Um Serviço Baseado em SNMP para Detecção de Falhas de Processos Distribuídos em Múltiplos Sistemas Autônomos

Dionei M. Moraes, Elias P. Duarte Jr.

Universidade Federal do Paraná (UFPR), Depto. Informática
Caixa Postal 19018 Curitiba PR 81531-980

{dioneimm,elias}@inf.ufpr.br

Abstract. *This work presents a failure detector service for Internet-based distributed systems that span multiple autonomous systems. An SNMP agent, called monitor, implements a MIB used as the interface to obtain global process state information. Monitors at different LANs communicate across the Internet using Web Services. Processes are monitored with heartbeats; if a* working *process remains silent for a timeout interval adaptively computed, the state is toggled to* suspect. *The process state is toggled to* crashed *only after this information is confirmed at the local operating system. The system was implemented and evaluated for monitored processes running both at a LAN and distributed throughout the world in PlanetLab. Experimental results are presented, showing CPU usage, failure detection latency, and mistake rate.*

Resumo. *Este trabalho apresenta um serviço de detecção de falhas para sistemas distribuídos executados em múltiplos sistemas autônomos da Internet. Um agente SNMP, dito monitor, implementa uma MIB que é usada como interface para a obtenção de informações sobre o estado global dos processos. Monitores em diferentes redes locais se comunicam através de Serviços Web. O monitoramento dos processos é feito através de* heartbeats. *Se um processo* sem-falha *fica silencioso por um intervalo de* timeout *dinâmico e adaptativo, passa a ser considerado suspeito. Um processo é classificado como falho exclusivamente após uma confirmação no sistema operacional local. O sistema foi implementado e avaliado com processos monitorados executando tanto em redes locais quanto no PlanetLab. São avaliados o consumo de CPU, o tempo de detecção de falhas e a taxa de engano.*

1. Introdução

A impossibilidade de executar o consenso [Fischer, Lynch and Paterson 1985] em sistemas distribuídos assíncronos [Greve 2005, Raynal 2004] sujeitos a falhas representa um desafio concreto para a execução de aplicações de alta disponibilidade na Internet. Dentro deste contexto, Chandra e Toueg introduziram o conceito de detectores de falhas [Chandra and Toueg 1996]: oráculos distribuídos que fornecem informações a respeito do estado de execução de processos de um sistema distribuído. Entre as características dos detectores destaca-se o fato de que podem cometer enganos, por isso são ditos *não*-confiáveis. Chandra e Toueg mostraram que se processos de um sistema assíncrono têm acesso a um detector de falhas, dependendo das propriedades desse detector, o consenso se torna possível.

O presente trabalho apresenta uma implementação de um serviço de detecção de falhas para sistemas distribuídos executados sobre múltiplos sistemas autônomos da Internet. A implementação é baseada em SNMP (*Simple Network Management Protocol*) e serviços Web. Aplicações distribuídas que necessitam de informações sobre o estado de seus processos antes de tomar decisões podem consultar o detector de falhas através de uma interface SNMP. O serviço tem um agente, dito monitor, executando em cada rede local onde houverem processos a serem monitorados. Esta monitoração é realizada através de mensagens enviadas periodicamente pelo processo monitorado, chamadas *heartbeats*. Além disso, são obtidas informações sobre os processos no próprio sistema operacional. O detector identifica se um determinado processo está *falho*, *suspeito* ou *sem-falha*. Um processo é considerado suspeito se não for recebido um *heartbeat* dentro do intervalo de *timeout* adaptativo. O sistema operacional é utilizado para confirmar processos falhos. Se os *heartbeats* são recebidos no intervalo estipulado, e se não são identificadas falhas através do sistema operacional, então o processo é considerado sem-falha.

O monitor é um agente SNMP que mantém uma MIB (*Management Information Base*) contendo informações de estado referentes aos processos locais e remotos. Um processo é identificável pelo seu endereço IP, porta, além do seu identificador no sistema operacional local (PID). São também mantidas estatísticas sobre o comportamento temporal dos processos monitorados, como o *timestamp* local referente ao último *heartbeat* enviado pelo processo, média do tempo de envio de *heartbeats*, e desvio médio. Os monitores de diferentes redes locais comunicam estas informações entre si através de operações SNMP utilizando serviços Web. Um processo de aplicação pode verificar o estado de outros processos consultando seu monitor local. O sistema foi implementado e avaliado com processos monitorados executando tanto em redes locais quanto no PlanetLab [PlanetLab]. São avaliados o consumo de CPU, tempo de detecção de falhas e taxa de engano.

Wiesmann, Urbán e Défago apresentam em [Wiesmann, Urbán and Défago 2006] o *framework* SNMP-FD - um serviço de detecção de falhas que assume todos os processos executando em uma única rede local. Apesar de ambos os trabalhos apresentarem implementações de detectores de falhas usando SNMP, são diversas as diferenças da arquitetura proposta neste trabalho para o SNMP-FD, como descrito a seguir. A principal diferença do presente trabalho é permitir a detecção de falhas de processos executados em múltiplos sistemas autônomos na Internet - em contraposição a falhas de processos executando em uma única rede local. Além disso, na arquitetura proposta, o SNMP é utilizado apenas como interface para que uma aplicação obtenha informações do detector. Ao contrário, o SNMP-FD utiliza o SNMP para todos os componentes da implementação bem como toda a comunicação. Ainda outra diferença é que o SNMP-FD utiliza um monitor executando em cada host, enquanto o sistema proposto utiliza apenas um monitor por rede local, independente do número de hosts que a compõem. O *timeout* utilizado para detectar suspeitas de processos não é fixo (como no SNMP-FD); a estratégia utilizada para este cálculo é uma estratégia dinâmica e adaptativa baseada no algoritmo utilizado pelo TCP.

O restante deste trabalho está organizado da seguinte maneira. A seção 2 descreve os detectores de falhas e o framework SNMP-FD. A arquitetura do serviço de detecção de falhas proposto é descrita na seção 3. A implementação bem como resultados experimentais obtidos em rede local e no PlanetLab são descritos na seção 4. A conclusão segue na

seção 5.

