting rabbitmq-server: SUCCESS
rabbitmq-server.
```

C.5 Versiones

Software	Versión
Ubuntu	10.04
RabbitMQ	2.7.1
Erlang	R13B03 (erts-5.7.4)

Apéndice D
MCollective

MCollective es un sistema de despliegue de servidores y de sistemas de ejecución de trabajos en paralelo. Su función principal dentro del ámbito de la administración de sistemas es la ejecución de una acción sobre un conjunto de servidores. Para lograr esto, MCollective se apoya en un *middleware* que proporciona un servicio publicador-suscriptor. En este tipo de programas, las publicaciones y suscripciones suelen implementarse mediante el paso de mensajes asíncrono. Una de las posibilidades que se pueden usar con MCollective es RabbitMQ (Apéndice C), que implementa el estándar de paso de mensajes AMQP (Advanced Message Queuing Protocol).

Dentro de la arquitectura MCollective, los componentes más importantes son:

- Cliente: Indica las órdenes a ejecutar.
- *Middleware*: Lleva las órdenes del cliente a los servidores.
- Servidor: Cumple las órdenes que le son enviadas.

El funcionamiento habitual de MCollective consiste en el lanzamiento de una orden desde el cliente que es transmitida a todos los servidores (*broadcast*) y éstos la llevan a cabo. En la figura D.1 se puede apreciar como la instrucción emitida por el cliente llega al corredor de mensajes (*broker*) y éste lo hace llegar a los servidores para que la cumplan.

D.1 Instalación

El primer paso consiste en la instalación de RubyGems. Para ello, consulta el apéndice A de instalación de Ruby y RubyGems.

Descargamos los paquetes `mcollective`, `mcollective-client` y `mcollective-common` de `http://downloads.puppetlabs.com/mcollective/`:

```
_: wget http://downloads.puppetlabs.com/mcollective/\
mcollective_1.2.1-1_all.deb
_: wget http://downloads.puppetlabs.com/mcollective/\
mcollective-client_1.2.1-1_all.deb
_: wget http://downloads.puppetlabs.com/mcollective/\
mcollective-common_1.2.1-1_all.deb
```

Dependiendo de si el nodo va a ser administrador o administrado tendremos que instalar unos u otros paquetes:

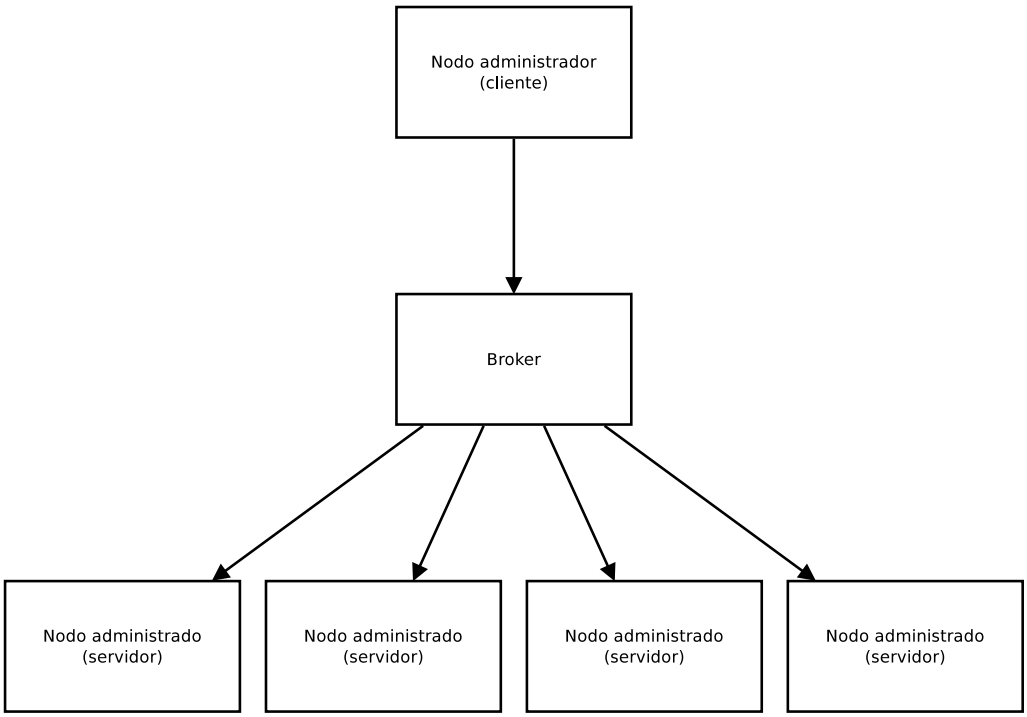

Figura D.1: Arquitectura de MCollective.

Paquete	Nodo administrador	Nodo administrado
mcollective-common	X	X
mcollective-client	X	
mcollective		X

Podemos instalar los paquetes de la siguiente manera:

