Um servidor de e-mail distribuído, flexível, escalável e tolerante a intrusões*

Leonardo Oliveira , Davi Arnaut , Carlos Maziero

[1]Programa de Pós-Graduação em Informática Aplicada
Pontifícia Universidade Católica do Paraná

{bispo,davi,maziero}@ppgia.pucpr.br

Abstract. *E-mail is an essential tool in today's Internet. However, architectures of production e-mail systems are prone to several problems that impact their scalability and open doors to dependability problems. Many solutions have been proposed to solve such deficiencies and to provide better scalability and reliability to such systems. This paper presents a distributed architecture for e-mail servers. Its design is centered on performance, scalability, configuration flexibility, and fault/intrusion tolerance aspects. The proposal uses techniques like autonomous cells struturing, storage fragmentation and scattering, mailbox cyphering, event-based mail processing and hook-based filtering.*

Resumo. *O e-mail é uma ferramenta essencial na Internet. No entanto, a arquitetura dos sistemas de e-mail convencionais sofre de problemas que limitam sua escalabilidade e abrem a porta para diversos tipos de ameaças. Alternativas vêm sendo propostas para sanar as deficiências e prover maior escalabilidade e confiabilidade aos sistemas de e-mail. Este artigo apresenta uma arquitetura distribuída para servidores de e-mail, cobrindo aspectos de desempenho, escalabilidade, flexibilidade de configuração, tolerância a faltas e a intrusões. Para tal, diversas técnicas foram empregadas, como a estruturação do sistema em células autônomas, fragmentação e dispersão do armazenamento, processamento de mensagens baseado em eventos e filtragem por ganchos (hooks).*

1. Introdução

Nos últimos dez anos o crescimento da Internet ocorreu de forma muito rápida, tanto em número de usuários quanto em complexidade da tecnologia. As ferramentas de e-mail se tornaram extremamente difundidas por serem um sistema de comunicação simples, barato e capaz de transportar vários tipos de informações em seu corpo. Estimativas feitas pelo IDC [IDC, 2003] apontam que em 2005 existirão cerca de 1,2 bilhões de caixas de correio eletrônico, com uma média de 3 caixas por pessoa, sendo que o tráfego diário de e-mail está previsto em aproximadamente 36 bilhões de mensagens.

O protocolo responsável pelo envio e recebimento de e-mail foi projetado para ser simples, pois na época de sua criação os e-mails eram restritos à comunidade acadêmica e a capacidade de processamento das máquinas era limitada. Com o aumento do uso dos e-mails, usuários mal intencionados passaram a utilizar esse sistema para propagar conteúdo malicioso, como vírus, *spams* e *scams*. Além disso, o aumento no uso de sistemas de e-mail levou à necessidade de tecnologias capazes de suportar de forma escalável, segura e confiável o armazenamento e o tráfego de mensagens eletrônicas.

*Projeto financiado pelo processo 401.581/2003-9 do edital de Software Livre CT-Info/CNPq 01/2003.

Este artigo apresenta uma arquitetura distribuída que oferece mecanismos de processamento de mensagens, tais como filtragem de *spam* e vírus, adição/remoção/modificação de anexos, modificação de cabeçalhos das mensagens, detecção de conteúdo indevido, triagem de mensagens, contabilização de uso e estatísticas detalhadas. A arquitetura aqui descrita foi concebida com forte preocupação nos aspectos de desempenho, flexibilidade, escalabilidade e tolerância a faltas e intrusões. Para tal, diversas técnicas foram empregadas, como a estruturação do sistema em células autônomas, fragmentação e dispersão do armazenamento, cifragem das caixas de correio, processamento de mensagens baseado em eventos e filtragem por ganchos (*hooks*).

O texto deste artigo está estruturado da seguinte forma: a seção 2 traz uma revisão das principais ameaças aos serviços de e-mail; a seção 3 apresenta os principais projetos realizados na área de servidores de e-mail distribuídos em larga escala; a seção 4 trás uma visão geral sobre o FDR, um modelo de armazenamento de dados seguro e tolerante a falta; a seção 5 apresenta a arquitetura desenvolvida neste projeto e detalha seus aspectos funcionais e não-funcionais; a seção 6 compara a arquitetura aqui apresentada com as demais arquiteturas presentes na literatura; por último, a seção 7 conclui o trabalho e delineia suas perspectivas de continuidade.

2. Ameaças ao serviço de e-mail

O sistema de e-mail foi inicialmente projetado para ser simples, pois seu público alvo era restrito à comunidade acadêmica. O protocolo SMTP, responsável pela transferência de e-mails entre servidores [Crocker, 1982, Resnick, 2001], não provê mecanismos robustos para autenticação e controle de acesso. Com a explosão no uso da Internet, o sistema de e-mail ganhou grande importância, expondo, com isso, as fragilidades das tecnologias envolvidas e vários problemas não previstos inicialmente. Dentre esses problemas destacam-se os *spams*, vírus e *scams* que comprometem o desempenho, robustez, segurança e usabilidade dos sistemas de e-mail atuais.

2.1. Spam

Spam ou *UCE* (*Unsolicited Commercial E-mail*) são os termos usados para designar mensagens não solicitadas (geralmente indesejáveis ao destinatário) que consomem recursos importantes (espaço em disco, banda de rede, etc) dos sistemas de computação. Estima-se que o tráfego médio diário de *spam* é de 17 bilhões de mensagens e que a tendência é aumentar para 23 bilhões até 2007 [IDC, 2004].

