UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UMA ARQUITETURA DISTRIBUÍDA APLICADA AO TRATAMENTO DE REGISTROS DE SEGURANÇA DE REDE

MARCELO DIAS HOLTZ

ORIENTADOR: RAFAEL TIMÓTEO DE SOUSA JÚNIOR

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA
PUBLICAÇÃO: PPGEE.DM – 480/12
BRASÍLIA / DF: MARÇO / 2012

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UMA ARQUITETURA DISTRIBUÍDA APLICADA AO TRATAMENTO DE REGISTROS DE SEGURANÇA DE REDE

MARCELO DIAS HOLTZ

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISISTOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

RAFAEL TIMÓTEO DE SOUSA JÚNIOR, Doutor, ENE/FT/UnB
(ORIENTADOR)

ANDERSON CLAYTON ALVES NASCIMENTO, Doutor, ENE/FT/UnB
(EXAMINADOR INTERNO)

GEORGES DANIEL AMVAME - NZE, Doutor, FGA/UnB
(EXAMINADOR EXTERNO)

DATA: BRASÍLIA/DF, 09 DE MARÇO DE 2012.

FICHA CATALOGRÁFICA

HOLTZ, MARCELO DIAS

Uma arquitetura distribuída aplicada ao tratamento de registros de segurança de rede [Distrito Federal] 2012, xi, 73p.

(ENE/FT/UnB, Mestre, Engenharia Elétrica, 2012).

Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia. Departamento de Engenharia Elétrica

1. MapReduce 2. Cloud Computing

3. IDS 4. Honeypot 4. Hadoop

I. ENE/FT/UnB II. Uma arquitetura distribuída aplicada ao tratamento de registros de segurança de rede

REFERÊNCIA BIBLIOGRÁFICA

HOLTZ, M. D. (2012). Uma arquitetura distribuída aplicada ao tratamento de registros de segurança de rede. Dissertação de Mestrado em Engenharia Elétrica, Publicação PPGEE.DM – 480/12, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 73p.

CESSÃO DE DIREITOS

_____

Marcelo Dias Holtz
Universidade de Brasília
Faculdade de Tecnologia
Brasília – DF – Brasil. CEP: 70.910-900
holtz@unb.br , mdholtz@gmail.com

AGRADECIMENTOS

Agradeço a todos que contribuíram para esse trabalho, dedicando esforços, apoio e partilhando companheirismo. O conhecimento é construído de forma conjunta, assim como a amizade e confiança. A todos os meus sinceros agradecimentos!

DEDICATÓRIA

Dedico esse trabalho primeiramente aos meus pais, exemplos em minha vida. E a pessoas que estiveram presentes e deram sentido a frases como "Eu não quero sobreviver. Eu quero viver!" e "Just breathe".

RESUMO

UMA ARQUITETURA DISTRIBUÍDA APLICADA AO TRATAMENTO DE REGISTROS DE SEGURANÇA DE REDE

Autor: Marcelo Dias Holtz

Orientador: Rafael Timóteo de Sousa Júnior

Programa de Pós-graduação em Engenharia Elétrica

Brasília, março de 2012

A Internet tem se tornado um ambiente cada vez mais hostil, visto o crescimento dos ataques bem como a gravidade dos danos causados por eles. Sistemas de segurança como IDS e honeypot são componentes essenciais em um ambiente de rede seguro, permitindo proativamente a detecção de atividades maliciosas e ataques. Através das informações fornecidas por esses sistemas é possível aplicar contramedidas e mitigar os ataques que, por outro lado, poderiam comprometer seriamente a segurança da rede. No entanto, o atual crescimento do volume de tráfego de rede compromete a maioria das técnicas de IDS e Honeypot. Por isso, tais medidas de proteção requerem novas abordagens capazes de lidar com grandes quantidades de tráfego durante a análise, mantendo o desempenho e a escalabilidade. O presente trabalho propõe uma arquitetura de computação distribuída executada em nuvem para tratamento dos logs de segurança. A coleta descentralizada de dados realizada por vários sensores reúne informações abrangentes em vários níveis, dando um panorama completo do sistema monitorado. Toda a informação coletada de várias fontes é armazenada, processada e correlacionada de forma distribuída, em conformidade com o paradigma de computação MapReduce. A arquitetura proposta é capaz de lidar eficientemente com grandes volumes de dados coletados e altas cargas de processamento, sendo facilmente escalável, além de apresentar capacidade de detecção de ataques complexos através da correlação das informações obtidas a partir de diferentes fontes, identificando padrões que não seriam aparentes através da captura de tráfego centralizado ou da análise de logs de um único host. A pesquisa resultou ainda no desenvolvimento de um sistema totalmente distribuído de coleta, armazenamento e processamento de logs de segurança, sistema esse que se encontra operacional para tratamento e combate a intrusões e atividades maliciosas.

ABSTRACT

A DISTRIBUTED ARCHITECTURE FOR NETWORK SECURITY DATA ANALYSIS

Author: Marcelo Dias Holtz

Advisor: Rafael Timóteo de Sousa Júnior

Programa de Pós-graduação em Engenharia Elétrica

Brasília, march of 2012

The Internet is becoming an increasingly hostile environment as a consequence of both the growing attack volume and the resulting damage. Network security systems such as Intrusion Detection Systems (IDS) and honeypots are essential components of a secure network environment, allowing for early detection of attacks and malicious activities. Analyzing the data provided by these systems it is possible to react quickly to malicious activities by applying the necessary countermeasures and mitigating attacks that could seriously compromise network service integrity. However, the current growing volume of network traffic overwhelms most of current IDS and Honeypot systems, raising the need for new data analysis approaches capable of handling massive network traffic data with the required performance and scalability. In this work we propose a distributed architecture based on the cloud computing paradigm for analyzing security related data and logs. Building on a decentralized data collection mechanism based on network sensors distributed among different network nodes, our architecture analyses each network environment as a whole, providing a thorough view of network activity and potentially harmful actions. All the acquired data is stored, processed and analyzed in a distributed environment through the MapReduce framework. The proposed architecture can be efficiently scaled to handle high data storage and processing requirements. Furthermore, taking into consideration data collected from different portions and levels of the network, our architecture is capable of detecting complex distributed attacks that wouldn't be perceived by centralized network data collection approaches. This work resulted into a totally distributed and efficient data acquisition, storage and processing system for network security related data and logs, representing a milestone in network security monitoring and countermeasures against network intrusions and malicious activities.

LISTA DE TABELAS

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Desde os primórdios da interconexão de computadores em redes, trava-se uma constante batalha contra atacantes que, para atingir os mais diversos objetivos, buscam comprometer a segurança de sistemas computacionais. Seja sob o pretexto de defender uma ideologia ou simplesmente o de causar prejuízos a um concorrente, todos os dias grandes massas de ataques cibernéticos se voltam contra os mais diversos alvos. Muitas vezes estes ataques são basicamente inócuos, quando executados por neófitos. Porém, quando bem orquestrados, podem causar prejuízos na ordem de bilhões de dólares ou até mesmo ceifar vidas humanas.

A guerra cibernética e o crime digital já são uma realidade, com milhões de ataques sendo executados de forma a atingir metas financeiras e estratégicas específicas. Além de contribuir para um significante aumento no número de atividades maliciosas executadas, estes fatores socioeconômicos tornam ainda mais evidente a importância do estudo da segurança da informação, devido a seu impacto na sociedade atual.

Levando em conta a importância estratégica e econômica de vários recursos computacionais e informacionais, a comunidade profissional e acadêmica da área há muito busca técnicas de proteção e alternativas seguras para estas aplicações mais críticas. Muitas vezes é possível alcançar um bom nível de segurança apenas aplicando medidas preventivas e técnicas consideradas seguras, como ocorre em aplicações criptográficas e de segurança demonstrável. Porém, em muitos cenários, não é possível ter uma garantia de segurança forte o suficiente devido à natureza aberta de vários tipos de sistemas, onde não é possível restringir as ações executadas por usuários legítimos e adversários.

Para grande parte das aplicações de redes de computadores conectadas à Internet é extremamente complexo, se não impossível, prover soluções de segurança que eficientemente eliminem todos os riscos de ataques e ameaças existentes. Isto ocorre porque em muitos casos é difícil limitar a interação do mundo externo com os sistemas internos a uma organização, mesmo quando aplicadas medidas restritivas, como firewalls. Por exemplo, não existem ainda proteções eficazes contra ataques de negação de serviço, aos quais estão sujeitos quaisquer sistemas conectados à Internet. Portanto, há sempre um risco ao se implementar tais sistemas.

Também se deve considerar a questão da segurança de aplicações. Cada vez mais se observam falhas de segurança inerentes a aplicações mal programadas e não propriamente à rede ou sistemas com os quais tais aplicações interagem. Estas falhas podem ser provenientes de um simples erro de programação ou de falhas complexas de projeto lógico, potencialmente comprometendo não só a aplicação, mas também a rede utilizada para sua hospedagem. Muitas vezes falhas de aplicação passam despercebidas, sendo detectadas somente após serem exploradas resultando em um ataque bem sucedido. Vale notar que tais falhas expõem um dado sistema às mais diversas ameaças mesmo que as melhores medidas de segurança de rede sejam aplicadas.

Tendo em vista a dificuldade de se prover proteções suficientemente fortes para aplicações em rede, torna-se importante o estudo de técnicas de detecção de ataques. Através de métodos eficientes para detecção precoce de atividades maliciosas é possível tomar ações (automáticas ou não) para conter o ataque e os possíveis danos e efeitos deste sobre uma dada rede ou conjunto de sistemas. Desta forma, mitigam-se os riscos de um ataque bem sucedido ou que cause danos significativos.

Dentre os diversos métodos utilizados para se identificar atividades maliciosas executadas contra um dado sistema ou rede, destacam-se os chamados sistemas de detecção de intrusão (IDS – *Intrusion Detection Systems*) e os *honeypots*. Os IDS normalmente se localizam em um *host* específico ou na borda de uma rede, analisando todo o tráfego originado e recebido de forma a detectar ataques e anomalias. Já *honeypots* geralmente são constituídos por elementos que simulam sistemas legítimos vulneráveis (porém sem valor real para a organização), coletando informações sobre os ataques recebidos.

Os IDS utilizam diversas técnicas para fazer a análise automática do tráfego que coletam e detectar atividades suspeitas. Grande parte destas técnicas é baseada na detecção de padrões de tráfego que identifiquem certa atividade maliciosa. Estes padrões são denominados assinaturas de ataque, já que representam na prática uma marca própria deixada por um ataque específico em meio ao tráfego recebido ou enviado por um certo *host*. Consequentemente, a eficácia de um IDS é determinada pela qualidade e abrangência de seu banco de dados de assinaturas de ataques, já que um ataque só é detectado se sua assinatura estiver presente neste banco de dados.

Já os sistemas de *honeypot* são de grande valia para se coletar informações sobre ataques e determinar as técnicas mais utilizadas para se perpetrar crimes digitais sem expor sistemas de produção. Ao executar atividades maliciosas contra um sistema de *honeypot*, um atacante pensa estar atacando um sistema real da organização, mas está na verdade em um ambiente controlado onde tal atacante não pode causar danos ou roubar informações valiosas. Estes sistemas armazenam informações sobre todo o tráfego de rede recebido e a interação realizada com o atacante, permitindo posterior análise do ataque realizado.

1.1 DEFINIÇÃO E MOTIVAÇÃO

Neste trabalho consideramos o problema de se analisar de forma eficiente e escalável grandes massas de dados de tráfego de rede e registros de segurança com o objetivo de detectar ataques e atividades maliciosas e obter informações estatísticas sobre estes. Esse problema por sua vez divide-se basicamente nos seguintes pontos: coletar, armazenar e processar de forma escalável volumes massivos de dados provenientes de redes de grande porte em tempo hábil para se obter informações relevantes.

O principal problema aberto na área de detecção de atividades maliciosas para redes com altos volumes de tráfego (como as encontradas na maioria dos casos atualmente) é obter um sistema que consiga tanto armazenar quanto processar grandes massas de dados de tráfego de forma eficiente e escalável. Além do mais, não há sistemas com capacidade de coleta de dados totalmente distribuída, com possibilidade de extração de informação em vários níveis, desde redes heterogêneas a logs de sistemas operacionais de cada host.

O desempenho de soluções de IDS e *honeypot* é largamente afetado pela eficiência do algoritmo de procura de padrões utilizados. Os métodos de análise dos dados coletados por estes sistemas devem procurar em todo o tráfego capturado por cada padrão de assinatura contido em seus bancos de dados. Portanto, é necessário utilizar um algoritmo de pesquisa de padrões capaz de analisar eficientemente as grandes quantidades de tráfego capturadas. Além disso, o tempo de comparação de padrões deve ser pequeno devido à crescente quantidade de padrões que devem ser buscados para detectar a grande variedade de ataques executados.

Por outro lado, historicamente as informações provenientes de *honeypots* são analisadas manualmente por especialistas de forma a identificar as técnicas utilizadas, gerar

informações estatísticas e criar dados específicos de assinaturas de ataques. Porém, foram desenvolvidos nos últimos anos métodos automáticos de atribuição de ataque capazes de analisar dados de sistemas de *honeypot* e prover informações estatísticas e dados acerca da origem e técnicas utilizadas nos ataques. Muitas vezes estes métodos se baseiam em técnicas semelhantes às de sistemas IDS para detectar ataques através de assinatura e padrões presentes no tráfego, além de diversos métodos de análise estatística.

As atividades maliciosas perpetradas atualmente possuem complexidade cada vez maior, se valendo de novos avanços tecnológicos para burlar os mais avançados mecanismos de segurança. Ao mesmo tempo, a quantidade de informações necessárias para se identificar e classificar um ataque cresce rapidamente, assim como o poder de processamento necessário para interpretar estas informações.

Com o constante aumento de sistemas conectados à Internet e a escalada no volume de ataques, a quantidade de dados que deve ser analisada por sistemas automáticos de detecção de intrusão e tratamento de dados de *honeypot* cresce exponencialmente. Estes dados são extraídos de várias fontes, incluindo tráfego de rede e também registros gerados por sistemas operacionais. Observa-se então que os sistemas e técnicas clássicas de detecção de intrusão e análise de dados de rede não conseguem ser efetivos devido à enorme quantidade de dados a serem analisados.

Os métodos atuais de detecção de intrusão são em sua grande maioria baseados em *hosts* únicos para coleta e análise dos dados. Esta abordagem centralizada e serial para processamento de dados não é capaz de lidar com a crescente quantidade de dados a serem analisados em um processo de detecção eficiente e abrangente. Verifica-se que estes métodos se tornam sobrecarregados devido a limitações atuais tanto de software quanto de hardware, não existindo ainda uma solução trivial para adaptá-los ao novo cenário. É necessário então buscar alternativas escaláveis para realizar este tipo de processamento.

O processamento e armazenamento paralelos se apresentam como uma ótima alternativa aos métodos atuais de análises de dados de rede e detecção de assinaturas de ataques. Porém, como mencionado anteriormente, os métodos atualmente implementados não são trivialmente paralelizáveis. Além disso, ainda que o algoritmo em si seja paralelizável, a grande maioria dos arcabouços de programação paralela existentes acrescenta grande

complexidade no processo de paralelização de processamento e armazenamento, tornando inviável uma conversão direta do código existente.

1.2 TRABALHOS RELACIONADOS

O número de ataques contra sistemas de informação e redes cresce cada vez mais, assim como os danos causados. Vários sistemas de proteção foram criados e são utilizados buscando conter essas atividades maliciosas, tendo seu papel cada vez mais importante nas corporações. IDS e Honeypots são utilizados de forma complementar. Ambos geram uma enorme quantidade de dados, os quais sem uma análise detalhada perdem seu interesse para a segurança e prevenção.

Juntamente com o problema de coleta, outros problemas clássicos ressurgem sob novas formas e atualmente não possuem soluções adequadas. Em particular, com tantos dados sendo coletados, se impõe a questão de como armazenar e processar eficientemente toda essa informação, além da questão de como escalar o tratamento dessa demanda.

Tradicionalmente, os métodos de análise de logs de IDS e Honeypot são baseados na identificação de padrões de tráfego através do uso de ferramentas padrão do sistema Unix e scripts customizados (Maheswari, 2007). Esses métodos consistem basicamente em analisar diretamente o texto-puro de arquivos de log ou transferir os dados para algum banco de dados, de onde a informação estatística é extraída através de buscas customizadas. Devido ao enorme crescimento da quantidade dos dados coletados e das altas taxas de tráfego de rede, tais métodos tendem a serem ineficientes devido à falta de escalabilidade e às altas cargas de processamento.

