Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 1, No. 2, pp. 81-92, Diciembre 2014
Selección de los Mejores Artículos de CoNCISa 2014

Balance de Carga y Tolerancia a Fallas en OPTIMUS (OPTImized Mail distribUtion System)

Fabiola Rosato[1], Javier Argüello[1], Yudith Cardinale[1]
rosato.fcrm@gmail.com, javier@ldc.usb.ve, yudith@ldc.usb.ve

[1] Departamento de Computación, Universidad Simón Bolívar, Caracas, Venezuela

Resumen: En este trabajo presentamos la implementación de una Plataforma Distribuida de envío masivo de correos electrónicos, llamada OPTIMUS (OPTImized Mail distribUtion System), que permite distribuir eficientemente las cargas de trabajo entre múltiples servidores de correos, optimizar el uso de los recursos disponibles, garantizar tolerancia a fallas, obteniendo como resultado una robusta plataforma de envío. La arquitectura de OPTIMUS extiende la implementación de servidores Postfix para incorporar el manejo de base de datos, balance de carga y tolerancia a fallas, y está conformada por múltiples servidores Postfix extendidos, conectados en una plataforma Peer-to-Peer. Los resultados experimentales muestran dos aspectos importantes: (i) OPTIMUS supera notablemente el desempeño de Postfix en su versión original centralizada, para cantidades significativamente grandes de correos electrónicos, y (ii) los mecanismos de tolerancia a fallas de OPTIMUS, no generan una degradación en el desempeño global del sistema, durante su actividad normal (sin fallas) y realizan una recuperación totalmente efectiva si ocurren fallas.

Palabras Clave: Correo Electrónico; Postfix; Sistemas P2P; Distribución Masiva; Balance de Carga; Tolerancia a Fallas.

Abstract: In this work, we present the implementation of a Distributed Platform for Massively Emails distribution, called OPTIMUS (OPTImized Mail distribUtion System). It allows efficient workload distribution among multiple email servers, optimizes the use of available resources, and provides fault tolerance in a robust distribution platform. OPTIMUS architecture extends the Postfix server implementation to incorporate data base management, workload balance, and fault tolerance. It is composed by several extended Postfix servers, connected in a Peer-to-Peer platform. Experimental results show that with a few amount of emails, Postfix (with its original centralized implementation) and OPTIMUS have similar performance. However, with huge amount of emails, OPTIMUS highly overcomes the performance of the original centralized Postfix version.

Keywords: E-Mail; Postfix; P2P Systems; Masive Distribution; Load Balancing; Fault Tolerance.

I. Introducción

La necesidad de tener correo electrónico surgió en los años sesenta, a partir del momento en que las personas comenzaron a tener acceso compartido a una misma computadora. En aquel entonces, el correo electrónico se limitaba al intercambio de mensajes entre usuarios de una misma máquina. Los primeros correos electrónicos se enviaron alrededor de 1970, a través del predecesor del Internet que conocemos hoy en día, ARPANET. En aquel entonces, la entrega de los mensajes era un proceso relativamente sencillo y consistía en mover archivos entre un servidor y otro que atendían a muchos usuarios [1].

Con el paso de los años y con la rápida evolución del Internet, el propósito del "e-mail" cobró un significado distinto: comunicar a las personas sin importar la distancia entre ellas. Eventualmente el correo electrónico cobró gran importancia en la comunicación, de manera particular con el nacimiento e incorporación de los dispositivos móviles [2]. Por ejemplo, en el ámbito del mercadeo, ha conseguido un papel trascendental, lo cual ha generado un incremento sustancial en la utilización del correo electrónico con fines publicitarios [3].

Actualmente el volumen de correos diarios se calcula en más de 190 mil millones [4]; por esta razón este medio de comunicación es uno de los más influyentes en la sociedad. A raíz de esto, aparecen las compañías de envíos de correo masivo, que en su mayoría siguen un enfoque centralizado y requieren de servidores y otros equipos que incurren en gastos, tales como tiempo, consumo de CPU, consumo de luz, etc.

La tendencia de la evolución de computación en la última década deja de lado los sistemas centralizados y da paso a los sistemas distribuidos masivamente. Estos sistemas masivos

se caracterizan por interconectar una gran cantidad de nodos, incluso millones de nodos; creando la necesidad de plantear nuevas formas de comunicación que incluyen la comunicación uno-a-muchos y que supere al tradicional estilo uno-a-uno [5]. La distribución masiva de correos electrónicos no escapa a esta tendencia.

Hoy en día existe software de distribución masiva de correos disponible y de código abierto, tales como Sendmail [6] y Postfix [7], que permiten el envío masivo de correos mediante el uso de múltiples servidores. Sin embargo, existen diferentes factores que pueden influir en su rendimiento, tales como las características físicas de cada servidor, la velocidad de la red o el ancho de banda que poseen. La gran mayoría de los agentes de correo electrónico manejan sus colas en disco en archivos planos, lo cual puede provocar gran actividad de entrada/salida (I/O) que en un sistema masivo puede ser crítico, pues representa un gran consumo de recursos. Otro aspecto relevante es la asignación de correos a ser enviados entre los servidores disponibles, ya que si éstos no siguen una política adecuada de balance de carga, pueden desperdiciar el uso de los recursos. Así mismo el aspecto de tolerancia a fallas no es ampliamente considerado en estas aplicaciones clásicas de distribución masiva de correos electrónicos.

Se estima que una arquitectura distribuida puede minimizar estas limitaciones a través de un algoritmo eficiente de distribución de correos entre los servidores, que tome en cuenta información del contexto de ejecución (carga de los servidores, localización física de los servidores y de los clientes, etc.) para decidir en un determinado instante cuáles servidores conviene utilizar para una tarea específica de envío masivo a una cantidad determinada de clientes, esto se traduce en la implementación de un enfoque de balance de carga eficiente. Por otro lado, si se mejora el manejo de las colas de mensajes con el uso de una base de datos (en lugar de archivos planos), se considera que se podría reducir la actividad de entrada y salida, y por lo tanto minimizar el consumo de recursos. Además, la implementación de mecanismos de tolerancia a fallas brindarían confiabilidad y alta disponibilidad de los servicios de distribución masiva de correos electrónicos.

En este contexto, este trabajo plantea una arquitectura distribuida de envío masivo de correos electrónicos, basada en la extensión de servidores Postfix, para capacitar el envío de correos almacenados en una base de datos y balancear la carga entre los servidores, de manera de maximizar el envío de los correos y minimizar los costos, además de ofrecer mecanismos de tolerancia a fallas. Llamamos a esta nueva arquitectura OPTIMUS (OPTImized Mail distribUtion System). En un trabajo previo presentamos la arquitectura general de OPTIMUS, conformada por múltiples servidores Postfix extendidos, conectados en una plataforma Peer-to-Peer (P2P), considerando la incorporación de una base de datos y un algoritmo de balance de carga [8]. En este artículo detallamos la implementación de la primera versión de la plataforma e incorporamos los mecanismos de tolerancia a fallas con la finalidad de proveer una plataforma robusta que optimice el envío masivo de correos electrónicos. Los resultados experimentales muestran que para cantidades relativamente pequeñas de mensajes electrónicos, tanto Postfix (en su versión original centralizada) como OPTIMUS tienen aproximadamente el mismo desempeño. Sin embargo, para cantidades significativamente grandes, OPTIMUS supera notablemente el desempeño de la versión original de Postfix. Además, también mostramos experimentalmente que los mecanismos de tolerancia a fallas de OPTIMUS, no generan una degradación en el desempeño global del sistema durante su actividad normal (sin fallas) y que realizan una recuperación totalmente efectiva si ocurren fallas.

La presentación de nuestro trabajo está organizada como sigue. En la Sección II se describe el entorno de las plataformas de distribución de correos electrónicos, incluyendo la descripción detallada de la arquitectura de Postfix. La arquitectura de OPTIMUS se presenta en la Sección III. Los resultados experimentales se muestran en la Sección IV. La Sección V expone las principales diferencias entre OPTIMUS y Postfix original. Finalmente, la Sección VI presenta las conclusiones y el trabajo que planeamos continuar realizando.