Os *spams* podem ocasionar diversos inconvenientes, como o não-recebimento de e-mails válidos (pela saturação das caixas de correio), o gasto desnecessário de tempo para receber mensagens indesejadas (fator crítico em conexões discadas), a perda de produtividade (aumento no tempo necessário para a leitura de e-mails), impacto na banda de rede (o tráfego de *spams* diminui a banda útil de rede das corporações), entre outros [Lindberg, 1999].

As principais técnicas de envio de *spam* são:

- **Entrega direta**: programas de entrega montam um servidor SMTP no computador *spammer* (gerador de *spam*) que enviam grandes volumes de mensagens diretamente a outros servidores de e-mail;
- ***Open relay***: um erro freqüente de configuração nas listas de controle de acesso de servidores de e-mail, conhecido como *open relay*, permite que um servidor seja usado como propagador involuntário de grandes volumes de *spam*;

- **Remetentes forjados**: fragilidades no protocolo SMTP permitem forjar campos dos cabeçalhos de e-mail, entre os quais o remetente; essa fragilidade é amplamente explorada por *spammers* e vírus.

Várias técnicas vêm sendo usadas para controle de *spam*. As principais são baseadas em listas de servidores confiáveis e não-confiáveis, ou na filtragem das mensagens recebidas com base em seu conteúdo:

- **Listas Negras (*Black Lists*)**: Servidores *RBL* (*Realtime Blackhole Lists*) distribuídos na Internet mantêm listas de endereços IP de geradores ou propagadores de *spam*, que podem ser consultadas por meio de DNS pelos servidores de e-mail para verificar a confiabilidade de um remetente [Jung and Sit, 2004]; essas listas são mantidas por meio de vários mecanismos, como contribuições de usuários, resultados de varreduras automáticas, etc.
- **Listas Brancas (*White Lists*):** cada servidor de e-mail pode manter uma lista de remetentes nos quais confia; essa lista é normalmente mantida por meio de pedidos de confirmação de envio que remetem a um formulário web ou outra abordagem similar [Hall, 1998]. Alguns sistemas alimentam suas listas brancas com outras técnicas, como forçar o remetente a tentar o mesmo envio várias vezes [Fromberger, 2004].
- **Filtros antispam**: programas de filtragem que classificam os e-mails de acordo com seu conteúdo; podem ser usadas técnicas estatísticas, de classificação *bayesiana*, redes neurais, etc [Sahami et al., 1998]. Muitos *spammers* tentam burlar tais filtros por meio de técnicas que permitem inserir textos em imagens, usar estruturas HTML para confundir os filtros, entre outras.

2.2. Vírus e *worms*

Os vírus e os *worms* de e-mail são programas que têm como objetivo inutilizar sistemas, destruir arquivos, roubar informações importantes de uma máquina ou simplesmente propagar-se ao extremo. Um vírus de e-mail normalmente é constituído por um e-mail com um anexo executável (ou um código HTML que permita carregar um arquivo executável armazenado remotamente). Esse código executável pode ser ativado pelo usuário ou até mesmo de forma automática, explorando vulnerabilidades no cliente de e-mail.

A propagação de um vírus acontece por meio da exploração de fragilidades nos clientes de e-mail e do uso da lista de contatos do usuário [Klensin, 2001]. Além disso, máquinas infectadas por vírus podem oferecer *backdoors* que permitem seu uso em diversos tipos de atividades nocivas.

Empresas de software desenvolvem antivírus e mantêm bases de assinaturas de vírus já conhecidos. Porém, o atraso na atualização dessas bases pode ocasionar infestações e propagação de vírus pela rede. Usuários de computadores pessoais precisam atualizar constantemente seus sistemas de antivírus. Com o intuito de minimizar este problema em serviços de e-mail, provedores de acesso estão implantando antivírus na entrada das mensagens, diminuindo a possibilidade de transporte de vírus anexados em mensagens. Todavia, estudos recentes demonstram que a abordagem baseada em bases de assinaturas nunca resolverá o problema, pois os novos vírus se propagam mais rapidamente que as atualizações dos antivírus [Zou et al., 2002].

2.3. Scam

Os *scams* são mensagens falsificadas que utilizam fragilidades do protocolo SMTP para construir ataques de engenharia social, visando enganar os destinatários, convencendo-os

a informar dados bancários, números de cartões de créditos e outras informações confidenciais. A solução para este problema é utilizar um sistema de assinatura/certificado digital ou então tecnologias que validem a autenticidade do servidor ou do remetente da mensagem.

3. Servidores de e-mail distribuídos

Diversas alternativas vem sendo propostas na literatura recente para prover escalabilidade, tolerância a faltas e disponibilidade em serviços de e-mail. Normalmente essas abordagens fazem uso da distribuição do serviço sobre um grupo de servidores em uma rede local. Nesta seção, as principais abordagens nesse sentido serão apresentadas: *Earthlink*, *Porcupine*, *Ninjamail* e *Gmail*.