Outras abordagens foram feitas se valendo do uso de algoritmos de mineração de dados clusterizados, como o DBSCAN (Ester, 1996). Em (Ghourabi, 2010) o DBSCAN foi utilizado para agrupar os pacotes capturados distinguindo o tráfego malicioso do normal. Em (He, 2008) foram utilizadas múltiplas séries de mineração de dados para analisar o fluxo de dados de modo a identificar anomalias e características de tráfego malicioso em ambientes geradores de grande massa de dados. Entretanto, ambos os métodos não conseguem extrair eficientemente dados estatísticos da análise, já que esses algoritmos não conseguem funcionar corretamente em um conjunto de dados com significativa diferença de densidade (Sang, 2008), além de necessitarem de muito espaço em memória para

carregar todo o conjunto de dados a ser analisado em memória principal (El-Sonbaty, 2004) e de não serem facilmente escaláveis.

Outras abordagens foram propostas buscando transferir o processamento e armazenamento dos dados coletados através do uso de agentes móveis (Guangchun, 2003), (Balasubramanuyan, 1998). A abordagem proposta em (Lee, 1998) baseia-se na coleta de dados por agentes móveis em diferentes hosts na rede e análise através de mineração de dados com o objetivo de detectar ataques e atividades maliciosas, obtendo assim bom balanceamento de carga e uma visão geral da rede analisada. Entretanto essas arquiteturas resolvem problemas relacionados ao armazenamento e processamento centralizado, mas em um cenário flexível e dinâmico, surgem vários problemas de segurança, como a autenticação da informação trocada entre os agentes móveis e a corrupção individual dos agentes por aplicações maliciosas como vírus e cavalos de tróia.

Outras abordagens interessantes são feitas por padrão rápido de reconhecimento através do uso de FPGAs (Field-programmable gate array) (Bu, 2004), (Sourdis, 2003), as quais são capazes de processar vários gigabits de dados por segundo, mas ainda não resolvem os problemas de armazenamento. Além disso, esses métodos são consideravelmente mais caros e menos escaláveis que as técnicas baseadas em software, uma vez que exigem um hardware construído propositalmente para esse fim.

Técnicas de processamento e análise de sinais foram propostas a fim de detectar atividades anormais através de análise de componentes principais (PCA) (Almotairi, 2008) e (Almotairi, 2009) através de algoritmos de MOS (Model Order Selection). Estes sistemas se baseiam em métodos manuais de análise de componentes principais para detectar ataques em meio a um conjunto de dados contendo os fluxos TCP de certo host. A lógica dessa abordagem consiste em assumir que os ataques são representados como componentes principais e as conexões e tráfego legítimos como ruído. Isto é motivado pelo fato de que o tráfego gerado por conexões e ataques direcionados a certa porta em certo host geram padrões que se assemelham a um sinal estruturado, enquanto que conexões legítimas a diferentes portas em instantes diversos de tempo se assemelham a ruído aleatório.

Apesar de introduzir a utilização de técnicas de MOS e PCA em sistemas de detecção de intrusão, os métodos propostos em (Almotairi, 2008) e (Almotairi, 2009) são

extremamente instáveis já que se baseiam em métodos de MOS arcaicos que requerem intervenção humana sendo, portanto, altamente sujeito a erros.

De forma a resolver este problema e obter um método de detecção automático de boa acurácia, foi proposto em (David, Costa, Nascimento, Amaral & Sousa Jr., 2011) um sistema de detecção de intrusão baseado no RADOI. Este método é automático e independente de qualquer intervenção humana, atingindo boa probabilidade de detecção.

Apesar de estes métodos apresentarem bons resultados e teoricamente serem paralelizáveis, ainda não existem implementações altamente paralelas dos algoritmos necessários para realizar a seleção de ordem do modelo. Portanto, a execução desses algoritmos torna-se um gargalo na utilização de métodos neles baseados para detecção de ataques em grandes massas de dados. Dada a necessidade de se executar tais algoritmos em um único núcleo de processamento local, estes métodos são inviáveis para conjuntos de dados com tamanho acima de algumas centenas de megabytes.

Mesmo que se obtenha uma versão paralela destes algoritmos ainda é necessário observar que a gerência de memória necessária para se executar estes métodos sobre conjuntos de dados com centenas de gigabytes é extremamente complexa. Uma alternativa teórica para contornar este problema foi proposta em (David, Costa, Holtz, Amaral & Sousa Jr., 2012). O método proposto consiste em dividir o processamento dos dados coletados entre cada um dos hosts responsáveis pela coleta, requerendo que somente os autovalores resultantes sejam enviados a um host central para serem agregados e se obter o resultado final. Apesar de este método ser viável na teoria, ainda não existem implementações funcionais correspondentes ou resultados práticos que demonstrem sua viabilidade, com exceção de resultados de simulação.

Além das questões apresentadas anteriormente, os métodos de detecção de intrusão baseados em seleção de ordem de modelo e análise de componentes principais têm dificuldades para levar em consideração também os dados coletados pelos sistemas operacionais e aplicações nos hosts, limitando-se somente dados de tráfego de rede. Especificamente, a dificuldade encontra-se no fato de que tais métodos não permitem incluir dados de logs de sistemas operacionais e aplicações na análise, de forma a correlacioná-los com dados de tráfego coletado.

Uma nova abordagem usando o MapReduce para analisar o tráfego de internet foi proposta em (Lee, 2010) e como resultado mostrou ser possível obter eficientemente dados estatísticos de grandes conjuntos de dados puros. As primeiras abordagens para analisar conjunto de dados de IDS foram propostas independentemente em (Yang, 2011) e (Holtz, David & Sousa Jr., 2011) e consistem na transferência do armazenamento e processamento para um ambiente de nuvem com o intuito de obter melhor eficiência, escalabilidade e poder de processamento. A abordagem em (Holtz, David & Sousa Jr., 2011) se difere por propor adicionalmente uma arquitetura de coleta distribuída de dados, a qual fornece maior compreensão e um panorama mais abrangente das atividades de rede através de diferentes sistemas individuais.

Em (Holtz, David, Peotta & Sousa Jr., 2011) foi proposta uma plataforma escalável de IDS distribuído, além de agregar tratamento de outros sistemas de segurança, como os *honeypots*, enriquecendo a análise através de correlação dos arquivos, obtendo assim informações cruciais que até então não eram possíveis de forma eficiente, prática e escalável, resolvendo os problemas levantados anteriormente, como processamento e armazenamento eficientes no tratamento de grandes massas de dados e será objeto de um estudo mais aprofundado nesse trabalho.

1.3 CONTRIBUIÇÕES

Nesta subseção são expostas as principais contribuições do presente trabalho.

A maioria dos sistemas de detecção de intrusão atuais é baseada em *hosts* únicos localizados em pontos estratégicos da rede para coletar dados. Isto representa um problema, já que estes sistemas somente consideram informações coletadas por cada um dos *hosts* individualmente, não obtendo um panorama geral das atividades de rede que ocorrem em diferentes pontos desta. Para resolver este problema, o presente trabalho propõe um sistema distribuído de detecção intrusão baseado em múltiplos sensores localizados em diferentes pontos da rede, que coletam e enviam os dados (incluindo tráfego de rede e *logs* de sistemas operacionais) para um sistema de arquivos distribuído. Após a coleta dos dados, estes são analisados e correlacionados entre si para obter informações e padrões que permitam a clara identificação de ataques e atividades maliciosas. Este sistema é capaz de detectar atividades maliciosas que não são evidentes a partir de um ponto único da rede, mas que podem ser percebidas quando a rede como um

todo é observada (por exemplo, *core melt attacks*). Não há na literatura atual sistemas IDS totalmente distribuídos, eficientes e escaláveis em termos de coleta, armazenamento e processamento de dados. A viabilidade deste sistema é estudada através de diversos experimentos de simulação onde se afere a velocidade de processamento dos dados e a velocidade de operações do sistema de arquivos distribuído para diferentes tamanhos de dados. Estes experimentos também são independentemente interessantes, já que demonstram o desempenho do sistema de arquivos distribuído utilizado e do paradigma MapReduce.

Os sistemas atuais de análise estatística de dados coletados em *honeypots* são baseados em algoritmos para análise local dos dados coletados, não sendo capazes de lidar com as grandes quantidades de dados coletados atualmente. Assim, é preciso dividir os dados a serem analisados em vários subconjuntos, tornando demorado e custoso o processo de se obter estatísticas gerais. De forma a resolver este problema, o sistema proposto e implementado é capaz de prover informações estatísticas acerca dos ataques e dados de rede coletados por *honeypots*. O sistema provê informações como: lista das principais origens de ataques, total de ataques por períodos de tempo (como meses ou anos) e serviços mais atacados. A implementação deste sistema foi testada utilizando dados reais provenientes de um *honeypot* operando na rede de uma grande instituição bancária da América Latina. O sistema alcança boa velocidade e apresenta ótima escalabilidade. Ele também pode ser utilizado como bloco construtor para sistemas mais complexos de atribuição de origem e identificação de novos ataques.

Estas contribuições representam um importante avanço para o campo de detecção de intrusão, análise de dados de *honeypots* e atribuição de ataques (*attack attribution*), já que permitem o processamento de grandes massas de dados de forma transparente e escalável. Sendo implementados em plataformas cuja eficiência já foi atestada por diversas aplicações de grande porte nas maiores empresas da indústria de Internet no mundo, estes sistemas podem ser aplicados em situações do mundo real e utilizados para prover melhor segurança para grandes redes corporativas e *backbones*. Ressalta-se que estes sistemas podem ser utilizados como subsistemas na construção de métodos mais complexos de atribuição de ataque e análise de dados para gerência de redes. Os resultados apresentados neste trabalho também caracterizam uma aplicação inovadora do paradigma de computação em nuvem baseado em MapReduce.

Em especial, a arquitetura proposta para sistemas distribuídos de detecção de intrusão apresenta um marco significativo no projeto de tais sistemas, traçando novas diretrizes para o planejamento e implementação de sistemas IDS adaptados à nova realidade da Internet e à computação em nuvem. Tirando proveito das capacidades de processamento e armazenamento ampliadas pelo paradigma de computação em nuvem, é possível construir sistemas de prevenção e detecção holísticos para as redes atuais, que se apresentam com complexidade e tamanho cada vez maiores. Logo, esta proposta de arquitetura abre o caminho para o projeto e implementação da próxima geração de sistemas de detecção de intrusão, que devem estar prontos a atender as altas e crescentes demandas das redes de comunicação atuais.

Como trabalhos futuros destaca-se principalmente a construção de um sistema de atribuição de ataque baseado em paradigma MapReduce e a implementação completa da arquitetura para sistemas distribuídos de detecção de intrusão proposta neste trabalho. A obtenção de sistemas eficientes de atribuição de ataques que ofereçam escalabilidade para grandes quantidades de dados é um importante problema aberto da literatura atual. Aplicar o paradigma MapReduce para paralelizar e tornar escaláveis os algoritmos utilizados atualmente e propor novos algoritmos otimizados para este paradigma é uma abordagem promissora para resolver esta questão.

1.4 ORGANIZAÇÃO DO TRABALHO

O presente trabalho encontra-se estruturado em seções que buscam de maneira lógica guiar o leitor, e de forma didática expor a pesquisa realizada.

A seção 2 traz conceitos e fundamentações necessários ao entendimento da pesquisa. Nela são abordadas as principais características e definições dos sistemas de segurança como IDS e *honeypot*, suas vantagens e desvantagens, topologias, valor agregado e etc. Na mesma seção são descritas as principais características da plataforma Hadoop assim como as implementações do framework MapReduce e sistema de arquivos distribuído, o HDFS. Como todo o ambiente é estruturado utilizando computação em nuvem, esse tema é abordado em uma subseção que descreve as principais características desse relativamente novo ambiente computacional.

A seção 3 aborda a topologia proposta na pesquisa, descrevendo quais aspectos inovadores do modelo de computação distribuída para tratamento de logs de segurança. Na seção seguinte, temos uma abordagem prática do que foi proposto, através de implementações e resultados que validam a pesquisa.

2. CONCEITOS E FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão abordados os conceitos e fundamentos de alguns sistemas de segurança que auxiliam na detecção de intrusão e análise de invasores na rede, abordando com mais detalhes os sistemas Honeypots e de Detecção de Intrusão (IDS). Será destacada a importância de cada sistema, analisando as vantagens e desvantagens. Mostra-se que, utilizados em conjunto, tais sistemas conseguem suprir suas deficiências, gerando logs de extrema importância, que se analisados da forma correta, produzem informação valiosa na prevenção, detecção e resposta a incidentes, garantindo a integridade e segurança da informação do ambiente que se utiliza desses meios de segurança. Além disso, nas próximas subseções são abordados as características do sistema Hadoop, bem como as implementações do framework MapReduce e o sistema de arquivos distribuído e suas características, o HDFS, se valendo do ambiente de computação em nuvem.

2.1. SISTEMAS DE SEGURANÇA

Várias novas ameaças surgem constantemente na internet criando uma demanda cada vez crescente na busca de mecanismos mais eficientes para detecção e identificação de atividades hostis. O custo das empresas com ataques virtuais cresceu cerca de 56% no ano de 2011 de acordo com um estudo feito pelo Instituto Ponemon. O levantamento apontou um impacto financeiro médio anual de US$ 5,9 milhões nas companhias pesquisadas tendo perdas contabilizadas de US$ 35 milhões. O número de ataques aumentou quase 45%, chegando a ter 72 ações semanais. Os impactos foram causados, em sua maioria, por códigos maliciosos, negação de serviços, roubo ou invasão de dispositivos e ataques baseados na internet. O maior custo das companhias está relacionado à detecção de invasores e à recuperação dos dados perdidos ou danificados.

O tempo médio para a resolução de um ataque cibernético subiu quatro dias em relação à última pesquisa, para dezoito dias, sendo que o custo médio para cada remediação foi de US$ 416 mil, alta de 70% em um ano. Segundo o presidente do instituto, Larry Ponemon,

o impacto econômico dos ataques sobe à medida em que as ações ficam mais sofisticadas e mais frequentes.

Os ataques às empresas ganharam destaque neste ano depois que hackers conseguiram acessar informações pessoais de mais de 70 milhões de usuários conectados à rede da Sony, causando prejuízo à gigante de tecnologia japonesa. Grupos como LulzSec e Anonymous também intensificaram ataques contra governos e sites públicos e invasões chamaram atenção para a existência de brechas de segurança em páginas e sistemas públicos e privados.

Diante desse ambiente de extrema importância e rápida transformação, sistemas de segurança foram criados para evitar ou pelo menos diminuir os riscos ou danos com ataques que comprometam os dados e sistemas de uma dada entidade. Nesse presente trabalho, iremos abordar o uso de Honeypots e sistemas de detecção de intrusão.

2.2. HONEYPOTS

Embora diferentes definições existam, um honeypot pode ser definido como um recurso de sistema de informação cujo valor reside em uso não autorizado ou ilícito de um recurso (Spitzner, 2003A). Honeypots se destinam a ser ativamente sondados e alvo de ataques, em vez de agir passivamente, como os sistemas de detecção de intrusão e firewalls. Em outras palavras, é uma ferramenta de estudo de segurança, cuja função principal é deter atacantes, detectar ataques, capturar e analisar ataques automatizados, como worms, além de fornecer informações importantes sobre a comunidade hacker. É um sistema que possui falhas de segurança, reais ou virtuais, colocadas de maneira proposital, a fim de que seja invadido e que o fruto desta invasão possa ser estudado.

Um honeypot tem como finalidade coletar códigos maliciosos, identificar varreduras e ataques automatizados, acompanhar as vulnerabilidades, motivar os atacantes, correlacionar informações com outras fontes, auxiliar os sistemas de detecção de intrusão, e manter atacantes afastados de sistemas importantes, desviando sua atenção para sistemas falsos.

2.2.1. Tipos de Honeypot

As pesquisas envolvendo honeypots e honeynets são baseadas em três áreas principais: Honeytokens, Honeyfarms e Honeypots Dinâmicos. (Mokube, 2007).

- Honeytokens: Conceito introduzido primeiramente por (Paes, 2003), essa abordagem trata-se de que qualquer informação falsa pode ser criada a fim de detectar uso não autorizado de algum sistema ou informação. Ao invés de se implementar servidores como iscas, implementam-se informações como iscas. Um usuário falso criado com uma senha extremamente fraca, documentos de texto e planilhas com informações falsas tratadas como confidenciais, tabelas criadas em banco de dados onde o acesso somente é feito por *Stored Procedure,* todos esses procedimentos podem ser tratados como honeytokens, cujo valor está na flexibilidade, tendo em vista que abre uma gama de possibilidades de verificação de atividades hostis, sendo utilizados principalmente para a detecção de atacantes internos, como por exemplo, funcionários com intenções maliciosas.
- Honeyfarms: A medida que as redes corporativas aumentam, com várias redes distribuídas pelo mundo, o uso de honeypots passa a ser inviável devido o alto custo de profissionais e recursos, para implementação e acompanhamento. Como solução para esse problema, o uso de honeyfarms simplifica esse processo, uma vez que todos os honeypost são concentrados em um único lugar. Toda atividade maliciosa existente na rede é direcionada para esse *pool* de honeypots, facilitando assim a gerência e análise dos dados além de viabilizar os custos da implementação. A figura 2-1 descreve a topologia de um honeyfarm.