II. Plataformas de Distribución de Correos Electrónicos

El correo electrónico surgió a principios de los años 70 y se utilizó por primera vez en ARPANET. Posteriormente, ARPANET fue sustituido por Internet, y con la rápida evolución de éste, el correo electrónico adoptó un rol predominante como medio de comunicación. Rápidamente, el correo electrónico se convirtió en un medio de comunicación dominante. Con la evolución de Internet, la redes se hicieron cada vez más complejas y empezaron a necesitar herramientas que facilitaran el intercambio y envío de archivos entre servidores y capaces de tratar la extensa gama nueva de servicios de correo [9].

Uno de los primeros programas utilizados para el intercambio de correo electrónico fue el paquete Sendmail [10], el cual tenía como objetivo lidiar con la diversidad de sistemas de correo. Inmediatamente, este servidor se convirtió en un programa que jugaba un rol predominante en Internet. Sendmail utiliza el protocolo SMTP y sigue siendo uno de los servidores más usados actualmente. Sin embargo, la estructura monolítica de su arquitectura se ha convertido en un factor que ha degradado su reputación y uso, ya que es propenso a numerosas fallas de seguridad y su configuración y mantenimiento pueden llegar a ser muy complicados.

Para sobreponer las limitaciones de Sendmail, surge Postfix como una alternativa segura y flexible [11][12]. Postfix fue desarrollado en los laboratorios de “IBM research”, por Wietse Venema, creador de software como SATAN y TCPwrappers, como reemplazo de Sendmail, que es considerado un sistema de envío de correo inseguro. Este agente de transporte de correo fue desarrollado a finales de los 90 y en un principio se llamó “IBM Secure Mailer”, hasta que se autorizó su distribución como una licencia de código abierto (*open source*) y se renombró como “Postfix”. La creación de Postfix fue impulsada por objetivos como confiabilidad, seguridad, desempeño,

flexibilidad, usabilidad y compatibilidad con Sendmail.

Las siguientes secciones describen la arquitectura, composición general del servicio de correo electrónico y los principales protocolos usados.

A. Componentes del Servicio de Correo

El correo electrónico está basado en una serie de estándares y protocolos que definen cómo son redactados y transferidos los mensajes. Dentro de este proceso intervienen distintos procesos que en conjunto llevan a cabo el envío del correo. Los principales componentes de un servicio de correo electrónico son:

- Agente de Usuario (MUA, por sus siglas en inglés de *Mail User Agent*), es el software que permite a los usuario leer y redactar los mensajes. No participan en el proceso de envío del correo.
- Agente de Entrega (MDA, por sus siglas en inglés de *Mail Delivery Agent*), tiene la tarea de almacenar el mensaje en un archivo o almacenarlo en alguna base de datos especializada para correos.
- Agente de Transporte (MTA, por sus siglas en inglés de *Mail Transport Agent*), es el encargado de determinar si el destino del mensaje es local o remoto. De ser local, el MTA entrega dicho mensaje al MDA, y en caso contrario, lo envía a otro Agente de Transporte ubicado en otro sistema. Si un mensaje no puede ser entregado, este agente es el encargado de devolver el mensaje al emisor original o notificar al administrador del sistema.

La interacción de estos componentes se muestra en la Figura 1. Para que los agentes de envío de correo electrónico puedan comunicarse entre sí, se estandarizó el intercambio de mensajes y de esta forma, diferentes programas pueden interoperar sin importar quién los desarrolle, siempre y cuando implementen uno de los protocolos: *Simple Mail Transfer Protocol (SMTP)* [13], *Extended Simple Mail Transfer Protocol (ESMTP)* [14] o *Local Mail Transfer Protocol (LMTP)* [15]. Tanto Sendmail, como Postfix pueden utilizar cualquiera de estos tres protocolos.

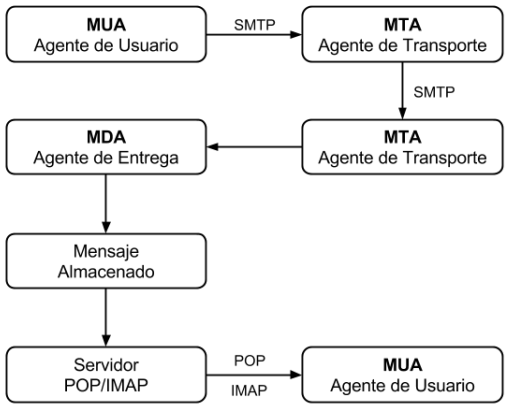

Figura 1: Flujo de un Mensaje [12]

B. Arquitectura de Postfix

1) ***Subdirectorios de las Colas de Postfix:*** Postfix tiene estructuras de tipo cola donde se almacenan los correos como archivos separados, utilizando identificadores únicos para cada nombre de archivo. Cada cola de mensajes es creada como un subdirectorio. Estos subdirectorios se presentan a continuación.

- **active**: Contiene los mensajes que están siendo procesados por el manejador de colas de Postfix.
- **bounce**: Contiene el registro de las causas por las cuales el mensaje fue rebotado.
- **corrupt**: Contiene los mensajes que no están en un formato adecuado.
- **defer**: Contiene los registros de las causas por las que ha sido demorado un envío, para que más adelante Postfix intente enviarlo de nuevo.
- **deferred**: Contiene los mensajes demorados que Postfix intentará enviar de nuevo.
- **incoming**: Contiene mensajes nuevos, recibidos de *hosts* SMTP remotos.
- **maildrop**: Contiene los mensajes entregados por el comando *sendmail* que están esperando ser procesados por Postfix.

2) ***Componentes:*** A diferencia de Sendmail, Postfix divide y asigna sus tareas a distintos programas individuales. La mayoría de los programas son demonios y son invocados por el *master daemon* quien es el primero en iniciar. Estos programas son los siguientes:

- **Bounce**: se encarga de devolver los correos rebotados a su emisor original. También es el encargado de diferir el correo dirigido a un *host* no disponible.
- **Cleanup**: procesa los encabezados del correo entrante y lo coloca en la cola *incoming*.
- **Qmgr**: procesa los mensajes en la cola *incoming*, determina dónde y cómo deben ser enviados e invoca al programa responsable de la entrega.
- **Error**: procesa el mensaje proveniente del *qmgr* y lo obliga a rebotar.
- **Local**: entrega el correo destinado a usuarios locales.
- **Pickup**: espera por mensajes en la cola *maildrop* y los envía al programa *cleanup* para que los procese.
- **Pipe**: reenvía el mensaje del *qmgr* a un programa externo.
- **Postdrop**: mueve los mensajes entrantes a la cola *maildrop*, cuando los usuarios normales no pueden escribir en ella.
- **Showq**: reporta los estatus de las colas de Postfix.
- **Smtp**: envía el mensaje a *hosts* externos utilizando el protocolo SMTP.
- **Smtpd**: servidor SMTP que recibe los correos provenientes de *hosts* remotos.
- **Trivial-rewrite**: recibe los mensajes del programa *cleanup* y se asegura que las direcciones de los encabezados cumplan el formato estándar.

3) ***Limitaciones:*** Uno de los problemas de Postfix al realizar envíos masivos, es que la cola de los correos entrantes, o "maildrop", está implementada como un subdirectorio con un conjunto de archivos que representan los correos, lo que

significa que al momento de escribir una gran cantidad de correos en la cola, se generará una gran actividad de entrada y salida. Por otro lado, se corre el riesgo de saturar la memoria provocando un desbordamiento de la cola. El otro problema radica en que no se podría tener, por ejemplo, un servidor a dedicación exclusiva para almacenar los correos y otro que se encargue de enviarlos; la arquitectura de Postfix no está diseñada para tales fines, de forma que todo tiene que estar centralizado en un único servidor, lo que disminuye la capacidad para aprovechar al máximo los recursos de cada máquina disponible.

En el caso en que Postfix, por alguna razón dejase de enviar correos, requeriría de un reinicio manual ya que no hay una reasignación automática que permita que los correos sean entregados de alguna otra manera alternativa. Las razones por las cuales Postfix puede detener su ejecución son fallas críticas del sistema, tales como corrupción del disco, problemas de memoria u otras.