```
_: dpkg -i mcollective*.deb
```

Una vez instalado MCollective, instalamos la librería `libstomp-ruby`:

```
_: apt-get install libstomp-ruby
```

D.2 Configuración

A continuación hay que editar el fichero de configuración de MCollective. En los nodos administradores (clientes) se encuentra en `/etc/mcollective/client.cfg` mientras que en los nodos administrados (servidores) se encuentra en `/etc/mcollective/server.cfg`. Nos aseguramos de que contengan los siguientes valores:

```
plugin.stomp.host = 155.210.155.ABC # Direccion IP del MQ broker
plugin.stomp.port = 61613
plugin.stomp.user = mcollective
plugin.stomp.password = mcollective
```

D.3 Comprobación de la instalación

En el servidor stomp lanza el siguiente comando:

```
_: /usr/sbin/service rabbitmq-server start
```

En los nodos administrados (servidores) lanza éste:

```
_: /usr/sbin/service mcollective start
```

Y en uno de los nodos administradores (clientes) lanzaremos el comando `mc-ping`. La salida obtenida al ejecutar el comando debería ser similar a la siguiente:

```
_: mc-ping
155.210.155.177                          time=46.06 ms
---- ping statistics ----
1 replies max: 46.06 min: 46.06 avg: 46.06
```

D.4 Versiones

Software	Versión
Ubuntu	10.04
MCollective	1.2.1-1
libstomp-ruby	1.8 (1.0.4-1)

Apéndice E
Facter y Puppet

E.1 Instalación de libopenssl-ruby

Si nuestro sistema no tiene instalado `libopenssl-ruby` tenemos que instalarlo:

```
_: apt-get install libopenssl-ruby
```

E.2 Creación del entorno de instalación de Facter y Puppet

Antes de instalar Facter y Puppet, hay que dar valores a ciertas variables:

```
_: FACTER_DIR="/root/facter-1.6.4"
_: PUPPET_DIR="/root/puppet-2.7.9"
_: PATH=$PATH:$FACTER_DIR/bin:$PUPPET_DIR/puppet/bin
_: RUBYLIB=$FACTER_DIR/lib:$PUPPET_DIR/lib
_: export PATH RUBYLIB
```

E.3 Instalación de Facter

Instalaremos Facter mediante una instalación por fuentes. Para ello primero tenemos que obtener las fuentes:

```
_: wget http://puppetlabs.com/downloads/facter/facter-1.6.4.tgz
```

Una vez conseguidas las fuentes lo descomprimimos:

```
_: gzip -d -c facter-1.6.4.tgz | tar xf -
```

Y posteriormente lo instalamos:

```
_: cd facter-1.6.4
_: ruby install.rb
```

E.4 Instalación de Puppet

Instalaremos Puppet también mediante fuentes. Obtenemos las fuentes:

```
_: wget http://puppetlabs.com/downloads/puppet/puppet-2.7.9.tgz
```

Lo descomprimimos:

```
_: gzip -d -c puppet-2.7.9.tgz | tar xf -
```

Y lo instalamos:

```
_: cd puppet-2.7.9
_: ruby install.rb
```

E.5 Configuración de Puppet

Nos aseguramos de que el fichero `/etc/puppet/puppet.conf` existe en nuestro sistema. Si no es así, podemos crearlo de la siguiente manera:

```
_: puppetmasterd --genconfig > /etc/puppet/puppet.conf
```

Tenemos que asegurarnos de que existen tanto un usuario como un grupo `puppet`. Si no existen, los creamos:

```
_: groupadd puppet
_: useradd -g puppet puppet
```

Por último tenemos que asegurarnos de que el directorio `/var/lib/puppet/rrd` pertenece al usuario y grupo `puppet`. Si no pertenece, cambiamos los permisos:

