Confiabilidade em sistemas de backup distribuído seguro usando a plataforma peer-to-peer

Trabalho Técnico

Marcos Pinheiro Duarte[1]

Rodrigo Elia Assad[2]

Silvio Romero de Lemos Meira[3]

[1]Centro de informática – Universidade Federal do Pernambuco (UFPE)
Caixa Postal 15.064 – 91.501-970 – Recife – PE – Brazil

{mpd@cin.ufpe.br, assad@cesar.org.br, silvio@cesar.org.br}

Abstract. *Reports of users are common about inconvenience and damage for loss of important files at work, such problems can be easily solved by means of backup. But, the high cost of mounting such a system (purchase of media, specialized computers, high-capacity hard drives, among others), becomes an obstacle to financial institutions which have a high number of employees. In this context the platforms peer-to-peer networks can be used to implement such systems using idle resources of the network itself where it is deployed. The criticality of this system requires that its development process incorporates mechanisms for quantification of reliability to ensure that files are injected in the network, are available when requested.*

Resumo. *Relatos de usuários são comuns acerca de transtornos e prejuízos pela perda de arquivos importantes de trabalho, tais problemas podem ser facilmente solucionados através de operações de backup.Porém, o alto custo de montagem de um sistema dessa natureza (compra de mídias, computadores especializados, discos rígidos de grande capacidade, entre outros), torna-se um empecilho financeiro para instituições as quais possuem um elevado numero de colaboradores. Nesse contexto as plataformas peer-to-peer podem ser utilizadas para implementar tais sistemas utilizando recursos ociosos da própria rede onde ele será implantado. A criticidade desse sistema exige que seu processo de desenvolvimento incorpore mecanismos de quantificação de confiabilidade de modo a garantir que os arquivos injetados na rede, estejam disponíveis quando solicitados.*

1. Introdução

É comum ouvir que algum colaborador de um projeto ou empresa perdeu todos os seus arquivos, e juntamente com ele boa parte do seu trabalho que não tinha cópia em outro lugar. Para evitar contratempos dessa natureza, as empresas estão procurando e investindo em soluções baratas e eficazes.

Fazer backups de máquinas pessoais é caro e pouco utilizado, pois, a quantidade de espaço requerido requer um investimento, principalmente em hardware,

muito alto. Outro fato é que injetar um conjunto de arquivos grande na Web é bastante lento e suscetível a diversos erros.

Terceirizar o serviço ainda não se mostrou uma solução eficiente e barata, por exemplo, em Michigan's college prover um serviço de backup onde cada máquina tem direito a 8 GB de espaço custa cerca de $30 por mês [12].

Nesse contexto, grids computacionais em peer-to-peer tem ganhado força por compartilhar seus recursos ociosos.

Uma maneira de minimizar os gastos dos sistemas de backup é fazer usufruto desta ociosidade de recursos. Mas, para que sistemas dessa natureza possam ser utilizados, é necessário que tenhamos confiança no mesmo. Fatores como disponibilidade, confiabilidade, segurança e proteção são muito importantes.

2. Confiança de Software

Boa parte da sociedade está familiarizada com os problemas dos sistemas computacionais. Algumas vezes por questões que fogem ao entendimento, os sistemas falham ao disponibilizar seus serviços. Ocasionalmente essas falhas podem causar a perda ou corromper os arquivos gerenciados por estes. Assim, poucos confiam totalmente nos microcomputadores que costumamos utilizar.

Segundo a norma ISO/IEC 9126 [1] podemos definir confiança como sendo a "capacidade do produto de software de manter um nível de desempenho especificado, quando usado em condições especificadas".

Avaliar se um sistema é confiável ou não envolve analisar dados quantitativos e qualitativos. Com base nesse contexto, Sommerville [3] classifica essa avaliação em quatro pilastras, são elas:

- **Disponibilidade**: É a probabilidade de um sistema, em determinado instante, ser operacional e capaz de fornecer os serviços requeridos.
- **Confiabilidade**: É a probabilidade de operação livre de falhas durante um tempo especificado, em um dado ambiente para um propósito especifico.
- **Segurança**: É um atributo que reflete a capacidade do sistema de operar, normal e anormalmente, sem ameaçar as pessoas ou ambiente.
- **Proteção**: Avaliação do ponto em que o sistema protege a si mesmo de ataques externos, que pode ser acidentais ou deliberados.

