Escalabilidad en sistemas de e-mail

Pablo E. Cingolani, Hernán J. González, Eduardo M. Panciera Molanes.

Resumen: La implementación de sistemas de e-mail que escalen fácilmente hasta varios miles o millones de usuarios, no es trivial. En este trabajo se ve un ejemplo de un sistema de mail altamente escalable, probado hoy en día para cientos de miles de usuarios basado en arquitecturas abiertas. El sistema fue desarrollado en Sinectis S.A. y se utiliza exitosamente como infraestructura de e-mail.

Introducción

Los problemas que surgen al escalar sistemas no suelen ser triviales ya que la mayoría no estan preparados para ser escalados, tanto por balanceo de carga como por distribución o por paralelizacion del procesamiento. En el caso del e-mail se presentan varios desafíos. Los servers no pueden desconectarse para ser reorganizados, ya que funciona 7x24 y los clientes exigen que nunca se detengan. Escalar el sistema implica en el mejor de los casos una migración cuidadosamente planeada para que no se efectúen cortes de servicio. Además si la cantidad de clientes crece rápidamente estas migraciones deben ser simples, ya que se efectuaran en forma periódica. La solución aquí mostrada fue implementada enteramente sobre arquitecturas abiertas (e.g. Linux, Sendmail, etc.) .En la primer parte del trabajo explicaremos brevemente el funcionamiento de los sistemas de mail[1] tradicionales. En la segunda se plantearán las problemáticas de escalar dichos sistemas. En la tercer parte se mostrarán las soluciones adoptadas a cada problema planteado. En la cuarta parte del trabajo se mostrarán futuras mejoras y problemas para perfeccionar.

Parte I: Funcionamiento del e-mail tradicional

Cuando una persona cuya dirección de e-mail es juan@foo.com desea enviar un mail a otra persona cuya dirección de e-mail es pepe@bar.com el proceso es el siguiente:

1. Juan escribe el mail en un programa y lo envía a su **mail relay** que supondremos que es el server **relay.foo.com**)
2. **relay.foo.com** busca cuáles son los responsables de mail del dominio **bar.com**, mediante una búsqueda de **mail exchangers** vía **DNS**[2] (supondremos que el mail exchanger de bar.com es **mx.bar.com**)
3. Una vez que **relay.foo.com** encuentra quien es el/los mail exchanger/s de **bar.com** establece una conexión TCP (port 25) y envía el mail mediante protocolo **SMTP**[3] (de no lograr establecer la conexión, reintentara luego de unas horas; después de varios reintentos fallido devuelve el mail con un mensaje de error)
4. **mx.bar.com** recibe el mail y verifica que el usuario al cual esta destinado exista (si no existe da un mensaje de error). Luego de eso mx.bar.com deposita el mail en la "casilla" del usuario[4] (normalmente lo agrega –append- al final de un archivo, que es la casilla del usuario).

En este momento, se considera que el mail fue enviado exitosamente. El próximo paso es cuando el usuario que recibe elmail (Pepe) desea leer sus mails.

5. Cuando el usuario entra a la red, se conecta mediante un cliente de mail, vía protocolo POP[5] (TCP al port 110) al server de pop de bar.com (supondremos que se llama **pop.bar.com**). El server pop examina la casilla del usuario y efectúa las operaciones correspondientes (bajar los mails, borrarlos, etc.).
6. En cada una de estas etapas los respectivos servers dejan registros de las operaciones efectuadas (archivos de **log**).

[1] En este trabajo se utilizaran los términos **e-mail, email** y **mail** indistintamente.

[2] Busca los registros **MX** del dominio bar.com (i.e. algo análogo a hacer "nslookup –q=mx bar.com)

[3] Simple Mail Transfer Protocol (RFC 821)

[4] Esto se denomina **local delivery**

[5] **Post Office Protocol** (RFC 918), también puede utilizarse IMAP (**Internet Message Access Protocol,** RFC 1730) o algún otro protocolo, conceptualmente similar.

Si bien aquí hemos dividido las operaciones en diferentes servers, lo más usual es que sea un único server que efectúa las operaciones. Lo cual es admisible si solo se manejan algunos miles de cuentas.

