UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Rafael Köhler Baggio

UMA FERRAMENTA PARA MONITORAÇÃO E ALERTAS SMS EM UM AMBIENTE DE CLUSTER COM ALTA DISPONIBILIDADE

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Mário Antônio Ribeiro Dantas

Florianópolis, Novembro de 2004.

UMA FERRAMENTA PARA MONITORAÇÃO E ALERTAS SMS EM UM AMBIENTE DE CLUSTER COM ALTA DISPONIBILIDADE

Rafael Köhler Baggio

_____
Prof. Dr. José Mazzucco Júnior
Coordenador do Curso

Banca Examinadora

_____
Orientador: Prof. Dr. Mário A. R. Dantas

_____
Prof. Dr. Roberto Willrich

_____
Prof. Dr. Vitório B. Mazzola

Florianópolis, Novembro de 2004

“E aí a solução está mais ou menos não sei aonde.”
- Júlio Felipe Szeremeta

Agradeço aos meus pais, ao meu orientador, aos membros da banca e a todas as pessoas que de alguma forma contribuíram para a elaboração deste trabalho. Agradeço também a todos os membros da turma cco992, que certamente contribuíram para a formação deste profissional. Um agradecimento especial para minha namorada Talice, que contribuiu para este trabalho com seu carinho, compreensão e amor.

RESUMO

Atualmente Computação de Alta Performance (HPC) está tornando-se mais necessária e ao mesmo tempo mais acessível. Isso deve-se a tendência de migração das plataformas especializadas para sistemas mais baratos e de propósito geral, ou seja, *clusters*. No aspecto de manutenção, isso implica em maior empenho por parte do responsável pelo sistema computacional, pois um *cluster* é formado por vários sistemas independentes.

Este trabalho tem como objeto a implementação de uma ferramenta de software que auxilie na tarefa de manutenção de um *cluster*, monitorando todos os seus nodos e, caso encontre alguma falha, alertando o responsável através de uma mensagem SMS (Short Message Service). O serviço SMS tem basicamente a mesma funcionalidade do serviço de *pager*, porém garantem a entrega da mensagem mesmo que o dispositivo destino esteja indisponível momentaneamente.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

CAPÍTULO 1

1. Introdução

Nos últimos tempos, a demanda por Computação de Alta Performance (HPC) vem se tornando cada vez maior. Cientistas dependem de máquinas com alto poder de processamento para poderem levar suas pesquisas a diante e conforme suas pesquisas avançam e tornam-se mais complexas, mais poder de processamento é necessário. Na área da genética, por exemplo, a quantidade de dados a serem processados é imensa.

Para suprir essa necessidade é preciso que ocorra o processamento de informações em paralelo, o que pode ser obtido utilizando-se supercomputadores. Pois com processamento seqüencial o tempo que um computador levaria fazendo cálculos seria absurdamente grande, a ponto de tornar-se inviável. Como supercomputador, podemos entender um sistema capaz de executar processamento em paralelo, atingindo um alto desempenho.

Uma abordagem para construção de supercomputadores é através da montagem de um *cluster* que distribui tarefas para serem processadas por vários computadores comuns aproveitando sua crescente capacidade de processamento e baixo custo. Um *cluster* é formado por vários computadores interligados entre si, sendo um deles o nodo mestre que é quem distribui as tarefas para serem processadas pelos demais nodos, chamados de escravos.

Além de ter alta capacidade de processamento, é interessante um *cluster* possuir também alta disponibilidade, ou seja, estar disponível para uso aproximadamente 100% do tempo. Diz-se aproximadamente pois na prática não pode-se garantir que ocorra 100% de disponibilidade de um sistema computacional, pois devido a sua natureza, tais sistemas estão sujeitos a falhas de hardware, software e elementos externos, como por exemplo a falta de energia elétrica. Uma maneira para alcançar alta disponibilidade, é através da redundância de componentes críticos, ou seja, essenciais

para o funcionamento do sistema computacional.

Apesar das vantagens, um cluster demanda maior atenção por parte dos administradores de rede, ou até mesmo uma equipe dedicada somente a tarefa de administrar o cluster. Pois quanto maior o número de nodos que compõem um cluster, mais trabalhosa e complexa se torna a tarefa de identificar e corrigir eventuais problemas que possam ocorrer com cada nodo.

Tendo isso em mente, o objetivo deste trabalho é a implementação de uma ferramenta de software que auxilie na tarefa de detectar eventuais falhas que possam ocorrer com os nodos de um *cluster*. Quando ocorrer a detecção de um nodo em estado de falha, um aviso será enviado a um telefone móvel, via SMS, para que o responsável pela administração do *cluster* tome uma atitude para corrigi-la.

CAPÍTULO 2

2. Cluster Computing

Supercomputadores podem ser construídos basicamente de duas maneiras. A primeira é construir uma plataforma especializada para uso específico, unindo componentes com tecnologia de desempenho extremamente alto, interligados de modo que possibilite o processamento paralelo de informações. Construir um supercomputador utilizando esta abordagem é extremamente caro e, após ter sido construído, o custo para expandir sua capacidade de processamento torna-se proibitivo, causando uma rápida obsolescência do equipamento se comparado com um *cluster*. Isso deve-se ao fato de tal abordagem utilizar tecnologias não disponíveis *off the shelf*. Entenda-se por *off the shelf* produtos prontamente disponíveis para venda no comércio comum, sem a necessidade de encomenda.

Outra abordagem é construir um supercomputador constituído da união de outros computadores de propósito geral (PCs) interligados através de uma rede. A esta abordagem denominamos *cluster* e a cada computador que compõe o *cluster* denominamos *nodo*. Os avanços recentes na tecnologia de redes de alta velocidade e de microprocessadores têm tornado os *clusters* um meio relativamente barato e eficiente para se alcançar alto desempenho em processamento paralelo.

Segundo a IEEE TFCC (*Task Force on Cluster Computing*) [5] "a tendência na computação paralela é migrar das plataformas especializadas para sistemas mais baratos e de propósito geral, que consistem na união de componentes independentes construídos a partir de estações de trabalho ou PCs uni ou multi-processados", ou seja, a construção de *clusters* com hardware e software *off the shelf* está redefinindo o conceito de supercomputador.

Além do fato de ter um custo reduzido e um desempenho elevado, outra característica

marcante dos *clusters* é sua capacidade de crescimento contínuo, seja adicionando mais nodos ou substituindo algum componente por outro com tecnologia de maior desempenho e conseqüentemente aumentando a capacidade global de processamento do *cluster*. Outra conseqüência dessa característica é o fato de ser possível adaptar a solução ao investimento financeiro que pretende-se fazer, dimensionando assim o desempenho que será obtido inicialmente, podendo ampliá-lo posteriormente quando for conveniente.

A estrutura básica de um *cluster* é: um nodo atuando como "mestre" que tem o papel de escalonador, distribuindo tarefas para todos os outros nodos, que atuam como "escravos", e montando o resultado final do processamento a partir de cada resultado parcial gerado pelos nodos escravos. Nesta arquitetura há um ponto único de falha evidente: o nodo mestre.

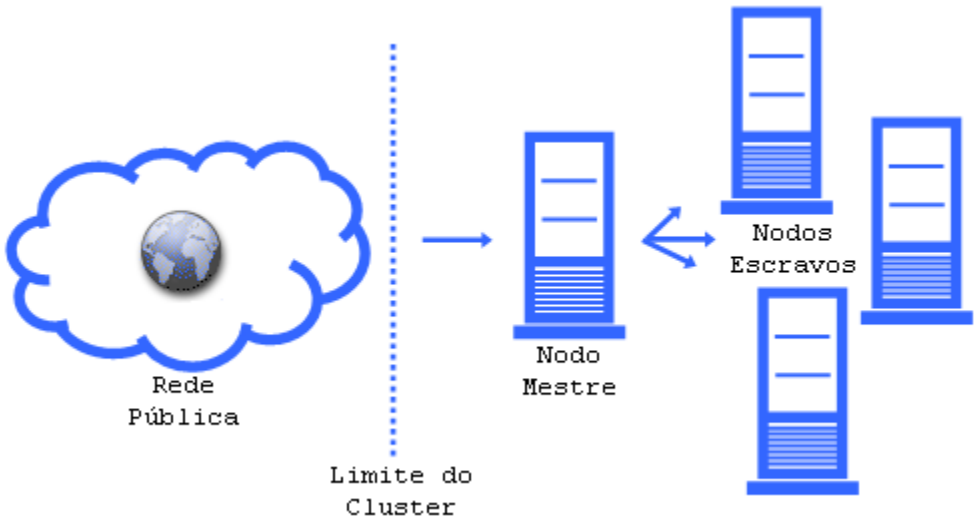

FIGURA 1 - Esquema básico de um *cluster* HPC.

A Figura 1 deixa claro o motivo de o nodo mestre ser um ponto único de falha, ele assume uma posição de ligação entre a rede pública e os nodos escravos do *cluster*, análoga à de uma ponte que é a única via de acesso a uma ilha. Caso ocorra uma falha que venha a tornar o nodo mestre indisponível, todo o cluster se tornará também, causando uma ociosidade que poderia ter sido evitada. Para solucionar este problema utiliza-se técnicas de alta disponibilidade específicas para o

ambiente de *cluster*.

2.1. Alta Disponibilidade

No contexto de sistemas computacionais em geral, Alta Disponibilidade (HA) refere-se a existência de componentes no sistema capazes de substituir instantaneamente outros componentes que tornem-se inativos, independendo da causa dessa inatividade. Desse modo garantindo que um sistema nunca, ou quase nunca, ficará indisponível em conseqüência de uma falha.

Novamente, como ocorre com supercomputadores, existe uma variedade de maneiras para alcançar esta característica "variando de soluções que utilizam hardware customizado e redundante para garantir disponibilidade, a soluções baseadas em software usando componentes de hardware *off the shelf*"[5]. A primeira maneira possibilita um maior nível de disponibilidade, porém tem um custo significativamente mais elevado que a segunda. Tal fato levou a uma popularização da segunda classe de soluções, que tipicamente tratam de um ponto único de falha no sistema.

Num ambiente de *cluster*, a alta disponibilidade é alcançada através da redundância do nodo mestre. O papel do nodo mestre secundário (redundante) é exatamente o mesmo que o nodo mestre primário, porém a maior parte do tempo de vida do cluster, este fica inativo, apenas monitorando o nodo mestre primário. Ao detectar a indisponibilidade do nodo mestre primário, o secundário assume seu papel, executando todas as funções do mestre primário de maneira idêntica. Desse modo essa substituição entre mestre primário e secundário é transparente para os usuários ou aplicações que utilizam o *cluster*. A Figura 2 destaca a diferença de arquitetura entre um *cluster* e um *cluster* que implementa as funcionalidades de alta disponibilidade: o nodo mestre secundário.

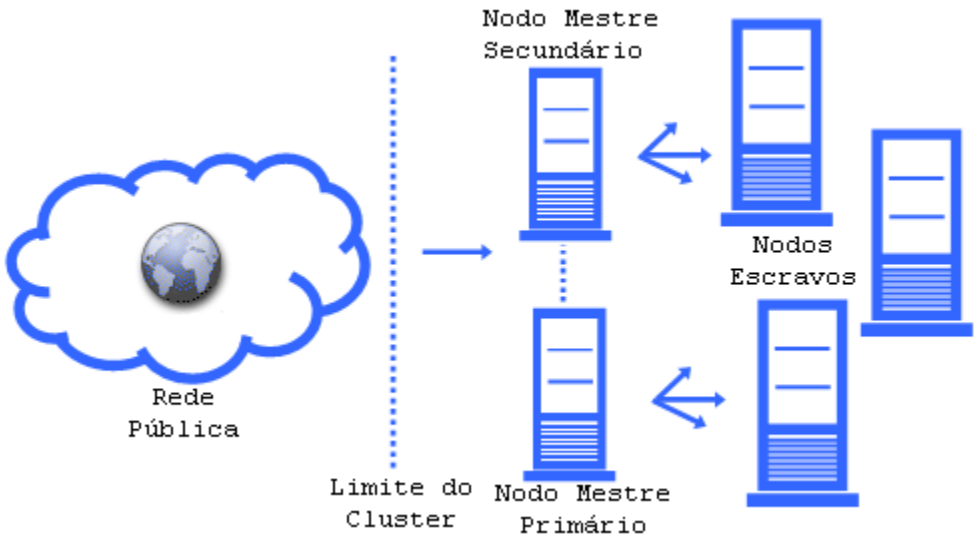

FIGURA 2 - Esquema básico de um *cluster* HA.

Para o melhor entendimento do assunto devemos ter claros os conceitos de alguns termos relacionados a alta disponibilidade:

Disponibilidade Contínua: Implica em serviço disponível 100% do tempo, sem ocorrer períodos em que esteja indisponível. Isso representa um estado ideal não alcançado por nenhum sistema existente atualmente. “Alta Disponibilidade não implica em Disponibilidade Contínua”[5].

Tolerância a Falhas: É um meio de alcançar níveis muito altos de disponibilidade. Um sistema tolerante a falhas tem a capacidade de continuar disponível apesar de uma falha de hardware ou software, e é caracterizado por redundância de hardware, incluindo CPU, memória, e subsistemas de E/S. “Alta Disponibilidade não implica em Tolerância a Falhas”[5].

Ponto Único de Falha: Um componente de hardware ou software cuja perda resulta na perda do serviço. Tais componentes não têm backup de componentes redundantes [5].

Recuperação de Falhas (Failover): Quando um componente em um sistema HA falha resultando em uma perda de serviço, o serviço é iniciado pelo sistema HA em outro componente do sistema. Essa transferência de serviço após uma falha no sistema é chamada de Recuperação de Falhas [5].

Com o crescimento de sua popularidade na última década, os *clusters* estão sendo amplamente utilizados em diferentes áreas, incluindo aplicações de missão crítica. Tipicamente tais aplicações não são tolerantes a eventuais falhas do sistema e a resultante interrupção do serviço, por isso necessitam que o sistema em que executam proporcione baixo tempo de indisponibilidade. Naturalmente isto levou a uma forte demanda por boas soluções de alta disponibilidade para o ambiente de *cluster*.

Existem diversas soluções baseadas em software comercial para alta disponibilidade, porém com a popularização recente do sistema operacional Linux e do software livre, surgiram vários projetos livres e de código aberto para implantação e alta disponibilidade de *clusters*.