2. Detectores de Falhas

O problema central no desenvolvimento de sistemas distribuídos tolerantes a falhas é o consenso [Turek and Shasha 1992, Greve 2005]. No consenso, processos precisam decidir sobre um mesmo valor, que depende de uma entrada inicial, mesmo na presença de falhas [Guerraoui and Rodrigues 2006]. Algoritmos que resolvem o problema do consenso podem ser utilizados para resolver vários outros problemas, como por exemplo, eleição de líder e agrupamento *(group membership)*. Em [Fischer, Lynch and Paterson 1985], Fisher, Lynch e Paterson provaram que o consenso não pode ser resolvido deterministicamente em sistemas assíncronos sujeitos a sequer uma falha. O modelo de falhas considerado neste trabalho é o da parada total, ou *crash*. Este resultado é conhecido como a impossibilidade FLP, das iniciais de seus autores. Esta impossibilidade se dá devido à dificuldade de se determinar se um processo está falho ou apenas lento.

Os detectores de falhas não-confiáveis foram propostos por Chandra e Toueg em [Chandra and Toueg 1996], que mostraram que, dependendo das propriedades que apresentam, detectores de falhas podem ser usados para resolver o problema do consenso em sistemas assíncronos sujeitos a falhas [Chandra and Toueg 1996]. Um detector de falhas é um *oráculo* distribuído que fornece informações a respeito do estado de processos em um sistema distribuído [Wiesmann, Urbán and Défago 2006]. Cada processo em um sistema distribuído acessa um módulo local do detector de falhas que mantém uma lista dos processos suspeitos de terem falhado. O detector de falhas pode cometer enganos; um módulo do detector de falhas pode adicionar processos à lista de suspeitos de forma equivocada. Se posteriormente este módulo identificar o engano que cometeu, o mesmo remove da lista de suspeitos os processos que não haviam falhado. Outro engano que o detector de falhas pode cometer é não suspeitar de um processo que falhou. Desta forma, cada módulo detector de falhas insere e remove processos da lista de suspeitos repetidamente. Além disso, em um determinado tempo, o detector de falhas, em processos distintos, pode ter listas de suspeitos diferentes. Cada processo identifica, através do detector de falhas, quais processos do sistema estão falhos.

Uma classificação de detectores de falhas, em termos de duas propriedades abstratas, *completude* e *exatidão*, é apresentada em [Chandra and Toueg 1996]. *Completude (completeness)* requer que processos falhos sejam suspeitos pelo detector, enquanto que a *precisão (accuracy)* restringe os enganos que o detector de falhas pode cometer. A completude é dita *fraca* se após um intervalo de tempo suficientemente grande, todo processo falho é permanentemente suspeito por *algum* processo correto; se a suspeita ocorre em *todos* os processos corretos, a completude é dita *forte*. A precisão também é classificada em forte (nenhum processo é suspeito antes de falhar) e fraca (ao menos um processo correto nunca é suspeito). A precisão pode ainda ser caracterizada "após um tempo"(*eventual*): a partir de um tempo os enganos não ocorrem.

As propriedades acima combinadas resultam em oito classes de detectores de falhas. O detector de falhas mais fraco que resolve o problema do consenso em sistemas assíncronos é o detector com completude fraca e exatidão fraca após um tempo, chamado $\diamond$W [Chandra, Hadzilacos and Toueg 1992]. A implementação prática do detector $\diamond$W, [Chandra and Toueg 1996] baseado em uma estratégia ingênua de *timeouts* predefinidos pode fazer com que todos os processos corretos sejam repetidamente adicionados

e removidos da lista de suspeitos, violando a precisão requerida do detector. Na teoria, incrementar o valor do *timeout* a cada engano assegura que em algum momento não haverá mais *timeout* prematuro em pelo menos um processo correto p. Entretanto esta estratégia pode resultar em detectores que não são práticos, especialmente tendo em vista que a latência da correção pode levar a atrasos na tomada de decisão de reconfiguração do sistema.

Em [Chen, Toueg and Aguilera 2002], Chen, Toueg e Aguilera especificam critérios para avaliar a qualidade de serviço (QoS - *Quality of Service*) de detectores de falhas. De maneira simplificada, duas famílias de métricas são propostas: tempo para detecção – que avalia quão rápido o detector de falhas suspeita de processos que falham; e precisão – quão bem um detector evita enganos. Há um compromisso entre estas duas métricas, intuitivamente: um detector que é muito rápido pode ser mais propenso a cometer enganos, isto é, suspeitar erroneamente de processos corretos. Tais enganos não são necessariamente permanentes: o detector pode parar de suspeitar de um processo correto mais tarde. Aguilera e outros propõem diversas métricas que refletem este comportamento, como, por exemplo, o tempo de repetição de enganos e a duração de um engano.

2.1. O *Framework* SNMP-FD

Wiesmann, Urbán e Défago apresentam em [Wiesmann, Urbán and Défago 2006] o *framework* SNMP-FD - um serviço de detecção de falhas de processos em redes locais baseado em SNMP (*Simple Network Management Protocol*). Antes de descrever o SNMP-FD, conceitos básicos do SNMP são brevemente apresentados com o objetivo de tornar o trabalho auto-contido.

O SNMP [Harrington, Presuhn and Wijnen 2002] é o padrão da Internet para gerência sendo hoje vastamente disponível na rede. Qualquer dispositivo que possua uma interface de rede pode ser monitorado e controlado através do SNMP. Além de dispositivos físicos, o SNMP permite a gerência de *software* e serviços. A cada dispositivo estão associadas informações de gerência, fundamentais para o seu gerenciamento. As informações de gerência são organizadas em uma MIB *(Management Information Base)*, que é uma base de dados virtual que reúne informações sobre os dispositivos gerenciados de uma rede [Presuhn 2002].