Parte II: Problemas de escala

En la mayoría de los pasos planteados en la **Parte I** se plantean problemas de escala, lo cual es lógico ya que el sistema no fue diseñado con un objetivo de escalabilidad en mente.

El primer problema que aparece es simplemente la busqueda de los datos de un usuario (e.g. password, directorio home, etc.). En la mayoría de los sistemas operativos estas busquedas se hacen en forma lineal (e.g. se buscan los datos del usuario en un archivo de texto plano -/etc/passwd)[6]. Un archivo del tipo "/etc/passwd" de Unix mide aproximadamente 50 bytes por usuario, para 1.000.000 de usuarios esto es 50Mbytes, lo cual implica leer, en promedio, 25Mbytes por cada consulta de datos de usuario. Es evidente que esto no es optimo.

El formato utilizado para la casilla (mailbox) hace que la operación de **local delivery** sea poco eficiente, ya que se hace un "append" a un archivo plano, para hacer esto se debe buscar el final del archivo y agregar los datos, lo cual resulta poco eficiente en archivos muy largos. También surgen problemas de file locking (e.g. si dos mails llegan al mismo tiempo para el mismo usuario). Estos problemas de file locking no suelen estar bien resueltos en los sistemas operativos (sobre todo en sistemas de archivos en red[7]), además se corre el riesgo de dejar el archivo en estado "locked" por algún error en el proceso de local delivery, con lo cual el usuario no puede recibir mas mails.

Al leer los mails, el server POP tiene que leer todo el archivo de texto plano del mail, para identificar los mensajes. Luego si desea borrar uno, debe reorganizar todo el archivo. En síntesis, la mayoría de las operaciones del server POP sobre la casilla, son poco eficientes. Dado que la cantidad de mails promedio en las casillas se incrementan con los años, y que el tamaño promedio de e-mail también se incrementa, estas operaciones son cada vez mas costosas computacionalmente con el transcurso de los años.

Otro problema tanto del local delivery como del server POP es que las casillas de mail suelen estar en un unico directorio (e.g. /var/spool/mail) dentro del server. En el caso de tener cientos de miles de casillas, solo el hecho de buscar un archivo en el directorio es una operación costosa ya que se deben recorres muchos i-nodos para encontrar los datos básicos del archivo (como el nombre).

Finalmente, los archivos de **log** que se van dejando en los distintos servers suelen ser problemáticos: muchos archivos, difíciles de coordinar, suelen ser muy grandes.

Parte III: Escalabilidad

Hay dos formas básicas de atacar el problema:

1. Servers mas potentes (mejor hardware, procesadores paralelos, etc.)
2. Distribución / balanceo de carga.

Si bien la primer opción suele ser bastante mas cara que la segunda, tiene la ventaja de poder tener todo centralizado en un solo server. Desgraciadamente en casos de necesitar muchas cuentas de e-mail, el costo del hardware no es lineal y lo hace impracticable. Evidentemente con el avance de la tecnología, la primer opción será cada vez mas fácil de alcanzar, pero también con el tiempo aumentan los requerimientos en cuanto a cantidad de usuarios, cantidad de e-mails por usuario, tamaño de los e-mails, etc. Si bien a largo plazo seguramente la primer opción funcione, en los próximos años (periodo corto/mediano plazo) la solución mas costo efectiva es la segunda.

La segunda opción incluye dos alternativas:

Balanceo: Es distribuir la carga de los procesos en forma igual entre los procesadores.

Distribución: Es distribuir la carga de los procesos en forma diferenciada entre los procesadores.