Para a realização deste trabalho foram utilizadas duas soluções de software livre, rodando em ambiente Linux. A implantação do *cluster* foi realizada utilizando-se o ambiente de Sistema de Imagem Única (SSI)[8] OSCAR (Open Source Cluster Application Resources)[3] versão 3.0 desenvolvido pelo Open Cluster Group (OCG)[7]. Após implantado o *cluster*, para obter a Alta Disponibilidade, foi utilizado o projeto HA-OSCAR (High Availability OSCAR)[4], também desenvolvido pelo OCG. Maiores detalhes do ambiente serão abordados no capítulo 4.

Apesar desses sistemas tipicamente implementarem mecanismos de recuperação de falhas, há casos de falhas impossíveis de serem corrigidas automaticamente. Nestes casos a interação do responsável pelo *cluster* é necessária para a reparação da falha e recuperação do estado consistente do *cluster*. Para isso o responsável deve fazer uma inspeção periódica nos nodos do cluster e, caso encontre algum problema, tomar as devidas providências. Num *cluster* com poucos nodos essa tarefa é relativamente simples, pois devido ao baixo número de nodos a quantidade de falhas

também será baixa e assim diminuindo a freqüência de inspeções. Com o crescimento do número de nodos essa tarefa torna-se mais trabalhosa e freqüente, pois a probabilidade de falhas aumentam linearmente em relação ao número de nodos.

Assim surge a necesseidade de alguma ferramenta que auxilie na tarefa de monitoração dos nodos. Além de auxiliar na monitoração dos nodos, a ferramenta deve ser capaz de alertar o responsável pelo *cluster*, sem que o mesmo precise estar fisicamente no mesmo ambiente onde o *cluster* está localizado, e ainda sem que precise explicitamente executar alguma ação para receber o alerta. Em outras palavras, o responsável pelo *cluster* será um agente passivo e remoto do sistema. A implementação de uma ferramenta com tais características é o objeto deste trabalho que será abordado com mais detalhes no capítulo 4.

2.2. OSCAR

OSCAR é um projeto *open source* desenvolvido pelo OCG a partir do ano de 2000, com a finalidade de facilitar a tarefa de construção de um *cluster* HPC (Computação de Alto Desempenho). A atual versão estável do projeto (OSCAR 3.0) consiste de um pacote de ferramentas com as finalidades de instalação e configuração, manutenção, programação paralela, sistema de filas e escalonador.

Este projeto é implementado conforme o paradigma SSI (Sistema de Imagem Única), isto é, todo o *cluster* é representado externamente por uma imagem de um único sistema. Em outras palavras, o usuário final tem a impressão de estar usando um sistema único, quando na realidade está utilizando um *cluster* que pode ser composto por milhares de sistemas.

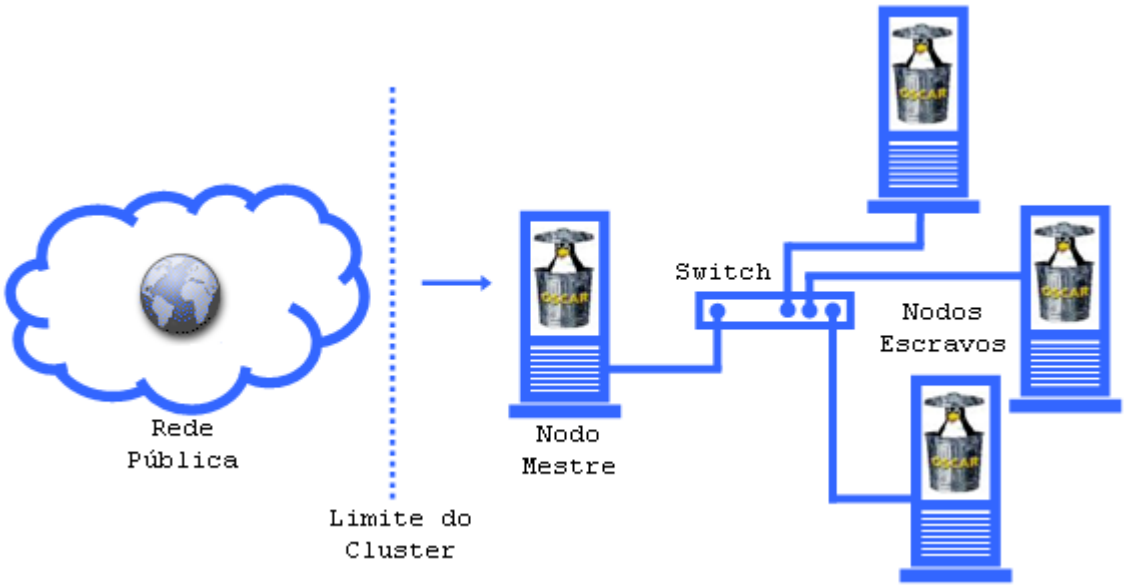

FIGURA 3 - Esquema de um *cluster* OSCAR.

A Figura 3 mostra o esquema de um *cluster* HPC implementado utilizando-se o pacote OSCAR. O nodo mestre é ligado aos nodos escravos através de um switch tipicamente *Fast Ethernet* (IEEE 802.3u), porém, outras tecnologias de rede de alta velocidade podem ser utilizadas, como Myrinet[9], por exemplo.

2.3. HA-OSCAR

HA-OSCAR, como o OSCAR, é um projeto *open source* que teve sua primeira versão beta publicada em março de 2004. Tem como objetivo prover a combinação entre Alta Disponibilidade e Computação de Alto Desempenho. Acrescenta a um *cluster* funcionalidades para lidar com aplicações de missão crítica, implementando várias técnicas de alta disponibilidade, como redundância de componente para eliminar o ponto único de falha, mecanismo de auto-reparo, mecanismo de detecção e recuperação de falhas, além de suportar *failover* e *failback* automáticos.

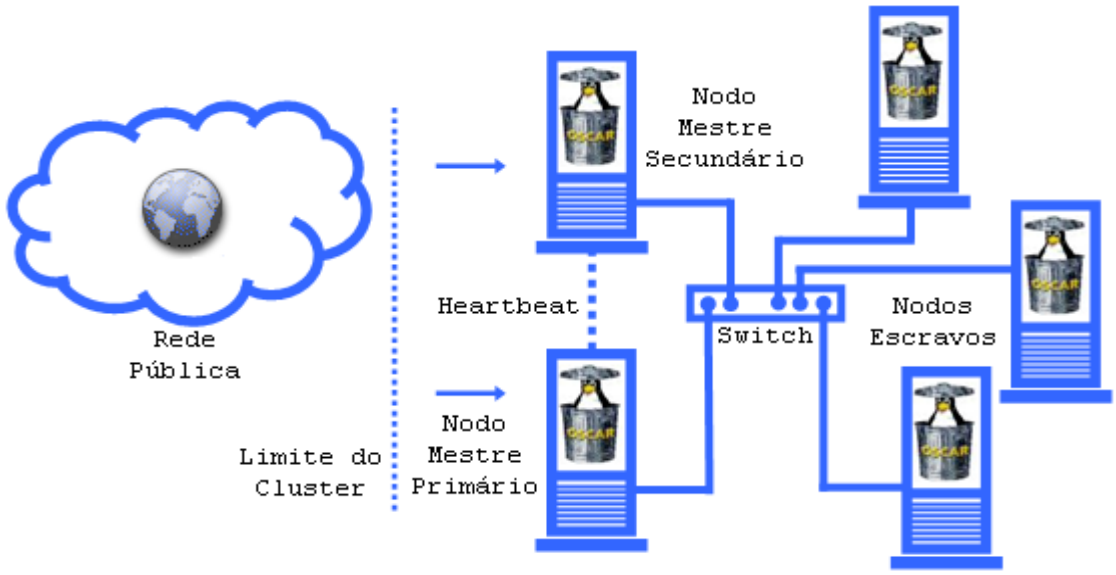

FIGURA 4 - Esquema de um *cluster* HA-OSCAR.

A Figura 4 mostra o esquema de um cluster implementado utilizando-se os pacotes OSCAR e HA-OSCAR, evidenciando o nodo mestre secundário e o canal de ligação entre os mestres primário e secundário, utilizado para monitoração pelo *Heartbeat*[10].

CAPÍTULO 3

3. Short Message Service

O Short Message Service (SMS) é o serviço que possibilita o envio e recebimento de mensagens curtas de e para telefones móveis. O texto das mensagens pode conter até 160 caracteres, podendo ser letras, números ou uma combinação de caracteres alfanuméricos, daí o termo “mensagens curtas”[6].

A funcionalidade do SMS é basicamente a mesma que a do serviço de pager, porém com uma vantagem fundamental: As mensagens SMS não necessitam que o telefone móvel esteja ligado ou dentro da área de cobertura para serem devidamente entregues. Isso deve-se ao fato de esse seviço funcionar na forma de “armazena e encaminha”. Em outras palavras, as mensagens não são enviadas diretamente da origem para o destino, mas através de uma Central SMS (SMSC)[6]. Falhas temporárias devido a indisponibilidade do dispositivo destino são identificadas e as mensagens armazenadas no SMSC até que o dispositivo destino torne-se disponível novamente ou o tempo de vida limite da mensagem seja atingido. Cada rede de telefonia móvel que suporte SMS, deve ter pelo menos um SMSC para gerenciar as mensagens.

3.1. SMS Gate

Para a realização deste trabalho foi utilizado o SMS Gate fabricado pela Taotronics Tecnologia [14], que consiste em uma plataforma de envio de mensagens de texto (SMS) via telefonia celular a partir de um microcomputador, incluindo hardware e software. As ferramentas de software são multiplataforma e permitem que qualquer aplicação utilize o hardware para enviar SMS de três maneiras diferentes: através de um servidor SMTP, bibliotecas de vínculo dinâmico e

utilitários de linha de comando.

Servidor SMTP (*Simple Mail Transfer Protocol*): Envia uma mensagem de texto via SMS. O servidor SMTP é compatível com qualquer aplicação de correio eletrônico que respeite este padrão, tais como: Outlook Express, Netscape Messenger, Mozilla Mail, KMail, Pine, etc.

Bibliotecas de vínculo dinâmico (DLL / SO): Envia mensagens SMS, diretamente ou via servidor SMTP. Permitem a integração do produto a qualquer aplicação.

Utilitários de linha de comando: Permitem o envio de mensagens SMS, diretamente ou via servidor SMTP, através do shell do sistema operacional;

Durante o desenvolvimento deste trabalho, não tivemos disponível esta ferramenta junto ao *cluster*, sendo necessário utilizar um SMS Gate que encontra-se instalado na rede privada da empresa fabricante, que gentilmente cedeu acesso através da rede pública para a realização deste projeto. Assim optou-se pela utilizaçao do servidor SMTP do SMS Gate para o envio dos alertas, tornando possível a abstração da localização física do mesmo. Esta abstração é possível pois o servidor SMTP será acessado pelo seu IP.

CAPÍTULO 4

4. Proposta e Resultados Experimentais

O presente trabalho se propôe a implementar uma ferramenta de monitoração e alerta remoto para o ambiente de um *cluster* HA-OSCAR. A necessidade de uma ferramenta capaz de monitorar o status de cada nodo do *cluster*, surge quando tem-se um *cluster* com um número elevado de nodos, onde a monitoração manual seria inviável, pois a freqüência das verificações executadas pelo responsável pelo *cluster* seria muito alta. Tal funcionalidade libera o responsável da monitoração direta dos nodos, porém não da análise dos resultados da monitoração. Assim, tão importante quanto a anterior, é a capacidade desta ferramenta alertar o responsável pelo *cluster* caso encontre alguma falha durante a monitoração.

Com uma ferramenta que possui tais características, o responsável pode assumir que o *cluster* está em seu estado consistente até que receba um alerta que diga o contrário, devendo assim executar o procedimento cabível para recuperar o *cluster* da falha que ocorreu. Isso reduz significativamente o tempo que o responsável pelo *cluster* dispenderia na tarefa de verificação/reparo do *cluster*, sendo necessária sua interação apenas quando houver de fato uma falha a ser corrigida.

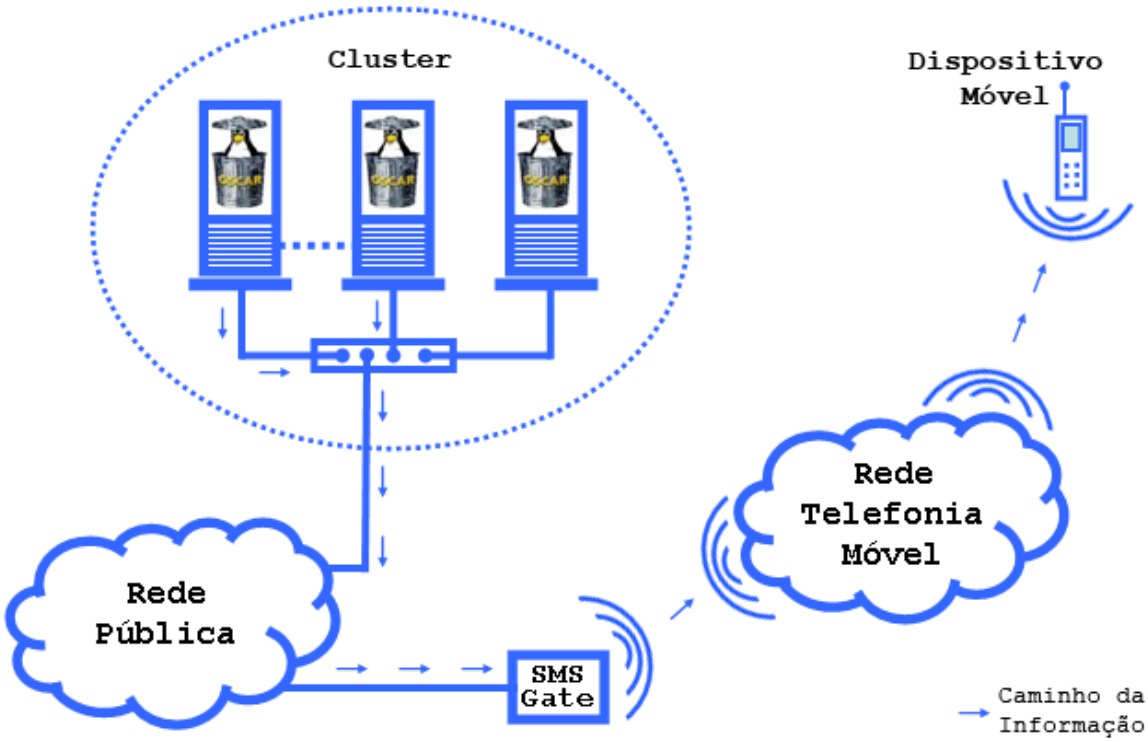

FIGURA 5 - Esquema geral e fluxo da informação na rede.