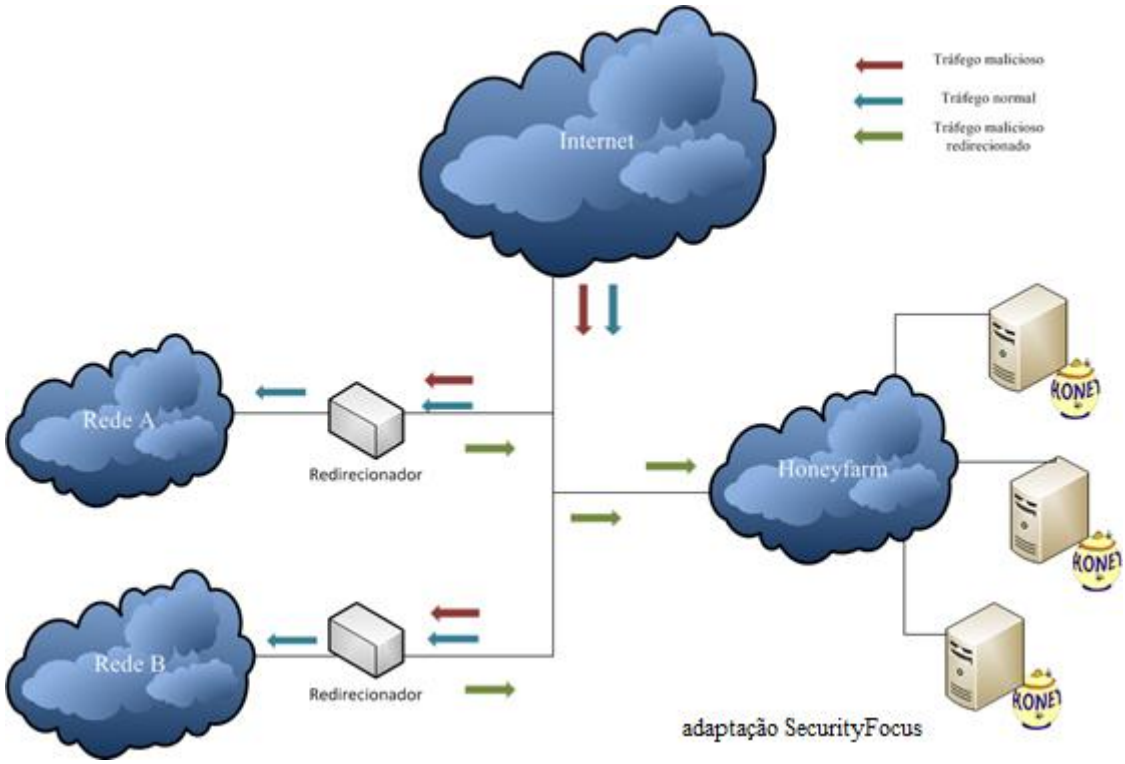


Figura 2-1 - Topologia de uma honeyfarm

- Honeypots Dinâmicos: Esse conceito foi introduzido em (Spitzer, 2003B). Devido a grande variação do ambiente a ser simulado pelos honeypots, surgiu a necessidade de um sistema auto adaptativo, que analise as mudanças ocorridas no ambiente e que da mesmo forma incorpore-as para o honeypot, como por exemplo: caso um novo servidor Windows seja criado na rede, o Honeypot dinâmico irá verificar essa mudança e passará a simular um novo Windows, trazendo assim maior facilidade de manutenção e assegurando que o sistema de segurança, de forma automática esteja sempre atualizado em relação ao sistema real de produção, porém esses honeypots ainda objetos de pesquisa, tendo em vista a sua complexidade de criação e a criação de tecnologias que viabilizem esse conceito.

2.2.2. Classificação dos Honeypots

Os honeypots são classificados de acordo com níveis em que podem ser interagidos pelos atacantes e pelo tipo de finalidade que foram criados, sendo estes Honeypots de pesquisa ou de produção. De acordo com o nível de envolvimento, podem ser divididos em duas categorias principais, de baixa e alta interação.

- Baixa interação: São sistemas mais simples, onde os serviços ou sistemas operacionais são emulados. Eles possuem facilidade de implementação e um menor risco de comprometimento, porém coletam menos informação de um atacante. Por serem mais utilizados em ambientes corporativos, onde há a necessidade de rápida intervenção e facilidade de execução, esses honeypots são comumente denominados honeypots de produção. Exemplo de honeypot de baixa interação: Specter, Honeyd, LaBrea, etc.
- Alta interação: Proveem um nível muito maior de interação, com sistemas operacionais e serviços reais. A complexidade de implementação desse sistema é muito maior, porém ele é de difícil detecção por parte dos atacantes por se assemelhar ao ambiente real de produção. Nesse modo, a informação coletada dos atacantes traz um detalhamento muito maior e assim é possível estabelecer um estudo mais aprofundado do perfil do atacante, das ameaças analisadas e dos alvos que eram pretendidos, obtendo dados extremamente valiosos que irão contribuir para aumentar a segurança da entidade. Justamente por esse nível de detalhamento e possibilidade de estudo, esses honeypots também são conhecidos como honeypots de pesquisa. A tabela 2-1 descreve a divisão de honeypots de acordo com o nível de interação.

Tabela 2-1 - Classificação de Honeypots

Baixa interação	Alta interação
Emulação de sistemas operacionais e serviços	Solução provê sistemas operacionas e serviços reais
Fácil instalação e manutenção	Maior complexidade na instalação e manutenção
Risco de comprometimento reduzido	Maior risco de comprometimento, tendo em vista o uso de um ambiente real
Coleta mais limitada de dados	Maior interação com o atacante gera maior quantidade e detalhamento dos dados coletados.

2.2.3. Vantagens, desvantagens e o valor dos honeypots

De acordo com as características já apresentadas, os honeypots são importantes mecanismos no combate a ataques e atividades maliciosas na rede. Como não possuem valor real de produção, qualquer interação com o honeypot pode ser considerada como uma tentativa hostil, diminuindo assim a enorme quantidade de falsos positivos, aumentando a confiabilidade dos logs coletados (Alata, 2006). Podemos destacar:

- São mecanismos simples, não necessitando de algoritmos complexos para a sua execução.
- Consomem recursos mínimos de hardware.
- São eficazes para novos ataques, uma vez que não precisam de padrões de assinaturas de ataques, contrário aos IDS.
- Honeypots funcionam bem em sistemas criptografados e em IPv6.

Em contrapartida, esses sistemas apresentam algumas limitações como:

- Campo de visão limitado, uma vez que só conseguirão registrar um ataque caso o atacante interaja com o honeypot.
- Há o risco de retomada do controle do sistema do por atacante e utilizando-o assim para comprometer os sistemas reais de produção, risco esse medido de acordo com o nível de interação do honeypot.

Os honeypots podem auxiliar na prevenção de ataques automatizados, que utilizam-se de ferramentas que escaneiam aleatoriamente toda uma rede a procura de vulnerabilidades. Dessa forma o sistema irá interagir com o atacante, confundindo-o, retardando e abrandando o ataque e as consequências para a organização. Toda a interação com o honeypot é registrada. Os logs coletados servirão de informação que poderá então ser usada para uma variedade de finalidades, incluindo análise de tendências, identificação de novas ferramentas ou métodos, identificação de atacantes e suas comunidades, cumprimento da lei, além de alerta precoce e de previsibilidade ou motivações.

Para aumentar o nível se segurança de uma empresa, os honeypots deverão ser utilizados em conjunto com outros mecanismos complementares, como os IDS. Esses sistemas atuando em conjunto, conseguem preencher as lacunas de suas limitações e auxiliar em todo o método de detecção, proteção e resposta a incidentes. A próxima subseção destina-se a analisar os sistemas de detecção de intrusão.

2.3. SISTEMAS DE DETECÇÃO DE INTRUSÃO - IDS

Sempre se faz necessário para uma boa política de segurança, termos ferramentas e dispositivos de segurança complementares a fim de garantir proteção aos dados de uma corporação.

Sendo o meio físico compartilhado, a probabilidade de que pessoas não autorizadas consigam acesso às informações sigilosas através de métodos ilícitos é grande, além disso, a simplicidade na obtenção de ferramentas automatizadas e códigos-fonte exploradores de vulnerabilidades de sistema é cada vez maior. Sistemas de detecção de intrusão, utilizados em conjunto com honeypots e firewalls aumentam consideravelmente o nível de proteção de uma rede, desde que estejam bem configurados.

Esses sistemas, idealizados em (Anderson, 1980) monitoram estações ou fluxos de rede a fim de descobrir atividades maliciosas. Quando um ataque ou uso impróprio é detectado, um alerta é gerado indicando que o sistema poderá ser comprometido, funcionando analogamente como um alarme de segurança gerado quando um ladrão invade determinado local protegido.

Em (Denning, 1987) foi proposto o primeiro modelo de detecção de intrusão, com a hipótese de que violações de segurança poderiam ser detectadas por monitoração de registros de um sistema de auditoria de padrões anormais de utilização do sistema. A partir desse modelo, novos tipos foram propostos, alterando a forma de detecção e topologias adotadas.

2.3.1. Formas de detecção

Basicamente os IDS utilizam duas formas principais para detectar as intrusões e ataques (Axelsson, 2000), (Debar, 1999). São elas:

- **Detecção por assinatura**: Nessa forma de detecção, as atividades do sistema são analisadas e comparadas com padrões pré-definidos de ataques e outras atividades maliciosas. Esses padrões são conhecidos como assinaturas. A tabela 2-2 descreve as vantagens e desvantagens da detecção por assinaturas.

Tabela 2-2 - Detecção por assinaturas: vantagens x desvantagens

Vantagens	Desvantagens
Eficiência: não gera um número esmagador de alarmes falsos.	Detectam apenas ataques pré-definidos em sua base de assinaturas e dessa forma, devem ser constantemente atualizados para não perder a sua eficácia.

De forma rápida e confiável é possível diagnosticar o uso de ferramentas de ataque ou técnicas específicas. Isso ajuda os gestores de segurança a priorizar medidas corretivas,	A maioria dos IDS que utilizam essa técnica, não consegue detectar variantes de ataques já conhecidos em sua base de assinaturas.
Permite as gerentes do sistema, independentemente do seu nível de especialização em segurança, monitorar e realizar procedimentos de contingência de ataques.	

- **Detecção por anomalias**: Esse tipo de detecção parte do princípio que atividades maliciosas e ataques tem ações diferentes das atividades padrão do sistema. Limites são definidos como admissíveis para uma atividade normal, como por exemplo, o número de arquivos acessados por um usuário em determinado período de tempo, o número de tentativas fracassadas de acesso ao sistema, quantidade de CPU utilizada por um processo, sendo esse limites definidos de forma estática ou heurística. Assim, perfis são montados correspondendo a atividades normais, em relação à usuários, hosts, sistemas, conexões de rede. Dado essas métricas comportamentais, atividades que destoem desse perfil são classificadas como ataques gerando alertas de atividades hostis (Porras, 1997). A tabela 2-3 descreve as vantagens e desvantagens da detecção por anomalias.

Tabela 2-3 - Detecção por anomalias: vantagens x desvantagens

Vantagens	**Desvantagens**
Permite a detecção de sintomas de ataques sem conhecimento específico dos detalhes do ataque sofrido.	Geralmente produzem um grande número de falsos positivos devido ao comportamento imprevisível de usuários e redes.
Produzem informações importantes que podem ser utilizadas para criação de novas assinaturas de ataques.	Necessitam de um extenso conjunto de registros de eventos do sistema para que os perfis de comportamento sejam definidos.

Em ambas as detecções, técnicas de redes neurais são utilizadas buscando-se melhorar a performance desses IDS (Ghosh, 1999). Assim o sistema torna-se adaptativo e mais imune a falsos positivos.

2.3.2. Tipos de IDS

Os IDS são comumente divididos de acordo com a topologia onde se encontram coletando e analisando dados para verificar tentativas de comprometimento do sistema. Dividem-se em IDS baseados em rede (NIDS – network intrusion detection systems) e baseados em estação (HIDS – host-based intrusion detection systems).

- NIDS: Esses IDS detectam ataques através da captura e análise de pacotes de rede. Através da escuta e dessa análise de toda a rede é possível verificar ataques a múltiplos hosts. Na figura 2-2 temos um exemplo de NIDS. A tabela 2-4 descreve as vantagens e desvantagens de um NIDS.

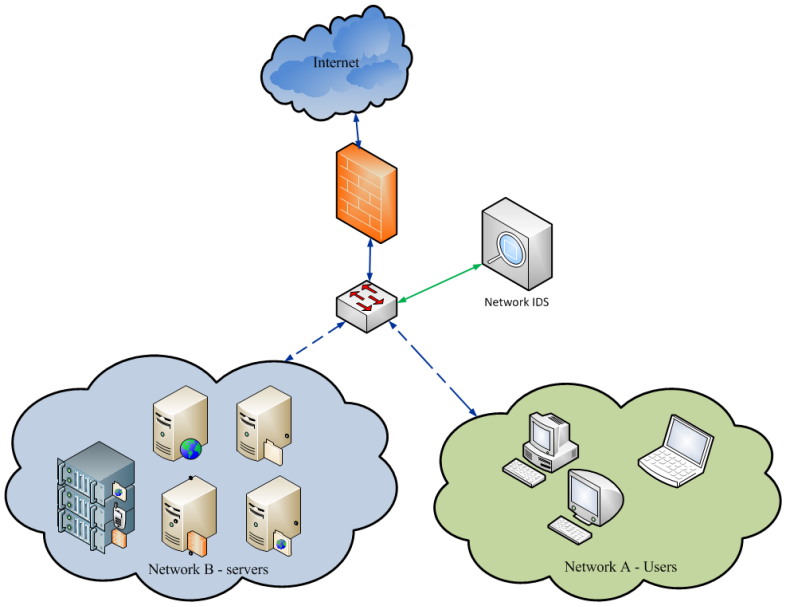


Figura 2-2- Topologia de um NIDS

A tabela abaixo faz um comparativo entre as vantagens e desvantagens desse sistema:

Tabela 2-4 - NIDS: Vantagens x Desvantagens

Vantagens	Desvantagens
Se bem localizado consegue monitorar uma	Dificuldade de processar todos os pacotes

grande a rede.	em grandes redes de tráfego intenso.
Rápido *deploy*. São dispositivos passivos na maioria das vezes, logo não necessitam de grandes adaptações, além de caso seja necessário um *rollback*, não traz grandes impactos para a rede.	Em redes baseadas em switches há a segmentação da própria rede, promovendo links dedicados entre os hosts e os serviços através do próprio switch. Problema resolvido com switches gerenciáveis, através da opção de espelhamento de porta, mas infelizmente não são todos os switches que possuem essa funcionalidade.
São seguros contra ataques e conseguem passar na maioria das vezes de forma "invisível" para os atacantes.	Não conseguem analisar dados criptografados. Problema agravado com o uso de VPNs.
	A maioria desses IDS não conseguem dizer se o ataque obteve sucesso ou não, apenas discernem quando o ataque foi iniciado. Assim os investigadores devem investigar manualmente qual host foi atacado e se foi na verdade comprometido.
	Problemas com ataques de rede que envolvem fragmentação de pacotes. Os pacotes malformados podem causar instabilidade no IDS e fazer com que ele entre em colapso.

- HIDS: Esses sistemas trabalham coletando e analisando dados internamente à máquina e não da rede, como o NIDS. Informações como registros e logs do sistema operacional, alteração no sistema de arquivos e assim por diante, aumentando a precisão da detecção dos ataques, diminuindo o número de alarmes falsos. Todos os hosts que deverão ser monitorados precisam de agentes locais instalados, podendo existir um host centralizador para gerenciar todas as máquinas

monitoradas, diferentemente do NIDS em que se utiliza normalmente apenas um IDS para toda a rede. A figura 2-3 que exemplifica a topologia utilizada por esse sistema.