[6] Si bien esto es típico de UNIX, en otros sistemas es similar

[7] NFS o similares

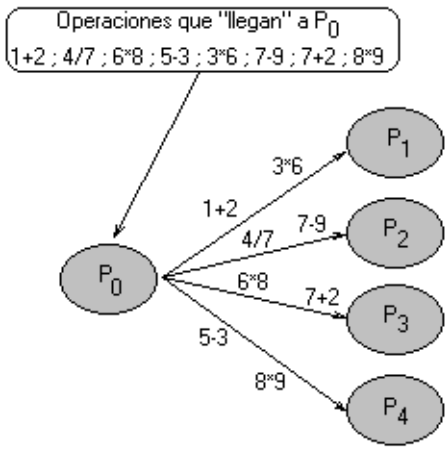

Fig. 1 [Balanceo de carga]

Para ver un ejemplo entre distribución y balanceo, supongamos que tengo 4 procesadores interconectados (e.g. 4 computadoras conectadas en red). Los llamaremos $\{P_1,P_2,P_3,P_4\}$ y tengo que efectuar calculos del tipo "z=x+y", "z=x-y", "z=x*y" y "z=x/y" que van "llegando" a un procesador P_0 (i.e. otra computadora) aleatoriamente a medida que pasa el tiempo. Balanceo seria si P_0 envía a cada uno de los demás procesadores a efectuar un calculo a medida que van "llegando", i.e. el primero a P_1, el segundo a P_2, el tercero a P_3, etc. (ver Fig.1).

Distribución es si P_0 envía, por ejemplo, solo las sumas a P_1, solo las restas a P_2, solo las multiplicaciones a P_3 y solo las divisiones a P_4.(ver Fig. 2)

El problema es atacado mediante métodos de distribución y de balanceo de cargas así como también combinaciones de ambos. Uno de los problemas que surgen es que el balanceador / distribuidor de carga pasa a ser un punto critico, tanto desde el punto de vista de falla del sistema como de la escalabilidad. Para evitar esto lo que se hace es una combinación de balanceo / distribución (i.e. se balancea la distribución de carga).

Existen varias formas de implementar un balanceo o una distribución de carga. La forma mas simple es mediante DNS, esto quiere decir que se ponen varios registros en el DNS que indican el nombre asociado a varias direcciones IP. Cada vez que se hace una consulta por el nombre, aparecen como respuestas todas las direcciones IP, en ordenes diferentes. De esta forma se implementa un balanceo de carga (estilo "round robin"). Otros métodos incluyen aplicaciones específicamente programadas para balancear o distribuir carga, corriendo en un server, hasta desde switches niveles[8] 3, 4 y superiores. En este trabajo se utilizaron técnicas de balanceo por DNS y distribución por software de aplicaciones específicamente programados con tal fin.

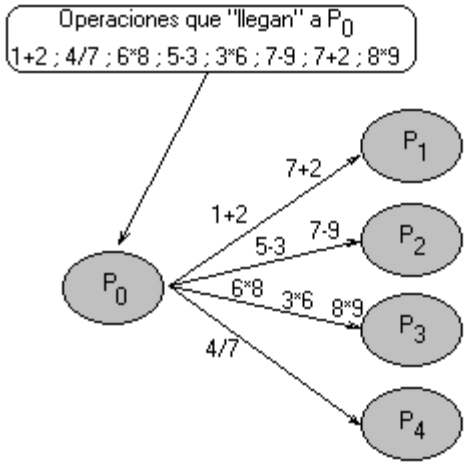

Fig. 2 [Distribución de carga]

III.A División de mail entrante / mail saliente

En el caso del e-mail, la primer forma de atacar el problema es utilizar servers separados para las operaciones que son fácilmente divisibles. Por ejemplo, los servers de relay (mail saliente) se pueden dividir de los servers que cumplen la función de mail exchanger (mails entrantes). Esto no es complicado, ya que las operaciones son casi totalmente independientes. Se pueden poner varios servers de relay balanceados mediante DNS, lo mismo para los mail excahngers; con lo cual se tienen dos grupos o "clusters" de server que cumplen funcionalidades diferentes. El caso del balanceo de mail exchangers, se analiza mas en detalle en el punto III.E.

III.B Logs

Otro problema fácil de solucionar es el de los archivos de log, simplemente se implementa

[8] Nos referimos a los niveles del modelo OSI

un server de **syslog** al cual se envían todos los logs de todos los servicios a través de la red[9].