Uma MIB contém um conjunto de *objetos* gerenciados. Um objeto gerenciado é uma visão abstrata de um dispositivo gerenciado de uma rede. Os objetos SNMP são, na verdade, variáveis simples e tabelas, associadas a características básicas, como tipo e permissão de leitura e escrita. Os objetos gerenciados são organizados em uma hierarquia de árvore e são identificados por um OID *(Object Identifier)*. As folhas desta árvore representam os objetos gerenciados e a sua estrutura separa os objetos em grupos. Uma MIB é descrita através de uma representação abstrata dos seus dados, utilizando a linguagem ASN.1 *(Abstract Syntax Notation One)*. As operações do SNMP possibilitam a leitura e alteração do valor de objetos nas MIBs, além do envio de alarmes (*traps*).

No SNMP-FD, toda a comunicação realizada internamente e com o detector de falhas é feita utilizando mensagens SNMP. As mensagens incluem, por exemplo, *heartbeats* e notificações na mudança do estado de execução de um processo. Elas contém informações das diferentes MIBs do sistema. As seguintes informações são incluídas: o identificador do *host* do agente monitorado, o intervalo entre *heartbeats*, o identificador e

o estado de cada processo monitorado, o intervalo entre *heartbeats* requerido (que precisa ser maior que o intervalo corrente), e o contador de *heartbeats*.

As MIBs mais relevantes para o serviço de detecção de falhas são descritas a seguir.

Host Resource MIB [Waldbusser and Grillo 2000]: MIB padrão para a obtenção de informações sobre recursos em um *host*. Os recursos que podem ser monitorados via esta interface incluem dispositivos, sistemas de arquivos, *softwares*, e processos em execução. Este último item é o de interesse para o serviço de detecção de falhas. Processos são identificados por um inteiro de 32 *bits* sem sinal.

Target and Notification MIBs [Levi, Meyer and Stewart 1999]: Estas MIBs especificam mecanismos de transporte, endereços de destino e filtros de mensagens SNMP.

Alarm Description MIB [Chrisholm and Romascanu 2004]: Descreve alarmes nos agentes. Um alarme é definido por uma indicação de falha persistente. Esta MIB define tabelas para descrever possíveis tipos de alarmes, as notificações associadas e os alarmes ativos.

Event MIB [Kavasseri and Stewart 2000]: pode ser usada para descrever eventos que podem disparar notificações se certas condições são satisfeitas.

O serviço pode operar em dispositivos e serviços que tenham um agente SNMP. São monitorados processos que executam em um conjunto de hosts. Os processos são classificados em monitorados e monitores. Processos monitores capturam informações sobre processos monitorados. Um processo pode ser monitor, monitorado ou ambos. Cada *host* executa um agente SNMP local para implementar um processo monitor. Processos monitorados e monitores são registrados no agente SNMP local. Os agentes são responsáveis por troca de informações de monitoração. Em caso de falha do agente, o mesmo é automaticamente reinicializado pelo sistema operacional. Agentes possuem uma cópia de seu estado interno em memória estável. Na reinicialização, o agente segue a convenção SNMP para agentes e envia uma mensagem de início para todos os alvos registrados.

Para validar o sistema, Wiesmann, Urbán e Défago implementaram o serviço SNMP-FD. A implementação foi feita em Java e a ferramenta aberta SNMP4J [SNMP4J]. Esta ferramenta é usada para implementar as MIBs do sistema, e também disponiliza as aplicações para assinatura a um sistema de notificações de mudanças nos estados de execução dos processos. Foi implementado um detector de falhas simples baseado em *timeouts*. Este detector assume estados para processos conforme proposto por Gorender [Gorender, Macêdo and Raynal 2005]. O detector de falhas retorna três possíveis valores para cada processo: confiável, suspeito ou falho. Um processo é considerado confiável se o *heartbeat* foi recebido num intervalo de tempo menor que o *timeout* estipulado pelo detector de falhas. Após este intervalo, se o *heartbeat* não for recebido, o processo é considerado suspeito. Um processo é considerado falho se uma informação de falha for recebida do agente local que o monitora.

Na próxima seção é apresentado o serviço de detecção de falhas proposto neste trabalho. Entre as diversas diferenças para o SNMP-FD - listadas na Introdução - destaca-se que processos podem executar em múltiplos sistemas autônomos, além do fato do SNMP ser usado apenas para prover a interface para o oráculo, e não para a comunicação

de mensagens internas do sistema.

3. Um Serviço baseado em SNMP para Detecção de Falhas em Sistemas Distribuídos na Internet

Nesta seção são apresentadas a arquitetura e a implementação do serviço de detecção de falhas proposto neste trabalho. Aplicações distribuídas podem utilizar o detector de falhas como um oráculo, a partir do qual podem obter informações sobre o estado de seus processos. Essas informações são obtidas através do acesso a uma MIB SNMP, utilizando aplicações tradicionais do SNMP. Além do SNMP, o detector proposto utiliza serviços Web, que viabilizam a monitoração de processos que estejam executando em múltiplas redes e sistemas autônomos. A aplicação usuária e o detector não precisam estar executando na mesma rede. Antes de detalhar a arquitetura, uma visão geral de conceitos básicos de serviços Web é apresentada para tornar o trabalho auto-contido.

Os serviços Web formam um conjunto de tecnologias abertas, padronizadas pelo W3C *(World Wide Web Consortium)* [w3c] para comunicação entre aplicações na Internet. O serviço provê um conjunto de funcionalidades através de uma interface padrão, que garante a interoperabilidade entre arquiteturas diferentes, e provê uma abstração sobre a implementação de serviços. Serviços Web são fortemente baseados em XML *(eXtensible Markup Language)*. Além da interoperabilidade e popularidade, outro fator que também incentiva o uso dos serviços Web é a possibilidade de utilização de portas normalmente abertas, o que permite a comunicação através de *firewalls*.

Existem diferentes arquiteturas para a implementação de serviços Web. Para garantir interoperabilidade, os serviços Web possuem diversos componentes padronizados. Em particular destaca-se o protocolo utilizado para troca de dados na Internet, o SOAP *(Simple Object Access Protocol)* [SOAP], suportado pela maioria das implementações de serviços WEB [Dialani, Miles, Moreau, Roure and Luck 2002].