O *Earthlink* [Christenson et al., 1997] utiliza o sistema de arquivos NFS (*Network File System*) para distribuir as caixas de correio em uma rede local. A entrada das mensagens no sistema é realizada por uma versão genérica do MTA (*Mail Transport Agent*) centralizado *Sendmail*. O balanceamento da carga de entrada é feito por meio de *Round-Robin DNS*. Ao completar a recepção de uma mensagem, o *Sendmail* invoca o MDA (*Mail Delivery Agent*) *mail.local* para entregar a mensagem recebida; o MDA foi modificado para suportar o sistema de armazenamento usado pelo sistema *Earthlink*. O *Earthlink* dispersa as caixas de correio dos usuários entre os diversos servidores utilizando um algoritmo simples que calcula a quantidade de caixas por servidor de acordo com o número de servidores.

O *Porcupine* [Saito et al., 1998, Saito et al., 1999] distribui o serviço de e-mail em um grupo de servidores denominado *mail cluster*. Uma tabela de *hash* distribuída é usada para distribuir e replicar os e-mails entre os diversos nós do *cluster*, buscando assegurar a disponibilidade das mensagens. Quando um MUA (*Mail User Agent*) faz uma requisição de e-mail, o servidor POP/IMAP consulta o serviço de hashing para localizar os nós que contêm a caixa do usuário e em seguida inicia uma comunicação RPC com cada um dos nós para obter as mensagens do usuário.

O *Ninjamail* [Behren et al., 2000] foi desenvolvido utilizando a arquitetura distribuída *Ninja* [Gribble et al., 2001] e o sistema de armazenamento e comunicação *OceanStore* [Kubiatowicz et al., 2000]. O *cluster* Ninja provê um ambiente distribuído de alto desempenho que inicia e finaliza trabalhos em diferentes nós automaticamente e cria tarefas assíncronas para comunicação entre os trabalhadores. A arquitetura *Ninja-Mail* é composta por diversos módulos; o módulo *MailStore* armazena, atualiza, localiza, retorna e remove as mensagens de um usuário no sistema. Os módulos de acesso são os componentes de comunicação de entrada/saída do servidor (POP, IMAP, HTTP, SMTP). Os módulos de extensão são desenvolvidos pelo usuário e facilitam a inclusão de novas funcionalidades no ambiente, como filtros de mensagens, localização de palavras no corpo das mensagens, etc. O sistema *Ninjamail* utiliza o ambiente *OceanStore* para armazenar as mensagens em um contexto distribuído.

O *Google Mail* (*Gmail*) [Google, 2004] é um ambiente de *webmail* construído sobre o *GFS – Google File System* [Ghemawat et al., 2003] para criar um ambiente distribuído de armazenamento escalável, confiável e disponível. O *Gmail* é basicamente um mecanismo de busca do sistema Google customizado para localizar mensagens de um usuário no GFS. O *cluster GFS* é dividido da seguinte forma: um nó *master*, diversos nós *chunkservers* (servidores de fragmentos) e clientes de acesso. Um arquivo é dividido pelo nó *master* em vários pedaços identificados por uma chave única de 64 bits que são armazenados pelos *chunkservers*. Para maior confiabilidade, esses pedaços são distribuídos e armazenados em múltiplos servidores (por default em três servidores dis-

tintos). O nó *master* é responsável por armazenar todos os dados pertinentes ao sistema (informações sobre controle de acesso, localização atual dos fragmentos, etc) e efetuar a gerência do serviço (informações sobre o coletor de lixo, migração de pedaços entre servidores, recuperação dos estados dos *chunkservers*, etc). Os clientes são os responsáveis por ler e gravar informações no sistema de arquivos. Eles interagem com o nó *master* para verificar informações sobre usuários e fragmentos; em seguida, se comunicam diretamente com os *chunkservers* responsáveis pelos fragmentos.

4. Fragmentação e disperção com redundância

O FDR (*Fragmentação e Disperção com Redundância*) é uma técnica que fragmenta um arquivo em pedaços de tamanho fixo e dissemina os fragmentos em diversos nós de armazenamento de dados. Esta técnica disponibiliza um ambiente de armazenamento seguro e tolerante a faltas em ambientes distribuídos [Fabre et al., 1994].

O método de fragmentação consiste em transformar o dado a ser armazenado em M páginas de tamanho fixo que serão cifradas, assinadas em seguida fragmentadas em um número N de fragmentos. Depois que todos os fragmentos tiverem sido produzidos, o sistema deverá enviar de forma aleatória cada um dos fragmentos aos nós de armazenamento. A política de disseminação deverá garantir que cada fragmento seja armazenado em mais de um nó, tornando o sistema tolerante a falta.

A técnica FDR complementa a segurança através de métodos de cifragem, tornando o armazenamento menos vulnerável a técnicas de cripto análise: Um invasor atacando um único nó de armazenamento não tem acesso a todos os fragmentos. Mesmo que ele consiga ter acesso a todos os N fragmentos, ainda assim deverão ser necessárias N!/2 análises criptograficas para reconstruir uma única página.

5. A arquitetura ACME

O objetivo do projeto ACME (*um Ambiente para a Classificação e Manipulação de E-mails*) é construir um ambiente distribuído de processamento de e-mails, com mecanismos robustos de manipulação e classificação de mensagens. A arquitetura do sistema foi concebida com uma forte preocupação nos aspectos de desempenho, flexibilidade, escalabilidade e tolerância a faltas e intrusões. Para tal, diversas técnicas foram empregadas, como a estruturação do sistema em células autônomas, fragmentação e dispersão do armazenamento, cifragem das caixas de correio e processamento de mensagens baseado em eventos.