A Figura 5 mostra o esquema geral do funcionamento da ferramenta e o fluxo da informação. Este software estará executando em ambos os nodos mestres, porém estará fazendo a monitoração apenas a partir do nodo mestre que estiver no papel de primário, monitorando os nodos escravos e o mestre secundário. Ao ocorrer a detecção de uma falha, o software enviará a mensagem ao servidor SMTP do SMS Gate, acessível através da rede pública, que por sua vez fará a comunicação com o modem SMS, o qual tem a tarefa de encaminhar a mensagem para a rede de telefonia móvel que se encarrega de entregá-la ao destinatário. Maiores detalhes do funcionamento do software será visto na seção 4.2.

4.1. O Ambiente

O ambiente específico para o qual este trabalho foi desenvolvido e testado, consiste em um *cluster* localizado no Laboratório de Desenvolvimento Web (LabWeb), Departamento de Informática e Estatística (INE), Centro Tecnológico (CTC), Universidade Federal de Santa Catarina

(UFSC). Este *cluster* foi implementado, pelos autores de "Alta Disponibilidade – Um Estudo de Caso em um Ambiente de Imagem Única de Produção"[2], utilizando o pacote de software livre OSCAR versão 3.0 em conjunto com o HA-OSCAR versão BETA e é constituído de três computadores, sendo um nodo mestre, um nodo mestre secundário e um nodo escravo.

Para isso foram utilizadas três máquinas homogêneas, ou seja, com a mesma configuração de hardware, interconectadas através de um Switch *Fast Ethernet* formando uma rede privada. Cada computador possui um processador Intel Pentium 4 operando a 1800MHz, 512MB de memória principal e disco rígido com capacidade para armazenar 40GB. A distribuição de IPs para os nodos do *cluster* no escopo da rede privada foi feita conforme a Tabela 1, mostrada a seguir.

Tabela 1 - Distribuição de IPs na rede privada.

Nodo	**IP**	**Hostname**
Mestre Primário	192.169.1.1	master1.oscardomain
Mestre Secundário	192.169.1.2	master2.oscardomain
Escravo	192.169.1.3	slave1.oscardomain

A Figura 6 mostra o esquema do *cluster* que foi utilizado no desenvolvimento deste trabalho.

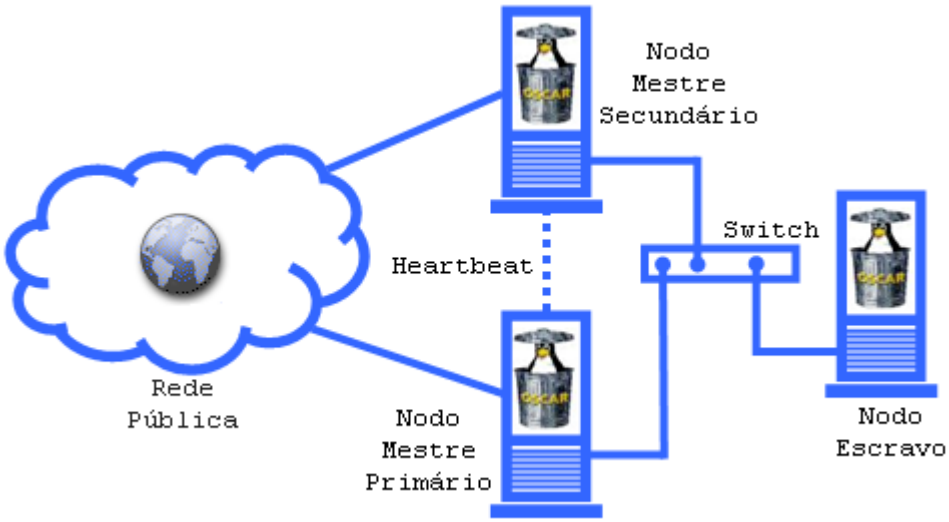

FIGURA 6 - Esquema do *cluster* utilizado no desenvolvimento

4.2. O Projeto

Para a implementação da ferramenta proposta foi utilizada a linguagem de programação Java v1.4.2[16], escolhida por uma série de razões, entre elas o fato de ser uma linguagem moderna, orientada a objetos e altamente portável, por possuir uma boa documentação online e pela disponibilidade de bibliotecas nativas para abstração de operações sobre rede. O ambiente de desenvolvimento utilizado foi o editor de texto open source Jext v5.0[17] e, para compilação e execução, as ferramentas de linha de comando *javac* e *java* providas pela Sun Microsystems[18] no pacote J2SE v1.4.2.

Foram definidas algumas estruturas como base para a implementação desta ferramenta. São elas: Nodo, Nodos e três listas auxiliares LNF, LNA e LNI. Outras estruturas colaterais foram definidas e serão abordadas mais adiante. A estrutura Nodos será referida como LN deste ponto em diante.

- **Nodo:** A estrutura Nodo foi definida para representar um nodo do *cluster*. Esta estrutura possui três atributos principais:
  - **IP:** Representa o endereço IP do nodo no escopo da rede privada.

- **ID:** Representa o identificador que foi definido para o nodo no arquivo de configuração. Caso não tenha sido definido será atribuído o último octedo do endereço IP ao ID.
- **CAE (Contador de Alertas Enviados):** Representa a quantidade de alertas que foram enviados informando que este nodo está em falha. Seu valor será entre 0 (zero) e o número máximo de alertas que devem ser enviados por falhas.

- **LN (Lista de Nodos):** Esta estrutura foi definida para conter todos os nodos do cluster. Para isso possui uma lista contendo todos os nodos escravos e um atributo representando o nodo mestre inativo. O conteúdo de seus atributos é lido do arquivo de configuração e permanece inalterado durante todo o tempo de execução da ferramenta. Cada elemento da lista de nodos escravos e o atributo nodo mestre inativo é definido pela estrutura Nodo.

- **LNF (Lista de Nodos em Falha):** Esta lista contém os nodos que foram identificados como estando em falha no momento em que são verificados. Assim que um alerta for enviado informando ao responsável pelo *cluster* quais nodos estão em falha, seus elementos são removidos e inseridos na LNA.

- **LNA (Lista de Nodos em Alerta):** Esta lista contém os nodos que estão em falha e devem ter seus alertas reenviados zero ou mais vezes. Todo elemento já teve seu primeiro alerta enviado antes de ser incluído nesta lista. Um elemento é removido da LNA em duas situações: caso saia do estado de falha, ficando funcional novamente ou o número de alertas enviados informando sua falha tenha atingido seu limite. No segundo caso o elemento removido é inserido na quarta lista: a LNI.

- **LNI (Lista de Nodos Ignorados):** Esta lista contém todos os nodos do *cluster* que já

tiveram seu número máximo de alertas enviados e continuam em estado de falha. Um elemento é removido da LNI somente quando seu estado passa a ser funcional novamente.

Sempre que um nodo é removido da LNA ou LNI seu atributo CAE é zerado e caso venha entrar em estado de falha novamente a quantidade de alertas que serão enviados será o número máximo definido no arquivo de configurações.

4.2.1. Arquivo de Configurações

Alguns valores que são usados durante a execução do software devem ser definidos pelo responsável pelo *cluster* como, por exemplo, o número de alertas e o tempo de espera. Para eliminar a necessidade de editar o código fonte e recompilar o software a cada alteração mínima que deseja-se fazer no comportamento da ferramenta, foi implementado um sistema de configuração através de arquivo de configurações em texto plano que pode ser editado em qualquer editor de texto comum. O formato do arquivo de configurações deve respeitar o seguinte padrão: "<nome do atributo>: <valor do atributo>" para declaração de atributos e apenas um por linha. Comentário é qualquer texto que venha após "//" na mesma linha. Todo texto contido no arquivo de configurações é sensível ao caso. A seguir serão enumerados os atributos que devem ser definidos no arquivo de configurações:

- **servidorSMTP:** Definição do endereço do servidor SMTP que será usado para enviar os alertas SMS. (ex. smtp.exemplo.com, 192.168.0.11)
  - Valores válidos: Seqüência de caracteres alfanuméricos (String)

- **portaSMTP:** Definição da porta que será utilizada para fazer a conexão ao servidor

SMTP. (ex. 25)

- Valores válidos: Valor inteiro no intervalo [-2^{15}, 2^{15}) (Short)

- **destinatario:** Definição do número do telefone móvel do destinatário das mensagens de alerta. A versão atual suporta apenas um destinatário. (ex. 04899144096)
  - Valores válidos: Seqüência de caracteres alfanuméricos (String)

- **remetente:** Definição do remetente que será informado nas mensagens de alerta. (ex. NodeWatch992@OSCAR)
  - Valores válidos: Seqüência de caracteres alfanuméricos (String)

- **portaConexao:** Definição da porta que será usada para conexão durante a verificação do estado de cada nodo. O responsável pelo *cluster* deve escolher uma porta que não estará aceitando conexões em nenhum nodo do *cluster*, maiores detalhes sobre o método de monitoração serão abordados na seção 4.2.4. (ex. 992)
  - Valores válidos: Valor inteiro no intervalo [-2^{15}, 2^{15}) (Short)

- **timeout:** Definição do tempo máximo de espera em milissegundos por uma conexão na porta definida pelo atributo portaConexao do nodo que está sendo verificado. (ex. 5000)
  - Valores válidos: Valor inteiro no intervalo [-2^{15}, 2^{15}) (Short)

- **tempoWait:** Definição do tempo em milissegundos que a ferramenta deverá esperar entre uma verificação e outra. (ex. 15000)
  - Valores válidos: Valor inteiro no intervalo [-2^{63}, 2^{63}) (Long)

- **tempoReenvio:** Definição do tempo em minutos que a ferramenta deverá esperar antes

de reenviar um alerta. (ex. 15)

- Valores válidos: Valor inteiro no intervalo $[-2^{15}, 2^{15})$ (Short)

- **mAF (Máximo de Alertas por Falha):** Definição do número máximo de alertas que a ferramenta deverá enviar por falha. (ex. 3)
  - Valores válidos: Valor inteiro no intervalo $[-2^{7}, 2^{7})$ (Byte)

- **NIC (Network Interface Card):** Definição da Interface de Rede a tomar como parâmetro ao assumir o papel de nodo Mestre Primário. O critério tomado para assumir papel de mestre primário será abordado com mais detalhes na seção 4.2.3. (ex. eth0)
  - Valores válidos: Seqüência de caracteres alfanuméricos (String)

- **pastaLog:** Definição do caminho no sistema de arquivos onde deverão ser armazenados os arquivos de log gerados pelo software.
  (ex. /var/log/NodeWatch992)
  - Valores válidos: Seqüência de caracteres alfanuméricos (String)

- **nomeArquivoLog:** Definição do nome do arquivo de log e do formato dos arquivos de log salvos. Maiores detalhes sobre a geração de logs serão abordados na seção 4.2.2. (ex. monitor.log)
  - Valores válidos: Seqüência de caracteres alfanuméricos (String)

- **tamArquivoLog:** Definição do tamanho máximo, em bytes, que o arquivo de log pode atingir. (ex. 5242880 => 5MB)
  - Valores válidos: Valor inteiro no intervalo $[-2^{31}, 2^{31})$ (int)

- **mestreInativo[IP]:** Definição do endereço IP do nodo mestre inativo. (ex. 192.168.0.1)

- Valores válidos: Seqüência de caracteres alfanuméricos (String)

- **mestreInativo[ID]:** Definição do identificador do nodo mestre inativo. A definição deste atributo é opcional. (ex. M01)
  - Valores válidos: Seqüência de caracteres alfanuméricos (String)

- **escravo[0..n][IP]:** Definição do endereço IP do nodo escravo *n*. (ex. 192.168.0.101)
  - Valores válidos: Seqüência de caracteres alfanuméricos (String)

- **escravo[0..n][ID]:** Definição do identificador do nodo escravo *n*. A definição deste atributo é opcional. (ex. E01)
  - Valores válidos: Seqüência de caracteres alfanuméricos (String)

Na definição dos atributos no arquivo de configurações deve-se observar que os valores válidos indicados referem-se ao tipo de dado do atributo e não a um valor semanticamente correto. Valores atribuídos de forma incorreta poderão causar erros imprevistos durante a execução do software, por exemplo a definição de uma porta fora da faixa de portas válidas do sistema operacional.

4.2.2. Arquivo de Log

A interface de saída escolhida para a ferramenta implementada é um arquivo de log, que registra cada detalhe importante que ocorre durante sua execução. Este registro é importante para armazenar os momentos em que ocorrem uma falha ou a recuperação de um estado de falha. Com estes dados disponíveis, o administrador do *cluster* ou mesmo uma ferramenta externa, pode gerar gráficos e tabelas estatísticas que refletirão o comportamento do cluster. Dados estatísticos são

importantes para auxiliar na tomada de decisões pelo responsável pelo *cluster*.

Com isso em mente foi implementado um sistema de log que não descarta nenhum dado antigo, apesar de existir um limitador do tamanho do arquivo de log, definido pelo atributo *tamArquivoLog* no arquivo de configurações. Caso a quantidade de dados contida no arquivo de log ultrapassar seu limite máximo, o arquivo atual é renomeado para *nomeArquivoLog.n*, onde *n* é o menor inteiro no intervalo [1,∞), sendo que um arquivo com este nome ainda não tenha sido criado. Ficando assim com, além do arquivo de log atual, *n* arquivos de log salvos e ordenados do mais antigo para o mais recente.

Os eventos que serão registrados em log são descritos a seguir:

- **Início da execução:** É registrado o momento em que o software iniciou sua execução. Com este dado pode-se ter controle do momento em que o nodo mestre inicializou o software e quantas vezes isso foi feito.

  # Thu Nov 04 19:07:25 BRST 2004 - Iniciando execução...

- **Assumindo papel de Mestre Primário:** É registrado o momento em que o nodo mestre em que o software está sendo executado assume o papel de Mestre Primário. Com este dado tem-se o controle das trocas de papel entre mestre primário e secundário, além de qual nodo mestre está no papel de primário atualmente.

  # Thu Nov 04 19:07:30 BRST 2004 - Assumindo posição de Nodo Mestre Primário...

- **Assumindo papel de Mestre Secundário:** É registrado o momento em que o nodo mestre em que o software está sendo executado assume o papel de Mestre Secundário. Com este dado tem-se o controle do momento e da quantidade de vezes que um nodo mestre tornou-se secundário.

  # Thu Nov 04 19:21:12 BRST 2004 - Assumindo posição de Nodo Mestre Secundário...

- **Transição Funcional/Falha detectada:** É registrado o momento em que um nodo entra

em estado de falha. Com este dado pode-se manter um controle sobre o número de falhas de cada nodo e seus respectivos momentos.

