

UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Leonardo Kunrath

**Estudo e implementação de um  
modelo de detecção de intrusão para  
grids computacionais utilizando  
análise comportamental distribuída**

Florianópolis, novembro de 2005

UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Leonardo Kunrath

# Estudo e implementação de um modelo de detecção de intrusão para grids computacionais utilizando análise comportamental distribuída

Trabalho submetido à Universidade Federal de Santa Catarina como parte  
dos requisitos para a disciplina de Projeto em Ciências da Computação II

Prof. Dr. Carlos Becker Westphall  
Orientador

Florianópolis, novembro de 2005

# Sumário

<b>Resumo.....</b>	<b>05</b>
<b>1 Introdução.....</b>	<b>06</b>
1.1 Objetivos.....	07
1.2 Organização do trabalho.....	07
<b>2 Detecção de Intrusão.....</b>	<b>09</b>
2.1 Tipos de IDSs.....	09
<b>3 Grids Computacionais.....</b>	<b>12</b>
3.1 Fatores Motivadores.....	14
3.2 Evolução dos grids.....	15
3.2.1 Primeira geração.....	15
3.2.2 Segunda geração.....	16
3.2.3 Terceira geração.....	16
3.3 Middleware.....	18
3.3.1 Globus.....	18
3.3.2 Outras plataformas de grid.....	19
<b>4 Proposta.....</b>	<b>20</b>
4.1 Introdução.....	20
4.2 Tecnologias utilizadas.....	21
4.3 Modelo proposto.....	21
4.4 Implementação.....	22
4.4.1 Coletor Local.....	22
4.4.2 Analisador.....	24
4.4.3 Validação.....	26
4.4.4 Resultados Obtidos.....	26
<b>5 Conclusão.....</b>	<b>27</b>
<b>Referências Bibliográficas.....</b>	<b>28</b>
<b>Anexo 1 – Código Fonte.....</b>	<b>31</b>
Anexo 1.1 – Coletor Local.....	31
Anexo 1.2 – Analisador.....	37
Anexo 1.3 – Aplicação Teste.....	42
<b>Anexo 2 – Artigo – Modelo de IDS Distribuído     Para Grids Computacionais.....</b>	<b>48</b>

## **Lista de Figuras**

<b>Figura 01 - Camadas de detecção de intrusão.....</b>	<b>11</b>
<b>Figura 02 - Heterogeneidade do grid.....</b>	<b>13</b>
<b>Figura 03 – Primeira geração da computação em grid..</b>	<b>15</b>
<b>Figura 04 – Terceira geração do grid.....</b>	<b>17</b>
<b>Figura 05 – Analisador do modelo de IDS para grid.....</b>	<b>22</b>
<b>Figura 06 – Diagrama de seqüência do Coletor Local... </b>	<b>23</b>
<b>Figura 07 – Modelo completo de IDS para grid.....</b>	<b>25</b>

# Resumo

A quantidade de computadores disponível em residências, ambientes de ensino, de pesquisa e de trabalho tornou-se tão grande nos últimos anos que o volume de informações trocadas pelos computadores, bem como o seu gerenciamento, se tornou algo muito difícil de ser trabalhado por um conjunto restrito de estações de gerenciamento de rede. Com o intuito de resolver problemas envolvendo gerenciamento de redes, bem como a manipulação de um grande fluxo de dados, e possibilitando que aplicações possam ser projetadas para atuar em larga escala, utilizando recursos computacionais geograficamente distribuídos, as tecnologias de grid foram se aperfeiçoando. Como a quantidade de recursos e máquinas ligadas aos grids poderia ser muito grande, estes foram evoluindo a ponto de permitir tal flexibilidade, bem como agregando aspectos de segurança, desempenho e manipulação de grandes massas de dados. Mas, à medida que a complexidade dos grids foi aumentando, também foram ampliadas as ameaças à segurança. Este trabalho faz um estudo das técnicas para detecção de ocorrências de intrusões, mais conhecidas como *Intrusion Detection System* (IDS), que normalmente são aplicadas a ambientes de gerenciamento de redes que requerem um alto nível de segurança, e sua aplicabilidade a ambientes de grids computacionais, uma vez que estas técnicas se apresentam como uma forma de se lidar com casos onde eventuais falhas na segurança vêm a pôr em risco o funcionamento de um grid computacional.

Palavras-chave: Grid, recursos computacionais, segurança, detecção de intrusão.

# 1 Introdução

A grande melhoria que ocorreu no desempenho e nas tecnologias relacionadas à comunicação entre computadores fez com que a troca de dados entre computadores se tornasse algo fácil, rápido e comum. Esta realidade tornou possível o aparecimento dos grids computacionais, que têm como meta permitir um melhor aproveitamento de recursos computacionais, através do compartilhamento de recursos heterogêneos e geograficamente distribuídos, preocupando-se com qualidade de serviço e segurança.

O grid nasceu como uma infra-estrutura alternativa para as aplicações chamadas de *aplicações de grande desafio*, que até alguns anos atrás tinham como meio de execução as plataformas de computação paralela. O grid teve como papel inicial interligar estas plataformas, compostas por clusters e supercomputadores [Roure 2003].

Hoje, a computação em grid é caracterizada por aplicações construídas a partir de bases de software comum, conhecidas como *middleware*, que viabilizam os aspectos relacionados à comunicação entre computadores e outros aparelhos que possam estar conectados ao grid, à segurança e à implementação do modelo de serviços, que permite organização e aproveitamento apropriado dos recursos computacionais disponíveis, entre outros aspectos. As aplicações, em termos de construção de software, são construídas sobre um framework, que é o software fornecido pelo middleware.

Um grid computacional é, atualmente, uma tecnologia que está em grande expansão. Muitas aplicações comerciais e científicas já estão adotando o grid, no intuito de suprir a necessidade de processar e armazenar grandes massas de dados em uma quantidade limitada de tempo. Um exemplo é o projeto SETI@HOME [SETI@HOME 1996], que busca sinais de vida inteligente fora da Terra, recolhendo *terabytes* de dados através de telescópio e distribuído esses dados para serem processados pelos computadores ligados no grid. No sítio do projeto, está disponibilizado o programa para que colaboradores possam fazer parte do grid e ajudar a processar esses dados. O interessante é que o programa entra em execução assim que o computador põe em funcionamento a proteção de tela, não atrapalhando a execução normal das atividades do colaborador.

Muitos estudos estão sendo realizados para a melhoria das tecnologias de grid em muitos sentidos: para o aumento do seu nível de segurança, tais como estudos de criptografia de dados, autenticação de usuários e máquinas, e detecção de intrusão; para a melhoria de desempenho, através de estudos de metodologias de garantia de QoS; uma melhor disposição e processamento de grandes massas de dados, utilizando-se de bancos de dados interligados aos grids; para a melhoria dos dispositivos móveis, que podem suprir sua baixa capacidade de processamento através da sua ligação com computadores, utilizando-se de grids. Todos estes aspectos tecnológicos referentes a grids estão em desenvolvimento e estão sendo amplamente estudados.

Um dos assuntos que estão sendo mais estudados em relação a grids computacionais é a segurança. O grid computacional, embora utilize abstrações,

e seja organizado em módulos e frameworks oferecidos pelos middlewares, possui uma estrutura muito complexa. Além disso, ele utiliza pontos geograficamente distribuídos, o que torna o sistema suscetível a violações físicas de máquinas ligadas ao grid. As estruturas de segurança do grid têm que considerar tudo isso, agindo de forma a garantir várias características de funcionamento, tais como a integridade e confidencialidade dos dados transmitidos, bem como a disponibilidade dos recursos existentes.

Mesmo assim, as medidas de segurança não dão uma garantia total contra ataques. Para isso, foram desenvolvidos os estudos em detecção de intrusão, que têm se mostrado uma forma eficiente de detectar violações em relação à segurança, recolhendo dados e tomando medidas de forma a evitar que o problema volte a acontecer, bloqueando o eventual invasor do sistema e gerando relatórios, para que os administradores da rede venham a ter conhecimento da invasão ou falha e possam tomar medidas para consertar o problema, como instalar atualizações ou corrigir o software.

A detecção de intrusão tem sido muito usada em servidores de redes, onde há a necessidade de manter o sistema funcionando ininterruptamente e atuando para garantir a velocidade e integridade na troca de pacotes na rede, para inúmeros usuários. Nesse sentido, os softwares de detecção de intrusão agem em sincronia com os softwares de bloqueio de conexões, conhecidos como firewalls.

## **1.1 Objetivos**

Este trabalho tem como objetivo geral propor um modelo de detecção de intrusão para grids computacionais.

Como objetivos específicos deste trabalho, podemos citar:

- Mostrar como a detecção de intrusão pode ser utilizada para proteger grids computacionais;
- A implementação de um protótipo do modelo apresentado;
- A implementação de um programa teste, que simule uma aplicação de grid, com o propósito básico de testar o funcionamento da implementação do modelo de detecção de intrusão para grids proposto.

## **1.2 Organização do trabalho**

Este trabalho foi dividido em seis capítulos, assim distribuídos:

- O primeiro capítulo é a introdução do trabalho, que apresenta a caracterização do problema, objetivos e organização do trabalho, e tem como intuito situar o leitor e contextualizar o tema da pesquisa.
- O segundo capítulo apresenta alguns conceitos sobre detecção de intrusão, suas características e tipos de sistemas de detecção de intrusão
- O terceiro capítulo apresenta os grids computacionais, suas características, a evolução dos grids, e os middlewares.
- O quarto capítulo apresenta a proposta deste trabalho, mostrando o modelo

proposto para detecção de intrusão em grids computacionais e como foi organizada a implementação prática do modelo, bem como os testes realizados.

- O quinto e último capítulo apresenta as conclusões do trabalho, bem como caracterização dos trabalhos futuros.



## 2 Detecção de intrusão

A detecção de intrusão é uma área de segurança de sistemas computacionais. Ela consiste do monitoramento de características de um computador ou rede para detectar se um eventual intruso conseguiu invadir o computador [McHugh 2000].

Segundo [Debar 1999], podemos caracterizar os programas de detecção de intrusão, ou sistemas de detecção de intrusão (IDSs), de acordo com os seguintes critérios: método de detecção, comportamento ao detectar um ataque, e fonte dos dados analisados.

### 2.1 Tipos de IDSs

Os sistemas de detecção de intrusão são classificados de acordo com a forma com que descobrem se houve uma invasão. Podemos classificar os IDSs em:

- Baseados em comportamento: através de medições indiretas, como o uso de memória, CPU e espaço de armazenamento; o IDS ficaria observando as características do uso de cada recurso, procurando algum comportamento anormal, para assim descobrir um invasor. Em um servidor onde não se usa programas que precisam de muito processamento, por exemplo, quando se detectasse que o uso da CPU está acima de certa porcentagem, seria caracterizado que ocorreu uma intrusão no servidor, pois a CPU estaria sendo utilizada por um invasor, atrapalhando, assim, o funcionamento normal de seus programas e com risco de que danos venham a ser causados pelo invasor.
- Baseados em conhecimento: ao se verificar um acesso de uma máquina já conhecida como invasora, ou acesso a um recurso restrito por uma máquina não autorizada, sabe-se, com certeza, que se trata de uma invasão.

Pelo comportamento que os IDSs tomam quando detectam uma intrusão, podemos classificar eles em:

- passivos: apenas registram a invasão em um arquivo de dados, mostram na tela do computador ou enviam um alerta a um administrador;
- ativos: além de detectar e registrar a invasão, tomam ações para bloquear a ação do invasor.

Pelo local de busca dos dados para detectar a intrusão, podemos ter IDSs que buscam dados em:

- pacotes de rede: também chamados de *network based IDSs*. Eles verificam, através dos pacotes, os remetentes dos mesmos e determinam, através do IP, se se trata de um invasor ou não;
- máquinas de uma rede: também chamados de *host based IDSs*. Eles verificam dados vindos das máquinas de uma rede, ou apenas da máquina onde estão

funcionando, como uso de memória, CPU, e disco rígido, como também podem buscar o conhecimento sobre quais são os invasores que tentaram invadir as outras máquinas, para se prevenir contra eles.

- protocolos: alguns tipos de invasão se dão através de protocolos anômalos, que se passam por protocolos como Telnet, HTTP, RPC e SMTP, por exemplo, mas, através de modificações, conseguem passar por barreiras estabelecidas em níveis mais altos de abstração. IDSs deste tipo têm como rotina enviar pacotes através dos protocolos utilizados, com a finalidade de verificar protocolos com anomalias.

E, finalmente, quanto à frequência de uso, eles podem ser IDSs com:

- análise periódica: a cada período de tempo o IDS é acordado e verifica se houve invasão;
- monitoramento contínuo: o IDS fica ligado o tempo todo, checando cada pacote que o computador recebe. Um IDS com esta característica normalmente prejudica bastante o desempenho do computador onde ele está atuando.

A figura 01 apresenta uma visão geral de um sistema simples (de apenas um computador), utilizando as técnicas atuais de proteção contra intrusos. Nota-se, pela figura, que ele acrescenta mais dois níveis, além da detecção *host based* e *network based*, que são os IDSs ativos: o firewall, que permite o bloqueio de conexões não desejadas; e, em vermelho, a detecção de intrusão passiva.