A partir de esta problemática, una de las modificaciones que decidimos realizar a la implementación original de Postfix, fue separar su código en dos módulos: uno que se encargue de recibir y almacenar los correos entrantes por enviar y otro que se dedique exclusivamente al envío de dichos correos. Además se propone la utilización de una Base de Datos como representación de la cola de correos entrantes, para reducir el posible *overhead* de entrada y salida y con el fin de facilitar la distribución de la carga de trabajo entre los servidores.

- Minimizar la recuperación manual de fallas.
- Garantizar el 100% de los envíos.
- Garantizar el mantenimiento de la plataforma ante una falla, sin incurrir en un impacto negativo del rendimiento.

III. OPTIMUS: Plataforma de Distribución Masiva de Correos Electrónicos

En el marco del auge de los sistemas distribuidos, este trabajo propone una adaptación de los servidores de envío masivo de correo electrónico a una arquitectura distribuida que, gracias a los avances tecnológicos en esta materia, permita superar así las limitaciones actuales de desempeño, tolerancia a fallas y seguridad de las que aún adolecen los servidores de correo en la actualidad para su uso a gran escala.

Con la finalidad de demostrar la factibilidad de la implementación distribuida de una plataforma de envío masivo de correos electrónicos, se evaluaron los dos servidores más populares en la actualidad y de código abierto: Sendmail y Postfix.

Sendmail tiene la ventaja de presentar una gran versatilidad para su configuración, sin embargo su código es completamente monolítico, difícil de comprender y mucho más de extender con los cambios necesarios para ajustarlo a las nuevas necesidades. Por otro lado, Postfix, además de ser robusto, presenta un código modular, donde cada funcionalidad está impecablemente definida en un módulo separado y muy apegado a las definiciones teóricas, lo cual facilita su comprensión y su extensión. Por lo anterior, se seleccionó Postfix como el servidor de correo a utilizar para adaptarlo a la plataforma planteada. Hasta donde pudimos verificar y donde alcanza nuestro conocimiento, no existe un software de distribución masiva de correos electrónicos como el propuesto en este trabajo.

Las próximas subsecciones presentan la descripción detallada de la arquitectura de OPTIMUS, que comprende una base de datos NoSQL, una arquitectura basada en agentes P2P que facilita la implementación del balance de carga y estrategias de tolerancia a fallas.

A. Adaptaciones a Postfix: Incorporación de una Base de Datos NoSQL

Con la finalidad de facilitar la distribuición y asignación de los correos electrónicos, se reemplazó el manejo de dos de las principales colas que utiliza Postfix en su versión original. Se decidió utlizar una base de datos que gestione y centralice las colas de correos entrantes y diferidos para cualquier configuración posible de la plataforma. A continuación se especifican los cambios realizados a Postfix:

- **maildrop**: En la versión original de Postfix, está implementado como un directorio, en donde cada correo se representa como un archivo. En cambio, en la versión modificada de OPTIMUS, se migró el *maildrop* a una Base de Datos NoSQL para centralizar la cola, en caso de haber múltiples servidores Postfix, en una sola Base de Datos.
- **sendmail & postdrop**: En Postfix, estas dos funciones se encuentran en archivos distintos y entre los dos se encargan de escribir en el *maildrop*, con el formato pertinente, los correos a ser enviados. En la versión adaptada a OPTIMUS, se unificaron estos archivos en uno solo que inserte en la Base de Datos el correo especificado, conservando el formato original que utiliza Postfix.
- **pickup**: Originalmente, el *pickup* escanea constantemente el *maildrop*, revisa si hay correos por enviar y se dispone a mandar cada uno de los mensajes que allí se encuentran. Por otro lado, con las modificaciones realizadas para OPTIMUS, el *pickup* recibe a través de una comunicación vía *sockets* con un agente de correos, al que se le denominó *mail agent*, los identificadores de los correos que le corresponde mandar, que se encuentran en una Base de Datos local, luego selecciona cada correo y lo envía. Esto convierte al *pickup* modificado en un servidor que escucha constantemente dichas peticiones.
- **deferred**: Similar a los cambios realizados para simular la cola *maildrop*.

OPTIMUS utilza una base de datos conocida como *Redis* que funciona bajo el paradigma NoSQL [16]. Es un servidor que mantiene en memoria estructuras de datos del tipo clave-valor, lo que proporciona un gran desempeño y a su vez puede ser usada como una base de datos persistente. Normalmente, las base de datos NoSQL son recomendadas para aplicaciones que se benefician del esquema libre que proporciona este paradigma y que requieren un gran desempeño y escalabilidad. OPTIMUS tiene la particularidad de ser una plataforma con un gran número de operaciones de escritura por lo que *Redis*

es el candidato ideal para almacenar los correos electrónicos que serán procesados para su envío.

Como se observa en la Figura 2, la arquitectura original de Postfix no puede separarse en varios módulos que se ejecuten en distintos servidores, mientras que con los cambios realizados, es posible separar las funciones del *sendmail* y del *postdrop* originales, para que escriban en una Base de Datos que puede ser remota, mientras que a partir del *pickup* en adelante se encarguen, desde otro posible servidor, de enviar los correos. Otro aspecto importante a resaltar es la utilización de dos niveles de Bases de Datos NoSQL que mantienen en memoria principal la información esencial para el envío de los correos. El primer nivel es el remoto, que consta de una visión global y contiene la información acerca de todos los correos de la plataforma, facilitando su distribución. El segundo nivel es local, conformado por una Base de Datos local que dispondrá para cada servidor de correo, los mensajes delegados a éstos. Esta decisión de diseño se tomó en aras de minimizar las conexiones a la Base de Datos global que podría generar un *overhead* elevado o incluso llegar al límite de conexiones. De esta forma, se filtra la cantidad de accesos a la Base de Datos global por medio de los agentes *Scheduler* e *Info*, cuyas funcionalidades serán explicadas en la próxima sección.

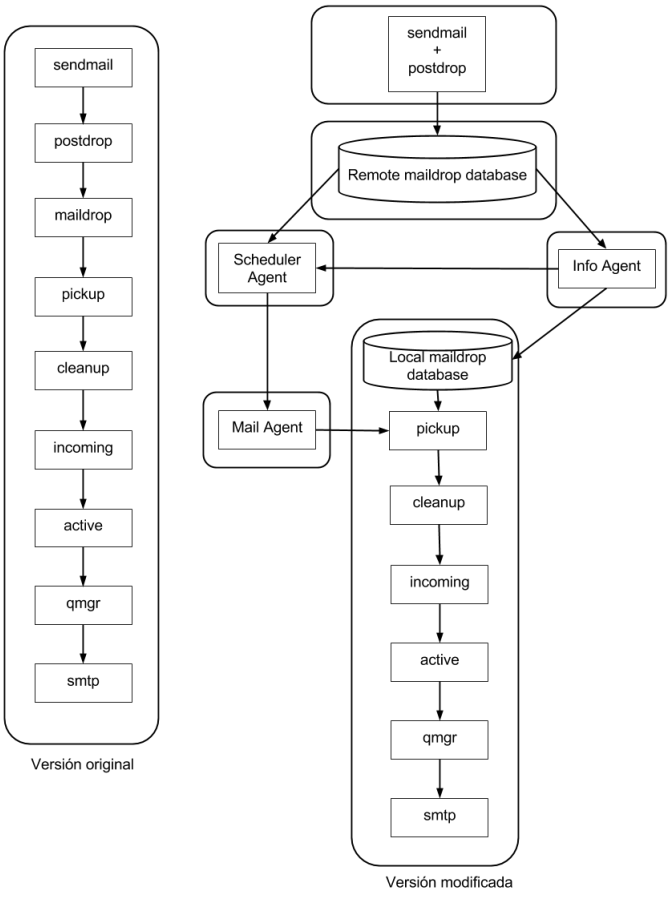

Figura 2: Flujo Original vs Flujo Modificado

B. Arquitectura P2P Multiagentes y Balance de Carga

La arquitectura de OPTIMUS se basa en capas (ver Figura 3), las cuales están representadas por tres agentes que se valen de las Bases de Datos NoSQL para llevar a cabo el envío de los correos:

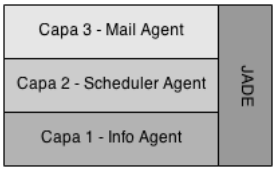

Figura 3: Arquitectura de OPTIMUS Basado en Capas

1) ***Capa 3 - Mail Agent****:* Agente encargado de contactar al demonio *pickup* de Postfix y enviarle los identificadores de los correos a ser procesados. Este agente recibe una notificación del *Scheduler Agent* la cual le indica qué lote de correo debe procesar. Luego, se dispone a enviar a Postfix todos los correos de ese lote. Todo *Mail Agent* está asociado a un único servidor Postfix. Esta capa no ofrece balance de cargas, ya que es quien está en contacto directo con el servidor Postfix y por lo tanto, el último eslabón de la cadena de envío; no obstante, los *Mail Agents* presentan rutinas de recuperación de fallas y se comunican con su base de datos local, en donde están representadas las colas de Postfix para almacenar los correos que le corresponden para su posterior envío.

