# Monitorización distribuida e inteligente de servidores con sistema operativo Linux

**Trabajo final de graduación de**
**Ingeniería en Sistemas de Información**

**Autor:** Federico Almada


**Universidad Siglo 21**

**2013**

*Abstract*

Server monitoring has become a key element for companies that are willing to protect their assets, either to prevent issues that could be predicted over time or to improve resource usage during their life cycle. In this paper, you will be able to understand why this activity is important for any kind of computers networks, but also get to know the different products that are currently available on the market to perform server monitoring. In the end, the feasibility of introducing P2P, a technology which could start a revolution on this particular market, is analyzed, in addition to suggesting other improvements to enable a smarter monitoring focused mainly on Linux servers.

*Resumen*

La monitorización de servidores se ha convertido en una pieza fundamental por parte de las empresas a la hora de proteger sus activos, tanto por la prevención ante sucesos predecibles, como también por la mejor utilización de los recursos a lo largo de su ciclo de vida. En este trabajo final, usted podrá comprender porque esta actividad es importante para cualquier red de sistemas, a la vez de conocer los distintos productos presentes en el mercado que permiten llevarla a cabo. Por último, se hace un estudio de factibilidad sobre la incorporación de tecnología P2P, que podría revolucionar este mercado, a la vez de sugerir mejoras que permitan una monitorización más inteligente, enfocada principalmente en servidores con sistema operativo Linux.

## Índice de contenido

# Introducción

Las industrias tienen un consumo cada vez mayor en tecnologías de información, destacando la renovación de servidores ya instalados, como también la construcción de sistemas nuevos. Esto genera una necesidad de mantenimiento de los recursos informáticos a fines de garantizar que los mismos funcionen debidamente.

En empresas pequeñas, el mantenimiento y control de los servidores puede hacerse de forma manual, pero al incrementar la complejidad y tamaño de las redes que estas poseen, se hace necesario utilizar herramientas de monitorización automática, que puedan alertarnos ante eventos extraordinarios y/o perjudiciales para el sistema.

Existen muchas herramientas de monitorización en el mercado, tanto pagas como gratuitas, de código propietario o abierto. La gran mayoría de estas herramientas están pensadas para redes corporativas medianas o grandes, debido a que hacen uso de estructuras *cliente-servidor*, requiriendo que el eslabón más importante de ese modelo (el servidor) posea una buena cantidad de recursos para manejar la información reportada por los demás eslabones (los clientes).

Sumado al incremento de complejidad de una red de servidores, nos encontramos con el aumento exponencial del trabajo necesario para poder mantener dicha red en forma saludable y funcional a los objetivos de la empresa. Por este motivo, los administradores de sistemas buscan soluciones sencillas, eficaces y eficientes, que aporten información procesada dentro de contexto, en vez de aportar sólo datos dispersos que requieran de un análisis posterior para poder evaluar el estado de la red.

En este trabajo final de graduación, se estudiarán los distintos productos disponibles de monitorización en el mercado, enfocados en el sistema operativo Linux, y se examinará la factibilidad técnica, económica y operacional de distintas mejoras, a fines de plantear una solución basada en la arquitectura del P2P (peer-to-peer) que incorpore algoritmos inteligentes para la labor de monitorizar servidores.

## Antecedentes

Al momento de realizar este trabajo de final de grado, no se han encontrado antecedentes relacionados de forma directa con la idea aquí propuesta, pero si avances en distintas áreas de la monitorización de servidores, que no están documentados formalmente, sino presentados como características de productos comerciales que se encuentran a la venta en el mercado.

## Supuestos

Debido a la falta de datos estadísticos de acceso público por parte de las empresas que desarrollan y/o comercializan software de monitorización de servidores, se opta por guiar el análisis con supuestos y preguntas, en vez de hipótesis, dado que esta última no podría ser demostrada en concreto y de forma objetiva para finalizar el trabajo.

En base al conocimiento previo de algunos productos de monitorización existentes en el mercado, como *IBM Tivoli Monitoring* y *Nagios*, partimos de los siguientes supuestos:

- El *software* de monitorización (para servidores Linux) disponible en el mercado, hace uso de una estructura compleja y con eslabones críticos, lo que supone un riesgo muy alto en el caso de que alguno de estos deje de estar disponible.
- Todas las soluciones existentes en el mercado utilizan una estructura *cliente-servidor*.
- La estructura *cliente-servidor* que utilizan los productos existentes en el mercado, generan un alto consumo de ancho de banda desde los nodos que se reportan, lo que conlleva la necesidad de planificar previamente la cantidad de actualizaciones, a fines de evitar una congestión de red en el enlace con el nodo servidor.
- A medida que aumenta la complejidad y tamaño de una red, las empresas proveedoras de las soluciones existentes recomiendan segmentar la misma a fines de simplificar la monitorización de los servidores. Esto conlleva la necesidad de agregar más nodos servidores a la red, introduciendo nuevos eslabones críticos que pueden fallar.

*Preguntas*

Las preguntas formuladas a continuación, tienen base en la experiencia propia de más de 5 años trabajando con productos de monitorización, aunque también respaldada por la opinión de profesionales que se desempeñan hace más de 3 años en el mismo rubro.

- ¿Es posible desarrollar un software de monitorización de servidores que utilice una estructura distinta a la de *cliente-servidor*?
- ¿Es posible eliminar los eslabones críticos presentes en redes de distintos tamaños respecto al software de monitorización de servidores?
- ¿Es viable el desarrollo de un software de monitorización basado en la tecnología P2P?

- ¿Es posible desarrollar un software de monitorización que haga uso de menos recursos en una red? (menos tráfico, menores requerimientos de hardware, menor mantenimiento por parte de los administradores de sistema).

## Justificación

Este trabajo final de graduación busca estudiar la factibilidad de introducir mejoras que supondrían un cambio importante en el funcionamiento de los productos de monitorización de servidores. La novedad fundamental radicaría en el cambio de una estructura cliente-servidor a una de P2P, lo que supone un salto importante gracias a las posibilidades de esta última.

En caso de encontrar que un producto como éste es factible, se podrán plantear nuevas formas de monitorizar redes, disminuyendo costos en general, y mejorar los tiempos de respuestas de las distintas herramientas que normalmente se utilizan para solucionar y/o prevenir problemas en una red de sistemas. Esto supondría una importante reducción de costos para empresas de gran tamaño, como también daría la posibilidad a empresas pequeñas de incorporar monitorización automatizada en sus redes, disminuyendo así los recursos necesarios para que estas estén tanto seguras como saludables.

En lo personal, y dada mi experiencia laboral de más de 5 años en la administración de servidores, me interesa conocer las posibilidades de crear un producto de monitorización que sea realmente útil para los administradores de sistemas, pero a la vez atractivo para las empresas que los utilizan en sus redes.

# Objetivos

## Objetivo general

Realizar una investigación sobre los actuales productos de monitorización de servidores con Linux, para evaluar la factibilidad de introducir mejoras innovadoras en el desarrollo de un prospecto de producto, obteniendo como fin último un listado de los requisitos básicos que sirva como cimiento para un desarrollo futuro.

## Objetivos específicos

- Analizar las características de los distintos productos disponibles en el mercado, tanto de código abierto como propietario, gratuitos y comerciales, para así identificar las características mínimas y necesarias que hagan este producto vendible.
- Evaluar la factibilidad de utilizar tecnología P2P en la monitorización de servidores, para hacer uso de una arquitectura distribuida que reemplace a la de *cliente-servidor* de forma efectiva y eficiente.
- Estudiar módulos y diagramas disponibles sobre monitorización de aspectos básicos como uso de procesador, sistemas de archivo, memoria de intercambio y disponibilidad de servicios de sistema, a fines de proponer mejoras que permitan una monitorización inteligente, minimizando la intervención humana.

## Límite y Alcance

*Límite del proyecto*

El límite de este trabajo comienza con el estudio del concepto de monitorización, finalizando con la entrega de una propuesta detallada de como debería ser un producto prototipo que cumpla con los requisitos básicos necesarios, haciendo uso de una arquitectura basada en P2P.

*Alcance del proyecto*

El trabajo se divide en tres partes:

1. Teoría: Definir conceptos básicos y tecnologías relacionadas con la monitorización, para lograr un mejor entendimiento del tema bajo estudio y su importancia histórica como futura.
2. Análisis: Realizar una recolección de datos de productos competidores en el mercado de la monitorización, analizar sus características y hacer una comparación entre estos para comprender que mercados cubren, que los distingue y en que se asemejan.
3. Propuesta: Definir las características de un producto prototipo que abarque las tecnologías a utilizar, describiendo los datos y algoritmos que permitirían su funcionamiento, requisitos básicos y destacados (en base a lo recolectado sobre productos competidores), finalizando con la factibilidad técnica, operacional y económica de desarrollarlo.

## Metodología

Dado que el tema de este trabajo final de graduación no cuenta con antecedentes publicados que tengan relación directa con el objetivo del mismo, es necesario que la investigación a desarrollar sea del tipo exploratorio.

Si bien existen estudios sobre monitorización de redes y servidores, algunos enfocados en P2P, todos estos son en su mayoría a nivel teórico o justificando los motivos del cambio a esta tecnología, pero ninguno analiza la factibilidad de crear un software capaz de operar bajo un entorno real que haga uso efectivo de una estructura distribuida.

Para poder profundizar sobre el tema, se hace enfoque en la búsqueda de contenido proveniente tanto de fuentes propias de las empresas que comercializan software de monitorización, como también *papers* de universidades sobre las distintas tecnologías involucradas en la problemática y propuesta del trabajo. Toda la información recolectada es almacenada en una herramienta llamada Zotero, que sirve como base de datos para citas y referencias de la bibliografía.

Luego de recolectar los datos de productos de monitorización disponibles en el mercado, se analizan los mismos para poder categorizarlos y comparar sus características, delimitando así cuales son los atributos mínimos que dan cimiento a un software de monitorización. Las tablas comparativas y los gráficos son utilizados para mejorar la comprensión sobre aquellos conceptos donde su aplicación sea necesaria y justificada.

# Marco Teórico

## Capítulo I: Monitorización

*Definición y concepto*

Según la Real Academia Española (RAE), monitorizar es la acción de "observar mediante aparatos especiales el curso de uno o varios parámetros fisiológicos o de otra naturaleza para poder detectar posibles anomalías". La monitorización, por tanto, es una forma de conocer el estado de una o varias entidades, cualquiera sean estas, a fines de prever cambios en el comportamiento y/o entorno que las rodean.

*Aplicación*

La monitorización se practica en distintos ámbitos, siendo un ejemplo el campo de la salud, cuando se monitoriza el estado de un paciente; en una fábrica, al monitorizar las máquinas de producción; en una compañía crediticia, al observar y analizar el comportamiento de sus clientes, con el fin de detectar actividades fraudulentas; y hasta en sistemas informáticos, donde se busca preservar la salud de los recursos tanto tangibles (hardware) como intangibles (datos y software).

En todos los casos antes descriptos, la monitorización cumple el mismo objetivo general, salvo que sus parámetros, objeto y modo de control difieren.

En una clínica, los parámetros podrían ser la presión arterial, las pulsaciones del corazón o el nivel de glucosa en sangre, siendo el objeto de estudio un paciente, y el modo de control un tensiómetro, un estetoscopio y un glucómetro respectivamente.

Si analizáramos una fábrica que produce algún ítem en particular, los parámetros podrían ser el peso y el contenido, mientras que el modo de control serían una balanza y un escáner que permita verificar si se cumple con los estándares de calidad prometidos.

Más adelante estudiaremos que parámetros se monitorizan en un servidor, y los distintos modos de hacer tal tarea.

*Origen y evolución*

La necesidad de monitorizar no surge hasta la Revolución Industrial, dado que anteriormente los

trabajos se realizaban de forma manual. De este modo, quien realizaba la tarea, aplicaba el sentido común al momento de construir un objeto, minimizando así la necesidad de aplicar correcciones. En cierto modo, el proceso de monitorización existía, sólo que no se lo veía como una actividad en sí misma, sino como parte del proceso de producción.

La introducción de máquinas complejas en la industria, con procesos automatizados, hizo más evidente la necesidad de la monitorización. La automatización de procesos hace que el rol del operario deje de estar centrado en el uso de la fuerza humana, pasando a tener más peso el uso de su inteligencia para controlar que las tareas sean realizadas correctamente por las máquinas. Invenciones como el telar de Jacquard, en 1801, dejan muy en claro este cambio, ya que el operario simplemente introducía el resultado esperado por una tarjeta perforada, para luego monitorizar que la máquina funcionase bien durante la fabricación. Finalmente, verificaba el producto saliente y hacía correcciones en caso de ser necesario. (Essinger, 2007)

Inicialmente, la monitorización de máquinas se hacía de forma manual, por el mismo operador, teniendo que interrumpir el proceso si veía que algo estaba fuera de los parámetros aceptables. Con el paso del tiempo, se fueron creando máquinas que sirven para monitorizar otras, reduciendo así tanto la carga de trabajo para el operador, como también la cantidad de personal necesario para controlar todo el entorno. Pese a todos los avances, la necesidad del "sentido común" y la carencia de uno propio por las máquinas (las mismas son programadas de forma limitada por los humanos), hacen necesario la presencia de un operario como último eslabón de la cadena, a fines de que todo funcione bien.

En resumen, la monitorización se puede hacer de forma automatizada (por una máquina) como de forma manual (por un operario), siendo ambos modos complementarios y no excluyentes.

*Monitorización de ordenadores*

El primer ordenador (del año 1948), llamado SSEM (Small-Scale Experimental Machine) o "The Baby" (cuya arquitectura fue la base para los ordenadores de hoy en día, siendo de tipo electrónico, con programas almacenados en memoria y basados en aritmética binaria), incluía ya un módulo "Monitor and Control". Este módulo era utilizado por el operador tanto para controlar la máquina, como también para ver el estado actual de la misma. (Napper, 1998)

La monitorización de ordenadores se puede realizar tanto a nivel de hardware como a nivel de software. En el primer caso, la información a analizar debe provenir de componentes eléctricos y su

salida será normalmente una luz prendida/apagada, de distinto color o incluso medidores que recaban sus datos de un sensor, según el tipo de control que se realice. A nivel de software, en cambio, la monitorización puede realizarse no sólo para componentes de hardware, sino también a nivel de aplicaciones, servicios de sistema, usuarios, seguridad del sistema, entre otros. (Schubert, 2008, cap. 4)

El incremento del uso del ordenador, junto con la aparición de modelos más veloces, de mayor capacidad de almacenamiento y de menor costo, han permitido mejorar la monitorización para que no sólo se alerte sobre sucesos extraordinarios, sino que también permita tener una vista general de la evolución del estado de un sistema en el tiempo, pudiendo incluso accionar previo a la ocurrencia de un problema, o dar recomendaciones (basadas en reglas preestablecidas) para disminuir el impacto ante un evento perjudicial.

## Capítulo II: Importancia de la Monitorización

*¿Por qué se monitorizan los servidores?*

En un estudio de Computer Economics (2011, págs. 4–6,18), se afirmaba que para el año 2011 un 60% de las empresas tendría un incremento en sus presupuestos de TI. Para el año 2012, una actualización del mismo estudio subía ese porcentaje a un 62% (Computer Economics, 2012). Como se puede notar, este porcentaje viene incrementando año tras año, y gran parte del presupuesto termina siendo destinado principalmente a la actualización de los sistemas actuales, seguido por la incorporación de nuevos sistemas, y en tercer lugar para mejorar la infraestructura TI.

Este incremento en el consumo de tecnología informática, requiere que las empresas inviertan parte de su presupuesto en la administración de los sistemas que, en el caso de redes complejas, puede significar un costo importante. Una de las soluciones más buscadas es la reducción de la complejidad de la infraestructura, pero esto sólo se puede lograr en entornos muy bien administrados y con estándares altos. En resumen, los casos no abundan. (Zukis, Loveland, Horowitz, Verweij, & Bauch, 2008, p. 3)

Otra de las alternativas viables que existen en el mercado, es la utilización de herramientas de monitorización, las cuales permiten reducir la complejidad de la tarea misma de administración de sistemas, ya que por medio del uso de reglas y alertas, un reducido grupo de personas puede hacerse cargo de un gran número de servidores.

Estas herramientas de monitorización funcionan en base a controles previamente establecidos (ya sea los que vienen por defecto, o los agregados en forma personalizada), que son ejecutados dentro de cada sistema en un intervalo de tiempo predeterminado, y en el caso de detectar alguna anomalía, permiten tomar acciones correctivas y/o informar mediante una alerta, para que un operador se haga cargo (normalmente, un administrador de sistema).

*¿Qué parámetros se monitorizan de un servidor?*

Toda herramienta de monitorización existente en el mercado cuenta mínimamente con la posibilidad de monitorizar los recursos básicos de sistema, siendo estos el procesador (CPU), la memoria, los discos rígidos y la red.

Cada uno de estos recursos se puede monitorizar de diversas formas con herramientas nativas que vienen en el sistema operativo, o se pueden implementar con llamadas directas al sistema, emulando el comportamiento de dichas herramientas y obteniendo resultados más precisos.

A continuación se listan los recursos y que tipo de información se debe conocer de cada uno:

- Procesador: El uso promedio del procesador, o también llamado "carga del sistema", es un valor importante a la hora de monitorizar este recurso. Con este dato podemos saber si un sistema está necesitando más ciclos de procesamiento de los que puede manejar (cuando el valor es mayor a 1, en el caso de un sistema con 1 solo procesador), o si está mal aprovechado, en el caso que el valor sea siempre cercano a 0. En cualquier entorno UNIX, estos valores se pueden obtener por medio del comando "uptime". Por otro lado, también es posible, e importante, conocer cuanto del procesador está siendo utilizado por el sistema en sí, por las aplicaciones, por espera de entrada y salida de disco, valores que se pueden obtener por medio del comando "top" o sus variantes. (Johnson, 2005, pp. 59–69)
- Memoria: Cuando hablamos de memoria, debemos referirnos tanto a la memoria física disponible (RAM), como también a la memoria de intercambio (SWAP), que puede crear por medio de un archivo o de una partición dedicada. Juntas, reciben el nombre de "memoria virtual", y establecen el límite de memoria asignada a un sistema. Para monitorizar este valor, se pueden utilizar herramientas de análisis de rendimiento, como "vmstat", u otras sencillas como "free", "ps", "top" y "swapon". Lo que normalmente se analiza de la memoria es la paginación, que es básicamente el movimiento de páginas de memoria entre la RAM y la SWAP. (Johnson, 2005, pp. 69–73). En el caso que haya mucho movimiento de datos, se produce el fenómeno de "trashing", en donde el sistema se pasa más tiempo moviendo páginas que haciendo tareas para el usuario, lo que resulta en una pérdida de rendimiento notable. (Denning, 1968, pp. 1–2)
- Disco rígido: La velocidad de los discos, y la latencia de estos para la lectura de datos, son aspectos que se tienen muy en cuenta a la hora de monitorizar. Una pérdida en velocidad de los discos indicaría que posiblemente esté dañado o que un conector no está funcionando bien. Un incremento en la latencia, podrían indicar que una aplicación está provocando un cuello de botella en el sistema, que existe algún problema de controladores o en el propio hardware. Estos datos se pueden obtener con herramientas como "iostat", exclusiva para monitorizar actividades de entrada y salida del sistema, "sar", que provee información

detallada del disco, como también su relación con otros recursos del sistema. Dependiendo de si los discos son locales o provienen de una “Storage Area Network” (SAN), existen herramientas específicas que se pueden utilizar, pero que varían según el fabricante de cada producto. (Johnson, 2005, pp. 73–78)

- Red: El uso de la red entre sistemas ha pasado a ser una parte fundamental dentro de empresas como en Internet. En este caso, el análisis se da tanto desde la velocidad de transmisión (recepción y envío de datos), como también las rutas que toman los paquetes, la latencia entre dos servidores, y hasta la velocidad de negociación entre los dispositivos que se comunican en la red. Existen muchas herramientas que nos ayudan a monitorizar la red, entre las que encontramos “netstat”, “nfsstat” “tcpdump”, “ethtool”, “ifconfig”, “route”, “ping”, “traceroute”, “ss” y “nslookup”. El comando más importante es “netstat”, que nos provee de estadísticas de red, como también la posibilidad de ver las conexiones activas, los puertos abiertos, y hasta las rutas por las que viajan los paquetes de datos. (Johnson, 2005, pp. 78–83)

*Seguridad*

Si bien la monitorización de los recursos básicos de un sistema es importante, también lo es la seguridad de la red y de cada uno de los componentes de la misma. Los *firewalls* o cortafuegos, que tienen como objetivo limitar las conexiones entrantes y salientes en bases a reglas establecidas, deben ser monitorizados para saber si están siendo atacados por múltiples conexiones entrantes de uno o varios nodos remotos, lo cual podría provocar una denegación de servicio o simplemente hacer un consumo excesivo de recursos provocando daños mayores.

Por otro lado, se debe poder monitorizar la cantidad de accesos remotos fallidos a un sistema (lo que podría indicar que alguien intenta entrar al servidor aplicando fuerza bruta); intentos de escalamiento de privilegios erróneos (para ser administrador y tener control total), y hasta la integridad de archivos de sistema, simplemente para garantizar que el servidor está intacto respecto a una configuración base definida por el administrador de sistemas.

Se pueden monitorizar muchos aspectos dependiendo del nivel de seguridad del sistema, como ser la validez de los certificados de SSL (Secure Socket Layer, encargados de proveer comunicaciones seguras y cifradas), los usuarios con claves vencidas o vacías, los permisos de archivos críticos del sistema, las reglas del módulo de seguridad de claves, entre otros.

Como se habrá notado, la seguridad puede abarcar muchos aspectos, y tenerlo en cuenta dentro de una herramienta de monitorización es de vital importancia para conseguir un producto completo y apetecible para las empresas.

*Aplicaciones/Procesos*

El uso más común que se le suele dar al software de monitorización es el control de aplicaciones/procesos para saber si estos están funcionando. De esta forma, el software nos alerta si la herramienta de copia de seguridad no está siendo ejecutada como se debe, o si la aplicación del cliente se acaba de cerrar de forma inesperada.

En la mayoría de los casos, este tipo de situaciones es el desenlace de una serie de problemas, por lo que si un proceso deja de correr, quien verifique tal alerta debe ser capaz de encontrar la causa raíz por la cual se llegó a esa situación, y solucionar el problema previo a reiniciar la aplicación o proceso afectado, caso contrario, el incidente volverá a repetirse.

*Aplicaciones críticas*

Podríamos decir que las aplicaciones críticas entran en el concepto de aplicaciones o procesos, aunque estas poseen mayor atención por parte de la empresa debido a su importancia. La criticidad se da por el hecho que no pueden estar fuera de servicio, ya sea porque son componentes vitales para otras aplicaciones, o bien porque su caída provocaría pérdidas importantes a la empresa (monetarias, de imagen pública o simplemente de capacidad ociosa de los recursos que se liberan cuando la aplicación no funciona).

En estos casos, la monitorización suele realizarse en menores intervalos de tiempo, con controles más estrictos y certeros, y con mayor prioridad por sobre otras reglas de monitorización. Usualmente, este tipo de aplicaciones suele tener alguna redundancia a nivel físico o lógico, como puede ser un *Cluster*, tecnología que se comentará más adelante en este mismo capítulo.

*Servicios de sistema*

Los servicios de sistema podrían considerarse como los servicios públicos en un hogar. Sin agua, electricidad y gas, una persona puede tener un difícil momento para hacer su rutina diaria. En los servidores, los servicios cumplen el rol de ayuda para que otras aplicaciones funcionen, e incluso para que las mismas herramientas de monitorización tengan una tarea más sencilla a la hora de

hacer su trabajo.

Podríamos decir que un servicio de sistema es un proceso, pero al ser de vital importancia para un servidor, entran en una categoría especial que los ubica incluso más arriba que las aplicaciones críticas. Aún así, el tiempo entre controles no es tan importante, ya que estos procesos son los últimos en cerrarse en el caso de una emergencia del servidor (como son esenciales, por diseño, el sistema trata de mantenerlos funcionando el mayor tiempo posible).

*Instancias web*

Cada vez que usamos nuestro navegador web para visitar un sitio en Internet, nos estamos conectando a un servidor web (un servicio capaz de mediar entre el usuario y el contenido almacenado, enviando información en formato legible para el software de navegación). Estos servidores web pueden manejar varias instancias, reproduciendo cada una un sitio web distinto, pero haciendo uso normalmente de una misma conexión a Internet.

En el caso de monitorización, no es importante monitorizar sólo el software del servidor web, sino también cada una de las instancias que corren sobre éste. La caída de una instancia puede indicarnos un problema en particular con un sitio, mientras que la caída de todas las instancias nos indicaría un problema mayor en el sistema o software relacionado.

Dado que en Internet la caída de un sitio web está asociada a una mala imagen, es necesario que las empresas consideren utilizar monitorización en las instancias de sus sitios, minimizando así el tiempo necesario para resolver el problema y volver el sitio a la normalidad. Caso contrario, quedan atados a la voluntad de algún visitante del sitio que reporte el problema.

*Instancias de datos*

Al igual que sucede con los servidores web y sus instancias, las bases de datos manejan instancias de datos. En este caso, cada instancia es una conexión a una base de datos.

La caída de una instancia de datos puede provocar que una aplicación deje de funcionar correctamente, que un sitio web muestre mensajes de error al no encontrar la información que debe mostrar, o hasta incluso causar impacto en otras instancias de datos dependientes de ésta.

La monitorización de las instancias de datos es una práctica muy común en entornos de producción, debido al alto impacto que puede tener sobre la actividad de la empresa.

*Clusters*

Karl Kooper (2005) define a un *cluster* como un sistema de tipo paralelo o distribuido, que consta de una serie de nodos interconectados y que es utilizado como un recurso unificado.

Los *clusters* son grupos de servidores enlazados, que pueden tener como finalidad la alta disponibilidad de un servicio (que siempre esté accesible), balanceo de carga (que el uso de una aplicación se reparta entre varios servidores para evitar congestión), o incluso para la realización de operaciones complejas que requieren menos tiempo gracias a la disponibilidad de más recursos.

Si bien un *cluster* puede ser visto como una aplicación en sí misma, ya que un software especial es el encargado de brindar estas funciones, la diferencia está en que éstos siempre trae monitorización propia incorporada (por ejemplo, para que siempre esté accesible un servicio, el *cluster* controla que siempre haya al menos un nodo funcionando), y la misma se realiza a un nivel superior, sirviendo como un complemento de control y alerta.

Por ende, el control de los *clusters* no es crítico para un producto de monitorización, pero si necesario para brindar una capa extra de seguridad sobre la disponibilidad del sistema.

*Hardware de red*

La mayoría de los dispositivos de red corporativos, tales como *routers*/*switches* (ambos encargados de gestionar redes y mover paquetes de datos entre estas), *firewalls* (dispositivo diseñado para controlar el acceso a una red de datos por medio de reglas), antivirus por hardware y *proxys* (permite la conexión entre dos servidores funcionando como intermediario), disponen de un puerto físico pensado para el mantenimiento y la auditoría de los mismos.

Pese a que estos dispositivos pueden ser monitorizados de forma manual, en redes complejas es muy común que los administradores prefieran tener un control centralizado y automatizado, que les permita no sólo saber cuando un dispositivo deja de funcionar, sino también cuando este se encuentra en una situación extraordinaria.

Las funciones disponibles y el software destinado a controlar estos dispositivos, variará acorde al fabricante de los mismos, por lo que los productos de monitorización suelen tener componentes adicionales (pagos) para controlar productos específicos ya sea por empresa o incluso por modelo. Lo que usualmente se puede controlar en cualquier dispositivo de estos es: disponibilidad, recursos básicos disponibles (CPU, memoria, uso de disco interno, tarjetas de red), y estado de las luces LED

(sirven para saber si existe una falla de hardware, o si el dispositivo está mal conectado, por ejemplo). (Schubert, 2008, cap. 4)

*Hardware de salida*

La monitorización de impresoras físicas suele darse en empresas que hacen uso extensivo de las mismas, ya sea para la impresión de reportes, como para la comunicación a clientes/socios.

Las impresoras pueden ser monitorizadas para determinar su disponibilidad, o bien con fines ecológicos (calcular consumo de luz o papel, emisiones de CO2) y económicos (conocer cuanto consume cada departamento, usuario, dispositivo por periodo). El fabricante de la impresora es el que determina que funciones están accesibles para la monitorización, y normalmente los modelos más básicos de impresora sólo pueden informar si la misma está accesible o no. (IBM, 2010; Nagios, 2009)

Por otro lado, ya no a nivel físico, también se pueden monitorizar impresoras virtuales, aunque en estos casos la monitorización se asemeja a la de un servicio, algo que ya hemos comentado anteriormente en este capítulo.