## Layered Security Reduces Network Risk

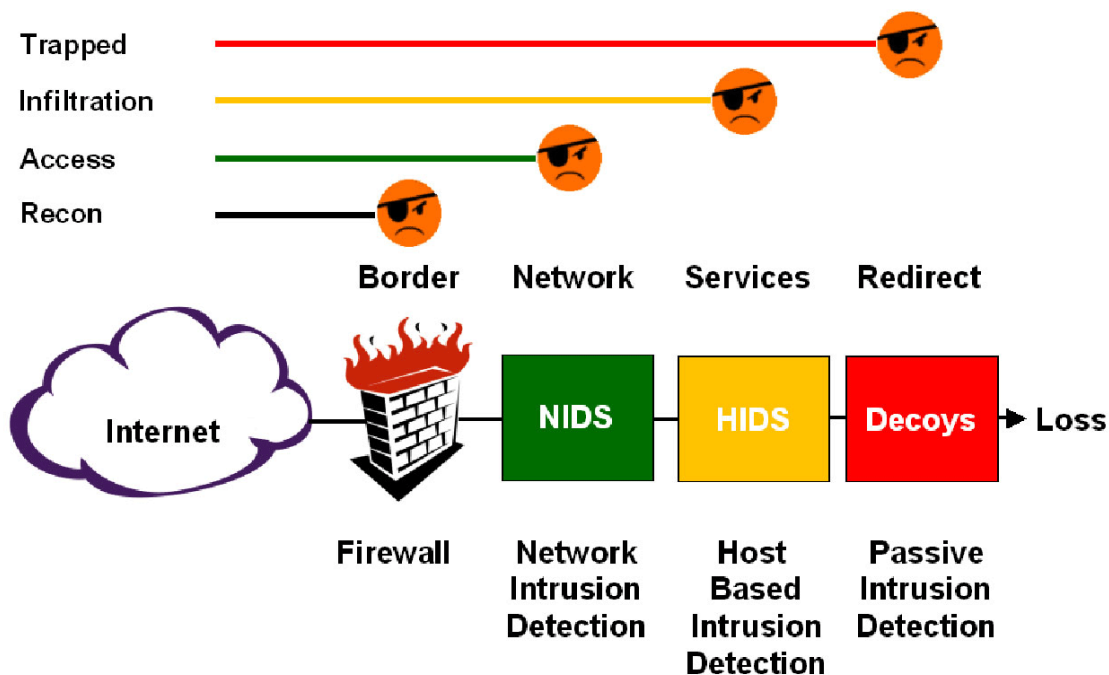


Figura 01 – Camadas de detecção de intrusão [isp-planet 2001].

Alguns programas de detecção de intrusão utilizam redes neurais para descobrir as intrusões. Redes neurais são uma área da computação estudada em inteligência artificial, e são uma técnica que simula o funcionamento de um conjunto de neurônios artificiais. As redes neurais se encaixam muito bem como tecnologia para a detecção de intrusão, e seu uso neste sentido é, atualmente, um grande alvo de estudos.

Um dos programas mais representativos e mais usados para detecção de intrusão atualmente é o SNORT [SNORT 2000]. Este programa é um NIDS e seu funcionamento é organizado de acordo com regras. Quando o programa detecta uma possível intrusão, ele gera um alerta e registra as informações em arquivos, para eventual tratamento através de algum programa ou por um ser humano.

### 3 Grids Computacionais

Os grids são uma evolução natural dos sistemas de computação distribuída. Desta forma, assim como nestes sistemas, eles funcionam através de uma rede, onde há uma gerência das mensagens trocadas entre os computadores e há o processamento de informações contidas nestas mensagens [Coulouris 2002] [Assunção 2003].

A idéia por trás do grid é o compartilhamento de recursos. Através do uso de uma rede, ou da internet, o grid é o ambiente que possibilita a interação entre os computadores, permitindo que eles compartilhem recursos, como processador, memória e periféricos, possibilitando que um grupo de computadores possa interagir como um conjunto com um poder computacional maior do que cada computador individualmente [Foster 1999].

O nome "grid" surgiu de uma comparação com o sistema de distribuição de energia elétrica, em inglês chamado *Electrical Power Grid*, que escondendo de seus usuários a forma como a energia é gerada e distribuída, oferece para os mesmos o acesso fácil, barato e seguro à energia elétrica, utilizando-se de inúmeros pontos de geração e cabos de transmissão. Assim também, o grid computacional é pensado como o meio para oferecer os mais diversos recursos computacionais, que estão geograficamente distribuídos, escondendo dos usuários a grande complexidade por trás.

Segundo Ian Foster, pesquisador do Argonne National Laboratory, no início da década de 1990, a comunidade científica percebeu que, com as redes de alta velocidade, passou a se fazer possível o compartilhamento de recursos em larga escala e para novas aplicações [Foster 1999].

As aplicações, para que funcionem em um ambiente de grid, devem estar preparadas para executar remotamente. Desde o início, para que estas aplicações tenham sucesso, os computadores remotos devem possuir todos os recursos necessários pela aplicação, podendo ser de software ou hardware [Bettoni 2004]. Ainda assim, não é para toda aplicação que o uso do grid vale a pena.

Uma aplicação que vale a pena é uma onde um satélite tira milhares de fotos por dia, e os dados são processados para se ter a previsão do tempo, por exemplo. É uma aplicação que poderia se encaixar perfeitamente em um grid, pois estaria distribuindo estes dados para serem executados remotamente e recolhendo os resultados. Já uma aplicação como um editor de texto, por exemplo, não necessita de interações com computadores remotos para que seu funcionamento ocorra da forma desejada.

Muitas empresas já estão utilizando a infra-estrutura de grid para muitas de suas aplicações. Nos horários em que os computadores não estão sendo utilizados, fora do expediente ou de noite, eles podem e são utilizados pelos grids computacionais para as mais diversas aplicações, desde a organização de bancos de dados distribuídos, processamento de vendas, preços de produtos, cálculo de valores de acordo com cotações de moedas ou mudanças de taxas.

Na figura 02, vemos como um grid pode interligar ambientes de supercomputação, clusters, centros de pesquisa, entre outros; de forma a se obter muitos recursos computacionais para todos os usuários do grid.

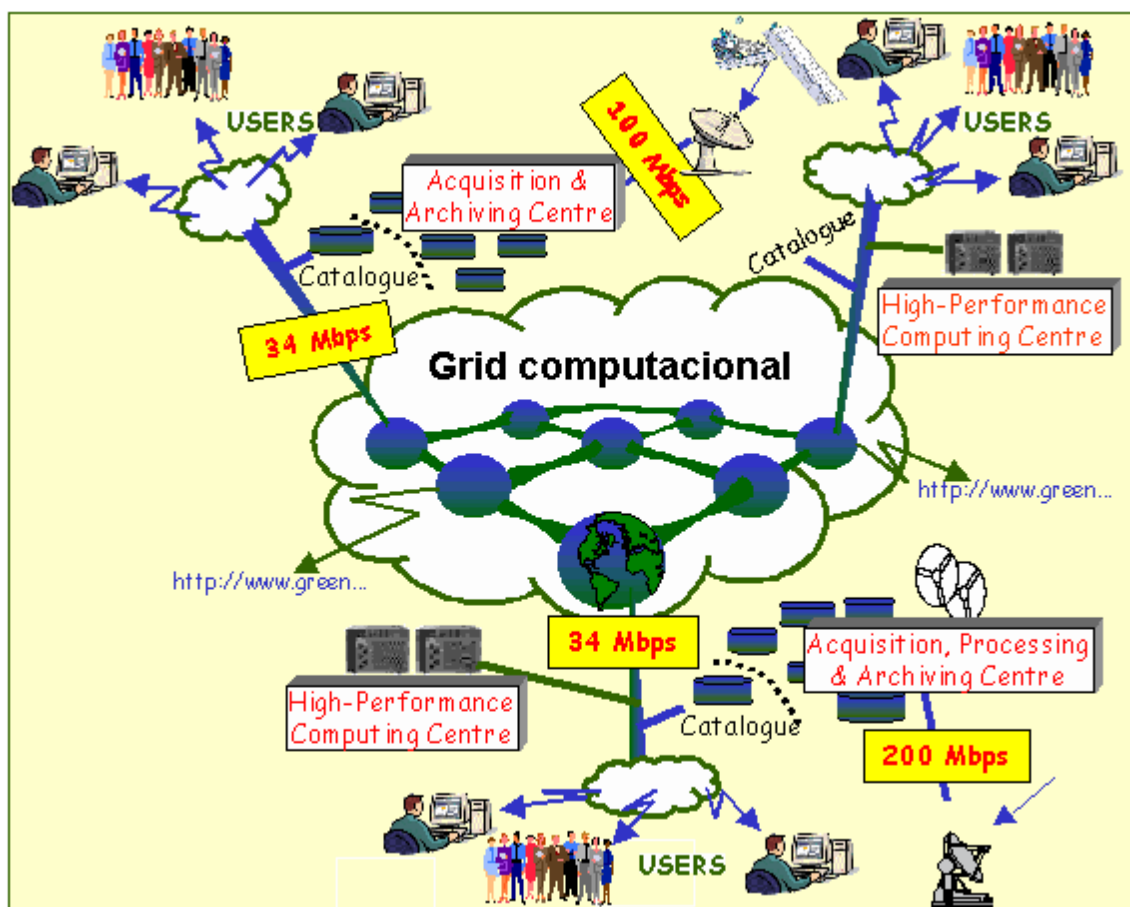


Figura 02 – Heterogeneidade do grid.

Outras aplicações que são adequadas para serem executadas em um grid são aplicações físicas e matemáticas. Um típico problema que utiliza cálculos físicos e matemáticos é o desenvolvimento de novas peças mecânicas. Por exemplo, no caso de se ter o desenvolvimento de uma carcaça para um automóvel, deve-se fazer diversos cálculos para ver qual a pressão a carcaça do automóvel pode suportar. O método ideal para isso é dividir em partes esses cálculos. Calcula-se a pressão sofrida em cada centímetro quadrado da carcaça, ou milímetro quadrado, e depois basta juntar os resultados para ver qual a espessura terá esta carcaça, ou o material mais adequado do qual ela será feita. Neste caso, pode-se fazer que cada milímetro quadrado a ser calculado será considerado uma tarefa, e estas serão dadas ao grid computacional para serem executadas pelos computadores remotamente.

Note que a aplicação descrita acima tem uma característica muito importante: cada tarefa a ser realizada (no caso o cálculo da pressão sobre um trecho de um milímetro quadrado da carcaça do automóvel) pode ser executada em paralelo como as outras tarefas. Em um caso de se ter cem computadores neste grid, em teoria, esta aplicação poderia ser executada até cem vezes mais rapidamente do que no caso de ser executada em um único computador. Esta característica de paralelismo, ou baixo acoplamento, caracterizado por uma

baixa dependência entre as partes do programa, faz com que ele possa ser muito bem aproveitado em um grid computacional.

### 3.1 Fatores motivadores

Desde a antiga ARPANET, que surgiu em 1969 como um projeto modesto do DARPA, que em seu surgimento tinha apenas quatro nós (e mais tarde evoluiu para a atual internet), a troca de informações entre computadores foi ganhando cada vez mais importância. O surgimento da *Ethernet*, que começou a ser utilizada em 1976, três anos depois de ser apresentada por Bob Metcalfe [Metcalfe 1976] em sua tese de PhD, permitindo a interligação de computadores e dispositivos, foi um dos marcos decisivos para que as redes de computadores chegassem ao que são hoje, bem como possibilitou o surgimento dos clusters.

No início da década de 1980, a ARPANET já contava com centenas de nós. Esta expansão foi possível graças ao fato de a responsabilidade pela ARPANET ter sido transferido dos interesses militares para os acadêmicos. Mas o fator que causou o maior crescimento e popularização da internet foi o surgimento do e-mail e da Web, como é destacado em [Berman 2003].

Ainda na década de 1980, foram feitos grandes estudos com relação a programação distribuída. A idéia era usar programas que poderiam se comunicar com outros programas remotos, trocando informações através de mensagens. Os sistemas operacionais da época começaram a desenvolver este suporte, em destaque o UNIX, que trabalhava muito bem este aspecto.

Mais tarde, o barateamento do hardware e a disseminação da internet fizeram com que uma infinidade de tecnologias surgissem e evoluíssem. Entre elas podemos citar uma das tecnologias que teve um papel fundamental para a fundamentação do grid da maneira que ele é hoje, que é a tecnologia dos Web Services [Horewicz 2004], que trabalha com requisição de serviços, onde um computador requisita um serviço, seja ele uma informação retirada de um banco de dados, ou o processamento de determinada informação, e o outro computador envia uma resposta.

Em paralelo com estas tecnologias surgidas a partir da internet, as pesquisas no campo da supercomputação também evoluíram bastante nas últimas duas décadas, graças à *Ethernet*, que proporcionou um controle da troca de informações em uma rede interna de forma rápida e fácil, e também graças à evolução dos sistemas operacionais, que passaram ter suporte para poder trabalhar com mais que um processo funcionando ao mesmo tempo e mais que uma *Thread*, o que gerou inúmeros estudos sobre a chamada *programação paralela*. Esta evolução levou ao surgimento dos *clusters* [Pitanga 2004], que foi a partir de onde iniciaram as pesquisas dos grids computacionais.

## 3.2 Evolução dos grids

O grid computacional sofreu grandes mudanças em sua estrutura desde a sua criação. Segundo [Roure 2003], pode-se dividir o desenvolvimento de grids computacionais em 3 diferentes gerações, desde sua criação até os atuais Web Services.

### 3.2.1 Primeira geração

A primeira geração é marcada pelo início do grid computacional, que tinha como objetivo interligar centros de supercomputação, de forma a se juntar uma grande quantidade de recursos computacionais para o uso em aplicações de alto desempenho. As principais preocupações do grid nesta época estavam focadas na comunicação entre os componentes do grid, no gerenciamento de recursos e no armazenamento e acesso de dados. Na figura 03, vemos como o grid, nesta geração, interligava supercomputadores, clusters, satélites e laboratórios de pesquisa.