Os serviços Web têm sido utilizados na gerência de redes [Vianna, Fioreze, Granville, Almeida and Tarouco 2006]. Além de outras funcionalidades, eles podem prover a comunicação entre entidades de gerência localizadas em sistemas autônomos distintos. Neste trabalho, serviços Web são utilizados como *gateways* para entidades que implementam o protocolo de gerência SNMP. Não é necessário o uso de serviços Web quando o comando SNMP é executado diretamente na rede local, e é interpretado pelo agente SNMP local. Entretanto, toda comunicação entre redes distintas é feita com serviços Web.

Neste trabalho o módulo de serviços Web interage com um agente SNMP através de comandos SNMP, obtendo informações sobre o estado de execução dos processos monitorados. Os serviços Web também interagem com as aplicações usuárias, recebendo requisições e devolvendo informações a respeito do estado de processos que estão executando fora da rede local. Módulos de serviços Web que executam em redes locais diferentes interagem entre si, com o intuito de trocar informações sobre os processos monitorados. É necessária a execução de um módulo de serviços Web para cada rede local onde existe ao menos um processo monitorado por uma aplicação externa à esta rede local, ou para cada rede local onde existe uma aplicação que monitora o estado de processos em outras redes locais. O serviço Web foi implementado neste trabalho utilizando-se a linguagem Python [Python].

A figura 1 ilustra a arquitetura do serviço de detecção de falhas proposto. Na figura são ilustrados quatro processos monitorados pelo detector: P_{A1}, P_{A2}, P_{B1} e P_{B2}. Neste exemplo, os processos estão executando em *hosts* diferentes. Os *hosts* $HOST_{A1}$, $HOST_{A2}$ e $HOST_{A3}$ pertencem à LAN_A, e os *hosts* $HOST_{B1}$, $HOST_{B2}$ e $HOST_{B3}$ pertencem à LAN_B. Cada rede local possui ainda um agente SNMP que mantém a *Process Status (PS) MIB*, um monitor, uma interface com a Internet através de serviços Web, e uma aplicação usuária do serviço de detecção de falhas.

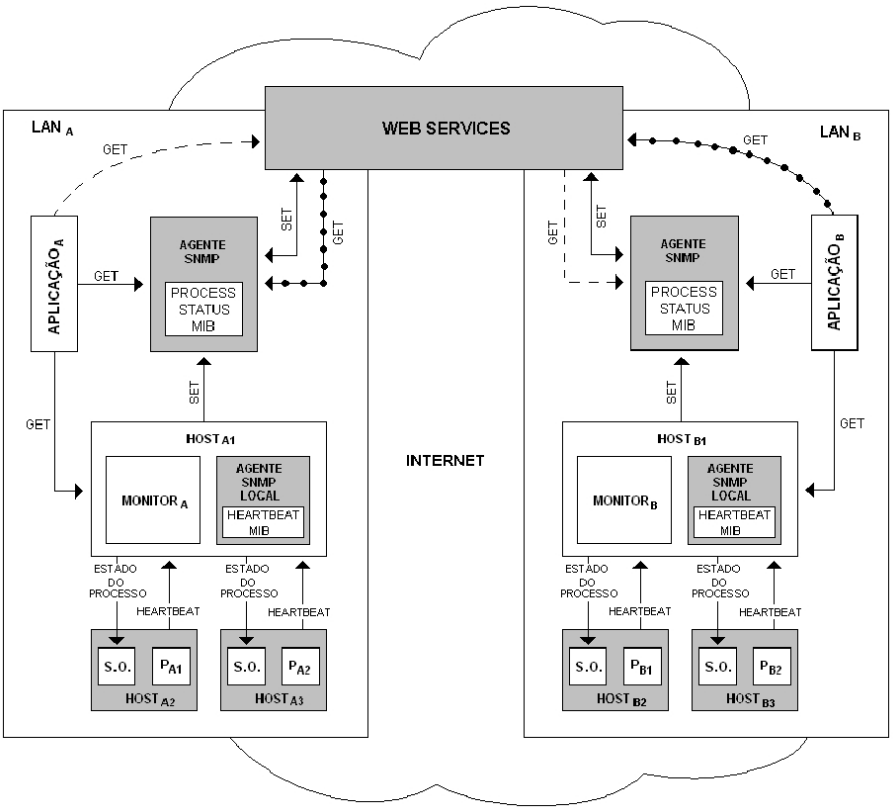

Figura 1. Arquitetura da ferramenta.

A aplicação é a entidade que utiliza o sistema para monitorar o estado de execução de seus processos, tanto dentro da rede local, quanto na Internet. A interface que a aplicação usa para acessar o detector consiste de comandos SNMP. Deste modo, o usuário do sistema pode implementar a aplicação conforme suas necessidades, podendo acessar o serviço desde que tenha acesso a comandos SNMP. Por exemplo, observe o cenário da figura 1, e suponha que a aplicação da LAN_A necessita conhecer o estado do processo P_{B1}. A aplicação deve utilizar o SNMP através dos serviços Web da LAN_A para requisitar o estado do processo P_{B1} ao módulo de serviços Web da LAN_B. Deve ser informada a identificação do processo P_{B1}. O módulo de serviços Web da LAN_B consulta o agente SNMP local e retorna o estado do processo P_{B1} ao módulo de serviços Web da LAN_A. O módulo de serviços Web da LAN_A repassa este resultado para a aplicação. Desta forma, não é necessário que as MIBs tenham conteúdo sincronizado, pois o módulo de serviços Web busca as informações na Internet de acordo com a demanda da aplicação. Observando a mesma figura 1, e supondo que a aplicação da LAN_A necessita conhecer o estado do processo P_{A1}. Basta a aplicação acessar o módulo SNMP para obter o estado

do processo.