```
_: chown puppet /var/lib/puppet/rrd
_: chgrp puppet /var/lib/puppet/rrd
```

Nota: Si este directorio todavía no existe seguimos adelante con la instalación y ya revisitaremos este punto.

E.6 Comprobación de la instalación

Vamos a comprobar de manera rápida que la instalación de Facter y Puppet fue exitosa:

```
_: facter
[Muestra datos del sistema]
_: puppet describe -s user
[Muestra una descripcion del tipo usuario]
```

Ahora creamos un manifiesto simple en el fichero `manifest.pp`:

```
1 file {'testfile':
2   path    => '/tmp/testfile',
3   ensure  => present,
4   mode    => 0640,
5   content => "I'm a test file.",
6 }
```

Lo aplicamos:

```
_: puppet apply manifest.pp
```

Y comprobamos el contenido del fichero `/tmp/testfile` que Puppet debería haber creado:

```
_: cat /tmp/testfile
I'm a test file.
```

Ahora es momento de volver al directorio `/var/lib/puppet/rrd` y comprobar que tanto el usuario como el grupo de ese directorio tienen como valor `puppet`.

E.6.1 Comprobación del maestro de Puppet

Para comprobar que el maestro de puppet puede ejecutarse como un demonio tenemos que hacer:

```
_: puppet master
Could not prepare for execution: Got 1 failure(s) while initializing: \
change from directory to file failed:
Could not set 'file on ensure: Is a directory - /var/lib/puppet/facts
```

Si aparece este fallo tenemos que ir al fichero de configuración `/etc/puppet/puppet.conf` y comentar la línea con la propiedad `factdest`. Esta es la solución propuesta para el bug.

Si lo ejecutamos de nuevo no debería dar más problemas:

```
_: puppet master
```

E.6.2 Comprobación del agente de Puppet

Nota: Supondremos que el maestro de Puppet se está ejecutando en una máquina que tiene por nombre `puppet.example.com`. La máquina desde la que se ejecuta el agente tiene por nombre `node1.example.com`

Desde la máquina que tiene el agente, lanzamos la siguiente orden:

```
_: puppet agent --server=puppet.example.com \
--no-daemonize --verbose   # Usa nombres de dominio completos
warning: peer certificate won't be verified in this SSL session
info: Caching certificate for ca
warning: peer certificate won't be verified in this SSL session
warning: peer certificate won't be verified in this SSL session
info: Creating a new SSL certificate request for appscale-image1.cloud.net
info: Certificate Request fingerprint (md5): ...
warning: peer certificate won't be verified in this SSL session
[Se queda colgado]
[Ctrl + C]
```

Vamos a la máquina en la que se está ejecutando el maestro y hacemos lo siguiente:

```
_: puppet cert --sign node1.example.com
notice: Signed certificate request for node1.example.com
notice: Removing file Puppet::SSL::CertificateRequest node1.example.com at \
'/etc/puppet/ssl/ca/requests/node1.example.com.pem'
```

Y de vuelta a la máquina del agente:

```
_: puppet agent --server=puppet.example.com --no-daemonize --verbose
warning: peer certificate won't be verified in this SSL session
info: Caching certificate for node1.example.com
notice: Starting Puppet client version 2.7.9
info: Caching certificate_revocation_list for ca
info: Caching catalog for node1.example.com
info: Applying configuration version '1331813472'
info: Creating state file /var/lib/puppet/state/state.yaml
notice: Finished catalog run in 0.03 seconds
[Se queda colgado]
[Ctrl + C]
notice: Caught INT; calling stop
```

Si no tienes una configuración preparada para tu nodo te aparecerá un error, pero no te preocupes: Puppet está funcionando correctamente. El error será similar a éste:

```
_: puppet agent --server=puppet.example.com --no-daemonize --verbose
notice: Starting Puppet client version 2.7.9
info: Caching certificate_revocation_list for ca
err: Could not retrieve catalog from remote server: Error 400 on SERVER: \
Could not find default node or by name with 'node1.example.com, node1' \
on node node1.example.com
notice: Using cached catalog
err: Could not retrieve catalog; skipping run
[Se queda colgado]
[Ctrl + C]
notice: Caught INT; calling stop
```

E.6.3 Comprobación del sistema Puppet

Creamos el fichero `/etc/puppet/manifests/site.pp` en la máquina del maestro con el siguiente contenido:

```
1  # Create "/tmp/testfile" if it doesn't exist.
2  class test_class {
3      file { "/tmp/testfile":
4          ensure => present,
5          mode   => 644,
6          owner  => root,
7          group  => root
8      }
9  }
10
11 # tell puppet on which client to run the class
12 node 'node1' {            # Notice it is node1 and not node1.example.com
13     include test_class
14 }
15
```