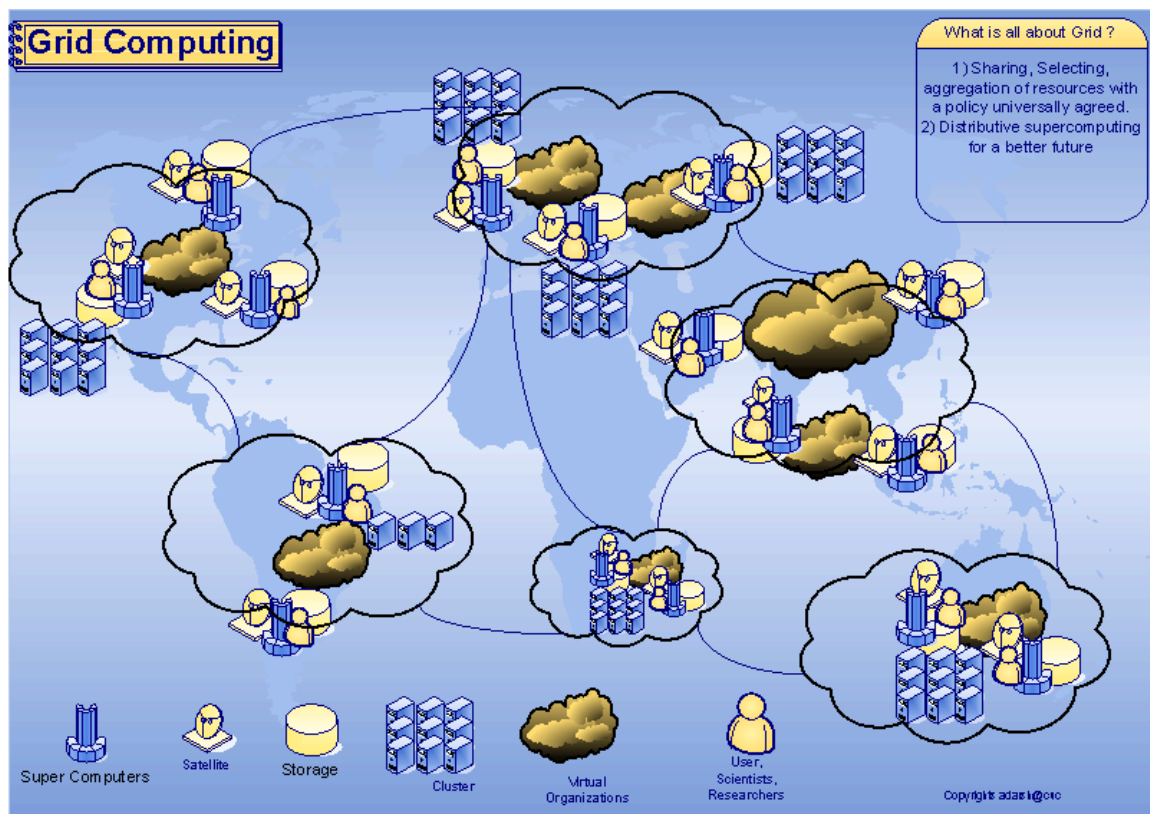


Figura 03 – Primeira geração da computação em grid.

Os principais projetos de grid desta época, precursores dos grids, foram o projeto I-WAY, que conseguiu com sucesso interligar, via ATM, 17 centros de computação dos EUA, e o projeto FAFNER.

### **3.2.2 Segunda geração**

A melhoria na infra-estrutura e velocidade das redes, o surgimento da internet de banda larga, bem como a adoção de padrões fez com que o grid se disseminasse em escala global. Nesta geração, os grids passaram a suportar não apenas supercomputadores, mas os mais diversos tipos de recursos, desde aparelhos científicos, supercomputadores e clusters, a computadores comuns.

Segundo Bettoni [Bettoni 2004], três fatores são marcados como de fundamental importância neste contexto:

->Heterogeneidade - os recursos com o qual o grid trabalha podem ser os mais diversos, podendo ser de software e hardware, e espalhados em diferentes domínios.

->Escalabilidade - o grid pode trabalhar com uma quantidade muito grande de máquinas interligadas e interagindo. O grid tem que estar preparado para evitar os problemas de desempenho decorrentes do crescimento da quantidade de máquinas e recursos. Assim, as aplicações devem estar preparadas para tolerar eventuais atrasos que possam ocorrer.

->Adaptabilidade - devido à quantidade de recursos ser muito grande e heterogênea, é praticamente certo que alguns deles venham a falhar. O grid deve estar preparado para este tipo de situação e saber se adaptar para que, mesmo que um recurso falhe, a aplicação consiga ser bem sucedida e não haja uma perda de desempenho significativa.

Para conseguir solucionar estes problemas, teve surgimento o middleware, que é uma camada de software entre o sistema operacional e as aplicações, que tem como finalidade tratar e esconder os detalhes do tratamento das conexões e do tratamento dos recursos heterogêneos, de forma a permitir que os usuários possam trabalhar com um ambiente homogêneo e padronizado.

Outro fator importante trabalhado nesta geração do grid diz respeito à segurança. Aspectos como integridade, confidencialidade e autenticação passaram a ser trabalhados como aspectos fundamentais para garantir a segurança do grid. A segurança do grid é algo complexo, pois o grid deve manter seu desempenho, ao mesmo tempo que coordena a interação de recursos, não podendo impedir o uso de recursos individualmente, e garantindo o funcionamento do sistema como um todo.

Além disso, esta geração trabalhou na dinamicidade do grid, de forma a permitir que mais dispositivos pudessem se anexar ao grid durante seu funcionamento, e pudessem também abandonar o grid.

### **3.2.3 Terceira geração**

Nesta geração, o principal foco de atenção está voltado ao aumento da flexibilidade do grid e mais rápido desenvolvimento de aplicações para grid, através de reaproveitamento de partes de software. Para permitir isso, o uso de padrões foi usado ao máximo.

Devido ao aumento da velocidade dos computadores pessoais da atualidade, a estrutura do grid começou a se organizar de forma a permitir que



se conseguisse um grande poder computacional através da interligação de muitos computadores pessoais, ainda permitindo que clusters e supercomputadores se interligassem ao grid, mesmo que, na atualidade, eles apareçam em menor número nos grids.

A arquitetura do grid encontrou como forma mais adequada aos seus objetivos um modelo que se baseia em Web Services. O modelo orientado a serviços adotado utiliza os chamados *metadados*, que são dados que não contêm informações dos clientes, mas sim contêm informações sobre outros dados. O modelo de grid mais utilizado nesta geração, do middleware Globus Toolkit, é uma extensão de Web Services, que foram chamados de Grid Services [Focke 2004]. Na figura 04, vemos uma estrutura de grid desta geração.

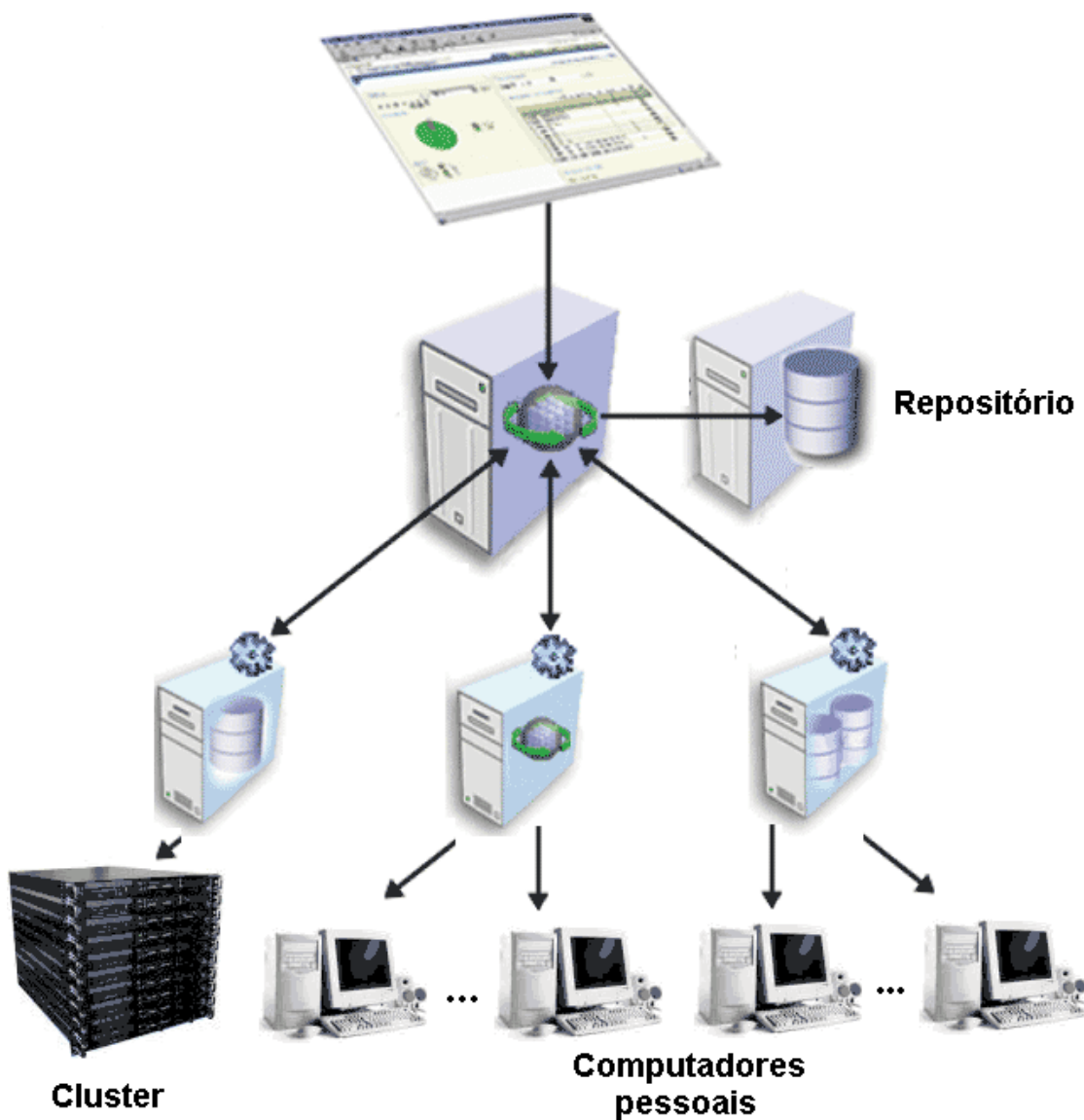


Figura 04 –Terceira geração do grid.

## 3.3 Middleware

Os middlewares são camadas de software que têm como função ficar entre o sistema operacional e as aplicações de grid. O seu papel foi ganhando importância à medida que a abrangência dos grids foi aumentando, e foram acrescentadas novas características, como segurança, escalabilidade, adaptabilidade, entre outras.

O princípio dos middlewares é que, quando se deseja construir uma aplicação para grid, não é necessário sair do zero. A parte do grid que cuida das conexões, autenticação, segurança e serviços, por exemplo, funcionalidades necessárias em qualquer desenvolvimento bem estruturado para aplicações em grid, são as partes que os middlewares oferecem. Assim, no desenvolvimento de aplicações de grid, as atenções ficam voltadas para a aplicação em si, que trabalhará sobre um ambiente pronto e padronizado. Um dos middlewares mais estudados e utilizados da atualidade é o Globus [Globus 1996].

### 3.3.1 Globus

O projeto The Globus Alliance [Globus 1996] é o projeto mais significativo em relação a grids computacionais da atualidade. Ele define um conjunto de padrões para grid chamados OGSA e OGSi, e distribui livremente o Globus Toolkit, um framework [Fayad 1999] para aplicações baseadas em grid que implementa estes padrões. O Globus Alliance é uma comunidade de organizações e indivíduos, que têm como objetivo desenvolver as tecnologias por traz do grid.

O OGSA, que é a sigla de *Open Grid Service Architecture*, define a arquitetura padrão para as aplicações baseadas em grid. O OGSA define a parte semântica do grid: o que ele é, o que ele deve ser capaz de fazer, em quais tecnologias ele deve se basear, mas não define os detalhes técnicos da implementação.

O OGSi, que é a sigla de *Open Grid Services Infrastructure*, define uma especificação formal para os serviços do grid, tendo como base todos os conceitos descritos pelo OGSA. Ele é uma especificação baseada nos padrões das tecnologias Web Services, como a linguagem de especificações WSDL, por exemplo.

O Globus Toolkit é uma implementação livre e de código aberto, das especificações do OGSi. Ele é um framework de grid, ou seja, uma implementação quase completa de um programa que rodará como uma aplicação de grid, cuja única parte implementada é a estrutura que permitirá que o programa rode como um programa de grid, ou seja, a parte que trata da comunicação entre computadores, segurança, implementação da lógica de serviços, uso de banco de dados, entre outros; enquanto o que fica faltando é a parte da aplicação que será o programa que rodará sobre o grid. Juntos, o framework e a aplicação desenvolvida funcionam como uma aplicação de grid.

As versões 3 e 4 do Globus Toolkit são implementações em Java, ao contrário de versões anteriores, onde a implementação era em C++. Isso porque

a linguagem de programação Java tem crescido muito nos últimos dez anos, e devido à suas características que garantem um desenvolvimento mais veloz e seguro. Além disso, ela possui uma quantidade de bibliotecas de software e documentação muito grande, incluindo bibliotecas prontas para se lidar com Internet e com bancos de dados.

O Globus Toolkit 3 (GT3), lançado oficialmente em julho de 2003, é o projeto voltado a grids mais referenciado da atualidade. A versão mais atual do Globus Toolkit é a versão 4 (GT4), lançado em 2005. Ambas são implementações em Java, distribuídas através de uma licença de software aberto livre. O site do Globus [Globus 1996] permite que se baixe o Globus Toolkit gratuitamente, tanto a versão compilada quanto o código fonte, permitindo assim que a comunidade possa não somente utilizar o software, mas também pesquisar sobre o código no intuito de desenvolver novas contribuições para o modelo.

A implementação foi construída de forma modular, ou seja, não é necessário testar tudo do zero quando se faz uma alteração, apenas o conjunto de classes daquele módulo. Isso foi feito com o propósito de suportar alterações mais facilmente, garantindo maior confiabilidade do software e permitindo que seja mais fácil de se trabalhar com o mesmo.

### 3.3.2 Outras Plataformas de Grid

Além do Globus, outros middlewares estão com projetos em desenvolvimento. Eles se distinguem do Globus por apresentar propostas diferentes para se trabalhar no desenvolvimento de aplicações de Grid.

Um dos projetos que se destaca é o GridLab [GridLab 2001]. Ele possui uma arquitetura definida em camadas bem definidas, chamada *GridLab Architecture* [Gridlab\_Architecture 2001]. Em sua camada mais alta, ele possui uma API chamada Grid Application Toolkit [GAT 2001] e um framework chamado GridSphere [GridSphere 2001]. As demais camadas são trabalhadas com aspectos de segurança, monitoramento, autenticação, manipulação e armazenamento de dados e gerenciamento de serviços.