Processos monitorados são processos de aplicação. Os três estados possíveis para os processos são: *sem-falha*, *suspeito* ou *falho*. Para ser monitorado, um processo precisa estar registrado no sistema. O processo deve incluir uma biblioteca que inicia uma *thread* para enviar *heartbeats* periodicamente ao monitor. O monitor continuamente calcula se esses *heartbeats* são recebidos antes do limite de *timeout*. Se estes *heartbeats* são recebidos pelo monitor dentro do intervalo de *timeout*, então o processo é dito sem-falha. Se os *heartbeats* não são recebidos pelo monitor dentro do intervalo de *timeout*, então o processo é dito suspeito. Ao ser considerado suspeito, o monitor verifica a existência do PID *(Process ID)* do processo no sistema operacional do *host* no qual o processo está sendo executado. Se o sistema operacional não retorna o PID do processo, então o processo é considerado falho, visto que o mesmo não existe mais na visão do sistema operacional. O *timeout* mencionado anteriormente é calculado utilizando uma variação do algoritmo utilizado pelo TCP para determinar o RTO *(Retransmission Timeout)*. Este algoritmo é detalhado adiante.

O agente SNMP mantém na PS-MIB informações de estados de processos, e é responsável pela interface SNMP disponibilizada pelo sistema para interagir com aplicações usuárias. É através desta MIB que a aplicação conhece o estado de seus processos que estão executando na rede local. A PS-MIB também responde a requisições SNMP vindas dos serviços Web, ou seja, vindas da Internet. A PS-MIB é uma tabela com duas entradas, uma para identificar os processos atraves de endereço IP, porta (TCP ou UDP), e PID, e outra entrada para armazenar o estado de cada processo, que pode ser *sem-falha*, *suspeito* ou *falho*. Cada rede local, que possui ao menos um processo monitorado, deve ter um agente SNMP que mantém a PS-MIB. Se uma rede é utilizada apenas para monitorar processos que estão executando em outras redes locais, então não há necessidade de execução deste agente SNMP. Neste caso, o acesso é feito diretamente através dos serviços Web.

O sistema pode ainda ser executado com uma opção de envio de notificações por parte da PS-MIB local para todas as outras redes locais, através dos serviços Web. Sendo assim, a PS-MIB além de armazenar as informações relacionadas aos processos locais, também armazena infomações dos processos remotos. Com essa opção, a MIB é atualizada tanto pelo módulo monitor, quanto pelos serviços Web.

O monitor é o módulo responsável pela supervisão dos processos pertencentes à aplicação na rede local. Para isto, utiliza uma estratégia baseada em *heartbeats*. O monitor também utiliza o sistema operacional local de cada processo para a determinação do seu estado. O monitor atualiza a MIB quando um novo processo se registra no sistema (através da biblioteca de envio de *heartbeats*), ou quando há mudança no estado de um processo. É necessária a execução de um monitor em cada rede local onde exista ao menos um processo monitorado. Se uma rede local é utilizada apenas para executar uma aplicação usuária, que necessita de informações sobre o estado de processos que estão executando em outras redes locais, então não há necessidade de execução do módulo monitor.

O monitor recebe *heartbeats* de todos os processos monitorados em um intervalo estipulado. Se o monitor não recebe um *heartbeat* de um processo sem-falha dentro de um tempo limite calculado, então o estado deste processo é atualizado para suspeito. O cálculo deste tempo limite é baseado no algoritmo utilizado no TCP para determinar o

RTO *(Retransmission Timeout)* [Jacobson 1988] [Paxson and Allman 2000], conforme a expressão abaixo.

$$Diferenca = Tempo\_hb_i - Tempo\_hb_{i-1}$$
$$Media_i = \alpha * Media_{i-1} + (1 - \alpha) * Diferenca$$
$$Desvio_i = \alpha * Desvio_{i-1} + (1 - \alpha) * |Media - Diferenca|$$
$$TempoLimite = Media_i + \beta * Desvio_i$$

Nesta expressão, $Tempo\_hb$ é a hora local no momento em que um *heartbeat* é recebido. $Diferenca$ corresponde à diferença entre a hora local no momento em que um *heartbeat* é recebido e a hora local correspondente ao recebimento do *heartbeat* anterior. $Media$ acumula o valor médio de $Diferenca$. Neste caso é dado um peso maior ao valor médio histórico, para que um único atraso pontual no valor da variável $Diferenca$ não interfira intensamente no valor do tempo limite. Isto permite uma redução na taxa de engano. $Desvio$ corresponde ao desvio médio, uma aproximação do desvio padrão. Quanto maior o ganho para $Desvio$, maior será o valor do tempo limite. $TempoLimite$ corresponde ao valor que é utilizado para determinar se um processo é suspeito ou não. Os valores iniciais de $Media$ e $Desvio$ são constantes atribuídas conforme o intervalo entre *heartbeats* enviados por cada processo. Para α e β foram atribuídos os valores 0.9 e 4 respectivamente. O cálculo do tempo limite se adapta ao intervalo entre *heartbeats*, sendo desnecessária a atribuição de um valor fixo. Desta forma, é possível ter processos enviando *heartbeats* com intervalos diferentes.

O monitor também mantém uma MIB, a HB-MIB, que armazena dados estatísticos sobre os *heartbeats*, como intervalo médio de recebimento, desvio padrão e hora do recebimento do último *heartbeat*. Esses dados também podem ser consultados pela aplicação usuária através de comandos SNMP.

4. Avaliação Experimental

Com o objetivo de demonstrar o funcionamento e avaliar o sistema, foram executados e são apresentados cinco experimentos: (1) consumo de CPU dos processos variando o intervalo entre *heartbeats*; (2) consumo de CPU do monitor variando o intervalo entre *heartbeats*; (3) tempo de notificação de falhas no PlanetLab; (4) taxa de engano ao longo do tempo; e (5) tempo de detecção de falhas variando o intervalo entre *heartbeats*. Os experimentos foram executados tanto em rede local quanto no PlanetLab [PlanetLab].