III.C Demasiados archivos en un mismo directorio

Para solucionar el problema de tener muchos archivos en un solo directorio, se implemento una especia de "hashing" de los directorios de las casillas de mail. Simplemente la casilla del usuario **juan** en lugar de estar en el directorio **/var/spool/mail/juan**, esta en el directorio **/var/spool/mail/j/u/a/juan**. De esta forma simple, se evita el problema. En caso de tener demasiados usuarios se puede utilizar otra "función de hashing" (comentaremos sobre esto en la sección IV). Puede ser necesario cambiar parte del codigo del server de POP y el programa de delivery local para que incorporen esta funcionalidad (al tratarse de arquitecturas abiertas, esto es fácil de implementar)

III.D Casilla de mail en un archivo lineal

Para solucionar los problemas que genera el hecho de tener una casilla en formato de archivo lineal, se puede implementar una casilla pop en forma de directorios. En lugar de que cada mail sea agregado a un archivo que es la casilla del usuario, cada mail es un archivo independiente dentro del directorio (que es la casilla del usuario). Con este método, se evitan también los problemas de file locking.

Una de las desventajas de este método es que se desaprovecha el espacio en disco, ya que cada archivo distinto tiene, en promedio, medio bloque desaprovechado, lo cual suele ser entre 1K y 4K dependiendo del particionamiento del disco (e.g. si hay 100.000 usuarios, que tienen en promedio 11 mensajes cada uno[10] y el disco tiene bloques de 4K, tenemos 2.2 GBytes desperdiciados). Este problema no es muy grave.

Otro problema es que ver un listado de mails es mas costoso computacionalmente con una casilla tipo directorio que con una casilla tipo archivo lineal si la cantidad de mails es muy grande y los mails son cortos. Para una cantidad de mails menor a 2000 por casilla (con un tamaño promedio de 11K), la diferencia del tiempo de lectura no es grande. Dado que habitualmente la cantidad de mails que tiene un usuario es mucho menor a 2000, esto no presenta un problema serio.

III.E Búsqueda lineal de los datos de usuarios

El paso siguiente fue evitar las búsquedas lineales de los datos de usuarios. Para esto se implemente un "password" server, que almacena los datos de los usuarios en memoria y mediante métodos de hashing, cuando un proceso necesita datos de un usuario, se los pide al password server (mediante protocolo simple implementado sobre UDP), el password server responde, sin necesidad de buscarlo en el disco, ya que los tiene en memoria y sin hacer una búsqueda lineal, ya que están indexados mediante un hash. Esto se podría haber implementado mediante un protocolo estándar, como LDAP, el inconveniente que veíamos es que LDAP funciona sobre TCP (por lo que requiere mas de 2 paquetes para una búsqueda simple) y que en general los servers de LDAP consultan una base de datos (una consulta a disco no es deseable). Algo similar sucedía con el protocolo RADIUS, así que la mejor opción fue implementar un protocolo muy simple que fuera rápido y eficiente para esta operación.

Afortunadamente existe en el sistema operativo una forma fácil de implementar una autentificación alternativa, mediante un mecanismo conocido como **"System Databases and Name Service Switch configuration"**[11]). Este mecanismo proporciona las interfaces para implementar el nexo entre nuestro password server y las aplicaciones que requieren datos de los usuarios, como los servers de SMTP, POP, etc. De no existir este mecanismo, dado que se utiliza software abierto, no es difícil programar el código fuente y recompilar el software para que pregunte los datos al password server en lugar de al archivo de usuarios.

La implementacion de software del password server es en su mayoría simple ya que, dado que las consultas y las búsquedas son rápidas no hay problemas de concurrencia, es decir que se puede implementar un proceso único que atienda y responda las consultas en orden. Los problemas de concurrencia pueden surgir en las actualizaciones de los datos de los usuarios, si se hace una actualización completa de toda los usuarios. Esto ultimo no se hace muy seguido, además se implementa mediante un sistema multi-threads que le permite al server seguir atendiendo consultas. En caso de falla de un server, hay un

[9] Esta es una facilidad que ya esta incorporada en el servicio **syslog**.

[10] Estos números son normales para un server de mail hoy en día.