Outro projeto que se destaca é o *CoABS Agent Grid* [CoABS 1998], que trabalha com um tipo de grid chamado grid de agentes [Assunção 2004]. Um grid de agentes trabalha com os chamados agentes de software [Bettoni 2004], que são uma espécie de objetos inteligentes, que trabalham normalmente como *threads*, podem se comunicar entre si e com o ambiente, e buscam cumprir objetivos através da realização de tarefas. Esse projeto está sendo muito pesquisado por envolver os agentes, trabalhados em inteligência artificial e no gerenciamento de redes, como é o exemplo dos modelos de agentes autônomos e dos sistemas multi-agentes [Wooldridge 2001].

## 4 Proposta

A detecção de intrusão aplicada a grids computacionais é uma área que ainda não foi muito explorada, e necessita de estudos. Foram poucas as iniciativas neste sentido, como é o caso do SANTA-G [Kenny 2005].

### 4.1 Introdução

Como dito anteriormente, a detecção de intrusão pode ser de dois tipos: a baseada em monitoramento da rede e a baseada em análise comportamental. A proposta visa fazer uma extensão da segunda, se adaptando ao problema do grid.

Como ponto de partida desta proposta, consideremos o seguinte cenário: em um determinado momento, durante o funcionamento do grid, um dos computadores pertencentes ao grid tenha sido invadido, ou seja, passa a ser controlado por uma pessoa que não foi autorizada a usar o computador. E, através do computador, o intruso consegue acesso aos recursos do grid.

Neste caso, o invasor poderia se utilizar de todas as vantagens do grid, enviando quaisquer tarefas para serem realizadas pelo grid, desde a tentativa de quebrar uma senha, até a de um ataque conjunto de negação de serviço a certo alvo. O invasor teria ao seu dispor todo poder computacional do grid, tanto para processamento quanto para armazenamento.

Nesse ponto é onde entra a aplicação de detecção de intrusão. Ela monitorará o uso de memória e processamento dos computadores do grid de forma que se possa identificar a presença de um invasor pela análise de seu uso do grid.

Em um grid pode-se estipular que cada usuário possa utilizar, no máximo, determinada porcentagem dos recursos disponíveis. Neste caso, se um invasor tentar utilizar o grid, ele provavelmente não conhecerá esta política de utilização, ultrapassaria a quantidade de recursos permitida e assim poderia ser identificado.

Outra situação seria a de um grid onde cada usuário deve mandar uma notificação de que vai utilizar o grid para o administrador da rede, indicando o horário em que estará utilizando. Um invasor, por não saber disso, não enviaria esta notificação e, desta forma, o detector de intrusão verificaria o uso do grid em um momento não previsto (não agendado), sendo assim o invasor descoberto.

Vale lembrar que há uma grande diferença na detecção de intrusão por análise comportamental em um único computador e a para um grid. Na primeira, analisa-se apenas as variáveis de sistema da máquina em questão para verificação se há comportamento malicioso. Na segunda, verifica-se as variáveis de sistema de todas as máquinas do grid ao mesmo tempo e, usando todos estes dados reunidos, verifica-se se o grid está sendo utilizado e em que proporção ele está utilizando os recursos disponíveis.

## 4.2 Tecnologias utilizadas

Foi utilizado o sistema operacional Linux para a implementação do trabalho. A distribuição de Linux usada foi o Ubuntu [Ubuntu 2004], que tem se mostrado uma distribuição (sistema operacional e mais um conjunto de programas) de Linux atualizada, estável, segura e fácil de instalar e usar.

A linguagem de programação utilizada foi a linguagem Java, por se apresentar como uma linguagem atual, fácil de se programar e com vários recursos e bibliotecas para acesso remoto.

Outras tecnologias que podem ser mencionadas por contribuírem no trabalho foram o programa *make*, que pode agrupar um conjunto de comandos para facilitar a compilação e organização na programação; o editor de texto adaptado para programação *Gedit*; e os arquivos do Linux */proc/stat* e */proc/meminfo*, que são arquivos atualizados pelo próprio sistema operacional, disponibilizando informações sobre o uso de processador e memória.

## 4.3 Modelo proposto

A implementação do modelo proposto foi inspirada pela idéia do programa jMon [jMon 1999], que é um programa que recolhe informações de uso de CPU, memória e uso de Swap de uma lista de computadores determinada pelo usuário. O jMon é um software livre, distribuído sobre a licença GPL [GPL 1989][GNU 1984], que garante que ele seja distribuído gratuitamente, juntamente com o código fonte, e possa ser modificado e utilizado para quaisquer outros fins; o que permitiu o seu estudo e, assim, sua contribuição para as idéias do trabalho.

O modelo proposto funciona da seguinte forma: há um computador não pertencente ao conjunto de computadores do grid no qual um aplicativo executa, chamado neste trabalho de *Analizador*.

A função do *Analizador* é recolher os dados de uso do grid e processá-los no sentido de determinar se houve uma invasão ou não. O *Analizador* recolherá periodicamente os dados de uso de processador e memória de cada um dos computadores ligados no grid.

Na figura 05, vemos como a aplicação *Analizador* está interligada com cada computador pertencente ao grid para, assim, recolher os dados do grid como um todo.

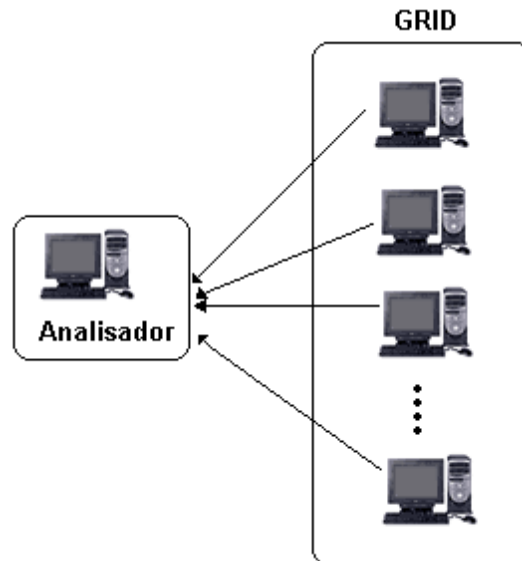


Figura 05 – *Analisador* do modelo de IDS para grid.

Depois de recolher estes dados de uso dos recursos processador e memória, o *Analisador* faz uma média de uso destes recursos para saber o grau de utilização do grid como um todo. No caso de este grau de utilização ultrapassar uma taxa pré-determinada, será considerado que o grid possivelmente foi invadido. Neste caso, o *Analisador* colocará em um arquivo as informações relevantes sobre esta invasão, como a hora em que aconteceu a invasão e o grau de utilização do grid encontrado.

## 4.4 Implementação

Para que o *Analisador* possa adquirir as informações de uso de processador e memória de cada máquina do grid, é necessário que em cada uma destas máquinas esteja rodando um programa que possa fornecer estas informações. Este programa foi chamado neste trabalho de *Coletor Local*.

### 4.4.1 Coletor Local

O *Coletor Local*, como dito acima, é executado em cada máquina do grid, e fica enviando para o *Analisador* as informações de uso de memória e CPU da máquina onde estiver sendo executado. O *Coletor Local* não está diretamente ligado no middleware ou na aplicação de grid, apenas fica em execução no mesmo computador. Como a aplicação de grid consome os recursos do computador, o *Coletor Local* fica checando a proporção de uso destes recursos. Note que esta solução é ideal para um grid onde os computadores são dedicados exclusivamente ao grid, mas mesmo assim é uma solução ainda aplicável a casos onde não há esta exclusividade, uma vez que a média de uso de recursos do grid não será drasticamente alterada quando um

host tem um de seus recursos utilizados por uma aplicação externa ao grid.

O programa *Coletor Local* está dividido da seguinte forma:

- Classe *Atualizador*: é uma sub classe da classe *Thread*, portanto atua como uma *Thread*. Possui os métodos *atualizaUsoDeCPU* e *atualizaUsoDeMem*. O *Coletor Local* utiliza um objeto desta classe para ficar atualizando as informações do uso de CPU (processador) e memória do computador onde está sendo executado, recolhendo, para isto, os dados de uso de CPU e memória dos arquivos do Linux */proc/stat* e */proc/meminfo*, respectivamente, e colocando estes dados em um objeto da classe *Estatistica*.
- Classe *Estatistica*: possui dois valores, chamados *memoriausada* e *cpuusada*. A *Thread Atualizador* fica atualizando estes valores periodicamente, e a *Thread Conexao* usa estes valores para enviar para o *Analizador*.
- Classe *Conexao*: também é uma sub classe da classe *Thread*. Ela abre uma conexão por socket com a aplicação *Analizador*, e fica enviando para ela as informações de uso de CPU e memória que estão no objeto da classe *Estatistica*.
- Classe *coletorlocal*: é a classe inicial, que contém o método *main* (o método de partida do Java). Ele contém um objeto da classe *ServerSocket*, que fica recebendo requisições de conexão de Socket; ou seja, ele fica esperando que o programa *Analizador* se conecte nele, para abrir uma conexão por Socket para interligar os dois programas. Quando a requisição chega, é criado um objeto da classe *Conexao* e iniciado ele como uma nova *Thread*, para interagir com o programa *Analizador*.

Podemos visualizar melhor o funcionamento do *Coletor Local* na figura 06.

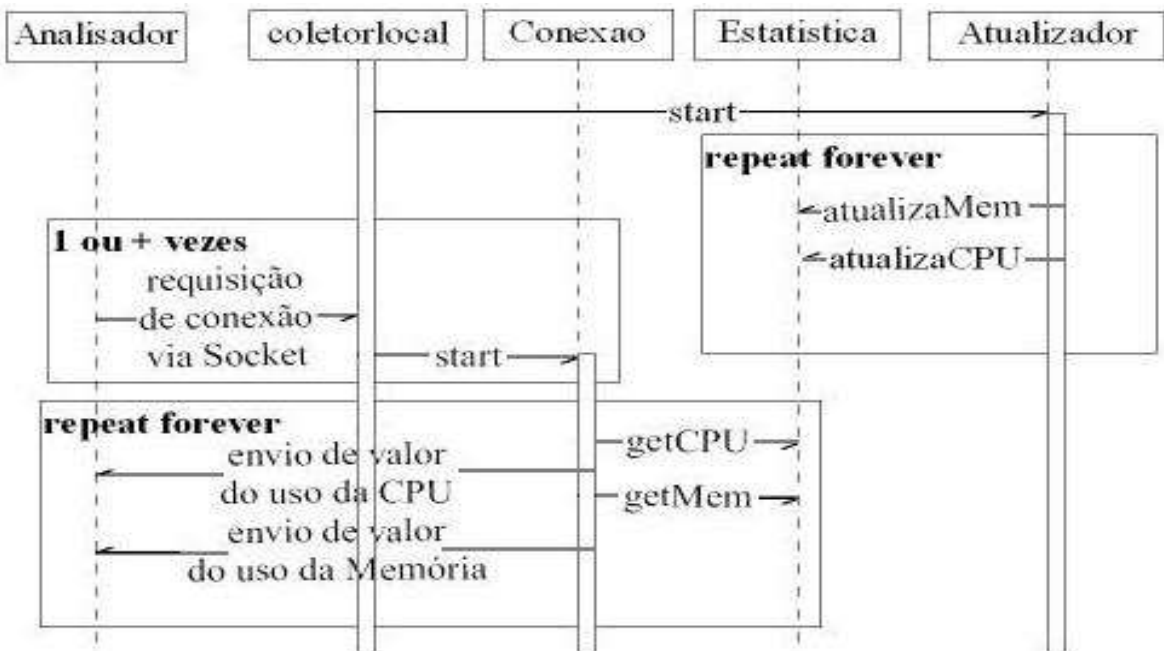


Figura 06 – Diagrama de seqüência do *Coletor Local*.

## 4.4.2 Analisador

Como dito anteriormente, o *Analisador* funcionará em uma máquina fora do grid. Ele se conectará em cada máquina pertencente ao grid, a fim de obter as informações de uso de recursos do grid como um todo.

Para se conectar com as máquinas do grid, o programa *Analisador* receberá um arquivo que contém uma lista de IPs das máquinas do grid, que deve ser passado para o programa *Analisador* no momento que ele inicia sua execução. Quando o programa começar a executar, ele lerá linha por linha deste arquivo, e, para cada IP lido, ele abrirá uma conexão via Socket com a máquina correspondente, se conectando ao programa *Coletor Local* que estará funcionando em cada uma delas.

Após todas as conexões estabelecidas, o programa *Analisador* receberá, periodicamente, os dados de uso de processador e memória de cada uma das máquinas do grid, enviadas pelo programa *Coletor Local* de cada máquina. Após receber estes dados, eles serão juntados, e será feita uma média dos valores recebidos, a fim de se obter um valor que corresponda ao uso de grid como um todo.

Quando o *Analisador* verificar que uma dessas médias ultrapassou um valor limite, ele considerará que está ocorrendo o uso abusivo do grid e, portanto, uma possível invasão. Assim, ele enviará os dados de uso de recursos para um *arquivo de log*, juntamente com o horário em que isto ocorreu, a fim de que o administrador do sistema possa tomar alguma medida corretiva para fazer o sistema voltar ao normal, tal como verificar quem está utilizando as máquinas do grid e expulsar um usuário invasor, através de bloqueio por firewall ou por outros meios. A figura 07 apresenta a disposição completa da aplicação de detecção de intrusão voltada ao grid.