A implementação do serviço de detecção de falhas foi feita na linguagem C (monitor SNMP) e Python (serviços Web). Em particular, para a comunicação entre processos em sistemas autônomos distintos, foram utilizadas as bibliotecas de serviços Web *xmlrpclib* [xmlrpclib] e *SimpleXMLRPCServer* [SimpleXMLRPCServer]. Um processo que utiliza a biblioteca *xmlrpclib* acessa métodos com parâmetros em processos remotos; o transporte é feito com HTTP. A biblioteca *SimpleXMLRPCServer* provê um arcabouço básico para implementação de serviços Web. Para implementar agentes SNMP foi utilizado o arcabouço Net-SNMP [Net-SNMP], versão 5.2.4. O Net-SNMP provê funcionalidades e ferramentas para implementação de sistemas SNMP, incluindo um agente extensível e aplicações SNMP diversas.

Para medir o consumo de CPU, monitor e processos monitorados foram executa-

dos em hosts distintos. Foi configurada uma variação no intervalo entre *heartbeats* enviados pelos processos monitorados. A figura 2 demonstra o consumo de CPU do monitor. Para cada ponto foram coletadas 50 amostras e o gráfico mostra a média e o intervalo de confiança de 95%. Durante todo o experimento ficou caracterizado um baixo consumo de CPU, que variou de 0 % até 0.3 %. Argumentamos que a diminuição do intervalo de *heartbeats* leva a resultados equivalentes aos que seriam obtidos com aumento do número de processos monitorados em termos da quantidade de *heartbeats* processados. Sob este ponto de vista o sistema se mostrou escalável.

A figura 3 demonstra o consumo de CPU dos processos monitorados, também com variação do intervalo entre *heartbeats*. Para cada ponto foram coletadas 50 amostras e o gráfico mostra a média e o intervalo de confiança de 95%. Para um intervalo entre *heartbeats* de 1 milisegundo, a média de consumo de CPU do processo ficou em aproximadamente 2 %, caindo para 0.7 % quando o intervalo entre *heartbeats* é fixado em 2 milisegundos. Para intervalos entre *heartbeats* acima de 2 milisegundos, o consumo de CPU não superou 1 %. Mais uma vez, fica caracterizado o baixo consumo de CPU.

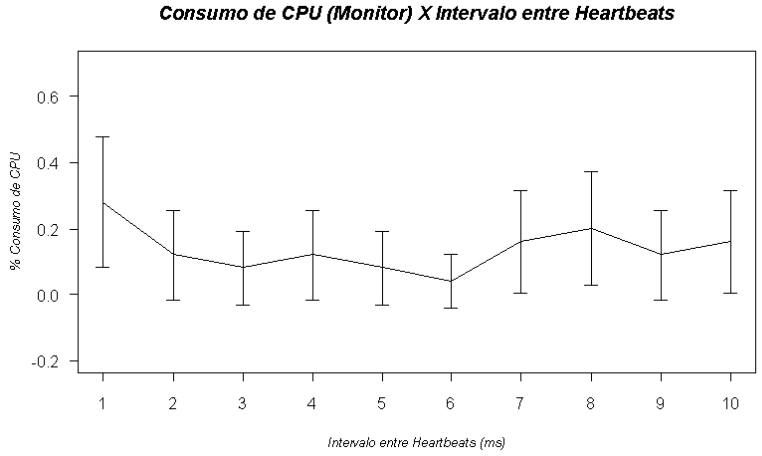

Figura 2. Consumo de CPU (Monitor) X Intervalo entre *Heartbeats* (ms)

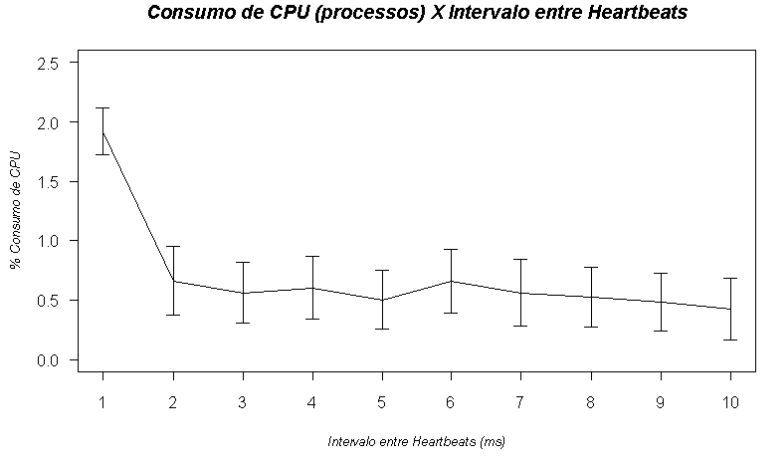

Figura 3. Consumo de CPU (Processos) X Intervalo entre *Heartbeats* (ms)

A figura 4 demonstra o comportamento do sistema no PlanetLab. O eixo x indica os nodos no PlanetLab nos quais foram feitas as medidas, e o eixo y indica o tempo de notificação em segundos. Este tempo foi calculado mantendo um servidor executando serviços Web no Brasil, que calcula o tempo gasto na notificação, após ter sido forçada uma suspeita de processo monitorado. Foram coletadas 50 amostras por nodo, e o gráfico ilustra a média e o intervalo de confiança de 95% destas amostras. Os clientes executam em hosts de cinco países de cinco continentes: América do Sul, América do Norte, Europa, Ásia e Oceania.

O gráfico na figura 4 demonstra que o tempo de notificação respeita a distância física entre os nodos. O cliente que executa no nodo Brasil, o mais próximo do servidor, apresenta tempo de notificação de aproximadamente 0.1 segundos. Conforme a distância física entre os nodos aumenta, o tempo de notificação também aumenta, chegando a aproximadamente 1,1 segundos para cliente que executa na Nova Zelândia. Vale observar que o tempo de notificação de falha é consideravelmente maior do que o tempo de detecção da falha na rede local do processo que sofre o evento, ilustrado na figura 6.