Disponibilidade e confiabilidade são em essência probabilidades e, portanto são expressos em valores numéricos. Já segurança e proteção são feitos com base em evidências na organização dos sistemas e normalmente não são expressos em valores numéricos. Freqüentemente são medidos através de níveis, e a ordem de grandeza destes denomina se um sistema é mais ou menos seguro/protegido do que outro.

O aumento de confiança gera um overhead de custo, pois envolvem atividades que demandam maior tempo e profissionais qualificados, como se pode observar no gráfico 01 a curva se mostra exponencial quando altos índices de confiabilidades são requeridos. De modo geral, pode-se dizer que não é possível que um

sistema seja 100 por cento confiável, uma vez que a curva de gastos tende ao infinito. Além dos altos investimentos, outro problema que encontrado seria o desempenho, tornar um sistema mais confiável envolve também escrever código extra, freqüentemente redundante, para a verificação de estados excepcionais e para possibilitar a recuperação a partir de falhas ocorridas no sistema.

Analisando ainda sobre uma ótica menos técnica e mais voltada ao negócio ainda possuímos atributos como:

- **Perda de informações**: É extremamente dispendioso coletar e manter dados, algumas vezes a própria base de dados pode valer muito mais que o sistema que o processa.
- **Prejuízos altos**: Para aplicações como controle de aeronaves, o custo de falha do sistema é maior do que o próprio sistema que o controla.
- **Readequação da confiança**: ao contrário do desempenho, que apresenta problemas em pequenos pedaços do software, a falta de confiança e/ou integridade tende a está distribuída em todo o sistema, tornam-se muito dispendioso eliminar os bugs e falhas apresentados.

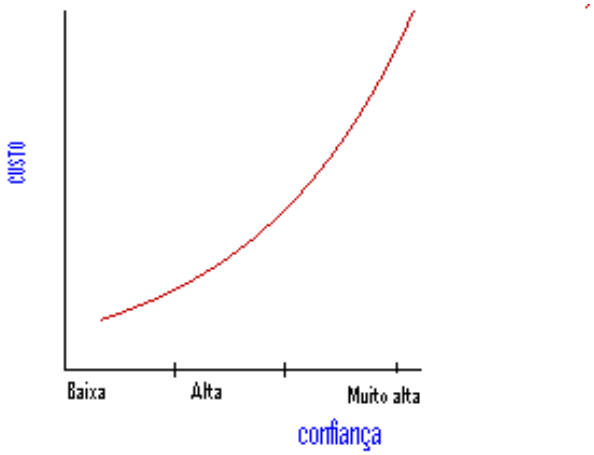


Gráfico que representa a confiança e o custo necessário para atingi-la.

As falhas de sistemas controlados por software causam normalmente inconveniência, mas não danos prolongados ou sérios. Mas, alguns tipos de aplicações, no entanto os problemas supracitados podem trazer danos físicos ou ameaça a vidas humanas. Esses sistemas no geral são chamados de sistemas críticos.

A confiança nesse tipo de software é um requisito essencial, em todos os aspectos (disponibilidade, confiabilidade, segurança e proteção). Existem 3 tipos principais de sistemas críticos: de segurança, missão e negócios.

- **Os sistemas críticos de segurança**: são aqueles cuja falha pode levar a ferimentos, grande prejuízo ambiental ou a perda de uma vida. Alguns exemplos desse tipo de sistema são os freios ABS implementados em alguns carros, sistema de

controle de produtos químicos de uma fábrica e sistema de administração de medicamentos em pacientes.

- **Os sistemas críticos de missão**: são aqueles cuja falha leva a prejudicar alguma atividade orientada a metas, como exemplo desse tipo de aplicação temos o controle de navegação das aeronaves.
- **Os sistemas críticos de negócio:** são aqueles cuja falha pode levar um negócio a falência, um bom exemplo desse tipo de aplicação seriam os sistemas bancários que movimentam uma grande quantidade de recursos financeiros.

Os altos custos de falhas que envolvem esse tipo de sistema significam que a integridade das técnicas e dos processos de desenvolvimento, em geral, são mais importantes do que os custos relativos à aplicação dessas técnicas. Como exemplo, o uso de especificação e verificação formal de um programa em relação a suas especificações, este tipo de validação pode consumir até 50 por cento do recurso total envolvido no projeto.