O ACME proporciona mecanismos de processamento de mensagens como filtragem de *spam* e vírus, adição/remoção/modificação de anexos, modificação de cabeçalhos das mensagens, detecção de conteúdo indevido, triagem de mensagens, contabilização de uso e estatísticas detalhadas. Esses mecanismos são parametrizáveis e podem ser modificados através de uma linguagem de *scripting* simples.

O sistema ACME foi estruturado em duas camadas: a *camada de recepção de mensagens*, que implementa o protocolo SMTP e demais funcionalidades de um servidor de e-mail e a *camada de manipulação e classificação de mensagens*, que utiliza o poder da linguagem interpretada Ruby [Fulton and Hurst, 2003]. A linguagem Ruby foi escolhida por ser simples, orientada a objetos e por possuir uma biblioteca padrão extensa. A escolha de uma linguagem orientada a objetos facilita a classificação e manipulação das mensagens que serão transformadas pela camada de recepção e armazenadas no repositório como objetos.

5.1. Gerenciador de eventos

O gerenciador de eventos ACME foi proposto para disponibilizar uma forma simples de incluir métodos de classificação, checagem e manipulação de informações trocadas entre servidores no envio/recebimento das mensagens. O gerenciador de eventos integra a linguagem de programação interpretada Ruby, que oferece mecanismos de execução de código sem a necessidade de recompilação do código. Para facilitar a manipulação das mensagens, cada mensagem recebida pelo servidor de e-mail deverá ser transformada em objeto Ruby (DMail). A seguir será detalhado o funcionamento do gerenciador de eventos.

O gerenciador de eventos ACME aguarda por eventos que serão gerados no decorrer de uma comunicação SMTP. Ao ocorrer um evento, o gerenciador de eventos faz uma chamada a um "gancho" associado ao evento gerado. Os ganchos são métodos codificados em Ruby por administradores de serviços de e-mail para criar um ambiente de classificação e manipulação de cada comando/mensagem trocada a partir do protocolo. O gancho chamado deverá executar a sua funcionalidade e em seguida retornar uma resposta SMTP para que a comunicação prossiga. Quando uma mensagem for recebida por completa o ACME transformará a mensagem em um objeto DMail que poderá ser manipulado de acordo com as regras do domínio.

Neste trabalho foi definido os seguintes ganchos: *smtp_connect*, *smtp_timeout*, *smtp_helo*, *smtp_ehlo*, *smtp_mail*, *smtp_rcpt*, *smtp_data*, *smtp_rset*, *smtp_vrfy*, *smtp_expn*, *smtp_help*, *smtp_noop* e *smtp_quit*, que correspondem a cada um dos comandos do protocolo. Esses eventos podem ou não ser implementados. Caso um gancho não seja implementado uma resposta padrão do protocolo SMTP deverá ser gerada. A seguir é apresentado um exemplo de código em Ruby para o gancho *rcpt_to*:

```
class SmtpdHook

  def initialize
  end

  def smtp_connect(addr)
    @log.info "New connection coming from #{addr}"
    return 220, "#{ENV['HOSTNAME']} Service ready"
  end

  def smtp_helo(arg)
    return 250, "Requested mail action okay, completed"
  end

  # RCPT TO:
  def smtp_rcpt(local_part, domain, source_route)
    if domain == "foo.bar" then
      begin
        if local_part == "postmaster" then
          local_part = "admin"
        else
          user = Etc::getpwnam(local_part)
        end
        return 250, "Requested mail action okay, completed"
      rescue ArgumentError
        return 450, "Recipient address rejected: Unknown local part"
      end
    else
      return 450, "Recipient address rejected: Domain not found"
    end
  end
```

```
  # ...
end
```

Neste exemplo, a mensagem deverá ser aceita se e somente se o domínio do destinatário for *foo.bar* e o usuário existir localmente. O servidor de mensagens atualiza sua tabela de estado se o código de retorno do método for positivo. É verificado também se o destinatário da mensagem é o *postmaster*; em caso afirmativo, a parte local é reescrita de tal forma que a mensagem seja redirecionada para o usuário *admin*, do contrário, a rotina verifica se existe um usuário local na máquina com o mesmo nome e aceita (usuário encontrado) ou rejeita (usuário inexistente) a mensagem.

Todos os ganchos (exceto o gancho *smtp_data*) possuem a mesma funcionalidade. Eles recebem informações vindas da comunicação, processam as informações e retornam um código de retorno SMTP. Já *smtp_data* recebe como parâmetro a mensagem na forma de um objeto Ruby, executa a manipulação da mensagem de acordo com o código definido e em seguida retorna a mensagem manipulada ao gerenciador de eventos para que esta seja armazenada na caixa de entrada ou então excluída (dependendo do código do *smtp_data*).

5.2. Células ACME

As funcionalidades do servidor de e-mail foram divididas em células. Uma *célula ACME* é a representação de um nó na arquitetura do sistema, sendo composta por módulos independentes que executam tarefas distintas. Essa abordagem simplifica a manutenção do sistema pois os módulos podem ser facilmente inseridos, removidos ou substituídos. Cada célula ACME é composta pelos seguintes módulos: *SMTP Manager*, *Route Manager*, *Delivery Manager*, *POP Manager* e *Repository*.