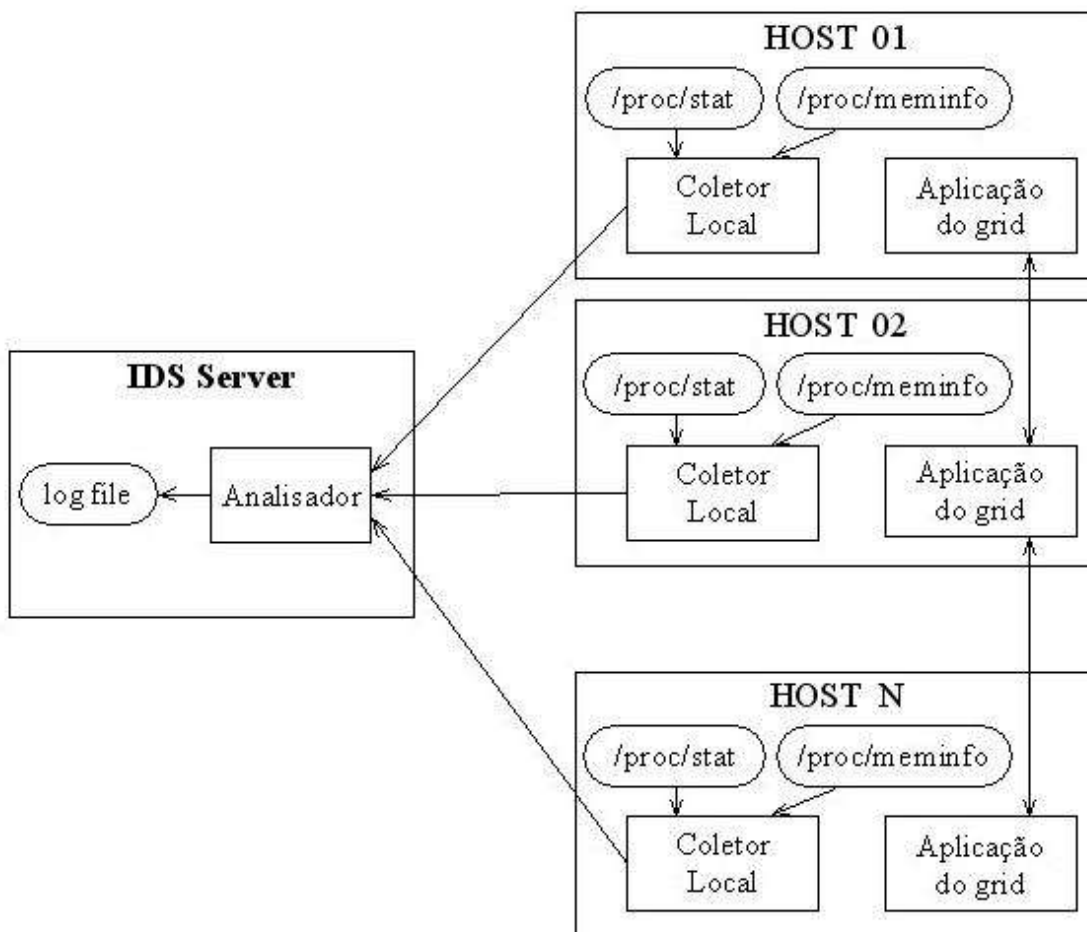


Figura 07 – Modelo completo de IDS para grid.

Os valores de limite para que o *Analisador* considere que houve uma invasão devem ser passados para o *Analisador*, assim como o arquivo com os IPs das máquinas do grid, ao início da sua execução. Desta forma, é o administrador do grid que escolherá os valores para os quais o grid pode estar sendo invadido. Nos testes realizados neste trabalho, foram estabelecidos os valores limite de 85% para uso de CPU e 70% para uso de memória. Portanto, se a média de utilização do uso de CPU das máquinas do grid ultrapassasse 85% ou a média de utilização de memória do grid ultrapassasse 70%, o *Analisador* consideraria como uma invasão e enviaria os dados de uso dos recursos e horário que ocorreu para o *arquivo de log*.

Vale ressaltar que foi considerada responsabilidade do administrador do sistema verificar o *arquivo de log* e tirar suas conclusões sobre o fato de ter ocorrido uma intrusão ou não. O programa implementado apenas indica possíveis intrusões; cabe ao administrador confirmar se o sistema realmente está sendo invadido, ou apenas está passando por uma situação anormal pré-determinada. Mesmo assim, o administrador poderá, posteriormente, e se achar necessário, construir um programa que leia este *arquivo de log* e tire conclusões sozinho, para tratar do grid que está sendo invadido. Este problema não foi

abordado neste trabalho, pois tratar uma invasão vai além do escopo estabelecido de apenas detectar intrusos.

### **4.4.3 Validação**

Para testar o *Coletor Local* e o *Analizador* foi criada uma aplicação distribuída que simula um grid. Esta aplicação recebeu o nome de *Aplicação Teste*, e sua função é simular uma aplicação em grid, utilizando os recursos de um conjunto de computadores, assim como um grid utilizaria estes recursos.

A escolha de uma aplicação que simula um grid, ao invés de um grid em si, foi feita no sentido de simplificar os testes, já que as características oferecidos por um middleware, tais como integridade, autenticação, criptografia, busca e escalonamento de recursos, entre outros, não seriam necessárias para a análise dos resultados.

### **4.4.4 Resultados Obtidos**

Com a aplicação de detecção de intrusão para grids computacionais pronta, e também da *Aplicação Teste*, foi montado um ambiente de testes para a validação do modelo proposto.

Os testes foram realizados com 2 e 3 computadores. Nestes, foi mantida em execução a *Aplicação Teste*, que simula o grid, e o *Coletor Local*. Em um outro computador, foi executado o *Analizador*, que ficou recolhendo os dados de cada *Coletor Local*, conseguindo, desta forma, dados sobre o uso do grid como um todo, e os analisando.

Foram testadas duas situações de funcionamento: uma com uso excessivo de processamento, e outra com uso excessivo de memória. Em ambos os casos, o sistema de detecção de intrusão cumpriu o que foi proposto: indicar os casos de uso excessivo dos recursos memória e processador.

## 5 Conclusões e Trabalhos Futuros

A detecção de intrusão é um método que permite verificar invasões em um computador ou sistema, como é o caso de um grid. Além disso, ela permite que as invasões sejam detectadas no momento em que acontecem, permitindo que se pare a ação do invasor.

Em um grid computacional, identificar violações de segurança é uma tarefa muito árdua, já que o grid é uma estrutura muito complexa. A detecção de intrusão aplicada ao grid é um método que permite detectar ataques e, assim, determinar mais facilmente falhas de segurança no grid.

O cenário que levou ao modelo proposto levou em conta o caso de um grid invadido e utilizado pelo invasor, uma vez que o grande atrativo de um grid é o seu grande poder computacional.

O modelo proposto foi de se criar uma aplicação de detecção de intrusões que coletasse os dados de uso de memória e processador de cada computador do grid (*Coletor Local*), periodicamente, enviando estas informações para um único computador (*Analisador*). De acordo com este modelo, esses dados são tratados (pelo *Analisador*) e, quando é detectada uma condição no sistema de uso excessivo de memória ou processador, essas informações são armazenadas em um arquivo.

Uma característica do modelo proposto foi que o computador com maior quantidade de processamento (o *Analisador*) está situado externamente ao grid, fazendo com que a detecção de intrusão não atrapalhe significativamente o processamento do grid.

A implementação foi testada através de uma aplicação que simula o funcionamento de um grid, distribuindo carga de processamento e usando memória de forma excessiva dos computadores do grid.

Uma contribuição ao trabalho realizado, que se encaixaria como trabalho futuro, seria uma aplicação que agiria depois que se tome o conhecimento de uma intrusão, tomando medidas de forma a banir o invasor do ambiente e garantir que ele não tivesse como invadir o grid novamente da mesma maneira.

Outra contribuição que poderia ser dada ao trabalho seria a expansão do algoritmo de detecção de intrusão. O modelo apresentado foi moldado baseando-se em comportamento do uso de recursos das máquinas do grid (host-based IDS). Além desse tipo de detecção de intrusão, existe o baseado em informações de pacotes transitados na rede (network-based IDS). Essa outra contribuição seria dada pelo monitoramento dos pacotes da rede onde está o grid, monitorando as conexões, a fim de se coletar uma quantidade maior de informações sobre possíveis invasores.

## Referências bibliográficas

**ASSUNÇÃO, M. D.** *Implementação e Análise de uma Arquitetura de Grids de Agentes para a Gerência de Redes e Sistemas*. Dissertação de Mestrado (Curso de Pós Graduação em Ciências da Computação), Universidade Federal de Santa Catarina. 2003.

**BERMAN, F.; FOX, G.; HEY, T.** *The Grid: Past, Present, Future. Grid Computing: Making the Global Infrastructure a Reality*. A special issue of Concurrency and Computatio: Practice and Experience. 2003.

**BETTONI, C. A.** *Segurança de Agentes em Grids para Gerência de Redes de Computadores*. Florianópolis. 2004.

**COABS.** *CoABS Agent Grid*. Sítio do projeto. Disponível em: <<http://www.objs.com/agility/tech-reports/9812-grid.html>>. Acessado em: 10 nov. 2004.

**COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.** *Distributed Systems: Concepts and Design*. Addison-Wesley, 2002.

**DEBAR, H.; DACIER, M.; WESPI, A.** *Towards a Taxonomy of Intrusion-Detection Systems*. Computer Networks, 31. p. 805-822. abr. 1999.

**DIE.NET.** 1996. Sítio do projeto. Disponível em: <<http://www.die.net/doc/Linux/man/man5/proc.5.html>>. Acessado em: 25 set. 2005.

**FAYAD, E. M.; SCHMIDT, D. C.; JOHNSON, R. E.** *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. 1999.

**FOCKE, N.** *Introduction to Grid Computing with Globus*. Florianópolis, out. 2004.

**FOSTER, I.; KESSELMAN, C.** *The Grid: Blueprint for a New Computing Infrastructure*. San Mateo: Morgan Kaufmann. 1999.

**GAT.** *Grid Application Toolkit*. 2001. Disponível em: <<https://www.gridlab.org/WorkPackages/wp-1/>>. Acessado em: 03 mar. 2005.

**GLOBUS.** *The Globus Alliance*, 1996. Sítio do projeto. Disponível em: <<http://www.globus.org/>>. Acesso em: 10 mar. 2005.

**GNU.** *GNU Project*. 1984. Sítio do projeto. Disponível em: <<http://www.gnu.org/>>. Acessado em: 23 jul. 2005.

**GPL.** *General Public License*. 1989. Sítio do projeto. Disponível em:

<<http://www.gnu.org/copyleft/gpl.html>>. Acesso em: 09 set. 2005.

**GRIDLAB.** *GRIDLAB: A Grid Application Toolkit and Testbed*. 2001. Sítio do projeto. Disponível em: <<https://www.gridlab.org/>>. Acessado em: 15 jan. 2005.

**GRIDLAB ARCHITECTURE.** Disponível em:  
<[https://www.gridlab.org/Images/gridlab\\_architecture.pdf](https://www.gridlab.org/Images/gridlab_architecture.pdf)>. Acessado em: 02 mar. 2005.

**GRIDSPHERE.** Disponível em: <<https://www.gridlab.org/WorkPackages/wp-4/>>. Acessado em: 03 mar. 2005.

**HOREWICZ, V. H. V.** *Utilização de Web Services em Ambientes de Grid Computacional*. Florianópolis. 2004.

**ISP-PLANET.** *White Paper: Intrusion Detection: Reducing Network Security Risk*. 2001. Pag. 03. Disponível em: <[http://www.isp-planet.com/perspectives/ids\\_p3.html](http://www.isp-planet.com/perspectives/ids_p3.html)>. Acessado em 23 set. 2005.

**JMON.** *Distributed Resource Monitor*. 1999. Sítio do projeto. Disponível em: <<http://www.dsp.sun.ac.za/~jbb/jmon/>>. Acessado em: 19 set. 2005.

**KENNY, S.; COGHLAN, B.** *towards a Grid-wide Intrusion Detection System*. Proc. EGC'05. p. 275-284. Amsterdam, Holanda, 14-16 fev. 2005.

**METCALFE, R.; BOGGS, D.** *Ethernet: Distributed Packet Switching for Local Computer Networks*. Proceedings of ACM National Computer Conference. Vol.19, num.5, 1976.

**MCHUGH, J.** *Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory*. Disponível em:  
<<http://www1.acm.org/pubs/citations/journals/tissec/2000-3-4/p262-mchugh/>>. ACM Transaction on Information and System Security, 3(4). nov. 2000.

**PITANGA, M.** *Computação em Cluster – O Estado da Arte da Computação*. Ed. Brasport. 344 p. Edição: 1. 2004.

**ROURE, D. et al.** *Grid Computing: Making the Global Infrastructure a Reality*, chapter *The Evolution of the Grid*. International Journal of Concurrency and Computation: Practice and Experience, v.15, n.11, New York, 2003.

**SETI@HOME.** *Search for extraterrestrial Intelligence*, 1996. Sítio do projeto. Disponível em: <<http://setiathome.ssl.berkeley.edu/>>. Acesso em: 15 dez. 2004.

**SNORT.** Sítio do projeto. 2000. Disponível em: <<http://www.snort.org/>>. Acesso em: 13 mar. 2005.

**UBUNTU.** *UBUNTU - Linux for Human Beings*. 2004. Sítio do projeto. Disponível em: <<http://www.ubuntulinux.org/>>. Acesso em: 10 ago. 2005.

**WOOLDRIDGE, M. J.** *An Introduction to Multiagent Systems*. Baffins Lane, Chechester, West Sussex, England. 348p. 2001.

# Anexo 1 – Código Fonte

O código fonte do programa de detecção de intrusões para grids computacionais está dividido em dois programas: o *Coletor Local*, que roda em cada host do grid, e o *Analísador*, que roda em um computador separado e coleta as informações.

Para validação foi construída a *Aplicação Teste*, que rodará em cada host, simulando um grid.

## Anexo 1.1 – Coletor Local

A árvore de diretórios e arquivos fonte deste programa é a seguinte:

```
./coletorlocal/  
  ./coletorlocal/Makefile  
  ./coletorlocal/bin/  
    ./coletorlocal/bin/Makefile  
  ./coletorlocal/src/  
    ./coletorlocal/src/coletorlocal.java  
    ./coletorlocal/src/Atualisador.java  
    ./coletorlocal/src/Conexao.java  
    ./coletorlocal/src/Estatistica.java  
    ./coletorlocal/src/Teste.java
```

O conteúdo dos arquivos fonte segue abaixo.