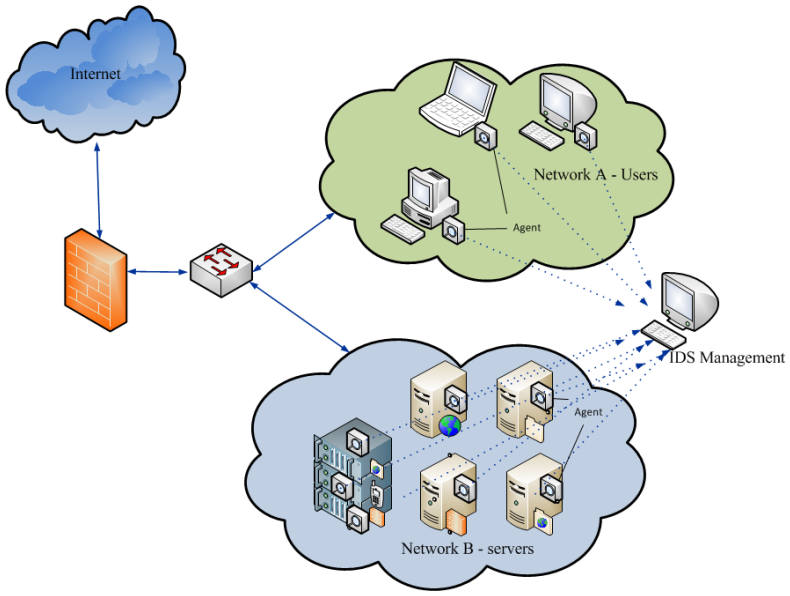


Figura 2-3 - Topologia de um HIDS

A tabela 2-5 descreve as vantagens e desvantagens do uso desse sistema:

Tabela 2-5 - HIDS – Vantagens x Desvantagens

Vantagens	**Desvantagens**
Através do monitoramento local de um host é possível detectar ataques que não são detectados por um NIDS	São difíceis de gerenciar, já que a cada host a ser monitorado deverá ser configurado.
Podem operar em ambientes onde o tráfego de rede é criptografado.	A informação a ser analisada reside no host alvo do ataque, e o host pode ter o IDS desativado como parte do ataque.
Não são afetados por redes baseadas em switches.	Não conseguem detectar ataques a toda rede já que monitoram apenas os pacotes de rede destinados à apenas um único host.
Através da auditoria de logs do sistema operacional é possível detectar cavalos de	Podem ser desativados por alguns ataques

tróia e outros ataques que envolvem violações de integridade, onde estes aparecem como inconsistências no processo de execução.	de negação de serviço.
	Quando se faz captura e análise de logs do sistema operacional, a quantidade de informação armazenada provavelmente requererá um armazenamento externo.
	Consome os recursos de hardware da máquina que está sendo monitorada.

2.4. COMPUTAÇÃO EM NUVEM

Como parte da solução adotada para tratar problemas de Big Data, a computação em nuvem é um assunto relativamente recente, um terreno a ser explorado e que no presente trabalho foi estrutura essencial para garantir o sucesso e aplicabilidade da solução. Esta seção aborda os conceitos da computação em nuvem bem como as características que fizeram com que fosse utilizada no desenvolvimento dessa pesquisa.

O termo computação em nuvem foi cunhado em 2006, em uma palestra de Eric Schmidt da Google descrevendo como sua empresa gerenciava seus próprios datacenters. Na figura 2-4 temos ideia do crescente interesse em torno dessa nova arquitetura de computação.

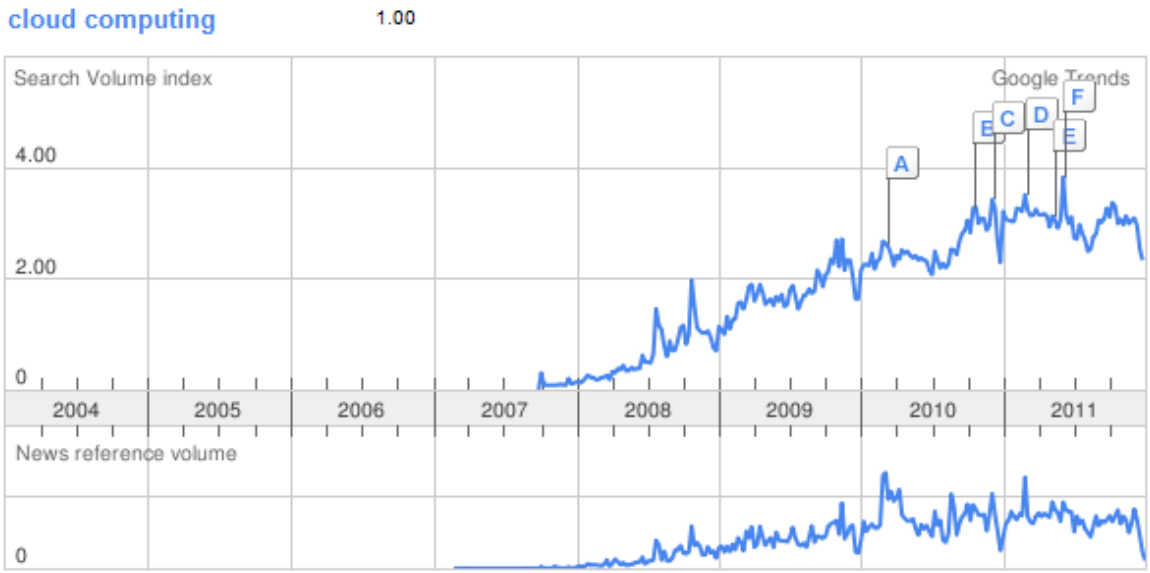


Figura 2-4 - Computação em nuvem

O termo passou a ser mais popular quando a Amazon anunciou sua oferta de EC2 (Elastic Computing Cloud), tornando-se assim grande referência mundial.

Após anos de pesquisa, rascunhos e versões preliminares, o Instituto Nacional de Padrões e Tecnologias (NIST) publicou recentemente, em setembro de 2011, a definição final de Computação em Nuvem (Mell, 2011). De acordo com o NIST, Computação em Nuvem é um modelo para acesso ubíquo, conveniente, sob demanda, a uma rede compartilhada de recursos de computação, que podem ser prontamente disponibilizados e liberados com esforço mínimo de gestão ou de interação com o provedor de serviços. A figura 2-5 exemplifica essa arquitetura.

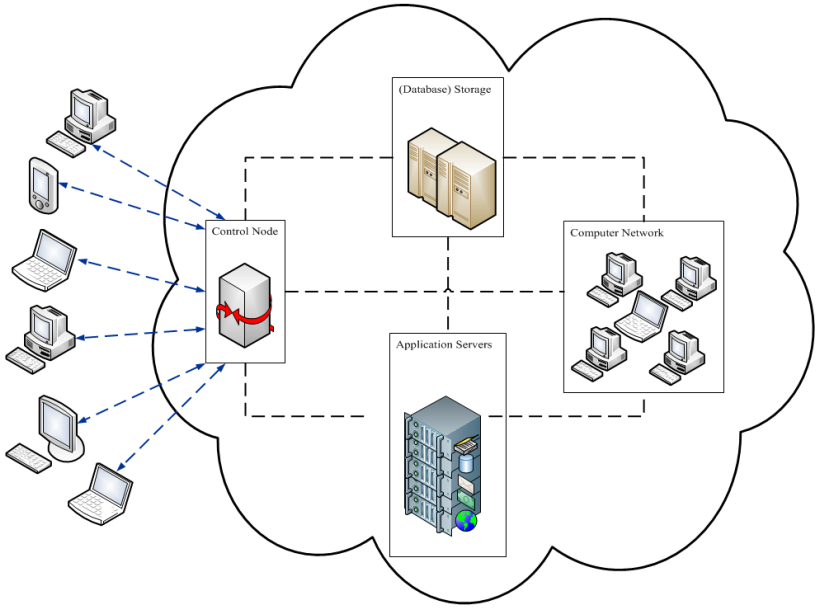


Figura 2-5 - Arquitetura: Computação em Nuvem

Esse modelo é composto por cinco características essenciais, três modelos de serviço e quatro modelos de implementação. Nas subseções seguintes serão abordadas todas as caraterizações definidas para esse novo modelo de computação.

2.4.1. CARACTERÍSTICAS ESSENCIAIS:

Autoatendimento sob demanda: Um consumidor pode unilateralmente dispor de capacidades de computação, tais como tempo de uso de servidor e armazenamento em rede, conforma sua demanda, automaticamente, sem a necessidade de interação humana com cada prestador de serviço.

Amplo acesso à rede: Recursos são disponibilizados através da nuvem e acessados por meio de mecanismos padrão, utilizando plataformas-cliente heterogêneas independentemente da capacidade de processamento, como por exemplo, celulares, tablets, notebooks, estações de trabalho e etc.

***Pooling* de recursos:** Os recursos de computação do provedor são agrupados para atender múltiplos consumidores através de um modelo multi-inquilino, com diferentes recursos físicos e virtuais atribuídos dinamicamente e realocados de novamente de acordo com a demanda do consumidor. Não é possível saber ao certo a localização exata desses recursos, sendo possível especificar um local apenas através de um nível maior de abstração, como por exemplo, estado, país ou datacenter.

Elasticidade rápida: Capacidades podem ser elasticamente provisionadas e liberadas, em alguns casos, automaticamente, para se ajustar à escala, crescente ou decrescente, compatível com a demanda.

Medição do serviço: Sistemas em nuvem controlam e otimizam automaticamente o uso dos recursos, aproveitando uma capacidade de medição em algum nível de abstração apropriado para o tipo de serviço. O uso de recursos pode ser monitorado, controlado e posto em relatórios, proporcionando transparência, tanto para o provedor quanto para o consumidor, do serviço utilizado.

2.4.2. MODELOS DE SERVIÇOS

Três modelos foram definidos, são eles:

IaaS – Infraestrutura como Serviço: A capacidade fornecida ao consumidor destina-se ao provisionamento de processamento, armazenamento, redes e outros recursos de computação fundamentais onde o consumidor é capaz de implementar e executar softwares arbitrários, que podem incluir sistemas operacionais e aplicativos. Nesse modelo, o consumidor não administra e nem controla a infraestrutura de nuvem subjacente, mas tem controle sobre sistemas operacionais, armazenamento e aplicativos implementados, e possivelmente um controle limitado de componentes de rede selecionados (por exemplo, firewalls do host).

SaaS – Software como Serviço: A capacidade fornecida ao consumidor destina-se à utilização dos aplicativos do provedor rodando em uma infraestrutura de nuvem. O consumidor não tem acesso nem controla a infraestrutura de rede subjacente, como servidores, armazenamento, firewalls.

PaaS – Plataforma como Serviço: A capacidade fornecida ao consumidor destina-se à infraestrutura criada ou comprada pelo consumidor para a nuvem, criada usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor.

2.4.3. MODELOS DE IMPLEMENTAÇÃO

De acordo com o uso e a forma de deploy, a computação em nuvem foi dividida em quatro modelos, são eles:

Nuvem privada: a infraestrutura de nuvem é provisionada para uso exclusivo por uma única organização, podendo compreender múltiplos consumidores. Ela pode ser controlada, gerenciada e operada pela organização, um terceiro ou uma combinação.

Nuvem comunitária: a infraestrutura de nuvem é provisionada para uso exclusivo por uma comunidade específica de consumidores de organizações que têm preocupações comuns, como por exemplo, nuvens específicas para profissionais que estudam a cura do câncer, nuvens para profissionais de física quântica e assim por diante.

Nuvem pública: a infraestrutura de nuvem é provisionada para uso aberto ao público em geral.

Nuvem híbrida: a infraestrutura de nuvem é uma composição de duas ou mais infraestruturas de nuvem distintas (privada, comunitária ou pública) que permanecem como entidades únicas, mas são unidas por tecnologia padronizada ou proprietária que permita a portabilidade de dados e aplicativos.

2.5. HADOOP

Nós vivemos na era dos dados. Não é simples medir o volume total de dados armazenados eletronicamente, mas em pesquisas realizadas pelo IDC, para o universo digital em 2009 tivemos um crescimento de 62%, apesar da recessão, chegando próximo a 800.000 petabytes, o que equivale a uma pilha de dvds de ida e volta para a lua. Em 2010, tivemos um crescimento de 1.2 zettabytes. Um zettabyte equivale a 10^{21}, 1.000.000.000.000.000.000.000, ou 2^{70} bytes, 1180591620717411303424 bytes, ou equivalente mil exabytes, um milhão de petabytes ou um bilhão de terabytes. Para se ter uma ideia do volume de dados nunca antes pensado, existe apenas mais uma unidade de medida acima de zettabyte, a yottabyte, acima desse valor não existe escala convencionada. Essa quantidade de 1.2 zettabytes equivale cerca de 75 bilhões de iPads de 16 GB totalmente carregados, dando para cobrir todo o estádio de Wembley, na Inglaterra, um dos maiores do mundo além da criação de uma torre de 545 km de altura. Para o ano de 2011 temos o universo digital estimado em 1.8 zettabytes, com previsão de crescimento para 2020, 44 vezes o de 2009, exemplificado na figura 2-6.

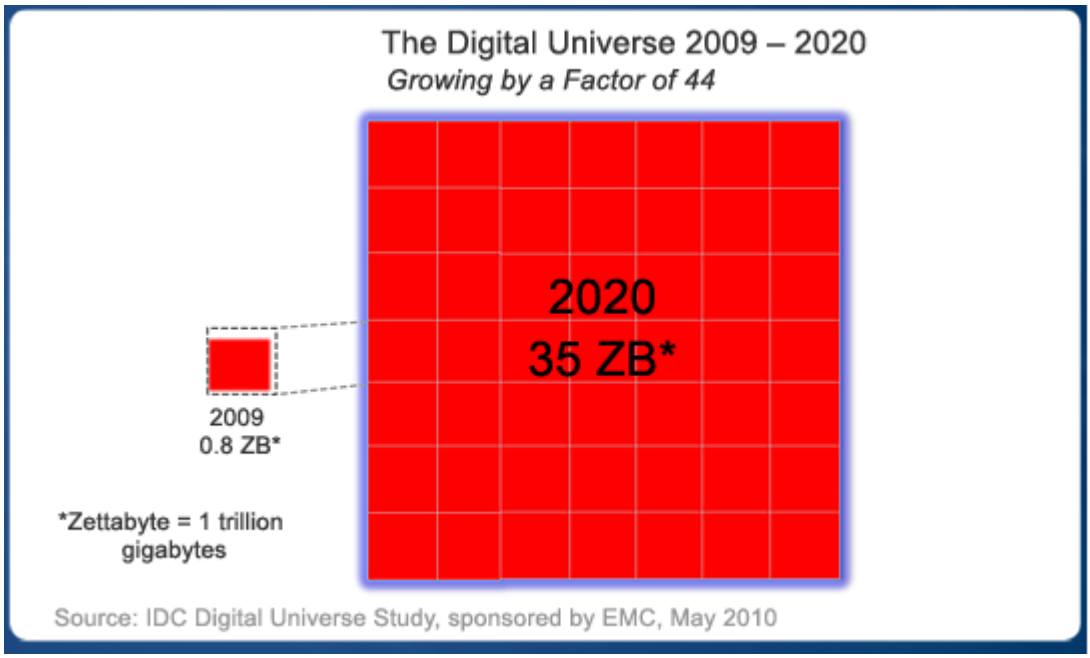


Figura 2-6 - Crescimento do universo digital

Todo esse crescimento se dá devido a fatores como aumento da velocidade e qualidade da internet, redução dos custos de infraestrutura e componentes de hardware como memórias e discos rígidos, criação de novos meios de geração de dados como rede de sensores, TV Digital. Tudo o que é feito nos dias de hoje, gera dados. A convergência tecnológica contribui para essa avalanche de dados, temos dados no computador da empresa, no pessoal, no celular e assim por diante. Nós estamos na era do Big Data! A Google processa 20 petabytes de dados por dia, o LHC produz 15 petabytes de dados por ano, o Facebook hospeda mais de 10 bilhões de fotos.

Vários problemas começaram a se tornar graves: enquanto a capacidade de armazenamento dos discos tem aumentado massivamente ao longo dos anos, a velocidade de acesso, ou seja, a taxa com a qual os dados podem ser lidos dos discos não teve um crescimento na mesma escala. Um disco comum em 1990 podia armazenar 1370 MB de dados e tinha uma velocidade de transferência de 4.4 MB/s, então para ler todos os dados do disco levava cerca de 5 minutos. Quase 20 anos depois, um disco de 1 terabyte passou a ser algo comum, mas com velocidade de transferência em torno de 100 MB/s, o que necessita de mais que 2 horas e meia para ler todos os dados do disco.