É fato que existe uma fatia de problemas a serem solucionados por software que precisam de cuidados especiais e investimento maior. Logo, necessita-se de uma maneira segura para quantificar e gerenciar a qualidade durante todo o ciclo de vida de tal software.

a. Métricas de confiança de software

Em meados de 1991 foi publicada a norma ISO/IEC 9126 [1] para definir características da qualidade de produto de software. Nesta, encontram-se subdivisões quanto aos modelos de qualidade, métricas externas, internas e de qualidade de uso. Desde sua definição, a norma passou por diversos ajustes e em 2003 especificou 26 métricas de confiabilidade. Estas são classificadas em interna, quando dizem respeito a um produto não executável do software (código fonte, especificação, entre outros) e externa que se aplicam a partir da execução e comportamento do software.

Dentre as 26 métricas supracitadas, oito são internas e as demais (dezoito) são externas. Respeitando o escopo do trabalho, foram descartadas as métricas internas, que implicam em intervenção humana e as métricas externas que não podem ser medias de maneira automática, restando sete itens que são descritos na tabela abaixo:

Métrica	Descrição	Fórmula matemática
Tempo médio entre falha (MTBF)	Taxa de freqüência em que o software falha em operação. (falha recuperável)	$X = T / A$, onde T representa o tempo total em operação, A representa o número total de falhas detectadas
Tempo médio de indisponibilidade	Qual é o tempo médio em que o sistema fica indisponível quando uma falha acontece.	$X = T/N$, onde T é o tempo total de indisponibilidade, N é o número de indisponibilidade

Remoção de defeitos	Quantos defeitos foram sanados	$X = A1/A2$, onde A1 é o número de defeitos corrigidos, A2 é o número de defeitos encontrados
Prevenção de indisponibilidade	Com que freqüência o sistema causa a indisponibilidade completa do ambiente de produção	$X = 1 - A/B$, onde A é o número de indisponibilidades e B é o número de falhas
Disponibilidade	Percentagem de tempo em que o sistema se encontra operacional e capaz de oferecer os serviços requerido	$X =$ To / (To + Tr), onde To é o tempo de operação do sistema, Tr é o tempo de reparo do sistema
Densidade de defeitos	A quantidade de defeitos detectada durante o período de teste	$X = A/B$, onde A é o número de defeitos encontrados, tamanho do sistema
Densidade de defeitos latente estimada	Quantidade de defeitos que existem ou ainda podem surgir como falhas futuras	$X = \|A1 - A2\|/B$, números de defeitos latentes estimados, A2 o número de defeitos encontrados e B o tamanho do sistema.

Conceitualmente falha e defeito são diferentes, o primeiro significa a interrupção do serviço, o segundo significa que houve um erro de programação (bug) no sistema.

As métricas existem para validar o quão o software atende a qualidade esperada, em outras palavras, é a quantificação de determinadas características para posterior comparação com os números desejáveis.

Para organizar todo o processo de coleta, analise e inferência dos resultados, estudiosos na área desenvolveram vários modelos.

b. Modelo de confiança de software

A quantidade de modelos de software vem crescendo paulatinamente. Atualmente pode-se encontrar mais de 200, criados desde a década de 70.

Crespo [4] concorda com Musa [5] quando afirma que modelos de confiabilidade de software servem para poder obter informações a cerca de:

- Número médio de falhas ocorrido em algum ponto no tempo;
- Número médio de falhas ocorridos em algum intervalo de tempo;
- A intensidade de falhas em algum ponto do tempo;
- A distribuição de probabilidades de falhas;

Ainda seguindo as colocações dos autores, temos que os modelos teriam a utilidade de:

- Avaliar quantitativamente as tecnologias empregadas;
- Oportunamente avaliar a qualidade de um projeto na sua fase de concepção;

- Pode ser utilizado para monitoramento do desempenho operacional do software;
- Controlar as falhas introduzidas durante alterações no software;

Conforme descrito oportunamente no inicio desse tópico, existem diversos modelos catalogados na literatura e um resultado satisfatório depende da escolha, pois um modelo não se adéqua bem a todas as realidades. Fatores como: tipo de ambiente, o conjunto de dados capturados, limite superior de falhas, cobertura dos testes, maturidade do software são essenciais.