[11] Esta configuración se suele encontrar en el **/etc/nsswitch.conf**

server secundario, el cliente envía el pedido de información al server secundario en caso de que el primario no responda en un tiempo predeterminado (actualmente 2 segundos). De esta forma se implementa un mecanismo robusto ante fallas.

III.F Balanceo de carga en mails entrantes

Para efectuar un balanceo de carga en los mails entrantes a los mail exchangers (servers que procesan los mails entrantes), el primer es poner varios en paralelo balanceados mediante DNS. Luego se separa la funcionalidad de mail exchanger de la funcionalidad de almacenamiento (storage) de las casillas, es decir que el server que recibe los mails no es el mismo server que los almacena. Para poder escalar fácilmente el sistema, se implemento un sistema de casillas distribuidas (i.e. las casillas no están todas en el mismo server), si bien toda la casilla de un usuario esta en un solo server, las casillas de todos los usuarios están distribuidas en varios servers.

Cuando un mail exchanger recibe un mail, valida el usuario comunicandose con el password server, y luego hace un delivery local a la casilla del usuario. Dado que el delivery local se hace a otro server y que dependiendo del usuario el server es distinto, se implementó un programa que distribuye los mails en los storage servers según corresponda. Este programa reemplaza el programa del delivery local que utiliza el Sendmail[12]. Estos programas client/server de delivery local (que denominamos pdelivery/pdeliveryd[13]), es un programa estilo client/server que funciona sobre TCP. Si bien se podría haber utilizado NFS, la performance de NFS no fue satisfactoria, por eso elegimos implementar un protocolo simple específicamente pensado para el delivery local.

III.G Distribución de las casillas de mail

Para distribuir las casillas de mail en los diferentes storage servers, se eligió un método muy simple: las casillas de los usuario que comiencen con {‘a’,‘b’,‘c’} en el primer server, {‘d’,‘e’,‘f’} en el segundo y así sucesivamente. Si bien el método es simple, y fácil de implementar, tiene un par de desventajas, que discutiremos en el punto IV.

Los server que tiene la funcionalidad de storage, también tiene un server de POP/IMAP, para que los usuarios tengan acceso a sus mails. La ventaja de que esta funcionalidad es que el server de POP/IMAP tiene acceso directo a la casilla del usuario.

III.H Distribución de carga de POP

Como se vio en el punto III.G las casillas estas distribuidas en varios storage servers y cada uno de ellos tiene un server POP/IMAP. Una opción para distribuir la carga sería simplemente decirle a cada usuario que configure el server POP que le corresponda, esto es poco practico con muchos usuarios y casi imposible si son cientos de miles, además dado que es habitual el movimiento de casillas, para ampliar el espacio de los servers, habría que hacer cambiar las configuraciones de miles de usuarios con cada ampliación.

Una opción mas practica (y realista) es implementar un distribuidor de carga POP. El usuario establece la conexión con alguno de los servers POP que aparecen en el DNS (en el ejemplo del punto I seria **pop.bar.com**). Este server es una aplicación que atiende por protocolo POP. Primero espera que el cliente envie el nombre del usuario; una vez obtenido encuentra el nombre del storage server correspondiente a dicho usuario. Luego conecta con una sesión POP con ese server, de ahí en adelante la aplicación lo único que hace es enviar la información hacia uno u otro lado (estilo proxy). De esta forma se logra distrubuir la carga de POP en forma que el usuario no lo ve (transparente).

En la practica los distribuidores de carga POP están funcionando en los mismos servers que los mail exchangers, pero no hay ningún motivo por el cual no se puedan separar.

III.I El sistema completo

En la figura 3 se muestra un esquema completo de todos los componentes del sistema de e-mail. y las interacciones entre los distintos servers, así como los protocolos.

Parte IV: Futuras mejoras

Si bien el sistema funciona exitosamente y escala con facilidad, existen algunas mejoras que se pueden implementar:

1. En lugar de utilizar las tres primeras letras para indexar los directorios de los usuarios, se

[12] Se utilizo **Sendmail** como programa server de SMTP para los mail exchangers.