2) ***Capa 2 - Scheduler Agent****:* Este agente es responsable de notificar y supervisar el envío de correo de uno o varios *Mail Agents* asociados. Recibe una notificación del *Info Agent* para indicarle la presencia de nuevos correos en la plataforma y posteriormente le notifica a los *Mail Agents* asociados si tienen correos por enviar, indicándoles el lote de mensajes respectivo. Luego monitorea los envíos y en caso de haber alguna demora en alguno de sus servidores de correo, redistribuye la carga. Los *Scheduler Agents* presentan tres características importantes: se encargan de balancear la carga al limitar el número de *Mail Agents* que pueden monitorear, de acuerdo a la cantidad de *Schedulers* y *Mail Agents* registrados en la plataforma. Por otro lado, se comunican con las bases de datos locales de sus *Mail Agents* para insertar los correos nuevos que deben enviar, de acuerdo a la información que obtengan de la base de datos global. Por último, presentan mecanismos de tolerancia a fallas, las cuales se detallarán más adelante.

3) ***Capa 1 - Info Agent****:* Este agente monitorea la base de datos global para determinar cuándo hay lotes de correos nuevos en la plataforma, enumerados por el *Sendmail*, según el orden de envío, y distribuye la carga de los correos por enviar entre los *Mail Agents* disponibles. Una vez distribuida la carga, notifica a los distintos *Scheduler Agents* que hay nuevos correos por procesar. Los *Info Agents* se enumeran de acuerdo al orden en el que inicien su ejecución, de manera que procese los lotes que sean múltiplos de su identificador, tal como se muestra en la Figura 4. De esta manera se garantiza que cada lote sea procesado por un *Info* distinto. Estos agentes se encargan de balancear la carga de los *Mail Agents*, al distribuir equitativamente la cantidad de correos que debe enviar cada

uno, y presenta mecanismos de tolerancia a fallas que se discutirán posteriormente.

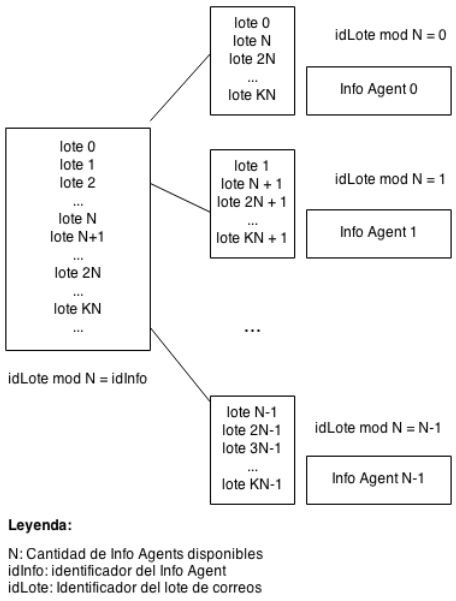

Figura 4: Distribución de Lotes de Correo entre Info Agents

Para establecer la plataforma multiagentes P2P se utilizó JADE (*Java Agent DEvelopment*), *framework* de software libre que simplifica el desarrollo de una arquitectura multiagente bajo el estandar FIPA. Se decidió utilizar esta herramienta gracias a que facilita la comunicación entre los agentes y su monitoreo. JADE provee mecanismos que permiten detectar eventos sobre los agentes que facilitan la gestión de los mismos dentro de OPTIMUS [17].

Cada uno de los agentes de las capas de la arquitectura de OPTIMUS, está implementado como un agente JADE y se utiliza este *framework* para gestionar la comunicación entre ellos, el ciclo de vida, la información de los servicios que ofrece cada uno y el estado de cada agente. Esto es posible gracias a los agentes inherentes al *framework* enumerados a continuación:

- *Agent Communication Channel (ACC)*: Provee el servicio de transporte de mensajes.
- *Agent Management System (AMS)*: Se encarga de la creación, eliminación, migración y monitoreo del ciclo de vida de los agentes. En la Figura 5 se ejemplifica el caso en el que un agente muere y el *AMS* detecta esta situación y lo notifica, enviando un evento, al resto de los agentes.
- *Directory Facility (DF)*: Se encarga de proveer información precisa y actualizada de los servicios que ofrecen los distintos agentes en la plataforma.

OPTIMUS se diseñó contemplando la eficiencia y distribución de tareas a través de todos sus agentes. En este sentido, los *Scheduler Agents* son responsables de notificar y supervisar a un conjunto de *Mail Agents*. La asignación *Scheduler Agent - Mail Agents* es dinámica y se realiza cuando un agente nace o termina cumpliendo con el Protocolo de Agentes OPTIMUS,

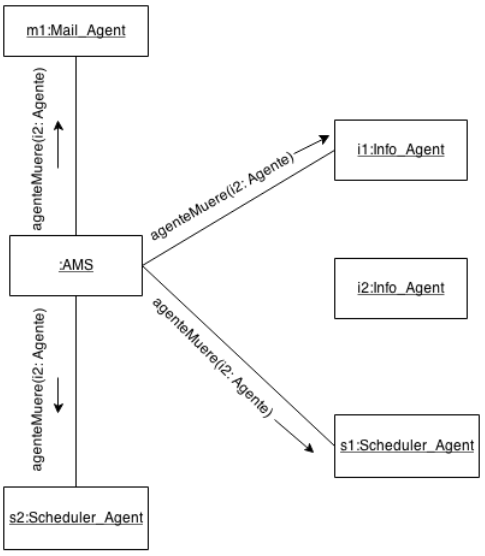

Figura 5: AMS: Notificar que un Agente Murió

con la finalidad de mantener el balance de carga entre los *Scheduler Agents*.

C. Protocolo de Agentes OPTIMUS: Balance de Carga

Cuando un *Mail Agent* nace, se llevan a cabo las siguientes operaciones:

- El *Mail Agent* envía un saludo a todos los *Scheduler Agents* disponibles en la plataforma anunciando su nacimiento.
- Cada *Scheduler Agent* realiza lo siguiente:
  - Consulta en la plataforma JADE el número de *Mail Agents* dispobibles.
  - Consulta en la plataforma JADE el número de *Scheduler Agents* disponibles.
  - Divide el número de *Mail Agents* entre el número de *Scheduler Agents* para obtener el máximo de *Mail Agents* que puede gestionar.
  - El *Scheduler* responde aceptando o rechazando el saludo. El saludo es aceptado si se cumple que: (i) El número de *Mail Agents* que tiene asociado es estrictamente menor que el máximo que puede gestionar, o (ii) Todos los *Schedulers* tienen asociados la misma cantidad de *Mail Agents*. El saludo es rechazado en caso contario.
- El *Mail Agent* recibe todas las respuestas y selecciona la primera que aceptó su saludo descartando el resto. Luego envía un *agent-hello* al *Scheduler Agent* seleccionado indicando su asociación.

Por ejemplo, en la Figura 6 hay dos *Scheduler Agents* y tres *Mail Agents*. Cuando nace *Mail Agent 4*, la cantidad máxima de cada *Scheduler Agent* sería 2. Por lo tanto, *Scheduler 1* rechaza el saludo y *Scheduler 2* lo acepta.