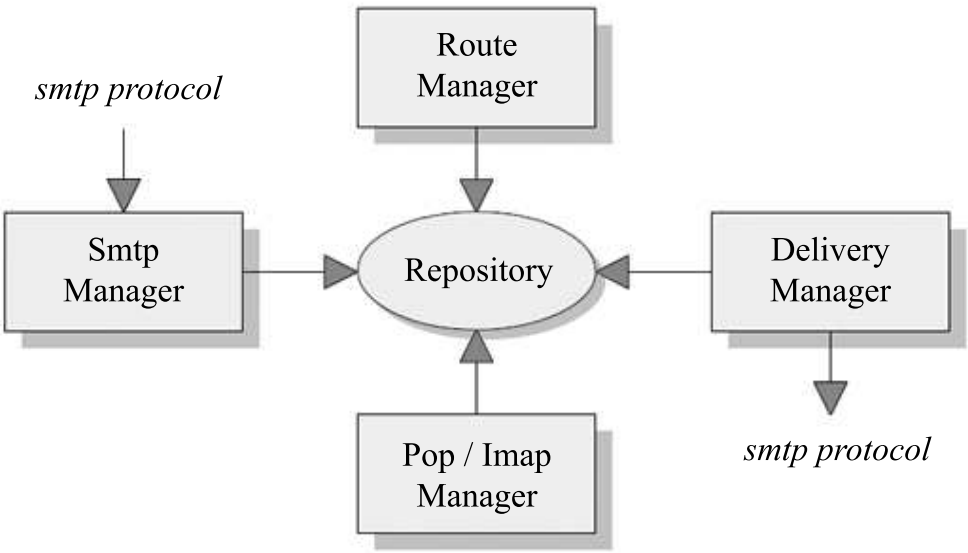

Figura 1: Célula ACME

A figura 1 apresenta o modelo de uma célula completa, composta pelos seguintes elementos:

- O *SMTP Manager* é o serviço de entrada; ele recebe as mensagens utilizando o protocolo SMTP e as armazena no repositório do sistema.

- O *Route Manager* é o serviço de roteamento de e-mails; ele requisita ao repositório as mensagens que tenham sido recebidas pelo sistema mas que ainda não foram processadas, valida suas informações, define a rota de cada mensagem (entrega local, entrega remota, expansão de aliases, etc) e as devolve ao repositório.
- O *Delivery Manager* é o serviço responsável pelas entregas remotas do sistema; ele recupera as mensagens que devem ser entregues remotamente e se comunica com outros sistemas de e-mail. Se uma mensagem não puder ser entregue, ele utiliza mecanismos de notificação e/ou ganchos do sistema para que seja tomada uma decisão sobre o destino da mensagem (ver seção 5.1).
- O *POP/IMAP Manager* é o serviço responsável pela entrega de mensagens aos clientes; ele recupera as mensagens armazenadas na caixa de correio de um usuário e as entrega utilizando o protocolo POP3 ou IMAP.
- O *Repository* é o serviço responsável por armazenar as mensagens do servidor de e-mail e criar um ambiente transparente de comunicação entre os demais módulos do sistema.

Em um único servidor físico podem coexistir diversas células; cada célula deverá possuir no mínimo um componente ativo. Além disso, a inserção e remoção de células não afetam o funcionamento geral do sistema pois cada módulo utiliza um mecanismo de *multicast* para descobrir os repositórios que fazem parte do sistema e interagir com eles.

5.3. Repositório e *tags*

O módulo central do sistema ACME é o *Repository*. Ele implementa um repositório distribuído que armazena e controla o acesso às mensagens, também aplicando políticas de armazenamento no sistema. Esta abordagem torna o isolamento e divisão de tarefas entre os demais módulos transparente, pois eles enxergam apenas uma interface de acesso ao repositório distribuído. A Figura 2 apresenta o repositório distribuído ACME.