Thu Nov 04 19:21:27 BRST 2004 - Nodo "E01" OFFLINE! :-(

- **Transição Falha/Funcional detectada:** É registrado o momento em que um nodo recupera-se de um estado de falha tornando-se funcional novamente. Com este dado pode-se manter um controle sobre o tempo necessário para que um nodo volte a ficar funcional.

Thu Nov 04 19:47:38 BRST 2004 - Nodo "E01" ONLINE! :-)

- **Nodo sendo ignorado:** É registrado o momento em que um nodo passa a ser ignorado pelo software. Um nodo ignorado significa que está em falha e não será mais enviado alertas sobre seu estado ao responsável pelo *cluster*. Com este dado pode-se fazer uma relação entre os nodos que entram em falha e os que chegam a ser ignorados.

Thu Nov 04 19:42:55 BRST 2004 - Nodo "E01" IGNORADO! :-P

- **Alerta sendo enviado:** É registrado a mensagem e o momento em que um alerta é enviado através do SMS Gate para o responsável pelo *cluster*. Com este dado pode-se manter um controle das mensagens que foram enviadas.

Thu Nov 04 19:21:27 BRST 2004 - Enviando alerta: --- FALHA DETECTADA --- Nodos em falha (1): 1: E01

- **Alerta sendo reenviado:** É registrado a mensagem e o momento em que um alerta é reenviado através do SMS Gate para o responsável pelo *cluster*. Este reenvio é feito quando um nodo permanece em falha por um período de tempo que é determinado no arquivo de configurações. Com este dado pode-se manter um controle dos nodos que permaneceram em falha tempo suficiente para ter seus alertas reenviados.

Thu Nov 04 19:38:03 BRST 2004 - Reenviando alerta: --- REAVISO --- Nodos em falha (1): 1: E01

4.2.3. Máquina de Estados Finitos

A máquina de estados finitos mostrada na Figura 7 representa o funcionamento do software desenvolvido.

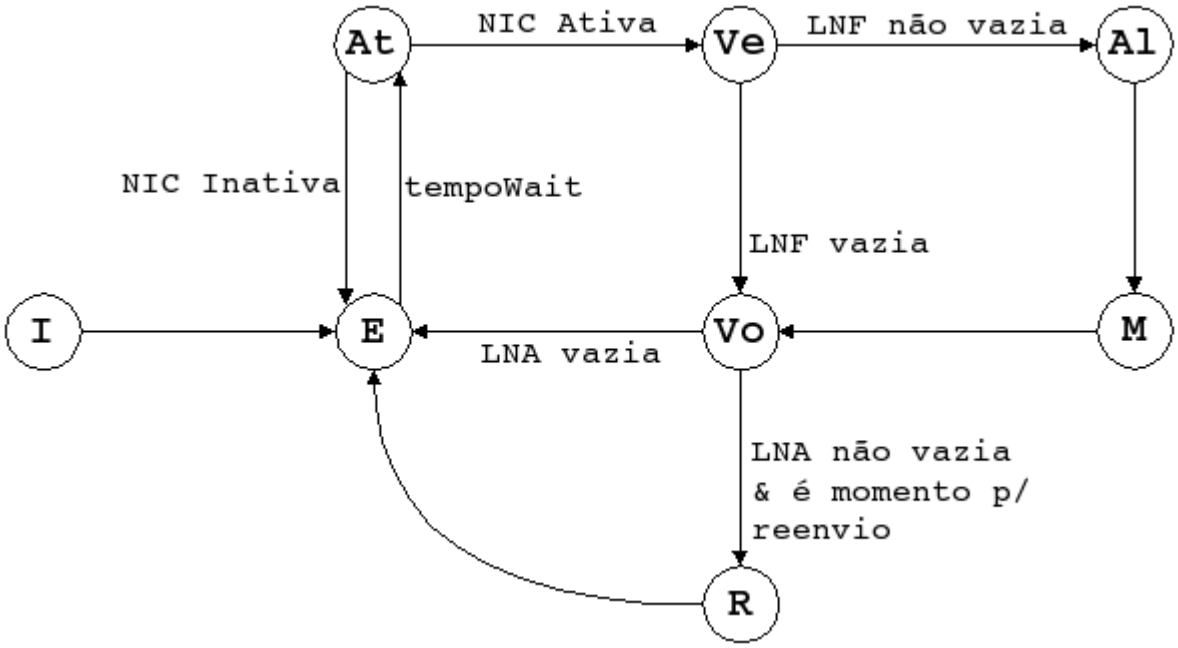

FIGURA 7 - Máquina de Estados Finitos

Estado INICIALIZA (I): É o estado inicial do software, no qual serão carregados os valores dos atributos definidos através do arquivo de configurações. Este estado é executado apenas uma vez na inicialização do software e avança para o estado ESPERA.

Estado ESPERA (E): Este estado identifica o primeiro estágio dos ciclos de execução do software. Após o período de tempo definido pelo atributo *tempoWait* ter

decorrido, o software entra no estado ATIVA.

Estado ATIVA (At): Neste estado o software verifica se o sistema no qual está executando está na posição de mestre primário ou secundário. Isto é feito verificando o estado da interface de rede definida pelo atributo NIC no arquivo de configurações, caso a interface esteja ativa o software assume que é mestre primário, caso contrário assume que é mestre secundário. Esta verificação é feita através do utilitário de linha de comando *netstat*[19], presente na maioria das distribuições Linux atuais. Caso o resultado da verificação indique que este sistema é mestre primário, o software avança para o estado VERIFICA, caso contrário retorna ao estado ESPERA. Ainda, caso seja detectado que ocorreu uma passagem de mestre primário para mestre secundário, as listas LNF, LNA e LNI serão tornadas vazias.

Estado VERIFICA (Ve): É neste estado que o software irá verificar o status de cada nodo pertencente ao cluster. Caso encontre algum nodo em falha, verificará se este está na LNI ou na LNA, caso não esteja este será adicionado à LNF e continuará a verificação. Para cada nodo que não esteja em falha, verificará se este está na LNI ou na LNA, caso esteja será removido. Após terminada a verificação de todos os nodos, caso a LNF esteja vazia, avança ao estado VOLTA, caso contrário, avança para o estado ALERTA.

Estado ALERTA (Al): Quando o software entrar neste estado, enviará um alerta por SMS para o responsável pelo *cluster* informando a quantidade total de nodos em falha e um identificador para cada um deles. Este identificador poderá ser definido na configuração do software, caso não seja definido será assumido como identificador o último octeto do IP do nodo. Por exemplo, se o IP do nodo for 192.168.0.101 o identificador será 101 se outro não tiver sido definido no arquivo de

configuração. Caso o alerta que esteja sendo enviado for o primeiro alerta desde a inicialização do software, sua data será armazenada com precisão de milissegundos. Após ter enviado o alerta, entrará no estado MARCA.

Estado MARCA (M): Neste estado os elementos que estão na LNF serão transferidos para a LNA, avançando para o estado VOLTA.

Estado VOLTA (Vo): Neste estado é verificado o conteúdo da LNA, caso esteja vazia avança para o estado ESPERA, caso contrário outras verificações serão feitas. A data atual é verificada, com precisão de milissegundos. Desta data é subtraída a data do primeiro alerta enviado, armazenada no estado ALERTA, daí teremos a quantidade de milissegundos decorridos entre o primeiro alerta e o momento atual. Se este intervalo de tempo for maior que um minuto, então ele é convertido em minutos e arredondado para baixo, é verificado se este valor é múltiplo do valor do atributo *tempoReenvio*, definido no arquivo de configurações. Este cálculo é representado matematicamente pela seguinte fórmula:

$$\mathbf{floor((m - n) / 60000) \bmod tr}$$

Onde: **m** representa o momento do primeiro alerta em milissegundos;

n representa o momento atual em milissegundos;

tr representa o valor do atributo *tempoReenvio*;

floor representa a função de arredondamento para baixo;

mod representa a função de resto da divisão.

Caso o resultado deste cálculo seja 0 (zero), significa que o intervalo é múltiplo de

tempoReenvio, então o software avança para o estado REENVIA. Caso contrário avança para o estado ESPERA.

Estado REENVIA (R): Neste estado será enviado um alerta SMS informando o número de nodos que estão na LNA e um identificador para cada um deles, satisfazendo as seguintes restrições: Caso o atributo CAE de um elemento seja 0 (zero) ele será incrementado e o nodo não será incluído no alerta. Caso o atributo CAE de um elemento seja maior que 0 (zero) e menor que o atributo *mAF* definido no arquivo de configurações, o contador será incrementado e o nodo será incluído no alerta. Caso o atributo CAE de um elemento seja maior que *mAF*, este será excluído da LNA e incluído na LNI, e o nodo não será incluído no alerta. Após enviado o alerta o software retornará para o estado ESPERA.

4.2.4 Método de Monitoração

Para verificar o estado de um nodo em particular é necessário que seja estabelecida uma comunicação entre o nodo mestre primário (monitorador) e o nodo em questão (monitorado). A partir desta comunicação o monitorador deve chegar a uma conclusão sobre o estado do monitorado: em falha ou funcional. Durante o desenvolvimento desta ferramenta três possíveis soluções foram supostas:

- **Ping:** Uma possível solução é utilizar a ferramenta de linha de comando *ping* para enviar datagramas ICMP ao nodo monitorado, o resultado obtido indica o tempo de resposta e a quantidade de pacotes perdidos. Isso é viável pois esta ferramenta é disponível na maioria dos sistemas operacionais atuais. Outro fato que levou a suposição desta solução é a falta de uma implementação nativa do protocolo ICMP na API da linguagem Java v1.4.2. Esta

solução foi descartada por depender do resultado de uma ferramenta externa para tomar uma decisão vital para a execução do software.

- **Mini aplicação Cliente/Servidor:** Outra possível solução é a implementação de uma aplicação de servidor que aguarda conexões em determinada porta. Tal aplicação seria executada em cada nodo monitorado e ao receber uma conexão do monitorador enviaria como resposta um sinal de que está em estado funcional. A decisão por parte do monitorador seria feita analisando a resposta recebida do nodo, caso nenhuma resposta fosse recebida em um determinado período de tempo, seria assumido que o monitorado está em estado de falha. Esta solução foi descartada por consumir recursos dos nodos escravos constantemente, ao ficar esperando uma conexão.
- **Recusa de conexão:** A solução que foi adotada é baseada na recusa de conexão pelo nodo monitorado. O monitorador tenta criar uma conexão com o monitorado em uma porta fechada, ou seja, que não esteja aguardando conexões. Caso o monitorador obtenha como resposta "Conexão Negada", significa que o nodo está em estado funcional, pois pôde tratar a tentativa de conexão do monitorador e responder de modo adequado. Partindo do pressuposto que a porta na qual está sendo tentada a conexão está fechada, qualquer outra resposta obtida, como "Conexão Expirou", é considerada como indicador de falha no nodo monitorado. Esta porta é definida no arquivo de configurações pelo responsável pelo *cluster*, que deve ter conhecimento prévio sobre o estado da porta em todos os nodos do *cluster*.

4.3. Estudos de Caso

Alguns estudos de casos foram realizados para verificar na prática as funcionalidades propostas pela ferramenta implementada. Nos estudos de casos realizados os atributos de

configuração mais relevantes foram definidos conforme os dados na Tabela 2.

TABELA 2 – Atributos definidos nos estudos de caso.

Atributo	Valor
timeout	5000
tempoWait	3000
tempoReenvio	3
mAF	3
NIC	eth0

Inicialmente o nodo mestre que estava no papel de primário foi o M01 (IP: 192.169.1.1). O registro do início da execução do software é mostrado a seguir:

Fri Nov 05 14:57:03 BRST 2004 - Iniciando execução...

Fri Nov 05 14:57:06 BRST 2004 - Assumindo posição de Nodo Mestre Primário...

4.3.1. Estudo de Caso 1

Neste Estudo de Caso foi verificado o comportamento da ferramenta ao detectar uma falha no nodo escravo E01. O fluxo de ações executadas ocorreu como a seguir:

Desligamento do nodo E01: O nodo foi desconectado da rede propositalmente para simular uma falha. O registro da detecção da falha e do alerta enviado são mostrados a seguir:

Fri Nov 05 15:00:40 BRST 2004 - Nodo "E01" OFFLINE! :-(

Fri Nov 05 15:00:40 BRST 2004 - Enviando alerta: --- FALHA DETECTADA --- Nodos em falha (1): 1: E01

Religamento do nodo E01: Assim que a mensagem SMS foi recebida, o nodo foi

reconectado a rede simulando uma recuperação do estado de falha, retornando ao estado consistente do *cluster*. O registro da recuperação da falha é mostrado a seguir:

Fri Nov 05 15:01:34 BRST 2004 - Nodo "E01" ONLINE! :-)

4.3.2. Estudo de Caso 2

Neste Estudo de Caso foi verificado o comportamento da ferramenta ao detectar uma falha no nodo mestre secundário M02. O fluxo de ações executadas ocorreu como a seguir:

Desligamento do nodo M02: O nodo foi desconectado da rede propositalmente para simular uma falha. O registro da detecção da falha e do alerta enviado são mostrados a seguir:

Fri Nov 05 15:04:04 BRST 2004 - Nodo "M02" OFFLINE! :-(

Fri Nov 05 15:04:05 BRST 2004 - Enviando alerta: --- FALHA DETECTADA --- Nodos em falha (1): 1: M02

Religamento do nodo M02: Assim que a mensagem SMS foi recebida, o nodo foi reconectado a rede simulando uma recuperação do estado de falha, retornando ao estado consistente do *cluster*. O registro da recuperação da falha é mostrado a seguir:

Fri Nov 05 15:05:52 BRST 2004 - Nodo "M02" ONLINE! :-)

4.3.3. Estudo de Caso 3

Neste Estudo de Caso foi verificado o comportamento da ferramenta ao detectar falhas simultâneas nos nodos mestre secundário M02 e escravo E01. O fluxo de ações executadas ocorreu como a seguir:

Desligamento dos nodos M02 e E01: Os nodos foram desconectados da rede propositalmente para simular uma falha. O registro da detecção das falhas e do alerta enviado são mostrados a seguir:

Fri Nov 05 15:12:16 BRST 2004 - Nodo "M02" OFFLINE! :-(

Fri Nov 05 15:12:19 BRST 2004 - Nodo "E01" OFFLINE! :-(

Fri Nov 05 15:12:19 BRST 2004 - Enviando alerta: --- FALHA DETECTADA --- Nodos em falha (2): 1: M02 2: E01