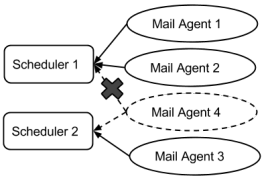

Figura 6: Nacimiento de un Mail Agent

Cuando un nuevo *Scheduler Agent* nace, se llevan a cabo las siguiente operaciones:

- El *Scheduler Agent* envía una notificación a todos los *Scheduler Agents* disponibles en la plataforma.
- Cada *Scheduler Agent* realiza lo siguiente:
  - Consulta en la plataforma JADE el número de *Mail Agents* disponibles.
  - Consulta en la plataforma JADE el número de *Scheduler Agents* disponibles.
  - Divide el numero de *Mail Agents* entre el número de *Scheduler Agents* antes de su nacimiento para obtener la cantidad máxima de *Mail Agents* que puede gestionar.
  - Si la cantidad de *Mail Agents* que tiene asociado es mayor que el máximo a gestionar, migra al nuevo *Scheduler Agent* la cantidad sobrante de *Mail Agents*.
  - Si la distribución de *Mail Agents* es equitativa entre los *Schedulers*, cada uno migra un *Mail Agent* al nuevo *Scheduler Agent*, sin que el nuevo llegue al máximo.

Por ejemplo, en la Figura 7 nace el *Scheduler 3* y notifica al resto de los *Schedulers* sobre su nacimiento (incluyéndose). Como antes habían 6 *Mail Agents* y dos *Schedulers*, el máximo era 3. Luego del nacimiento de *Scheduler 3*, el nuevo máximo sería 2, por lo que la carga de *Mail Agents* se redistribuye.

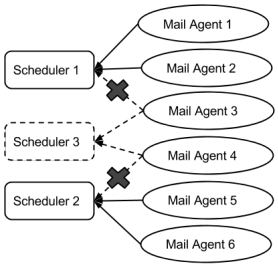

Figura 7: Nacimiento de un Scheduler Agent

D. Tolerancia a Fallas

Una de la principales motivaciones de este proyecto es la realización de una plataforma distribuida de envío masivo de correo electrónico que sea robusta y tolerante a fallas.

Como se mencionó anteriormente, la arquitectura de OPTIMUS está basada en capas, donde cada una de éstas desempeña un rol importante para la supervisión del resto de la plataforma. En líneas generales, cada capa ofrece tolerancia a sus capas subsiguientes.

El alcance de esta sección se enfoca en evaluar la tolerancia a fallas y recuperación de cada una de las capas de OPTIMUS y del resto de sus componentes.

*1) **Tolerancia de Ausencias**:* Al momento de distribuir los correos es posible encontrar escenarios de ausencia total de los agentes de las capas de OPTIMUS.

- Envío de correos en ausencia de agentes de capa 1 (*Info Agent*): Como se explicó anteriormente en el flujo normal de OPTIMUS, cuando se desee enviar un nuevo lote de correos, el *Sendmail* inserta el lote en la Base de Datos, como "lote entrante". En caso de que no exista ningún *Info Agent* que procese los lotes nuevos, estos persisten en la Base de Datos global hasta que se inicie algún *Info Agent*.
- Envío de correos en ausencia de agentes de capa 2 y/o de capa 3 (*Scheduler Agent* o *Mail Agent*): Cuando un *Info Agent* toma un lote de los "lotes entrantes", se asegura de que existan tanto algún *Scheduler Agent*, como algún *Mail Agent* disponibles para terminar de procesar y distribuir los correos. Este procedimiento se ilustra en el algoritmo mostrado en la Figura 8.

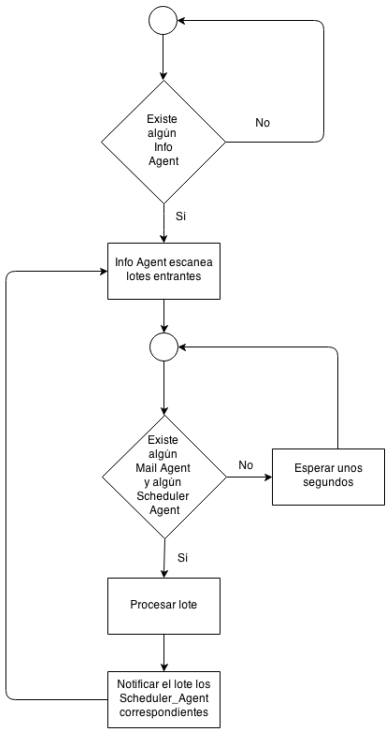

Figura 8: Algoritmo de Tolerancia a Fallas en Ausencia de Capa 2 y/o Capa 3

*2) **Tolerancia a Fallas de la Capa 1 - Info Agent**:* Cuando un *Info Agent* muere, el resto de los agentes de tipo *Info* recibe una notificación y reajustan sus identificadores, de manera de que se redistribuyan los lotes entre los *Info Agents* restantes, como se muestra en la Figura 9. Luego, cada uno consulta en la base de datos los lotes existentes, de forma tal que retome todos los lotes de correo que le pertenecían al agente que se detuvo. Esta rutina se ilustra en la Figura 10.

*3) **Tolerancia a Fallas de la Capa 2 - Scheduler Agent**:* Cuando un *Scheduler Agent* termina inesperadamente, se ejecutan los siguientes pasos:

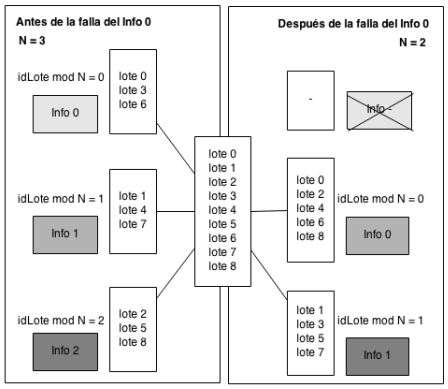

Figura 9: Recálculo de Identificadores de Info Agents

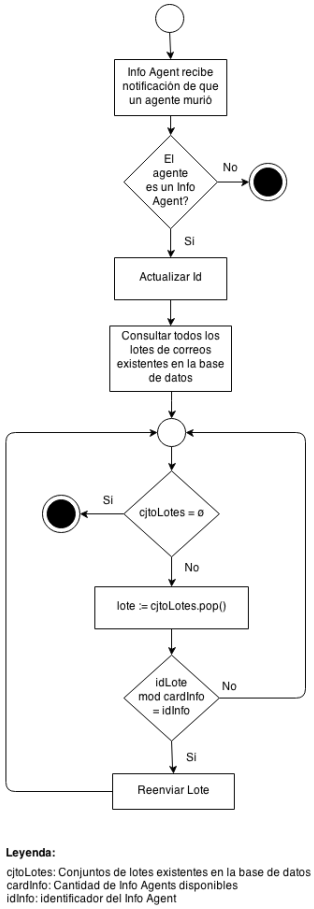

Figura 10: Algoritmo de Tolerancia a Fallas Capa 1

- Cada uno de sus *Mail Agents* detecta (a través de una excepción de JADE) que su *Scheduler Agent* terminó e inician el protocolo de nacimiento de *Mail Agents* para ser asignado a otro *Scheduler Agent*. Por ejemplo, en la Figura 11, el *Scheduler 2* falla y cada uno de sus *Mail Agents* se distribuye entre el resto de los *Schedulers*.
- Un *Info Agent* de capa 1, detecta la muerte del *Scheduler Agent*, el *Info* obtiene todos los *Mail Agents* que le pertenecían al *Scheduler* que murió, revisa si estos *Mail Agents* siguen en ejecución y de ser así, reasigna los lotes que les correspondía enviar. En caso que algún *Mail Agent* también haya fallado, se crea un lote nuevo con todos los correos del *Mail Agent* fallido y se reenvían. La Figura 12, muestra el algoritmo del *Info Agent* en el caso de detectar la falla de un *Scheduler Agent*.