*Monitorización para análisis de rendimiento*

La monitorización orientada al rendimiento utiliza varios de los ítems antes descriptos en este capítulo, pero en vez de estar enfocada a solucionar problemas, hace énfasis en prevenirlos a futuro antes de que surjan.

En una situación donde buscamos determinar un problema, comparamos el estado actual con el ideal, y en base a las reglas establecidas, decidimos si está bien. Por otro lado, si pretendemos evitar un problema potencial (a futuro) de rendimiento, es necesario que obtengamos datos históricos de comportamiento para poder predecir lo que sucederá, haciendo necesario entonces el almacenamiento de datos obtenidos por los controles de rutina.

Una vez obtenidos estos datos, los mismos no deberán ser analizados en tiempo real junto con los chequeos de rutina, sino en un segundo plano (en lo posible, con un servidor dedicado sólo a esta función), y siempre usando la mayor cantidad de historial posible para poder conseguir la mejor predicción, ya que esto ayudará a decidir que acciones se tomarán.

El análisis de rendimiento puede resultar complejo, ya que el mismo depende de muchas variables propias del sistema, como ser la velocidad del procesador, memoria disponible,

controladores de red, tamaño y velocidad de los discos, como otros propios de lo que se ejecuta en el servidor, destacando el propio rendimiento de la aplicación (según el lenguaje en el que esté programada, por ejemplo), el tipo de carga de trabajo que tenga, las relaciones con otras aplicaciones y su forma de comunicación, y hasta aspectos como el propio uso que hace el usuario (Sandra K. Johnson, Gerrit Huizenga, & Badari Pulavarty, 2005, cap. 4).

Luego de realizar el análisis, será necesario tomar decisiones en base a la información obtenida, ya sea para reducir (cuando hay capacidad ociosa) o aumentar los recursos de un servidor (cuando se prevea un posible cuello de botella o falta de recursos a futuro).

## Capítulo III: Sistemas operativos

*Concepto*

Según Silberschatz (1998, pág. 3), "Un sistema operativo es un programa que actúa como intermediario entre un usuario y el hardware de una computadora. El propósito de un sistema operativo es proveer un entorno en donde un usuario pueda ejecutar programas. Por tanto, la principal meta de un sistema operativo es hacer que una computadora sea práctica de usar. Una meta secundaria es que el hardware sea utilizado en una forma eficiente".

Este concepto demuestra la importancia del sistema operativo para que un grupo de componentes de hardware sea útil para un usuario. En caso de no existir, el usuario estaría obligado a operar directamente sobre el hardware (algo que es posible incluso teniendo un sistema operativo, pero no estrictamente necesario), haciendo mucho más complejo e inútil el hecho de usar una computadora.

*Interacción con el software de monitorización*

Tal como se menciono en capítulos anteriores, la monitorización más básica se realiza sobre recursos del servidor, que al final de la cadena están relacionados con algún componente físico de hardware. Estos componentes son accedidos por medio del sistema operativo, gracias al uso de controladores (también conocidos como *drivers*) que son simplemente un conjunto de archivos con librerías y funciones que entienden el funcionamiento del hardware (es decir, cada componente de hardware posee al menos un controlador desarrollado, normalmente, por su fabricante).

El software de monitorización, por tanto, depende tanto de los controladores instalados como del sistema operativo en que se ejecuta. Por ejemplo, un disco rígido podría tener funciones avanzadas como SMART (Self Monitoring Analysis and Reporting Technology, tecnología para detectar errores y alertar al usuario antes que un disco sea ilegible), que sin el controlador adecuado, no pueden ser aprovechadas por el sistema. Esta limitante presenta un problema para el software de monitorización, que debe estar preparado para operar en distintos sistemas operativos, y asumir las limitaciones propias de los controladores que cada sistema tenga instalados.

Por otro lado, la inexistencia de un controlador provocaría que un componente de hardware no pueda ser entendido por el sistema operativo, y por ende, tampoco monitorizado. Aunque podemos considerar un caso más extremo, en donde un controlador mal diseñado no alerte sobre un problema (hasta la degradación total del componente de hardware), o genere alertas innecesarias sobre

situaciones inexistentes (insumiendo trabajo por parte del software de monitorización o bien por el administrador del sistema).

En resumen, el software de monitorización debe estar preparado para operar en un sistema operativo en particular, lo cual hace de vital importancia elegir una opción que tenga buena cuota de mercado, y buenas perspectivas a futuro.

A continuación estudiaremos tres de las alternativas más populares en el mercado de servidores actualmente.

### UNIX

Este sistema operativo, multiusuario y multitarea, nació en el año 1969 en los laboratorios de Bell (AT&T), y hoy en día sirve como cimiento para varios productos, entre ellos BSD, Solaris, AIX y HP-UX. El producto original dejó de ser liberado a público luego de su versión 7, la cual fue utilizada como base para crear los productos antes mencionados. El término UNIX es asociado a sistemas operativos de servidores, donde tiene una buena cuota de mercado, aunque también existen variantes para máquinas de escritorio, como es el caso de Mac OS.

### Windows

El producto insignia de Microsoft nace en el año 1981 y actualmente goza de una buena cuota de mercado en las computadoras de escritorio. En cuanto al mercado de servidores, si bien tiene presencia, no es tanto su peso debido a la existencia de varios competidores importantes. Una de sus ventajas principales es la facilidad de desarrollar aplicaciones para él, como también el uso casi exclusivo de x86. Esto último también les juega en contra, ya que limita el mercado potencial al cual puede ofrecerle el producto. En el año 2012, y ante la creciente popularidad de los procesadores ARM, Microsoft anunció inversiones varias para darles soporte en Windows Server.

### GNU/Linux

La primera versión del sistema operativo (SO) Linux apareció en el año 1991, gracias a su creador Linus Torvalds, quien inicialmente buscaba crear una variante de UNIX para la arquitectura del PC IBM (Intel 80386). Este SO tuvo éxito, en gran parte por la posterior incorporación de paquetes libres de la Free Software Foundation (FSF), como también por la naturaleza de su licencia (Stallings, 2005, pp. 95–96). Hoy en día, en la comunidad de código abierto se hace

referencia al sistema como GNU/Linux, ya que Linux sólo corresponde al núcleo del sistema. Comercialmente se lo suele promocionar simplemente como Linux, por lo que en todo este trabajo se lo mencionará de este modo ya que es el término más familiar y reconocido por el usuario común.

Si bien podríamos considerar a Linux como un hijo de UNIX, el mismo ha conseguido diferenciarse en muchos aspectos, que estudiaremos en el próximo capítulo. Linux ha sido un sistema operativo que ha ganado adeptos desde sus inicios, y hoy en día se encuentra quitando cuota de mercado a las demás alternativas, debido tanto a su coste como a su capacidad de operar en distintas arquitecturas.

## Capítulo IV: ¿Por qué Linux?

Al ser de código abierto, el *kernel* de Linux puede ser modificado por cualquiera y adaptado a otras arquitecturas (actualmente presente en más de 20, incluyendo x86, PowerPC, System Z, SPARC, ARM, entre otras), e incluido para distribuir de forma gratuita o comercial, siempre que se cumpla con las condiciones de la licencia GPL. (Gillen, 2009, p. 2; Johnson, 2005, p. 58)

El menor costo de Linux, en comparación con otros sistemas operativos como Windows y las variantes de UNIX, le permitió inicialmente entrar en las empresas por medio de proyectos de reciclado de máquinas y actualización de versiones ya no soportadas de los productos que le hacían competencia. Al soportar un gran abanico de aplicaciones de su competencia directa, UNIX, y ofrecer una estabilidad similar, pudo mejorar notablemente su cuota de mercado, llegando hoy a estar presente en entornos de producción de grandes empresas, con un crecimiento notable año tras año. (Gillen, Stergiades, & Waldman, 2008, pp. 1–6)

Un aspecto no menor de este sistema operativo, es el hecho de no provocar el famoso efecto del *vendor lock-in*, en donde una empresa monopoliza el producto y sus clientes son cautivos de la misma, ya que el hecho de querer cambiar a un producto de la competencia le provocaría un alto coste, o simplemente por no existir la posibilidad de una migración sencilla. Al ser un producto desarrollado por muchas empresas (como patrocinadoras), cada parte involucrada intenta hacer de Linux un sistema altamente compatible (como en el caso de las arquitecturas mencionadas al comienzo), y de fácil acceso para que cualquiera pueda adaptarlo a sus necesidades, rompiendo este temido efecto que afecta a tantas empresas hasta el momento.

### *Actualidad*

Hoy en día Linux está presente en todo tipo de entornos, desde desarrollo hasta producción, donde se ejecutan servidores de aplicación, bases de datos, aplicaciones críticas y en tiempo real, por lo que se puede decir que ya es lo suficientemente maduro como para dar soporte a aplicaciones de cualquier tipo de empresa (desde pequeñas, hasta multinacionales). (Gillen, 2009, p. 1,5)

A modo de ejemplo, durante el año 2010, Linux consiguió un hito importante al sumar la Bolsa de Valores de Londres (LSE, London Stock Exchange) como uno de sus grandes usuarios a nivel mundial. Gracias a este cambio, la LSE no sólo redujo sus costos operativos, sino que también consiguió romper el récord en velocidad de transacciones.

Por otro lado, el ranking TOP500 (Top500, 2011), que recolecta información de las primeras quinientas computadoras públicas con mayor rendimiento a nivel mundial, ubicaba en su edición 2011 a Linux en la cima con el 91,4% (457 de 500) en cuanto a cantidad, cubriendo incluso las diez primeras posiciones sin otro sistema operativo que le compita. En la actualización de Junio del 2012, Linux pasa a ocupar el 92,4% (462 de 500) del total (Top500, 2012), demostrando así la importancia de Linux tanto a nivel industrial, académico y de investigación, ya sea por su bajo costo para crear supercomputadoras, como también a su alto rendimiento cuando está bien administrado y configurado.

## Capítulo V: Proceso y tipos de monitorización

*¿Cómo se monitoriza un servidor?*

Para poder monitorizar un servidor, es necesario primero decidir que tipo de monitorización se desea realizar sobre éste:

- Con agente
- Sin agente
- Híbrida

La diferencia entre las 3 formas de monitorización, radica en la presencia de un agente instalado en los servidores que se desean monitorizar. Esto repercute en los datos que se podrán capturar y, por ende, en el posterior análisis. Por otro lado, todas las formas de monitorización presentan como factor común la presencia de al menos un servidor central, el cual se encarga de la recolección de datos. (eMachines, 2009)

Para poder comprender como es el proceso de monitorización, necesitaremos primero entender que es un agente, y luego detallar cómo funcionaría en cada uno de los modos anteriormente mencionados.

*¿Qué es un agente?*

Un agente es un paquete de *software* que tiene como finalidad realizar una tarea específica en relación con un recurso informático (una aplicación, un servicio de sistema, *hardware*, etc.). Se los configura para recolectar información específica y relevante acorde al objetivo que tienen.

Se los conoce normalmente como “agente autónomo” o “agente inteligente”, debido a que su tarea es realizada de forma automática, sin intervención de otra aplicación o de un usuario.

Estos agentes, suelen venir como adicionales dentro de los instaladores de productos de monitorización, o pueden ser adquiridos pagando una licencia aparte en el caso de productos muy específicos como bases de datos, aplicaciones críticas, entre otros.

Los agentes pueden ser de tipo individual o “multi-agente”, en donde suelen ser configurados para cooperar con otros agentes instalados en el mismo sistema. (James Sayles, 2009, p. 1,2)

*Monitorización basada en agentes*

La monitorización basada en agentes requiere inicialmente la instalación del *software* agente en cada uno de los servidores objetivo. Este *software* debe tener una versión disponible para el sistema operativo que queremos utilizar, lo cual puede ser una limitante a la hora de elegir este tipo de productos. A su vez, el agregar un agente de monitorización implica también tener en cuenta los requerimientos de memoria, procesador y de espacio libre en disco para que el mismo pueda operar correctamente, por lo que primero es necesario determinar los requerimientos y planificar su instalación.

Una vez instalado el agente, es necesario darlo de alta en el servidor central (asumiendo que ya contamos con uno), para que el servidor objetivo sepa a quien enviar los datos recolectados en su entorno.

Si el agente tiene un objetivo que no está definido por defecto, es necesario realizar una configuración manual del mismo. En el caso que el objetivo si esté definido por defecto, entonces lo único que hay que hacer es activar el mismo para la monitorización.

Por último, es necesario configurar un depósito de datos, en donde se recolectarán todos los datos recibidos por los agentes de los distintos servidores monitorizados. (Vasfi Gucer et al., 2005, cap. 3)

*Monitorización sin agentes*

Al no necesitar la instalación de un agente, este tipo de monitorización requiere menos planificación en cuanto a requerimientos en cada servidor. Aún así, es importante tener en cuenta, ante la presencia de un cortafuegos, que algunos puertos deberán ser habilitados para que pueda haber una comunicación fluida entre los servidores monitorizados y el nodo que recolecta los datos.

En muchos casos, se hace uso de protocolos como SNMP (Simple Network Management Protocol), WMI (Windows Management Instrumentation, para administración de datos y políticas en Windows), CIFS (Common Internet File System, antes llamado Server Message Block, para compartir archivos e impresoras), SSH (Secure Shell, shell segura de acceso multiplataforma), entre otros, para obtener información del sistema. Protocolos como FTP (File Transfer Protocol, para la transferencia de archivos), y otros de correo como IMAP (Internet Message Access Protocol), POP (Post Office Protocol), SMTP (Simple Mail Transfer Protocol), también pueden ser usados para obtener información, aunque en estos casos limitando la salida a la función del servicio que brindan.

Las reglas de monitorización se configuran una vez agregados los nodos en el servidor central, sea que este agregado fuese manual o automático por el software. (Alexei Vladishev & Eugeny Grigorjev, 2008, cap. 2).

*Monitorización híbrida*

La monitorización híbrida busca solucionar los problemas de depender de un agente, y ofrece ambas soluciones en el mismo paquete.

La ventaja de este modo es que permite obtener todo tipo de información posible, al combinar las fuentes de recolección de la monitorización por agente como la de sin agentes, pudiendo tener agentes más livianos al no tener que recolectar datos que pueden provenir de servicios ya disponibles de forma remota (BMC, 2006, pp. 2–4).

Este tipo de monitorización es el más completo, aunque puede resultar más complejo en su instalación, teniendo que asumir las desventajas de la monitorización con agente (instalación y requerimientos para el funcionamiento del software), y la de sin agentes (aspectos de seguridad para la recolección de datos) en el caso que se requieran ambos para un mismo servidor objetivo.

## Capítulo VI: Protocolo simple de administración de red

*¿Qué es SNMP?*

El *Simple Network Management Protocol* (en español, Protocolo simple de administración de red) es un protocolo que opera en la capa de aplicación (capa 7) del modelo OSI, y que nos permite administrar dispositivos conectados en una red por medio de un conjunto sencillo de operaciones que pueden ser realizadas de forma remota.

El protocolo nace en el año 1988, debido a la necesidad creciente de facilitar la administración de dispositivos en una red, enfocado inicialmente en *routers*, pero luego ampliando su cobertura a todo tipo de dispositivos con un IP asignado, como ser servidores Unix/Windows, módems, impresoras, servidores NAS y hasta ordenadores de escritorio (Mauro Douglas & Kevin Schmidt, 2005, cap. 1).

Hasta el momento, existen tres versiones de SNMP publicadas en la IETF (Internet Engineering Task Force, grupo encargado de revisar y aprobar especificaciones que se convierten en estándar para Internet), teniendo como último estándar la versión SNMP v3 (STD0062) definida por varios RFC (3411 a 3418). Las versiones 1 y 2 han sido catalogadas como históricas por la IETF, aunque podemos encontrar dispositivos que operen sólo con estas versiones en el mercado, debido a la antigüedad de los mismos.

Los cambios más importantes en la última versión están enfocados principalmente en la seguridad, tanto a la hora de establecer una conexión (anteriormente, la clave y datos se enviaban en texto plano, ahora todo lo transmitido es cifrado), como también para mantener la integridad de los datos, a fines de evitar que un tercero pueda modificarlos utilizando una técnica de intermediario en la conexión. Por otro lado, se introducen cambios a nivel de conceptos y arquitectura del protocolo, pasando a tener "entidades SNMP" en vez de administradores y agentes, lo cual hace más sencillo entender el protocolo en general, y permite establecer una base para futuras mejoras del mismo (Mauro Douglas & Kevin Schmidt, 2005, cap. 3).

¿Cómo funciona?

En la primera versión de este protocolo, existían los dispositivos administrados, los agentes y los administradores (luego pasaron a ser llamados "entidades SNMP"). Los primeros eran los dispositivos que se monitorizaban (dicha tarea es realizada gracias a los agentes que estos poseen),

mientras que los últimos eran quienes gestionaban la monitorización (también conocidos como NMS, Network Management Stations). En este contexto, los administradores son los encargados de realizar *pooling* (sondeo) sobre los dispositivos administrados (enviar consultas de información), como también de gestionar las *traps* (eventos que se envían desde los agentes, informando un cambio de estado) y actuar en consecuencia. Del lado de los agentes, estos vienen instalados por defecto en la mayoría de los dispositivos IP, incluyendo reglas previamente establecidas por los fabricantes para alertar ante problemas, facilitando notablemente la tarea de configuración y administración de la red en general (Mauro Douglas & Kevin Schmidt, 2005, cap. 1).

Es necesario también mencionar los conceptos de *Structure of Management Information* (SMI), que provee una definición de lo que se administra y sus posibles estados, como también del *Management Information Base* (MIB), que se utiliza como una base de datos de los objetos administrados por cada agente, utilizando la sintaxis propia de SMI para guardar la información. Todo esta información es conocida por los NMS, lo que les permite simplificar notablemente la recolección de datos (ya que estarían, en cierto modo, estandarizados), y a su vez la comparación entre distintos dispositivos de funcionalidades similares.

*¿Por qué no usar sólo SNMP?*

Tal como se mencionó en el capítulo anterior, el protocolo SNMP es el que nos viene a la mente cuando hablamos de monitorización sin agentes (pese a que usa un agente). Esto se debe un gran número de fabricantes hacen sus dispositivos compatibles con este protocolo estándar (principalmente usando SNMP v1 y v3), lo cual le da peso en el mercado, a la vez de conseguir una mayor base de productos disponibles que funcionan con él ni bien instalados, sin mayores complicaciones.

Pese a que este protocolo es reconocido, presenta limitaciones en cuanto a lo que puede monitorizar por defecto, cómo lo hace y también a la dificultad misma de hacer uso de éste en entornos gigantes (donde normalmente sería más útil). Estas dificultades están mayormente asociadas a la planificación de su uso, y no al mantenimiento en sí (que suele ser mínimo en relación con la monitorización basada en agentes). En general, todo esto provoca que los administradores decidan evitar su uso, limitarlo sólo a funciones básicas de estado (accesible/no accesible) de los dispositivos, o bien para controles más globales, como en el caso de controles sobre la congestión de redes.

En los casos que se requiera realizar un control específico que no es cubierto por SNMP, deberemos recurrir a la instalación de agentes extensibles (los cuales requieren de conocimientos de programación), pasando así a una monitorización híbrida, lo cual nos indica que este protocolo por sí mismo puede no ser suficiente para un entorno de producción, debido a los constantes cambios (Mauro Douglas & Kevin Schmidt, 2005, col. 11).

*SNMP vs SNMP Traps*

SNMP opera por defecto como una comunicación de dos vías. El nodo servidor le envía al nodo cliente una petición de información, para lo que luego el nodo cliente le responda con los datos solicitados. Estas peticiones del nodo servidor suelen realizarse en periodos de tiempo predeterminados según el tamaño de la red, para evitar así cuellos de botella debido a la cantidad de mensajes. A medida que la red va creciendo, los periodos pueden ser demasiado largos para detectar un problema a tiempo, por lo que en su momento se tuvo que pensar en una alternativa.

SNMP Traps (o SNMP con trampas) surge como la solución a los problemas de escalabilidad en redes crecientes de gran tamaño, y simplifica el proceso teniendo al nodo cliente como responsable de notificar cuando un evento ocurre. Entre las ventajas que ofrecen las trampas, tenemos la posibilidad de notificar inmediatamente un problema ni bien sucede, como también reducir la cantidad de mensajes periódicos en una red, disminuyendo así la carga total sobre el nodo servidor en el caso que se usen trampas de forma exclusiva.

Dejando de lado las ventajas y desventajas de cada opción, es necesario que ambas coexistan en un entorno monitorizado, lo que limita algunos de los beneficios de usar trampas (ya que el consumo de tráfico se mantendría), aunque permite que se pueda obtener un estado de red constantemente actualizado con alertas prioritarias en el caso que alguna trampa se active. Los productos más complejos de monitorización hacen uso de ambas opciones para brindar la mejor cobertura posible.

## Capítulo VII: Tipos de arquitectura

Al momento de programar una aplicación que hace uso de los recursos de red, es necesario decidir el tipo de arquitectura que se utilizará para la misma, a fines de garantizar las comunicaciones.

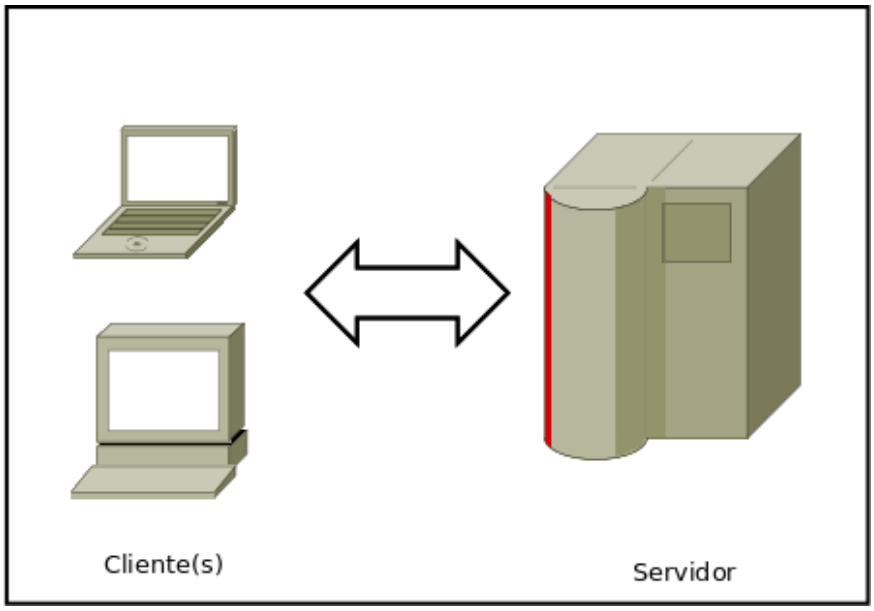
La arquitectura elegida influirá en aspectos importantes de un sistema, como ser los puntos críticos (¿qué pasa si un nodo desaparece?), los recursos necesarios (¿cuánto tráfico de red se generaría?), y hasta las posibilidades de crecimiento a futuro, también conocido como escalabilidad (¿podrá un sólo servidor manejar el doble de tráfico en el futuro?).

A continuación se detallan algunas arquitecturas utilizadas, junto con sus ventajas y desventajas.

### *Arquitectura cliente-servidor*

Esta arquitectura, también conocida como de 2 capas, es la que surge inicialmente con la aparición de las comunicaciones en red y reemplaza el uso de la arquitectura monolítica (de 1 capa), altamente crítica, en donde todo el sistema se concentraba en un mismo nodo.

En la arquitectura cliente-servidor existen dos partes, siendo la primera encargada de contener la aplicación (o interfaz gráfica) que es utilizada por el usuario, mientras que la segunda se encarga de gestionar el almacenamiento en la base de datos, y es la que cuenta con la mayor cantidad de recursos, dado que puede ser consultado por uno o más clientes.

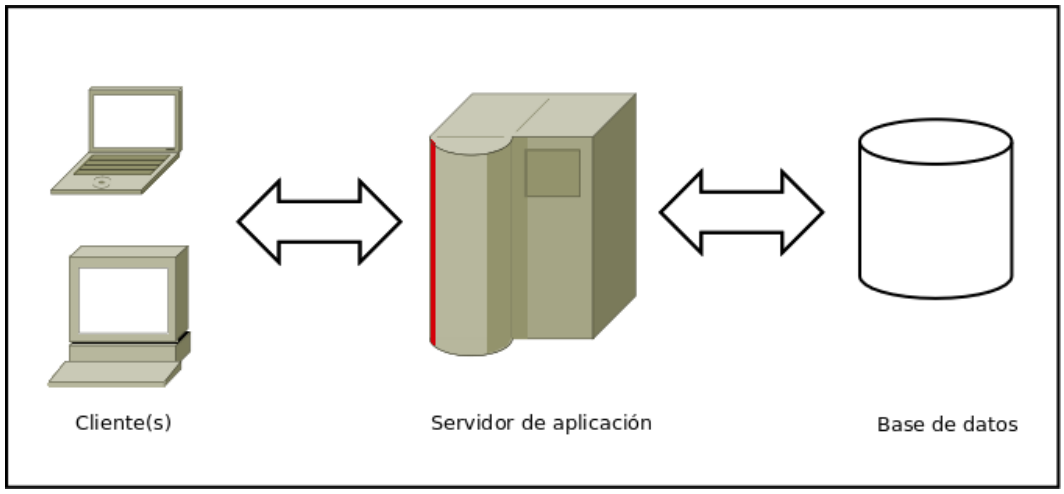


Según Rüdiger (2001), “un servidor es el punto central de la red, ya que provee servicios y contenido, mientras que el cliente sólo solicita contenido o pide la ejecución de servicios, pero sin compartir recursos.”

*Arquitectura multi-capa*

En este tipo de arquitectura, la aplicación funciona de modo similar al de cliente-servidor, con la diferencia que el último componente es distribuido entre dos o más partes. El componente intermedio pasa a llamarse servidor de aplicación y es el encargado de mediar entre ambos extremos (usualmente, contra una base de datos), separando así las funciones, y permitiendo el balanceo de carga.

Uno de los beneficios de esta arquitectura es la posibilidad de reemplazar los componentes de la misma (ya que pueden existir dos o más servidores de aplicación o bases de datos), sin necesidad de afectar a las demás parte del sistema, obteniendo mejores resultados en cuanto a la disponibilidad.

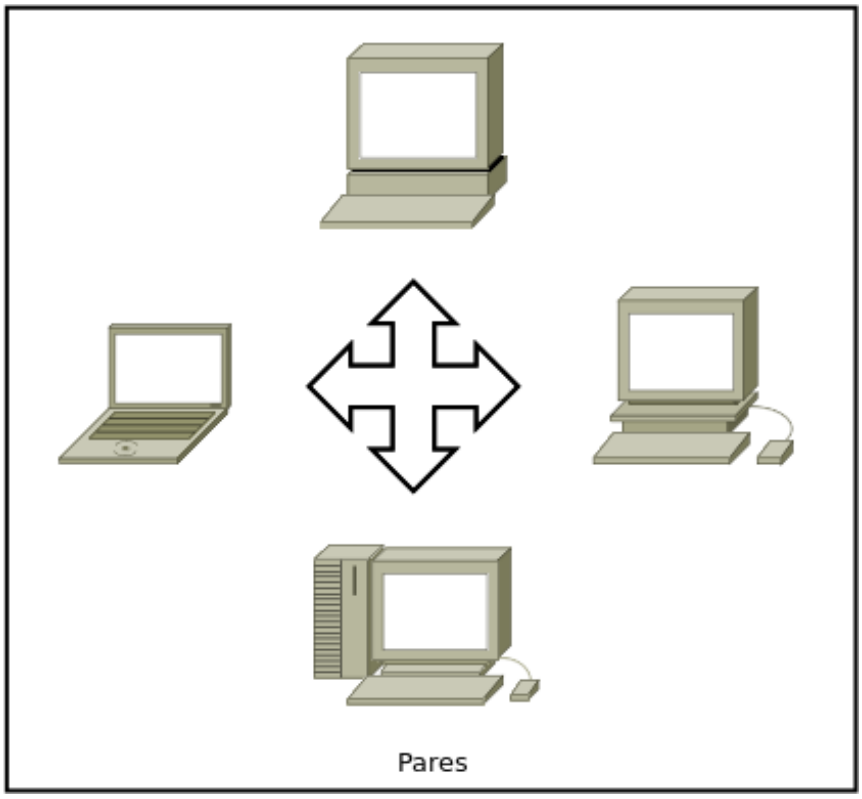



*Arquitectura descentralizada*

Al descentralizar una aplicación, lo que se consigue principalmente es eliminar puntos críticos de fallo, obteniendo mayor redundancia que en las arquitecturas multi-capa y de dos capas.