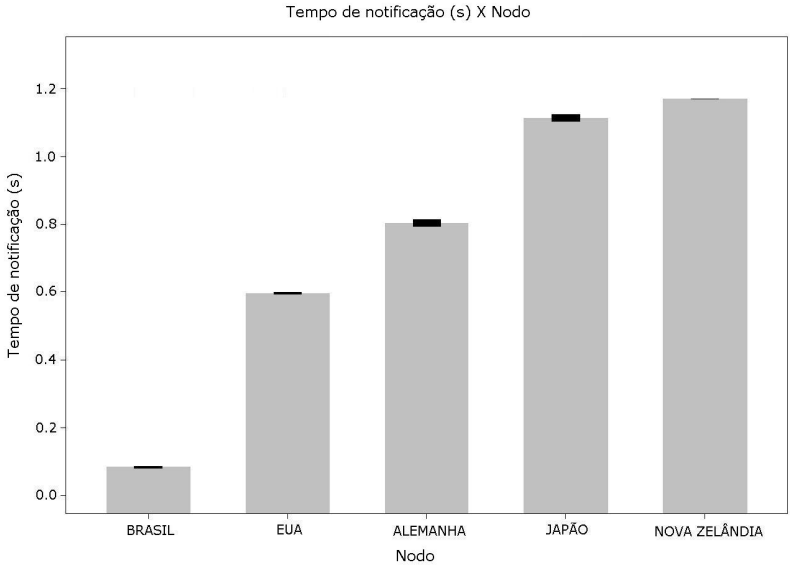

Figura 4. Tempo de Notificação (s) vs. Nodos do PlanetLab

A figura 5 ilustra a taxa de enganos por segundo, com variação ao longo do tempo. O intervalo entre *heartbeats* foi fixado em 1 milissegundo para este experimento. Monitor e processo monitorado são executados em hosts diferentes em uma mesma rede local. Para este experimento foi utilizado um processo que não falhou durante todo o período de teste, ou seja, toda falha detectada pelo monitor caracteriza um engano. Como é utilizado o algoritmo de RTO do TCP, espera-se que a quantidade de engano diminua com o passar do tempo, até estabilizar. Este resultado pode ser observado no gráfico. Para este experimento foram coletadas 60 amostras para cada minuto, e o gráfico mostra a média e o intervalo de confiança de 95%. Observando o gráfico, percebe-se que a média da taxa de engano ficou em aproximadamente 1,9 enganos nos primeiros 3 minutos, caindo para 0,5 enganos por segundo no minuto 12, estabilizando a partir deste valor, permanecendo abaixo de 0,7 enganos por segundo no restante do tempo. A partir do minuto 27 até o restante do tempo, ocorre uma queda suave na taxa de engano, e nota-se também

que o intervalo de confiança fica menor neste período, comprovando a tendência que o algoritmo tem de se adaptar às condições da rede.

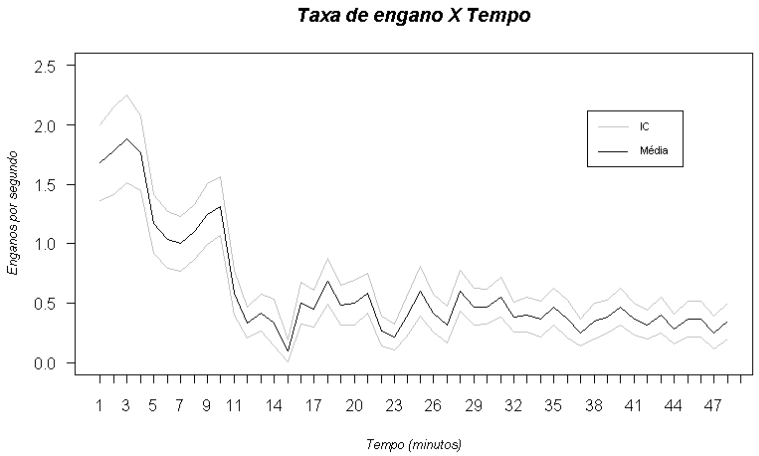

Figura 5. Taxa de Engano X Tempo (min).

A figura 6 ilustra o tempo de detecção de falhas em milissegundos, variando o intervalo entre *heartbeats* também na ordem de milissegundos. Monitor e processo monitorado foram executados em um mesmo *host*, permitindo a utilização do relógio local. Foi forçada a falha de um processo. Tanto o tempo da falha, quanto o tempo no qual o monitor detecta essa falha são registrados. A diferença entre esses dois tempos é o chamado *tempo de detecção*. O gráfico de linha da figura 6 reflete a média de 60 amostras, e as retas verticais ilustram o intervalo de confiança de 95 %. O eixo x contém o intervalo entre *heartbeats*, variando entre 7 e 21 milisegundos. No eixo y está representado o tempo de detecção em milissegundos. Teoricamente, o tempo de detecção nunca pode ser maior do que o intervalo entre *heartbeats*, e no gráfico pode-se perceber este comportamento. Para um intervalo entre *heartbeats* de 7 ms, o tempo de detecção é de aproximadamente 13 ms. Essa relação se mostra proporcional; à medida em que o intervalo entre *heartbeats* aumenta, o tempo de detecção também aumenta, sendo que no intervalo entre *heartbeats* de 21 ms, o tempo de detecção é de aproximadamente 26 ms.

5. Conclusão

Neste trabalho apresentamos um serviço de deteção de falhas de processos de sistemas distribuídos executados em múltiplos sistemas autônomos da Internet. O detector de falhas funciona como um oráculo distribuído que disponibiliza informações sobre o estado dos processos monitorados. Este serviço pode ser utilizado por aplicações que invocam o oráculo usando comandos SNMP. Foi apresentada a arquitetura do serviço. Para permitir a comunicação entre sistemas autônomos distintos foram utilizados serviços Web que usam o HTTP como transporte.