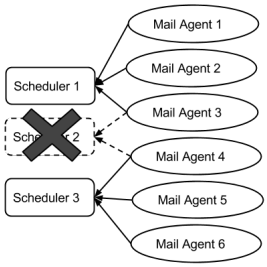

Figura 11: Redistribución de Cargas Cuando un Scheduler Agent Falla

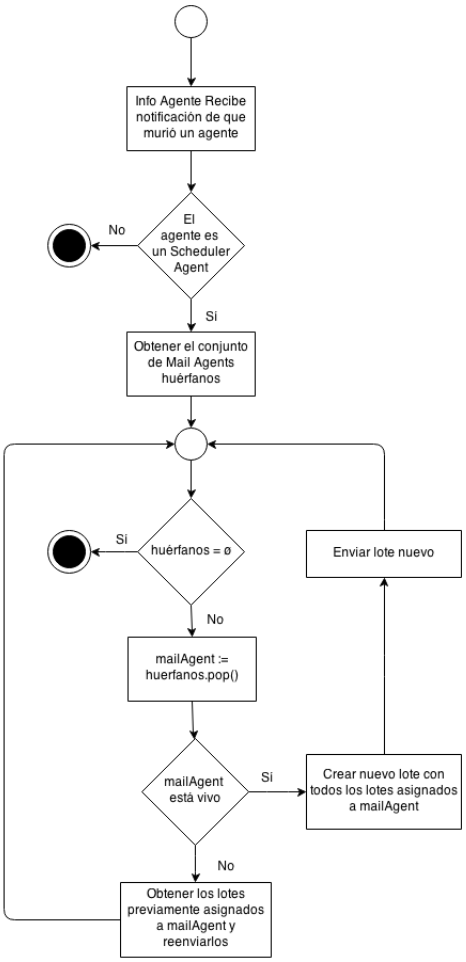

Figura 12: Algoritmo de Info Agent al Detectar Fallas Capa 2

4) ***Tolerancia a Fallas de la Capa 3 - Mail Agent****:* Cuando un *Mail Agent* termina inesperadamente, el *Scheduler Agent* responsable de ese *Mail Agent* detecta la falla (a través de una excepción de JADE) y realiza los siguientes pasos:

- Reasigna todos los lotes de correos que le correspondían al *Mail Agent* fallido, de acuerdo al algoritmo mostrado en la Figura 13.

- Inicia el protocolo de nacimiento de *Scheduler Agent* que le permite distribuir la carga de *Mail Agents* correctamente entre los *Scheduler Agents* disponibles.

Figura 13: Algoritmo de Tolerancia a Fallas Capa 3

Por otro lado, los *Scheduler Agents* revisan periódicamente si todos sus *Mail Agents* están enviando correos, de manera que detecte si alguno de ellos está retrasado en sus envíos, previendo que esto puede significar que *Postfix* presenta alguna falla. En este caso, se le solicita al *Mail Agent* retrasado, que compruebe su conexión con el servidor *Postfix* y en caso de no poder establecer esta conexión, el *Mail Agent* detiene su ejecución, forzando de esta forma que se reasigne su carga. Este comportamiento se ilustra en los algoritmos (para *Scheduler Agents* y *Mail Agents*) mostrados en la Figura 14.

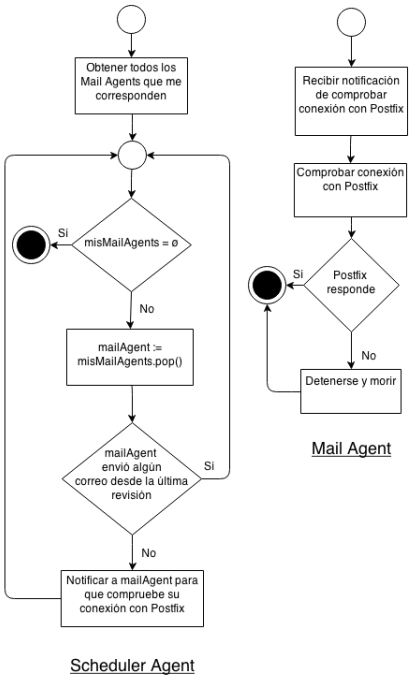

Figura 14: Algoritmos de Tolerancia a Fallas Capa 3: Mail Agent Demorado

5) ***Tolerancia a Fallas de JADE****:* El *framework* JADE permite crear plataformas distribuidas donde los agentes se ejecutan en contenedores asociados a la plataforma. Sin embargo, presenta una gran dependencia de un contenedor principal donde coexisten los agentes inherentes a JADE, el AMS y el DF. Esto último significa que existe un potencial punto de falla que puede ocasionar que toda la plataforma interrumpa su funcionamiento.

Combinando dos caracteristicas de JADE es posible desplegar una plataforma tolerante a fallas a nivel de la capa transversal de la arquitectura de OPTIMUS, que denominamos como *Capa JADE*. Gracias al *Main Replication* Service, se puede replicar el contenedor principal y el *AMS* que reside en él. De esta forma, si ocurre un falla y el contenedor principal deja de existir, las réplicas son capaces de detectar esto y comenzar las acciones de recuperación. Por otro lado, JADE proporciona el *Directory Facility* conocido como *DF* que provee información sobre los servicios que ofrecen los agentes en la plataforma, en el caso particular de este trabajo, se diferencian tres tipos de servicios para cada agente: *Info Agent*, *Mail Agent* y *Scheduler Agent*. Este catálogo también reside en el contenedor principal por lo que es importante proporcionar algún mecanismo que garantice su persistencia en caso de ocurrir alguna falla. Para lograr esto, JADE tiene la capacidad de vincular al *DF* con una Base de Datos de tal forma que se garantice la persistencia de la información contenida en este catálogo.

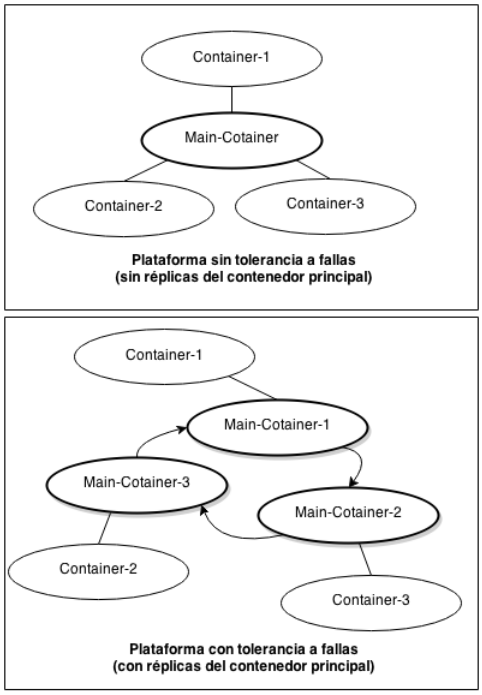

Figura 15: Tolerancia a Fallas JADE: Topología Estrella vs. Anillo [17]

La Figura 15 ilustra las topologías correspondientes a una plataforma en forma de estrella sin tolerancia y sin réplicas del contenedor principal, en contraposición a otra en forma de anillo que presenta réplicas del contenedor principal, las cuales monitorean al maestro y en caso de ocurrir una falla se reajustan, asignando un nuevo maestro. Por lo tanto, esta última provee tolerancia a fallas mientras que la primera no. La Capa JADE de OPTIMUS implementa la topología tolerante

a fallas.

La Tabla I muestra un resumen de las características que presenta cada uno de los agentes de OPTIMUS, en relación a los aspectos tratados en esta sección. Se puede observar que todas las capas de OPTIMUS ofrecen tolerancia a fallas y que las capas 1 y 2 son las encargadas del balance de carga de la plataforma.

Tabla I: Características de los Agentes de OPTIMUS

Característica	Capa 1	Capa 2	Capa 3	Capa JADE
Balance de Cargas	x	x		
Comunicación con Postfix			x	
Conexión con Base de Datos Global	x	x	x	
Conexión con Base de Datos Local		x	x	
Tolerancia a Fallas	x	x	x	x

IV. Evaluación Experimental

A. Evaluación del Rendimiento

Con la intención de evaluar el rendimiento de OPTIMUS frente a la versión original de Postfix, realizamos experimentos con lotes de correos de distintos tamaños y separamos las pruebas en dos escenarios: el primero, para medir el tiempo de inserción y procesamiento de los correos y el segundo, para medir el tiempo de envío de los mensajes.