La idea de la descentralización, es que ningún “nodo” sea más importante que otro (al menos, inicialmente), y que todos compartan por igual sus recursos para garantizar la disponibilidad de la red. En cierto modo, cada participante de una red descentralizada, cumple el rol de cliente y servidor al mismo tiempo, por lo que el software debe contener ambos tipos de funciones.

En una arquitectura de este tipo, pueden ingresar y salir nodos en cualquier momento, provocando cambios de estado en el panorama general de la red (debido al cambio en los recursos disponibles), pero no afectando así al servicio mismo que la red provea.




En el próximo capítulo se estudiará más en detalle las redes de pares, conocidas como P2P, que han provocado uno de los mayores cambios en el mundo de las comunicaciones, al permitir la descentralización en la distribución de contenidos, logrando incluso solucionar problemas presentes en otras arquitecturas, gracias a que la migración hacia éstas se puede realizar de forma transparente e incluso operar en forma paralela.

## Capítulo VIII: Tecnología P2P

*¿Qué es el P2P?*

Según la RFC 5694 (Camarillo, 2009), "un sistema es considerado P2P si los elementos que lo forman comparten sus recursos para proveer el servicio que [el sistema] fue diseñado para brindar. Los elementos proveen servicios al mismo tiempo que solicitan de otros [elementos]."

Por tanto, podríamos decir que cada elemento de un sistema P2P actúa como un par, y ninguno de los mismos destaca en jerarquía ni responsabilidades.

Aún así, existen sistemas P2P cuyos elementos actúan sólo como clientes o servidores (haciendo analogía de la arquitectura cliente-servidor). En estos casos, si estos elementos son la minoría, el sistema puede seguir considerándose P2P, caso contrario, pasarían a formar parte de otro tipo de arquitectura, ya que no cumplirían con el principio básico de que todos comparten por igual. Estas diferencias las veremos más adelante en este capítulo, cuando analicemos los distintos tipos de P2P.

*Historia del P2P*

El concepto de P2P no es nuevo, pese a que ha visto su mayor auge luego del año 2000, ya que existe desde los tiempos de ARPANET (1969), en donde los nodos participantes de estas redes eran llamados "pares", y luego en 1972 con el surgimiento de los *MTA* (Message Transfer Agents), que hacían tanto el papel de servidores cuando un usuario enviaba un mensaje, como también de cliente al conectarse a otros MTA para transmitir el contenido del mensaje. Luego surgirían otros ejemplos, como UseNet (1979) que permitía el envío de mensaje entre participantes independientemente de Internet, y hasta los inicios del conocido sistema de DNS (1984), que hacía de cliente y servidor al mismo tiempo con otros servidores DNS para poder armar una mapa de Internet (Clarke, 2004).

El hito comercial más importante del P2P fue con la salida de *Napster*, que por el año 1998 (hasta el 2001), supo crear altos volúmenes de tráfico en Internet debido a la posibilidad de compartir contenido entre pares, sin necesidad de intermediarios. Luego de este producto, eliminado del mercado debido a cuestiones legales, nacieron alternativas como *gnutella*, *eDonkey*, y *Bit Torrent* en el mismo segmento, pero ya pensando en una nueva generación de este tipo de redes (Koegel Buford, 2009, cap. 1).

Otras aplicaciones supieron comenzar a explotar los beneficios de las redes de pares para realizar

llamados por Internet (VoP2P, Skype), ver películas por *streaming* (P2PTV) y hasta proyectos como SETI@Home, para poder compartir los recursos de hardware de nuestro ordenador en beneficio de la ciencia.

*Ejemplos de P2P*

Si analizamos la tecnología detrás de BitTorrent, uno de los protocolos más usados en el intercambio de archivos, nos encontramos con una aplicación del tipo P2P que posee varias características interesantes.

Un usuario descarga un software que soporte el protocolo, y al conectarse se une a un *swarm* (enjambre en inglés). En esta red de nodos, cada participante puede descargar como también compartir archivos. Cada archivo presente en la red está dividido en pequeñas piezas, las cuales sirven para aliviar la carga de intercambio entre los participantes. Si un nodo posee el total de las piezas y sólo comparte (sube datos), recibe el nombre de *seeder* (sembrador en inglés), mientras que los nodos que siguen descargando reciben el nombre de *leecher* (sanguijuela en inglés) (Camarillo, 2009, p. 6).

Este protocolo ha demostrado ser lo suficientemente robusto como para ser utilizado por una gran cantidad de sitios sin casi inversión alguna (cada participante de la red aporta sus propios recursos). A su vez, algunas empresas han sabido aprovechar las bondades del BitTorrent para reducir costos de infraestructura, como ser el caso del creador de juegos Blizzard, que con su herramienta “Blizzard Downloader” hicieron uso de este protocolo para que sus clientes puedan descargar juegos u obtener actualizaciones periódicas, eliminando así el costo que tendría en el caso de usar tecnología FTP (del tipo Cliente-servidor).

*Tipos de P2P*

Los sistemas P2P suelen ser clasificados de muchas formas, dado que se los puede analizar desde distintos aspectos. Si optamos por darle peso a la presencia de un nodo más importante que el resto, podemos hablar de un tipo centralizado, descentralizado o híbrido. Un ejemplo de esto, sería el software anteriormente mencionado Napster, que básicamente era centralizado, ya que sin la presencia del servidor de la empresa, no era posible unirse a la red (Ghosemajumder, 2002, pp. 17–18). En el caso de un tipo híbrido, hablaríamos de una red en donde existe un servidor más importante, pero que ante la caída del mismo, la red puede seguir funcionando sin problemas (o con funciones reducidas, como por ejemplo, sin la posibilidad de que se realicen búsquedas).

En cambio, si optamos por verlo desde el punto de vista del índice de datos de la red, entonces hablaríamos de un tipo local, centralizado o distribuido. En el caso de un tipo local, implicaría que cada nodo es egoísta y guarda información que sólo le concierne a él, reduciendo al mínimo la potencia de la tecnología P2P. Si hablamos de un tipo centralizado, un servidor hace de punto referente, y ante cualquier consulta, es el encargado de responder donde están los datos. Finalmente, en el caso de distribuido, podemos considerar la opción de *Distributed Hash Tables* (DHT), utilizado por BitTorrent, que permite la presencia de datos de la red en distintos nodos, acelerando así los resultados ante consultas de estado, como también disminuyendo la dependencia con determinados nodos.

Mezclando ambas vistas, tenemos una tercera categoría en donde podemos decir que hay redes P2P con estructura, sin estructura e híbridas. Las sin estructura se dividen en deterministas (Napster, BitTorrent) y no deterministas (KaZaa, gnutella), teniendo como diferencia principal que las primeras tienen un nodo central que almacena los índices de contenido de la red, mientras que en la segunda se consulta en cada nodo ante cualquier búsqueda. En el caso de las redes con estructura, la diferencia se da por el uso de DHT, teniendo un índice distribuido para aquellas que si usan tablas de *hash*, por lo que cada nodo guarda pequeñas cantidades de información que sirven a un fin general. Finalmente, las redes de tipo híbrida según esta visión, sacan provecho de las ventajas anteriormente mencionadas por los otros dos tipos, permitiendo ya sea consultas complejas en menor tiempo (Structella) o bien el descubrimiento de nodos en menores intervalos (Kelips) (Rajiv Ranjan, Aaron Harwood, & Rajkumar Buyya, 2007, pp. 11–14).

*Funciones de un sistema P2P*

Según la RFC 5694, toda red P2P contiene ciertas funciones que son comunes en general, como también otras que varían acorde a la naturaleza de la actividad de la red.

Entre las funciones comunes, nos encontramos con la identificación de los nodos, que sería el primer paso necesario en el cual un nuevo participante se presenta. Luego, existe una autorización para el acceso a la red, en donde básicamente los otros nodos reconocen al nuevo participante como miembro de la misma. Finalmente, el nuevo nodo debe salir a conocer a sus otros compañeros, por lo que existe una tercera función de descubrimiento de nodos. Todas estas funciones de inicio, pueden ser realizadas contra un servidor central que recibe el nombre de *bootstrap* (servidor de arranque) (Camarillo, 2009, pp. 6–9).

Una vez que el nodo se encuentra en la red, y dependiendo de la actividad de la misma, nos podemos encontrar con otras funciones como ser el indexado de datos y de archivos (en una red como BitTorrent), el cálculo de resultados (como Seti@Home), y el transporte de mensajes entre nodos (presente en la mayoría de las redes P2P).

Algunas de estas funciones serán analizadas en más detalle posteriormente en este capítulo.

*Seguridad y Vulnerabilidades*

Las redes P2P son más exitosas cuando son de acceso libre, ya que permiten ingresar a cualquier nodo, incrementando así la cantidad de recursos disponibles como también el tamaño de la red misma. Esta característica implica un problema de seguridad, ya que cada nodo que ingresa a la red debe ser lo suficientemente confiable, o de lo contrario, la red podría sufrir un ataque.

Entre los problemas de seguridad más comunes de las redes P2P, nos encontramos con los ataques de *DDOS* (denegación de servicio distribuida), *MitM* (Man in the Middle), gusanos de red, entre otros (Engle & Javed, 2006). A continuación analizaremos cada uno de estos ataques, como también posibles soluciones que permiten reducir o anular el impacto.

Los ataques de denegación de servicio distribuida normalmente tienen como objetivo atacar los servicios de una red, haciendo que estos queden inaccesibles. En las redes P2P, el ataque más común de este tipo se enfoca en inundarlas con paquetes inválidos, a fines de evitar que los nodos participantes se puedan comunicar efectivamente. Una de las formas más sencillas de evitar este problema se denomina *pricing* y consiste en imponer un desafío complejo a cada nodo que se suma

a la red, haciendo que cada ingreso tenga un costo de computación, lo cual retrasaría y desmotivaría a un atacante. Por otro lado, se puede agregar la necesidad de renovar las credenciales temporalmente, evitando que un atacante pueda permanecer mucho tiempo en la red sumando voluntarios, lo cual aumentaría la complejidad del ataque.

El ataque de intermediario, o *MitM*, consiste en ubicar un nodo en la red que haga las veces de mediador para la interacción entre dos nodos, pudiendo falsificar o confundir a los participantes. Este tipo de ataque es altamente probable en redes P2P, debido a la naturaleza de las mismas. Una solución efectiva para este tipo de problemas es autenticar cada nodo contra un nodo central de autoridad, o bien contar con firmas digitales que sean utilizadas para que los nodos se comuniquen, evitando así que un tercero pueda interferir en la comunicación.

Una forma de ataque más sofisticado sería con el uso de gusanos (*worms*), los cuales deben ser programados pensando directamente en los clientes P2P. En este caso, si una red sólo cuenta con un único paquete de *software* para acceder, es muy fácil para un atacante conseguir la totalidad del control, ya que ante una vulnerabilidad, todos los participantes pueden ser víctimas. Es obvio entonces, que la heterogeneidad de *software* permitiría reducir este tipo de ataque. A su vez, otra recomendación que se suele dar, es el uso de lenguajes de *tipado fuerte*, es decir que no permiten violaciones de los tipos de datos, evitando así el aprovechamiento de vulnerabilidades básicas como la sobrecarga de *buffer*.

Entre los ataques más complejos, nos encontramos con el racional, que apunta a disminuir la cantidad de recursos de los nodos participantes. Esto se puede dar ya sea porque la configuración de un nodo es egoísta, también conocido como *Free Riding (Karl Kooper, 2005, p. 1)*, o bien porque un tercero está impidiendo que todos los recursos del mismo sean accesibles, lo cual hace complejo detectar la causa. BitTorrent, uno de los casos más interesantes a nivel P2P, soluciona este problema al poner como regla que quien más comparte, más puede descargar, evitando así el accionar egoísta de sus participantes. En el caso que el ataque provenga de un tercero, el nodo afectado notará la reducción de confianza en sus pares, provocando algún tipo de alerta para consigo mismo.

Finalmente, los ataques de tipo Sybil y Eclipse, buscan desmembrar la red ya sea logrando volumen, o separando la misma en varias particiones respectivamente. En cada caso, el atacante buscará hacerse con el control de los mensajes entre participantes, pudiendo ejecutar ataque de intermediario o denegación de servicio. La solución, nuevamente, pasa por hacer complejo el ingreso de nuevos nodos a la red, como también de tener algún tipo de identificación que evite o

retrase el acceso a un atacante.

*Free Riding y Whitewashing*

Tal como se mencionaba en el ataque racional, el *Free Riding* es un problema que surge cuando un participante decide ser egoísta con sus pares, provocando así una diferencia entre los recursos que comparte y los que utiliza. Esta técnica es ampliamente conocida por los usuarios de servicios P2P de intercambio de archivos, ya que al no tener conexiones rápidas o ser penalizados por sus proveedores de Internet, deciden limitar el ancho de banda compartido.

A esta técnica se le suma el *White-Washing*, que busca eliminar las consecuencias de las penalizaciones por *Free Riding*, al salir y entrar nuevamente en una red haciéndose pasar por otro nodo nuevo (Feldman, Papadimitriou, Chuang, & Stoica, 2004, p. 4).

Las consecuencias a corto plazo de tener participantes que apliquen estas técnicas, son la reducción de recursos en la red, como también una merma en la actividad. Si miramos en el mediano y largo plazo, esto podría lograr incluso que una red deje de ser interesante para los nodos que no son egoístas, provocando que estos dejen de compartir o simplemente se desconecten. Como una red P2P necesita de los recursos de sus participantes para subsistir, una consecuencia como la antes descripta sería ideal para que la misma deje de existir o pierda relevancia.

La adopción de un sistema de identificación único para cada nodo en la red, disminuye los problemas del *Free Riding* como también hacen imposible el *White-Washing* (Feldman et al., 2004, p. 6).

*Inscripción e identificación*

A lo largo de este capítulo se ha resaltado una y otra vez la necesidad de la identificación de cada uno de los nodos participantes en la red, ya sea por seguridad como también para evitar técnicas egoístas de consumo de recursos. Esto plantea como requisito un registro único y centralizado, o ampliamente conocido por todos los nodos, que sirva como base para que cada nodo pueda autenticar en su ingreso a la red. Por ende, cada nodo nuevo deberá no sólo inscribirse a una red (en su primer acceso), sino también autenticar y obtener autorización para operar en cada conexión.

Todos estos controles no son requerimientos indispensables para lograr una red de P2P, pero si necesarios en el caso que busquemos integridad, disponibilidad y seguridad.

*Descubrimiento de nodos*

*Bootstrapping* es el nombre que recibe el proceso por el cual un nodo nuevo pasa para poder unirse a una red P2P. Este proceso puede ser aplicado con distintos métodos, que describiremos más adelante, y requiere de al menos un nodo que reciba al nuevo, para poder intercambiar información sobre el mapa de la red. Este nodo receptor, recibe el nombre de *par rendevouz*, y debe poder ser capaz de unir al nuevo nodo como un par más, o bien rechazar la petición a la vez de proveerle una lista de otros posibles pares. Sea cual fuere el método de *bootstrapping* seleccionado, este debe ser justo para que todos los nodos de la red se repartan la carga, y ninguno sea saturado por peticiones de nuevos pares (las cuales pasarían a ser rechazadas, demorando el tiempo de ingreso a la red).

Las alternativas incluyen el uso de un servidor de *bootstrapping*, descubrimiento por *broadcast* o *multicast*, uso de un *overlay* universal, o por medio de *caché* de pares conocidos de sesiones pasadas. Cada alternativa tiene sus defectos, por lo que el uso de dos métodos no se desaconseja. En el primer caso, la existencia de un servidor único para aceptar nuevos nodos limita el tamaño de la red, como también introduce un punto de fallo único. Luego, el descubrimiento por medio de la difusión (*broadcast* o *multicast*) es limitado en cuanto al alcance (la misma subred) y a la dependencia de tecnologías nuevas (*multicast* no es soportado por defecto). Finalmente, el uso de una capa universal, eleva la complejidad de la red, mientras que el método que utiliza la caché sólo puede usado para nodos que se unen por segunda vez a una red, ya que en su primera conexión no contarían con datos históricos almacenados (Koegel Buford, 2009, cap. 6).

*Requerimientos para un P2P*

A diferencia de los requerimientos en cualquier aplicación de tipo cliente-servidor, las aplicaciones P2P reparten este costo entre todos sus pares, minimizando los requisitos para montar una de estas redes, ya que cada nodo que se une a la misma comparte sus recursos (CPU, memoria, disco, y lo más importante, ancho de banda), por lo que mientras más crece, mayor es su capacidad. Aún así, se debe aclarar que el consumo de red de estas aplicaciones suele ser mayor para cada par, ya que no sólo actúan como cliente, sino también como servidor, por lo que se suele decir que sólo se hace uso del excedente de recursos, caso contrario el costo para el usuario sería demasiado alto en relación al beneficio obtenido (Koegel Buford, 2009, cap. 1).

El tipo de aplicación que haga uso de esta tecnología será el determinante para los requerimientos a nivel cliente. Si hablamos de una aplicación como SETI, el uso del CPU será

intensivo, mientras que si hablamos de una aplicación de distribución de vídeos, entonces el uso de ancho de banda será el determinante.

A nivel físico, estas redes no presentan ningún requerimiento especial, ya que se pueden montar utilizando el *hardware* ya disponible en nuestros hogares (desde un teléfono inteligente hasta un ordenador de escritorio).

Finalmente, a nivel lógico, estas redes presentan un reto para los administradores, tanto desde el aspecto de seguridad como de tráfico en la red. El aspecto de seguridad ya ha sido comentado con anterioridad en este trabajo, mientras que el tráfico de red es algo relativo a la aplicación en uso, por lo que será analizado más adelante en el apartado dedicado a la propuesta.

*Centralizado vs Descentralizado*

Las redes P2P son más bien conocidas actualmente por ser descentralizadas, en el sentido que no hace falta un nodo central/especial para que funcionen. Si bien esto es cierto en gran medida, es necesario aclarar que en la mayoría de las redes grandes de este tipo se hace uso de supernodos, los cuales vendrían a adoptar el papel de nodos centrales, de forma temporal, mientras existen.

Aún así, una red P2P puede ser centralizada desde sus inicios, algo que suele ser utilizado por empresas que promueven estos productos para poder conseguir un empuje inicial en el uso de sus redes. Cada nuevo nodo en una red P2P pública debe ser considerado como egoísta, en el sentido que sólo permanecerá en la red mientras exista algo que lo retenga, caso contrario se desconectará y el tamaño de la red (como sus recursos totales) se reducirá, provocando un efecto similar en otros nodos que se unan más adelante (ya que la red nunca crecerá).

Comparando ambos modos, podríamos decir que las redes centralizadas tienen como desventaja que el tamaño está limitado a las capacidades del nodo central, algo que no sucede en una red descentralizada. Por otro lado, como beneficio, el hecho de tener un nodo central, hace posible el uso de algoritmos de autenticación e identificación más efectivos, como también acelera los tiempos a la hora de reorganizar la red (ante el ingreso y salida de nodos), y permite tener un panorama global del estado en todo momento. Finalmente, en una red descentralizada, el costo del mantenimiento de la misma suele ser mayor debido a que no es posible predecir el comportamiento de los pares que participan, como tampoco la disponibilidad de recursos para un determinado momento a futuro.

*Algoritmos de asignación de ruta*

Al momento de enviar un paquete dentro de una red de pares, es necesario encontrar la mejor ruta disponible para que el paquete llegue en el menor tiempo posible o en la menor cantidad de saltos. Para que esto suceda, se hace uso de distintos algoritmos de asignación de rutas, los cuales deben funcionar en un contexto cambiante, con nodos que se unen a la red, mientras que otros se retiran al mismo tiempo. Cada vez que sucede un cambio en la red, es necesario que las tablas de rutas de varios nodos sean actualizadas para evitar enviar paquetes a, o a través de, nodos que ya no están más, como también aprovechar la presencia de nuevos nodos intermediarios que podrían entregar un paquete en menos tiempo.

Los algoritmos de asignación de rutas varían de acuerdo a si trabajamos sobre una red con o sin estructura. En cada caso, debemos asumir que cada nodo posee al menos una lista de otros pares a los que conectarse, a los cuales llamaremos vecinos o pares cercanos.

Para las redes no estructuradas, se puede hacer uso del *flooding*, en donde cada vecino recibe un paquete y lo reenvía a sus propios pares cercanos. En cada salto, se analiza el contenido de lo recibido para determinar si se puede cumplir con lo pedido, para lo cual se enviará un mensaje de respuesta al nodo origen. Caso contrario, descartará el paquete y lo añadirá a una lista temporal a fines de rechazar el paquete en caso de que le llegue nuevamente. Cada paquete enviado tiene fijado un *TTL*, que es el tiempo de vida, y que va decreciendo en cada salto. Una vez que llega a cero, el paquete es descartado y no se reenvía al próximo nodo, evitando así una inundación total de la red con paquetes viejos. Si una consulta falla, entonces se reenvía nuevamente con un *TTL* mayor, a fines de llegar a más nodos de la red. De esta característica surge una variación al *flooding*, conocida como *expanding ring*, en la cual se envía la primera consulta con un *TTL* muy bajo y se va incrementando con el paso del tiempo. Si los contenidos de la red están repartidos homogéneamente, entonces la probabilidad de encontrarlo rápido son muy altas (Koegel Buford, 2009, cap. 3).

Otro tipo de algoritmo en redes no estructuradas es la caminata aleatoria, en donde el nodo origen elige un vecino aleatoriamente y envía una consulta. En cada nodo, se hace una nueva selección aleatoria en el caso de no poder satisfacer lo solicitado. Al igual que en el *flooding*, cada envío tiene un *TTL* asociado. Para incrementar las posibilidades de éxito, el nodo origen suele enviar la misma consulta a múltiples vecinos al mismo tiempo.

En cuanto a las redes estructuradas, los algoritmos de asignación de rutas que existen nacieron

como una evolución sobre las redes no estructuradas, en la búsqueda de proveer garantías de resolver consultas o localizar un objeto en un tiempo determinado. Estos algoritmos necesitan, y hacen uso de, una geometría de ruta específica, lo que sería básicamente un grafo estático. Distintas dimensiones son tenidas en cuenta para este tipo de algoritmos, como ser el máximo de saltos que toma una consulta en una red de N pares, la organización del espacio de direcciones de pares, como decidir cual es el siguiente salto, la geometría y mantenimiento de la red, entre otras (Koegel Buford, 2009, cap. 4). Los mismos serán analizados más adelante en detalle.

En este último tipo de algoritmos es necesario considerar la geometría bajo condiciones estáticas y dinámicas, ya que se debe planificar no sólo para cuando una red tiene un gran número de nodos, sino también para cuando estos escasean. Por lo que se debe intentar mantener un bajo costo de mantenimiento para que la red funcione, disminuyendo así el tráfico de estado, como también considerar la distancia promedio y la peor. Toda la información que guarda el algoritmo debe estar almacenada en una tabla de rutas, siendo la misma dependiente del tipo de algoritmo utilizado, por lo que puede que esta prime tener entre las primeras posiciones a los nodos con más recursos o bien los nodos más cercanos, impactando así los resultados obtenidos a la hora de enviar una consulta.

*Algoritmos de asignación de rutas para redes estructuradas*

En Koegel Buford (2009, cap. 4), algunos de estos algoritmos son agrupados por distintas categorías, tales como:

- De grado logarítmico con prefijo de asignación de ruta: PRR, S-PRR, Tapestry, Pastry
- Anillo con malla de grado logarítmico: Chord, DKS
- De grado constante: Koorde, Cycloid
- De grado logarítmico con métricas de distancia especializadas: CAN, Kademlia
- De salto O(1): Kelips, OneHop, EpiChord, D1HT

En el primer grupo, nos encontramos con PRR (nombre en referencia a sus autores, Plaxton, Rajaraman y Richa) como el precursor, en donde los nodos asumen los roles de servidores (poseen los datos), *routers* (se encargan de reenviar mensajes) y clientes (hacen las consultas). Cada nodo recibe un identificador único, y cada mensaje enviado se va transmitiendo dígito a dígito de dicho identificador. El salto siguiente está delimitado por el nodo cuyo identificador coincida con el siguiente dígito del identificador buscado (de a uno por vez). Por lo que si el identificador es de 3

dígitos, en 3 saltos consigue alcanzar cualquier nodo en una red de 512 nodos (3 dígitos = de 000 a 111 = 8 identificadores a la potencia 3 = 512). Si la red fuese de 4 dígitos, entonces el tamaño total sería de 16^4 = 65536 nodos. Entre sus características más importantes, está la tolerancia al fallo (por cada salto nos encontraremos con 8 nodos disponibles para saltar en el caso anterior expuesto), el bajo costo a nivel local para mantener la tabla de rutas, como también la eficiencia que garantiza llegar en pocos saltos a cualquier nodo de la red. El único problema de PRR es que no acepta ingresos/egresos dinámicos, por lo que han surgido algunos algoritmos que mejoran esta deficiencia como S-PRR y Tapestry. En el caso de este último, cada nodo recibe un identificador y almacena en su tabla de ruta no sólo los nodos cercanos, sino también el contenido que estos poseen. En cierto modo, es más fácil encontrar a alguien que sepa quien tiene un dato, que alguien que realmente lo tenga. Esto permite incluso que dos o más nodos puedan compartir contenido igual en distintos lugares de la red, minimizando así el tiempo de encontrarlo (tendrían el mismo identificador de contenido). Este algoritmo a su vez agrega la posibilidad de mantenimiento de la red, que funciona a base de solicitudes periódicas de *ping* entre nodos para saber si siguen vivos y disponen del mismo contenido. Finalmente en este grupo nos encontramos con Pastry que es una mejora sobre Tapestry, teniendo cada nodo un identificador de 128-bits, y están generados de tal forma que queden distribuidos de forma uniforme en el espacio de direcciones.

En la categoría de anillo con malla de grado logarítmico, nos encontramos inicialmente con Chord, en donde cada nodo mantiene una lista de vecinos formando un anillo lógico. Los identificadores son asignados por medio de una función de *hash* consistente (por ejemplo, basado en la IP), por lo que si un nodo sale e ingresa con similares características, se posicionará en el mismo lugar o en una posición muy cercana. La información que se almacena en cada *finger table* (tabla de rutas), estará solo relacionada con los nodos activos, minimizando así el costo de mantenimiento de la red. Cuando un nodo quiere enviar una consulta, primero analiza su propia tabla de rutas, si encuentra la clave asociada, envía la petición a tal nodo, sino redirige la petición al nodo cuya clave sea menor a la buscada, que tendrá la tarea de reenviar el mensaje nuevamente siguiendo el mismo ciclo que el nodo que originó la petición. Por otro lado, en el algoritmo DKS, se usa una función similar a Chord, con la excepción que el anillo se parte en k porciones, minimizando aún más la información almacenada por cada par. Luego, a la hora de enviar una consulta, revisa su tabla de rutas para determinar entre que nodos se encuentra la clave objetivo, y envía la petición al nodo inicial de la misma. En una red de 30 nodos, partida en 3, existirá un grupo del 0 al 9, del 10 al 19 y del 20 al 29 para el nodo X. Si se quiere enviar una consulta al nodo 17,

entonces el mensaje se enviará al nodo 10, quien será encargado de seguir con los mismos pasos pero utilizando su propia visión de la red. En cada salto, la cantidad de nodos hacia destino es como mucho, inferior al número de nodos por partición.

Entre los algoritmos de grado constante, nos encontramos con características tales como que el número de interconexiones entre nodos tiene un máximo, y es totalmente independiente del tamaño de la red en sí. Esto minimiza notablemente los costos de mantenimiento de la red en la teoría, pero provoca que en la práctica sea muy difícil de reproducir. Dentro de los algoritmos tenemos a Koorde, que aplica un grafo dentro de un anillo al estilo de Chord, como también Cycloid, que construye un grafo y aplica la asignación de rutas por prefijo como en Pastry para localizar identificadores.

En el cuarto grupo, nos encontramos con algoritmos que organizan los identificadores en dos dimensiones (como las coordenadas en un mapa), haciendo que cada consulta se vaya aproximando cada vez más al objetivo en cada salto. Los algoritmos que se agrupan aquí son CAN (*Content Addressable Network*), el cual utiliza identificadores tanto para los nodos como para el contenido de cada uno, y Kademlia, que permite realizar consultas de forma paralela en varios nodos cercanos, como también actualiza su tabla de rutas durante las consultas, evitando así un costo de mantenimiento periódico.