[13] La letra “d” al final de **“pdeliveryd”**, indica que es un daemon (server). La “p” de **“pdelivery”** denota el nivel de implementación.

pueden utilizar otros métodos. Las alternativas que se analizan incluyen desde utilizar las dos primeras letras y la ultima o la primera y las dos ultimas. En la Fig. 4 se muestra una comparación en la eficacia de los métodos de hashing propuestos.

2. El método de distribución de las casillas en los distintos servers requiere una migración de casillas cuando se efectúa una ampliación (i.e. cuando se agrega un server de storage). Se puede utilizar una relación cualquiera entre usuario-server (en lugar de la relación rígida por la primer letra del nombre) y almacenar este dato en el password server. De esta forma se pueden agregar storage servers sin necesidad de migraciones de casillas, en una forma más flexible. Esto haría más rápida y sencilla aun la escalabilidad.
3. Un ultimo punto es que los datos de los usuarios se envían sin encriptar, ya que todos los servers están en una red privada protegida. En caso de querer implementar esta solución en forma distribuida, se debe hacer que el protocolo del password server encripte los datos.

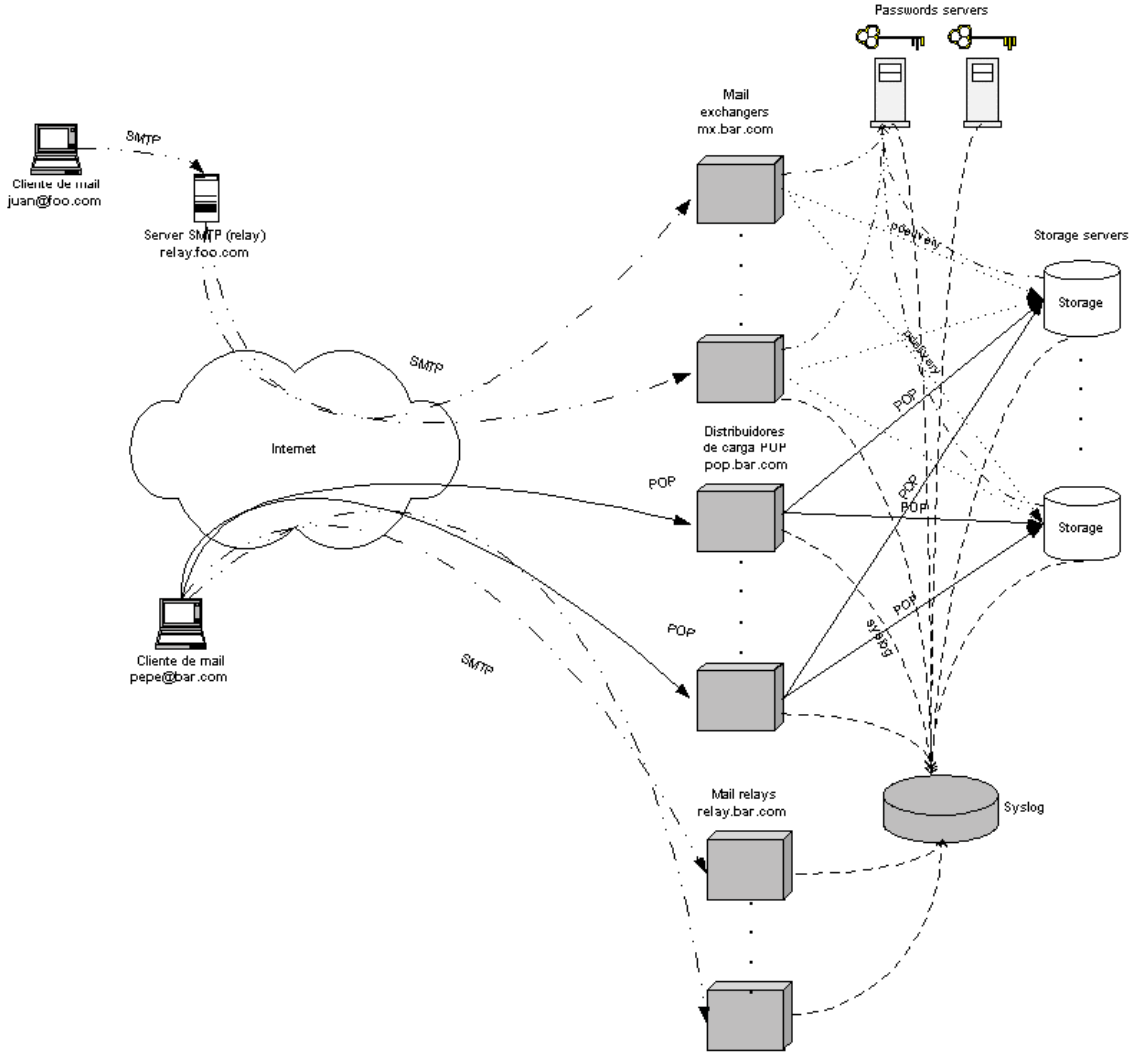

Fig. 3 [Sistema de mail completo]

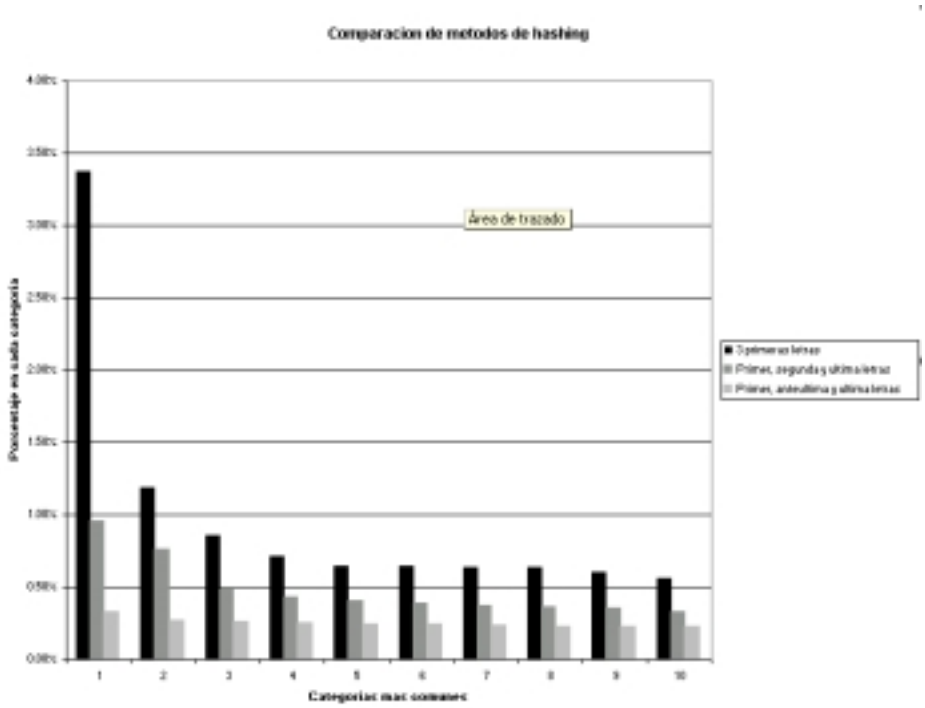

Fig. 4 [Comparacion de los metodos de hashing]

Referencias:

- N. Christenson, T. Bosserman, D. Beckemeyer. **"A highly Scalable Electronic Mail Sercive Using Open Systems"**, Dec. 1997.
- N. Christenson . **"Large News"**, talk transcription, Cisco campus, San Jose, Feb. 1998.
- Yasushi Saito, Eric Hoffman, Brian Bershad, Henry Levy, David Becker. **"The Porcupine Scalable Mail Server"**, Dep. of Computer Science and Engineering, University of Washingon & Laboratoire d'Informatique de Paris 6 CNRS.
- N. Christenson, D. Beckemeyer, T. Baker. **"A Scalable New Architecture on a Single Spool"**, Jun. 1997.
- B. Knowles, N. Christenson. **"Desing and implementation of Highly Scalable E-mail Systems"**, LISA XIV talk, Dec. 2000.