Quando se trata de confiabilidade em sistemas de backup seguro utilizando peer-to-peer, é necessária a existência de mecanismos de monitoramento e quantificação de confiabilidade. Na literatura existem dois tipos de mecanismos de controle, os baseados em tempo e em cobertura.

Os baseados em domínio do tempo representam a confiabilidade em ao longo do tempo. Segundo Lyu[7], os modelos mais utilizados são: Jelinski e Moranda, geométrico, Goel-okumoto model, Musa basic model, NHPP Melhorado (ENHPP) e os Bayesianos.

Modelos baseados em cobertura medem a confiabilidade ao longo do progresso das atividades, onde a confiabilidade não é mais função do tempo, apenas da cobertura dos critérios de teste escolhido.

Pelos critérios adotados este trabalho, a medida adotada será a confiabilidade em função do tempo.

i. Jelinski e Moranda

O modelo desenvolvido por Jelinski e Moranda [17] em 1972 foi um dos primeros modelos de confiabilidade propostos e ainda é amplamente utilizado e referenciado. Este assume que o tempo entre falhas segue uma distribuição exponencial cujo parâmetro é proporcional ao número de defeitos restantes no software. O tempo médio entre o momento que ocorreu uma falha t_{i-1} e o momento que ocorrerá a próxima falha t_i é dado por:

$$\Delta t = \frac{1}{\varphi(N - i + 1)}$$

onde N é o número de defeitos presentes no programa no início dos testes e o parâmetro φ é uma constante de proporcionalidade.

O modelo parte das seguintes hipóteses:

- O número de defeitos N no início dos testes é um valor fixo mas desconhecido.

- Cada defeito tem a mesma probabilidade de ser ativado, causando uma falha, e a taxa de falhas de cada defeito é constante $(z(t) = \varphi)$.
- As falhas não estão correlacionadas. Ou seja, dado N e φ, os tempos entre as falhas $\Delta t_1, \Delta t_2, ..., \Delta t_n$ são variáveis aleatórias independentes e distribuídas exponencialmente.
- A remoção dos defeitos após a ocorrência de uma falha é instantânea e não introduz novos defeitos no software.

Como conseqüência destas suposições, a taxa de falhas do sistema após a remoção do defeito (i — 1) é proporcional ao número de defeitos restantes no software. Assim, temos:

$$z(\Delta t|t_{i-1}) = \varphi\left(N - M(t_{i-1})\right) = \varphi\left(N - (i-1)\right)$$

O modelo pertence a uma classe de modelos de tipo binomial. Para estes modelos, a função de intensidade de falhas é o produto do número de defeitos inicial do sistema e a densidade de probabilidade do tempo de ativação de uma falha $f_a(t)$ [22]. As funções valor médio e intensidade de falhas para este modelo são:

$$\mu(t) = N\left(1 - e^{-\varphi t}\right)$$

$$\lambda(t) = N\varphi e^{-\varphi t} = \varphi\left(N - \mu(t)\right)$$

A confiabilidade $R(t_i)$antes da remoção do defeito i é dada por:

$$R(t_i) = e^{-\varphi(N-(i-1))t_i}$$

ii. Modelo Geométrico

O modelo geométrico [23] assume que o programa contém, inicialmente, um número ilimitado de defeitos e que os mesmos estão divididos em classes de dificuldade, com probabilidades diferentes de ativação. À medida que os defeitos que se encontram nas classes das mais fáceis de serem detectados são corrigidos, a probabilidade de se observar uma falha diminui. A distribuição das probabilidades dos tempos entre falhas é exponencial e sua média decresce exponencialmente.

O modelo parte das seguintes hipóteses:

- Existe um número infinito de defeitos no software.
- Cada defeito tem a mesma chance de ser encontrado dentro de uma mesma classe de dificuldade.
- A taxa de falhas forma uma progressão geométrica e é constante entre a detecção de defeitos, ou seja, $z(t) = D\varphi^{i-1}$.
- As falhas não estão correlacionadas.