Religamento do nodo E01: Assim que a mensagem SMS foi recebida, o nodo E01 foi reconectado a rede simulando uma recuperação do estado de falha do nodo, entretanto o estado do *cluster* continua inconsistente devido a falha no nodo M02 permanecer. O registro da recuperação da falha no nodo E01 é mostrado a seguir:

Fri Nov 05 15:13:01 BRST 2004 - Nodo "E01" ONLINE! :-)

Reenvio do alerta sobre a falha no nodo M02: Como o estado de falha no nodo M02 permaneceu por tempo suficiente para ter seu número máximo de alertas enviados e ser ignorado, foram enviados mais dois alertas sobre seu estado antes de ser ignorado. O registro dos alertas reenviados e do nodo sendo ignorado é mostrado a seguir:

Fri Nov 05 15:15:25 BRST 2004 - Reenviando alerta: --- REAVISO --- Nodos em falha (1): 1: M02

Fri Nov 05 15:18:22 BRST 2004 - Reenviando alerta: --- REAVISO --- Nodos em falha (1): 1: M02

Fri Nov 05 15:21:22 BRST 2004 - Nodo "M02" IGNORADO! :-P

Religamento do nodo M02: Logo que o nodo M02 passou a ser ignorado pela ferramenta, o mesmo foi reconectado a rede simulando uma recuperação do estado de falha do nodo, retornando ao estado consistente do *cluster*. O registro da recuperação da falha no nodo M02 é mostrado a seguir:

Fri Nov 05 15:22:49 BRST 2004 - Nodo "M02" ONLINE! :-)

4.3.4. Estudo de Caso 4

Neste Estudo de Caso foi verificado o comportamento da ferramenta ao detectar uma falha no nodo mestre primário M01. O fluxo de ações executadas ocorreu como a seguir:

Desligamento do nodo M01: O nodo foi desconectado da rede propositalmente para simular uma falha. Ao ocorrer uma falha no nodo mestre primário o HA-OSCAR é responsável pela substituição do mesmo pelo mestre secundário, ativando as interfaces de rede necessárias para comunicação com a rede privada e pública. Ao detectar a ativação da interface de rede definida pelo atributo NIC do arquivo de configurações, a ferramenta assume que o nodo no qual está sendo executada assmiu o papel de mestre primário. O registro do momento em que é assumido o papel de mestre primário, da detecção da falha no mestre secundário (primário que entrou em falha) e do alerta enviado são mostrados a seguir:

Fri Nov 05 14:56:21 BRST 2004 - Iniciando execução...

Fri Nov 05 15:25:15 BRST 2004 - Assumindo posição de Nodo Mestre Primário...

Fri Nov 05 15:25:19 BRST 2004 - Nodo "M01" OFFLINE! :-(

Fri Nov 05 15:25:20 BRST 2004 - Enviando alerta: --- FALHA DETECTADA --- Nodos em falha (1): 1: M01

Religamento do nodo M01: Assim que a mensagem SMS foi recebida, o nodo foi reconectado a rede simulando uma recuperação do estado de falha, retornando ao estado consistente do *cluster*. O registro da recuperação da falha é mostrado a seguir:

Fri Nov 05 15:26:02 BRST 2004 - Nodo "M01" ONLINE! :-)

4.3.5. Arquivo de log completo

Aqui é exibido o arquivo de log completo dos estudos de caso realizados:

Arquivo de log criado pelo nodo M01:

Fri Nov 05 14:57:03 BRST 2004 - Iniciando execução...

Fri Nov 05 14:57:06 BRST 2004 - Assumindo posição de Nodo Mestre Primário...

Fri Nov 05 15:00:40 BRST 2004 - Nodo "E01" OFFLINE! :-(

Fri Nov 05 15:00:40 BRST 2004 - Enviando alerta: --- FALHA DETECTADA --- Nodos em falha (1): 1: E01

Fri Nov 05 15:01:34 BRST 2004 - Nodo "E01" ONLINE! :-)

Fri Nov 05 15:04:04 BRST 2004 - Nodo "M02" OFFLINE! :-(

Fri Nov 05 15:04:05 BRST 2004 - Enviando alerta: --- FALHA DETECTADA --- Nodos em falha (1): 1: M02

Fri Nov 05 15:05:52 BRST 2004 - Nodo "M02" ONLINE! :-)

Fri Nov 05 15:12:16 BRST 2004 - Nodo "M02" OFFLINE! :-(

Fri Nov 05 15:12:19 BRST 2004 - Nodo "E01" OFFLINE! :-(

Fri Nov 05 15:12:19 BRST 2004 - Enviando alerta: --- FALHA DETECTADA --- Nodos em falha (2): 1: M02 2: E01

Fri Nov 05 15:13:01 BRST 2004 - Nodo "E01" ONLINE! :-)

Fri Nov 05 15:15:25 BRST 2004 - Reenviando alerta: --- REAVISO --- Nodos em falha (1): 1: M02

Fri Nov 05 15:18:22 BRST 2004 - Reenviando alerta: --- REAVISO --- Nodos em falha (1): 1: M02

Fri Nov 05 15:21:22 BRST 2004 - Nodo "M02" IGNORADO! :-P

Fri Nov 05 15:22:49 BRST 2004 - Nodo "M02" ONLINE! :-)

Arquivo de log criado pelo nodo M02:

Fri Nov 05 14:56:21 BRST 2004 - Iniciando execução...

Fri Nov 05 15:25:15 BRST 2004 - Assumindo posição de Nodo Mestre Primário...

Fri Nov 05 15:25:19 BRST 2004 - Nodo "M01" OFFLINE! :-(

Fri Nov 05 15:25:20 BRST 2004 - Enviando alerta: --- FALHA DETECTADA --- Nodos em falha (1): 1: M01

Fri Nov 05 15:26:02 BRST 2004 - Nodo "M01" ONLINE! :-)

CAPÍTULO 5

5. Conclusões e Trabalhos Futuros

A tarefa de administrar um *cluster* com muitos nodos está longe de ser trivial, e sem uma ferramenta de auxílio demandaria atenção contínua de uma equipe responsável pelo *cluster*. Com uma ferramenta ficando responsável pela tarefa de monitorar cada nodo do *cluster* e enviar um alerta caso ocorra alguma falha, o responsável apenas necessitaria intervir caso ocorresse uma falha, para corrigí-la.

Nos testes realizados a ferramenta implementada apresentou bons resultados, mostrando as funcinalidades que foram propostas: detectando os nodos que entraram em estado de falha e alertando o responsável pelo *cluster* via SMS em tempo aceitável.

Como trabalho futuro seria interessante a utilização de um dispositivo de envio de mensagens SMS para a rede de telefonia móvel conectado localmente em cada nodo mestre, de forma a garantir que o alerta será enviado independendo das condições da rede pública.

6. Referências

[1] R. M. Alvin, F. L. L. Grossmann, "**Implementação de uma Arquitetura para Serviço Distribuído com Grande Disponibilidade em Ambiente Linux**."
Disponível em <http://www.sbc.org.br/reic/edicoes/2002e3/cientificos>. Acesso em: 04 dez. 2003

[2] S. L. Moser, C. Rista, M. A. R. Dantas, "**Alta Disponibilidade – Um Estudo de Caso em um Ambiente de Imagem Única de Produção**", 2004

[3] **OSCAR – Open Source Cluster Application Resources.** Disponível em <http://oscar.sourceforge.net>. Acesso em: 12 abr. 2004

[4] **HA-OSCAR – High Availability Open Source Cluster Application Resources**. Disponível em <http://www.cenit.latech.edu/oscar/>. Acesso em 12 abr. 2004

[5] **IEEE Task Force on Cluster Computing**. Disponível em <http://www.ieeetfcc.org/>. Acesso em 24 set. 2004

[6] **GSM Favorites.** Disponível em <http://www.gsmfavorites.com/sms/>. Acesso em 27 set. 2004

[7] **Open Cluster Group.** Disponível em <http://www.openclustergroup.org/>. Acesso em 10 out. 2004

[8] B. des Ligneris, S. Scott, T. N. N. Gorsuch, "**Open Source Application Resources (OSCAR): design, implementation and interest for the [computer] scientific community"**, 2003

[9] **Myricom.** Disponível em <http://www.myri.com/myrinet/overview/index.html>. Acesso em 16 out. 2004.

[10] **High-Availability Linux Project.** Disponível em <http://linux-ha.org/>. Acesso em 16 out. 2004.

[11] ETSI GTS 04.11 V3.3.0 (1995-01), **"European digital cellular telecommunications system (Phase 1);Point-to-point Short Message Service;Support on Mobile Radio Interface (GSM 04.11)".**

[12] ETSI GTS GSM 04.11 V5.0.0 (1996-02)**, "Digital cellular telecommunications system (Phase 2+) (GSM);Point-to-Point (PP) Short Message Service (SMS) support on mobile radio**

interface (GSM 04.11)".

[13] ETSI GTS GSM 04.11 V5.1.0 (1996-03), **"Digital cellular telecommunications system (Phase 2+) (GSM);Point-to-Point (PP) Short Message Service (SMS) support on mobile radio interface (GSM 04.11)".**

[14] **Taotronics Tecnologia.** Disponível em <http://www.taotronics.com.br/>. Acesso em 18 out. 2004.

[15] **RFC 2821 – Simple Mail Transfer Protocol (SMTP).** Disponível em <http://www.faqs.org/rfcs/rfc2821.html>. Acesso em 24 out. 2004.

[16] **JavaTM 2 Platform, Standard Edition, v 1.4.2 API Specification.** Disponível em <http://java.sun.com/j2se/1.4.2/docs/api/index.html>. Acesso em 18 out. 2004.

[17] **Jext 5.0.** Disponível em <http://www.jext.org>. Acesso em 11 set. 2004.

[18] **Sun Microsystems.** Disponível em <http://www.sun.com/>. Acesso em 18 out. 2004.

[19] Philip Blundell, "N**et-Tools v 1.60"**. Disponível em <http://freshmeat.net/projects/net-tools/>. Acesso em 02 nov. 2004.

ANEXOS

ANEXO 1 – CÓDIGO FONTE

Classe NodeWatch992.java

```
import java.util.*;
import java.io.*;

public class NodeWatch992 extends Thread {

   private Date momentoPrimeiroAlerta;

   // proximo estado da maq de estados a ser executado
   private byte proximoEstado = 0;

   // atributo indicando se este nodo é o mestre primário
   // ou o mestre secundário
   private boolean souPrimario = false;

   // cria o objeto que fará o log do sistema
   private Logger log = new Logger();
   // cria o objeto que fará a verificacao dos nodos
   private Verificador verificador = new Verificador();
   private Alertador alertador = new Alertador();
   private Configurador conf;

   // Lista de Nodos
   private Nodos LN = new Nodos();
   // Lista de Nodos em Falha
   private Vector LNF = new Vector();
   // Lista de Nodos em Alerta
   private Vector LNA = new Vector();
   // Lista de Nodos Ignorados
   private Vector LNI = new Vector();


   public NodeWatch992() {
   }

   public static void main(String args[]) {
      NodeWatch992 nodeWatch992 = new NodeWatch992();
      nodeWatch992.run();
   }

   public void run() {
      while (true) {
         this.proximoEstado();
      }
   }
```

```
private void proximoEstado() {
   switch (this.proximoEstado) {
   case 0:
      this.inicializa();
      break;
   case 1:
      this.espera();
      break;
   case 2:
      this.ativa();
      break;
   case 3:
      this.verifica();
      break;
   case 4:
      this.volta();
      break;
   case 5:
      this.reenvia();
      break;
   case 6:
      this.alerta();
      break;
   case 7:
      this.marca();
      break;
   }
}

// implementação do estado INICIALIZA da maq de estados finitos (0)
private void inicializa() {
   this.conf = Configurador.singleton();
   this.conf.carreagaDoArquivo("NodeWatch992.conf");
   this.log.registra("Iniciando execução...");

   this.LN = this.conf.getNodos();

   this.proximoEstado = 1; // ESPERA
}

// implementação do estado ESPERA da maq de estados finitos (1)
private void espera() {
   try {
      sleep(this.conf.getTempoWait());
      this.proximoEstado = 2; // ATIVA
   } catch (Exception e) {
      e.printStackTrace();
   }
}

// implementação do estado ATIVA da maq de estados finitos (2)
```

```
private void ativa() {
   try {
      boolean achou = false;
      Process p = Runtime.getRuntime().exec("netstat -i");
      BufferedReader resultado = new BufferedReader(
            new InputStreamReader(p.getInputStream()));

      String line;
      while ((line = resultado.readLine()) != null) {
         if (line.indexOf(this.conf.getNIC()) > -1) {
            achou = true;
         }
      }
      resultado.close();
      if (achou) {
         if (!this.souPrimario) {
            this.log.registra("Assumindo posição de Nodo Mestre Primário...");
         }
         this.souPrimario = true;
         this.proximoEstado = 3; // VERIFICA
      } else {
         if (this.souPrimario) {
            this.log.registra("Assumindo posição de Nodo Mestre Secundário...");
            this.limpaListas();
         }
         this.souPrimario = false;
         this.proximoEstado = 1; // ESPERA
      }
   } catch (Exception e) {
      this.log.registra("Impossível verificar estado da Interface de Rede!");
   }
}

private void limpaListas() {
   this.LNF.clear();
   this.LNA.clear();
   this.LNI.clear();
}

// implementação do estado VERIFICA da maq de estados finitos (3)
private void verifica() {
   // verificando o mestre inativo
   this.verificaNodo(this.LN.getMestreInativo());

   // verificando os escravos
   int numEscravos = this.LN.getNumEscravos();
   for (int i = 0; i < numEscravos; i++) {
      this.verificaNodo((Nodo) this.LN.getEscravo(i));
   }
   // se nenhum nodo foi adicionado a LNF então tudo OK!
   if (this.LNF.isEmpty()) {
      this.proximoEstado = 4; // VOLTA
```

```
    } else {
        this.proximoEstado = 6; // ALERTA
    }
}

private void verificaNodo(Nodo n) {
    if (!this.verificador.verificaNodo(n)) {
        if (!this.LNI.contains(n) && !this.LNA.contains(n)) {
            this.LNF.add(n);
            this.log.registra(this.msgNodoMorto(n.getID()));
        }
    } else {
        if (this.LNI.remove(n) || this.LNA.remove(n)) {
            n.resetCAE();
            this.log.registra(this.msgNodoOK(n.getID()));
        }
    }
}

// implementação do estado VOLTA da maq de estados finitos (4)
private void volta() {
    if (this.LNA.isEmpty()) {
        this.proximoEstado = 1; // ESPERA
        return;
    }
    if (this.devoReenviar()) {
        this.proximoEstado = 5; // REENVIA
    } else {
        this.proximoEstado = 1; // ESPERA
    }
}

private boolean devoReenviar() {
    // Se a LNA estiver vazia não tem alerta para reenviar
    if (this.LNA.size() == 0) {
        return false;
    }
    long diferenca = (new Date()).getTime() -
            this.momentoPrimeiroAlerta.getTime();
    // Se a diferenca é maior que 1 minuto...
    if (diferenca >= 60000) {
        if ((Math.floor(diferenca /
                60000) % this.conf.getTempoReenvio()) == 0) {
            return true;
        }
    }
    return false;
}

// implementação do estado REENVIA da maq de estados finitos (5)
private void reenvia() {
    int numNA = this.LNA.size();
```

```
    String msg = "";
    byte c = 0;
    Vector NI = new Vector();
    Nodo n;

    for (int i = 0; i < numNA; i++) {
       n = (Nodo) this.LNA.get(i);
       if (n.getCAE() > 0) {
          if (n.getCAE() < this.conf.getMAF()) {
             c++;
             msg += (i + 1) + ": "+n.getID() + "\n";
          } else {
             this.log.registra(this.msgNodoIgnorado(n.getID()));
             this.LNI.add(n);
             NI.add(n);
          }
       }
       n.incrementaCAE();
    }

    this.LNA.removeAll(NI);

    if (c > 0) {
       msg = "--- REAVISO ---\nNodos em falha ("+c + "):\n"+msg;
       this.log.registra("Reenviando alerta: " +
             msg.replaceAll("\n", " "));
       this.alertador.envia(msg);
    }
    this.proximoEstado = 1; // ESPERA
}

// implementação do estado ALERTA da maq de estados finitos (6)
private void alerta() {
    int numNF = this.LNF.size();
    String msg =
          "--- FALHA DETECTADA ---\nNodos em falha ("+numNF + "):\n";
    for (int i = 0; i < numNF; i++) {
       msg += (i + 1) + ": "+((Nodo) this.LNF.get(i)).getID() + "\n";
    }
    this.log.registra("Enviando alerta: " + msg.replaceAll("\n", " "));
    // Se este é o primeiro alerta, registra seu momento.
    if (this.momentoPrimeiroAlerta == null) {
       this.momentoPrimeiroAlerta = new Date();
    }
    this.alertador.envia(msg);
    this.proximoEstado = 7; // MARCA
}

// implementação do estado MARCA da maq de estados finitos (7)
private void marca() {
    int numNF = this.LNF.size();
    for (int i = 0; i < numNF; i++) {
```

```
            this.LNA.add(this.LNF.remove(0));
        }
        this.proximoEstado = 4; // VOLTA
    }

    private void imprimeListas() {
        System.out.println(this.getLNF() + " / " + this.getLNA() +
                " / " + this.getLNI());
    }

    private String getLNF() {
        String s = "LNF: [ ";
        for (int i = 0; i < this.LNF.size(); i++) {
            s += ((Nodo) this.LNF.get(i)).getID() + " ";
        }
        s += "]";
        return s;
    }

    private String getLNA() {
        String s = "LNA: [ ";
        for (int i = 0; i < this.LNA.size(); i++) {
            s += ((Nodo) this.LNA.get(i)).getID() + " ";
        }
        s += "]";
        return s;
    }

    private String getLNI() {
        String s = "LNI: [ ";
        for (int i = 0; i < this.LNI.size(); i++) {
            s += ((Nodo) this.LNI.get(i)).getID() + " ";
        }
        s += "]";
        return s;
    }

    private String msgVerificando(String ip) {
        return "Verificando nodo \""+ip + "\"...";
    }

    private String msgNodoOK(String id) {
        return "Nodo \""+id + "\" ONLINE! :-)";
    }

    private String msgNodoMorto(String id) {
        return "Nodo \""+id + "\" OFFLINE! :-(";
    }

    private String msgNodoIgnorado(String id) {
        return "Nodo \""+id + "\" IGNORADO! :-P";
    }
```

```
}
```