### **./coletorlocal/Makefile**

```
/////////////////////////////////////////////////////////////////  
all: clean compile  
  
compile:  
    javac -sourcepath src -d bin src/*.java  
  
clean:  
    rm -f src/*~ bin/*.class  
  
run:  
    make run -C bin
```

### **./coletorlocal/bin/Makefile**

```
/////////////////////////////////////////////////////////////////  
run:  
    java coletorlocal
```

```
////////////////////////////////////////////////////////////////
```

## **./coletorlocal/src/coletorlocal.java**

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

public class coletorlocal
{
    public static void main(String args[])
    {
        System.out.println("Coletor local iniciado!!!");
        Estatistica est=new Estatistica();
        Atualizador at=new Atualizador(est);
        at.start();

//        Teste test=new Teste(est);
//        test.start();

        try
        {
            ServerSocket ss=new ServerSocket(10777);
            while(true)
            {
                (new Conexao(ss.accept(),est)).start();
                System.out.println("Nova conexao iniciada.
Esperando proxima...");
            }
        }
        catch (IOException ioe)
        {
            System.out.println("Erro em analisador no metodo
main");
            System.exit(-1);
        }
        est.show();
    }
};
/////////////////////////////////////////////////////////////////
```

## **./coletorlocal/src/Atualizador.java**

```
/////////////////////////////////////////////////////////////////
import java.lang.Thread;
import java.lang.InterruptedException;
import java.io.File;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Atualizador extends Thread
{
    private Estatistica est;
    private int totaltempoUsuario=0;
    private int totaltempoUsuarioNice=0;
    private int totaltempoKernel=0;
    private int totaltempoiddle=0;

    public Atualizador(Estatistica e)
    {
```



```

        est=e;
    }

    public void atualizaUsoDeCPU() throws IOException
    {
        try
        {
            BufferedReader br = new BufferedReader(new FileReader
(new File("/proc/stat")));
            String linha = br.readLine();
            String[] partes = linha.split(" ");
            int indice=1;
            while(partes[indice].length()==0) indice++;

            int tUsuario=(new Integer(partes[indice])).intValue
()); /*tempo usuario usando cpu*/
            int tUsuarioNice=(new Integer(partes[indice+1])).
intValue(); /*tempo usuario nice usando*/
            int tKernel=(new Integer(partes[indice+2])).intValue
()); /*tempo kernel usando cpu*/
            int tiddle=(new Integer(partes[indice+3])).intValue
()); /*tempo de cpu livre*/

            int tempodecpuusada=tUsuario +tUsuarioNice + tKernel
- totaltempoUsuario - totaltempoUsuarioNice - totaltempoKernel;
            totaltempoUsuario=tUsuario;
            totaltempoUsuarioNice=tUsuarioNice;
            totaltempoKernel=tKernel;

            double porcentagemdecpuusada=((double)
tempodecpuusada/(tempodecpuusada+(tiddle-totaltempoiddle))*100;
            totaltempoiddle=tiddle;

            br.close();

            est.atualizaUsoDeCPU(porcentagemdecpuusada);
        }
        catch(IOException ioe)
        {
            throw ioe;
        }
    }

    public void atualizaUsoDeMem() throws IOException
    {
        try
        {
            BufferedReader br = new BufferedReader(new FileReader
(new File("/proc/meminfo")));

            String linha = br.readLine();
            String[] partes = linha.split(" ");
            int indice=1;
            while(partes[indice].length()==0) indice++;
            int memtotal=(new Integer(partes[indice])).intValue();

            linha = br.readLine();
            partes = linha.split(" ");
            indice=1;
            while(partes[indice].length()==0) indice++;
            int memlivre=(new Integer(partes[indice])).intValue();

```



```

public Conexao(Socket s,Estatistica e)
{
    conexao=s;
    est=e;
}

public void run()
{
    boolean baux=true;
    while(baux)
    {
        baux=false;
        try
        {
            ObjectOutputStream out=new ObjectOutputStream
(conexao.getOutputStream());
            out.writeInt(47);
            out.flush();
            ObjectInputStream in=new ObjectInputStream
(conexao.getInputStream());

            if(in.readInt()==74)
            {
                System.out.println("*Conexao iniciada
corretamente");
            }
            else
            {
                System.out.println("Erro iniciando
conexao");
                break;
            }

            /*****FAZ ALGO AQUI*****/
            double cpu_usage;
            double mem_usage;
            boolean finaliza=false;
            while(true)
            {
                cpu_usage=est.usoDeCPU();
                mem_usage=est.usoDeMem();
                out.writeDouble(cpu_usage);
                out.writeDouble(mem_usage);
                out.flush();

                finaliza=in.readBoolean();
                if(finaliza)
                {
                    break;
                }
                sleep(13);
            }

            System.out.println(" fechando conexao
corretamente\n");
            conexao.close();
        }
        catch (IOException ioe)
        {
            System.out.println("Erro na conexao");
        }
    }
}

```



```

/////////////////////////////////////////////////////////////////
import java.lang.Thread;
import java.lang.InterruptedException;

public class Teste extends Thread
{
    Estatistica est;

    public Teste(Estatistica e)
    {
        est=e;
    }

    public void run()
    {
        while(true)
        {
            try
            {
                est.show();
                sleep(13);
            }
            catch (InterruptedException ie)
            {
                est.sinalizaErro(1); //1=erro no sleep
                break;
            }
        }
    }
}
/////////////////////////////////////////////////////////////////

```

## Anexo 1.2 – Analisador

A árvore de diretórios e arquivos fonte deste programa é a seguinte:

```

./analisador/
./analisador/Makefile
./analisador/hosts
./analisador/bin/
./analisador/bin/Makefile
./analisador/src/
./analisador/src/analisador.java
./analisador/src/ConexaoAnalisadorColetor.java
./analisador/src/GridData.java

```

O conteúdo dos arquivos fonte segue abaixo.

**./analisador/Makefile**

```

/////////////////////////////////////////////////////////////////

```

```
all: clean compile

compile:
    javac -sourcepath src -d bin src/*.java

clean:
    rm -f src/*~ bin/*.class

run:
    make -C bin run
```

```
////////////////////////////////////////////////////////////////
```

### **./analizador/hosts**

```
////////////////////////////////////////////////////////////////
```

```
150.162.63.4
150.162.63.5
```

```
////////////////////////////////////////////////////////////////
```

### **./analizador/bin/Makefile**

```
////////////////////////////////////////////////////////////////
```

```
run:
    java analisador ./analizador/hosts ./analizador/resultados 70 55
```

```
////////////////////////////////////////////////////////////////
```

### **./analizador/src/analizador.java**

```
////////////////////////////////////////////////////////////////
```

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

public class analisador
{
    public static void main(String args[])
    {
        System.out.println("\nAnalisador iniciado! ");
        System.out.println(" args[0] = "+args[0]+";\n args[1] =
"+args[1]+";\n args[2] = "+args[2]+";\n args[3] = "+args[3]);
        try
        {
            BufferedReader br = new BufferedReader(new FileReader
(new File(args[0])));
            BufferedWriter bw = new BufferedWriter(new FileWriter
(new File(args[1])));

            GridData gd=new GridData(new Double(args[2]).
doubleValue(),new Double(args[3]).doubleValue());

            String ip=br.readLine();
            int i=0;
            while(ip!=null)
            {
                (new ConexaoAnalisadorColetor(ip,gd,i)).start();
                i++;
            }
        }
    }
}
```

```
        ip=br.readLine();
    }

    String relatorio=null;
    while(true)
    {
        relatorio=gd.verificaIntrusao();
        if(relatorio!=null)
        {
            bw.write(relatorio);
            bw.newLine();
            bw.flush();
        }
        Thread.sleep(15000);
    }
}
catch (FileNotFoundException fnfe)
{
    System.out.println("Erro: verifique se o arquivo de
hosts foi passado corretamente");
    System.exit(0);
}
catch (InterruptedException ie)
{
    System.out.println("Erro no metodo sleep");
    System.exit(0);
}
catch (IOException ioe)
{
    System.out.println("Erro: verifique a rede, ou se o
servidor encontra-se ligado");
    System.exit(0);
}
}
};
```

### **./analizador/src/ConexaoAnalizadorColetor.java**

```
////////////////////////////////////
import java.io.*;
import java.net.*;
import java.lang.Thread;

public class ConexaoAnalizadorColetor extends Thread
{
    Socket conexao;
    String ip;
    GridData dados;
    int id;

    public ConexaoAnalizadorColetor(String s,GridData g,int
identificador)
    {
        ip=s;
        dados=g;
        id=identificador;
        dados.adicionaConexao();
    }

    public void run()
```

```

    {
        while(true)
        {
            try
            {
                conexao=new Socket(ip,10777);
                ObjectInputStream in=new ObjectInputStream
(conexao.getInputStream());
                ObjectOutputStream out=new ObjectOutputStream
(conexao.getOutputStream());

                out.writeInt(74);
                out.flush();
                if(in.readInt()!=47)
                {
                    System.out.println("Erro testando
confiabilidade da conexao");
                    System.exit(-1);
                }
                boolean finaliza=false;
                String linhaDeAnalise="";
                while(!finaliza)
                {
                    dados.atualizaUsoDeCPU(id,in.readDouble
());
                    dados.atualizaUsoDeMem(id,in.readDouble
());

                    out.writeBoolean(finaliza);
                    out.flush();
                    Thread.sleep(5000);
                }
                conexao.close();
            }
            catch (InterruptedException ie)
            {
                System.out.println("Erro no metodo sleep");
                break;
            }
            catch (IOException ioe)
            {
                System.out.println("Erro: verifique a rede, ou
se o servidor encontra-se ligado");
                break;
            }
        }
        System.out.println("Thread "+id+" finalizando");
    }
}

```

```

////////////////////////////////////

```

### **./analizador/src/GridData.java**

```

////////////////////////////////////
import java.util.LinkedList;
import java.lang.Double;
import java.sql.*;
import java.io.*;
import java.net.*;
import java.util.*;

```



```

public class GridData
{
    protected LinkedList<Double> gridusodecpu=new LinkedList<Double>
();
    protected LinkedList<Double> gridusodemem=new LinkedList<Double>
();
    protected double limiteDeUsoDeCPU;
    protected double limiteDeUsoDeMem;

    public GridData(double cpumax,double memmax)
    {
        limiteDeUsoDeCPU=cpumax;
        limiteDeUsoDeMem=memmax;
    }

    public synchronized void adicionaConexao()
    {
        gridusodecpu.add(new Double(30.0));
        gridusodemem.add(new Double(30.0));
    }

    public synchronized String verificaIntrusao()
    {
        double mediacpu=0;
        for(Double usoDeCPUDeCadaNodoDoGrid:gridusodecpu)
        {
            mediacpu+=usoDeCPUDeCadaNodoDoGrid.doubleValue();
        }
        mediacpu=mediacpu/gridusodecpu.size();
        double mediamem=0;
        for(Double usoDeMemDeCadaNodoDoGrid:gridusodemem)
        {
            mediamem+=usoDeMemDeCadaNodoDoGrid.doubleValue();
        }
        mediamem=mediamem/gridusodemem.size();

        Time tempo=new Time(GregorianCalendar.getInstance()).
getTimeInMillis());
        String relatorio=tempo.toString()+"  USO DE CPU =
"+mediacpu+" ; USO DE MEM = "+mediamem;
        System.out.println(relatorio);

        if((mediacpu>limiteDeUsoDeCPU)|| (mediamem>limiteDeUsoDeMem))
        {
            return relatorio;
        }
        else
        {
            return null;
        }
    }

    public synchronized void atualizaUsoDeCPU(int id,double novoValor)
    {
        Double d=gridusodecpu.set(id,new Double(novoValor));
    }

    public synchronized void atualizaUsoDeMem(int id,double novoValor)
    {
        Double d=gridusodemem.set(id,new Double(novoValor));
    }
}

```



## **./aplicacaoteste/testerequisitor/src/requisitor.java**

```
////////////////////////////////////  
import java.io.*;  
import java.net.*;  
import java.sql.*;  
import java.util.*;  
  
public class requisitor  
{  
    public static void main(String args[])  
    {  
        System.out.println("teste requisitor iniciado!");  
        for(int i=0;i<1;i++)  
        {  
            (new Servico(i,20200+i,"localhost")).start();  
        }  
    }  
};  
  
////////////////////////////////////
```

## **./aplicacaoteste/testerequisitor/src/Servico.java**

```
////////////////////////////////////  
import java.io.*;  
import java.lang.Thread;  
import java.net.*;  
  
public class Servico extends Thread  
{  
    private int id;  
    private int porta;  
    private String ip;  
  
    public Servico(int i, int p,String ipe)  
    {  
        id=i;  
        porta=p;  
        ip=ipe;  
    }  
  
    public void run()  
    {  
        System.out.println("usando recurso "+id+"\n");  
        while(true)  
        {  
            try  
            {  
                Socket conexao=new Socket(ip,porta);  
                ObjectInputStream in=new ObjectInputStream  
(conexao.getInputStream());  
                ObjectOutputStream out=new ObjectOutputStream  
(conexao.getOutputStream());  
                if(in.readInt()!=74)  
                {  
                    System.out.println("Erro testando  
confiabilidade da conexao");  
                    break;  
                }  
                out.writeInt(47);  
            }  
            catch (Exception e)  
            {  
                e.printStackTrace();  
            }  
        }  
    }  
};  
  
////////////////////////////////////
```



## **./aplicacaoteste/testerealizador/src/realizador.java**

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

public class realizador
{
    public static void main(String args[])
    {
        for(int i=0;i<1;i++)
        {
            try
            {
                (new Recurso(i,new ServerSocket(20200+i))).
start();
            }
            catch (IOException ioe)
            {
                System.out.println("Erro no construtor do
ServerSocket");
                System.exit(-1);
            }
        }
    }
};
/////////////////////////////////////////////////////////////////
```

## **./aplicacaoteste/testerequisitor/src/Recurso.java**

```
/////////////////////////////////////////////////////////////////
import java.io.*;
import java.net.*;
import java.util.LinkedList;
import java.lang.*;

public class Recurso extends Thread
{
    private int id;
    private ServerSocket ss;

    public Recurso(int i,ServerSocket s)
    {
        id=i;
        ss=s;
    }

    public double recursoQueUsaProcessador(double xx)
    {
        double x=xx,y=1;
        for(int i=0;i<4000000;i++)
        {
            for(int j=0;j<1000;j++)
            {
                x*=y;
            }
            x*=x;
        }
        return x;
    }
}
```