En el último grupo tenemos los algoritmos de salto O(1), o de un sólo salto, en donde se busca solucionar el problema de la asignación de rutas, haciendo que cada nodo pueda llegar a cualquier otro de forma directa en el mejor de los casos. Las desventajas de estos algoritmos es el alto costo de mantenimiento, como también el costo local de cada par para conocer todos los nodos en la red, lo cual se hace inviable en la práctica para redes de gran tamaño. Entre estos algoritmos se encuentran Kelips, que hace uso de *multicast*, para propagar los ingresos/egresos de la red, como también de una función de *gossip* que consiste en pequeños trozos de información enviados entre nodos, para actualizar el estado de la red periódicamente. A su vez tenemos otros algoritmos como OneHop, que divide la red en porciones con un líder designado que hace de organizador en cada grupo, y EpiChord que no mantiene una tabla de rutas total, sino que divide en porciones la red y se preocupa por lograr buenos resultados en grandes redes, mientras que intenta lograr resultados no tan mediocres en redes con muchos ingresos/egresos.

Cada grupo de algoritmos cuenta con características que lo hacen ideal para un tipo de entorno específico, delimitado por el tamaño de la red, la cantidad de ingreso ingresos/egresos de nodos y

hasta el tipo de aplicación que se ejecutará por encima.

*¿Cómo elegir un algoritmo de asignación de rutas?*

Cada uno de los algoritmos anteriormente presentados, están pensados para desenvolverse correctamente dadas ciertas condiciones del entorno. Por ejemplo, si una red posee una gran cantidad de ingresos/egresos, sería incorrecto hacer uso de un algoritmo que intente mantener rutas estáticas, ya que estaría constantemente haciendo mantenimiento de la red para poder cumplir con su rol básico.

Ante la inmensa cantidad de escenarios posibles, es necesario trabajar con una serie de parámetros que nos garanticen, al menos, que un algoritmo va a funcionar correctamente independientemente de si el entorno es el adecuado. Según Koegel Buford (2009, cap. 4), estos parámetros son:

- Convergencia: cuando se puede garantizar que usando el algoritmo, cualquier nodo va a poder conectarse con otro nodo de la red en una determinada cantidad máxima de saltos
- Estabilidad: analizar que tan estable es el algoritmo cuando se presenta una alta tasa de ingresos o egresos, y pensar si en algún momento puede desestabilizarse (sea para redes muy grandes o muy pequeñas)
- Grado de nodo: la cantidad de nodos vecinos con los cuales un par permanece en contacto continuo (esto impactará tanto la tabla de rutas como el costo de mantenimiento)
- Total de saltos: la cantidad de saltos que debe dar un mensaje para llegar de un nodo a cualquier otro dentro de la red (promedio y peor caso)
- Grado de tolerancia a fallos: cuantos nodos pueden fallar sin que se pierdan datos o que los mensajes fallen en llegar a destino.
- Costo de mantenimiento: el ancho de banda que se debe consumir por nodo para poder poder mantener la estructura de la red, como también su relación con la tasa de ingresos/egresos
- Balance de carga: ver si un grupo de nodos posee más carga en la red que otros, y si los mensajes son distribuidos de manera uniforme entre los pares

Queda claro que cada parámetro impacta de alguna forma a otro, ya que si buscamos que la

cantidad de saltos en el peor caso sea baja, estaremos incrementando el grado de nodo. Por otro lado, si buscamos poco costo de mantenimiento, entonces tendremos menos actualizaciones periódicas, ocasionando una potencial desestabilización de la red si la misma llega a ser de gran tamaño.

*DHT: Tablas distribuidas de Hash*

Este concepto aparece reiteradas veces cuando hablamos de redes P2P estructuradas, ya que es el cimiento base para que estas puedan funcionar de forma descentralizada. El DHT es básicamente un sistema que permite compartir datos mediantes nodos, sin necesidad de un nodo central que intermedie. Su función básica es proveer un servicio de búsqueda, el cual hace uso de tablas que contienen *hashes* de los nodos y contenidos que se encuentran en la red. Cada participante de la red, posee parte de esta tabla (de ahí la palabra distribuida), permitiendo que la misma sea escalable a redes muy grandes, y adaptándose en casos de ingresos/egresos de nodos, o incluso errores de red.

La importancia del DHT en las redes P2P radica por un lado en la nula necesidad de un nodo central que permita el inicio de búsquedas sobre los contenidos o nodos que están en la red (aunque se puede implementar con nodos centrales si se desea), como también por el hecho que es un sistema ya probado y que funciona, lo que disminuye la complejidad a la hora de implementar aplicaciones que hagan uso del mismo.

Al analizar su funcionamiento, nos encontramos con que cada objeto (s) disponible en la red tiene un identificador único (sid), resultado de una operación de *hashing* (como ser SHA1), asociado al nodo que lo posee (p). Esta información se envía a cada nodo participante por medio de una función insertar(sid,s). En el caso que un nodo quiera obtener dicho archivo, sólo necesitará hacer una petición con la función obtener(sid), la cual será enviada al nodo que posee los datos (el que ejecutó la función de insertar anteriormente). (Koegel Buford, 2009, p. 34)

DHT fue inicialmente adoptado por Chord, Pastry, Tapestry y CAN. Actualmente se implementan en protocolos como BitTorrent, en donde se adoptó principalmente por la característica de *trackerless,* que le permite evitar que los nodos centrales deban cargar con la búsqueda de nodos/contenido y trasladar ese trabajo a los mismos clientes de las redes P2P.

# Capítulo IX: Productos de monitorización

*Software de monitorización*

Existe una amplia gama de productos disponibles en el mercado, con versiones en distintas plataformas, y que se adoptan a alguno de los 3 modos de monitorización (agente, sin agente, híbrido).

En este capítulo se presentará una selección de los productos actualmente existentes en el mercado que estén disponibles públicamente, que tengan -al menos- una versión publicada, y que cuenten mínimamente con una empresa que brinde soporte para los mismos. La información será provista en formato de fichas, a fines de facilitar la comparación entre productos. Al final del capítulo se destacarán aquellos productos con las características más importantes.

En cada ficha, se encontrará con la siguiente información:

- Empresa que lo provee/soporta
- Tipo de licencia: comercial, libre, mixta
- Versión analizada o más actual (con un asterisco si fue probada durante este trabajo)
- Tipo de monitorización que emplea el producto: con agente, sin agente, híbrida
- Tipo de comunicación que soporta: sólo SNMP o SNMP con traps
- Sistemas operativos soportados en su totalidad, como también si posee soporte nativo para Linux o si es por medio de SSH/SNMP
- Tipo de panel de control que utiliza: por aplicación nativa, móvil o web
- Soporte para MIB (Base de Información de Administración para SNMP)
- Si permite personalizar o crear componentes adicionales para mejorar el producto
- Mercado al que se dirige el principal desarrollador con el producto, según sus características y como se comercializa
- Mención de funciones que sean especiales a este producto
- Comentarios oportunos en los casos que corresponda

| *ActiveXperts Network Monitor* | |
|---|---|
| Empresa | ActiveXperts |
| Licencia | Comercial |
| Versión | 7.4 |
| Tipo de Monitorización | Sin agente |
| Tipo de comunicación | SNMP |
| S.O. soportados | Windows de forma nativa, UNIX/Linux usando SSH |
| Soporte para Linux | Si, pero no nativo |
| Tipo de panel de control | Aplicación (sólo Windows) y web |
| Soporte para MIB | Si, como adicional no incluido por defecto |
| Personalizado | De reportes y monitores (con PowerShell y VBScript) |
| Mercado objetivo | Empresas grandes, empresas de servicios IT |
| Funciones especiales | Soporte para programación con PowerShell y VBScript en Windows, bash para Linux/UNIX. Destaca la generación de reportes detallados de distintos niveles (técnicos y para toma de decisiones). Soporte para monitores externos (de humedad, luz, movimiento, entre otros), múltiples bases de datos y varias plataformas de virtualización de Windows |
| Web | http://www.activexperts.com/activmonitor/ |
| Comentarios | |
| El producto está pensado para empresas que proveen servicios IT, en el sentido que permite manejar varias redes (de distintos clientes). La aplicación de escritorio sólo funciona en Windows, lo que limita su uso, aunque cuenta con la opción Web que ofrece casi las mismas funciones. La monitorización sin agente le permite controlar plataformas distintas a Windows, pero los controles se hacen vía SNMP o enviando comandos por SSH, lo que limita mucho las posibilidades del producto, ya que es muy difícil encontrar redes en donde no coexistan dos o más plataformas distintas interconectadas. | |

| *Axence nVision* | |
|---|---|
| Empresa | Axence |
| Licencia | Comercial |
| Versión | 6.1.2 |
| Tipo de Monitorización | Sin agente |
| Tipo de comunicación | SNMP |
| S.O. soportados | Windows, UNIX y Linux |
| Soporte para Linux | Si, no nativo |
| Tipo de panel de control | Aplicación (Windows) |
| Soporte para MIB | No |
| Personalizado | Sólo en reportes y acciones correctivas ante incidentes |
| Mercado objetivo | Empresas pequeñas o proveedores de servicios de empresas de este tamaño |
| Funciones especiales | Detección automática de nuevos nodos, mapeo gráfico de la red, soporte para WMI, soporte para monitores externos de temperatura, humedad, entre otros; monitoriza *routers* y *switches*, como también tráfico de red. |
| Web | http://www.axencesoftware.com/es/nvision/network |
| Comentarios | |
| nVision cuenta con 5 módulos en total, para monitorizar, inventario, actividad de usuarios, *tickets* de soporte, y de protección de datos, lo que representa un producto integral. La idea del producto es relacionar la información de los distintos módulos para poder brindar un estado de red completo al usuario, pudiendo incluso generar un *ticket* para que alguien tome una acción correctiva. Sólo opera sobre Windows, aunque puede monitorizar de forma remota y limitada servidores de otras plataformas. | |

| *BMC MainView* | |
|---|---|
| Empresa | BMC |
| Licencia | Comercial |
| Versión | - |
| Tipo de Monitorización | Sin agente |
| Tipo de comunicación | SNMP |
| S.O. soportados | z/OS, z/Linux, z/VM |
| Soporte para Linux | Sí, solo en hardware mainframe |
| Tipo de panel de control | Web |
| Soporte para MIB | No |
| Personalizado | Mínimo, de reportes y acciones de respuesta |
| Mercado objetivo | Empresas grandes con sistemas mainframe |
| Funciones especiales | Soporte para SAN, generación de reportes sobre capacidad disponible y predicciones, resolución de problema automática, soporte para mainframes. |
| Web | http://www.bmc.com/products/brand/mainview.html |
| Comentarios | |
| MainView se ocupa de un segmento muy particular como lo es el Mainframe. Este producto cuenta con soporte para monitorizar no sólo los nodos servidores, sino también todo el hardware que lo rodea, incluyendo la Storage Area Network. | |

| *Cacti* | |
|---|---|
| Empresa | The Cacti Group, Inc |
| Licencia | Código abierto |
| Versión | 0.8.8a |
| Tipo de Monitorización | Con o sin agente, dependiendo de las necesidades |
| Tipo de comunicación | SNMP, SNMP Traps |
| S.O. soportados | Linux |
| Soporte para Linux | Si, nativo |
| Tipo de panel de control | Web |
| Soporte para MIB | Si (no nativo) |
| Personalizado | Por medio de *plugins* y *templates* |
| Mercado objetivo | PyMes, aunque soporta redes de gran tamaño |
| Funciones especiales | Se puede personalizar los reportes, las alertas, los monitores, y hasta integrarlo con otras aplicaciones disponibles de monitorización. Soporta la creación de gráficos de tráfico de red. Está más orientado al análisis histórico de datos que a la monitorización en tiempo real. |
| Web | http://www.cacti.net/ |
| Comentarios | |
| Cacti es un producto de código abierto que hace uso de RRDTool, una herramienta también libre que permite almacenar series temporales de datos (como actividad), de manera eficiente. La comunidad de Cacti ha extendido el uso de la herramienta con nuevos adicionales ya sea en formato de *plugins* o *templates*. Es una herramienta básica, pero efectiva en su tarea de análisis histórico de datos. El uso y administración puede resultar complejo para redes grandes, sobretodo teniendo en cuenta que no es una herramienta de monitorización en sí, sino que debe adaptarse por medio de los *plugins* disponibles, o combinando con algún otro producto como Nagios. | |

| *GFi Network Server Monitor* | |
|---|---|
| Empresa | GFI |
| Licencia | Comercial |
| Versión | 7.0 |
| Tipo de Monitorización | Sin agente |
| Tipo de comunicación | SNMP |
| S.O. soportados | Windows, Linux |
| Soporte para Linux | Por medio de SSH y SNMP |
| Tipo de panel de control | Aplicación (Windows), Web |
| Soporte para MIB | No |
| Personalizado | VBScript para Windows; Bash scripts para Linux |
| Mercado objetivo | PyMes |
| Funciones especiales | Acciones automatizadas (reiniciar un servidor, reiniciar servicios, etc); generación de reportes complejos sobre estado de red; soporte para bases de datos por ODBC o ADO; monitorización de Microsoft Exchange, ISA e IIS nativo |
| Web | http://www.gfi.com/network-server-monitoring |
| Comentarios | |
| El software de control base está disponible sólo para Windows, y el soporte para Linux está limitado a las funciones que se puedan ejecutar de forma remota y dependiendo de los privilegios del usuario creado, o por medio de SNMP. | |

| *HP OpenView Network Node Manager i (HP NNMi)* | |
|---|---|
| Empresa | Hewlett Packard |
| Licencia | Comercial |
| Versión | 9.20 |
| Tipo de Monitorización | Sin agente, aunque con opción de agente instalando HP Ops Manager |
| Tipo de comunicación | SNMP, SNMP Traps (con add-on) |
| S.O. soportados | Windows, Linux, HP-UX, Solaris |
| Soporte para Linux | Si |
| Tipo de panel de control | Aplicación (sólo Windows), Web |
| Soporte para MIB | Si |
| Personalizado | Mínimo, es necesario adquirir otros productos OpenView |
| Mercado objetivo | Empresas que proveen servicios IT |
| Funciones especiales | Integración con muchos productos de la gama OpenView, mejorando la utilidad de la aplicación. |
| Web | http://www.openview.hp.com/products/nnm/ |
| Comentarios | |
| Este producto es usado por HP para brindar sus servicios, y le compite muy de cerca a IBM ITM. Tiene características muy interesantes si analizamos el conjunto de productos OpenView (la i en NNMi es un reflejo de que se enfoca en la integración, según HP). El producto por sí solo es útil para monitorizar, pero no muestra su total potencial hasta adquirir los otros adicionales. Cuenta con soporte para casi todas las plataformas, lo que lo hace candidato a empresas que brindan servicios de IT. La opción de sin o con agente, le permite trabajar sobre redes de distintos tamaños, pudiendo elegir entre minimizar el costo de mantenimiento o maximizar la cantidad de datos recolectados respectivamente. | |

| *IBM Tivoli Monitoring* | |
|---|---|
| Empresa | IBM |
| Licencia | Comercial |
| Versión | 6.2 (*) |
| Tipo de Monitorización | Con agente aunque soporta sin agente |
| Tipo de comunicación | SNMP, SNMP Traps (con add-on) |
| S.O. soportados | Windows, AIX, Solaris, HP-UX y Linux |
| Soporte para Linux | Si, con varios agentes disponibles |
| Tipo de panel de control | Aplicación |
| Soporte para MIB | Si |
| Personalizado | Se pueden personalizar los monitores y alertas; se pueden crear *scripts* de monitorización que deben ser distribuidos manualmente. Se integra fácilmente con otros productos de la gama Tivoli, permitiendo monitorizar bases de datos con agentes nativos, productos específicos de Microsoft e incluso plataformas de virtualización. |
| Mercado objetivo | Empresas medianas o grandes, debido al coste en infraestructura para poder hacer uso de las funciones más importantes; empresas que brindan servicios de IT |
| Funciones especiales | Soporte nativo para funciones propias de AIX virtualizado (único entre cualquier producto de monitorización), soporte nativo para herramientas de SAP/PeopleSoft/Siebel; agente disponible para todas las plataformas; soporte para creación de monitores con simples scripts. |
| Web | http://www.ibm.com/software/tivoli/products/monitor/ |
| Comentarios | |
| A diferencia de HP NNMi, el producto de IBM da soporte a todas las plataformas, incluso HP-UX, de forma nativa (por medio de Java). Este producto es el usado por IBM para ofrecer sus servicios a nivel mundial, como también por otras empresas proveedoras de servicios IT. A fines de reducir el costo en memoria, el producto cuenta con varios agentes por plataforma que se encargan de tareas específicas. La instalación del producto es un poco engorrosa, y debido a que es con agente, necesita ser instalado manualmente en cada nodo a monitorizar, como también actualizado de la misma forma. | |

| *Kaseya Network Monitor* | |
|---|---|
| Empresa | Kaseya |
| Licencia | Comercial |
| Versión | 4.1 Distributed Edition |
| Tipo de Monitorización | Sin agente |
| Tipo de comunicación | SNMP |
| S.O. soportados | Windows |
| Soporte para Linux | Si, no nativo |
| Tipo de panel de control | Web |
| Soporte para MIB | Si, como adicional (depende de la versión) |
| Personalizado | Muy básico, no permite adaptar a necesidades |
| Mercado objetivo | Empresas proveedoras de servicios IT o con redes muy amplias. Existe una versión estándar para empresas pequeñas. |
| Funciones especiales | Detección automática de nuevos nodos, sugerencias de que monitorizar en cada nuevo nodo detectado, herramienta de reportes con gráficos, soporte para múltiples redes en una misma vista (versión distribuida). Monitoriza archivos/directorios; soporte para virtualización sobre VMWare. |
| Web | http://www.kaseya.com/features/network-monitor.aspx |
| Comentarios | |
| El hecho de no contar con agente facilita mucho su uso en redes de gran tamaño, pero también limita notablemente los tipos de monitores que se pueden utilizar para los nodos en la red. Dispone de dos versiones, siendo la distribuida la mejor en características, mientras que la estándar no goza de soporte para múltiples redes (cada red necesitaría su propia instalación). El panel de control es vía web, lo que permite operarlo fácilmente desde cualquier sistema operativo, siendo bastante completo y amigable en cuanto a su interfaz. | |

| *Microsoft System Center 2012* | |
|---|---|
| Empresa | Microsoft |
| Licencia | Comercial |
| Versión | 2012 |
| Tipo de Monitorización | Con agente, sin agente para UNIX |
| Tipo de comunicación | SNMP |
| S.O. soportados | Windows y UNIX (sólo en versión Operations) |
| Soporte para Linux | Sólo en la versión Operations |
| Tipo de panel de control | Aplicación (sólo Windows) y Web |
| Soporte para MIB | Si |
| Personalizado | Permite personalizar reportes |
| Mercado objetivo | Tiene dos versiones, Essentials para empresas pequeñas y medianas, Operations para redes más grandes o proveedores de servicios IT |
| Funciones especiales | Fuerte integración con productos Microsoft; soporte para scripts PowerShell en nodos Windows; soporte para distintas plataformas de virtualización |
| Web | http://www.microsoft.com/en-us/server-cloud/system-center |
| Comentarios | |
| El producto cuenta con dos versiones, siendo la más básica Essentials que sólo soporta servidores o clientes Windows, mientras que la versión de Operations da soporte genérico a plataformas UNIX. Está fuertemente integrado con productos Microsoft, obteniendo el nivel más granular de datos en esta plataforma. Lamentablemente, es un producto que se enfoca en Windows principalmente, haciendo que el soporte en las otras plataformas sea bien sencillo, limitando mucho su uso en redes heterogéneas. | |

| *Monitor One* | |
|---|---|
| Empresa | FineConnection |
| Licencia | Comercial/Gratuita |
| Versión | FP1.111.415 |
| Tipo de Monitorización | Sin agente |
| Tipo de comunicación | SNMP, SNMP Traps |
| S.O. soportados | Windows |
| Soporte para Linux | Si, no nativo |
| Tipo de panel de control | Aplicación (sólo Windows), Web |
| Soporte para MIB | Si |
| Personalizado | Mínimo, sólo reportes |
| Mercado objetivo | PyMes |
| Funciones especiales | Soporte de SNMP Traps, disponibilidad de adicionales para monitorizar SNMP que son gratuitos, soporte para dispositivos externos de control de temperatura, humedad, energía, entre otros. |
| Web | http://www.fineconnection.com/ |
| Comentarios | |
| El producto cuenta con una versión gratuita que es completa, pero se cierra aleatoriamente para incentivar la compra del producto comercial. Cuenta con un buen soporte de dispositivos gracias a que soporta SNMP hasta la versión 3. El único problema es que el panel de control funciona sólo sobre Windows y la interfaz web no permite realizar muchas operaciones más que ver el estado de la red y generar reportes. Es un producto sencillo para redes pequeñas o medianas. | |

| *Nagios Core* | |
|---|---|
| Empresa | Nagios Enterprises, LLC |
| Licencia | Código Abierto |
| Versión | 3.4.1 (*) |
| Tipo de Monitorización | Con o sin agente, ambas pueden coexistir para un nodo monitorizado |
| Tipo de comunicación | SNMP, SNMP Traps |
| S.O. soportados | Windows, Linux, UNIX |
| Soporte para Linux | Si, nativo |
| Tipo de panel de control | Web |
| Soporte para MIB | No nativo en Linux y UNIX, no en Windows |
| Personalizado | Si, en componentes, reportes y presentación |
| Mercado objetivo | PyMes |
| Funciones especiales | Al ser de código abierto, se puede personalizar por completo, tanto en *plugins*, como en reportes y *templates*. Las características especiales dependerán de las necesidades de quien lo instale. |
| Web | http://www.nagios.com/products/nagioscore |
| Comentarios | |
| Nagios Core es la versión gratuita y de código abierto de Nagios Enterprises. En esta versión encontramos características que están a prueba para ser promovidas a la versión XI. El proceso de instalación es complicado, y la administración por nodo no es muy sencilla. El servidor central requiere de un servicio para brindar web (Apache), una base de datos, y librería de imágenes (para la generación de los gráficos). El código se baja de la página y se compila, lo cual puede no ser deseable en un entorno de empresa grande. | |

| *Nagios XI* | |
|---|---|
| Empresa | Nagios Enterprises, LLC |
| Licencia | Comercial |
| Versión | 2011R3.2 (*) |
| Tipo de Monitorización | Con o sin agente, ambas pueden coexistir para un nodo monitorizado |
| Tipo de comunicación | SNMP, SNMP Traps |
| S.O. soportados | Windows, Linux, UNIX |
| Soporte para Linux | Si, nativo |
| Tipo de panel de control | Web |
| Soporte para MIB | No nativo en Linux y UNIX, no en Windows |
| Personalizado | Si, en componentes, reportes y presentación |
| Mercado objetivo | PyMes y empresas de gran tamaño |
| Funciones especiales | Informes de rendimiento histórico, agrupado de nodos y/o servicios. |
| Web | http://www.nagios.com/products/nagiosxi |
| Comentarios | |
| Nagios XI es la versión pulida, tanto en interfaz gráfica como en componentes, de Nagios Enterprises. A diferencia de la versión Core, el proceso de instalación en XI es automático, y en menos de 30 minutos ya podemos estar utilizando el producto. El agregado de nodos es muy sencillo, como también el personalizado de los monitores y alertas (tiene un asistente para hacerlo aún más sencillo). Cuenta con soporte, ya que es pago, y los componentes adicionales son validados por los desarrolladores oficiales, por lo que se garantiza calidad en lo que se instala. Cabe destacar que empresas como AOL, ComCast, DHL, AT&T, Linksys, JPMorganChase, Shell, SONY, Toshiba, Yahoo!, entre otras, figuran como clientes de Nagios Enterprise. | |

| *Nimsoft Monitor* | |
|---|---|
| Empresa | Nimsoft |
| Licencia | Comercial |
| Versión | - (*) |
| Tipo de Monitorización | Con agente |
| Tipo de comunicación | SNMP, SNMP Traps |
| S.O. soportados | Linux, UNIX (AIX, Solaris, HP UX, TRU 64), Windows, iSeries (IBM Power Systems), ESX |
| Soporte para Linux | Si, con agente nativo |
| Tipo de panel de control | Aplicación (multiplataforma); Web |
| Soporte para MIB | Si |
| Personalizado | Si, desde reportes, tipos de monitores, muy complejo pero con muchas opciones para personalizarlo. |
| Mercado objetivo | Se vende como SaaS por lo que está orientado a cualquier tipo de cliente que prefiera pagar por uso y dejar que la empresa Nimsoft se haga cargo de la monitorización, aunque también ofrece paneles de control por si algún tercero quiere administrarlo por su cuenta. |
| Funciones especiales | Análisis de causa raíz; Descubrimiento de red automático; Actualizaciones automáticas; soporte para APIs/SDKs; compatible con ITIL; Análisis de rendimiento; disponibilidad de herramientas para administrar vía móvil |
| Web | http://www.nimsoft.com/solutions/nimsoft-monitor.html |
| Comentarios | |
| El producto de Nimsoft está pensado para clientes exigentes que quieren controlar desde lo físico hasta lo virtual. Soporta múltiples sistemas operativos y arquitecturas de hardware, como también componentes físicos tales como *switches*, *routers*, SAN de tipo EMC o NetApp, y hasta virtualización por medio de ESX, RHEV, IBM PowerVM, entre otros. Más que un software, es un servicio (SaaS), por lo que siempre está actualizándose a todos sus clientes de forma constante, ofreciendo características que otros productos del mercado cobran lanzando versiones más nuevas. El único problema de venderlo como un SaaS limita su acceso a clientes con altas medidas de seguridad como gobiernos, empresas financieras o de industrias importantes. | |

| *OpenNMS* | |
|---|---|
| Empresa | OpenNMS Group |
| Licencia | Código Abierto |
| Versión | 1.10 |
| Tipo de Monitorización | Sin agente |
| Tipo de comunicación | SNMP, SNMP Traps |
| S.O. soportados | Linux, Windows |
| Soporte para Linux | Si, remoto |
| Tipo de panel de control | Web |
| Soporte para MIB | Si, no nativo |
| Personalizado | Si, de reportes, monitores, alertas |
| Mercado objetivo | Empresas pequeñas y medianas |
| Funciones especiales | Detección automática de nuevos nodos en la red, análisis de rendimiento, generación de reportes complejos, identificación de problemas recurrentes, monitorización de actualizaciones de software, inventario de nodos |
| Web | http://www.opennms.org/ |
| Comentarios | |
| OpenNMS es un producto de código abierto que no cuenta con una versión comercial, pero brindan soporte pago en el caso de empresas que lo requieran. El producto es simple, pero puede ser personalizado para sacarle el máximo provecho, ya sea creando nuevos *plugins* y monitores, o bien instalando los ya disponibles en la comunidad. | |

| *Pandora FMS* | |
|---|---|
| Empresa | Ártica ST |
| Licencia | Código abierto y versión comercial (Enterprise) |
| Versión | 4.0.2 (*) |
| Tipo de Monitorización | Con o sin agente |
| Tipo de comunicación | SNMP, SNMP Traps |
| S.O. soportados | Windows, Linux, FreeBSD, Solaris, HP-UX, AIX, MacOS |
| Soporte para Linux | Si, nativo |
| Tipo de panel de control | Web |
| Soporte para MIB | Si |
| Personalizado | En ambas versiones, reportes, *dashboard*, monitores y módulos (plugins) |
| Mercado objetivo | Empresas de todo tamaño, está orientado a ser flexible |
| Funciones especiales | Integración con otros productos de monitorización como HP NNM, soporte para múltiples bases de datos, disponibilidad de API externa, interfaz CLI para controlar el servidor principal, exploración automática de redes, sistema de inventariado |
| Web | http://www.pandorafms.com/ |
| Comentarios | |
| Las letras FMS del nombre de este producto significan “Sistema de Monitorización Flexible” (en inglés), por lo que el producto busca monitorizar no sólo servidores, sino cualquier tipo de dispositivo que tenga soporte para SNMP. Posee soporte nativo de agente para varias plataformas, y puede ser instalado fácilmente en caso de no querer usar los agentes. Al tener una versión de código abierto, cuenta con una comunidad de desarrolladores y colaboradores que brindan un soporte (gratuito) muy completo. | |