As funções valor médio e intensidade de falhas para este modelo são:

$$\mu(t) = \frac{1}{\beta} ln\left[D\beta exp(\beta)t + 1\right]$$

$$\lambda(t) = \frac{Dexp(\beta)}{D\beta exp(\beta)t + 1}$$

Onde, $\beta = -ln(\varphi)$ para $0 < \varphi < 1$

iii. Modelo de Goel-Okumoto (G-O)

O modelo proposto por Goel e Okumoto [16] também é chamado de modelo de processo de Poisson não homogéneo (Non-Homogeneous Poisson Process, NHPP) e é baseado nas seguintes suposições:

- O número de falhas no software que ocorrem em $(t, t+\Delta T]$ com $\Delta t \to 0$ é proporcional ao número esperado de defeitos não encontrados, $N - \mu(t)$. A constante de proporcionalidade é φ.
- Os números de falhas detectadas em cada um dos intervalos $(0, t_1), (0, t_2), ..., (0, t_n)$ são valores independentes entre si.
- A taxa de falhas de cada defeito é uma constante φ.
- A correção dos defeitos não introduz novos defeitos.
- O número de falhas ocorridas durante um tempo t segue uma distribuição de Poisson com valor médio. $\mu(t)$.

Estas suposições resultam na seguinte formulação matemática para a média do número esperado de falhas observadas durante um tempo í e a função de intensidade de falhas:

$$\mu(t) = N\left(1 - e^{-\varphi t}\right)$$

$$\lambda(t) = N\varphi e^{-\varphi t} = \varphi\left(N - \mu(t)\right)$$

Uma vez que cada defeito é removido sem a introdução de novos defeitos, o número de defeitos presentes no software no início dos testes é igual ao número de falhas que terão ocorrido após um tempo de testes infinito. De acordo com as suposições, $\mathbf{M}(\infty)$ segue uma distribuição de Poisson com uma média igual a N. Então, N é o número esperado de defeitos iniciais no software. E essa é uma das

principais diferenças deste modelo comparado ao modelo de Jelinski-Moranda, onde N é um valor fixo mas desconhecido.

iv. Modelo de Musa

Os dois modelos anteriores consideram apenas o tempo transcorrido e não fazem considerações sobre o esforço requerido nas atividades de teste. Com isto, assumem implicitamente que este esforço é constante durante todo o processo. Porém, na prática, isto não é verdadeiro. Faltam neste modelo algumas suposições que considerem este esforço.

O modelo de Musa foi o primeiro a incorporar o esforço requerido nos testes de software exigindo que as medidas de tempo sejam feitas baseadas no tempo de processamento ou o tempo de execução do programa, r. Por isso, este modelo também é conhecido como modelo de tempo de execução.

A intensidade de falhas do sistema é dada por:

$$\lambda(t) = NB\varphi e^{-B\varphi\tau}$$

onde N tem o mesmo significado que no modelo G-O e B é o fator de redução de defeitos, o qual é uma constante que relaciona a taxa de correção de defeitos com a taxa de risco de cada defeito. A equação anterior pode ser reescrita como:

$$\lambda(t) = \varphi\,(N - \mu(t)) = Kf\,(N - \mu(t))$$

onde φ é formulado como o produto da freqüência de execução linear f, pela taxa de redução de defeitos K, que pode ser interpretada como o número médio de falhas ocorrentes por defeito remanescente no código durante uma execução linear do programa [18].

Esta última equação sugere que o modelo do Musa seja matematicamente equivalente aos modelos de Jelinski-Moranda e Goel-Okumoto. Isto de fato é verdadeiro, sendo que a principal diferença é a formulação da taxa de falhas por defeito φ.

v. O Modelo NHPP Melhorado (ENHPP)

O modelo NHPP melhorado (Enhanced Non-homogeneous Poisson Process, ENHPP) incorpora dados de cobertura de teste em sua formulação analítica

como uma medida de esforço de teste [19]. A cobertura de teste é definida como a razão entre o número de elementos executados pelo conjunto de testes e o número total de elementos requeridos pelo critério de teste. O modelo faz as seguintes suposições:

- Os defeitos estão uniformemente distribuídos entre os elementos requeridos pelo critério de teste. Isto é, ao ser executado, cada elemento tem a mesma probabilidade de apresentar um defeito.
- A probabilidade de um elemento apresentar um defeito ao ser executado em um dado tempo t é $c_d(t)$.
- Os defeitos são corrigidos sem efeitos colaterais.
- A função valor médio dos defeitos encontrados em um dado tempo t é:

$$\mu(t) = c(t)\check{N}$$

onde $c(t)$ é uma função da cobertura dos testes e $\check{N}$ é o número esperado de defeitos ao se atingir a cobertura total, diferindo dos outros modelos onde N era o número total de defeitos presentes no início dos testes. A intensidade de falhas para o modelo ENHPP é expressa por:

$$\lambda(t) = z(t)(\check{N} - \mu(t))$$

onde $z(t)$ é a função da taxa de risco por defeito. A confiabilidade é dada por:

$$R(t/s) = e^{-\check{N}K(c(s-t)-c(s))}$$

onde s é o tempo em que ocorreu a última falha e t é o tempo transcorrido desde a última falha.