Com o objetivo de demonstrar o funcionamento e avaliar o sistema, foram executados e são apresentados resultados de cinco experimentos, executados em rede local e em hosts de cinco continentes: América do Sul, América do Norte, Europa, Ásia e Oceania. Os experimentos mediram o consumo de CPU variando o intervalo entre *heartbeats*; o tempo de notificação de falhas no PlanetLab; a taxa de engano ao longo do tempo; e o

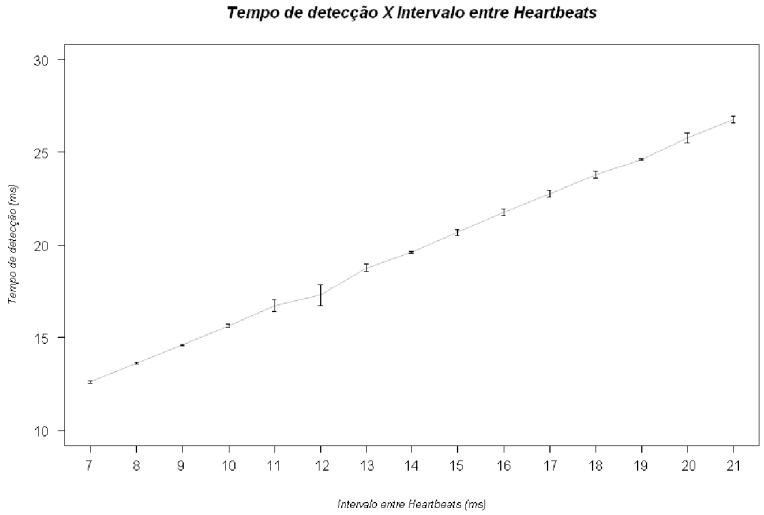

Figura 6. Tempo de Detecção (ms) X Intervalo entre *Heartbeats* (ms)

tempo de detecção de falhas variando o intervalo entre *heartbeats*. Os resultados permitem concluir que o sistema é eficaz e apresenta baixa sobrecarga.

Entre os trabalhos futuros, a ferramenta está sendo usada para implementação de acordo sobre múltiplos sistemas autônomos da Internet, em particular envolvendo replicação distribuída.

Referências

Greve, F. G. P. (2005). Protocolos Fundamentais para o Desenvolvimento de Aplicações Robustas. Minicurso, XXIII Simpósio Brasileiro de Redes de Computadores (SBRC).

Chandra, T., Hadzilacos, V. and Toueg, S. (1992). The Weakest Failure Detector for Solving Consensus. *Proceedings of the Eleventh ACM Symposium on Principles of Distributed Computing*.

Chandra, T. and Toueg, S. (1996). Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the Association for Computing Machinery*, Vol. 43, No. 2.

Chen, W., Toueg, S. and Aguilera, M.K. (2002). On The Quality of Service of Failure Detectors. *IEEE Transactions on Computers*.

Chrisholm, S. and Romascanu, D. (2004). Alarm Management Information Base. *Request for Comments 3877 (RFC3877)*.

Dialani, V., Miles, S., Moreau, L., Roure, D. D. and Luck, M. (2002). Transparent Fault Tolerance for Web Services Based Architectures. *Eighth International Europar Conference*.

Fischer, M. J., Lynch, N. A. and Paterson, M. S. (1985). Impossibility of Distributed Consensus with one Faulty Process. *Journal of the Association for Computing Machinery*, Vol. 32, No. 2.

Gorender, S., Macêdo, R. and Raynal, M. (2005). A Hybrid and Adaptative Model for Fault-Tolerant Distributed Computing. *Proc. of the Int. Conf. on Dependable Systems and Networks (DSN)*.

Guerraoui, R. and Rodrigues, L. (2006). Introduction to Reliable Distributed Programming. *Springer*.

Harrington, D., Presuhn, R. and Wijnen, B. (2002). An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. *Request for Comments 3411 (RFC3411)*.

Jacobson, V. (1988). Congestion Avoidance and Control. *Computer Communication Review*, Vol. 18, No. 4.

Kavasseri, R. and Stewart, B. (2000). Event MIB. *Request for Comments 2981 (RFC2981)*.

Levi, D., Meyer, P. and Stewart, B. (1999). SNMP Applications. *Request for Comments 2573 (RFC2573)*.

Net-SNMP. http://www.net-snmp.org/. Acesso em dezembro/2009.

Paxson, V. and Allman, M. (2000). Computing TCP's Retransmission Timer. *Request for Comments 2988 (RFC2988)*.

PlanetLab. http://www.planet-lab.org/. Acesso em agosto/2009.

Presuhn, R. (2002). Management Information Base (MIB) for the Simple Network Management Protocol (SNMP). *Request for Comments 3418 (RFC3418)*.

Presuhn, R. (2002). Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). *Request for Comments 3416 (RFC3416)*.

Python Programming Language. http://www.python.org/. Acesso em agosto/2009.

Raynal, M. (2004). Detecting Crash Failures in Asynchronous Systems: What? Why? How?. *Proc. Int. Conference on Dependable Systems and Networks (DSN'04)*.

SimpleXMLRPCServer. http://docs.python.org/library/simplexmlrpcserver.html. Acesso em dezembro/2009.

SNMP4J. http://www.snmp4j.org/. Acesso em dezembro/2009.

SOAP (Simple Object Access Protocol). http://www.w3.org/TR/soap/. Acesso em agosto/2009.

Turek, J. and Shasha, D. (1992). The Many Faces of Consensus in Distributed Systems. *Computer*, páginas 8–17, Vol. 25, No. 6.

Vianna, R. L., Fioreze, T., Granville, L. Z., Almeida, M. J. B. and Tarouco, L. M. R. (2006). Comparando Aspectos de Desempenho do Protocolo SNMP com Diferentes Estratégias de Gateways Web Services. *In: 24 Simpósio Brasileiro de Redes de Computadores (SBRC 2006)*, Curitiba, PR.

Waldbusser, S. and Grillo, P. (2000). Host Resources MIB. *Request for Comments 2790 (RFC2790)*.

Wiesmann, M., Urbán, P. and Défago, X. (2006). An SNMP Based Failure Detection Service. *IEEE Symposion on Reliable Distributed Systems (SRDS)*.

W3C (The World Wide Web Consortium). http://www.w3.org/. Acesso em agosto/2009.

xmlrpclib. http://docs.python.org/library/xmlrpclib.html. Acesso em dezembro/2009.