| *PRTG Network Monitor* | |
|---|---|
| Empresa | Paessler |
| Licencia | Comercial |
| Versión | 9.2 |
| Tipo de Monitorización | Sin agentes |
| Tipo de comunicación | SNMP |
| S.O. soportados | Windows y UNIX |
| Soporte para Linux | Si, no nativo |
| Tipo de panel de control | Web |
| Soporte para MIB | Si, como aplicación externa |
| Personalizado | Mínimo, de reportes, gráficas y *dashboards* |
| Mercado objetivo | Empresas medianas o grandes, con redes muy distribuidas |
| Funciones especiales | Excelente herramienta de reporte con gráficas muy completas sobre estado de la red; posibilidad de generar dependencias en los servicios monitorizados, lo que permite obtener un panorama general fácilmente; monitores pro-activos basados en datos históricos. |
| Web | http://www.paessler.com/prtg |
| Comentarios | |
| Este producto de Paessler se enfoca en redes de gran tamaño que estén distribuidas, contando con opciones que permiten ver el estado actual de manera fácil y entendible. La interfaz web cuenta con muchas opciones en cuanto a reportes, y se caracteriza por ser fácil de usar. Al no tener agentes, se puede instalar y dejar funcionando en pocas horas, incluso en entornos bien dispersos geográficamente. | |

| *SNMPc Enterprise Edition* | |
|---|---|
| Empresa | Castle Rock Computing |
| Licencia | Comercial |
| Versión | 8.0 |
| Tipo de Monitorización | Con agente |
| Tipo de comunicación | SNMP, SNMP Traps |
| S.O. soportados | Windows |
| Soporte para Linux | Sólo por SNMP |
| Tipo de panel de control | Aplicación (Windows), cliente web (como adicional) |
| Soporte para MIB | Si |
| Personalizado | Sólo de las alertas disponibles |
| Mercado objetivo | Clientes con redes bien amplias, multinacionales o proveedores de servicios nacionales |
| Funciones especiales | Descubrimiento de dispositivos de red automático (gracias al agente); soporte para SNMP v1, 2c y 3; pensado para IPv6 |
| Web | http://www.castlerock.com/products/snmpc |
| Comentarios | |
| Es un producto que no podría ser aprovechado por empresas de pequeño tamaño, debido a que sólo se enfoca en redes de gran tamaño y dispersas. El uso de un agente para Windows limita su uso en redes mixtas con varios sistemas operativos. | |

| *VeraxNMS* | |
|---|---|
| Empresa | VeraxSystems |
| Licencia | Comercial |
| Versión | 1.9.5 |
| Tipo de Monitorización | Sin agente |
| Tipo de comunicación | SNMP |
| S.O. soportados | Windows, AIX, Solaris, AS/400, Linux |
| Soporte para Linux | Sí, solo en versión Enterprise y Premium |
| Tipo de panel de control | Web |
| Soporte para MIB | Si |
| Personalizado | Mínimo, de reportes y *dashboard* |
| Mercado objetivo | Empresas con entornos distribuidos o proveedores de servicios IT |
| Funciones especiales | Soporte para Cloud Computing, múltiples bases de datos, generación de reportes complejos, mapa geográfico de ubicación de nodos, reportes de rendimiento |
| Web | http://www.veraxsystems.com/en/products/nms |
| Comentarios | |
| Cuenta con cinco versiones, entre ellas una llamada Express y gratuita, que sólo cuenta con monitores para Windows, mientras que la versión Enterprise y Premium sí lo soporta, junto con opciones de virtualización, bases de datos y dispositivos externos. | |

| *WhatsUp Gold* | |
|---|---|
| Empresa | IPSwitch |
| Licencia | Comercial |
| Versión | 15 |
| Tipo de Monitorización | Sin agente |
| Tipo de comunicación | SNMP, SNMP Traps |
| S.O. soportados | Windows, Linux, UNIX |
| Soporte para Linux | Sí, solo en versión Premium y distribuida (no nativo) |
| Tipo de panel de control | Web, Aplicación (sólo Windows) como alternativa |
| Soporte para MIB | Si, como adicional |
| Personalizado | Mínimo, de reportes y *dashboard* |
| Mercado objetivo | PyMes y grandes empresas, testeado en redes de hasta 100 mil nodos |
| Funciones especiales | Herramienta de *tickets*, generación de reportes complejos, interfaz web muy completa con acceso a todas las funciones, descubrimiento automático de nuevos nodos, +200 reportes por defecto sobre que tener en cuenta para monitorizar |
| Web | http://www.whatsupgold.com/products/whatsup-gold-core/ |
| Comentarios | |
| Este producto de IPSwitch viene pensado para instalar rápidamente en cualquier red de gran tamaño, con 3 versiones disponibles, estándar, premium y distribuida. Las dos versiones más complejas incluyen gráficas de estado de red en tiempo real, soporte para WMI en nodos Windows y soporte para entornos UNIX. La opción estándar es bien básica y sólo pensada para redes con Windows y dispositivos con soporte SNMP. | |

| *Zabbix* | |
|---|---|
| Empresa | Zabbix SIA |
| Licencia | Código abierto |
| Versión | 2.0.2 (*) |
| Tipo de Monitorización | Con o sin agente |
| Tipo de comunicación | SNMP, SNMP Traps |
| S.O. soportados | AIX, FreeBSD, Linux, OpenBSD, Solaris, Windows |
| Soporte para Linux | Si, nativo |
| Tipo de panel de control | Web |
| Soporte para MIB | Si, no nativo en Linux |
| Personalizado | Si, en reportes, monitores, alertas, acciones predefinidas. |
| Mercado objetivo | PyMes y empresas de gran tamaño |
| Funciones especiales | Soporte de SNMP Traps, SQLite, PHP, OpenSSL, agentes nativos en múltiples plataformas, descubrimiento automático de nodos |
| Web | http://www.zabbix.com/ |
| Comentarios | |
| El hecho de contar con agentes nativos en múltiples plataformas, le da una ventaja de rendimiento a este producto. Por otro lado, aquellas plataformas no soportadas, pueden ser monitorizadas sin agente, aunque al no ser nativo los resultados quizás no sean óptimos. De fácil instalación, sin costo alguno para iniciarse, es una de las herramientas elegidas por pequeñas empresas para monitorizar sus redes. | |

| *Zenoss* | |
|---|---|
| Empresa | Zenoss, Inc |
| Licencia | Comercial (con soporte) y Código abierto (versión comunitaria) |
| Versión | 4.2.0 (*) |
| Tipo de Monitorización | Sin agente (usa SNMP en Linux y WMI en Windows) |
| Tipo de comunicación | SNMP |
| S.O. soportados | Linux (64 bits), Mac, Unix, Windows |
| Soporte para Linux | Si, remoto |
| Tipo de panel de control | Web |
| Soporte para MIB | No nativo en Linux y UNIX, no en Windows |
| Personalizado | De interfaz gráfica web, posibilidad de añadir funcionalidad con *plugins* de versión comunitaria o algunos otros pagos |
| Mercado objetivo | PyMes y entornos muy distribuidos con posibilidades de uso en Cloud Computing |
| Funciones especiales | Descubrimiento de nodos y dispositivos; mapeo de la red en cuanto a relaciones de dependencias; "*Automated Root Cause Analysis*" para encontrar la causa de un problema de manera sencilla y con pruebas históricas (sólo con paquete comercial); Análisis de estado de la red; Respuestas automáticas antes eventos con condiciones; Compatibilidad con servicios de Cloud Computing. |
| Web | http://www.zenoss.com/ |
| Comentarios | |
| Zenoss es, hoy en día, la competencia directa de Nagios en cuanto a la monitorización de red en el mundo del código abierto. Ambos poseen versiones comerciales y comunitarias, los que les permite tener presencia en la industria, como también un buen soporte por parte de desarrolladores y usuarios. El software es bastante más complejo, pero a la vez más completo y amigable que su competidor, ya que fue desarrollado desde sus inicios con esa idea en mente. El producto cuenta con clientes importantes como VMWare, Motorola, LinkedIn, Broadcom, Deutsche Bank, entre otros. | |

*Tabla comparativa de productos*

Debido a la extensa cantidad de productos, es necesario que destaquemos sólo aquellos aspectos más relevantes para este trabajo. Para compararlos, nos enfocaremos en los siguientes aspectos:

- Soporte en Linux: Esto es primordial para el trabajo, ya que inicialmente se busca dar soporte nativo sólo a servidores con este sistema operativo.
- Tipo de monitor que utiliza: Esta característica nos permitirá entender que productos predominan en el mercado actualmente, con/sin agente, o híbridos.
- Comentarios: Se detallan características puntuales de cada producto, que pueden aportar al trabajo o permitir el descarte del producto al no apuntar al mismo mercado objetivo.

A continuación, la tabla comparativa con todos los productos analizados:

| *Producto* | *¿Soporta Linux?* | *Tipo de Monitor* | *Comentarios* |
|---|---|---|---|
| ActiveXperts Network Monitor | Si, remoto | Sin agente | Enfocado en productos Microsoft |
| Axence nVision | Si, remoto | Sin agente | Producto completo, enfoque en Microsoft y redes |
| BMC MainView | Si, remoto | Sin agente | Enfocado en mainframe |
| Cacti | Si, nativo | Híbrido | Enfoque en reportes y datos históricos |
| GFi Network Server Monitor | Si, remoto | Sin agente | Enfocado en productos Microsoft |
| HP OpenView NNMi | Si, nativo | Híbrido | Requiere de otros productos OpenView para funcionar |
| IBM Tivoli Monitoring | Si, nativo | Híbrido | Alto costo en arquitectura necesaria |
| Kaseya Network Monitor | Si, remoto | Sin agente | Sugerencia sobre que monitorizar |
| Microsoft System Center 2012 | Si, remoto | Híbrido | Enfocado en productos Microsoft |
| Monitor One | Si, remoto | Sin agente | Enfocado en productos Microsoft |
| Nagios Core | Si, nativo | Híbrido | Es necesario compilar el código |
| Nagios XI | Si, nativo | Híbrido | Versión comercial del Nagios Core |
| Nimsoft Monitor | Si, nativo | Con agente | Vendido como SaaS |
| OpenNMS | Si, remoto | Sin agente | Enfoque en análisis de rendimiento e inventarios |
| Pandora FMS | Si, nativo | Híbrido | Enfocado en ser flexible y adaptable a distintos entornos |
| PRTG Network Monitor | Si, remoto | Sin agente | Enfoque en acciones pro-activas, datos históricos |
| SNMPc Enterprise Edition | Si, remoto | Con agente | Sin agente en Linux |
| VeraxNMS | Si, remoto | Sin agente | Enfoque en Cloud Computing y bases de datos |
| WhatsUp Gold | Si, remoto | Sin agente | Solución completa, con herramienta de *tickets* incluida |
| Zabbix | Si | Híbrido | Soporte multiplataforma, enfoque en alertas |
| Zenoss | Si, remoto | Sin agente | Enfocado en Cloud Computing, con sistema de análisis de causa raíz automatizado |

En resumen, podemos destacar lo siguiente:

- Tenemos que dejar de lado aquellos productos que se enfocan en productos Microsoft como factor de venta, ya que no ofrecen soluciones interesantes para la plataforma que estamos analizando (Linux)
- Existen productos en las 3 categorías de tipo de monitorización, pero priman las sin agente (11 sobre 21), luego las híbridas (8 sobre 21) y finalmente las con agente (2 sobre 21). Esto denota una cierta tendencia a vender productos sin agente, lo cual puede ser por el amplio soporte al protocolo SNMP en casi cualquier dispositivo interconectado, o bien por el alto costo de mantenimiento del desarrollo de un agente para cada plataforma soportada.
- Vemos productos sencillos que sólo se enfocan en la monitorización, como también otros que intentan ofrecer un paquete completo, ya sea en el mismo producto, o fomentando la venta de otros productos relacionados y vendidos por el mismo desarrollador.
- La existencia de un sólo producto meramente enfocado en Mainframe, nos deja entrever que no es un tipo de plataforma muy utilizado actualmente, o bien que existe una necesidad en dicho mercado para un soporte futuro.
- Nimsoft se promueve como un Software as a Service, denotando quizás una tendencia en este mercado.
- Varios productos traen soporte para virtualización, pero pocos realmente soportan todas las plataformas que existen hoy en día de forma nativa, lo cual muestra una fragmentación del mercado.
- Los productos complejos en su instalación suelen tener comunidades grandes de desarrolladores, pero sólo son usados por empresas pequeñas (ejemplo: Nagios Core).
- Los productos de código abierto analizados tienen ya sea una versión comercial más estable u ofrecen soporte técnico del producto como una forma de generar ingresos.

## Propuesta

En los capítulos anteriores, se hace mención sobre las tecnologías relacionadas a este trabajo, a fines de entender como funcionan tanto los productos de monitorización de servidores, el *peer-to-peer*, y cuales son los aspectos más importantes de la monitorización en sí. En la propuesta, se detallan las características que se proponen para este prototipo de producto, se explica como serían las funciones más importantes y necesarias, se analiza la factibilidad económica, técnica y operacional, finalizando con una mención de aquellos atributos que harían de este un producto único e innovador en el mercado.

## Aspectos generales

*Tecnologías a utilizar*

El producto utilizaría la tecnología P2P como base para permitir la comunicación entre los nodos monitorizados, de modo que no exista la necesidad de un nodo central dedicado a recolectar datos. Este aspecto eliminaría un eslabón crítico que poseen productos que actualmente se venden en este rubro, por lo que sería una característica innovadora, y muy valiosa como factor de venta.

Para poder hacer uso de P2P, necesitamos definir un algoritmo de asignación de rutas, tal como hemos visto en el capítulo 8. Entre todos los tipos de algoritmo estudiados, debemos elegir uno que se amolde a una red de servidores, por lo que se optaría por un enfoque estructurado. Entre las categorías de algoritmos estructurados, el que más se adecúa a la idea del producto es el de grado logarítmico con prefijo de asignación de ruta. Más adelante, en el apartado de funcionamiento, comentaremos los distintos factores que se han tenido en cuenta para seleccionar este tipo de algoritmo, como también las mejoras necesarias que debería incorporar para que funcione acorde a las necesidades del producto.

*Sistemas operativos soportados*

Tal como hemos visto en los capítulos 3 y 4 del marco teórico, el sistema operativo Linux es el candidato ideal para este tipo de productos. Por un lado, es el producto con mayor crecimiento del mercado en el mundo de los servidores, gozando de mucha atención corporativa. A su vez, el hecho de ser libre y, en muchas distribuciones, gratuito, nos permitiría sumar adeptos que quieran simplemente probar el producto, sin tener ningún costo adicional asociado.

Si bien Linux sería el sistema operativo elegido, cometeríamos un error al no darle soporte a otras plataformas como Windows o las variedades de UNIX, ya que esto convertiría al mismo en un producto menos tentador para las empresas. Por ende, otras plataformas y dispositivos sin Linux serían soportadas por medio del uso de SNMP (con *traps* si el dispositivo monitorizado lo soporta). A futuro, es necesario considerar el desarrollo de agentes nativos para aquellas plataformas que tengan una buena cuota de mercado, ya que cada nuevo nodo implicaría una mejora en la calidad de la red, debido al uso de tecnología P2P, lo cual comentaremos más adelante.

*Tipo de monitorización*

Dado que el producto estaría enfocado en un sólo sistema operativo inicialmente, aunque también dando la opción de monitorizar otros dispositivos de forma remota, el tipo de monitorización elegido sería el híbrido. Los beneficios de la monitorización con y sin agente serían más interesantes en un entorno P2P, dado que, por ejemplo, tanto la instalación como la actualización de componentes se podría realizar soportado por la misma red de nodos en funcionamiento. El nodo principal se encargaría de actualizar sus nodos hijos (intermediarios) y luego cada uno de estos haría lo mismo con sus nodos hijos, llegando a la totalidad de los nodos de la red en una fracción de tiempo, y sin necesariamente impactar por completo al nodo principal.

Por otro lado, cada nodo con agente se encargaría de monitorizar aquellos nodos sin agente cercanos (ya sea por estar en una misma subred, o bien por tener un identificador de nodo similar), minimizando así el tráfico entre estos dispositivos monitorizados remotamente contra el servidor principal. Cada dispositivo intermediario en la red simplemente transmitirá información resumida del estado de cada dispositivo junto con su propio estado, lo que reduciría el costo de cálculo y red para el conjunto de nodos en su totalidad.

*Tipos de nodos y responsabilidades*

La red contaría con tres tipos de nodos a nivel global. El primer nodo que ingrese a la red, o aquél que posea mayor puntuación en cuanto a recursos disponibles, sería conocido como el “nodo principal”. Como segundo tipo, nos encontraremos con los “nodos de primera línea”, quienes cumplirán las funciones del nodo principal, pero sólo para su árbol de hijos. Por último, nos encontramos con los nodos comunes.

Como se supone que cada nodo en la red comparte y hace uso de recursos ajenos para beneficio mutuo, los nodos de primera línea o el principal, asumen las responsabilidades de un nodo común (las heredan de abajo para arriba).

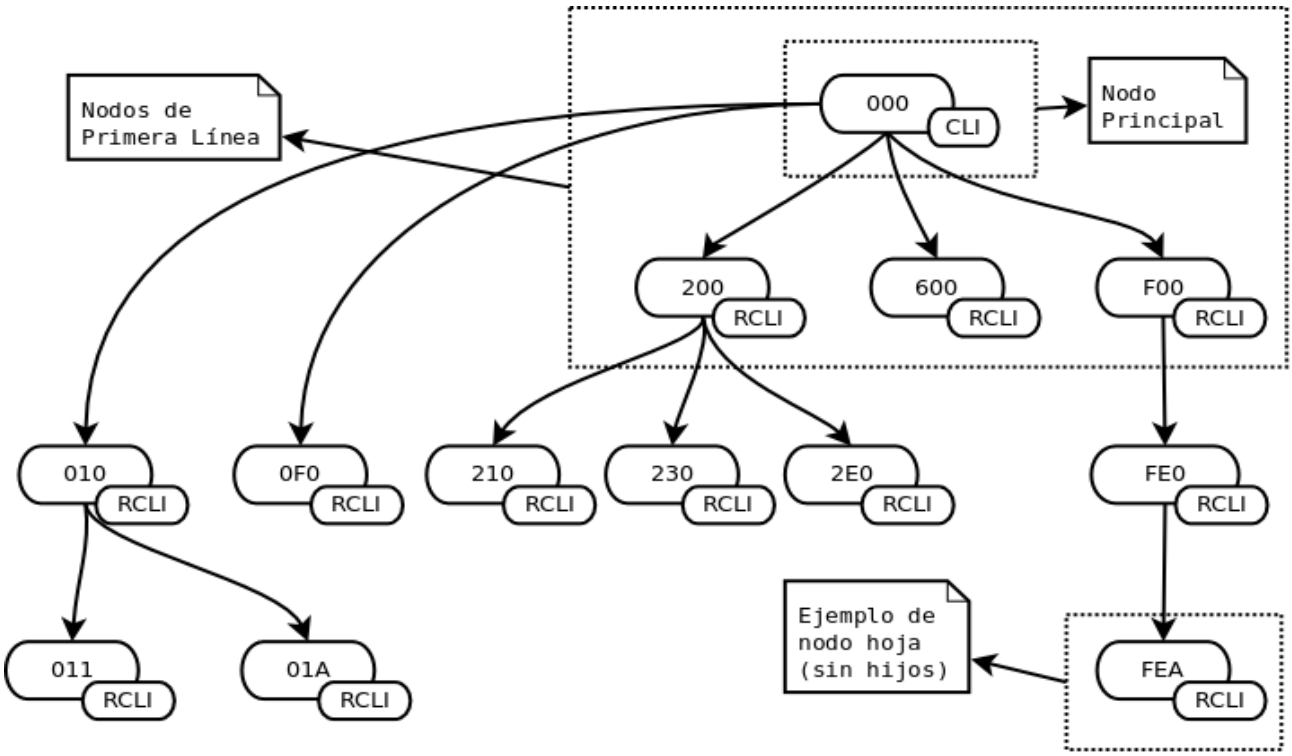
En el caso del “nodo principal”, su principal responsabilidad es la manutención de la red en un estado saludable, por lo que periódicamente se encargará de revisar la conexión entre los distintos nodos, como también reorganizar la posición de estos en el caso que sea necesario. Por otro lado, cuando un nodo ingrese a la red, deberá autenticar por medio del nodo principal, evitando así que cualquiera pueda ingresar a la red sin ser validado. Básicamente, si un nodo no existe en la tabla de nodos del nodo principal, entonces este no existe para ninguno de los pares que componen la red.

Finalmente, este nodo será el principal proveedor de estadísticas a la interfaz web (de forma periódica o bajo pedido) y a quienes accedan por medio de la CLI remota.

En cuanto a los “nodos de primera línea”, estos serán los responsables de reportar las alertas que provengan de sus hijos a la herramienta que se utilice para capturar los *tickets* de soporte, como también de enviar estadísticas al nodo principal (a pedido), calcular la puntuación de los nodos dentro de su árbol de hijos, y escuchar por *broadcast* la aparición de nuevos nodos interesados en unirse a la red.

A nivel global, todos los nodos tendrán como responsabilidad mínima el mantener actualizada su tabla de rutas (también conocida como tabla de nodos conocidos), a la vez de reportar cualquier salida de la red para que otro nodo pueda reemplazarlos en caso de ser necesario. Cada nodo poseerá una llave privada para poder intercambiar mensajes entre nodos, por lo que deberá asegurar esta clave y no relacionarse con máquinas que estén fuera de la red misma. Por último, en caso de tener hijos, serán responsables de mantener a estos informados de cualquier cambio que se produzca por encima de su nivel, evitando así la pérdida de sincronización de la red y aliviando el trabajo de mantenimiento al nodo principal.

*Vista panorámica – Topología de 3 niveles máximo*



*Consideraciones de mensajería entre nodos*

Como comentamos anteriormente, cada nodo posee una clave privada para poder firmar los mensajes que intercambiará con otros nodos cercanos de la red. Por ende, cada nodo también conoce la clave pública de estos nodos, y la almacena a fines de validar cada mensaje entrante.

Cada nodo hace uso de una vía directa hacia su nodo de primera línea, aunque si no logra contactar con este, puede dirigirse al nodo principal para que solvente el problema de conexión o lo reasigne a otro nodo de primera línea.

Por defecto, los nodos de primera línea dan soporte al nodo principal, esto quiere decir que todas las comunicaciones que vayan hacia el nodo principal, van a pasar primero por un nodo de primera línea (salvo en el caso de los nodos cuya primera línea es el mismo nodo principal).

En el caso de que el protocolo elegido de comunicación sea UDP, cada mensaje enviado esperará un mensaje de confirmación (ACK) por parte del nodo destino, antes de eliminar dicho mensaje de su tabla de alertas.

*Reglas de monitorización: tipos y prioridades*

Para poder brindar una monitorización inteligente, existirán distintos tipos de reglas que se podrán aplicar de forma manual, por un operador, o bien de forma automática dependiendo de los procesos/servicios que se estén ejecutando en el sistema.

Entre los tipos, nos encontramos con:

- Locales:
    - Sólo afectan al nodo en cuestión y son almacenadas por dicho nodo nomás
    - La información que va en el mensaje no es opcional (contiene información detallada de la alerta)
    - Tienen prioridad sobre los demás tipos de reglas
    - Se propagan del nodo principal al nodo elegido
- De tipo:
    - Funcionan si detectan la presencia de una condición determinada (aplicación X, base de datos, servidor web, software de cluster)

    - Todos los servidores las reciben por igual, y las almacenan indefinidamente
    - Tienen prioridad sobre las de categoría y globales
- De categoría:
    - Reglas que afectan a un grupo de servidores elegidos manualmente por el operador
    - Operador puede agrupar por tipo de entorno (producción, desarrollo, etc), por tipo de aplicación (Aplicación X, Aplicación Z), o bien por algún criterio propio relacionado al tipo de negocio
    - Sólo tienen prioridad sobre las reglas globales
    - Se propagan de nodo a nodo, sólo se almacenan si el nodo corresponde a la categoría
- Globales:
    - Afectan a todos los nodos por igual y son las de menor prioridad.
    - Se propagan de nodo a nodo, y se almacenan indefinidamente

*Seguridad de la red*

Como hemos visto en el capítulo 8, en el apartado sobre seguridad en las redes P2P, este tema no es menor, por lo que muchas consideraciones deben ser tomadas en cuenta ya sea para minimizar el efecto de posibles ataques o para mitigar de raíz las causas y evitar problemas.

Cada nodo que acceda a la red, deberá autenticar contra el nodo principal resolviendo un desafío con costo computacional (técnica de *Pricing*). Asumiendo que un nodo reingresa a la red y contacta un nodo conocido de una conexión previa, este será dirigido hacia el nodo principal para autenticar nuevamente. En el caso que un nodo no exista en las tablas del nodo principal, se considerará a dicho nodo como un desconocido y por ende no podrá compartir recursos hasta que se autentique. Esto será a su vez útil para la propagación de actualizaciones (de reglas o de software), ya que sólo serán aceptados paquetes provenientes de nodos autenticados, minimizando así la posibilidad de un ataque de *tampering*.

Dado que la estructura de la red poseerá un nodo principal, este podría ser víctima de un ataque de denegación de servicio. Ante la imposibilidad de evitar un ataque de ese tipo, la solución pasa porque cada nodo de primera línea pueda asumir rápidamente su lugar si detecta una desconexión,

como también pueda rearmar sus tablas de datos usando la información que provean los demás nodos de primera línea.

Por otro lado, cada paquete enviado en la red, será firmado con la clave privada del nodo origen, posibilitando que el receptor pueda validar (con la llave pública) si la información realmente proviene de quien dice el mensaje, haciendo mucho más complicado un ataque de MITM (*Man in the middle*, captura y/o modificación del mensaje).

A grandes rasgos, las consideraciones generales de la red en cuanto a seguridad son:

- Uso de cifrado con clave pública/privada para el intercambio de mensajes
- Mensajes con *hash* integrado para validar la integridad
- Pro-activo contra ataques de denegación, reemplazando nodos afectados en la red cuando se detecta que están siendo atacados (generando una alerta) y haciendo uso de filtrado de paquetes

Como tema relacionado, ante la posibilidad de una ruptura de red (*network split*), se considera que cada nodo que no encuentre ningún otro nodo por *broadcast*, genere un *ticket* de alerta para avisar de la situación. En caso de que requiera intervención manual de un operador, el mismo deberá conectar manualmente el nodo a una red existente. Caso contrario, el nodo actuará como un nodo principal de su red actual.

## Datos de estado

Para poder operar, cada nodo en la red necesita conocer cierta información mínima que le permita comunicarse con los demás pares, como también información de estado a fines de garantizar aspectos como la seguridad (de la red en sí y de cada nodo), y la disponibilidad de los recursos de la red. En este apartado, comentaremos detalles de la información que se almacena, los parámetros de la red, y como sería la estructura de las tablas de datos que tendrían los nodos.

### *Información almacenada en cada nodo*

Cada nodo recibe un "identificador de nodo" que es básicamente un valor hexadecimal único en la red. Este identificador será necesario para que otros nodos de la red puedan comunicarse con él, ya que sin el mismo las comunicaciones no alcanzarán el destino apropiado. A su vez, cada nodo ha de conocer quien es el "nodo principal" de la red, el "nodo de primera línea" asignado al árbol que pertenece (que puede ser el mismo que el nodo principal), como también el "nodo padre" (es decir, el nodo que se encuentra por encima de él en la red). Cada nodo que tenga hijos, conocerá los identificadores de estos a fines de informarles de cambios en la red cuando sean necesarios. Por último, cada nodo almacenará un valor de tiempo de permanencia con su actual identificador de nodo, que únicamente volverá a cero cuando éste cambie, y sólo se actualizará en una unidad después de cada mantenimiento de red.

### *Parámetros generales de la red*

Con el propósito de garantizar que cada nodo de la red actúe del mismo modo ante una situación particular, se hará uso de una serie de parámetros generales o globales. Estos parámetros serán conocidos por todos los nodos, sin importar su jerarquía, y consistirán en las variables que se mencionan a continuación:

- Máximo de niveles: por defecto un valor de 3, y será lo que defina el total de nodos posibles dentro de una red. Al ser de 16 nodos por nivel, se logra un total de 16^3 = 4096 nodos en la configuración por defecto. Los identificadores irán desde 000 a FFF.
- IP del WebCLI: el WebCLI (detallado más adelante en el apartado de Características Mínimas) tendrá un IP único accesible por los nodos de la red, y este será útil para que los nodos sepan a quien aceptar conexiones cuando se quiera hacer un cambio de configuración

(sólo cuando estos estén como “nodo principal”, que será el único autorizado para contactar el WebCLI).