vi. Modelos Bayesianos

O raciocínio bayesiano é baseado na suposição de que as variáveis de interesse são governadas por uma distribuição de probabilidade e que as decisões corretas podem ser realizadas analisando estas probabilidades junto com os dados observados [20].

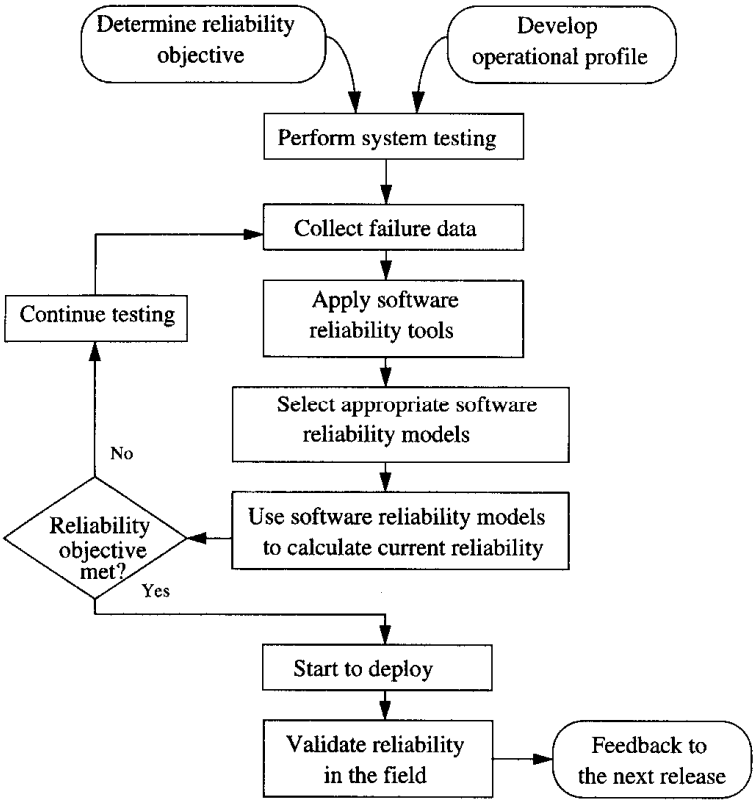
Diferentemente dos modelos anteriores, os modelos bayesianos não necessitam de dados de falha para ser formulado. Uma formulação inicial pode ser realizada utilizando dados de outros projetos anteriores ou até pela experiência de um profissional em projetos anteriores [21].

À medida que o software é testado, o modelo é atualizado, mesmo quando não ocorrem falhas, pois o modelo bayesiano considera também o tempo de execução livre de falhas.

3. Design

a. Confiabilidade

A confiabilidade do sistema dar-se-á agregando ao processo de desenvolvimento escolhido, mecanismos de quantificação, histórico e controle de confiança, para tal, alguns componentes e sua seqüência de execução devem ser respeitadas, podemos ver o fluxo conforme a figura abaixo:



Os primeiros passos dizem respeito a definir os objetivos da confiabilidade e definir um profile operacional. O primeiro significa levantar o que tem que ser alcançado para atingir a satisfação dos usuários, o segundo, é uma estimativa de uso por parte dos desenvolvedores e o levantamento dos tipos de entrada.

A atividade posterior é coletar os dados de testes para entrada em alguma ferramenta de automação, onde está irá trabalhar o resultado com base no modelo de confiabilidade escolhido.

A resposta de todo o processo é numérica, e caso não tenha se chagado aos objetivos definidos inicialmente, o modelo é realimentado até que se chegue aos objetivos pré-definidos. Caso, estes objetivos tenham sido alcançados, a aplicação entra em produção e os feedbacks colhidos com todo o processo alimentam a próxima etapa de desenvolvimento.

b. Proteção

O sistema não deve permitir que as informações sejam entendidas somente por quem as interessam. Com base nessa premissa, o sistema exige autenticação, troca de informação e estruturas de arquivos injetados na rede devem estar criptografadas.