O tempo de leitura de dados num único disco é longo, o de escrita é ainda maior. A maneira óbvia de reduzir esse tempo é ler os dados de múltiplos discos de uma só vez. Se

tivéssemos 100 discos, cada um armazenando um centésimo dos dados, trabalhando em paralelo, poderíamos ler os todos os dados em menos de 2 minutos. Trabalhar em paralelo reduz o tempo de leitura e escrita, porém há alguns problemas específicos. O primeiro a ser resolvido é de falhas de hardware. Como vários hardwares estão sendo utilizados, a probabilidade que algum tenha defeito é alta. A forma mais comum de resolver esse problema é evitar a perda de dados através da replicação: cópias redundantes dos dados são mantidas pelo sistema. Isso é proposto por tecnologias como o RAID, que é um arranjo de discos independentes, bem como é feito de forma mais elaborada por sistemas de arquivos distribuídos. O segundo problema a ser atacado é que a maioria das tarefas de análise necessita ser capaz de combinar os dados distribuídos nos outros 99 discos. Vários sistemas de arquivos permitem a combinação dos dados de múltiplas fontes, porém fazer isso corretamente é um desafio complexo. Esse problema é abstraído quando se utiliza o paradigma MapReduce, que será abordado com detalhes nas próximas seções desse trabalho.

Com esse cenário de entraves encontramos em face a vários outros problemas importantes e complexos: dados são gerados de forma desenfreada, como armazenar e processar toda essa quantidade de dados? Como conseguir transformá-la em informação? Como contornar os problemas de recursos finitos como memória RAM, HD, largura de banda? Falhas como falta de energia, falhas de comunicação e hardware, como garantir uma tolerância a falhas? Surge então uma nova plataforma de computação em nuvem e programação paralela, que irá nos ajudar a resolver estas e outras perguntas complexas.

Em 2006 nasce o Apache Hadoop originado do projeto Apache Nutch, tendo Doug Cutting como seu principal fundador, também principal nome do Apache Lucene, uma biblioteca de pesquisa de texto amplamente utilizada. A ideia surgiu da necessidade de criar um motor de busca e indexação de páginas web. Construir um motor de busca web do início é um objetivo extremamente ambicioso, não somente porque causa da complexidade de criação do software de busca e indexação mas também pelo desafio de criá-lo sem um time dedicado de operações, sem um aporte financeiro suficiente. Foi com esse cenário de dificuldade a serem transpostas que Doug Cutting começou a desenvolver o Hadoop, com o projeto Nutch em 2002. Em 2003 a Google revelou a criação de um sistema de arquivos distribuído chamado GFS (Ghemawat, 2003). Em 2004 Cutting anuncia a implementação open source inspirado no GFS, chamada NDFS (Nutch Distributed Filesystem). O projeto

começou a tomar rápido avanço quando um novo paradigma de programação chamado MapReduce foi apresentado pela Google em 2004 (Ghemawat, 2004) e foi portado para a maioria dos algoritmos do Nutch em 2005. A implementação do NDFS e MapReduce no projeto Nutch foi aplicada além do domínio da pesquisa e em 2006 saiu do Nutch e tomou forma como um projeto independente do Lucene chamado Hadoop. Concomitantemente Doug Cutting entrou para a Yahoo!, a qual passou a dedicar uma equipe exclusiva e recursos para aplicar o Hadoop em produção na escala web. Em janeiro de 2008, o projeto Hadoop tornou-se o principal projeto da Apache, confirmando seu grande sucesso, várias gigantes além da Yahoo! como Facebook, Last.fm, New York Times passaram a utilizá-lo pra realizar atividades de alto processamento que antes não eram possíveis. Em abril de 2008, o Hadoop quebrou o recorde mundial tornando o mais rápido sistema a ordenar 1 terabyte de dados, em 209 segundos. O anterior havia feito em 297 segundos. Em maio de 2009, a Yahoo! anunciou a ordenação de 1 terabyte de dados em apenas 62 segundos. Através da figura 2-7 pode-se perceber o grande crescimento do Hadoop, através de uma pesquisa de tendências usando a ferramenta Google Trends®. O estudo dessa plataforma ainda é recente no Brasil, tendo como principais países pesquisadores Coréia do Sul, Taiwan, Índia, China e Estados Unidos.

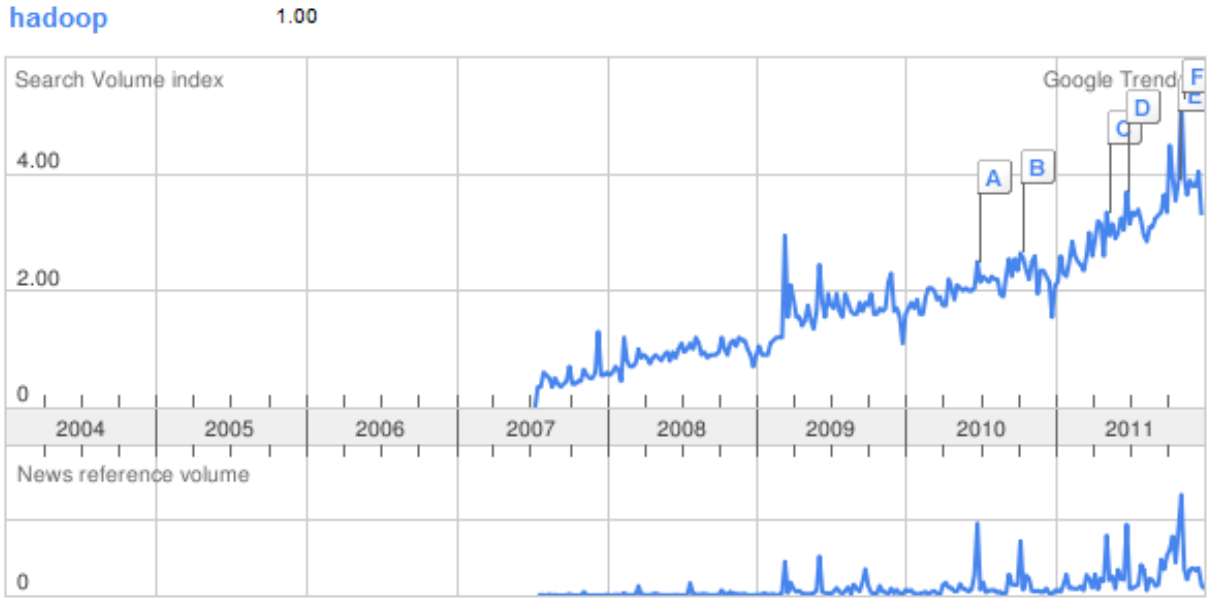


Figura 2-7 - Hadoop trends

Assim surgiu o Hadoop, uma plataforma de computação distribuída elaborada em linguagem Java (Gosling, 2005), sob licença Apache, tornando-se um dos sistemas mais robustos para processamento e armazenamento massivo de dados.

Desde então, essa plataforma tem sido utilizada para tarefas que exijam alto poder de processamento, oferecendo enorme escalabilidade a baixo custo. O Hadoop é formado por dois componentes críticos (Cutting, 2011), um sistema de arquivos distribuído chamado HDFS (Hadoop Distributed File System) que pode formar enormes clusteres em cima de máquinas *commodity*, o que reduz drasticamente o custo de implementação e um framework distribuído de processamento de dados chamado MapReduce. O HDFS provê características importantes como tratamento automático de falhas, alta largura de banda e armazenamento em cluster, além do MapReduce prover um processamento distribuído tolerante a falhas, essa junção de características provê um sistema altamente escalável. Tais componentes e suas características serão abordadas com mais detalhes nas seções posteriores.

O Hadoop possui uma ampla suíte de aplicativos criados de subprojetos do Hadoop principal. A figura 2-8 descreve essa estrutura:

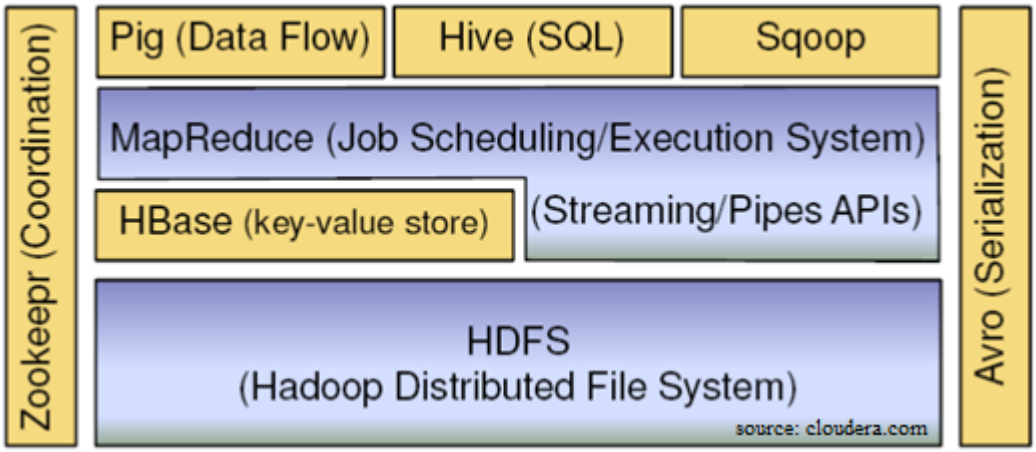


Figura 2-8 - Suite Hadoop

Os principais componentes e suas funcionalidades são:

- **Core**: Conjunto de componentes e interfaces para o sistema de arquivos distribuído e E/S em geral (serialização, Java RPC, estrutura de dados persistente).
- **Avro**: Sistema mais eficiente para serialização de dados entre linguagens RPC.

- **MapReduce**: Modelo de processamento de dados distribuídos e ambiente de execução que atua em grandes clusteres de máquinas comuns.
- **HDFS:** Sistema de arquivos distribuído que é implementado em grandes clusteres de máquinas comuns.
- **Pig:** Linguagem de fluxo de dados e ambiente de execução para exploração de grandes conjuntos de dados.
- **HBase:** Banco de dados distribuído e orientado a colunas. O HBase utiliza o HDFS para seu armazenamento subjacente e suporta tanto processamento em lotes usando o MapReduce como *queries* online.
- **ZooKeeper:** Serviço de coordenação distribuído. Ele fornece primitivas básicas para construir aplicações distribuídas.
- **Hive:** É um *data warehouse* distribuído.Traduz as pesquisas feitas em SQL para Jobs MapReduce.
- **Sqoop**: Importa dados de bancos de dados SQL para Hive.

Essas ferramentas contribuem para os mais variados fins, o Pig auxilia nas operações de ETL (extração, transformação e carga). O Hive auxilia em operações de BI (business intelligence). O Sqoop é uma importante ferramenta para manipulação de Sistemas Gerenciadores de Banco de dados relacionais (SGBDR ou do inglês, RDBMS).

2.5.1. Comparação com outros sistemas

A abordagem adotada por MapReduce pode parecer uma abordagem de força bruta. Nessa abordagem, a premissa é que todo conjunto de dados – ou pelo menos parte dele – seja processado a cada consulta. Aí está o seu poder. O MapReduce é um processador de consulta em *batch* e a capacidade de executar uma consulta *ad hoc* no *dataset* inteiro e obter os resultados em um tempo razoável é totalmente transformadora. Ele muda a maneira de pensar sobre os dados, oferecendo às pessoas a oportunidade de inovar com os dados. Questões que levaram muito tempo para serem respondidas agora podem ter suas dúvidas sanadas, que por sua vez, nos leva a novos questionamentos. Essa seção busca esclarecer as principais dúvidas e questionamentos em relação ao Hadoop e sistemas que se assemelham em estrutura e conceitos como os SGBDR, Computação em Grade e Computação voluntária.

- SGBDR: Porque não podemos usar banco de dados com vários discos para fazer uma análise em lote em larga escala? Porque o MapReduce é necessário?

Uma das respostas para essas questões advém de uma característica dos discos: o tempo de busca (*seek*) é incrivelmente maior que a taxa de transferência. Buscar a informação é o processo de mover fisicamente a cabeça de leitura do disco sobre as bandejas acessando determinado cilindro, acessando assim os dados. Isso caracteriza a latência de operação do disco, enquanto a taxa de transferência corresponde à largura de banda do disco. Se o padrão de acesso aos dados é dominado por *seeks*, isso levará mais tempo para ler ou escrever grandes porções do *dataset* do que o tempo de transferência desses dados. Por outro lado, para atualizar os registros de uma pequena quantidade de dados, o tradicional *B-Tree* (Stoncbrakcr, 2005), (estrutura de dados limitada pela taxa de *seeks* que consegue executar) funciona bem (Pollari-Malmi, 1996). Para atualizar a maior parte de um banco de dados, a B-tree é menos eficiente (Graefe, 2011) que o MapReduce, o qual utiliza operações de *sort/merge* para reconstruir o banco de dados (Yang, 2007). Na tabela 2-6 há uma comparação com os dois sistemas:

Tabela 2-6 - MapReduce x SGBDR Tradicional

	SGBDR Tradicional	MapReduce
Tamanho de dados	Gigabytes	Petabytes
Acesso	Interativo e em lotes	Lotes
Atualização	Ler e escrever várias vezes	Escrever uma vez, ler várias vezes
Estrutura	Estática	Dinâmica
Escalabilidade	Não-linear	Linear

MapReduce é ideal para resolver problemas onde se faz necessário analisar todo conjunto de dados, em lotes e com consultas ad hoc. Um SGBDR é bom para queries online e conjuntos de dados que são constantemente atualizados. Os sistemas também se diferem em relação à estrutura de dados em que opera, um SGBDR trabalha com dados estruturados, organizados dentro de entidades que tem formato definido, por exemplo, XML (Bray, 2000). Esse é o domínio do SGBDR. Dados que não possuem uma estrutura interna definida, como por exemplo, texto simples ou um arquivo de imagem, são considerados não estruturados, assim como arquivos que possuem estruturas semi-

definidas, sendo caracterizados como dados semiestruturados, como por exemplo, planilhas, as quais possuem uma estrutura de grade de células, entretanto as células internas podem armazenar qualquer tipo de dados. O MapReduce trabalha muito bem com dados não estruturados e semiestruturados, as entradas de chaves e valores para o MapReduce não são uma propriedade intrínseca dos dados, elas são escolhidas pela pessoa que está analisando os dados, o que traz enorme robustez ao MapReduce.

Dados relacionais sem frequentemente normalizados para conservar sua integridade e remover redundâncias. Normalizar logs é perder informação. Por exemplo, logs de um servidor web é um bom exemplo onde a normalização não é devida, pois um determinado *hostname* irá aparecer várias vezes e a cada vez armazenará informações importantes. Por essa razão o MapReduce é bem adequado para analisar arquivos de log de todos os tipos. Além disso, o MapReduce é um modelo de programação linearmente escalável (Dean, 2010). Os programadores escrevem apenas duas funções – a função mapear e a reduzir – cada uma delas define um mapeamento a partir de um conjunto de pares de chave-valor para outro. Essas funções abstraem o tamanho dos dados ou o cluster que estão em execução, então elas podem ser usadas sem modificações tanto para um pequeno ou massivo conjunto de dados. Mais importante, se dobrarmos o tamanho dos dados de entrada, o processo levará duas vezes o tempo. Mas se ao contrário, dobrarmos o tamanho do cluster, o processo levará um tempo bem menor. Isso não é necessariamente verdade com consultas SQL (Ordonez, 2000).

- Computação em Grade: A computação de alta performance (HPC) (Buya, 1999) e computação em grade tem sido usada por anos para processar dados em larga escala, usando APIs como *Message Passing Interface* (MPI) (Snir, 1998)**.**

Amplamente, a abordagem de HPC é utilizada para distribuir o processamento através de clusteres de máquinas, que acessam um sistema de arquivos compartilhado, hospedado por uma SAN (Morris, 2003). Isso funciona muito bem para trabalhos predominantemente de computação intensiva, mas torna-se um problema quando os nós necessitam acessar enormes volumes de dados, dessa forma, a largura de banda torna-se um gargalo e os nós do cluster ociosos.

MapReduce procura colocar os dados juntamente com o nó que irá processar, logo o acesso aos dados é rápido porque ele encontra-se armazenado localmente. Essa característica é conhecida por *data locality*, e é o coração do MapReduce e uma das razões

para a excelente performance. Reconhecendo que a largura de banda é o mais precioso recurso em um ambiente de datacenter, as implementações de MapReduce possuem tamanho ótimo de dados para preservá-la.

A MPI provê um bom controle aos programadores, mas requer que eles manipulem mecanismos de fluxo de dados, feitos em C de baixo nível, através de rotinas e construções como sockets, bem como algoritmos de alto nível para realizar as análises. O Mapreduce opera somente em alto nível: os programadores pensam em termos de pares de funções de chave e valor, onde os fluxos de dados são implícitos.