- IP de herramienta de Tickets: otro IP será almacenado para saber a que IP deben conectarse los nodos de primera línea a la hora de reportar alertas. Este IP será conocido por todos los nodos, aunque sólo usado por los de primera línea (incluido el nodo principal).
- Tipo de tráfico: Esta variable podrá ser UDP, TCP o Mixto. En el caso que sea mixto, el sistema hará uso del tipo de conexión más conveniente acorde a la situación. Si el mensaje es urgente y/o los recursos de red son escasos, se enviará por UDP para minimizar el consumo.
- Mínimo de tiempo de mantenimiento y Máximo de tiempo de mantenimiento: ambos valores serán tenidos en cuenta por el “nodo principal” a fines de ejecutar los ciclos de mantenimiento. Por defecto cada ciclo tendrá un mínimo de 30 segundos, y un máximo de 180 segundos. El sistema decidirá, en base a la información de estado disponible, cuando es el mejor momento para pedir una actualización de estado a los nodos de la red, evitando así congestionar la misma y causar problemas en una tarea de mantenimiento.
- Máximo de alertas por nodo: un valor que representa la cantidad de alertas que un nodo mantiene en su memoria, a fines de evitar que el mismo colapse por alertas generadas. Este valor será usado por los nodos de primera línea, teniendo este valor multiplicado por la cantidad de hijos, dividido 2. Asumiendo 16 hijos, con un valor máximo de alertas de 10 por nodo, tendría un total de 16*10/2 entradas, o sea 80 en total.
- Máximo de reintentos de alertas: cantidad de veces que un nodo debe reportar la alerta si no consigue confirmación del nodo de primera línea. Este parámetro afecta a los nodos de primera línea de forma distinta, al tener estos que usar el valor contra la herramienta de *tickets*. En caso que el máximo sea alcanzado, se tomarán medidas para escalar el problema.

*Tabla: Nodos conocidos*

Esta tabla sería utilizada por todos los nodos y contendría información sobre los hijos del nodo (máximo de 16), el nodo padre, el nodo de primera línea asignado, y el nodo principal. En el peor de los casos, se mantendrían un total de 19 registros (un nodo de segundo nivel en un máximo de 3 niveles, por ejemplo).

- Identificador de nodo: haciendo referencia al ID del nodo conocido
- IP del nodo: el IP dentro de la red que identifica al nodo conocido
- Ultimo Ping: valor en segundos de la última vez que se pudo contactar al nodo
- Puntuación de nodo: valor calculado para la promoción de nodos
- Timeouts: por defecto 0, y se incrementa cuando un nodo falla en responder un mensaje, permitiendo la limpieza de nodos caídos
- Llave pública: necesario para el intercambio de información con el nodo en cuestión

La información de esta tabla es mantenida en cada actualización de red, y sirve como referencia para cuando un nodo reingresa a la red (ya que puede contactar al nodo vecino más cercano para unirse nuevamente).

*Tabla: Nodos*

La tabla "Nodos" es una tabla similar a la tabla de "Nodos Conocidos", con la salvedad que guarda información sobre la totalidad de nodos de la red. Esta sería considerada como una tabla maestra, y en una red con máximo de 3 niveles tendría un total de 4096 entradas.

- Identificador de nodo: hace referencia al ID asignado a dicho nodo
- IP del nodo: el IP dentro de la red que el nodo está utilizando
- Puntuación del nodo: valor calculado para la promoción de nodos
- Identificador de categoría: arreglo de ID (de categoría) a las que pertenece el nodo
- Hostname: nombre del nodo en su respectivo dominio
- Dominio: nombre del dominio al que pertenece el nodo
- Llave pública: necesario para el intercambio de información entre nodos
- Versión actual: Útil para poder identificar nodos aún no actualizados

Al posee información sobre toda la red, esta tabla solo estará disponible en su totalidad para el nodo principal y los nodos de primera línea, por lo que en una red de 3 niveles, habría un máximo de 16 réplicas de esta información, evitando así que la red pierda su estado actual en cualquier momento dado de tiempo.

*Tabla: Alertas*

Debido a que estamos trabajando con un sistema de monitorización, uno de los requisitos mínimos es que exista una colección de datos relacionados con las alertas. Por tanto, cada nodo poseerá una tabla con información de las alertas generadas por sí mismo y por nodos hijos. En los casos de nodos de primera línea, estos poseerán más información, al ser quienes reportan las alertas a las herramientas de *tickets*.

- Identificador de nodo: ID del nodo que creó la alerta
- Timestamp: Hora en segundos del momento en que se generó la alerta
- Identificador de regla: ID que hace referencia a la regla que se reporta, con fines de organización
- Valor actual: Valor reportado que causó la alerta
- Mensaje: Un campo opcional, que permitirá enviar más información en algunos tipos de alerta, como salidas de archivos de registros, o salidas de comandos predefinidos
- Criticidad: Un campo especial, para poder dar prioridad del mensaje sobre otros de menor importancia (ejemplo, un nodo caído)
- Enviado: Un campo de estado, permite al nodo que genera la alerta saber si su paquete fue recibido por el nodo de primera línea (en caso de UDP), y le permite al nodo de primera línea identificar si la alerta fue reportada a la herramienta de *tickets* correctamente.

Esta tabla se va limpiando automáticamente usando FIFO en caso de que el valor de “Máximo de alertas por nodo” se haya alcanzado. Aún así, los mensajes con criticidad alta reciben un trato especial a fines de que no sean descartados sin al menos un reporte exitoso.

*Tabla: Reglas*

A fines de que las alertas se generen, es necesario que el sistema de monitorización tenga reglas para saber que es lo que tiene que alertar. Por ende, esta tabla contiene información de que se monitoriza, por quien, cómo, entre otros aspectos.

- Identificador de regla: Un valor único que permite identificar la regla en las alertas
- Script de control: El comando necesario para verificar la alerta (puede ser la ubicación de un

*script*)

- Script de corrección: comando o ubicación de un *script* para aplicar una corrección rápida y predefinida a una regla en particular (puede ser nulo)
- Rango mínimo y Rango Máximo: valores que permiten identificar los rangos válidos en donde no existe un problema
- Valor problemático: Valor cuando se debe generar una alerta de criticidad baja porque existe un problema
- Valor crítico: Valor cuando se debe generar una alerta de criticidad alta porque el problema es grave
- Frecuencia: Valor en minutos que permita controlar cada cuanto se ejecuta la regla
- Reintentos: A fines de evitar falsas alertas, este valor permite saber cuantas veces ha de reintentar antes de generar una alerta
- Frecuencia de Reintentos: Valor en minutos en que deben realizarse los reintentos para evitar generar falsas alertas
- Alerta Crítica: un valor binario que indica si es necesario reintentar antes de enviar la alerta (0), o enviar en el primer intento con criticidad alta (1)
- Categorías afectadas: arreglo de identificadores que son afectados por la regla, puede ser nulo
- Tipos afectados: arreglo de identificadores de tipos de servidores que son afectados por la regla, puede ser nulo
- Identificador de nodo: En caso de que la regla haya sido creada para un nodo en particular, aquí estará el valor del identificador de dicho nodo, sino estará vacío

Todos los nodos poseen la tabla de reglas, con algunas salvedades: los nodos de primera línea poseen todas las reglas, mientras que los demás nodos sólo poseen las reglas globales y de tipos, mientras que sólo mantienen las reglas de categoría y locales que los afectan. Es decir, los nodos que no son de primera línea, no almacenan reglas locales de otros nodos, ni tampoco reglas de otras categorías a las que no pertenecen. Las reglas de tipos si las poseen, debido a que las mismas se accionan por condiciones (paquetes disponibles, procesos ejecutándose, existencia de un usuario,

entre otros), y estas pueden ser cumplidas en un momento puntual, pero no siempre (es decir, puede variar el “tipo” de servidor en el tiempo).

*Tabla: Nodos SNMP*

Al aplicar un tipo de monitorización híbrida, tendremos servidores con sistema operativo Linux usando un agente nativo, como también podremos tener servidores con cualquier sistema operativo que no hace uso del agente, y aprovechar las bondades remotas del SNMP. Para estos casos, haremos uso de una tabla de nodos que controlan a estos nodos SNMP, permitiendo distribuir la carga de controlar dispositivos como impresoras, dispositivos de red, o incluso páginas web, instancias de bases de datos remotas, etc, entre los nodos de la red P2P.

- Identificador del nodo SNMP: Permite identificar dentro de la red el nodo SNMP de forma única.
- IP del nodo SNMP: La dirección IP que se utilizará para monitorizar el recurso
- Identificador del nodo maestro: En este caso sería el nodo (que posee agente) encargado de monitorizar este nodo remoto

Cada recurso que es monitorizado por SNMP recibe un identificador secuencial atado a la cantidad máxima de nodos según el total de niveles de la red (es decir, en una red de 3 niveles, podrá haber un máximo de 4096 nodos con agentes y 4096 nodos sin agentes por cada nodo). A la hora de identificar dicho recurso, se utilizará el valor del identificador del nodo maestro que lo monitoriza y el valor del nodo SNMP mismo. Ejemplo, si el nodo es 01F, los nodos SNMP administrados por él tendrán un valor 01F000, 01F001 y así consecutivamente.

Esta tabla es mantenida en su totalidad por los nodos de primera línea, aunque los demás nodos solo poseerán información sobre aquellos nodos SNMP que tengan asignados a sí mismos (cuyo Identificador de nodo coincida con el Identificador de nodo maestro).

*Otras tablas*

Las descripciones de algunas tablas no merecen mayor detalle debido a que son simples relaciones de un identificador con un nombre o bien porque son funcionales a otras tablas y funciones de mayor importancia. Algunas de estas tablas son:

Categorías: contiene el nombre de la categoría, y el identificador

Tipos: contiene el identificador del tipo de nodo, el nombre asignado y el condicional utilizado para verificar si un par debería ser considerado dentro de dicho grupo

Apagado programado: contiene información sobre los nodos afectados, una fecha de inicio y final del evento, como también un campo de información para futura referencia de porqué sucedió.

## Funcionamiento

*Justificación del algoritmo de asignación de rutas*

El algoritmo a utilizar está basado en PRR (mencionado en el apartado de "Algoritmos de asignación de rutas" en el capítulo de "Tecnología P2P") con la diferencia que utilizará identificadores con valores hexadecimales, permitiendo así más nodos por nivel, como también la adaptación de algunas de sus funciones en relación al tipo de red (de empresa) al que se destina el producto. A su vez, posee funciones de mantenimiento de red, haciéndolo más complejo que PRR.

Se opta por PRR, más que nada por ser el algoritmo de asignación de rutas que sirvió como precursor de otros conocidos (Tapestry, Pastry, Chord), dado que posee buenos conceptos de base para armar redes P2P. Entre las similitudes con el algoritmo a implementar, hace que cada objeto posea un nodo raíz (su padre), es afectado en contextos de alta tasas de ingresos/egresos (aunque en menor medida por algunas mejoras propuestas), permite ubicar a cualquier nodo en la red usando una tabla de rutas de tamaño fijo, y el diseño hace necesario que se conozca la totalidad de la red para funcionar (responsabilidad que tienen los nodos de primera línea). Entre las diferencias, se agrega la posibilidad de insertar/eliminar nodos de forma dinámica (por lo que se añade un algoritmo de mantenimiento) sin que se requiera reordenar al resto (salvo en casos de promoción de nodos).

PRR es un algoritmo de asignación de grado logarítmico con prefijo de asignación de rutas, lo cual implica dos aspectos:

- Cada mensaje enviado va saltando de nodo en nodo usando los valores de izquierda a derecha de su identificador hasta coincidir con el destinatario correcto. Si el nodo 0000 enviase un mensaje al nodo 9999, entonces en primer caso, buscaría un nodo conocido cuyo prefijo sea 9 (primer valor de la izquierda). Una vez que este nodo, supongamos 9104, recibe el mensaje, lo reenvía al nodo cuyo prefijo sea 99 (primer y segundo valor de la izquierda). Este nuevo nodo, 9943, hace lo mismo, y lo envía hacia el nodo 9995 que cumple con un prefijo 999, quien hace nuevamente la misma tarea y lo reenvía al nodo destinatario 9999. El camino entonces fue de 0000 → 9104 → 9943 → 9995 → 9999, por lo que en cada nodo el prefijo fue de ayuda para acercarse al nodo destino.
- El grado logarítmico se consigue porque la cantidad de niveles se puede delimitar

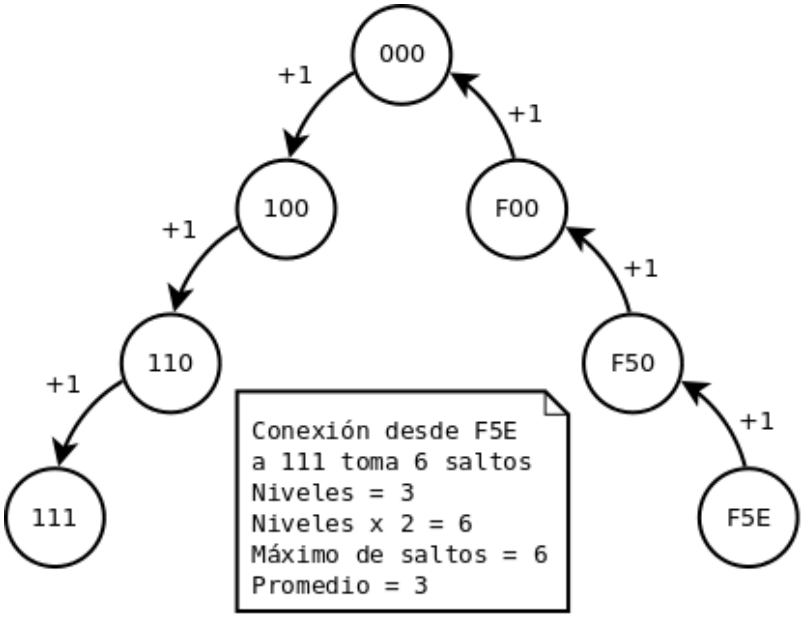
fácilmente por el logaritmo “$\log_{16} N$”, siendo N el total de nodos necesarios y 16 en relación a la identificación hexadecimal (de 0 hasta 9, luego A hasta F).

Para poder comprender la selección de este tipo de algoritmo, debemos analizar el tipo de entorno al que se enfrentaría el producto:

- Va a operar en una red de servidores, por lo que no es muy común tener una alta tasa de ingresos/egresos todo el tiempo, minimizando así los problemas por estabilidad de la red, e incluso reduciendo la necesidad de un mantenimiento continuo.
- Dado que no se espera una alta rotación de nodos, la posibilidad que la red colapse por la desconexión de muchos nodos al mismo tiempo es mínima, y sólo existiría en el caso de un evento programado o una auténtica catástrofe. Si fuese por un evento programado, se podrían “apagar” zonas de servidores (ya sea identificándolos por tipo, categoría o incluso subredes) a fines de evitar que se empiecen a generar alertas innecesarias.

Si analizamos el algoritmo según los parámetros establecidos por Koegel, podemos justificar cada ítem como se detalla a continuación:

- Total de saltos: con un diseño de Z niveles, cualquier nodo puede alcanzar a otro en un máximo de Z x 2 saltos, con un promedio de Z saltos.




- Estabilidad: como se ha dicho anteriormente, al estar pensado para una red de empresas, las salidas e ingresos serán atípicas por lo que no han sido una prioridad en la elección del

algoritmo

- Grado de nodo: cada nodo poseerá información de sus hijos (máximo de 16), su nodo padre, el nodo de primera línea que le corresponde, y el nodo principal. Un total máximo de 19 relaciones en todos los nodos, salvo en el caso de los nodos de primera línea que conocen la totalidad de la red por medio de la tabla de Nodos.
- Convergencia: si bien cada nodo puede llegar a cualquier otro nodo de la red en un máximo de "Niveles x 2" saltos, esto no será necesario ya que las comunicaciones se realizarán desde abajo para arriba (nodos hacia nodo de primera línea) o de arriba hacia abajo (nodos de primera línea a hijos o árbol de hijos) y no de forma distribuida entre nodos de distintos árboles, salvo cuando haya que reorganizar la red. Es decir, la comunicación es posible (la convergencia se da) pero no será usada en la comunicación típica.
- Grado de tolerancia a fallos: La tabla de "Nodos conocidos" permite restablecer rápidamente el estado de la red si un nodo de importancia desaparece, por lo que la tolerancia a fallos es alta para este tipo de algoritmo.
- Costo de mantenimiento: La subdivisión de tareas entre los nodos de primera línea disminuyen el tráfico general de la red, aunque el tráfico resultante puede ser mayor al que exista en una estructura de cliente-servidor (propio de las redes P2P).
- Balance de carga: Los nodos de primera línea, incluido el nodo principal, son los que tendrán mayor carga de trabajo, pero es necesario una topología de este estilo para minimizar la cantidad de tráfico de la red en total, sino cada nodo reportaría de forma independiente a las herramientas de *tickets*, y el resultado sería un exceso de tráfico, produciendo incluso peores resultados que una red con modelo cliente-servidor.

*Algoritmo de ingreso de nuevo nodo*

En el capítulo sobre tecnología P2P, comentamos la existencia de una función esencial para este tipo de redes que recibe el nombre de "*bootstrapping*", y consiste básicamente en identificar los nodos que ingresan, autorizar su acceso y hacer que estos descubran los demás nodos o pares de la red.

Para este algoritmo, se consideran dos tipos de configuraciones, una cuando el nodo es nuevo a la red y otra cuando el mismo se conecta nuevamente.

Cada nodo que ingresa tiene consigo mismo información necesaria para el proceso. Por un lado está su dirección IP (IPNodo), su nombre en el dominio (NombreNodo), y un conjunto de nodos que conoce (NodosConocidos) ya sea por una configuración manual de un administrador, por haber sido parte de la red anteriormente, o bien por encontrar nodos conectados en la subred. A su vez, para poder efectivamente formar parte de la red, debe conocer al nodo que está al mando de la red (NodoPrincipal), a fines de hacer contacto con él y poder autenticar su sesión. Una vez que el nodo ingresa a la red, obtiene su identificador (IDNodo) y posteriormente se ejecuta un algoritmo de mantenimiento (AgregarNodo) para hacerle saber a los nodos cercanos que tienen un nuevo compañero.

En el caso que un nodo no tenga NodosConocidos, este los ubicará (si existen) realizando una búsqueda por *broadcast*, para luego poder conocer quien es el nodo principal y así poder autenticar en la red.

```
Proceso = IngresoNodo
Entorno = IPNodo, NombreNodo, NodosConocidos
// Se buscan nuevos nodos ni bien se comienza, que se agregan
// a los nodos conocidos
NodosConocidos << BuscarNodosPorBroadcast
Para cada {Nodo} en {NodosConocidos}
    Contactar Nodo
    Si {Nodo responde} entonces
          Consultar quien es NodoPrincipal
          Autenticar con NodoPrincipal
          NodoPrincipal ejecuta AgregarNodo(DatosNodo)
          Si { NodoPrincipal asignó IDNodo } entonces
                Salir de bucle Para
          FinSi
    FinSi
FinPara
Si { IDNodo es nulo } entonces
    // Si entramos acá, entonces el nodo se convierte en principal
    IDNodo = 0
    NodoPrincipal = NombreNodo
    GenerarTicket(NuevaRedDisponible,IPNodo,NombreNodo)
FinSi
```

En presencia de redes interconectadas, será necesario que todas puedan intercambiar información por medio del puerto de la aplicación de monitorización (con una regla en el cortafuegos, si existiese). Por otro lado, sería necesario que los administradores configuren previamente un listado

de nodos conocidos en el caso que estén agregando nodos en otras redes, ya que el *broadcast* sólo se ejecuta en la subred actual y puede que se genere múltiples redes de monitorización en un entorno distribuido, sin que ese sea necesariamente el objetivo de quien los administra.

*Algoritmo de salida anunciada*

El ingreso de nodos es tan importante como la partida de los mismos cuando nos referimos a redes P2P. Esto se debe a que, como se ha mencionado repetidas veces, cada nodo aporta recursos propios en la red, lo que permite que las tareas se puedan ejecutar de manera más rápida y eficiente.

En el caso de las salidas de los nodos, nos podemos encontrar con salidas anunciadas, en donde el mismo nodo sabe que se va a desconectar y, por ende, avisa a sus pares (a modo de saludo de despedida), como también encontraremos salidas no anunciadas, en los casos que un nodo simplemente pierda conectividad con la red, sea apagado bruscamente o simplemente la aplicación de monitorización falle, entre otras posibles causas.

La idea de una salida anunciada es, por un lado, evitar un alto costo de mantenimiento para el total de la red, como por otro, ser solidario con el resto de los nodos para evitar que estos pierdan conectividad con los demás participantes.

El algoritmo para esta tarea implica que cuando un nodo sale (IDNodoActual), revisa si tiene hijos (NodosHijos). Si los tiene, elige a uno de estos (IDNodoHijo) en relación a la puntuación (basado en recursos disponibles) para que lo suceda en su posición, y luego le avisa al resto de los hijos para que estos actualicen su tabla de nodos conocidos (IDPadre). En el caso que el nodo que sale no sea el nodo principal, este deberá avisar sobre su partida, y el nuevo nodo que ocupa su posición se pondrá en contacto también para actualizar el estado de la red.

```
Proceso = SalidaAnunciada
Entorno = IDNodoActual, NodosHijos
Si { IDNodoActual sale } entonces
    Si { NodosHijos no es nulo } entonces
          IDNodoActual elige nodo con más puntos de NodosHijos
          IDNodoActual avisa a NodosHijos que IDNodoElegido es el IDPadre
          IDNodoActual envía información de estado a IDNodoElegido
          Si { IDNodoActual no es NodoPrincipal } entonces
                IDNodoActual avisa a NodoPrincipal de su salida
                IDNodoActual avisa a IDNodoPadre
                IDNodoElegido contacta a NodoPrincipal actualizando estado
          FinSi
    SiNo
```

```
            IDNodoActual avisa a IDNodoPadre
        FinSi
        // Sea que tenga hijos o no, al final se desconecta
        IDNodoActual desconecta
    FinSi
```

Si bien la función implica un consumo de tiempo, ya que un nodo debe elegir su sucesor previo a desconectarse, se ha de tener en cuenta que todos estos datos ya son mantenidos por los nodos padres. En el caso que la desconexión provenga de un nodo hoja (sin hijos), entonces el proceso es más sencillo ya que sólo avisa a su nodo más próximo (el nodo padre).

*Algoritmo de salida por desconexión*

Cuando un nodo sale por su propia cuenta, el aviso proviene de sí mismo. Si el nodo sale de forma inesperada, una salida por desconexión será advertida por algún nodo cercano ante el intento de intercambio de datos, o bien por algunos de los nodos de primera línea cuando se hace una actualización del estado de red.

Para este problema tendremos varios escenarios, siendo estos cuando el nodo caído es el nodo principal, un nodo de primera línea o cualquier otro tipo de nodo. La diferencia radica básicamente en que si el nodo principal desaparece, los nodos de primera línea han de ser capaces de restablecer el estado de la red entre ellos (la suma de la información que estos poseen, es igual a la información que posee el nodo principal). En el caso que sea un nodo de primera línea, el nodo principal deberá decidir quien es el sucesor en base a la información contenida en su tabla de nodos. Y, si el nodo caído no es de primera línea (tampoco principal, por inclusión), entonces el accionar será simplemente reorganizar la red al nivel del nodo afectado.

```
Proceso = NodoCaido
Entorno = IDNodoActual, IDNodoCaido, IDNodoPrimeraLinea, IDNodoPrincipal
// Si el nodo reportado como caído es el principal
Si { IDNodoCaido = IDNodoPrincipal } entonces
    IDNodoActual avisa a su IDNodoPrimeraLinea
    IDNodoPrimeraLínea verifica que no sea un apagado programado
    IDNodoPrimeraLinea valida desconexión de IDNodoCaido
    Si { IDNodoCaido está desconectado } entonces
          GenerarAlerta(NodoCaido,IDNodoCaido)
          IDNodoPrimeraLinea avisa a sus nodos del mismo nivel
          Nodos de primera linea eligen el de mayor puntuación
          IDNodoElegido pasa a ser el nodo de primera linea con más puntos
          IDNodoElegido promueve un nodo hijo para que lo reemplace
          IDNodoElegido pide información a otros nodos de primera línea
```

```
            IDNodoElegido ejecuta MantenimientoRed
        SiNo
            // Se genera una alerta por problemas de interconexión entre nodos
            GenerarAlerta(ProblemaRed,IDNodoActual,IDNodoCaido)
        FinSi
    // Si el nodo reportado como caído es de primera línea
    Sino Si { IDNodoCaido es Nodo de Primera Línea } entonces
        IDNodoActual avisa a IDNodoPrincipal
        IDNodoPrincipal verifica que no sea un apagado programado
        IDNodoPrincipal valida desconexión de IDNodoCaido
        Si { IDNodoCaido está desconectado } entonces
            GenerarAlerta(NodoCaido,IDNodoCaido)
            IDNodoPrincipal elige reemplazo
            IDNodoPrincipal ejecuta MantenimientoRed
        SiNo
            // Se genera una alerta por problemas de interconexión entre nodos
            GenerarAlerta(ProblemaRed,IDNodoActual,IDNodoCaido)
        FinSi
    // Si el nodo no es de primera línea (por ende, tampoco principal)
    Sino
        IDNodoActual avisa a su IDNodoPrimeraLinea
        IDNodoPrimeraLinea valida desconexión de IDNodoCaido
        Si { IDNodoCaido está desconectado } entonces
            GenerarAlerta(NodoCaido,IDNodoCaido)
            IDNodoPrimeraLinea elige reemplazo
            IDNodoPrimeraLinea avisa a IDNodoPrincipal
            IDNodoPrincipal ejecuta MantenimientoRed
        SiNo
            // Se genera una alerta por problemas de interconexión entre nodos
            GenerarAlerta(ProblemaRed,IDNodoActual,IDNodoCaido)
        FinSi
    FinSi
```

Si un nodo genera varias alertas de desconexión con otro par en particular, el administrador podrá crear una regla de exclusión entre ambos nodos, en el caso que esto sea así por diseño de la red misma (por reglas de seguridad como cortafuegos o zonas desmilitarizadas).

*Algoritmo de cálculo de puntos por nodo*

La puntuación que recibe cada nodo en la red, sirve de base para la promoción de nodos en situaciones de mantenimiento de red, o bien cuando un nodo se desconecta. Estos puntos que obtiene cada participante, son calculados en base a los recursos disponibles en un periodo

determinado de tiempo (configurado por el administrador) de cada sistema, entre los que se destacan los ciclos de procesador (CPU), la memoria virtual (RAM+área de intercambio o SWAP), el total de horas que un sistema estuvo encendido (conocido como *uptime*), el nivel de carga del sistema, y la latencia de red que posee en relación a la herramienta de WebCLI, en caso que se haga uso de la misma.

El administrador de la red podrá configurar que relación tiene el cálculo de puntos en base a cada recurso del sistema, lo cual será influenciado por el tipo de red. En el caso de una red con un amplio número de nodos, se buscará por un lado que el sistema mejor puntuado tenga disponible mucha memoria virtual (para poder almacenar más información de monitorización en memoria), como también que un sistema posea un prolongado tiempo de actividad (ya que indica que es un nodo que no se apaga usualmente). En redes más pequeñas, quizás el nivel de carga del sistema sea el valor que más se priorice, dado que sólo recibirán el papel de "Nodo Principal" (o de primera línea) aquellos nodos que no estén muy ocupados en su actividad rutinaria. Como sea, la decisión quedará de lado del administrador, y este decidirá que es lo que prima para cada escenario, pudiendo modificar esta configuración en vivo, y logrando que los nodos se reorganicen automáticamente (tras un mantenimiento de red o una promoción otorgada por alguno de los nodos de primera línea).