Classe Nodos.java

```
import java.util.*;

public class Nodos extends Object {

   private Vector escravos = new Vector();
   private Nodo mestreInativo;

   public Nodos() {
   }

   public void addNodoEscravo(String ip) {
      String id = ip.substring(ip.lastIndexOf(".") + 1);
      Nodo nodo = new Nodo();
      nodo.setIP(ip);
      nodo.setID(id);
      this.escravos.add(nodo);
   }

   public void addNodoEscravo(String ip, String id) {
      Nodo nodo = new Nodo();
      nodo.setIP(ip);
      nodo.setID(id);
      this.escravos.add(nodo);
   }

   public Nodo getMestreInativo() {
      return this.mestreInativo;
   }

   public void setMestreInativo(String ip) {
      String id = ip.substring(ip.lastIndexOf(".") + 1);
      Nodo nodo = new Nodo();
      nodo.setIP(ip);
      nodo.setID(id);
      this.mestreInativo = nodo;
   }

   public void setMestreInativo(String ip, String id) {
      Nodo nodo = new Nodo();
      nodo.setIP(ip);
      nodo.setID(id);
      this.mestreInativo = nodo;
   }

   public Nodo getEscravo(int indice) {
      return (Nodo) this.escravos.get(indice);
```

```
    }

    public Vector getEscravos() {
        return this.escravos;
    }

    public int getNumEscravos() {
        return this.escravos.size();
    }
}
```

Classe Nodo.java

```
public class Nodo extends Object {

    // ip do nodo
    private String IP;

    // identificador do nodo
    private String ID;

    // contador de alertas enviados
    private short CAE;

    public Nodo() {
        this.CAE = 0;
    }

    public void setIP(String ip) {
        this.IP = ip;
    }

    public String getIP() {
        return this.IP;
    }

    public void setID(String id) {
        this.ID = id;
    }

    public String getID() {
        return this.ID;
    }

    public void incrementaCAE() {
        this.CAE++;
    }

    public short getCAE() {
        return this.CAE;
```

```
    }

    public void resetCAE() {
        this.CAE = 0;
    }
}
```

Classe Verificador.java

```
import java.io.*;
import java.net.*;

public class Verificador extends Object {

    private Configurador conf;

    public Verificador() {
        this.conf = Configurador.singleton();
    }

    public boolean verificaNodo(Nodo n) {
        Socket s;
        try {
            s = new Socket();
            s.connect( new InetSocketAddress(n.getIP(),
                    this.conf.getPortaConexao()), this.conf.getTimeout());
            return true;
        } catch (IOException io) {
            String resultado = io.getMessage();
            if (resultado.equals("Connection refused")) {
                return true;
            }
        }
        return false;
    }
}
```

Classe Alertador.java

```
public class Alertador extends Object {

    private SMTP smtp;
    private Configurador conf;
```

```
    public Alertador() {
        this.smtp = new SMTP();
        this.conf = Configurador.singleton();
    }

    public void envia(String mensagem) {
        smtp.setRemetente(this.conf.getRemetente());
        smtp.setDestinatario(this.conf.getDestinatario());
        if (smtp.conecta(this.conf.getServidorSMTP(), this.conf.getPortaSMTP())) {
            smtp.envia(mensagem);
            smtp.desconecta();
        }
    }
}
```

Classe Configurador.java

```
import java.util.*;
import java.io.*;

public class Configurador extends Object {

    static private Configurador _instancia = null;

    /* Variáveis usadas no envio de alertas */
    // Hostname ou IP do servidor SMTP
    private String servidorSMTP;

    // Porta para na qual o servidor SMTP está aceitando conexões
    private short portaSMTP;

    // Destinatário das mensagens de alerta
    private String destinatario;

    // Remetente das mensagens de alerta
    private String remetente;

    /* Variáveis usadas na verificacao dos nodos */
    // Porta que será usada para testar se a conexão é negada em um nodo
    private short portaConexao;

    // Tempo máximo na tentativa de conexão
    private short timeout;

    /* Variáveis utilizadas no fluxo principal de execução do software */
    // quantidade de milisegundos a esperar (estado ESPERA)
    private long tempoWait;

    // quantidade de minutos antes de ocorrer um reenvio
```

```
private short tempoReenvio;

// quantidade máxima de Alertas por Falha
private byte mAF;

// Network Interface Card
// Utilizada para assumir papel de nodo Mestre Primário
private String NIC;

// Os nodos
private Nodos nodos;

/* Variáveis utilizadas na manipulação do log do software */
// Pasta onde deverão ser armazenados os arquivos de log
private String pastaLog;

// Formato do nome do arquivo de log
private String nomeArquivoLog;

// Tamanho máximo (em bytes) do arquivo de log
private int tamArquivoLog;


private Configurador() {
   this.nodos = new Nodos();
}

static public Configurador singleton() {
   if (_instancia == null) {
      _instancia = new Configurador();
   }
   return _instancia;
}

// carrega as configuracoes do arquivo de conf
public boolean carreagaDoArquivo(String nomeArquivo) {
   Properties propriedades = new Properties();
   try {
      FileInputStream arquivo = new FileInputStream(nomeArquivo);
      propriedades.load(arquivo);
      arquivo.close();
   } catch (IOException e) {
      return false;
   }
   this.setServidorSMTP(propriedades.getProperty("servidorSMTP"));
   this.setDestinatario(propriedades.getProperty("destinatario"));
   this.setRemetente (propriedades.getProperty("remetente"));
   this.setNIC (propriedades.getProperty("NIC"));
   this.setPastaLog (propriedades.getProperty("pastaLog"));
   this.setNomeArquivoLog (propriedades.getProperty("nomeArquivoLog"));
   this.setTamArquivoLog (
         (new Integer(propriedades.getProperty("tamArquivoLog"))).
```

```
            intValue());
        this.setPortaSMTP (
            (new Short(propriedades.getProperty("portaSMTP"))).
            shortValue());
        this.setPortaConexao(
            (new Short(propriedades.getProperty("portaConexao"))).
            shortValue());
        this.setTempoReenvio (
            (new Short(propriedades.getProperty("tempoReenvio"))).
            shortValue());
        this.setMAF (
            (new Byte(propriedades.getProperty("mAF"))).byteValue());
        this.setTimeout ( (new Short(propriedades.getProperty("timeout"))).
            shortValue());
        this.setTempoWait (
            (new Long(propriedades.getProperty("tempoWait"))).
            longValue());

        String miIP = propriedades.getProperty("mestreInativo[IP]");
        String miID = propriedades.getProperty("mestreInativo[ID]", "indefinido");
        if (miID.equals("indefinido")) {
            this.nodos.setMestreInativo(miIP);
        } else {
            this.nodos.setMestreInativo(miIP, miID);
        }

        int i = 0;
        String nIP;
        while (!(nIP = propriedades.getProperty("escravo["+i + "][IP]",
            "indefinido")). equals("indefinido")) {
            String nID =
                propriedades.getProperty("escravo["+i + "][ID]", "indefinido");
            if (nID.equals("indefinido")) {
                this.nodos.addNodoEscravo(nIP);
            } else {
                this.nodos.addNodoEscravo(nIP, nID);
            }
            i++;
        }
        return true;
    }

    public String getServidorSMTP() {
        return this.servidorSMTP;
    }

    public void setServidorSMTP(String servidorSMTP) {
        this.servidorSMTP = servidorSMTP;
    }

    public String getDestinatario() {
        return this.destinatario;
```

```
}

public void setDestinatario(String destinatario) {
   this.destinatario = destinatario;
}

public String getRemetente() {
   return this.remetente;
}

public void setRemetente(String remetente) {
   this.remetente = remetente;
}

public short getPortaSMTP() {
   return this.portaSMTP;
}

public void setPortaSMTP(short portaSMTP) {
   this.portaSMTP = portaSMTP;
}

public long getTempoWait() {
   return this.tempoWait;
}

public void setTempoWait(long tempoWait) {
   this.tempoWait = tempoWait;
}

public short getTempoReenvio() {
   return this.tempoReenvio;
}

public void setTempoReenvio(short tempoReenvio) {
   this.tempoReenvio = tempoReenvio;
}

public short getMAF() {
   return this.mAF;
}

public void setMAF(byte mAF) {
   this.mAF = mAF;
}

public short getPortaConexao() {
   return this.portaConexao;
}

public void setPortaConexao(short portaConexao) {
   this.portaConexao = portaConexao;
```

```
    }

    public short getTimeout() {
        return this.timeout;
    }

    public void setTimeout(short timeout) {
        this.timeout = timeout;
    }

    public Nodos getNodos() {
        return this.nodos;
    }

    public void setNodos(Nodos nodos) {
        this.nodos = nodos;
    }

    public String getNIC() {
        return this.NIC;
    }

    public void setNIC(String NIC) {
        this.NIC = NIC;
    }

    public String getPastaLog() {
        return this.pastaLog;
    }

    public void setPastaLog(String pastaLog) {
        this.pastaLog = pastaLog;
    }

    public String getNomeArquivoLog() {
        return this.nomeArquivoLog;
    }

    public void setNomeArquivoLog(String nomeArquivoLog) {
        this.nomeArquivoLog = nomeArquivoLog;
    }

    public int getTamArquivoLog() {
        return this.tamArquivoLog;
    }

    public void setTamArquivoLog(int tamArquivoLog) {
        this.tamArquivoLog = tamArquivoLog;
    }
}
```

Classe Logger.java

```
import java.io.*;
import java.util.*;

public class Logger extends Object {

   private File arquivo;
   private Configurador conf;

   public Logger() {
      this.conf = Configurador.singleton();
   }

   public boolean registra(String linha) {
      Date agora = new Date();
      linha = agora.toString() + " - "+linha + "\n";
      try {
         if (this.abreArquivo()) {
            FileWriter fw = new FileWriter(this.arquivo, true);
            fw.write(linha, 0, linha.length());
            fw.flush();
            fw.close();
            return true;
         } else {
            return false;
         }
      } catch (Exception e) {
         e.printStackTrace();
         System.out.println("Impossível escrever no arquivo de log.");
      }
      return false;
   }

   private boolean abreArquivo() throws Exception {
      File arquivo = new File(this.conf.getPastaLog() + "/"+this.conf.getNomeArquivoLog());
      if (!arquivo.exists()) {
         if (!arquivo.createNewFile()) {
            return false;
         }
      }
      // se o arquivo de log atingiu seu tamanho maximo...
      if (arquivo.length() >= this.conf.getTamArquivoLog()) {
         File f;
         int i = 1;
         do {
            f = new File(this.conf.getPastaLog() + "/"+this.conf.getNomeArquivoLog() + "."+i);
            i++;
         } while (f.exists());
         // renomeia o arquivo atual para nomeArquivo.n
         if (!arquivo.renameTo(f)) {
```

```
            return false;
         }
         // cria um arquivo com o nome correto
         if (!arquivo.createNewFile()) {
            return false;
         }
      }
      this.arquivo = arquivo;
      return true;
   }
}
```