1) ***Configuración de la Plataforma de Prueba****:* Para la implementación de OPTIMUS, se utilizó Redis 2.8.8 como Base de Datos, JADE 4.3 como plataforma P2P y se modificó la versión 2.11.0 de Postfix. Para las pruebas se utilizaron tres *PCs* con las siguientes especificaciones:

- Host 1: Intel(R) Pentium(R) D CPU 3.40GHz. SO: Debian 6 32bits. RAM: 1GB
- Host 2: Intel(R) Pentium(R) D CPU 3.40GHz. SO: Debian 6 32bits. RAM: 1GB
- Host 3: Intel(R) Pentium(R) D CPU 3.40GHz. SO: Debian 6 32bits. RAM: 1GB

2) ***Escenarios****:* La evaluación experimental se realizó utilizando las tres máquinas descritas anteriormente, de la siguiente manera: el *Host 1* desempeñó la función de enviador, el *Host 2* alojaba la Base de Datos Global y el *Host 3* se encargaba de recibir los mensajes.

Esta evaluación consistió en dos pruebas. En la primera se midió el desempeño de Postfix y de OPTIMUS insertando una cantidad determinada de correos en la cola *maildrop*. Por otra parte, en la segunda prueba se evaluó el desempeño tomando en cuenta el tiempo de envío de una cantidad determinada de correos. Ambas pruebas se realizaron utilizando lotes de 100, 1000, 10.000 y 100.000 mensajes, los cuales eran idénticos, y cada una de las prueba se repitió diez veces. Se reportan los promedios de las diez repeticiones.

3) ***Desempeño****:* La Tabla II muestra los resultados del desempeño de ambas arquitecturas insertando los mensajes en el *maildrop* y procesándolos para su posterior envío.

Tabla II: Tiempo Promedio de Inserción de Correos a *maildrop* de Postfix vs. OPTIMUS con Base de Datos Remota

Cantidad de correos	Tiempo de inserción (seg)		Porcentaje de mejora
	Postfix	OPTIMUS	
100	0.4997	1.83696	-267.6
1.000	5.2117	26.9182	-416.5
10.000	55.084	226.0528	-310.4
100.000	2654.62	2417.4916	8.9

Se observa que para un número pequeño de correos, el desempeño de OPTIMUS está muy por debajo del desempeño de Postfix, ya que nuestra arquitectura realiza una gran cantidad de operaciones, como distribución de cargas o accesos a Bases de Datos locales y remotas, lo cual ralentiza el procesamiento. No obstante, a medida que se aumenta el número de mensajes de manera significativa, OPTIMUS comienza a presentar leves mejoras en su desempeño.

Por otro lado, la Tabla III, refleja los resultados del desempeño de ambas arquitecturas para el envío de los correos.

Tabla III: Tiempo Promedio de Envío de Postfix vs. OPTIMUS

Cantidad de correos	Tiempo de inserción (seg)		Porcentaje de mejora
	Postfix	OPTIMUS	
100	1	1.1	-10
1.000	6.9	10.1	-46.4
10.000	110.7	101	8.8
100.000	1457.375	1022	30

En este caso, el desempeño de OPTIMUS continúa por debajo del desempeño de Postfix para cantidades de mensajes pequeñas, pero alcanza un 30% de mejora para el caso de los 100 mil correos.

Las Figuras 16 y 17 reflejan un comportamiento lineal del desempeño de OPTIMUS, tanto para el caso de procesamiento e inserción como para el de envío de correos, mientras que para Postfix, el desempeño empeora a medida que aumenta la cantidad de correos y no mantiene la linealidad.

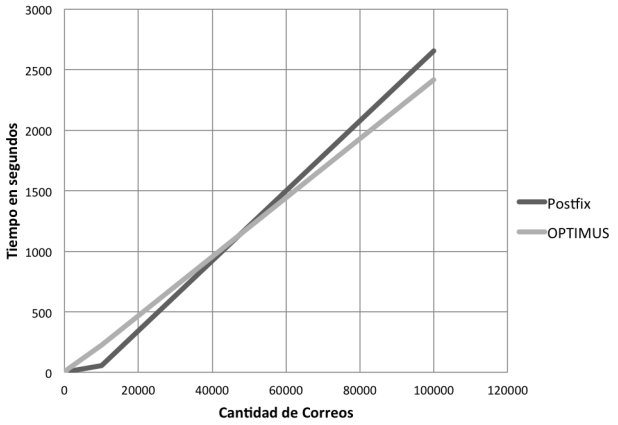

Figura 16: Inserción: Postfix vs. OPTIMUS con Base de Datos Remota

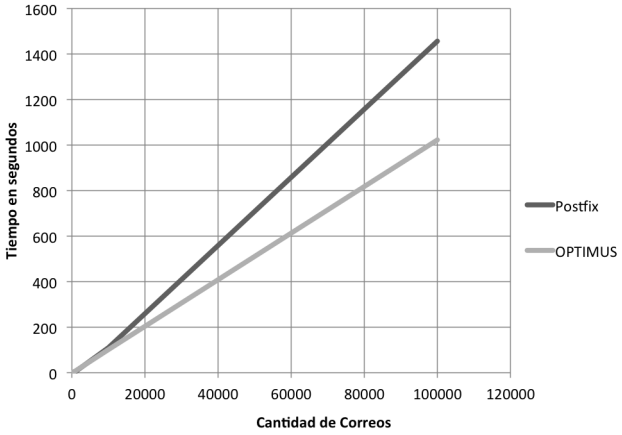

Figura 17: Envío: Postfix vs. OPTIMUS

B. Análisis del Impacto de la Tolerancia a Fallas sobre el Rendimiento

Una de los principales aspectos que caracteriza a OPTIMUS es la tolerancia a fallas que busca principalemente:

- Garantizar el envío de todos los correos electrónicos.
- Reasignar adecuadamente los correos que no pudieron ser entregados por motivo de alguna falla.

Estos objetivos se logran, aún en presencia de fallas, con la implementación de los algoritmos de tolerancia a fallas y recuperación. Realizamos un análisis experimental para determinar el nivel de intrusividad de la tolerancia a fallas, durante la actividad normal y el nivel de envío de correos duplicados ante una recuperación de una falla.

*1) **Nivel de Intrusividad**:* Cuando ocurre una falla de cualquiera de los agentes de OPTIMUS o JADE, ésta es detectada vía excepciones. Por lo tanto, no implica un *overhead* de tiempo durante la actividad normal del sistema (en ausencia de fallas). Esto implica que los algoritmos de tolerancia a fallas sólo son ejecutados en presencia de fallas. Un caso diferente ocurre cuando la falla se genera a nivel de Postfix. Para detectar estas fallas, los *Scheduler Agents* monitorean cada cierto tiempo que los lotes de sus *Mail Agents* se estén enviando. Este monitoreo períodico podría ocasionar intrusión en el rendimiento de la plataforma. Para medir este nivel de intrusión, realizamos dos pruebas con dos escenarios distintos, ambos con ausencia de fallas:

- Envío sin monitoreo: Se realizaron 10 envíos con lotes de 10 mil correos, sin el comportamiento que monitorea que los *Mail Agents* de un *Scheduler* estén enviando correos. Configuración de agentes: un *Mail Agent*, un *Scheduler* y un *Info Agent*.
- Envío con monitoreo: Se realizaron 10 envíos con lotes de 10 mil correos, con el comportamiento que monitorea que los *Mail Agents* de un *Scheduler* estén enviando correos. Configuración de agentes: un *Mail Agent*, un *Scheduler* y un *Info Agent*.

En la Tabla IV se observan los resultados de los experimentos realizados bajo los escenarios anteriormente descritos. Estos muestran que esta rutina de tolerancia a fallas incurre solo en un 11% de intrusión para una configuración con una sola instancia de cada tipo de agentes. Esta intrusión se debe a que la rutina se ejecuta muy seguido, y podría aminorarse si se amplía el período de tiempo entre una ejecución y otra.

Tabla IV: Intrusión del Comportamiento Monitor en el Desempeño de OPTIMUS

OPTIMUS sin monitoreo (segundos)	OPTIMUS con monitoreo (segundos)	% Intrusión
207	232.5	11

En resumen, tras llevar a cabo el experimento anterior, se observó que la implementación de la tolerancia a fallas no causa un impacto sustancial en el rendimiento de OPTIMUS en ausencia de fallas, dado que las rutinas de tolerancia únicamente se ejecutan una vez detectadas las fallas o, en el caso del monitoreo de los *Mail Agents*, el comportamiento se solapa con otras rutinas de envío y procesamiento y no genera un impacto negativo considerable en el rendimiento.