A diferencia de otros algoritmos, en este caso no entraremos en el cálculo mismo de la puntuación (ya que eso será personalizado para cada red), sino más bien el proceso que realizará cada nodo para recabar la información necesaria que permita obtener la puntuación final.

```
Proceso = CalcularPuntos
Entorno = IDNodoActual, IDNodoPadre, [IPWebCLI]
// Obtenemos primero información del sistema
Obtener estadísticas de CPU, Memoria Virtual, Nivel de carga
Obtener cantidad de días que nodo actual estuvo funcionando
Si { IPWebCLI no es nula } entonces
    Obtener latencia de conexión con WebCLI
FinSi
Obtener configuración de cálculo de puntos (definida por administrador)
PuntosNodoActual = valor obtenido usando la configuración con los parámetros
Si { IDNodoPadre no es nula } entonces
    Si { IDNodoActual tiene hijos } entonces
          Envia puntos de nodos hijos a IDNodoPadre
    FinSi
    Informar PuntosNodoActual a IDNodoPadre
FinSi
```

La puntuación obtenida no afectará la red hasta que no surja un evento (como un mantenimiento

de red periódico), y sólo lo hará si los puntos obtenidos superan a otro nodo con mayor jerarquía en la red. Como cada nodo que tenga hijos pasa la información a su nodo padre, en algún momento la información llega al nodo principal quien tiene información sobre el total de la red, como también al nodo de primera línea correspondiente, quienes harán uso de dichos puntos en caso que sea necesario una promoción de nodo (por desconexión o mantenimiento).

La elección de que parámetros son más importantes según la red en la que se haga uso del producto, deberán ser definidos con sumo cuidado, ya que estos podrían provocar promociones muy seguidas luego de cada mantenimiento de red, algo que no permitiría la estabilidad y sería poco útil.

*Algoritmo de promoción de nodo*

La promoción de nodo es una de las características más importantes agregadas sobre la base del algoritmo PRR, ya que permite que un nodo pueda ascender (o descender) en cuanto a su ubicación dentro de la red, lo que cambia completamente su diseño y permite que se vaya adaptando a las necesidades actuales en cada momento.

La idea de este algoritmo es permitir que los nodos con mayor puntuación dentro de la red, determinada por sus recursos y la configuración particular de cada escenario, se ubiquen en las posiciones más altas, ya que son los mejores candidatos a ocupar el rol de nodos de primera línea, en el cual se incluye el nodo principal también.

El proceso es iniciado por los nodos de primera línea, quienes detectan que hay candidatos que pueden ser promocionados. Esto lo hacen comparando las puntuaciones ya disponibles, que son enviadas periódicamente por medio del algoritmo de mantenimiento de red. Si un nodo elegido ha sido reubicado recientemente (reflejado por el valor de permanencia almacenado en el mismo), este será descartado para favorecer la estabilidad de la red y evitar una excesiva rotación de nodos.

```
Proceso = PromocionNodo
Entorno = PuntuacionesNodos
// NodoPrimeraLinea solo posee información sobre su propio árbol de nodos
NodoPrimeraLinea detecta candidato a promoción por puntuación
NodoPrimeraLinea comunica potencial promoción al NodoPrincipal
// NodoPrincipal examina mejores candidatos de cada árbol
Si { NodoPrincipal no tiene mejor candidato } entonces
    NodoPrincipal revisa tiempo de permanencia de NodoElegido
    Si { tiempo de permanencia es menor a 3 } entonces cancela
    Si { NodoElegido no tiene última versión de software } entonces cancela
    NodoPrincipal comunica a NodoElegido su nuevo IDNodoPromovido
```

```
    NodoPrincipal comunica a NodoReemplazado su nuevo IDNodo
    // Se intercambian información de hijos y padre (nodos conocidos)
    NodoElegido contacta a NodoReemplazado e intercambian datos de estado
    NodoElegido contacta a nuevos hijos y padre para actualizar estado
    NodoReemplazado contacta a nuevos hijos y padre para actualizar estado
    NodoPrincipal contacta a NodoPrimeraLinea con actualización de estado
    NodoPrincipal actualiza su propia tabla de Nodos
SiNo
    NodoPrincipal envía mensaje de rechazo de promoción
FinSi
```

Como cada nodo de primera línea conoce a los hijos de su árbol, el proceso logra que en unas pocas ejecuciones los nodos con mayor puntuación actual estén situados por encima de la jerarquía. Recordemos que el nodo principal también es considerado como un nodo de primera línea, por lo que puede lanzar una orden (a sí mismo) para reemplazar a cualquier nodo de primera línea que no pueda justificar su posición actual.

*Algoritmo de mantenimiento de red*

El proceso de mantenimiento de red es invocado periódicamente por el nodo principal, a fines de mantener el estado de la red lo más actual posible. Como este proceso toma tiempo, la idea es que no se ejecute múltiples veces en paralelo, ni que tampoco haya una ejecución tras otra sin descanso, ya que el consumo de tráfico de la red aumentaría drásticamente.

Cada vez que se ejecuta un mantenimiento de red, el nodo principal puede decidir tomar acciones preventivas (como promover nodos) o bien de corrección (como reasignar nodos SNMP cuyos nodos maestros están caídos, o eliminar nodos caídos).

```
Proceso = MantenimientoRed
Entorno = TimestampInicio
// Revisamos que no haya un mantenimiento previo aún en ejecución
Si { MantenimientoRed no está activo } entonces
    // Actualización de estado
    NodoPrincipal solicita a nodos hijos una actualización de estado
    NodosPrimeraLinea envían pedido de CalcularPuntos a cada nodo hijo
    NodoPrincipal actualiza su tabla de datos con la información recibida
    // Corrección de nodos SNMP huerfanos (nodos padres caídos)
    Si { ExistenNodosSNMPhuerfanos } entonces
          NodoPrincipal asigna hijos SNMP a otro nodo disponible
    FinSi
    // Promociones por mantenimiento
    Si ( NodoPrincipal encuentra promociones potenciales ) entonces
```

```
            NodoPrincipal ejecuta algoritmo de PromocionNodo
        FinSi
    SiNo
        // Alertamos al operador que dos mantenimientos se encontraron
        // para que pueda adjustar el tiempo entre mantenimientos periódicos
        GenerarAlerta(ProblemaMantenimiento,TiempoMinimoMantenimiento)
    FinSi
```

*Algoritmo de actualización automática*

Una herramienta de monitorización necesita tener todo su entorno al día para funcionar de la mejor forma. El entorno puede (y va a) cambiar constantemente, con la finalidad de mejorar el control que se tiene sobre la red. Estos cambios pueden venir desde el código mismo de la aplicación de monitorización, las reglas con las que se generan las alertas, o incluso las secuencias de comandos que usan los monitores en cada nodo.

En el caso de esta red P2P de monitorización, necesitamos aún más que cada nodo esté al día constantemente, ya que en cualquier momento dado pueden ser promovidos a una posición de primera línea, en donde deberán tener control sobre los demás nodos hijos de dicho árbol.

Por otro lado, el hecho de que la red sea P2P, permite sacar provecho a las conexiones mismas para propagar las actualizaciones desde el nodo más importante (principal), hacia los nodos hojas (sin hijos), sin que todo el trabajo se deba hacer manualmente por un operador, o que la carga se sitúe en un nodo servidor (como pasaría en una red cliente-servidor).

En este proceso se contempla la posibilidad de que un operador agregue una regla nueva por un lado, o bien que instale una actualización del programa en sí. La diferencia se da en que la regla puede no ser aceptada por todos los nodos, mientras que las actualizaciones se aplican de forma general, sin distinción alguna.

Cuando se actualizan reglas globales, de categoría o de tipo, la información se propaga nodo a nodo, mientras que las reglas locales viajan del nodo principal al nodo elegido nomás. En cuanto a las reglas de categoría, estas son eliminadas del nodo local si el nodo no es parte de dicha categoría, salvo el caso que el nodo sea de primera línea (estos nodos deben poseer toda la información de su árbol en caso que falle el nodo principal).

Finalmente, las actualizaciones de software se propagan nodo a nodo. En cada caso, la actualización se aplica luego de hacer una copia de seguridad de la versión actual. Si al intentar reiniciar sesión con la nueva versión falla, entonces se restaura la versión anterior del *software* y esa

actualización no se propaga a los siguientes nodos para evitar un efecto de cascada. Ante el primer fallo, se genera una alerta para que un operador corrija el problema.

```
Proceso = AutoActualizacion
Entorno = IDNodoActual, TipoActualizacion, Contenido
Si { TipoActualizacion = regla } entonces
    Si { TipoRegla = [Global,Categoría,Tipo] } entonces
          Si { IDNodoActual tiene hijos } entonces
                Para cada hijo
                      Enviar regla (Contenido)
                FinPara
          FinSi
          // Descartamos las reglas que no son de la categoría del nodo local
          // Sólo aplica a los nodos que no son de primera línea
          Si { TipoRegla = Categoría &&
                IDCategoriaRegla != IDCategoriaLocal } entonces
                Si { IDNodoActual no es de primera línea }
                      Descartar regla
                FinSi
          FinSi
    SiNo
          // Reglas sólo locales
          NodoPrincipal envía regla local a nodo elegido
          NodoPrincipal envía regla local al nodo de primera línea del nodo
          elegido
    FinSi
SiNo Si { TipoActualizacion = software } entonces
    Realizar copia de seguridad de versión actual
    Instalar actualización
    Reiniciar sesión // Sin desconectar de la red, sólo recargar software
    Si { Nueva versión funciona } entonces
          Si { IDNodoActual tiene hijos } entonces
                Enviar actualización a cada hijo
          FinSi
    SiNo
          Restaurar copia de seguridad
          Reiniciar sesión
          GenerarAlerta(ActualizacionFallida,IDNodoActual,MensajeError)
    FinSi
FinSi
```

El uso de la misma red P2P para propagar las actualizaciones (de reglas o de *software*) permiten obtener un beneficio extra en cuanto al consumo de recursos. Si la red es muy grande, el beneficio es mayor, ya que todo el trabajo queda repartido en la totalidad de los nodos.

*Algoritmo de agregado de nodos*

Cada vez que se agrega un nodo a la red, es necesario ejecutar ciertas acciones para que el nuevo integrante pueda interactuar con el resto de los participantes. El "Nodo Principal" es quien toma la mayor parte de actividad en esta tarea, ya que él tiene la responsabilidad de autenticar nuevos nodos y de darles la bienvenida. El único caso en que un nodo sea rechazado, es por no tener espacio en la red, algo que está delimitado por el valor de máximo de nodos (relacionado con el máximo de niveles de los parámetros generales de la red).

```
Proceso = AgregarNodo
Entorno = DatosNodo
NodoPrincipal revisa total de nodos en la red
Si { TotalNodos <= Máximo de nodos } entonces
    // La elección se da balanceando la carga, el que menos tenga
    // más probabilidades tiene de recibir un nuevo nodo en su árbol
    NodoPrincipal elige un nodo de primera línea para que agregue el nuevoNodo
    NodoPrincipal envía DatosNodo a nodo de primera línea elegido
    Nodo de primera línea agrega nuevoNodo en su tabla de Nodos
    // El nodo de primera línea presenta nuevoNodo a su padre o hijos
    Nodo de primera línea avisa a nodos cercanos sobre nuevoNodo
    Nodo de primera línea envía cambios de nodos al NodoPrincipal
    NodoPrincipal entrega IDNodo a nuevoNodo
SiNo
    GenerarAlerta(SinEspacioRed,TotalNodos)
FinSi
```

*Algoritmo de monitorización de nodos SNMP*

El algoritmo de agregado de nodos se ejecuta cuando un nodo quiere unirse a la red, mientras que en el caso de los nodos SNMP, el pedido se inicia por un administrador del sistema. Este operador puede intentar agregar un nodo SNMP que ya conoce, o bien puede utilizar los recursos de la red P2P para escanear por nuevos dispositivos, distribuyendo la tarea, minimizando los esfuerzos y maximizando el alcance del escaneo.

```
Proceso = AgregarNodoSNMP
Entorno = [IPNodoSNMP]
Si { IPNodoSNMP es vacío } entonces
    // Se busca un escaneo automático por parte del nodo principal
    NodoPrincipal filtra sus NodosHijos
    Para cada {Hijo} de {NodosHijos} hacer
          Hijo elige un nodo de su árbol de hijos de forma aleatoria
          Hijo envía al nodo elegido un pedido de escaneo de dispositivos
```

```
            Si { Resultado no es vacío } entonces
                  Reportar nodos SNMP encontrados al NodoPrincipal
            FinSi
        FinPara
    SiNo
        // NodoPrincipal es el encargado de revisar que el nodo exista
        NodoPrincipal contacta IPNodoSNMP
        Si { IPNodoSNMP existe } entonces
            IPNodoSNMP se agrega a NodosNuevosSNMP
        SiNo
            Operador recibe un mensaje de error al agregar nodo SNMP
        FinSi
    FinSi
    // Finalmente se asignan los nodos SNMP a distintos nodos no SNMP
    // El nodo principal busca que cada nodo de primera línea esté balanceado
    // Cada nodo de primera línea balancea la carga de nodos SNMP por nodo
    Para cada {NodoNuevoSNMP} de {NodosNuevosSNMP} hacer
        NodoPrincipal compara NodoNuevoSNMP con los nodos que ya posee
        NodoPrincipal agrega NodoNuevoSNMP a su tabla nodos SNMP
        NodoPrincipal elige un NodoPrimeraLinea balanceando carga
        NodoPrimeraLinea elige un nodo de su árbol balanceando carga
        NodoElegido agrega NodoNuevoSNMP a su tabla
        NodoElegido envía primer reporte a NodoPrimeraLinea
    FinPara
```

En ambos casos, al inicio hay un intento de contacto con el nodo SNMP para validar que realmente existe, ya sea por escaneo o por contacto directo. Una vez que se confirma que existen nodos para agregar, el nodo elegido para tal tarea (que es seleccionado balanceando la carga para que cada nodo tenga una cantidad pareja de nodos SNMP), añade el nodo a su propia tabla, y hace su primer monitorización. El nodo principal y el nodo de primera línea agregan la referencia a su tabla también para saber quien se encarga de cada nodo SNMP.

Algoritmo de generación de alertas

En el algoritmo de generación de alertas, cada nodo debe ser capaz de enviar la información a su nodo de primera línea. Dado el caso que quien reporte una alerta sea uno de los nodos de primera línea, entonces este deberá directamente reportar a la herramienta de tickets.

```
Proceso = GenerarAlerta
Entorno = IDAlerta, [parámetros]
Si { Nodo actual es de primera línea } entonces
    NodoDestinatario es la herramienta de tickets
SiNo
```

```
    NodoDestinatario es el nodo de primera línea del actual nodo
FinSi
Para (Reintentos ← 0) hasta (Maximo de Reintentos de alerta) hacer
    Enviar datos de alerta a NodoDestinatario
    Esperar confirmación de recibido
    Si { NodoDestinatario confirma recepción } entonces
          Guardar alerta actual a nivel local como enviada
          Salir del bucle Para
    FinSi
FinPara
Si { Confirmación no recibida } entonces
    Guarda alerta a nivel local como envío fallido
    Enviar datos de alerta a nodo principal
    Si { NodoPrincipal confirma generación de alerta } entonces
          Marcar alerta como enviada
    FinSi
FinSi
```

La tabla de alertas se va vaciando periódicamente en base a la cantidad máxima de alertas configuradas por nodo. Las alertas que primero se eliminan, son aquellas que fueron enviadas, luego las más antiguas. Este orden se da para evitar, en el mejor de los casos, la eliminación de alertas que aún pueden ser reales.

Por último, las alertas se almacenan en cada nodo a fines de tener un listado de sucesos recientes que puedan ser analizados por un operador (mediante la CLI local), en caso que este tuviese que tomar acción por una alerta recibida.

Algoritmo de monitorización

Para lograr que el sistema de monitorización sea inteligente, es necesario que cada regla posea validaciones previas a fines de evitar que, por ejemplo, una regla se ejecute en un nodo que no corresponda, o bien que la misma regla se esté ejecutando múltiples veces en el mismo nodo.

Al comienzo del proceso, se valida que la regla no esté en ejecución (esto solo sucedería en caso que la frecuencia de reintento por la cantidad de reintentos sea mayor a la frecuencia general de la regla en sí), que el identificador de nodo sea igual al del nodo actual (esto solo afectaría a los nodos de primera línea que conservan reglas de su árbol de hijos), que la categoría coincida, y si el condicional del tipo de nodo de la regla aplica para el nodo actual. A su vez, si el nodo está programado como apagado, se evita la ejecución de reglas en su totalidad.

Luego de todas esas validaciones, la regla se ejecuta y se verifica si el valor resultante está

dentro de los rangos mínimos o máximos aceptables. En caso que no sea así, se contrasta dicho valor con el valor crítico o problemático. Si el valor es mayor o igual al problemático, entonces se verifica si el valor es mayor o igual a crítico o bien si la regla es crítica de por sí (alerta crítica es verdadera), para lo cual se procede a enviar la alerta. En caso que la alerta no sea crítica, entonces se procede a reintentar nuevamente. Una vez que se llega al límite de reintentos, si la alerta no fue cancelada en el proceso y aún no se envío, entonces se procede a enviar la misma.

```
Proceso = MonitorizarRegla
Entorno = DatosRegla
// Primero evitamos que dos reglas se ejecuten al mismo tiempo
Si { IDRegla ya está en ejecución } entonces (cancelar)
Si { IDNodoActual está en apagado programado } entonces (cancelar)
// Verificaciones previas, se obtiene IDNodoRegla, IDCategoriaRegla,
// IDTipoRegla
Si { IDNodoRegla != IDNodoActual } entonces (cancelar)
Si { IDCategoriaRegla != IDCategoriaNodoActual } entonces
          CategoriaErronea ← 1
FinSi
Si { IDTipoRegla no es vacía } entonces
    Ejecutar CondiciónTipo
    Si { IDTipoRegla no es válido para el nodo actual } entonces
          TipoErroneo ← 1
    FinSi
FinSi
// Una vez que validamos que la regla corresponde al nodo
Si { TipoErroneo != 1 || CategoriaErronea != 1 } entonces
    Para (TotalReintentos ← 0) hasta (ReintentosRegla) con Paso 1 hacer
          Ejecutar script de control y obtener ValorResultado
          Si { ValorResultado no está dentro del Rango (Min/Max) } entonces
                Si { Script de corrección no es nulo } entonces
                      Ejecutar script de corrección
                      Obtener nuevo ValorResultado
                FinSi
                Si { ValorResultado >= ValorProblematico } entonces
                      Si { AlertaCritica es verdadero ||
                      ValorResultado >= ValorCritico } entonces
                            GenerarAlerta(IDRegla,ValorResultado)
                            AlertaEnviada = verdadero
                      Sino
                            Esperar tiempo de FrecuenciaReintentos
                      FinSi
                FinSi
          SiNo
```

```
                AlertaCancelada = verdadero
                Salir de bucle de reintentos
            FinSi
        FinPara
        Si { AlertaEnviada no es verdadero && AlertaCancelada es falso} entonces
            GenerarAlerta(IDRegla,ValorResultado)
        FinSi
    FinSi
```

Con este algoritmo conseguimos que una regla no crítica sea válida correctamente antes de generar una alerta, evitando así un falso positivo. Por otro lado, considera los tipos y categorías, impidiendo que una regla se ejecute en un nodo que no corresponde.

Esta función se ejecuta en múltiples hilos por el mismo proceso del software de monitorización, haciendo posible que varias reglas sean validadas en simultáneo.

*Algoritmo de apagado programado*

Las redes de servidores requieren de un mantenimiento constante. Este trabajo usualmente es realizado por la misma gente que administra los sistemas, y en su labor pueden llegar a generar las condiciones necesarias para que un monitor detecte una falla en un sistema. Por este motivo, es necesario proveer a los administradores de herramientas para que puedan avisarle al *software* de monitorización que las alertas son falsas y deberían ser descartadas.

En algunos de los productos que fueron analizados previamente en este trabajo, existen soluciones incorporadas para ofrecer esta función, como las listas de *blackout* en IBM ITM o las excepciones en Nagios, por lo que es un requerimiento a tener en cuenta.

La implementación del apagado programado en este caso, aplicaría al nodo en sí, y dado que cada par posee acceso al RCLI, un usuario operador podría activar el apagado de forma local, aunque también podría hacerlo desde la CLI del nodo principal. La única diferencia desde donde se activa está en que localmente se hace de forma inmediata, mientras que desde el nodo principal se puede programar el apagado para un tiempo futuro y por un periodo predeterminado (horario de inicio y horario de fin). En ambos casos, se podría ingresar un texto a modo de justificación para futura referencia.

```
Proceso = ApagadoProgramado
Entorno = IDNodoBlackout, FechaInicio, [FechaFin]
Si { IDNodoActual es IDNodoPrincipal } entonces
    Guardar IDNodoBlackout, FechaInicio, FechaFin, Motivo
    Enviar datos al nodo IDNodoBlackout
```

```
SiNo
    // Entonces el apagado programado es local
    Guardar IDNodoBlackout, FechaInicio, FechaFin, Motivo
    Enviar datos al IDNodoPrimeraLinea
    IDNodoPrimeraLinea envía datos al IDNodoPrincipal
FinSi
```

Ya sea un apagado programado local o remoto, los únicos nodos que se enteran del estado son los nodos principal, de primera línea y el nodo mismo afectado. Los demás nodos pueden seguir interactuando con el nodo "apagado", y en caso de que se generen alertas de problemas referidas a éste, las mismas serán descartadas por el nodo de primera línea ni bien las reciba.

Otros algoritmos

Algunos algoritmos no son descriptos ya sea por su simplicidad (como validar la conexión entre dos nodos) o porque no representan una acción importante, sino más bien de soporte a otros eventos más relevantes al trabajo desarrollado (como la búsqueda de nodos por *broadcast*). La idea de este apartado fue comentar aquellos algoritmos que realmente importan en este producto, evitando entrar en detalles innecesarios.

## Características mínimas

En el capítulo de Software de Monitorización, del marco teórico, se analizaron las características que poseen los productos que hoy en día se venden en el mercado. Entre estos, encontramos características comunes como la presencia de una interfaz de control y la existencia, en la mayoría de los casos, de reglas de monitorización por defecto. En este apartado se describe como serían estos aspectos, y las razones de tal elección.

*Interfaz de control*

La mayoría de los productos analizados en el marco teórico, poseen una interfaz web como panel de control. Esto se debe, más que nada, a que un navegador puede ser accedido desde cualquier sistema operativo con un navegador, e incluso desde teléfonos móviles o tabletas digitales, ampliando la posibilidad de acceso a cualquier tipo de usuario. En este caso, el uso de un sistema web para controlar la monitorización, incrementa la complejidad del sistema, ya que en un entorno de cliente-servidor, el servidor sería el encargado de servir la página web. En cambio, en esta red P2P que estaremos creando, el nodo web pasaría a ser ya sea el "nodo principal" o cualquier otro "nodo de primera línea" siguiendo esta idea. Dado que la presencia de un servidor web implica ciertas consideraciones en cuanto a seguridad y consumo de recursos, se opta porque la interfaz web exista, pero que esta sea provista por un nodo específico en todo momento, siendo éste no necesariamente un "nodo de primera línea", e incluso pudiendo no ser parte de la red P2P de monitorización. La interfaz web recibirá el nombre de WebCLI, ya que su función esencial será la de enviar comandos al "nodo principal" ante las acciones de un usuario en la misma.

En el caso que un administrador no desee hacer uso de la WebCLI, existirá la herramienta CLI por defecto, que consistirá básicamente en un entorno de línea de comandos que permita configurar el "nodo principal", como también enviar órdenes a los demás nodos de la red. Si el administrador no cuenta con acceso remoto al "nodo principal", podrá hacer uso del RCLI, que será como el CLI, pero donde la ejecución de los comandos se hace de forma remota. La diferencia entre ambas utilidades es el entorno, mientras CLI ejecuta comandos a nivel local, RCLI permite ejecutar comandos de forma remota para controlar la red.

Por otro lado, no se haría uso de una aplicación (gráfica) de escritorio como existe para algunos de los productos de monitorización analizados. El motivo de esta decisión se basa en la dependencia del entorno gráfico en sí y la dificultad de cubrir la mayor cantidad de plataformas posibles. De este

modo, teniendo la CLI, cualquier administrador podrá controlar la red simplemente conectándose a un nodo cualquiera por SSH, sin otros requisitos.

*Categorías de monitorización por defecto*

De los distintos productos analizados, nos encontramos con un gran porcentaje de categorías que son monitorizadas por defecto en todos los sistemas. Si bien los monitores varían según la plataforma analizada, a nivel de hardware se suele monitorizar los mismos aspectos, siendo estos CPU, memoria RAM, dispositivos de red, placas de fibra (HBA), discos internos y fuente de energía. A nivel de software, y tal como lo hemos visto en el capítulo de monitorización, se controlan por defecto los servicios de sistema (también llamados demonios), aplicaciones específicas como bases de datos o productos de *clustering.* En estos últimos dos casos, varía mucho dependiendo de quien sea el fabricante y si este posee o no otros productos propios (ejemplo, IBM ITM monitoriza por defecto su propia base de datos DB2).

A continuación se listan los monitores más importantes que vendrían por defecto para que el producto pueda ser aceptado por el mercado en una primera etapa:

- CPU: Uso de CPU y funcionamiento correcto de cada procesador
- Memoria virtual: Uso de memoria RAM y memoria de intercambio, funcionamiento correcto de cada módulo de memoria
- Nodo caído: Nodo que no responde a un *ping* o conexión a un puerto específico
- Sistemas de archivos: Uso de los sistemas de archivos, disponibilidad (si puede escribirse en el mismo o si está en modo de sólo lectura)
- Red: Uso de dispositivos de red, velocidad de cada placa como también si están disponibles o no, y el estado del enlace (si recibe y envía al mismo tiempo, o si está en modo Half Duplex, por ejemplo)
- Seguridad: usuario bloqueado, gran cantidad de accesos fallidos a un usuario cualquiera o administrador (root), chequeo de integridad de archivos de sistema o configuración
- *Cluster*: control de estado del *cluster*, cambio de nodo (*failover)*, falla en un recurso o grupo de recursos, disociación de nodos (*split brain*)
- Instancias web y de base de datos: disponibilidad y estado actual en caso que el producto lo

soporte

- Hardware remoto: impresoras, *switches*, *routers*, cortafuegos, *proxys*, antivirus por hardware, entre otros

## Características destacadas

Ya mencionadas las características mínimas y necesarias en el capítulo anterior, debemos centrarnos en las características que diferencien el producto de la competencia, haciendo énfasis en el uso de la tecnología P2P e incluso aportando una mirada distinta sobre la monitorización inteligente de alertas y sobre la monitorización por SNMP de nodos remotos de forma distribuida.

### *Geometría del diseño*

Debido a que este producto tiene como base el uso de una red P2P para compartir los recursos y poder monitorizar todos los nodos participantes, la geometría del diseño está pensada con ese objetivo, lo que permite obtener ciertos beneficios que no están presentes, por ejemplo, en una red de tipo cliente-servidor.

Entre los beneficios nos encontramos con la posibilidad de que cualquier nodo de la red pueda desaparecer (incluso el "nodo principal" o cualquier "nodo de primera línea") sin necesariamente obtener un resultado catastrófico. Por el propio algoritmo implementado para la red (analizado previamente), los nodos pueden reconstruir la misma con información histórica obtenida de sus nodos superiores y nodos hijos. Incluso en el peor caso que una gran cantidad de nodos desapareciera en un intervalo corto de tiempo, los mecanismos de reconstrucción podrían volver todo a la normalidad en cuestión de minutos, aprovechando las tablas de nodos conocidos de los pares que aún están conectados a la red.

Por otro lado, si bien la carga de los nodos está distribuida, algunos poseen más carga que otros debido a sus responsabilidades. Esta carga, asumiendo que la red forma un árbol, se centraliza sobre la parte superior de la estructura (en los "nodos de primera línea", incluido el principal), aunque todos los nodos del árbol poseen una carga mínima debido a la necesidad de monitorizar la salud de la red en su totalidad (cada nodo es responsable de la red, en cierto sentido).

### *Monitorización inteligente de alertas*

Parte del título de este trabajo final de graduación hace referencia a la monitorización inteligente de servidores. Cuando hablamos de inteligencia, no hacemos referencia a que el sistema sea capaz de entender las situaciones y resolverlas de igual forma que lo haría un ser humano, sino más bien que pueda evaluar más condiciones en el entorno antes de generar una falsa alerta sobre un problema. Por este motivo, cada regla tendrá un rango mínimo y máximo aceptable (ver apartado de

Datos de Estado), que permitirá definir cuando el monitor está en una situación aceptable. En caso que un monitor detecte que una regla está fuera de estos valores, tendrá que analizar si la situación actual está contemplada dentro del valor problemático o de un valor crítico. Si posee un *script* de corrección, entonces intentará solucionar el problema y obtener un nuevo valor actualizado. En el caso que el valor sea sólo problemático, el sistema aplicará el valor de reintentos y su frecuencia para validar si realmente la situación se sigue dando (se genera una alerta) o ya dejó de existir (se evita una alerta innecesaria). Caso contrario, si la regla considera que el valor crítico ha sido alcanzado, entonces la alerta se generará sin reintentos, ya que se asume que es necesaria una acción inmediata para remediar la situación. Como escenario particular, algunas alertas podrán ser configuradas como críticas (ejemplo, un nodo caído), haciendo que el monitor alerte sin importar otros parámetros de la configuración.