Classe SMTP.java

```
import java.net.*;
import java.io.*;

public class SMTP extends Object {

   private Socket s;
   private DataInputStream entrada;
   private OutputStreamWriter saida;

   private boolean conectado = false;

   private String host;
   private String remetente;
   private String destinatario;

   public boolean conecta(String host, short porta) {
      try {
         this.s = new Socket(host, porta);
         this.entrada = new DataInputStream(s.getInputStream());
         this.saida = new OutputStreamWriter(s.getOutputStream());
         this.conectado = true;
         this.host = host;
         this.verificaResposta();
         return true;
      } catch (Exception e) {
         return false;
      }
   }

   public boolean desconecta() {
      if (this.conectado) {
         try {
            this.saida.close();
            this.entrada.close();
            this.s.close();
            return true;
```

```
        } catch (Exception e) {
            return false;
        }
    }
    return false;
}

public boolean verificaResposta() {
    String resposta;
    try {
        while ((resposta = this.entrada.readLine()) != null) {
            if ( (resposta.indexOf("220") != -1) || (resposta.indexOf("354") != -1) ) {
                return true;
            }
        }
    } catch (Exception e) {
        return false;
    }
    return false;
}

public void setDestinatario(String destinatario) {
    this.destinatario = destinatario;
}

public void setRemetente(String remetente) {
    this.remetente = remetente;
}

public boolean envia(String mensagem) {
    this.helo();
    if (this.verificaResposta()) {
        this.mail();
        if (this.verificaResposta()) {
            this.rcpt();
            if (this.verificaResposta()) {
                this.data(mensagem);
                if (this.verificaResposta()) {
                    this.quit();
                    if (this.verificaResposta()) {
                        return true;
                    }
                }
            }
        }
    }
    System.out.println("sai com erro");
    return false;
}

private synchronized boolean helo() {
    String msg = "HELO " + this.host + " \r\n";
```

```
    int tamanho = msg.length();
    try {
        this.saida.write(msg, 0, tamanho);
        this.saida.flush();
        System.out.println(">> "+msg);
        return true;
    } catch (Exception e) {
        return false;
    }
}

private synchronized boolean mail() {
    String msg = "MAIL FROM: " + this.remetente + "\r\n";
    int tamanho = msg.length();
    try {
        this.saida.write(msg, 0, tamanho);
        this.saida.flush();
        return true;
    } catch (Exception e) {
        return false;
    }
}

private synchronized boolean rcpt() {
    String msg = "RCPT TO: <" + this.destinatario + ">\r\n";
    int tamanho = msg.length();
    try {
        this.saida.write(msg, 0, tamanho);
        this.saida.flush();
        return true;
    } catch (Exception e) {
        return false;
    }
}

private synchronized boolean data(String mensagem) {
    String msg = "DATA\r\n";
    int tamanho = msg.length();
    try {
        this.saida.write(msg, 0, tamanho);
        this.saida.flush();

        msg  = "X-Mailer: NodeWatch992\r\n";
        msg += "Content-Type: text/plain;\r\n";
        msg += "\r\n" + mensagem + "\r\n.\r\n";
        tamanho = msg.length();
        this.saida.write(msg, 0, tamanho);
        this.saida.flush();
        return true;
    } catch (Exception e) {
        return false;
    }
```

```
    }

    private synchronized boolean quit() {
        String msg = "QUIT\r\n";
        int tamanho = msg.length();
        try {
            this.saida.write(msg, 0, tamanho);
            this.saida.flush();
            return true;
        } catch (Exception e) {
            return false;
        }
    }
}
```

ANEXO 2 – ARTIGO

UMA FERRAMENTA PARA MONITORAÇÃO E ALERTAS SMS EM UM AMBIENTE DE CLUSTER COM ALTA DISPONIBILIDADE

M.A.R.Dantas
Departamento de Informática e Estatística (INE)
Universidade de Santa Catarina (UFSC)
88040-900 – Florianópolis – Brasil
E-mail : mario@inf.ufsc.br

R. K. Baggio
Departamento de Informática e Estatística (INE)
Universidade de Santa Catarina (UFSC)
88040-900 – Florianópolis – Brasil
E-mail: baggio@inf.ufsc.br

Resumo *- Atualmente Computação de Alta Performance (HPC) está tornando-se mais necessária e ao mesmo tempo mais acessível. Isso deve-se a tendência de migração das plataformas especializadas para sistemas mais baratos e de propósito geral, ou seja, clusters. No aspecto de manutenção, isso implica em maior empenho por parte do responsável pelo sistema computacional, pois um cluster é formado por vários sistemas independentes. Este trabalho tem como objeto a implementação de uma ferramenta de software que auxilie na tarefa de manutenção de um cluster, monitorando todos os seus nodos e, caso encontre alguma falha, alertando o responsável através de uma mensagem SMS (Short Message Service). O serviço SMS tem basicamente a mesma funcionalidade do serviço de pager, porém garantem a entrega da mensagem mesmo que o dispositivo destino esteja indisponível momentaneamente.*

I. INTRODUÇÃO

Nos últimos tempos, a demanda por Computação de Alta Performance (HPC) vem se tornando cada vez maior. Cientistas dependem de máquinas com alto poder de processamento para poderem levar suas pesquisas a diante e conforme suas pesquisas avançam e tornam-se mais complexas, mais poder de processamento é necessário.

Para suprir essa necessidade é preciso que ocorra o processamento de informações em paralelo, pois com processamento seqüencial o tempo que um computador levaria fazendo cálculos seria muito elevado, a ponto de tornar-se inviável. Tal façanha pode ser alcançada utilizando-se supercomputadores, ou seja, sistemas capazes de executar processamento em paralelo, atingindo um alto desempenho.

Uma abordagem para construção de supercomputadores é através da montagem de um cluster. Clusters são compostos por vários computadores interligados, sendo um deles o nodo mestre que é quem distribui as tarefas para serem processadas pelos demais nodos, chamados de escravos.

Além de ter alta capacidade de processamento, é interessante um cluster possuir também alta disponibilidade, ou seja, estar disponível para uso aproximadamente 100% do tempo. Diz-se aproximadamente, pois na prática não se pode garantir que ocorra 100% de disponibilidade de um sistema computacional, pois devido a sua natureza, tais sistemas estão sujeitos à falhas de hardware, de software e elementos externos.

Apesar das vantagens, um cluster demanda maior atenção por parte dos responsáveis pela tarefa de administrar o cluster. Pois quanto maior o número de nodos que compõem um cluster, mais trabalhosa e complexa se torna a tarefa de identificar e corrigir eventuais problemas que possam ocorrer com cada nodo.

Este trabalho tem como objetivo a implementação de uma ferramenta de software que auxilie na tarefa de detectar eventuais falhas que possam ocorrer com os nodos de um cluster. Quando for detectado que um nodo encontra-se em estado de falha, um aviso será enviado a um telefone móvel, via SMS, para que o responsável pela administração do cluster tome uma atitude para corrigi-la.

Este artigo é organizado como a seguir. Na seção II nós introduziremos alguns conceitos de Cluster Computing. O ambiente e o projeto serão descritos na seção III. Os estudos de caso realizados com a ferramenta desenvolvida são mostrados na seção IV. E finalmente na seção V são apresentadas as conclusões e trabalhos futuros.

II. CLUSTER COMPUTING

2.1. Cluster

A estrutura básica de um cluster é: um nodo atuando como "mestre" que tem o papel de escalonador, distribuindo tarefas para todos os outros nodos, que atuam como "escravos", e montando o resultado final do processamento a partir de cada resultado parcial gerado pelos nodos escravos. Nesta arquitetura há um ponto único de falha evidente: o nodo mestre.

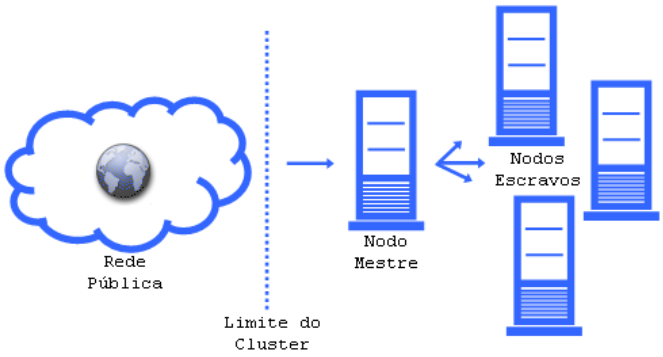

FIGURA 1 - Esquema básico de um cluster HPC.

A Figura 1 deixa claro o motivo de o nodo mestre ser um ponto único de falha, ele assume uma posição de ligação entre a rede pública e os nodos escravos do cluster, análoga à de uma ponte que é a única via de acesso a uma ilha. Caso ocorra uma falha que venha a tornar o nodo mestre indisponível, todo o cluster se tornará também, causando uma ociosidade que poderia ter sido evitada. Para solucionar este problema utiliza-se técnicas de alta disponibilidade específicas para o ambiente de cluster.

2.2. Alta Disponibilidade

No contexto de sistemas computacionais em geral, Alta Disponibilidade (HA) refere-se à existência de componentes no sistema capazes de substituir instantaneamente outros componentes que tornem-se inativos, independendo da causa dessa inatividade. Desse modo garantindo que um sistema nunca, ou quase nunca, ficará indisponível em conseqüência de uma falha.

Num ambiente de cluster, a alta disponibilidade é alcançada através da redundância do nodo mestre. O papel do nodo mestre secundário (redundante) é exatamente o mesmo que o nodo mestre primário, porém a maior parte do tempo de vida do cluster, este fica inativo, apenas monitorando o nodo mestre primário. Ao detectar a indisponibilidade do nodo mestre primário, o secundário assume seu papel, executando todas as funções do mestre primário de maneira idêntica. Desse modo essa substituição entre mestre primário e secundário é transparente para os usuários ou aplicações que utilizam o cluster.

A Figura 2 destaca a diferença de arquitetura entre um cluster e um cluster que implementa as funcionalidades de alta disponibilidade: o nodo mestre secundário.

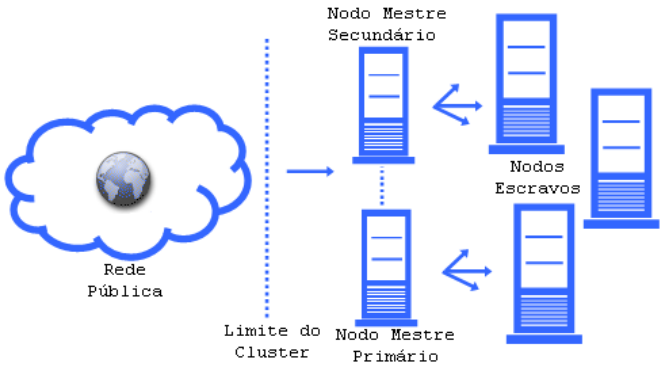

FIGURA 2 - Esquema básico de um cluster HA.

III. O PROJETO

A necessidade de uma ferramenta capaz de monitorar o status de cada nodo do cluster surge quando se tem um cluster com um número elevado de nodos, onde a monitoração manual seria inviável, pois a freqüência das verificações executadas pelo responsável pelo cluster seria muito alta. Tal funcionalidade libera o responsável da monitoração direta dos nodos, porém não da análise dos resultados da monitoração. Assim, tão importante quanto a anterior, é a capacidade de esta ferramenta alertar o responsável pelo cluster caso encontre alguma falha durante a monitoração.

Com uma ferramenta que possui tais características, o responsável pode assumir que o cluster está em seu estado consistente até que receba um alerta que diga o contrário, devendo assim executar o procedimento cabível para recuperar o cluster da falha que ocorreu. Isso reduz significativamente o tempo que o responsável pelo cluster dispenderia na tarefa de verificação/reparo do cluster, sendo necessária sua interação apenas quando houver de fato uma falha a ser corrigida.

Para a realização deste trabalho foi utilizado o SMS Gate fabricado pela Taotronics Tecnologia [14], que consiste em uma plataforma de envio de mensagens de texto (SMS) via telefonia celular a partir de um microcomputador, incluindo hardware e software. As ferramentas de software são multiplataforma e permitem que qualquer aplicação utilize o hardware para enviar SMS de três maneiras diferentes: através de um servidor SMTP, bibliotecas de vínculo dinâmico e utilitários de linha de comando.

A Figura 3 mostra o esquema geral do funcionamento da ferramenta e o fluxo da informação. Este software estará executando em ambos os nodos mestres, porém estará fazendo a monitoração apenas a partir do nodo mestre que estiver no papel de primário, monitorando os nodos escravos e o mestre secundário. Ao ocorrer a detecção de uma falha, o software enviará a mensagem ao servidor SMTP do SMS Gate, acessível através da rede pública, que por sua vez fará a comunicação com o modem SMS, o qual tem a tarefa de encaminhar a mensagem para a rede de telefonia móvel que se encarrega de entrega-la ao destinatário.

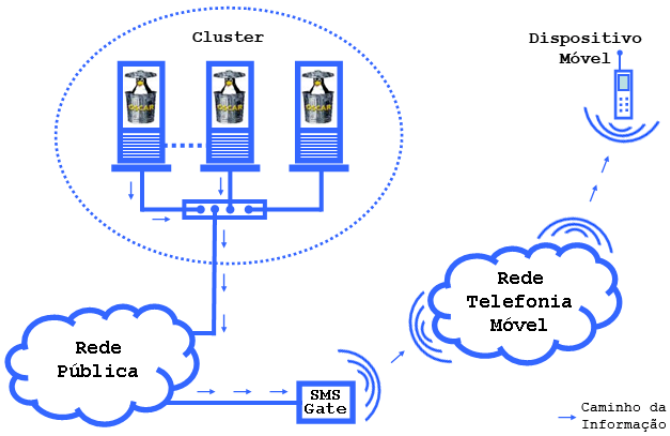

FIGURA 3 - Esquema geral e fluxo da informação na rede.

3.1. O Ambiente

O ambiente específico para o qual este trabalho foi desenvolvido e testado consiste em um cluster construído utilizando-se o pacote de software livre OSCAR versão 3.0 em conjunto com o HA-OSCAR versão BETA.

Este cluster é constituído de três computadores PC comuns, sendo um nodo mestre, um nodo mestre secundário e um nodo escravo. As máquinas que compõem o cluster são homogêneas, ou seja, com a mesma configuração de hardware, interconectadas através de um Switch Fast Ethernet formando uma rede privada. Cada computador possui um processador Intel Pentium 4 operando a 1800MHz, 512MB de memória principal e disco rígido com capacidade para armazenar 40GB. O sistema operacional utilizado foi o Red Hat Linux 9.0, por recomendação dos autores do OSCAR.