A autenticação não será implementada mediante uma infra-estrutura de certificação e chave pública, será utilizado o método de Diffie-Hellman para troca de chaves. Cada peer autentica e adiciona um número seqüencial para cada mensagem, e um valor de função de criptografia para a mensagem e para o número criptografado.

c. Disponibilidade

Um dos pontos mais críticos do sistema está na disponibilidade, pois sempre que um usuário precisar restaurar seu backup este deverá encontrar-se acessível. Conceitualmente pode se dizer que é a probabilidade de um sistema está disponível quando solicitado, conforme definição no capítulo 2.

Em sistemas de backup como este, onde os arquivos são quebrados em pedaços e distribuídos, a disponibilidade está relacionada a um conjunto de peers os quais possuam todas as partes necessárias para a operação de restauração.

Faz-se necessário um rankeamento dos peers que apresentam maior disponibilidade para distribuição desses pedaços e assim aumentar as chances de acesso dos mesmos quando solicitados.

4. Considerações finais

Em suma, este trabalho procurou focar-se, dentro do contexto backup em redes peer-to-peer, no levantamento sobre o que é confiabilidade, suas características, como se mensura e os modelos mais utilizados. O design do sistema em peer-to-peer permite por essência a escalabilidade, ou seja, adicionar novos recursos é transparente ao sistema e para quem faz usufruto. Por fim, é importante salientar seu custo, que se mostra mais acessível pelo fato de utilizar recursos ociosos no conjunto de computadores os quais compõe a rede.

5. Trabalhos futuros

Para continuidade de tal trabalho, é preciso desenvolver o conjunto de atividades:

1- Selecionar um processo de desenvolvimento e adequá-lo ao processo de confiabilidade levantado

2- Estabelecer critérios de escolha para a ferramenta de análise de confiabilidade
3- Implementar a arquitetura
4- Elaborar o algoritmo de rankeamento dos peers.

6. Trabalhos relacionados

Existem diversos trabalhos relacionados a backup usando a plataforma peer-to-peer, nessa sessão será descrito os projeto e características de cada um.

pStore é uma ferramenta que combina um algoritmo chamado distributed hash table (DHT) que quebra de arquivos em tamanhos fixos e uma técnica de controle de versão [13]. A idéia é bastante completa, há suporte para encriptação de arquivos, mantendo a privacidade nas máquinas clientes bem como manter versões de arquivos já salvos, reaproveitando os pedaços de arquivos que se repetem em versões anteriores. Oferece suporte as operações de inserção, atualização, recuperação e deleção de arquivos. Contudo, o projeto ainda é embrionário (fase de pesquisa) e não foi implementado. Um ponto a ser evoluído ainda no sistema é que o mesmo faz uso do método rsync [14] na sincronização novas versões de arquivos já salvos, que é um método pouco eficiente.

Pastiche é um sistema de backup coorporativo que oferece um mecanismo de confiabilidade, detecção de erro, proteção contra peers maliciosos e integridade [12]. Faz uso de fingerprints, que é uma maneira inferir a similaridade entre dois peers. Esta semelhança é medida levando em consideração a quantidade de blocos de arquivos que na ótica cliente/servidor eles têm em comum, otimizando assim o processo de transferência de arquivo.

Samsara estimula a troca de recurso de maneira igualitária sem uso de uma terceira entidade no sistema para intermediar a comunicação[11]. O mecanismo utilizado para garantir a troca justa de recursos é um esquema punitivo para o peer que por ventura perdem dados, atividade típica de usuários que querem utilizar e não disponibilizar recursos. Uma outra característica é que a quantidade de espaço consumida da rede, tem que ser disponibilizada no disco local para uso no sistema.

CleverSafe Dispersed Storange é uma aplicação open source desenvolvida para a plataforma CentOS. Faz uso do algoritmo de dispersão (IDA) e recuperação da informação injetada na rede. A ferramenta garante escalabilidade por não ter servidores centralizados e uma garantia de disponibilidade dos backups de 99,999999999999 (12 noves) por cento, podendo chegar a 16 noves de acordo com as configurações. A ferramenta ainda apresenta problemas de encriptação, autenticação e controle de acesso e orienta seus usuários a configurar sua rede com proteção de um firewall e/ou uma VPN.