*Monitorización SNMP por cercanía*

El producto propuesto hace uso de un agente nativo en servidores con sistema operativo Linux, pero asumir que una red de empresa va a tener un entorno completamente homogéneo, es prácticamente una utopía. En una red cualquiera, es normal que convivan distintos tipos de plataformas (Windows, Linux, UNIX, entre otros) como también existan dispositivos de red, impresoras, dispositivos de seguridad (cortafuegos, antivirus, antispam). Por este motivo, se hace necesario considerar la monitorización SNMP, haciendo que el producto se transforme en uno de monitorización híbrida.

En una red de cliente-servidor, el nodo que cumple el rol de servidor es quien tiene los monitores para revisar los dispositivos que son monitorizados por SNMP. Dado que en el caso de una red P2P contamos con recursos disponibles en cada nodo de la red, es necesario y oportuno distribuir la carga de estos monitores entre los distintos participantes.

A la hora de agregar un nodo, el administrador podrá escanear la red buscando nuevos dispositivos aún no monitorizados, o bien podrá agregar uno específico ingresando la dirección IP o el nombre del dispositivo según como aparezca en el DNS. Una vez que el dispositivo es detectado y se cargan las reglas básicas para controlarlo, el nodo principal se encargará de buscar entre los nodos de primera línea, algún candidato dentro de sus árboles de hijos que pueda hacerse cargo de la monitorización. De este modo, la carga de los dispositivos SNMP quedará distribuida en la totalidad de la red, y podrá realizarse en intervalos de tiempo más cortos que los que se darían en una red de cliente-servidor, en donde sólo el nodo principal revisa la totalidad de dispositivos.

## Análisis FODA

Dado que el producto aún no ha sido desarrollado, sino que por medio de este trabajo se plantea como un prototipo, el análisis presente a continuación se sitúa en el año 2013 y detalla las fortalezas, oportunidades, debilidades y amenazas con las que se encontraría el software en el caso hipotético que ya estuviera a la venta en el mercado.

| Internas | Externas |
|---|---|
| **Fortalezas** | **Oportunidades** |
| F1 - Uso de tecnología P2P<br>F2 - Monitorización híbrida con soporte para dispositivos de forma distribuida<br>F3 - Monitorización inteligente basada en distintos tipos de reglas y parámetros flexibles<br>F4 - Independencia de entornos costosos | O1 - Mercado estancado en innovación<br>O2 - Crecimiento uso de servidores con sistema operativo Linux |
| **Debilidades** | **Amenazas** |
| D1 - Mayor consumo de red<br>D2 - De código abierto<br>D3 - Ingresos sólo por soporte técnico<br>D4 - Temores sobre seguridad en entornos P2P | A1 - Empresas competidoras de renombre<br>A2 - Uso creciente de nubes computacionales (Cloud Computing)<br>A3 - Resistencia al cambio |

*Fortalezas*

**F1 – Uso de tecnología P2P**

La tecnología P2P permite que el software pueda ofrecer características como redundancia, mayor nivel de tolerancia a fallos y nuevas posibilidades en lo respectivo a cubrir redes extensas.

**F2 – Monitorización híbrida con soporte para dispositivos de forma distribuida**

El soporte de forma distribuida, haciendo uso de monitorización híbrida, permite reducir la carga de los nodos principales de una red de monitorización, y repartir la misma entre los distintos nodos, mejorando el tiempo de respuesta ante eventos extraordinarios.

**F3 – Monitorización inteligente basada en distintos tipos de reglas y parámetros flexibles**

El uso de algoritmos optimizados permite reducir falsos positivos, a la vez de aprovechar las reglas para segmentar de distintas formas las redes monitorizadas (por aplicación, por tipo de servidor, por zona de red, o personalizado por el administrador), incrementando la eficiencia en el uso de recursos (tanto humanos, como de los sistemas informáticos).

**F4 – Independencia de entornos costosos**

La inexistencia de nodos especiales de monitorización permite reducir drásticamente el costo de la infraestructura, pudiendo aprovechar los recursos libres de los nodos activos en distintos momentos del día, evitando así tener capacidad ociosa en la red.

*Oportunidades*

**O1 – Mercado estancado en innovación**

El escaso nivel de competencia que existe en el mercado de monitorización ofrecen una oportunidad excelente para innovar y destacarse con un producto distinto.

**O2 – Crecimiento uso de servidores con sistema operativo Linux**

El uso creciente de servidores con sistema operativo Linux, nos presenta un amplio mercado al que ofrecerle un producto nuevo, como también da indicios que el crecimiento no va a frenar en el futuro inmediato según lo comentado en el capítulo sobre este sistema operativo.

*Debilidades*

**D1 – Mayor consumo de red**

Tal como se ha mencionado durante la propuesta, el producto tiene un mayor consumo de recursos de red debido al uso de una arquitectura basada en P2P, lo que hace necesario ciclos de mantenimiento cuyo costo es mayor al que suele tener una arquitectura de tipo cliente-servidor.

**D2 – De código abierto**

En el capítulo sobre el sistema operativo basado en el kernel de Linux, comentábamos como éste (de código abierto) tuvo sus inicios en laboratorios y ordenadores viejos. Esto se debe principalmente a que las empresas no solían confiar en productos de código abierto en ese entonces. Si bien el panorama ha cambiado notablemente desde 1991, algunas empresas grandes suelen rechazar el uso de software de código abierto, a menos que la empresa que lo soporte tenga una buena imagen en el mercado y brinde un buen servicio.

**D3 – Ingresos sólo por soporte técnico**

Debido a que el producto es de código abierto, vender el mismo en conjunto con un servicio de soporte técnico sería la opción más correcta (como lo hacen Red Hat o Novell con sus

distribuciones de Linux). Este aspecto representa una debilidad, ya que como se ha dicho en el ítem anterior, las grandes empresas sólo contratan a los grandes o conocidos del mercado, mientras que pocas empresas medianas o pequeñas suelen optar por adquirir sus servicios de nuevos emprendimientos.

**D4 – Temores sobre seguridad en entornos P2P**

Si bien el uso del P2P se plantea como una fortaleza, tal como se mencionó en el capítulo dedicado a esta tecnología, la misma presenta ciertos desafíos en lo respectivo a la seguridad. Por una parte, las empresas grandes tienen políticas de filtrado de tráfico que podrían impactar el producto, como también suele tener una connotación negativa el uso del término P2P en lo que respecta a aplicaciones (debido a problemas legales de aplicaciones como Napster).

*Amenazas*

**A1 – Empresas competidoras de renombre**

Tal como se menciona en el capítulo de análisis de productos competidores, empresas como IBM o HP se encuentran ofreciendo productos en este segmento, lo que representa una amenaza importante debido a sus grandes presupuestos y su amplia cartera de clientes.

**A2 – Uso creciente de nubes computacionales (*Cloud Computing*)**

Empresas como Amazon (AWS), Microsoft (Azure) y Google poseen productos enfocados en el *Cloud Computing*, por lo que muchos de sus clientes están tomando interés en esta plataforma. Si bien no es certero que las empresas dejen sus centros de datos para migrar por completo a estos entornos alquilados, se debe tener en cuenta como una amenaza posible a mediano plazo.

**A3 – Resistencia al cambio**

Es un hecho que las personas se resisten al cambio, del mismo modo que la mayoría de las empresas se acostumbran a un producto y luego intentan no migrar a menos que sea estrictamente necesario. Este problema, aparejado con que el producto está basado en tecnología P2P, representa una debilidad importante.

Estrategias

**E1 – Uso de tecnología P2P y menor costo total como aspecto de venta**

Ante un mercado estancado en cuanto a innovación, el uso de tecnología P2P podría usarse como un

factor de venta diferenciador respecto a la competencia. Aspectos como la monitorización inteligente ayudarían al producto, pero dado que muchos competidores ya hacen uso de esa palabra, quizás no sería suficiente como para interesar a un potencial cliente. La independencia respecto a entornos costosos, sin necesidad de servidores exclusivos de monitorización, darían otro aspecto de relevancia para quienes analicen la adquisición del producto.

**E2 – Menor costo total de infraestructura**

La promoción indirecta del sistema operativo Linux, que proviene del uso en crecimiento del mismo, sumado a la independencia de entornos costosos promocionada por este producto, más la inexistencia de un pago por el producto (sino más bien por el servicio de soporte técnico), permitirían vender una solución que tenga un menor costo total de infraestructura, lo cual sería tentador para empresas pequeñas y medianas, aunque también para empresas grandes buscando reducir costos. Al ser de código abierto, una empresa cualquiera podría probar el producto sin compromiso, y luego contratar el soporte técnico en caso de que les interese.

**E3 – Mejorar algoritmos de mantenimiento para minimizar consumo de recursos de red**

Los algoritmos de mantenimiento de red son los principales responsables de que el producto tenga un alto consumo de red, pero a la vez son críticos para que una aplicación P2P se comunique con todos sus nodos. La mejora continua de estos algoritmos, estudiando los distintos escenarios reales de clientes, permitiría ir optimizando el código para reducir la huella del producto en cuanto al consumo de red y de otros recursos del sistema.

**E4 – Incentivar el aporte comunitario de extensiones**

La mayoría de las comunidades de código abierto hacen uso de sus comunidades principalmente para reportar problemas, aunque algunas también permiten el aporte por medio de extensiones, como es el caso del navegador web *Mozilla Firefox*, o incluso un producto de monitorización como *Nagios Core*. El aporte de la comunidad muchas veces se suele traducir en mejoras que terminan en el producto comercial (previo acuerdo con el desarrollador), lo que redunda en un beneficio tanto para la comunidad como para la empresa que vende el producto.

**E5 – Ciclos de desarrollo corto y visibles, con uso de licencia abierta**

Una diferencia entre los productos de código abierto y cerrados, es que los primeros permiten que la comunidad pueda ir probando versiones nuevas a medida que salen, lo que reduce drásticamente los costos en cuanto a la prueba de nuevas funciones (aclaración, sólo para entornos que no son de

producción). El hacer que los ciclos de desarrollo sean cortos y visibles a la comunidad, permitiría ganarse a la comunidad y limitar la posible entrada de competidores conocidos que, utilizando el código (ya que es abierto), puedan crear un producto similar y venderlo por su cuenta. Otro aspecto a considerar, sería el uso de una licencia abierta que no permita una posterior limitación de libertades, como lo es la licencia GPL v3.

**E6 – Soporte técnico en base a los estándares del mercado**

Al ofrecer soporte técnico como una forma de generar ingresos por el producto, el servicio ofrecido debería adecuarse a los estándares de las empresas grandes que actualmente compiten en este mercado, como IBM o HP, lo que motivaría a que clientes potenciales consideren migrar.

**E7 – Promocionar beneficios del P2P**

El término P2P ha recibido mala publicidad por casos como *Napster* (mencionado en el capítulo dedicado a esta tecnología), lo que podría ser problemático al intentar ofrecer el producto. Una campaña promocionando los beneficios del P2P, en cuanto a redundancia, reducción de costos, más las ventajas asociadas con este tipo de producto, podrían ayudar a disminuir este miedo, e incluso reducir la resistencia al cambio por parte de las empresas.

**E8 – Soporte de *Cloud Computing***

Si bien el *Cloud Computing* es una tecnología muy reciente, las empresas podrían interesarse más en el futuro inmediato por lo que considerar el desarrollo de una versión que funcione sobre entornos privados, o bien a modo *ad-hoc* en nubes públicas, sería una buena forma de limitar el impacto negativo e incluso obtener un nuevo mercado para explotar.

**E9 – Enfoque sobre seguridad en entornos P2P**

A la hora de promover el producto, sería importante hacer hincapié en las medidas de seguridad comentadas en el apartado de aspectos generales de la propuesta. Esto permitiría reducir el miedo sobre la tecnología P2P como también promover el uso de políticas más flexibles que acepten a productos así en las redes corporativas.

Matriz FODA

| | **Fuerzas** | **Debilidades** |
|---|---|---|
| **Oportunidades** | Estrategias FO<br>E1 – Uso de tecnología P2P y menor costo total como aspecto de venta<br>E2 – Menor costo total de infraestructura | Estrategias DO<br>E2 – Menor costo total de infraestructura<br>E3 – Mejorar algoritmos de mantenimiento para minimizar consumo de recursos de red<br>E9 – Enfoque sobre seguridad en entornos P2P |
| **Amenazas** | Estrategias FA<br>E4 – Incentivar el aporte comunitario de extensiones<br>E5 – Ciclos de desarrollo corto y visibles, con uso de licencia abierta<br>E6 – Soporte técnico en base a los estándares del mercado | Estrategias DA<br>E7 – Promocionar beneficios del P2P<br>E8 – Soporte de *Cloud Computing* |

## Factibilidad

No se puede realizar un análisis detallado de factibilidad del producto propuesto, debido a que presenta múltiples condiciones que juntas son impredecibles, por lo que se opta por un enfoque en donde se analiza cada aspecto importante de forma separada. En el siguiente apartado, la conclusión definirá la posibilidad de que un producto así pueda ser creado, focalizándose más bien en las preguntas formuladas al comienzo de este trabajo.

### *Factibilidad técnica*

Lo primero que se debe estudiar es si realmente se puede usar P2P para un producto de este tipo, ya que ninguna empresa ha intentado (al menos, de público conocimiento) siquiera incursionar en esta tecnología. La respuesta a esta pregunta, y basándonos en los datos recolectados a través de este trabajo, es si. Existen múltiples librerías disponibles, en distintos lenguajes de programación, para desarrollar redes P2P fácilmente, como también adaptarlas a productos específicos, siempre y cuando se sepa con antelación que tipo de algoritmo de asignación de rutas es el indicado, lo cual también hemos estudiado en el presente trabajo.

Por otro lado, los requisitos de cada sistema que participe en la red serían mayores con respecto a otros productos de la competencia, ya que cada nodo podría comportarse como nodo principal en cualquier momento dado. Aún así, la inexistencia de un nodo central dedicado, reduce notablemente el costo de infraestructura, permitiendo a pequeñas empresas monitorizar su entorno con su actual red de servidores, sin necesidad de cambio alguno.

En cuanto a los algoritmos mejorados, como hemos visto anteriormente, estos no presentan mayor dificultad de ser implementados en cualquier lenguaje existente actualmente. La elección del lenguaje impactaría de forma directa en el tiempo de desarrollo del producto, ya que algunos disponen de librerías de código abierto que facilitan enormemente la tarea a la hora de programar una aplicación P2P.

### *Factibilidad operacional*

En lo operacional, nos encontramos con un dilema al no poder predecir si las empresas optarían por un producto innovador, en detrimento de las opciones ya disponibles (y conocidas) en el mercado. La ventaja de que el producto sea distribuido como de código abierto, habilitaría a cualquier administrador para probar el sistema sin ningún compromiso, funcionando como un factor

de venta crítico tal cual lo fue Linux en su momento.

El otro aspecto a tener en cuenta, sería la migración en sí de sistemas de monitorización a este producto, por lo que se deberían considerar opciones de asistentes virtuales para permitir exportar reglas y alertas de productos conocidos como Nagios, ITM, Zenoss, entre otros. Esto reduciría los tiempos de migración y permitiría reducir las barreras propias que se generan en una empresa por miedo a los cambios.

*Factibilidad económica*

A nivel económico, es imposible predecir el costo de desarrollo, ya que se busca que el modelo de negocio apunte al soporte y sea sustentado principalmente por un desarrollo de código abierto. El costo de fallar en un producto tal sería cercano a nulo, solo provocando la propia pérdida de tiempo (y el costo asociado) por parte de los participantes. Como todo proyecto de código abierto, este debería ser lanzado a público una vez que la idea base está lista (la descrita en este trabajo), para que luego la misma comunidad sea quien encamine el proyecto, junto con la participación de las empresas que le den soporte técnico al producto de manera comercial.

## Conclusión

Al comienzo de este trabajo, se plantearon varios interrogantes que han sido estudiados en el marco teórico y luego analizados en profundidad dentro de la propuesta.

Primero es necesario destacar que en la teoría es posible crear un software de monitorización de servidores que haga uso de una estructura distinta a la típicamente usada de cliente-servidor. Nada impide que se adopte la tecnología P2P para este tipo de tareas, algo que queda demostrado claramente con la aplicación de la misma a cientos de aplicaciones, tal como hemos estudiado en el marco teórico. Las dudas restantes se disipan al saber que existen algoritmos de asignación de rutas compatibles para el producto propuesto, por lo que vale afirmar que si, es posible crear un producto P2P de monitorización de servidores.

En cuanto a la segunda pregunta, los eslabones críticos que se presentan en las redes no pueden ser eliminados por completo, ya que siempre necesitaremos un nodo que actúe como organizador, quien a su vez será el encargado de alertar a los administradores cuando exista un problema. La estructura basada en P2P permitiría, en cierto modo, reducir el riesgo de que la caída del nodo principal impacte por completo la monitorización de la red, ya que al poder ser sustituido rápidamente (y sin intervención humana), la pérdida no sería notable y sería necesario un breve periodo de tiempo para que la red vuelva automáticamente a su estado normal.

Respecto a la viabilidad de desarrollar un software de monitorización que utilice exclusivamente P2P, si, es posible. La tecnología actualmente está disponible, hay muchos lenguajes que tienen librerías prefabricadas para el encaminado de paquetes en P2P, lo que facilitaría mucho la tarea para un desarrollo rápido.

El costo de desarrollar un producto de software no necesariamente podría estar al alcance de cualquier empresa, por lo que el modelo de desarrollo debería ser enfocado en el código abierto (en parte por obligación si se usaran librerías abiertas para el desarrollo), lo que aseguraría una comunidad que mantenga el interés en el proyecto, y facilite el acceso del producto a empresas de todo tamaño. Por otro lado, el hecho de que el producto sea distinto a lo ya ofrecido en el mercado, sumado a la resistencia al cambio, podría suponer un problema para venderlo, por lo que ofrecerlo como de código abierto permitiría a cualquiera probarlo sin compromiso alguno. Productos que fueron analizados en este trabajo, como Nagios y Zenoss, poseen tanto una gran comunidad como también clientes importantes (mencionados en sus respectivas fichas en el Capítulo IX), por lo que

no sería difícil lograr un resultado similar promoviéndolo como un producto innovador, más escalable (gracias a las bondades del P2P) y menos complejo (en relación a su administración).

En cuanto al consumo de recursos, debido a que la naturaleza del P2P requiere que cada cliente actúe como servidor, el producto se vería limitado en este aspecto. Todos los clientes pasarían a consumir más recursos si son parte de los nodos de primera línea, y ya de promedio consumirían más por el intercambio de tráfico para reorganizar la red, como para compartir a sus pares más inmediatos. El equilibrio estaría dado por el tipo de encaminado de paquetes que se usaría, como también por la estructura misma de la red. Lo que sí se presume que disminuiría notablemente, es el costo de trabajo para los administradores, ya que el software haría uso de distintos algoritmos para minimizar los falsos positivos, como también se actualizaría automáticamente por medio de la red P2P, eliminando así el trabajo mismo de ir manualmente nodo por nodo actualizando como sucede con la mayoría de los productos de monitorización con agente e híbridos.

Por último, y a nivel personal, este trabajo me ha permitido comprender como funcionan los productos de monitorización y conocer las diferencias entre los mismos, para poder tener una opinión formada a la hora de decidir sobre su uso en una red de sistemas. En cuanto a la propuesta en sí, me gustaría ponerla en práctica en un futuro cercano, ya que creo que significaría un cambio importante para el mercado, como también una apertura a nuevos competidores que actualmente desarrollan productos enfocados en P2P.

## Bibliografía


Alexei Vladishev, & Eugeny Grigorjev. (2008). Zabbix Manual v1.6. Recuperado a partir de http://www.zabbix.com/downloads/ZABBIX%20Manual%20v1.6.pdf

BMC. (2006). Agentless or Agent-based Monitoring? Recuperado a partir de http://i.i.com.com/cnwk.1d/html/itp/BMC_Agent_based_Monitoring.pdf

Camarillo, G. (2009). RFC 5694 - Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability. Recuperado octubre 8, 2011, a partir de http://tools.ietf.org/html/rfc5694

Clarke, R. (2004, noviembre 1). Peer-to-Peer (P2P) - An Overview. Recuperado octubre 8, 2011, a partir de http://www.rogerclarke.com/EC/P2POview.html

Computer Economics. (2011). IT Spending and Staffing Benchmarks 2011/2012. Recuperado octubre 8, 2011, a partir de http://www.computereconomics.com/page.cfm?name=IT%20Spending%20and%20Staffing%20Study

Computer Economics. (2012). IT Spending and Staffing Benchmarks 2012/2013. Recuperado a partir de http://www.computereconomics.com/temp/ISS2012Ch01Execsum002571.pdf

Denning, P. (1968). Trashing: Its causes and prevention. Recuperado a partir de http://www.cs.uwaterloo.ca/~brecht/courses/702/Possible-Readings/vm-and-gc/thrashing-denning-afips-1968.pdf

eMachines. (2009). Agent-based or Agentless? Recuperado a partir de http://downloads.emachines.com/service_docs/Gateway_Agent_Agentless.pdf

Engle, M., & Javed, K. (2006). Vulnerabilities of P2P Systems and a Critical Look at their Solutions. Recuperado a partir de http://www.medianet.kent.edu/techreports/TR2006-11-01-p2pvuln-EK.pdf

Essinger, J. (2007). Chapter 4 - The Emperor's new clothes. *Jacquard's web : how a hand-loom led to the birth of the information age*. Oxford: Oxford University Press.

Feldman, M., Papadimitriou, C., Chuang, J., & Stoica, I. (2004). FreeRiding and Whitewashing in Peer-to-Peer Systems. Recuperado a partir de

http://netecon.seas.harvard.edu/PINS04/Papers/feldman_04.pdf

Ghosemajumder, S. (2002). Advanced Peer-Based Technology Business Models. Recuperado a partir de http://shumans.com/p2p-business-models.pdf

Gillen, A. (2009). Linux in the Mainstream: Growing Deployment of Business-Critical Workloads. IDC. Recuperado a partir de ftp://ps.boulder.ibm.com/linux/pdfs/IDC-Business_Critical_Workloads_on_Linux.pdf

Gillen, A., Stergiades, E., & Waldman, B. (2008). The role of Linux servers and commercial workloads. IDC. Recuperado a partir de http://www.mandrivafr.org/dossiers/association/Pratique/Livre_blanc/Documentations/IDC_Workloads.pdf

IBM. (2010). ITM for Multi-function Devices User Guide.

James Sayles. (2009). Reducing the myopia regarding software agent technologies. Autonomic Software. Recuperado a partir de http://www.autonomic-software.com/pdf/Agent%20vs%20%20Agent-less%20White%20Paper.pdf

Johnson, S. (2005). Performance Analysis Tools. *Linux server performance tuning*. Prentice Hall Professional Technical Reference,. Recuperado a partir de http://i.techrepublic.com.com/downloads/home/ibmpress_linux_performance_4.pdf

Karl Kooper. (2005). *The Linux Enterprise Cluster : build a highly available cluster with commodity hardware and free software*. No Starch Press.

Koegel Buford, J. (2009). *P2P networking and applications*. Amsterdam ;;Boston :: Elsevier/Morgan Kaufmann,.

Mauro Douglas, & Kevin Schmidt. (2005). *Essential SNMP* (2nd edition.).

Nagios. (2009). Monitoring Network Printers. Recuperado noviembre 17, 2011, a partir de http://nagios.sourceforge.net/docs/3_0/monitoring-printers.html

Napper, B. (1998). Manchester Baby Computer. Recuperado octubre 8, 2011, a partir de http://www.computer50.org/mark1/new.baby.html

Rajiv Ranjan, Aaron Harwood, & Rajkumar Buyya. (2007). Peer-to-Peer Based Resource Discovery in Global Grids: A Tutorial. Recuperado a partir de

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.4049&rep=rep1&type=pdf

Rüdiger, S. (2001). A Definition of Peer-to-Peer Networking for the Classification of Peer-to- Peer Architectures and Applications. Recuperado a partir de http://csdl2.computer.org/comp/proceedings/p2p/2001/1503/00/15030101.pdf

Sandra K. Johnson, Gerrit Huizenga, & Badari Pulavarty. (2005). *Performance Tuning for Linux Servers*.

Schubert, M. (2008). *Nagios 3 enterprise network monitoring including plug-ins and hardware devices*. Burlington, MA :: Syngress Pub.,.

Silberschatz, A. (1998). *Operating system concepts* (5th ed.). Reading Mass.: Addison Wesley Longman.

Stallings, W. (2005). *Sistemas operativos : aspectos internos y principios de diseño*. Madrid: Pearson Prentice Hall.

Top500. (2011, noviembre). Top500.org - List of supercomputers (2011). Recuperado a partir de http://www.top500.org/static/lists/2011/11/TOP500_201111.xls

Top500. (2012, junio). Top500.org - List of supercomputers (2012). Recuperado a partir de http://www.top500.org/static/lists/2012/06/TOP500_201206.xls

Vasfi Gucer, Charles Beganskas, Fabrizio Salustri, Jerry Saulman, Mamadou Toure, John Willis, Brett Yamazi, et al. (2005). Getting Started with IBM Tivoli Monitoring 6.1 on Distribuited Environments. IBM Redbooks. Recuperado a partir de http://www.redbooks.ibm.com/redbooks/pdfs/sg247143.pdf

Zukis, B., Loveland, G., Horowitz, P., Verweij, G., & Bauch, B. (2008). Why isn't IT spending creating more value? PricewaterhouseCoopers. Recuperado a partir de http://www.pwc.com/en_US/us/increasing-it-effectiveness/assets/it_spending_creating_value.pdf

# Formulario descriptivo del Trabajo Final de Graduación

Este formulario estará completo sólo si se acompaña de la presentación de un resumen en castellano y un abstract en inglés del TFG

El mismo deberá incorporarse a las versiones impresas del TFG, previa aprobación del resumen en castellano por parte de la CAE evaluadora.

Recomendaciones para la generación del "resumen" o "abstract" (inglés)

"Constituye una anticipación condensada del problema que se desarrollará en forma más extensa en el trabajo escrito. Su objetivo es orientar al lector a identificar el contenido básico del texto en forma rápida y a determinar su relevancia. Su extensión varía entre 150/350 palabras. Incluye en forma clara y breve: los objetivos y alcances del estudio, los procedimientos básicos, los contenidos y los resultados. Escrito en un solo párrafo, en tercera persona, contiene únicamente ideas centrales; no tiene citas, abreviaturas, ni referencias bibliográficas. En general el autor debe asegurar que el resumen refleje correctamente el propósito y el contenido, sin incluir información que no esté presente en el cuerpo del escrito.
Debe ser conciso y específico". Deberá contener seis palabras clave.

## Identificación del Autor

| | |
|---|---|
| Apellido y nombre del autor: | **Almada Libra, Federico Ezequiel** |
| E-mail: | **federico.almada@gmail.com** |
| Título de grado que obtiene: | **Ingeniería en Sistemas de Información** |

## Identificación del Trabajo Final de Graduación

| | |
|---|---|
| Título del TFG en español | **Monitorización distribuida e inteligente de servidores con sistema operativo Linux** |
| Título del TFG en inglés | **Distributed and intelligent monitoring for Linux based servers** |
| Tipo de TFG (PAP, PIA, IDC) | **PIA** |
| Integrantes de la CAE | **Mgter. Fernando Frías**<br>**Ing. Jorge Cassi** |
| Fecha de último coloquio con la CAE | **09/05/2013** |
| Versión digital del TFG: contenido y tipo de archivo en el que fue guardado | **Contiene TFG en formato DOC, ODT y PDF, CV en formato PDF** |

## Autorización de publicación en formato electrónico

Autorizo por la presente, a la Biblioteca de la Universidad Empresarial Siglo 21 a publicar la versión electrónica de mi tesis. (marcar con una cruz lo que corresponda)

**Autorización de Publicación electrónica: Inmediata**

☒ **Si, inmediatamente**

☐ **Si, después de ….. mes(es)**

☐ **No autorizo**

**Firma del alumno**