```

}

public byte recursoQueUsaMemoria(double xx)
{
    LinkedList<Byte> llb=new LinkedList<Byte>();

    for(int i=0;i<1000000;i++) //1 mb por vez
    {
        llb.add(new Byte((byte)xx));
    }
    return llb.get(1000111);
}

public void run()
{
    System.out.println("usando Recurso "+id+"\n");
    while(true)
    {
        ObjectInputStream in;
        ObjectOutputStream out;
        try
        {
            Socket conexao=ss.accept();
            out=new ObjectOutputStream
(conexao.getOutputStream());

            out.writeInt(74);
            out.flush();
            in=new ObjectInputStream(conexao.getInputStream
());

            if(in.readInt()!=47)
            {
                System.out.println("Erro testando
confiabilidade da conexao");
                break;
            }

            double x=1.0;
            int j=1;
            boolean finaliza=false;
            while(!finaliza)
            {
                System.out.println("Realizando servico
"+j+++ e mandando resposta de volta");
                if(id==0)
                    out.writeDouble
(recursoQueUsaProcessador(x));
                else
                    out.writeDouble
(recursoQueUsaMemoria(x));

                out.flush();
                x=in.readDouble();
                finaliza=(x!=1.0);
                if(j>1000000)
                    j=0;
            }
            System.out.println("TUDO OK");
            conexao.close();
        }
        catch (IOException ioe)
        {

```