3.2. Desenvolvimento

Alguns valores que são usados durante a execução do software devem ser definidos pelo responsável pelo cluster. Para eliminar a necessidade de editar o código fonte e recompilar o software a cada alteração mínima que se deseja fazer no comportamento da ferramenta, foi implementado um sistema de configuração através de arquivo de configurações em texto plano que pode ser editado em qualquer editor de texto comum.

Foram definidas algumas estruturas como base para a implementação desta ferramenta. São elas: Nodo, Nodos e três listas auxiliares LNF, LNA e LNI. Outras estruturas colaterais foram definidas e serão abordadas mais adiante

- **Nodo:** A estrutura Nodo foi definida para representar um nodo do cluster. Esta estrutura possui três atributos principais:
    - **IP:** Representa o endereço IP do nodo no escopo da rede privada.
    - **ID:** Representa o identificador que foi definido para o nodo no arquivo de configuração. Caso não tenha sido definido será atribuído o último octeto do endereço IP ao ID.
    - **CAE (Contador de Alertas Enviados):** Representa a quantidade de alertas que foram enviados informando que este nodo está em falha. Seu valor será entre 0 (zero) e o número máximo de alertas que devem ser enviados por falhas.

- **LN (Lista de Nodos):** Esta estrutura foi definida para conter todos os nodos do cluster. Para isso possui uma lista contendo todos os nodos escravos e um atributo representando o nodo mestre inativo. O conteúdo de seus atributos é lido do arquivo de configuração e permanece inalterado durante todo o tempo de execução da ferramenta. Cada elemento da lista de nodos escravos e o atributo nodo mestre inativo são definidos pela estrutura Nodo.

- **LNF (Lista de Nodos em Falha):** Esta lista contém os nodos que foram identificados como estando em falha no momento em que são verificados. Assim que um alerta for enviado informando ao responsável pelo cluster quais nodos estão em falha, seus elementos são removidos e inseridos na LNA.

- **LNA (Lista de Nodos em Alerta):** Esta lista contém os nodos que estão em falha e devem ter seus alertas reenviados zero ou mais vezes. Todo elemento já teve seu primeiro alerta enviado antes de ser incluído nesta lista. Um elemento é removido da LNA em duas situações: caso saia do estado de falha, ficando funcional novamente ou o número de alertas enviados informando sua falha tenha atingido seu limite. No segundo caso o elemento removido é inserido na quarta lista: a LNI.

- **LNI (Lista de Nodos Ignorados):** Esta lista contém todos os nodos do cluster que já tiveram seu número máximo de alertas enviados e continuam em estado de falha. Um elemento é removido da LNI somente quando seu estado passa a ser funcional novamente.

Sempre que um nodo é removido da LNA ou LNI seu atributo CAE é zerado e caso venha entrar em estado de falha novamente a quantidade de alertas que serão enviados será o número máximo definido no arquivo de configurações.

3.2.1. Máquina de Estados Finitos

A máquina de estados finitos mostrada na Figura 4 representa o funcionamento do software desenvolvido.

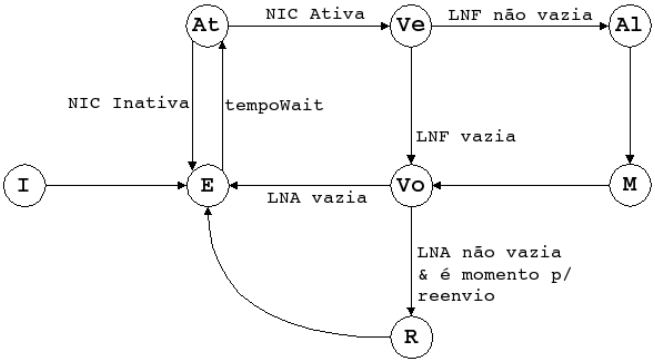

FIGURA 4 - Máquina de Estados Finitos

Estado INICIALIZA (I): É o estado inicial do software, no qual serão carregados os valores dos atributos definidos através do arquivo de configurações. Este estado é executado apenas uma vez na inicialização do software e avança para o estado ESPERA.

Estado ESPERA (E): Este estado identifica o primeiro estágio dos ciclos de execução do software. Após o período de tempo definido pelo atributo tempoWait ter decorrido, o software entra no estado ATIVA.

Estado ATIVA (At): Neste estado o software verifica se

o sistema no qual está executando está na posição de mestre primário ou secundário. Isto é feito verificando o estado da interface de rede definida pelo atributo NIC no arquivo de configurações, caso a interface esteja ativa o software assume que é mestre primário, caso contrário assume que é mestre secundário. Esta verificação é feita através do utilitário de linha de comando netstat[19], presente na maioria das distribuições Linux atuais. Caso o resultado da verificação indique que este sistema é mestre primário, o software avança para o estado VERIFICA, caso contrário retorna ao estado ESPERA. Ainda, caso seja detectado que ocorreu uma passagem de mestre primário para mestre secundário, as listas LNF, LNA e LNI serão tornadas vazias.

Estado VERIFICA (Ve): É neste estado que o software irá verificar o status de cada nodo pertencente ao cluster. Caso encontre algum nodo em falha, verificará se este está na LNI ou na LNA, caso não esteja este será adicionado à LNF e continuará a verificação. Para cada nodo que não esteja em falha, verificará se este está na LNI ou na LNA, caso esteja será removido. Depois de terminada a verificação de todos os nodos, caso a LNF esteja vazia, avança ao estado VOLTA, caso contrário, avança para o estado ALERTA.

Estado ALERTA (Al): Quando o software entrar neste estado, enviará um alerta por SMS para o responsável pelo cluster informando a quantidade total de nodos em falha e um identificador para cada um deles. Este identificador poderá ser definido na configuração do software, caso não seja definido será assumido como identificador o último octeto do IP do nodo. Por exemplo, se o IP do nodo for 192.168.0.101 o identificador será 101 se outro não tiver sido definido no arquivo de configuração. Caso o alerta que esteja sendo enviado for o primeiro alerta desde a inicialização do software, sua data será armazenada com precisão de milissegundos. Após ter enviado o alerta, entrará no estado MARCA.

Estado MARCA (M): Neste estado os elementos que estão na LNF serão transferidos para a LNA, avançando para o estado VOLTA.

Estado VOLTA (Vo): Neste estado é verificado o conteúdo da LNA, caso esteja vazia avança para o estado ESPERA, caso contrário outras verificações serão feitas. A data atual é verificada, com precisão de milissegundos. Desta data é subtraída a data do primeiro alerta enviado, armazenada no estado ALERTA, daí teremos a quantidade de milissegundos decorridos entre o primeiro alerta e o momento atual. Se este intervalo de tempo for maior que um minuto, então ele é convertido em minutos e arredondado para baixo, é verificado se este valor é múltiplo do valor do atributo tempoReenvio, definido no arquivo de configurações. Caso isso seja verdade, então o software avança para o estado REENVIA. Caso contrário avança para o estado ESPERA.

Estado REENVIA (R): Neste estado será enviado um alerta SMS informando o número de nodos que estão na LNA e um identificador para cada um deles, satisfazendo as seguintes restrições: Caso o atributo CAE de um elemento seja 0 (zero) ele será incrementado e o nodo não será incluído no alerta. Caso o atributo CAE de um elemento seja maior que 0 (zero) e menor que o atributo mAF definido no arquivo de configurações, o contador será incrementado e o nodo será incluído no alerta. Caso o atributo CAE de um elemento seja maior que mAF, este será excluído da LNA e incluído na LNI, e o nodo não será incluído no alerta. Depois de enviado o alerta o software retornará para o estado ESPERA.

3.2.2. Método de Verificação

Para verificar o estado de um nodo em particular é necessário que seja estabelecida uma comunicação entre o nodo mestre primário (monitor) e o nodo em questão (monitorado). A partir desta comunicação o monitor deve chegar a uma conclusão sobre o estado do monitorado: em falha ou funcional.

A solução que foi adotada é baseada na recusa de conexão pelo nodo monitorado. O monitor tenta criar uma conexão com o monitorado em uma porta fechada, ou seja, que não esteja aguardando conexões. Caso o monitor obtenha como resposta “Conexão Negada”, significa que o nodo está em estado funcional, pois pôde tratar a tentativa de conexão do monitor e responder de modo adequado. Partindo do pressuposto que a porta na qual está sendo tentada a conexão está fechada, qualquer outra resposta obtida, como “Conexão Expirou”, é considerada como indicador de falha no nodo monitorado. Esta porta é definida no arquivo de configurações pelo responsável pelo cluster, que deve ter conhecimento prévio sobre o estado da porta em todos os nodos do cluster.

IV. ESTUDOS DE CASO

Alguns estudos de casos foram realizados para verificar na prática as funcionalidades propostas pela ferramenta implementada.

4.1. Estudo de Caso 1

Neste Estudo de Caso foi verificado o comportamento da ferramenta ao detectar falhas simultâneas nos nodos mestre secundário (M02) e escravo (E01).

Para simular as falhas nos nodos M02 e E01, os mesmos foram desconectados fisicamente da rede privada. Pouco após o alerta ter sido enviado, o nodo E01 foi reconectado a rede simulando uma recuperação do estado de falha do nodo. Entretanto o estado do cluster continua inconsistente devido à falha no nodo M02, que permaneceu neste estado até ter todos seus alertas enviados e ser adicionado a LNI.

Logo que o nodo M02 passou a ser ignorado pela ferramenta, o mesmo foi reconectado a rede simulando uma recuperação do estado de falha do nodo, retornando ao estado consistente do cluster.

4.2. Estudo de Caso 2

Neste Estudo de Caso foi verificado o comportamento da ferramenta ao detectar uma falha no nodo mestre primário M01.

O nodo foi desconectado fisicamente da rede para simular uma falha. Ao ocorrer uma falha no nodo mestre primário o HA-OSCAR é responsável pela substituição do mesmo pelo mestre secundário, ativando as interfaces de rede necessárias para comunicação com a rede privada e pública. Ao detectar a ativação da interface de rede definida pelo atributo NIC do arquivo de configurações, a ferramenta assume que o nodo no qual está sendo executada assumiu o papel de mestre primário e inicia então a monitoração.

Assim que a mensagem SMS foi recebida, o nodo foi reconectado a rede simulando uma recuperação do estado de falha, retornando ao estado consistente do cluster.

V. CONCLUSÕES E TRABALHOS FUTUROS

A tarefa de administrar um cluster com muitos nodos está longe de ser trivial, e sem uma ferramenta de auxílio demandaria atenção contínua de uma equipe responsável pelo cluster. Com uma ferramenta ficando responsável pela tarefa de monitorar cada nodo do cluster e enviar um alerta caso ocorra alguma falha, o responsável apenas necessitaria intervir caso ocorresse uma falha, para corrigi-la. Nos testes realizados a ferramenta implementada apresentou bons resultados, mostrando as funcionalidades que foram propostas: detectando os nodos que entraram em estado de falha e alertando o responsável pelo cluster via SMS em tempo aceitável.

Como trabalho futuro seria interessante a utilização de um dispositivo de envio de mensagens SMS para a rede de telefonia móvel conectado localmente em cada nodo mestre, de forma a garantir que o alerta será enviado independendo das condições da rede pública.

VI. REFERÊNCIAS

[1] R. M. Alvin, F. L. L. Grossmann, "Implementação de uma Arquitetura para Serviço Distribuído com Grande Disponibilidade em Ambiente Linux." Disponível em <http://www.sbc.org.br/reic/edicoes/2002e3/cientificos>. Acesso em: 04 dez. 2003

[2] S. L. Moser, C. Rista, M. A. R. Dantas, "Alta Disponibilidade – Um Estudo de Caso em um Ambiente de Imagem Única de Produção", 2004

[3] OSCAR – Open Source Cluster Application Resources. Disponível em <http://oscar.sourceforge.net>. Acesso em: 12 abr. 2004

[4] HAOSCAR – High Availability Open Source Cluster Application Resources. Disponível em <http://www.cenit.latech.edu/oscar/>. Acesso em 12 abr. 2004

[5] IEEE Task Force on Cluster Computing. Disponível em <http://www.ieeetfcc.org/>. Acesso em 24 set. 2004

[6] GSM Favorites. Disponível em <http://www.gsmfavorites.com/sms/>. Acesso em 27 set. 2004

[7] Open Cluster Group. Disponível em <http://www.openclustergroup.org/>. Acesso em 10 out. 2004

[8] B. des Ligneris, S. Scott, T. N. N. Gorsuch, "Open Source Application Resources (OSCAR): design, implementation and interest for the [computer] scientific community", 2003

[9] Myricom. Disponível em <http://www.myri.com/myrinet/overview/index.html>. Acesso em 16 out. 2004.

[10] HighAvailability Linux Project. Disponível em <http://linuxha.org/>. Acesso em 16 out. 2004.

[11] ETSI GTS 04.11 V3.3.0 (199501), "European digital cellular telecommunications system (Phase 1); Point-to-point Short Message Service; Support on Mobile Radio Interface (GSM 04.11)".

[12] ETSI GTS GSM 04.11 V5.0.0 (199602), "Digital cellular telecommunications system (Phase 2+) (GSM); Point-to-Point (PP) Short Message Service (SMS) support on mobile radio interface (GSM 04.11)".

[13] ETSI GTS GSM 04.11 V5.1.0 (199603), "Digital cellular telecommunications system (Phase 2+) (GSM); Point-to-Point (PP) Short Message Service (SMS) support on mobile radio interface (GSM 04.11)".

[14] Taotronics Tecnologia. Disponível em <http://www.taotronics.com.br/>. Acesso em 18 out. 2004.

[15] RFC 2821 – Simple Mail Transfer Protocol (SMTP). Disponível em <http://www.faqs.org/rfcs/rfc2821.html>. Acesso em 24 out. 2004.

[16] JavaTM 2 Platform, Standard Edition, v 1.4.2 API Specification. Disponível em <http://java.sun.com/j2se/1.4.2/docs/api/index.html>. Acesso em 18 out. 2004.

[17] Jext 5.0. Disponível em <http://www.jext.org>. Acesso em 11 set. 2004.

[18] Sun Microsystems. Disponível em <http://www.sun.com/>. Acesso em 18 out. 2004.

[19] Philip Blundell, "NetTools v 1.60". Disponível em <http://freshmeat.net/projects/nettools/>. Acesso em 02 nov. 2004.