Coordenar os processos de computação distribuída é um desafio. Um dos apectos mais complexos é o controle de falhas. O MapReduce poupa os programadores, em caso de falha, a tarefa é reagendada para máquinas saudáveis. Ele possui uma arquitetura conhecida por *shared-nothing*, isso significa que nenhuma tarefa é dependente de outra. Em contraste, utilizando MPI, os programadores têm que gerenciar explicitamente os pontos de restauração e controle de falhas, o que dá mais poder ao programador, porém de forma muita mais complexa para escrever.

A simplicidade com que o MapReduce trata os dados não é fator limitante desse paradigma de programação, podendo ser usado também para tarefas não-triviais, como por exemplo, análise de imagens, problemas baseados em grafos, algoritmos para aprendizagem de máquina (Anil, 2012).

- Computação voluntária: À primeira vista, o Hadoop e MapReduce são comparados aos projetos de computação voluntária, como o SETI@HOME (Phillips, 1999), (Korpela, 2001).

SETI é um projeto que busca identificar vida extraterrestre, onde voluntários doam tempo de processamento caso seus computadores estejam ociosos, processando dados coletados pelo radio telescópio a procura de sinais de vida inteligente fora da Terra. SETI@home é o mais conhecido dos projetos de computação voluntária, destacando-se também o GIMPS, *Great Internet Mersenne Prime Search* (Woltman, 2012), que busca encontrar enormes números primos e o Folding@home, que busca entender o mapeamento genético de proteínas e como está relacionado com as doenças (Larson, 2002).

Os projetos de computação voluntária trabalham dividindo o problema em pequenos pedaços chamados unidades de trabalho, as quais são enviadas a computadores por todo o mundo para serem analisadas. Por exemplo, uma unidade de trabalho para no SETI@home tem cerca de 0,35 MB de dados coletados pelo rádio telescópio, e leva horas ou dias para ser analisada em um computador doméstico. Quando a unidade é analisada, os resultados são enviados de volta ao servidor e o cliente tem outra tarefa atribuída. Como precaução para combater as fraudes, cada unidade de trabalho é enviada para três diferentes máquinas e é necessário que pelo menos duas delas contenham o mesmo resultado para que ele seja então aceito como válido.

Superficialmente, o SETI@home pode ser considerado similar ao MapReduce, devido a quebra do problema em pequenos pedaços independentes para serem processados em paralelo, mas há várias diferenças significativas. O problema de SETI@home é de capacidade de processamento, o qual é conveniente que seja realizado por centenas de milhares de computadores espalhados pelo mundo, tendo atualmente cerca de 228,182 hosts ativos. O tempo de transferência das unidades de trabalho ofusca o tempo de processamento desse dado. Os voluntários estão doando ciclos de CPU e não largura de banda.

MapReduce é desenhado para processar tarefas em ambientes confiáveis, em hardware conhecido., com armazenamento local de dados. Nas próximas seções será abordado em detalhes os componentes críticos do Hadoop.

2.5.2. Arquitetura MapReduce

A necessidade de aplicações cada vez mais escaláveis nos dias atuais faz com que precisemos “desnormalizar” os bancos de dados. Por questões de desempenho, os dados devem ser distribuídos em datacenters distintos, logo, uma simples *query* pelo *id* do dado não é possível. De nada adiantaria ter uma *foreign key* se temos tabelas espalhadas em diversos *data sources*. MapReduce é um modelo de programação e estrutura de software para escrever aplicações que processam rapidamente quantidades massivas de dados em paralelo em grandes clusters de nós de computação. Introduzido pela Google pela primeira vez em (Ghemawat, 2004) tornou-se um novo paradigma computacional. Inspirado em primitivas de programação funcional, como a LISP (McCarthy, 1978), foi criado o

framework que permitisse a manipulação de grande volume de dados de forma paralela e distribuída, além de prover tolerância a falha, escalonamento de I/O e monitoramento.

O modelo de programação MapReduce consiste na construção de um programa formado por duas operações básicas: *map* e r*educe*. A operação de *map* recebe um par chave/valor e gera um conjunto intermediário de dados, também no formato chave/valor. A operação de *reduce* é executada para cada chave intermediária, com todos os conjuntos de valores intermediários associados àquela chave combinados. Em geral a operação de *map* é usada para encontrar algo, e a operação de *reduce* é usada para fazer a sumarização do resultado. Na figura 2-10 temos um pseudocódigo feito como exemplo para a resolução de um problema: contar o número de ocorrências de uma palavra em uma coleção de grandes arquivos.

Contar_palavras.java

```
1  map(String input_key, String input_value):
2      // input_key: document name
3      // input_value: document contents
4      for each word w in input_value:
5          EmitIntermediate(w, "1");
6
7  reduce(String output_key, Iterator intermediate_values):
8      // output_key: a word
9      // output_values: a list of counts
10     int result = 0;
11     for each v in intermediate_values:
12         result += ParseInt(v);
13     Emit(AsString(result));
```

Figura 2-9 - Exemplo de programa MapReduce

Para cara palavra do documento de entrada, a função map emite o valor ’1′ associado à chave que representa a palavra em questão. A função de *reduce* soma todas as contagens emitidas para uma mesma chave.

A execução de um programa MapReduce segue determinados passos, que são mostrados em um esquemático da figura 2-11:

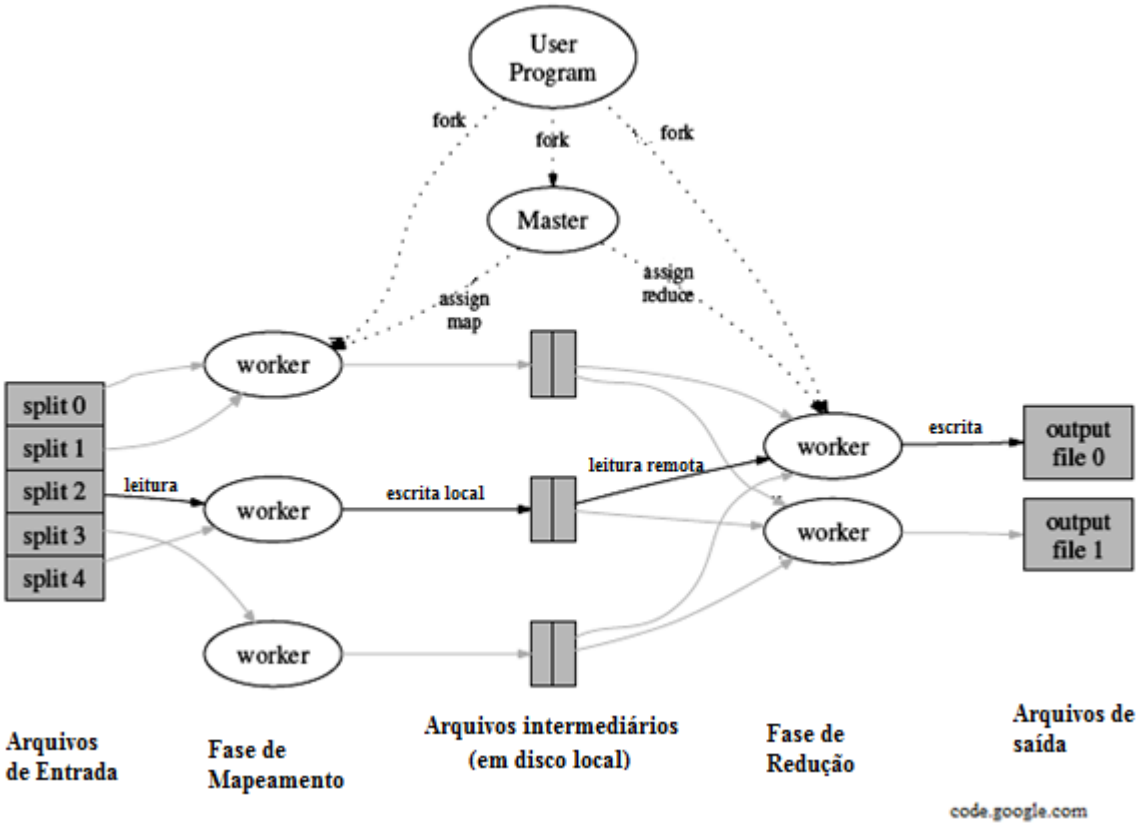


Figura 2-10 - Execução de um programa MapReduce

1. A biblioteca MapReduce, no programa do usuário, primeiro divide os dados de entrada em M pedaços, em nosso caso em pedaços tipicamente de 64 MB, por causa do HDFS . Em seguida inicia várias cópias do programa em um cluster de computadores.
2. Uma das cópias do programa é especial – o *master*. As outras cópias são denominadas *workers* e recebem trabalho do *master*. Existem M tarefas de Map e R tarefas de *reduce* para serem assinaladas. O *master* seleciona *workers* ociosos e assinala a eles uma tarefa de *map* ou de *reduce*.
3. Um *worker* que possui uma tarefa de *map* lê o conteúdo correspondente ao pedaço da entrada. Ele interpreta os pares chave/valor a partir dos dados de entrada e passa como parâmetro para a função de *map* do usuário. Os pares chave/valor intermediários produzidos pela função de *map* são armazenados em memória.
4. Periodicamente os pares de dados dos *buffers* são escritos em disco, particionados em R regiões pela função de particionamento. A localização desses pares de dados no disco é informada ao *master*, que irá repassar essa localização para os *workers* com tarefas de *reduce*.

5. Quando um *worker* de *reduce* é notificado da localização dos dados pelo *master*, este usa uma chamada de procedimento remota para buscar os dados do disco local dos *workers* de *map*. Quando os dados já foram lidos, ele ordena os dados pelas chaves intermediárias, para que todas as ocorrências de uma mesma chave seja agrupada junto. A operação de ordenação é necessária, pois muitas chaves diferentes são mapeadas para a mesma tarefa de *reduce*. Se a quantidade de dados intermediários é muito grande para caber em memória, uma ordenação externa é usada.
6. O *worker* de *reduce* percorre os dados intermediários já ordenados e para cada chave encontrada, ele passa a chave os valores intermediários para a função de *reduce* definida pelo usuário. A saída de cada função de *reduce* é adicionada ao final de um arquivo de saída para aquela partição de *reduce*.
7. Quando todas as tarefas de *map* e *reduce* foram terminadas, o master retorna o programa do usuário.

Uma vez que o framework de MapReduce foi criado para ajudar o processamento de uma quantidade enorme de dados em um cluster com inúmeras máquinas, lidar com falhas é algo essencial. O processo *master* envia um ping periodicamente para cada *worker*. Se o *master* não receber uma resposta de um *worker* em um certo período de tempo, o *master* assume que aquela máquina falhou. Todas as tarefas de *map* completadas pelo *worker* são resetadas para seu estado inicial e são re-escalonadas para outro worker. Essas tarefas precisam ser reexecutadas em caso de falha, pois seus arquivos de saída são armazenados no disco local da máquina que falhou e, portanto, ficam inacessíveis. O mesmo acontece com as tarefas de *map* ou *reduce* que estão em progresso. Tarefas de *reduce* completadas não precisam ser reexecutadas, pois seus arquivos de saída são armazenados no sistema de arquivo global, como HDFS.

No caso do processo *master* falhar, é necessário um controle mais complexo, pois o processo *master* é o elo entre a execução das tarefas de *map* e *reduce*. O processo *master* deve executar checkpoints periódicos de suas estruturas de dados. Em caso de falha, uma nova instância pode ser levantada, recuperando a partir do último estado que foi salvo. O MapReduce assume que só existirá um único processo *master* (*Single Master*), portanto, falhas neste processo são indesejáveis.

Ao final da execução do programa, algumas máquinas, apesar de ainda responderem, podem apresentar um tempo de resposta muito inferior a média das outras máquinas. Por exemplo, falhas nos discos podem reduzir a taxa de leitura de 30MB/s para 1MB/s. Para evitar que esses processos atrasem a execução do programa, quando o programa está perto de terminar, algumas cópias das tarefas restantes são iniciadas (tarefas backup). A tarefa será marcada como completada assim que, ou a tarefa primária ou uma tarefa de backup, responder. A título de exemplo, um programa de ordenação, pode demorar até 44% mais tempo se o mecanismo de tarefas de backup estiver desligado (White, 2009).

A largura de banda é um recurso escasso neste ambiente de computação. Buscando contornar esse problema, o MapReduce faz com que a quantidade de comunicação seja reduzida pelo fato de que os dados de entrada estão armazenados no disco local das máquinas que fazem parte do cluster, compondo um sistema de arquivos distribuído, que será detalhado na próxima seção, o HDFS, que em conjunto com o MapReduce formam a estrutura essencial para um ambiente escalável, tolerante a falhas, com alto poder de processamento e armazenamento, imprescindível na construção desse ambiente de pesquisa, inovando no método de análise e tratamento de logs de segurança.

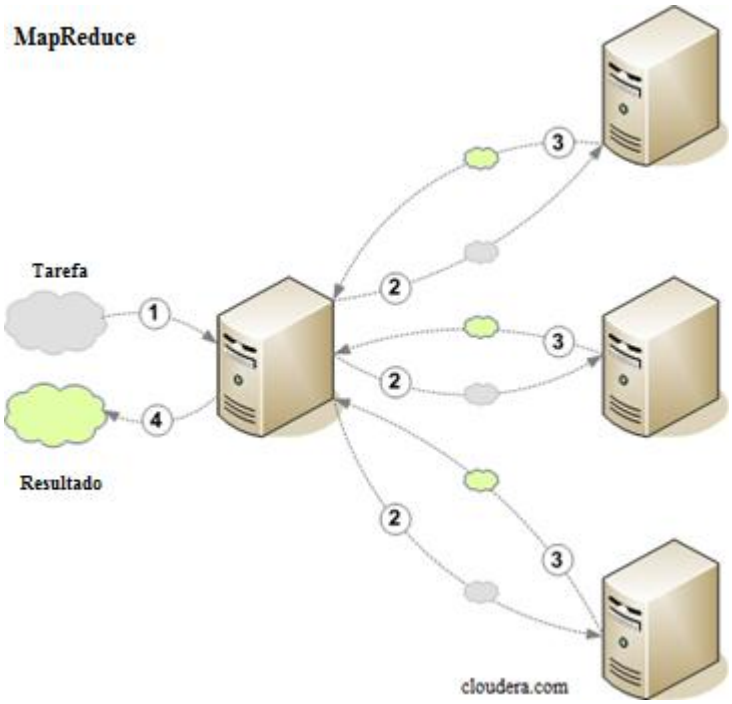


Figura 2-11 - Divisão de tarefas e obtenção de resultados no MapReduce

A figura 2-12 mostra a sequência dos processos realizados na execução de uma tarefa MapReduce. Em (1) a tarefa é iniciada pelo nó *master*, o qual divide o processo em vários subprocessos que realizam as tarefas de mapeamento e redução, em (2), realizados pelos nós slaves. Cada nó slave processa a tarefa atribuída e retorna o resultado ao no master (3). O nó master consolida os resultados enviados por cada nó slave fazendo a checagem dos resultados (4).

2.5.3. Hadoop Distributed File System - HDFS

Hadoop Distributed File System (HDFS) é o sistema de armazenamento primário voltado para computação de alto desempenho, escalável e com ênfase na consistência e disponibilidade dos dados. O HDFS cria várias réplicas de blocos de dados e os distribui em nós de computação ao longo de um cluster permitindo confiabilidade e cálculos extremamente rápidos. Sua concepção foi inspirada no Google File System (Ghemawat, 2003) e possui características como:

- O HDFS é executado em clusters de centenas ou milhares de nós constituídos de *commodity* hardware, o qual possui baixo custo e pode ser fornecido por diversos fabricantes. Falhas de hardware são normas e não exceção. Esse grande número de elementos envolvidos gera uma probabilidade não trivial para falhas. Consequentemente o HDFS faz uso intensivo de somas de verificação e replicação para detecção e mascaramento de problemas;
- O sistema de arquivos possui uma interface muito próxima ao padrão POSIX (Josey, 2008). Foi projetado para processamento em lotes e não uso interativo pelos usuários. A ênfase está na alta taxa de transferência de dados em detrimento da baixa latência.
- O sistema é otimizado para arquivos grandes, na ordem de gigabytes ou terabytes de tamanho. Para armazenar arquivos pequenos de forma eficiente, o HDFS conta com uma funcionalidade parecida com o comando tar do Unix que armazena vários arquivos num só;
- O sistema suporta apenas um escritor, porém vários leitores simultaneamente. Dados somente podem ser adicionados ao final do arquivo. Ou seja, não são permitidas escritas aleatórias.

A arquitetura do HDFS segue um padrão de mestre-escravos (*master/slaves*). O *master*, servidor de metadados é denominado namenode. Já os escravos são denominados datanodes e são servidores de armazenamento e processamento. A figura 2-12 ilustra como essa arquitetura é projetada.