```
                System.out.println("IOException: Erro:  
verifique a rede ou hosts");  
                break;  
            }  
        }  
    }  
};
```

////////////////////////////////////

## **Anexo 2 – Artigo**

Segue, a partir da próxima página, o artigo requerido.



# Modelo de IDS Distribuído Para Grids Computacionais

Leonardo Kunrath

<sup>1</sup>Departamento de Informática e Estatística - Universidade Federal de Santa Catarina  
(UFSC)  
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brazil

[leok@inf.ufsc.br](mailto:leok@inf.ufsc.br)

**Resumo:** *Com o intuito de processar e manipular grandes quantias de dados através de aplicações de larga escala, utilizando recursos computacionais geograficamente distribuídos, as tecnologias de grid foram sendo desenvolvidas. Mas, à medida que a complexidade dos grids foi aumentando, também foram ampliadas as ameaças à segurança. Este trabalho faz um estudo das técnicas para detecção de ocorrências de intrusões, mais conhecidas como Intrusion Detection System (IDS), e sua aplicabilidade a ambientes de grids computacionais, uma vez que estas técnicas se apresentam como uma forma de se lidar com casos onde eventuais falhas na segurança vêm a pôr em risco o funcionamento de um grid computacional.*

**Abstract:** *To solve problems involving network management and the increasing quantity of data being transferred, and allowing new software to run in an environment of many computers, using geographically distributed computational resources, grid technologies were developed. But while the complexity of the grids increased, the security risks also increased. In this paper, we discuss intrusion detection systems (IDS) and their applicability in grid environments, and propose a new approach to increase grid security: the techniques of detecting intrusions in grids.*

## 1. Introdução

A grande melhoria que ocorreu no desempenho e nas tecnologias relacionadas à comunicação entre computadores fez com que a troca de dados entre computadores se tornasse algo fácil, rápido e comum. Esta realidade tornou possível o aparecimento dos grids computacionais, que têm como meta permitir um melhor aproveitamento de recursos computacionais, através do compartilhamento de recursos heterogêneos e geograficamente distribuídos, preocupando-se com qualidade de serviço e segurança.

O grid nasceu como uma infra-estrutura alternativa para as aplicações chamadas de *aplicações de grande desafio*, que até alguns anos atrás tinham como meio de execução as plataformas de computação paralela. O grid teve como papel inicial interligar estas plataformas, compostas por clusters e supercomputadores [Roure 2003].

Hoje, a computação em grid é caracterizada por aplicações construídas a partir de bases de software comum, conhecidas como *middleware*, que viabilizam os aspectos relacionados à comunicação entre computadores e outros aparelhos que possam estar

conectados ao grid, à segurança e à implementação do modelo de serviços, que permite organização e aproveitamento apropriado dos recursos computacionais disponíveis, entre outros aspectos.

Um grid computacional é, atualmente, uma tecnologia que está em grande expansão. Muitas aplicações comerciais e científicas já estão adotando o grid, no intuito de suprir a necessidade de processar e armazenar grandes massas de dados em uma quantidade limitada de tempo. Um exemplo é o projeto SETI@HOME [SETI@HOME 1996], que busca sinais de vida inteligente fora da Terra, recolhendo *terabytes* de dados através de telescópio e distribuindo estes dados para serem processados pelos computadores ligados no grid. No sítio do projeto, está disponibilizado o programa para que colaboradores possam fazer parte do grid e ajudar a processar esses dados. O interessante é que o programa entra em execução assim que o computador põe em funcionamento a proteção de tela, não atrapalhando a execução normal das atividades do colaborador.

Muitos estudos estão sendo realizados para a melhoria das tecnologias de grid em muitos sentidos: para o aumento do seu nível de segurança, tais como estudos de criptografia de dados, autenticação de usuários e máquinas, e detecção de intrusão; para a melhoria de desempenho, através de estudos de metodologias de garantia de QoS; uma melhor disposição e processamento de grandes massas de dados, utilizando-se de bancos de dados interligados aos grids; para a melhoria dos dispositivos móveis, que podem suprir sua baixa capacidade de processamento através da sua ligação com computadores, utilizando-se de grids. Todos estes aspectos tecnológicos referentes a grids estão em desenvolvimento e estão sendo amplamente estudados.

Um dos assuntos que estão sendo mais estudados em relação a grids computacionais é a segurança. O grid computacional, embora utilize abstrações, e seja organizado em módulos e frameworks oferecidos pelos middlewares, possui uma estrutura muito complexa. Além disso, ele utiliza pontos geograficamente distribuídos, o que torna o sistema suscetível a violações físicas de máquinas ligadas ao grid. As estruturas de segurança do grid têm que considerar tudo isso, agindo de forma a garantir várias características de funcionamento, tais como a integridade e confidencialidade dos dados transmitidos, bem como a disponibilidade dos recursos existentes.

Mesmo assim, as medidas de segurança não dão uma garantia total contra ataques. Para isso, foram desenvolvidos os estudos em detecção de intrusão, que têm se mostrado uma forma eficiente de detectar violações, recolhendo dados e tomando medidas de forma a evitar que o problema volte a acontecer, bloqueando o eventual invasor do sistema e gerando relatórios, para que os administradores da rede venham a ter conhecimento da invasão ou falha e possam tomar medidas para consertar o problema, como instalar atualizações ou corrigir o software.

A detecção de intrusão tem sido muito usada em servidores de redes, onde há a necessidade de manter o sistema funcionando ininterruptamente e atuando para garantir a velocidade e integridade na troca de pacotes na rede, para inúmeros usuários. Nesse sentido, os softwares de detecção de intrusão agem em sincronia com os softwares de bloqueio de conexões, conhecidos como firewalls.

Este artigo está organizado da seguinte forma: a seção 2 apresenta um estudo teórico sobre detecção de intrusão, a seção 3 sobre grids computacionais, a seção 4 aborda detecção de intrusão para grids, a seção 5 apresenta o modelo proposto, a seção 6 apresenta os resultados obtidos e na seção 7 são comentadas as conclusões.

## 2. Detecção de intrusão

A detecção de intrusão é uma área de segurança de sistemas computacionais. Ela consiste do monitoramento de características de um computador ou rede para detectar se um eventual intruso conseguiu invadir o computador [McHugh 2000].

Segundo [Debar 1999], podemos caracterizar os programas de detecção de intrusão, ou sistemas de detecção de intrusão (IDSs), de acordo com os seguintes critérios: método de detecção, comportamento ao detectar um ataque, e fonte dos dados analisados.

### 2.1 Tipos de IDSs

Os sistemas de detecção de intrusão são classificados de acordo com a forma com que descobrem se houve uma invasão. Podemos classificar os IDSs em:

- Baseados em comportamento: através de medições indiretas, como o uso de memória, CPU e espaço de armazenamento; o IDS ficaria observando as características do uso de cada recurso, procurando algum comportamento anormal, para assim descobrir um invasor. Em um servidor onde não se usa programas que precisam de muito processamento, por exemplo, quando se detectasse que o uso da CPU está acima de certa porcentagem, seria caracterizado que ocorreu uma intrusão no servidor, pois a CPU estaria sendo utilizada por um invasor, atrapalhando, assim, o funcionamento normal de seus programas e com risco de que danos venham a ser causados pelo invasor.
- Baseados em conhecimento: ao se verificar um acesso de uma máquina já conhecida como invasora, ou acesso a um recurso restrito por uma máquina não autorizada, sabe-se, com certeza, que se trata de uma invasão.

Pelo comportamento que os IDSs tomam quando detectam uma intrusão, podemos classificar eles em:

- passivos: apenas registram a invasão em um arquivo de dados, mostram na tela do computador ou enviam um alerta a um administrador;
- ativos: além de detectar e registrar a invasão, tomam ações para bloquear a ação do invasor.

Pelo local de busca dos dados para detectar a intrusão, podemos ter IDSs que buscam dados em:

- pacotes de rede: também chamados de *network based IDSs* (NIDS). Eles verificam, através dos pacotes, os remetentes dos mesmos e determinam, através do IP, se se trata de um invasor ou não;

- máquinas de uma rede: também chamados de *host based IDSs (HIDS)*. Eles verificam dados vindos das máquinas de uma rede, ou apenas da máquina onde estão funcionando, como uso de memória, CPU, e disco rígido, como também podem buscar o conhecimento sobre quais são os invasores que tentaram invadir as outras máquinas, para se prevenir contra eles.

- protocolos: alguns tipos de invasão se dão através de protocolos anômalos, que se passam por protocolos como Telnet, HTTP, RPC e SMTP, por exemplo, mas, através de modificações, conseguem passar por barreiras estabelecidas em níveis mais altos de abstração. IDSs deste tipo têm como rotina enviar pacotes através dos protocolos utilizados, com a finalidade de verificar protocolos com anomalias.

E, finalmente, quanto à frequência de uso, eles podem ser IDSs com:

- análise periódica: a cada período de tempo o IDS é acordado e verifica se houve invasão;
- monitoramento contínuo: o IDS fica ligado o tempo todo, checando cada pacote que o computador recebe. Um IDS com esta característica normalmente prejudica bastante o desempenho do computador onde ele está atuando.

Um dos programas mais representativos e mais usados para detecção de intrusão atualmente é o SNORT [SNORT 2000]. Este programa é um NIDS e seu funcionamento é organizado de acordo com regras. Quando o programa detecta uma possível intrusão, ele gera um alerta e registra as informações em arquivos, para eventual tratamento através de algum programa ou por um ser humano.

### **3. Grids Computacionais**

Os grids são uma evolução natural dos sistemas de computação distribuída. Desta forma, assim como nestes sistemas, eles funcionam através de uma rede, onde há uma gerência das mensagens trocadas entre os computadores e há o processamento de informações contidas nestas mensagens [Coulouris 2002] [Assunção 2003].

A idéia por trás do grid é o compartilhamento de recursos. Através do uso de uma rede, ou da internet, o grid é o ambiente que possibilita a interação entre os computadores, permitindo que eles compartilhem recursos, como processador, memória e periféricos, possibilitando que um grupo de computadores possa interagir como um conjunto com um poder computacional maior do que cada computador individualmente [Foster 1999].

As aplicações, para que funcionem em um ambiente de grid, devem estar preparadas para executar remotamente. Desde o início, para que estas aplicações tenham sucesso, os computadores remotos devem possuir todos os recursos necessários pela aplicação, podendo ser de software ou hardware [Bettoni 2004]. Ainda assim, não é para toda aplicação que o uso do grid vale a pena.

Muitas empresas já estão utilizando a infra-estrutura de grid para muitas de suas aplicações. Nos horários em que os computadores não estão sendo utilizados, fora do expediente ou de noite, eles podem e são utilizados pelos grids computacionais para as mais diversas aplicações, desde a organização de bancos de dados distribuídos,

processamento de vendas, preços de produtos, cálculo de valores de acordo com cotações de moedas ou mudanças de taxas.

Na figura 01, vemos como um grid pode interligar ambientes de supercomputação, clusters, centros de pesquisa, entre outros; de forma a se obter muitos recursos computacionais para todos os usuários do grid.

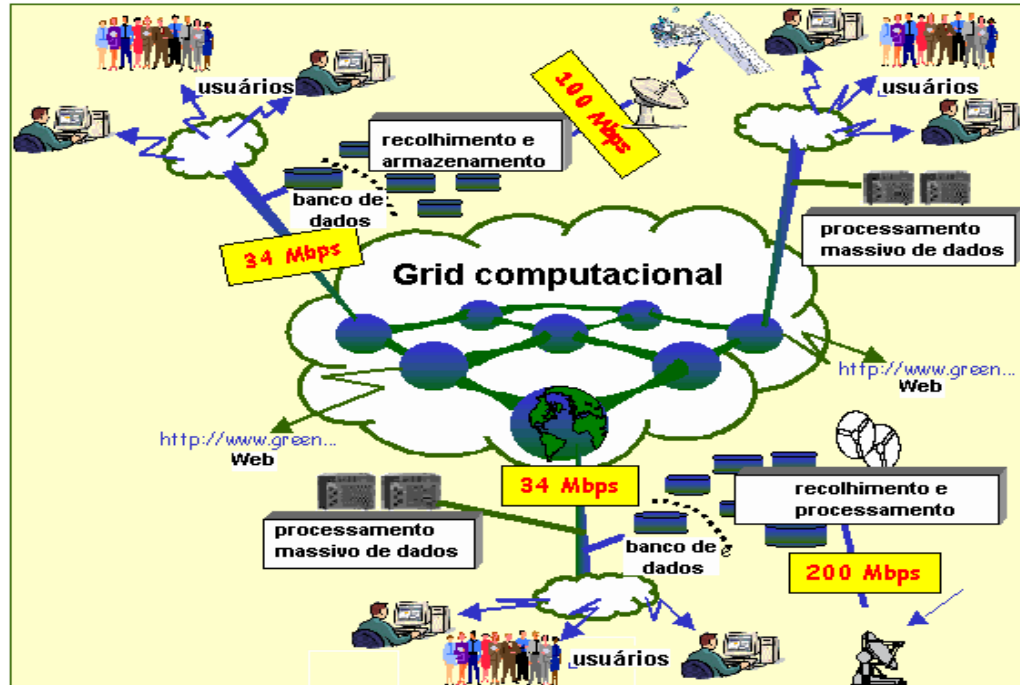


Figura 01 – Ambiente de um grid. A figura retrata a diversidade de um ambiente de grid.

### 3.1 Middleware

Os middlewares são camadas de software que têm como função ficar entre o sistema operacional e as aplicações de grid. O seu papel foi ganhando importância à medida que a abrangência dos grids foi aumentando, e foram acrescentadas novas características, como segurança, escalabilidade, adaptabilidade, entre outras.

O princípio dos middlewares é que, quando se deseja construir uma aplicação para grid, não é necessário sair do zero. A parte do grid que cuida das conexões, autenticação, segurança e serviços, por exemplo, funcionalidades necessárias em qualquer desenvolvimento bem estruturado para aplicações em grid, são as partes que os middlewares oferecem. Assim, no desenvolvimento de aplicações de grid, as atenções ficam voltadas para a aplicação em si, que trabalhará sobre um ambiente pronto e padronizado. Um dos middlewares mais estudados e utilizados da atualidade é o Globus [Globus 1996].

## 4. IDS para grids computacionais

A detecção de intrusão aplicada a grids computacionais é uma área que ainda não foi muito explorada, e necessita de estudos. Foram poucas as iniciativas neste sentido, como é o caso do SANTA-G [Kenny 2005].

Como ponto de partida, consideremos o seguinte cenário: em um determinado momento, durante o funcionamento do grid, um dos computadores pertencentes ao grid é invadido, ou seja, passa a ser controlado por uma pessoa que não foi autorizada a usar o computador. E, através do computador, o intruso consegue acesso aos recursos do grid.

Neste caso, o invasor poderia se utilizar de todas as vantagens do grid, enviando quaisquer tarefas para serem realizadas pelo grid, desde a tentativa de quebrar uma senha, até a de um ataque conjunto de negação de serviço a certo alvo. O invasor teria ao seu dispor todo poder computacional do grid, tanto para processamento quanto para armazenamento.

Nesse ponto é onde entra a aplicação de detecção de intrusão. Ela monitorará o uso de memória e processamento dos computadores do grid de forma que se possa identificar a presença de um invasor pela análise de seu uso do grid.

Em um grid pode-se estipular que cada usuário possa utilizar, no máximo, determinada porcentagem dos recursos disponíveis. Neste caso, se um invasor tentar utilizar o grid, ele provavelmente não conhecerá esta política de utilização, ultrapassaria a quantidade de recursos permitida e assim poderia ser identificado.

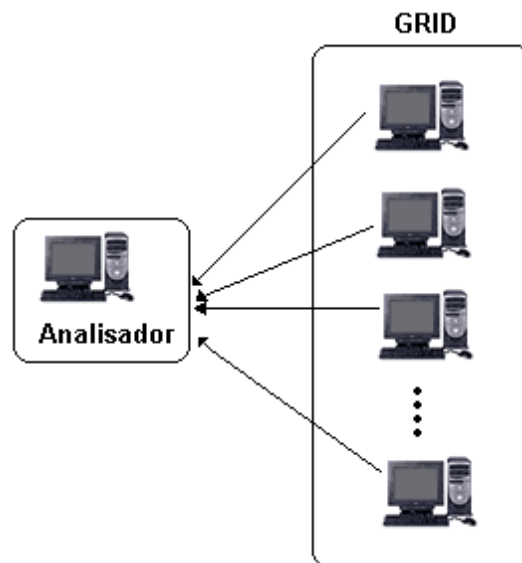
Outra situação seria a de um grid onde cada usuário deve mandar uma notificação de que vai utilizar o grid para o administrador da rede, indicando o horário em que estará utilizando. Um invasor, por não saber disso, não enviaria esta notificação e, desta forma, o detector de intrusão verificaria o uso do grid em um momento não previsto (não agendado), sendo assim o invasor descoberto.

Vale lembrar que há uma grande diferença na detecção de intrusão por análise comportamental em um único computador e a para um grid. Na primeira, analisa-se apenas as variáveis de sistema da máquina em questão para verificação se há comportamento malicioso. Na segunda, verifica-se as variáveis de sistema de todas as máquinas do grid ao mesmo tempo e, usando todos estes dados reunidos, verifica-se se o grid está sendo utilizado e em que proporção ele está utilizando os recursos disponíveis.

## **5. Modelo proposto de IDS distribuído para grids**

O modelo proposto funciona da seguinte forma: há um computador não pertencente ao conjunto de computadores do grid no qual um aplicativo executa, chamado neste trabalho de *Analizador*.

A função do *Analizador* é recolher os dados de uso do grid e processá-los no sentido de determinar se houve uma invasão ou não. O *Analizador* recolherá periodicamente os dados de uso de processador e memória de cada um dos computadores ligados no grid. Na figura abaixo, vemos como a aplicação Analizador está interligada com cada computador pertencente ao grid para, assim, recolher os dados do grid como um todo.



**Figura 02 – Analisador do modelo de IDS para grid.**

Depois de recolher estes dados de uso dos recursos processador e memória, o *Analisador* faz uma média de uso destes recursos para saber o grau de utilização do grid como um todo. No caso de este grau de utilização ultrapassar uma taxa pré-determinada, será considerado que o grid possivelmente foi invadido. Neste caso, o *Analisador* colocará em um arquivo as informações relevantes sobre esta invasão, como a hora em que aconteceu a invasão e o grau de utilização do grid encontrado.

Para que o *Analisador* possa adquirir as informações de uso de processador e memória de cada máquina do grid, é necessário que em cada uma destas máquinas esteja executando um programa que possa fornecer estas informações. Este programa foi chamado neste trabalho de *Coletor Local*.

O *Coletor Local*, como dito acima, é executado em cada máquina do grid, e fica enviando para o *Analisador* as informações de uso de memória e CPU da máquina onde estiver sendo executado. O *Coletor Local* não está diretamente ligado no middleware ou na aplicação de grid, apenas fica em execução no mesmo computador. Como a aplicação de grid consome os recursos do computador, o *Coletor Local* fica checando a proporção de uso destes recursos. Note que esta solução é ideal para um grid onde os computadores são dedicados exclusivamente ao grid, mas mesmo assim é uma solução ainda aplicável a casos onde não há esta exclusividade, uma vez que a média de uso de recursos do grid não será drasticamente alterada quando um host tem um de seus recursos utilizados por uma aplicação externa ao grid.

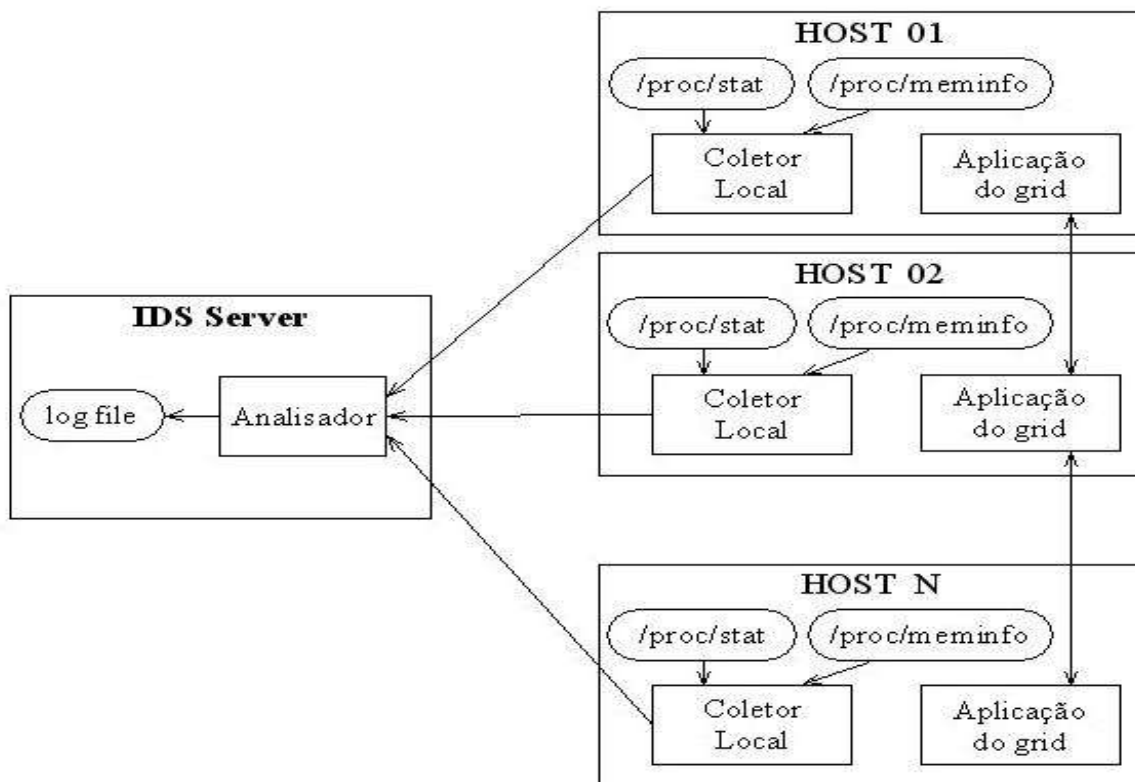


Figura 03 – Modelo completo de IDS distribuído para grids computacionais.

## 6. Resultados Obtidos

Para testar o *Coletor Local* e o *Analizador* foi criada uma aplicação distribuída que simula um grid. Esta aplicação recebeu o nome de *Aplicação Teste*, e sua função é simular uma aplicação em grid, utilizando os recursos de um conjunto de computadores, assim como um grid utilizaria estes recursos.

A escolha de uma aplicação que simula um grid, ao invés de um grid em si, foi feita no sentido de simplificar os testes, uma vez que o uso de um grid real não traria nenhum benefício adicional aos resultados que seriam obtidos, já que as características oferecidos por um middleware, tais como integridade, autenticação, criptografia, busca e escalonamento de recursos, entre outros, não seriam necessárias para a análise dos resultados.

Com a aplicação de detecção de intrusão para grids computacionais pronta, assim como a *Aplicação Teste*, foi montado um ambiente de testes para a validação do modelo proposto.

Os testes foram realizados com 2 e 3 computadores. Nestes, foi mantida em execução a *Aplicação Teste*, que simula o grid, e o *Coletor Local*. Em um outro computador, foi executado o *Analizador*, que ficou recolhendo os dados de cada *Coletor Local*, conseguindo, desta forma, dados sobre o uso do grid como um todo, e os analisando.



Foram testadas duas situações de funcionamento: uma com uso excessivo de processamento e outra com uso excessivo de memória. Em ambos os casos o sistema de detecção de intrusão cumpriu o que foi proposto: indicar os casos de uso excessivo dos recursos memória e processador.

## 7. Conclusões e trabalhos futuros

Em um grid computacional, identificar violações de segurança é uma tarefa muito árdua, já que o grid é uma estrutura muito complexa. A detecção de intrusão aplicada ao grid é um método que permite detectar ataques e, assim, determinar mais facilmente como o grid foi invadido.

O cenário que levou ao modelo proposto levou em conta o caso de um grid invadido e utilizado pelo invasor, uma vez que o grande atrativo de um grid computacional é o seu grande poder computacional.

O modelo proposto foi de se criar uma aplicação de detecção de intrusões que coletasse os dados de uso de memória e processador de cada computador do grid (*Coletor Local*), enviando estas informações para um único computador (*Analizador*). De acordo com este modelo, esses dados são tratados (pelo Analizador) e, quando é detectada uma condição no sistema de uso excessivo de memória ou processador, essas informações são armazenadas em um arquivo.

Uma contribuição ao trabalho realizado, que se encaixaria como trabalho futuro, seria uma aplicação que agiria depois que se tome o conhecimento de uma intrusão, tomando medidas de forma a banir o invasor do ambiente e garantir que ele não tivesse como invadir o grid novamente da mesma maneira.

Outra contribuição que poderia ser dada ao trabalho seria a expansão do algoritmo de detecção de intrusão. O modelo apresentado foi moldado baseando-se em comportamento do uso de recursos das máquinas do grid (host-based IDS). Além desse tipo de detecção de intrusão, existe o tipo baseado em informações de pacotes transitados na rede (network-based IDS). Essa outra contribuição seria dada pelo monitoramento dos pacotes da rede onde está o grid, monitorando as conexões, a fim de se ter mais informações utilizáveis para a detecção de intrusos.

## Referências

**ASSUNÇÃO, M. D.** (2003), “Implementação e Análise de uma Arquitetura de Grids de Agentes para a Gerência de Redes e Sistemas”, DISSERTAÇÃO (MESTRADO) – departamento do INE, UFSC, Florianópolis.

**BETTONI, C. A.** (2004), “Segurança de Agentes em Grids para Gerência de Redes de Computadores”, DISSERTAÇÃO (MESTRADO), departamento do INE, UFSC, Florianópolis.

**COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.** (2002), “Distributed Systems: Concepts and Design”, 772 p. Addison-Wesley, Boston.

**DEBAR, H.; DACIER, M.; WESPI, A.** (1999), “Towards a Taxonomy of Intrusion-

- Detection Systems”, 31. p. In: Computer Networks.
- FOSTER, I.; KESSELMAN, C.** (1999), “The Grid: Blueprint for a New Computing Infrastructure”, Morgan Kaufmann, San Mateo.
- GLOBUS.** (1996), “The Globus Alliance”, <http://www.globus.org/>.
- KENNY, S.; COGHLAN, B.** (2005), “Towards a Grid-wide Intrusion Detection System”, In: Proc.EGC’05, Amsterdam.
- MCHUGH, J.** (2000), “Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory”, In: ACM Transaction on Information and System Security, <http://www1.acm.org/pubs/citations/journals/tissec/2000-3-4/p262-mchugh/>.
- ROURE, D. et al.** (2003), “Grid Computing: Making the Global Infrastructure a Reality”, In: International Journal of Concurrency and Computation: Practice and Experience, New York.
- SETI@HOME.** (1996), “Search for extraterrestrial Intelligence”, <http://setiathome.ssl.berkeley.edu/>.
- SNORT.** (2000), “Snort – the de facto standard for intrusion detection/prevention”, <http://www.snort.org/>.