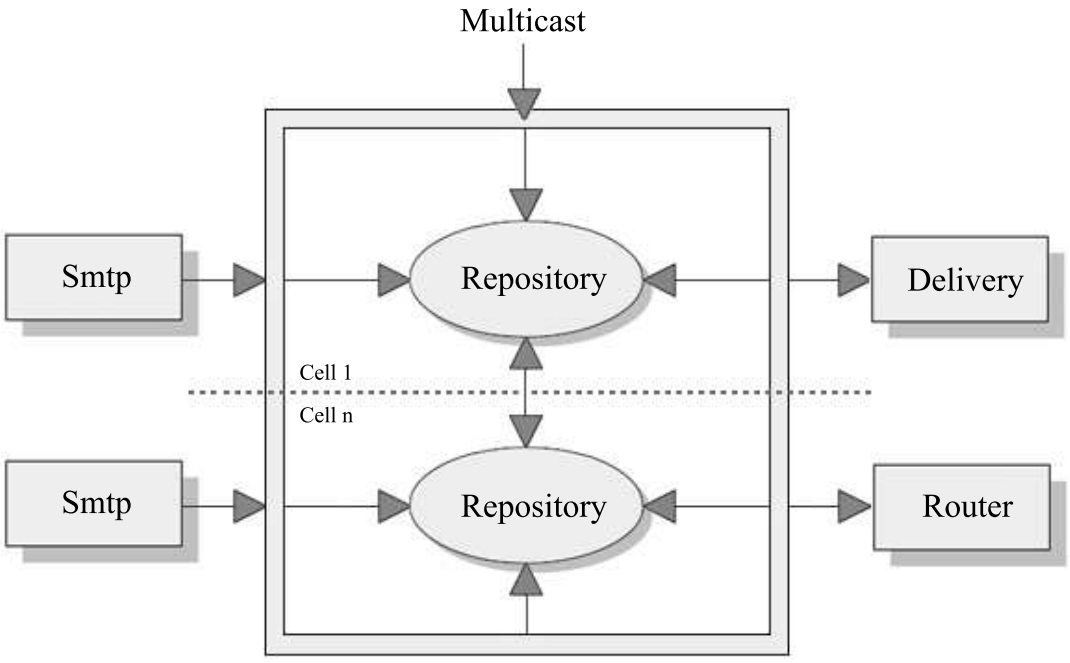

Figura 2: Grupos de Repositório

Para obter segurança e tolerância a faltas no armazenamento das mensagens foi utilizada a técnica FDR [Fabre et al., 1994], onde cada mensagem recebida pelo servidor

deve ser fragmentada e distribuída de forma redundante pelas várias células do sistema. A fragmentação garante que não seja possível obter informações de um fragmento e a dispersão deve assegurar que exista um único subconjunto de fragmentos em cada nó do servidor e que mais de um nó possua uma cópia deste subconjunto.

O repositório armazena *tags* e objetos. *Tags* são tuplas chave/valor, enquanto os objetos são quaisquer tipos de dados identificados exclusivamente por UUIDs (*Universal Unique IDentifiers*). Não pode haver colisão entre os UUIDs dos objetos armazenados, pois esses identificadores serão as chaves para acessar os objetos depositados; se o repositório detectar uma colisão no armazenamento de um novo objeto, a operação será rejeitada. Cada fragmento gerado por FDR carrega consigo o UUID do objeto original e seu novo UUID. O novo identificador será utilizado para verificar a integridade dos fragmentos distribuídos, e o UUID do objeto original será utilizado para referenciar a mensagem dentro do sistema.

Chave:	**Valor:**
`usuario@dominio.com`	`7f4abb47-d7c2-492e-9025-56ff2f20cdf5`

As *tags* podem ser utilizadas de forma genérica para armazenar qualquer tipo de valor no repositório. Caso haja colisão entre chaves, ambas os valores associados serão acessados seqüencialmente:

Chave:	**Valor:**
`usuario@dominio.com`	`7f4abb47-d7c2-492e-9025-56ff2f20cdf5`
`usuario@dominio.com`	`a18a3c09-3849-423a-9275-876fa2bf41`

Chave:	**Valor:**
`senha$usuario@dominio.com`	`$1$SFpSblp$SQkMdekdA1ulNEU1mt1`

Chave:	**Valor:**
`blacklist$usuario@dominio.com`	`dark@moor.com`
`blacklist$usuario@dominio.com`	`halo@ween.net`

Dessa forma o repositório pode ser utilizado para armazenar qualquer informação, Como por exemplo listas de controle de acesso ou autenticação de usuários. O processamento pode ser exemplificado pela classe de ganchos do *POP Manager*:

```
class PopHook

  def initialize
  end

  def pop_user(user)
    @local = user
  end

  def pop_password(password)
    if tag_retrieve( "senha$" + @local, hash ) == true then
      if Digest::MD5.hexdigest(password) == hash then
        return true
      else
        return false
      end
```

```
    else
      return false
    end
  end

  # ...

end
```

As *tags* também são utilizadas para evitar a duplicação de mensagens que possuam diversos destinatários. Em sistemas convencionais a mensagem é duplicada para cada usuário, elevando a quantidade de espaço em disco necessária para armazenar as diversas cópias da mesma mensagem. Com o auxílio das *tags*, apenas a mensagem original é armazenada no repositório e uma *tag* de referência é inserida para cada destinatário da mensagem. Por exemplo:

Chave:	**Valor:**
`7f4abb47-d7c2-492e-9025-56ff2f20cdf5`	`mega@metal.com`
`7f4abb47-d7c2-492e-9025-56ff2f20cdf5`	`master@plan.org`
`7f4abb47-d7c2-492e-9025-56ff2f20cdf5`	`cordel@fogo.com`

Chave:	**Valor:**
`mega@metal.com`	`7f4abb47-d7c2-492e-9025-56ff2f20cdf5`
`mega@metal.com`	`6d24f44c-171b-4e81-b6d1-76abff73dbd`
`master@plan.org`	`7f4abb47-d7c2-492e-9025-56ff2f20cdf5`

Toda vez que um usuário remover uma mensagem, apenas a *tag* de referência será removida. Quando mais nenhuma *tag* referenciar o objeto, o mesmo removido do repositório. Esse processo funciona como uma coleta de lixo; dentro de cada componente do repositório existem monitores programados para realizar coletas de lixo periódicas, removendo mensagens que não foram explicitamente removidas (objetos não referenciados), auxiliando e diminuindo a carga dos componentes de acesso à mensagem (IMAP/POP).

5.4. Aspectos de implementação

A arquitetura de implementação predominante entre os componentes do sistema é a AMPED (*Asymmetric Multi-Process Event-Driven*) definida em [Pai et al., 1999]. Essa arquitetura, ilustrada na Figura 3, combina a programação baseada em eventos com a utilização de processos auxiliares para realizar tarefas que possam bloquear o laço principal do servidor. O processo principal baseado em eventos realiza todas as operações não-bloqueantes. Quando uma operação que possa bloquear o laço principal é necessária, um sinal é enviado, através de um mecanismo de notificação de eventos, para um processo auxiliar que irá realizar a operação. Assim que a operação for finalizada, o processo auxiliar envia um sinal de volta para o processo requisitante, notificando-o de que a operação solicitada foi completada.