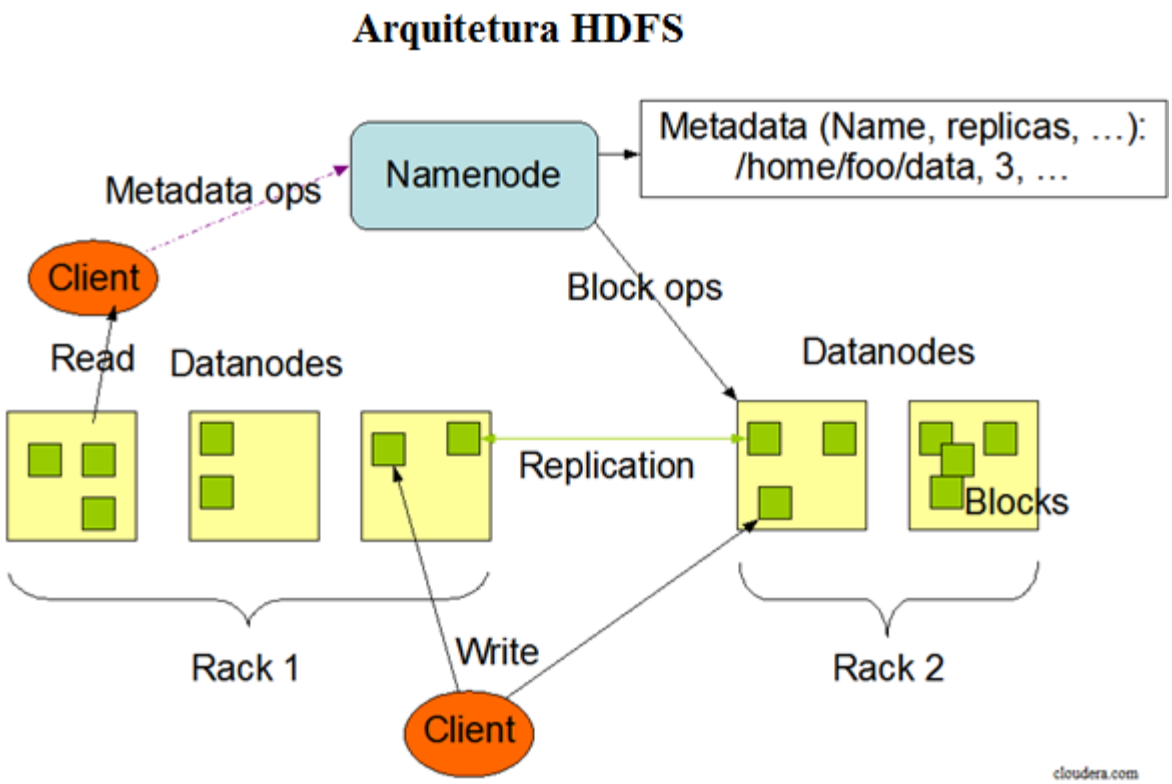


Figura 2-12 - Arquitetura do HDFS

Assim como em sistemas de arquivos convencionais, arquivos no HDFS são compostos por blocos de tamanho fixo. Os datanodes são os responsáveis por armazenar os blocos, os quais são representados localmente por dois arquivos. Um armazena os dados do bloco, enquanto o outro guarda seus metadados com identificação do bloco, versão e somas de verificação. Os blocos possuem um tamanho grande, com valores típicos de 64 MB e 128 MB. A razão para isso é que se o tamanho do bloco for tal que tempo de sua transferência for consideravelmente maior que o tempo de *seek* do disco, a transferência do bloco pode ocorrer na mesma taxa de transferência do disco. Blocos são replicados, o que garante proteção contra perda de dados e bom desempenho de leitura, visto que várias réplicas podem ser lidas em paralelo. O nível de replicação pode ser definido individualmente para cada arquivo. O algoritmo padrão de posicionamento de réplicas tenta garantir um bom *trade-off* entre confiabilidade e desempenho de leitura e escrita. Pelo menos duas réplicas

são postas no mesmo rack, mas em nós diferentes, e pelo menos outra réplica é posta em outro rack distinto.

O namenode é o responsável por gerenciar a árvore de diretórios e arquivos, usando estruturas similares a inodes. Desta forma ele guarda informações como o nome, caminho, tempo de acesso e modificação, permissões, dono, grupo, cotas e a relação de blocos que compõe o arquivo. Periodicamente, os datanodes enviam mensagens de *heartbeat* para informar o namenode de que estão ativos e de suas respectivas capacidades livres atuais. Também enviam, mas com uma periodicidade menor, *block reports*, que são relatórios de todos os blocos que o datanode é capaz de oferecer. Com isso, o namenode consegue ter uma visão global da localização e do nível de replicação de cada bloco no sistema. Caso o nível de replicação de um bloco não esteja de acordo com o que foi definido, o namenode instrui datanodes a copiarem réplicas entre si, por meio de comandos enviados em resposta às mensagens de *heartbeat.* Por questões de desempenho, o namenode mantém todo seu estado em memória principal.

Para evitar que todo o sistema de arquivos seja perdido em caso de falha do namenode, uma combinação de checkpoints e logs transacionais é empregada para tornar a recuperação possível. Assim, operações que modifiquem o espaço de nomes são registradas em memória secundária estável. Ao iniciar, o próprio namenode cria um novo checkpoint a partir da fusão do log com o checkpoint anterior. Durante a execução, um segundo nó, o backupnode, fornece checkpoints adicionais para o namenode, uma vez que o backupnode recebe constantemente notificações de alterações no espaço de nomes. Informações relativas à localização das réplicas não são persistidas em checkpoints ou logs por serem voláteis e mudarem muito rapidamente. Desta forma, quando o namenode inicia, ele espera por *block reports* da maioria dos datanodes para reconstruir o mapeamento entre blocos e datanodes. Só quando este mapeamento está reconstruído é que o namenode passa a atender as requisições dos clientes.

Os usuários do HDFS interagem com o sistema por meio de chamadas de procedimento remoto, assim como a comunicação feita entre o namenode e os datanodes. Para leituras, o cliente solicita ao namenode a lista de blocos do arquivo de interesse bem como os datanodes que contém as réplicas dos blocos. O cliente pode então ler dos datanodes mais próximos, garantido assim uma boa vazão. Para escritas, o cliente solicita que o namenode aloque um conjunto de datanodes para armazenar as réplicas. Durante a escrita, os

datanodes são organizados em um pipeline, com um datanode repassando para o seguinte os dados a serem escritos. Conforme já foi dito, apenas um escritor por arquivo é permitido e o controle disso é feito por um mecanismo de *leasing*. Um processo escritor deve solicitar uma *lease* e renová-la periodicamente para pode continuar a escrever. Do contrário a *lease* pode expirar e ser concedida a um novo escritor. É importante salientar que todo tráfego de dados ocorre diretamente entre cliente e datanodes, sem intermediação do namenode, de forma que este último não se torne um gargalo.

3. SISTEMA DISTRIBUÍDO DE COLETA, ARMAZENAMENTO E PROCESSAMENTO DE LOGS DE SEGURANÇA

Nesse capítulo será abordado a arquitetura proposta bem como a escolha dos componentes necessários para implementá-la e como ela se difere dos sistemas existentes.

Várias abordagens foram feitas procurando a resolução dos problemas de tratamento de grandes massas de dados, através do uso de agentes móveis, análise de componentes principais através de seleção de ordem do modelo, data mining, uso de FPGAs, porém todas têm suas ressalvas e não resolvem por completo os problemas em questão. Temos uma arquitetura que se divide principalmente em três áreas, a coleta, feita por sensores, o armazenamento e processamento, realizado em computação em nuvem através do framework MapReduce e do sistema de arquivos distribuído HDFS e por último uma interface de gerência que será responsável pelo controle do ambiente e visualização dos resultados. A arquitetura do sistema criado está descrita na figura 3-1 e foi proposta inicialmente em (Holtz, 2011B):

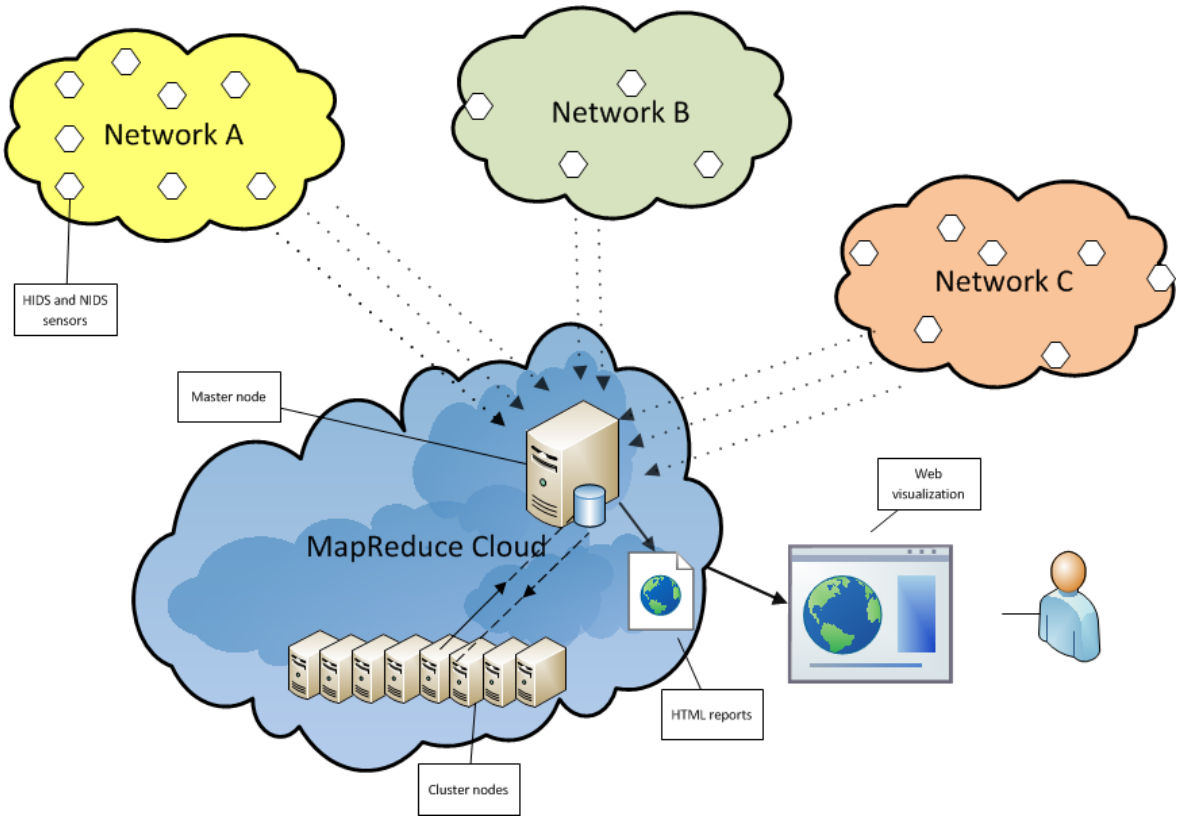


Figura 3-1 - Arquitetura distribuída de coleta e tratamento de logs

A. **Sensores.** Sistemas de segurança devem ser utilizados em conjuntos com outras ferramentas visando bloquear as lacunas inerentes aos próprios sistemas. IDS apresentam limitações conforme foi levantado. Dessa forma, para obtermos uma solução mais completa possível, o sistema proposto leva em consideração o uso de IDS baseados em rede e em host em conjunto com o uso de honeypots, formando sensores espalhados por toda a rede. A proposição desse sistema leva a um efeito de avalanche de dados, sendo agravado pelo uso de redes de alta velocidade.

O uso de vários sensores tem o propósito de construção de um sistema robusto e abrangente, que através da agregação de vários pontos de coleta proverá informações importantes em vários níveis de informação. Sistema clássicos de IDS não conseguem identificar eficientemente ataques em redes heterogêneas devido a sua natureza inerentemente centralizada. Nesse novo modelo, será possível coletar informações de toda a rede em uma panorama geral, além de informações individuais de hosts monitorados, bem como todo o tráfego criptografado na rede. O uso de honeypots nos dará informações comportamentais de invasores e que serão úteis para o estudo e elaboração de medidas mais eficientes de proteção e a criação de novas assinaturas de ataques.

Os sensores espalhados na rede foram compostos por IDS baseados em rede e em host. Foi utilizada a ferramenta Snort para coletar as tentativas de intrusão nas redes monitoradas. Já para detecção de ataques utilizando dados criptografados e em hosts locais, foi utilizada a ferramenta OSSEC. Esses IDS atualmente apresentam-se como as soluções mais robustas e com comunidades de apoio, por isso foram escolhidas para o projeto. Os sensores ainda foram complementados com o uso do honeypot Honeyd, que é um honeyot de baixa interação, nas quais foram simulados sistemas operacionais de vários tipos hospedando servidores FTP, HTTP, banco de dados e etc, e faixas de IP não alocados, que serviram como armadilhas. Os sensores foram espalhados em várias redes de classes diferentes, cobrindo inclusive VPNs e buscando maior detalhamento e enriquecimento das informações, logs de sistemas operacionais como audit.log e syslog foram coletados de servidores reais, permitindo assim a extração em vários níveis de informação, nas camadas de rede, enlace e aplicação.

Todo esse sistema irá gerar uma quantidade massiva de dados, que sem um sistema distribuído de armazenamento e processamento baseado em nuvem, de nada adiantaria devido à complexidade de manipulação desses dados.

B. Infraestrutura de nuvem: Esse ambiente de nuvem foi criado utilizando hardware *commodity*, ou seja, máquinas simples sem características especiais como servidores ou máquinas HPC, tendo, portanto, um custo muito acessível e promovendo enorme escalabilidade. Nós são facilmente adicionados ou removidos, adaptando-se à demanda exigida. Esse cenário foi criado a partir de um cluster de cinco máquinas *slaves* e um nó *master*, tendo capacidade inicial de armazenamento de 1.42 TB como volume HDFS. Outra vantagem apresentada pelo HDFS e ponto passível na definição de sua escolha, é por ser executado em máquinas virtuais java, o que aumenta a portabilidade e a escalabilidade da solução, que já os nós acrescidos para aumentar o poder de processamento e armazenamento do cluster são independentes do tipo de sistema operacional e não precisam ser dedicados exclusivamente para o sistema proposto.

Os dados coletados pelos sensores são armazenados e compactados através de compressão Lempel-Ziv-Oberhumer (LZO). Essa escolha se deve ao fato que o LZOP (implementação de código livre do LZO sobe licença GPL) apesar de gerar arquivos um pouco maiores do que a GZIP, por exemplo, consome menos de um décimo de uso de CPU. Uma janela de cinco minutos foi escolhida para o tempo de coleta e envio, já que o sistema de arquivos não é desenhado para múltiplas operações de escrita em tempo real. Esse tempo foi escolhido através de testes de desempenho e considerado totalmente satisfatório e necessário para o bom funcionamento do sistema. Todos os dados anteriormente enviados ao HDFS serão acessíveis para processamento e análise e serão incrementados a cada cinco minutos, através do envio em lotes dos arquivos, através de scripts feitos em shellscript e comandos de manipulação do HDFS. Temos assim os dados coletados e armazenados de forma distribuída, que precisam ser então processados de maneira eficiente e escalável, já que toda essa quantidade terá uma demanda por armazenamento cada vez maior.

O uso do paradigma de computação MapReduce e do sistema de arquivos HDFS proveu um ambiente escalável, eficiente e tolerante a falhas necessário. O MapReduce teve ganhos no tempo de processamento na ordem de 72% quando comparado a outras ferramentas de análise de fluxo de rede, como o *flow-tools*. A nuvem é composta por um nó *master* que será responsável por coletar os dados e distribuí-los entre os demais nós da rede, conhecidos como *slaves*. Um nível de replicação três foi definido buscando maior robustez para medidas de recuperação de desastres, já que o mesmo dado é alocado em três nós diferentes, permitindo assim a sua recuperação em caso de falhas. O nó *master* é

responsável por gerenciar o sistema de arquivos e enviar os trabalhos de MapReduce para processamento e coleta de resultados.

Nesse ambiente, o nó *master* é composto principalmente pelo JobTracker e pelo NameNode. O JobTracker é um serviço que determina os nós *slaves* que terão acesso necessário aos dados e que estão disponíveis para receber as tarefas. Com base na informação recolhida individualmente a partir dos nós *slaves*, as tarefas são atribuídas aos próprios nós *slaves*. O JobTracker também recolhe informações sobre cada tarefa em execução e agendadas dos nós *slaves* as quais são utilizadas para alocar novos nós e prover ao usuário informações estatísticas e gerenciais sobre o estado global da nuvem. O NameNode possui estrutura de dados e diretórios do HDFS assim como os inodes. As aplicações MapReduce criadas acessam o serviço do NameNode para executar as tarefas como adicionar, copiar, mover ou deletar arquivos armazenados nos volumes HDFS.