*2) **Correos Duplicados**:* Otra posible consecuencia de estos algoritmos de tolerancia y recuperación, es la aparición de correos duplicados. Esto se debe a que cuando ocurre una falla en algún *Mail Agent*, sus servidores Postfix pueden seguir procesando correos que no han sido aún eliminados de la Base de Datos global. Luego, cuando se activa una rutina de tolerancia a fallas y reasigne la carga que tenía asignado el *Mail Agent*, se redistribuyen correos que ya están siendo procesados por otro servidor de correo, causando que estos mensajes se envíen duplicados. Es decir, los correos duplicados son la consecuencia de no poder sincronizar de forma atómica la base de datos local, con la global.

Con la finalidad de obtener un porcentaje estimado de la cantidad de correos duplicados que podrían generarse en un envío de OPTIMUS, se llevaron a cabo 10 envíos de 10 mil correos, con dos *Mail Agents*, un *Scheduler Agent* y un *Info Agent*. En todas las iteraciones, se forzó una falla en uno de los *Mail Agents* y se esperó a que se reasignaran las cargas en cada caso. Al finalizar el experimento, se estimó que, en promedio, se duplicaron 8% de los correos enviados.

Es posible evitar los mensajes duplicados, pero esto requiere de un registro excesivo de todas las acciones que se realizan en la plataforma, lo que podría derivar en un cuello de botella y un consumo elevado de recursos. Por esta razón, OPTIMUS se centra en garantizar la entrega de la totalidad de los correos, más que en garantizar la unicidad de cada uno de los mensajes e ignora la posibilidad de que se duplique una pequeña cantidad de correos.

V. OPTIMUS Frente a Postfix Original

La principal característica de la arquitectura de OPTIMUS es la tolerancia a fallas y la distribución de tareas, lo cual garantiza robustez y el ahorro de recursos y tiempo durante el envío de correos. A continuación se listan las diferencias más importantes entre OPTIMUS y Postfix.

- Una ventaja de OPTIMUS sobre Postfix, es su capacidad de detectar y recuperar las fallas que puedan ocurrir en

sus servidores activos de correos, mientras que Postfix, a pesar de que es capaz de detectar alguna falla y mantener la persistencia de los correos, no presenta mecanismos de recuperación. Esto hace a OPTIMUS más robusto y tolerante a fallas que Postfix.

- Otro aspecto importante de OPTIMUS es la capacidad de modularizar los componentes y ubicarlos en distintas máquinas, lo que lo convierte en una plataforma escalable.
- La utilización de una Base de Datos NoSQL que comprime la información y la mantiene en memoria principal, minimiza de forma significativa los accesos a disco y la utilización del espacio de memoria, principalmente para los casos de cargas pesadas de correos. Esto permite que OPTIMUS presente un mejor desempeño con grandes cantidades de mensajes.
- En materia de seguridad, nuestra arquitectura tiene algunas debilidades a tomar en consideración. El hecho de no revisar la integridad de los correos, tras la constante transferencia de mensajes desde la Base de Datos global hasta las Bases de Datos locales o la misma existencia de una Base de Datos centralizada que al sucumbir ante algún ataque, ocasionaría el colapso de toda la plataforma. Por otro lado, Postfix es totalmente centralizado y presenta mecanismos como la ejecución de sus procesos en *chroot*, que elevan su nivel de seguridad.
- OPTIMUS permite distribuir equitativamente el balance de cargas entre un conjunto de servidores Postfix, con la finalidad de optimizar el envío de correos y obtener mejoras en el desempeño. Por esta razón OPTIMUS consigue un mejor rendimiento para cargas pesadas de mensajes, que una sola instancia de Postfix.

La Tabla V resume la comparación de las características antes mencionadas. Cabe destacar que las debilidades de seguridad están planteadas como trabajo futuro. Una solución factible es la utilización del protocolo SSL para la transferencia de los correos de la Base de Datos principal, hasta las locales y para la comunicación entre agentes.

Tabla V: Cuadro Comparativo entre Postfix y OPTIMUS

Características	Postfix	OPTIMUS
Tolerancia a fallas	bajo	alta
Robustez	mediana	alta
Escalabilidad	baja	alta
Desempeño con cargas pesadas	bajo	alto
Seguridad	alta	baja

VI. Conclusiones y Trabajo Futuro

En el presente trabajo se plantea la arquitectura de OPTIMUS, una plataforma distribuida para el envío masivo de correos electrónicos utilizando una adaptación del servidor de correo Postfix con un sistema multiagente P2P.

De acuerdo a los resultados experimentales y las comparaciones realizadas, se puede concluir que la arquitectura planteada es más eficiente en utilización de recursos y en desempeño que Postfix para grandes cantidades de mensajes, lo cual representa una ventaja contundente a la hora del envío masivo de correos. Por otra parte, al manejar múltiples servidores de correo Postfix, OPTIMUS muestra una mejora en el desempeño gracias al balance de cargas entre los distintos agentes y, por lo tanto, entre los distintos servidores de correo. Aunado a esto, OPTIMUS ofrece mecanismos de tolerancia a fallas que no incurren en intrusión significativa en el desempeño y que garantizan el envío total de los mensajes. De esta manera, se puede afirmar que esta plataforma es robusta y tolerante a fallas.

Como trabajo futuro se desea incorporar un componente de Inteligencia Artificial que permita decidir, de acuerdo a la configuración de las máquinas y su estado en tiempo real, la distribución más eficiente posible de los correos electrónicos entre los distintos servidores que existen en la plataforma, optimizando de esta manera el uso de los recursos. Además, se debe comprobar el comportamiento aparentemente lineal que presenta el desempeño de OPTIMUS con pruebas que consideren lotes de correos más grandes que los utilizados en este trabajo.

Por último, pretendemos realizar mejoras en materia de seguridad, mediante la utilización de SSL para el cifrado de la comunicación entre Bases de Datos y entre los agentes, así como otras herramientas de seguridad para proteger la integridad de la máquina donde se ejecuta la Base de Datos principal.

References

[1] N. Christenson, *Sendmail Performance Tunning*, 1st edition, Pearson Education, September 2002.

[2] *The Ultimate Mobile Email Statistics Overview*, http://www.emailmonday.com/mobile-email-usage-statistics.

[3] D. Chaffey, *Email Marketing Statistics 2014*, 2014.

[4] S. Radicati and J. Levenstein, *Mail Statistics Report, 2013-2017*, Palo Alto, California, USA, 2013.

[5] D. Nesset, *Massively Distributed Systems: Design Issues and Challenges*, in proceedings of the Workshop on Embedded Systems, Cambridge, Massachussets, USA, 1999.

[6] *Sendmail.Now*, http:www.sendmail.com/sm/open_source.

[7] *The Postfix Home Page*, http://www.postfix.org.

[8] F. Rosato, J. Argüello, and Y. Cardinale, *OPTIMUS: OPTImized Mail distribUtion System*, in proceedings of the XL Latin American Computing Conference (CLEI 2014), Montevideo, Uruguay, September 2014.

[9] N. Cox, *Electronic Messaging*, 1st edition, Auerbach Publications, November 1999.

[10] F. Avolio and P. Vixie, *Sendmail: Theory and Practice*, 1st edition, Digital Press, March 1995.

[11] R. Blum, *Postfix*, 1st edition, Sams Publishing, May 2001.

[12] K. Dent, *Postfix - The Definitive Guide: A Secure and Easy-to-Use MTA for Unix. Applications and Inter-Networking Technologies*, 1st edition, O'Reilly Media, 2003.

[13] C. Hunt, *Sendmail Cookbook*, 1st edition, O'Reilly Media, December 2003.

[14] J. Klensin, N. Freed, M. Rose, E. Stefferud, and D. Crocker, *SMTP Service Extensions*, RFC 1869, November 1995.

[15] J. Myers, *Local Mail Transfer Protocol*, RFC 2033, October 1996.

[16] T. Macedo and F. Oliveira, *Redis Cookbook*, 1st edition, O'Reilly Media, August 2011.

[17] F. Luigi, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*, 1st edition, Willey, April 2007.