O mecanismo de notificação de eventos utilizado pelo sistema é o POSIX-RT (*Real-Time Signals*). Os sinais RT provêem um mecanismo alternativo para receber sinais, as chamadas de sistema *sigwaitinfo* e *sigtimedwait* habilitam os processos a verificar suas filas de sinais e recebê-los sem a necessidade de serem interrompidos de forma preemptiva. Ao contrário dos tradicionais sinais UNIX, os sinais RT podem carregar dados, tais como estruturas que descrevem a que se destina cada sinal. Um benefício adicional

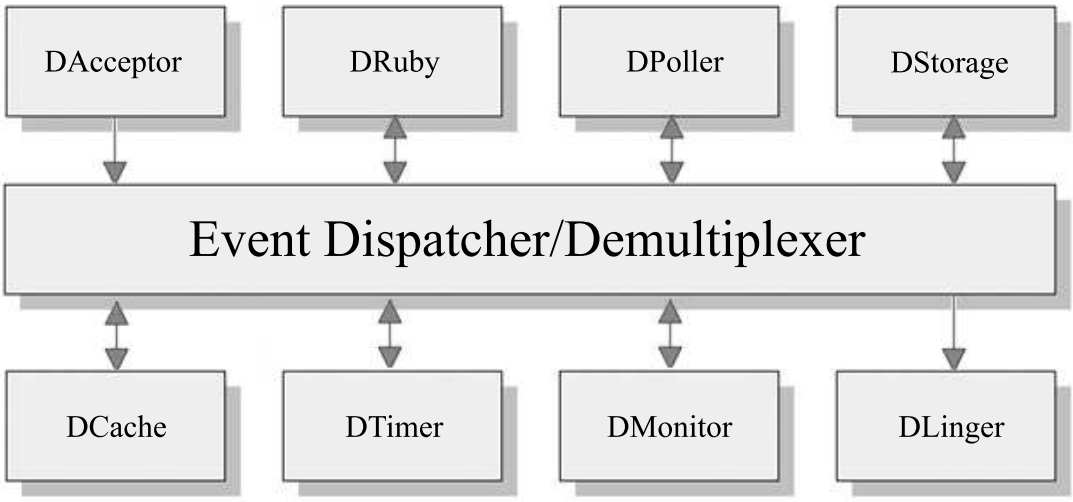

Figura 3: Asymmetric Multi-Process Event-Driven

dos sinais RT é que podem ser enfileirados e entregues um de cada vez, ordenadamente, deixando a aplicação livre para consultar os sinais quando lhe for mais conveniente.

As tarefas auxiliares são implementadas como *threads* que dividem o mesmo espaço de memória. Essa característica, somada ao uso dos sinais RT, permite que as *threads* troquem referências de estruturas compartilhadas sem a necessidade de sincronização explicita, passando referências de uma tarefa para outra por meio dos sinais. A cada grupo de tarefas que realizam uma operação específica é atribuído um número de sinal que deve ser recebido pelas mesmas. As tarefas ficam à espera do sinal especificado para seu grupo. Quando a tarefa principal envia um sinal, o *kernel* encaminha esse sinal para a primeira tarefa que está aguardando por sinais desse tipo; se não houver nenhuma tarefa em espera, o sinal é enfileirado para ser entregue quando alguma tarefa estiver disponível.

6. Trabalhos correlatos

Conforme visto na seção 3, o sistema *Earthlink* [Christenson et al., 1997] possui uma arquitetura simples que utiliza um sistema monolítico para receber as mensagens (*Send-mail*) e tecnologias já consolidadas para prover armazenamento nas caixas de correio do usuário. Ele utiliza um servidor NFS para "enxergar" seu sistema de armazenamento distribuído como um único sistema de arquivos.

Por sua vez, a arquitetura do sistema *Porcupine* [Saito et al., 1998, Saito et al., 1999] possui seu próprio sistema de envio e recebimento de mensagens e utiliza uma tabela hash em cada nó do servidor utilizada para localizar e armazenar mensagens no sistema. As informações sobre os usuários são armazenadas em um servidor centralizado NIS, que constitui um ponto único de falha. Além disso, não são considerados aspectos de cifragem das mensagens e otimização no armazenamento de e-mails com vários destinatários.

O *Ninjamail* [Behren et al., 2000] é o servidor de e-mail que mais se assemelha com a arquitetura aqui descrita. Através do sistema de armazenamento *OceanStore* [Kubiatowicz et al., 2000], ele disponibiliza um sistema de armazenamento global e auto-configurável de mensagens; os módulos de extensão proporcionam um ambiente flexível, onde novas funcionalidades podem ser inseridas sem a necessidade de alterações no

código original.

O *Gmail* [Google, 2004] utiliza o sistema de armazenamento *Google File System* [Ghemawat et al., 2003] para disponibilizar um ambiente de armazenamento distribuído capaz de ofertar um grande espaço de armazenamento para seus usuários. Não são contempladas características de flexibilidade de configuração.