Gibs [8] é um ferramenta open source implementada em Python que pratica backup incremental, ou seja, quando solicitado por automaticamente ou por um usuário sobre um arquivo previamente salvo, de maneira inteligente o arquivo não é salvo novamente, salvo modificações feitas no mesmo. Um ponto negativo dessa ferramenta é que embora ofereça segurança, economia de recursos, entre outros ela foi

concebida para pessoas que entendam o funcionamento do processo backup e instalação da ferramenta. Em resumo, não é fácil entendimento para o usuário final.

Resilia [9] é um protótipo de sistema de backup que combina peer-to-peer com compartilhando secreto e seguro de arquivos, faz uso de um esquema de replicação para aumentar a disponibilidade e permite a reconstrução de backups perdidos por falhas de nodos que compõe a rede.

7. Referências

[1] NBR ISO/IEC 9126, Engenharia de software - Qualidade de produto. Modelo de qualidade, 2003.

[2] A. Tridgell and P. Macherras. The rsync algorithm.Technical report, TR-CS-96-05, Australian National University, Jun 1996.

[3] Sommerville, lan. Engenharia de Software/lan Sommerville. São Paulo, 2003. P. 299-311.

[4] CRESPO, Adalberto N., Modelos de Confiabilidade de Softwares baseados em cobertura de critérios estruturais de tese de doutoramento de teste de software, UNICAMP, Campinas: 1997.

[5] MUSA, J. D., Ianino , A ., Okumoto, K., Software Reliability-Measurement, Prediction, Application, Mc Graw-Hill, New York, 1987.

[6] Emin Martinian. Distributed internet backup system (dibs). http://www.csua.berkeley.edu/_emin/source_code/dibs/.

[7] Handbook of software reliability engineering / Michael R. Lyu, editor in chief p. cm. Includes index. ISBN 0-07-039400-8 (alk. paper) 1. Computer software—Reliability— Handbooks, manuals, etc. I. Lyu, Michael R. QA76.76.R44H36 1995005.1 — dc20.

[8] Emin Martinian. Distributed internet backup system (dibs). http://www.mit.edu/~emin/source_code/dibs/index.html .

[9] Fernando Meira. Resilia: A safe & secure backup-system. Final year project, Engineerng Faculty of the University of Porto, May 2005.

[10] CleverSafe. Cleversafe dispersed storage project. http://www.cleversafe.org/dispersed-storage.

[11] Landon P. Cox and Brian D. Noble. Samsara: honor among thieves in peer-to-peer storage. In SOSP ’03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 120–132, New York, NY, USA, 2003. ACM Press.

[12] Landon P. Cox, Christopher D. Murray, and Brian D. Noble. Pastiche: making backup cheap and easy. SIGOPS Oper. Syst. Rev., 36(SI):285–298, 2002.

[13] Christopher Batten, Kenneth Barr, Arvind Saraf, and Stanley Trepetin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCSTM-632,

Massachusetts Institute of Technology Laboratory for Computer Science, October 2002.

[14] A. Tridgell and P. Macherras. The rsync algorithm.Technical report, TR-CS-96-05, Australian National University, Jun 1996.

[15] Jeantao, Pan. Software Reliability. Carnegie Mellon University 18-849b Dependable Embedded Systems Spring 1999.

[16] A. L Goel and K. Okumoto. Time-dependent error-detection rate model for software reliability and other performance measures. IEEE Transactions on Reliability, 28:206-211, 1979.

[17] P. L. Moranda and Z. Jelinski. Final report on software reliability study. Technical Report MADC Report, No, 63921, McDonnell Douglas Astronautics Company, 1972.

[18] Ganesh J. Pai. A survey of softwre reliability models. Technical report, Department of DCE, University of Virginia, 2002.

[19] Swapna S. Gokhale and Kishor S. Trivedi. A time/structure based software reliability model. Annals of Softwre Engineering, 8:85-121, 1999.

[20] Tom Mitchell. Machine Learning. McGraw-Hill, 1997.

[21] Dinesh D. Narkhede. Bayesian model for software reliability. Technical report, Indian Institute of Technology, 2001.

[22] John J. Marciniak. Encyclopedia of Software Engineering, volume 2. John Winsley & Sons, 1994.

[23] P. L. Moranda. Predictions of software reliability during debugging. In Proceedings ofthe Annual Reliability and Maintainability Symposium, 1975.