```
16 node 'puppet' {            # Notice it is puppet and not puppet.example.com
17 }
```

Y desde la máquina del agente tecleamos esto:

```
_: puppet agent --server=puppet.example.com --no-daemonize --verbose
```

Comprobamos que en la máquina del agente se ha creado el fichero `/tmp/testfile`:

```
_: cat /tmp/testfile
I'm a test file.
```

E.7 Problemas

Durante la ejecución de Puppet pueden ocurrir varios errores. Como la información de Puppet puede ser en ocasiones algo críptica se muestra a continuación una colección de los errores más comunes y su solución:

```
_: puppet master
err: /File[/var/lib/puppet/facts]/ensure: change from directory to file \
failed: Could not set 'file on ensure: Is a directory - /var/lib/puppet/facts
```

Solución: Comenta la propiedad factdest en el fichero de configuración de Puppet en la máquina del maestro.

```
_: puppet agent --server=puppet.example.com --no-daemonize --verbose
err: Could not retrieve catalog from remote server: Error 400 on SERVER: \
Could not find default node or by name with ...
```

Solución: Necesitas incluir el nombre del agente en el fichero `site.pp` de la máquina del maestro. O puedes incluir un nodo por defecto:

```
1 node 'default' {
2 }
```

```
_: puppet master
err: Could not prepare for execution: Retrieved certificate does not match \
private key; please remove certificate from server and regenerate it with \
the current key
```

Solución: Cambia el nombre del directorio `/etc/puppet/ssl` a `/etc/puppet/ssl.old` y prueba de nuevo.

```
_: puppet agent --server=puppet.example.com --no-daemonize --verbose
err: Could not send report: SSL_connect returned=1 errno=0 state=SSLv3 read \
server certificate B: certificate verify failed. This is often because the \
time is out of sync on the server or client
```

Solución: Compleja, mejor mira aquí.

```
_: puppet master
err: Could not run: Could not create PID file: /var/lib/puppet/run/master.pid
```

Solución: El maestro de Puppet ya se está ejecutando.

```
_: puppet apply manifest.pp
err: Could not evaluate: Puppet::Util::Log requires a message
```

Solution: Puppet ha salido de manera abrupta. Si estás ejecutando un proveedor asegúrate de que sales de él con `return` y no con `exit`.

E.8 Versiones

Software	Versión
Ubuntu	10.04
Ruby	1.8.7
Facter	1.6.4
Puppet	2.7.9
libopenssl-ruby	4.2

Apéndice F

Arquitectura Web

F.1 Balanceador de carga

Usaremos nginx como balanceador de carga, y no como servidor web, que es la manera más habitual de verlo en funcionamiento.

F.1.1 Instalación

Para instalar nginx, haz lo siguiente:

```
_: apt-get update
_: apt-get upgrade
_: apt-get install nginx
```

F.1.2 Configuración

Para configurar nginx debemos añadir la parte de balanceo de carga al fichero de configuración `/etc/nginx/nginx.conf`. Como este fichero no es excesivamente grande, se muestra en su totalidad con la parte modificada resaltada:

```
user www-data;
worker_processes  1;

error_log  /var/log/nginx/error.log;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
    # multi_accept on;
}

http {
    include       /etc/nginx/mime.types;

    access_log        /var/log/nginx/access.log;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;
    tcp_nodelay        on;

    gzip  on;
    gzip_disable "MSIE [1-6]\.(?!.*SV1)";

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;


    ### Modified
    upstream web_servers {
      server 155.210.155.73:4567;
      server 155.210.155.178:4567;
    }

    server {
      listen 155.210.155.175:80;
      location / {
        proxy_pass http://web_servers;
      }
    }
    ### End of modification


}
```

F.1.3 Ejecución

Para iniciar nginx haremos uso del *script* localizado en `/etc/init.d`. Dicho *script* puede ser usado tanto para iniciarlo:

```
_: /etc/init.d/nginx start
```

como para pararlo:

```
_: /etc/init.d/nginx stop
```

F.1.4 Versiones

Software	Versión
nginx	0.7.65

F.2 Servidor web

Como servidor web usaremos WEBrick, que es el servidor web que viene por defecto con la instalación de Ruby. Instalando Ruby instalaremos a la vez el servidor web.

F.2.1 Instalación

Lo primero que hay que hacer es instalar Ruby y RubyGems. Para ello, consulta el apéndice A de instalación de Ruby.

Una vez instalado Ruby y RubyGems, instalaremos el paquete `ruby-dev`:

```
_: apt-get install ruby1.8-dev
```

Y ahora instalaremos el soporte necesario para interactuar con la base de datos:

```
_: apt-get install libmysqlclient-dev
_: gem install mysql
```

La aplicación web será desarrollada usando Sinatra. Antes de comenzar con la instalación, comprueba que tu versión de RubyGems es igual o superior a la 1.3.6. Esto puede hacerse fácilmente de la siguiente manera:

```
_: gem --version
1.3.6
```

Para instalar Sinatra, haremos lo siguiente:

```
_: gem install sinatra
Successfully installed rack-1.4.1
Successfully installed rack-protection-1.2.0
Successfully installed tilt-1.3.3
Successfully installed sinatra-1.3.2
4 gems installed
Installing ri documentation for rack-1.4.1...
...
Installing ri documentation for sinatra-1.3.2...
Installing RDoc documentation for rack-1.4.1...
...
Installing RDoc documentation for sinatra-1.3.2...
```

Nota: Puede llevar un tiempo hasta que el proceso de instalación muestre cosas por pantalla.

F.2.2 Ejecución

Una vez que la instalación ha finalizado, vamos a crear la primera aplicación web. Guarda el siguiente fichero como `web.rb`:

```
1 require 'rubygems'
2 require 'sinatra'
3
4 get '/' do
5   'Hello world!'
6 end
```

y lanza el servidor web:

```
_: ruby web.rb
```

Nota: Para salir Ctrl + C.

En tu navegador web preferido ve a la dirección `http://localhost:4567` y encontrarás la aplicación web que acabamos de crear.

F.2.3 Añadiendo soporte para la base de datos

Para interactuar con la base de datos usaremos ActiveRecord. Este componente es parte de Ruby On Rails, pero también existe como una *gem* independiente. Vamos a instalarla:

```
_: gem install activerecord
```

Ahora vamos a crear una segunda aplicación web. Guarda el siguiente fichero como `web2.rb`:

```
1 require 'rubygems'
2 require 'sinatra'
3 require 'active_record'
4
```

```
5  class Article < ActiveRecord::Base
6  end
7
8  get '/' do
9     'Hello there!'
10 end
```

y lanza el servidor web como antes:

```
_: ruby web2.rb
```

En tu navegador web ve a la dirección `http://localhost:4567` y encontrarás la aplicación web. Muestra lo mismo que la primera aplicación web, pero hemos incluido (aunque no usado) el soporte para la base de datos.

F.2.4 Versiones

Software	Versión
Ruby	1.8.7
RubyGems	1.8.21
Sinatra	1.3.2
ActiveRecord	3.2.3

F.3 Base de datos

Como servidor de base de datos usaremos MySQL.

F.3.1 Instalación

Para instalar MySQL, haz lo siguiente:

```
_: apt-get install mysql-server
```

F.3.2 Configuración

Para configurar los ajustes básicos hay que editar el fichero `/etc/mysql/my.cnf`. Por ejemplo, si vamos a aceptar conexiones de otra máquina, hay que modificar el parámetro `bind_address`. En nuestro caso, lo modificaremos para aceptar conexiones del servidor web:

```
bind_address = 155.210.155.73
```

Nota: Si estás usando Ubuntu 10.04 debido a un *bug* es mejor que comentes toda la línea. Además nosotros usaremos dos servidores web, así que mejor la comentamos:

```
#bind_address = 155.210.155.73
```

Vamos a reiniciar el servidor para que los cambios surtan efecto:

```
_: /usr/bin/service mysql restart
```

Para comprobar que ha sido instalado correctamente, podemos hacer lo siguiente:

```
_: mysql -u root -p     # Introduce MySQL's root password
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 34
Server version: 5.1.61-0ubuntu0.10.04.1 (Ubuntu)
...
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE mydb;
```

F.3.3 Ejecución

Para ejecutar el servidor de bases de datos usaremos el programa `service` localizado en `/usr/bin/service`. Sirve tanto para iniciarlo:

```
_: /usr/bin/service mysql start
```

como para pararlo:

```
_: /usr/bin/service mysql stop
```

F.3.4 Versiones

Software	Versión
mysql	5.1.61