Diferente das propostas apresentadas, o sistema ACME armazena as mensagens em um repositório distribuído que usa a técnica FDR para cifrar, fragmentar e dispersar as mensagens de forma protegida e redundante entre os diversos nós do servidor. Foi adicionado ao repositório o conceito de *tag*, onde vários usuários podem apontar para uma mesma mensagem, diminuindo a necessidade de armazenamento de mensagens replicadas. Além disso, o sistema ACME utiliza uma linguagem de configuração orientada a objetos que facilita a administração das informações que trafegam pelo sistema e simplifica a incorporação de novos mecanismos.

7. Conclusão e trabalhos futuros

Neste artigo foi apresentada a arquitetura de um servidor de e-mail distribuído, flexível e modular que utiliza uma linguagem de configuração integrada ao sistema de envio/recepção e armazenamento das mensagens. A arquitetura modular baseada em células simplifica a manutenção do sistema pois um ou mais módulos (que formam uma célula) podem ser facilmente ativados/desligados/substituídos sem a necessidade de parar o sistema.

O módulo de ganchos facilita a administração das informações que trafegam pelo sistema. O ACME pode encerrar uma conexão, criar filtros de mensagens, integrar-se com outros programas externos, alterar informações do sistema, classificar e manipular mensagens, entre outras, através de ganchos implementados em uma linguagem de programação interpretada orientada a objetos. Finalmente, o repositório com FDR cifra, fragmenta e dispersa as mensagens de forma redundante entre os diversos nós do servidor trazendo uma maior segurança e disponibilidade aos seus usuários.

O servidor de e-mail ACME está na fase de finalização da implementação. Os módulos *SmtpManager*, *PopManager* e *DeliveryManager* já estão operacionais. Os demais componentes estão em fase de implementação; estima-se que até outubro de 2005 todo o sistema esteja funcionando. Em paralelo ao desenvolvimento do sistema, estão sendo desenvolvidos ainda um sistema de gerência de confiança entre servidores de e-mail, que visa o combate ao spam e scam, e também um sistema de gerência de listas de confiança, baseado na técnica de *greylisting* descrita em [Harris, 2004].

Referências

Behren, J., Czerwinski, S., Joseph, A., Brewer, E., and Kubiatowicz, J. (2000). NinjaMail: The design of a high-performance clustered, distributed e-mail system. In *ICPP International Conference on Parallel Processing*.

Christenson, N., Bosserman, T., and Beckemeyer, D. (1997). A highly scalable electronic mail service using open systems. In *USENIX Symposium on Internet Technologies and Systems*, pages 1–15.

Crocker, D. (1982). Standard for the format of ARPA Internet text messages. IETF RFC 822.

Fabre, J.-C., Deswarte, Y., and Randell, B. (1994). Designing secure and reliable applications using fragmentation-redundancy-scattering: An object-oriented approach. In *European Dependable Computing Conference*, pages 21–38. Springer-Verlag.

Fromberger, M. (2004). Bayesian classification of unsollicited e-mail. http://thayer.dartmouth.edu/ sting/sw/bayes-spam.pdf.

Fulton, H. and Hurst, G. (2003). *The Ruby Way*. Sams.

Ghemawat, S., Gobioff, H., and Leung, S. (2003). The Google file system. In *ACM Symposium on Operating Systems Principles*, pages 29–43.

Google (2004). Google Mail. http://gmail.google.com.

Gribble, S., Welsh, M., von Behren, R., Brewer, E., Culler, D., Borisov, N., Czerwinski, S., Gummadi, R., Hill, J., Joseph, A., Katz, R., Mao, M., Ross, S., and Zhao, B. (2001). The Ninja architecture for robust internet-scale systems and services. *Computer Networks*, 35(4):473–497.

Hall, R. J. (1998). How to avoid unwanted email. *Communications of the ACM*, 41(3):88–95.

Harris, E. (2004). The next step in the spam control war: Greylisting. http://projects.puremagic.com/greylisting/whitepaper.html.

IDC (2003). Third annual email usage forecast and analysis, 2001-2005. IDC Report W25335.

IDC (2004). The true cost of spam and the value of antispam solutions. IDC Report.

Jung, J. and Sit, E. (2004). An Empirical Study of Spam Traffic and the Use of DNS Black Lists. In *Internet Measurement Conference*, Taormina, Italy.

Klensin, J. (2001). Simple Mail Transfer Protocol. IETF RFC 2821.

Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. (2000). OceanStore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems*.

Lindberg, G. (1999). Anti-spam recommendations for SMTP MTAs. IETF RFC 2505.

Pai, V. S., Druschel, P., and Zwaenepoel, W. (1999). Flash: An efficient and portable Web server. In *Proceedings of the USENIX 1999 Annual Technical Conference*.

Resnick, P. (2001). Internet message format. IETF RFC 2822.

Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. (1998). A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin. AAAI Technical Report WS-98-05.

Saito, Y., Bershad, B., and Levy, H. (1999). Manageability, availability and performance in Porcupine: A highly scalable, cluster-based mail service. In *Symposium on Operating Systems Principles*.

Saito, Y., Hoffman, E., Bershad, B., Levy, H., Becker, D., and Folliot, B. (1998). The Porcupine scalable mail server. In *8th ACM SIGOPS European workshop on Support for Composing Distributed Applications*, pages 48–52.

Zou, C. C., Gong, W., and Towsley, D. (2002). Code Red worm propagation modeling and analysis. In *9th ACM conference on Computer and Communications Security*, pages 138–147.