Os outros nós *slaves* do cluster executam dois outros serviços os quais compõem o ambiente de processamento e arquitetura de armazenamento de dados. A figura 3-2 descreve esse ambiente em camadas da estrutura do cluster de múltiplos nós, o qual foi utilizado no sistema, buscando obter melhor desempenho possível do ambiente distribuído.

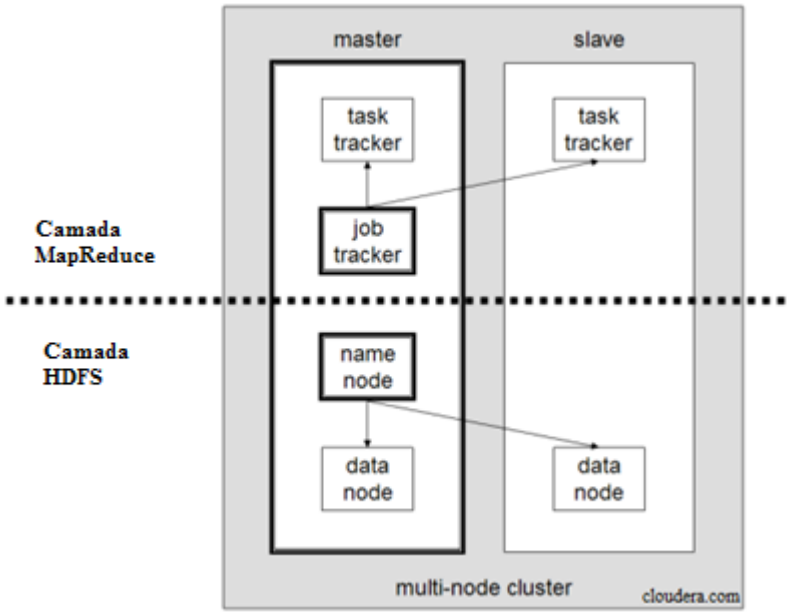


Figura 3-2 - Divisão de serviços em um cluster Hadoop de múltiplos nós

O TaskTracker recebe as tarefas, isto é, Map, Reduce e Shuffle, do JobTracker executado no nó *master* e reporta de volta ao *master* o status atual e das tarefas agendadas. Ele tem

um conjunto de *slots* virtuais os quais determinam o número de tarefas que podem ser aceitas e processadas. O DataNode gerencia o armazenamento local das fatias de dados alocados em cada nó *slave* através do nó *master*.

Quando o nó *master* recebe um trabalho, o JobTracker consulta o DataNode de cada nó *slave* para determinar quais nós já têm os dados necessários armazenados localmente e então aloca um *slot* com o TaskTracker em execução no nó dado. Se ele não encontrar um nó que já contenha os dados, será alocado o primeiro *slot* vazio encontrado em qualquer nó. Normalmente, um arquivo armazenado no volume HDFS é replicado em vários nós *slaves* através de seus respectivos DataNodes. Quando um TaskTracker recebe um novo trabalho, que consiste em uma nova tarefa MapReduce para ser executada e a descrição do conjunto de dados necessário, ele gera um processo filho para executá-lo e em seguida reporta seus status para o JobTracker. A figura 3-3 descreve a estrutura de um trabalho MapReduce.

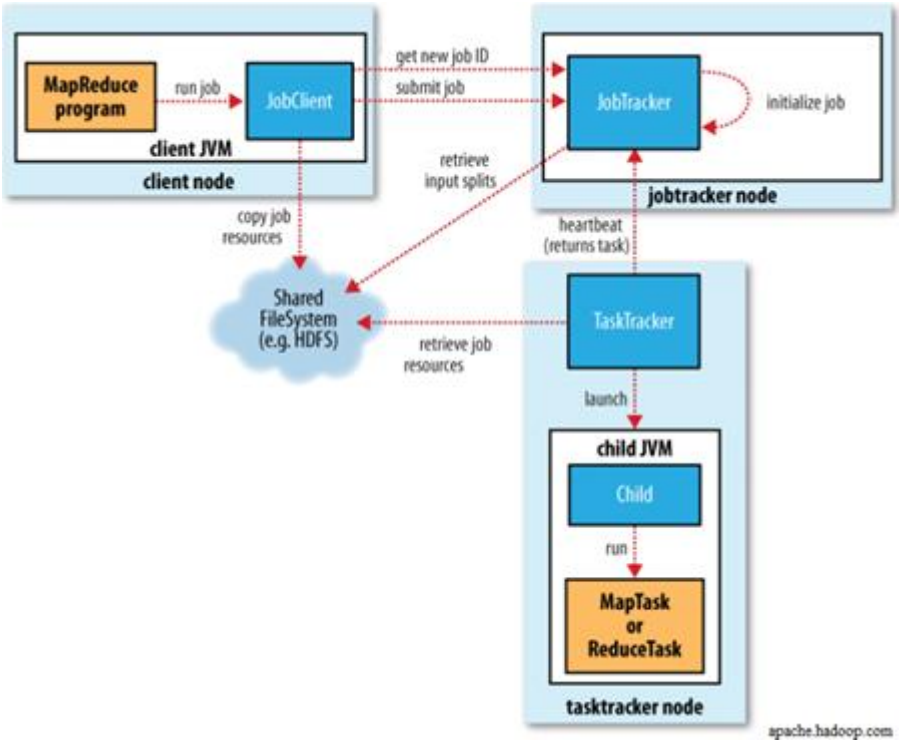


Figura 3-3 - Estrutura de um trabalho MapReduce

Na pesquisa e construção do sistema, foram criados trabalhos MapReduce para analisar todo os dados coletados dos sensores espalhados na rede, bem como de dados externos que porventura necessitem ser analisados. Esses trabalhos são complexos, e buscando ganho de produtividade, foi utilizada a ferramenta Pig, que possui uma linguagem de programação e

compiladores de alto nível criada para se programar trabalhos MapReduce otimizados, com maior rapidez e um maior nível de abstração, já que são necessários múltiplos estágios de interação com o cluster e definição das funções Map e Reduce pelo próprio usuário. Esses scripts são únicos e foram criados para analisar os arquivos e logs dos sistema de segurança com o intuito de retirar as informações importantes e necessárias para uma análise completa. Mais detalhes dessa implementação serão vistos na próxima seção.

Temos assim um ambiente computacionalmente distribuído, onde a rede não mais será um gargalo nem terá limitações impostas por capacidade de processamento e armazenamento, o que tem sido um problema em aberto nas abordagens atuais no tratamento de dados de logs de segurança.

C. Interface de gerenciamento e análise:

O sistema proposto, ainda fornece uma interface web para gerenciamento do cluster, análise dos dados, geração de relatórios e correlacionamento, possibilitando ainda a inclusão de dados coletados em outras redes e aplicações, o que aumenta a valoração do sistema já que ele não é inerente ao sistema pré-configurado. Essa interface tem o intuito de facilitar a manipulação de todo o sistema garantindo ainda maior maneabilidade. Composta por linguagem PHP, javascript e scripts em background em shellscript, os dados resultantes da análise podem ser exportados em formatos JSON para garantir maior desempenho na transferência dos dados, já que possui menos overhead quando comparado a formatos como o XML, sendo também armazenados num banco MySQL para consultas pontuais.

Buscando melhores resultados a interface foi criada para permitir o controle de usuários, gerenciamento do cluster, testes de desempenho, análise e visualização dos trabalhos MapReduce, relatórios e gráficos, correlação de logs, possibilidade da análise de outros formatos de logs além da portabilidade web, não tendo restrição de acesso por localidade ou tipo de plataforma. Esses relatórios poderão ser utilizados no estudo de tendências de ataques, criação de novos padrões de assinaturas, estudos comportamentais de invasores e criação de políticas preventivas de segurança da informação, abrindo caminhos para normas formas de detecção de intrusão e análise de ataques. A figura 3-4 descreve a tela inicial da interface desenvolvida para o sistema em estudo, o sistema como um todo foi chamado de Logdoop:

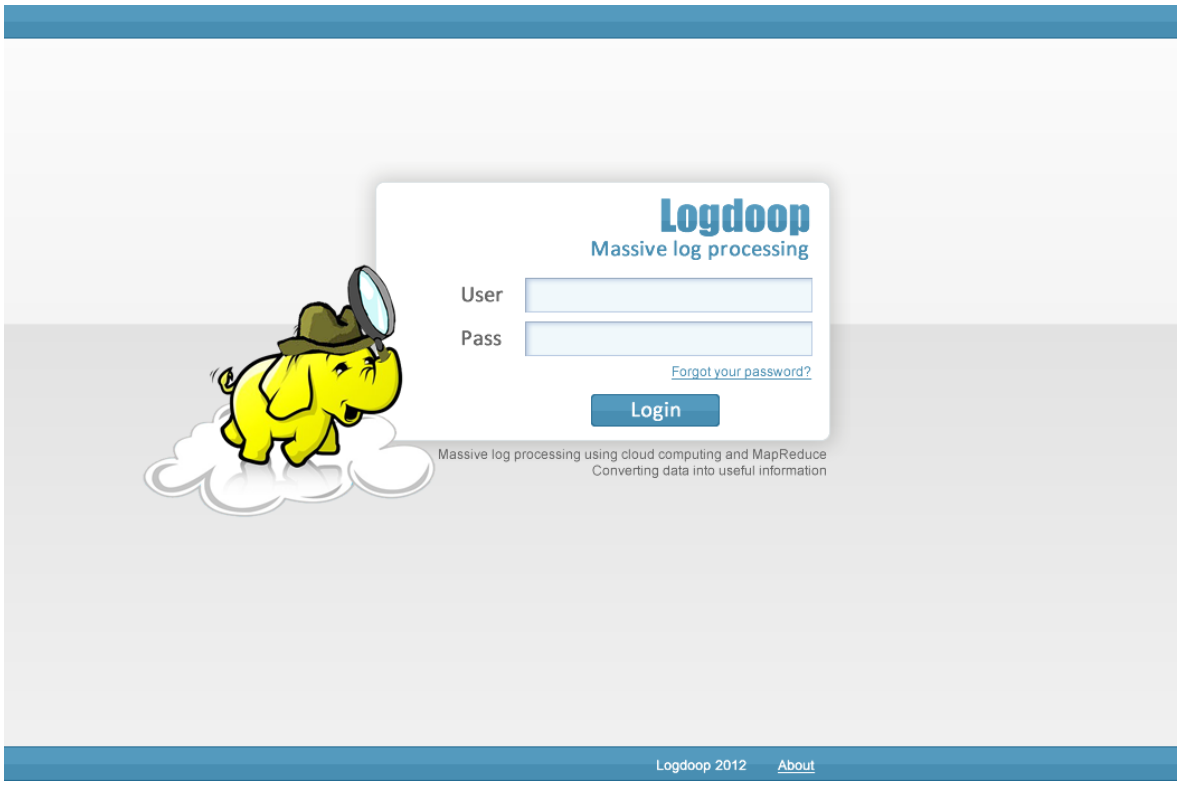


Figura 3-4 - Logdoop – Interface web – Tela inicial

É importante notar que a arquitetura proposta possui um grande valor agregado, já que além de prover novas diretrizes no tratamento de grandes massas de dados e sistema de segurança, teve também como resultado um sistema final a ser explorado além dos campos de pesquisa, podendo ser uma ferramenta essencial para grandes empresas que buscam medidas para contenção de ataques e danos causados por eles. Toda a arquitetura foi construída utilizando componentes de código aberto, o que reduz drasticamente o custo de desenvolvimento e implementação.

4. IMPLEMENTAÇÃO E RESULTADOS

Essa seção é destinada aos aspectos práticos da pesquisa realizada e da construção do sistema resultante, mostrando a viabilidade da solução, os aspectos inovadores e resultados.

A nuvem criada para o armazenamento e processamento dos dados coletados foi formada com seis máquinas *commodity*, arquitetura simples de máquinas comuns, portanto um cluster de custo bem acessível. Das seis máquinas, uma foi configurada como nó central, sendo o *master* e as cinco demais como nós *slaves*. Cada nó é composto pela configuração descrita na tabela 4-1.

Tabela 4-1 – Configuração de hardware dos nós.

Processador	Intel® Core 2 Duo CPU E7500 2.93 GHz
Memória RAM	4 GB DDR667
Disco Rígido	300 GB
Rede	100 Mbps

O framework MapReduce foi implementado em todos os nós e foi obtida uma capacidade de armazenamento do volume HDFS de 1.42 TB. Conforme mencionado, essa nuvem formada traz grande portabilidade já que não precisa ser utilizada exclusivamente para o nossas atividades. Ela não precisa ser tratada unicamente como cluster e ter sua atividade dedicada, assim como são tratados os clusteres tradicionais, o que diminui bastante o gasto com a infraestrutura, já que essas máquinas podem ser utilizadas para outras funções assim como novas máquinas podem ser facilmente incluídas, aumentando o poder de armazenamento e processamento da nuvem.

Primeiramente foram feitos testes de entrada/saída do sistema de arquivos distribuído e ordenação de dados, chamado de *sort*, para verificar o desempenho da nuvem criada, utilizando o HDFS e o *framework* MapReduce. O primeiro experimento consistiu na criação de novos arquivos no nó *master* com variação de tamanho contendo dados aleatórios os quais foram movidos pra o volume HDFS para medir o desempenho do

sistema de arquivos. A figura 4-1 exemplifica o tempo gasto para escrever um arquivo variando de 1 a 250 Megabytes para o sistema de arquivos distribuído.

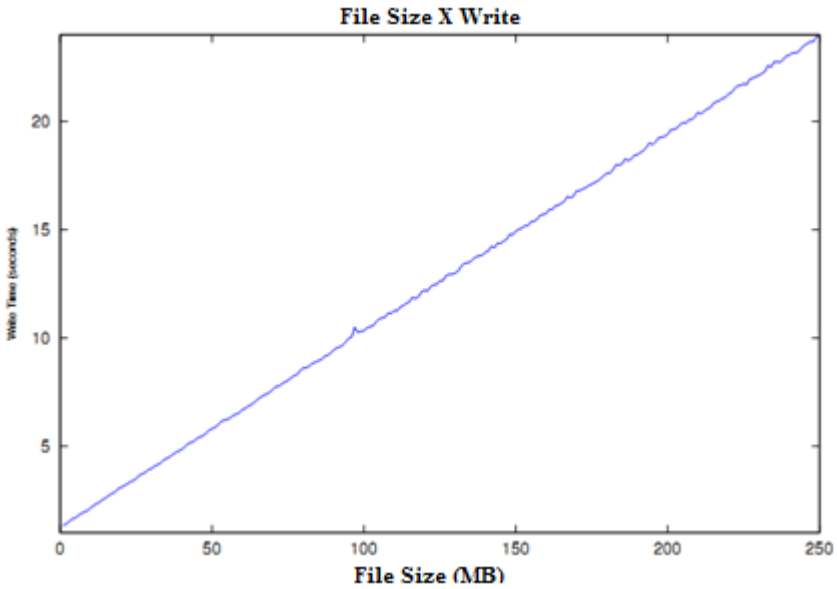


Figura 4-1 - Desempenho de escrita: Tamanho de arquivo x Tempo

Esses arquivos foram escolhidos para representar a quantidade de dados coletados por um sensor durante um minuto. Perceba que 250 Megabytes por minuto durante o um dia gera aproximadamente 351 Gigabytes, o que certamente é maior que muitas redes poderiam gerar em termos de *logs* de segurança. Esse experimento mostrou que o tempo de escrita aumenta linearmente de acordo com o tamanho dos arquivos, já que os arquivos precisam ser fatiados e distribuídos pelos nós. É claro que o sistema seria escalado para uma maior quantidade de dados. Além do mais, esses tempos seriam significantemente reduzidos se fosse utilizado uma rede de alta velocidade, tais como Gigabit ethernet, ou se mais nós forem adicionados ao cluster.

O segundo experimento consistiu na ordenação de dados aleatórios com variação de 1 megabyte até 250 megabytes usando o algoritmo de Quick-Sort. Nesse experimento os mesmos arquivos criados para o primeiro foram utilizados, os quais são interessantes já que a entropia dos dados aleatórios é consideravelmente maior que a entropia dos logs de dados reais de tráfego de rede, representando o pior caso para ordenação e uso de algoritmos de identificação de padrão. Os tempos dos arquivos testados na ordenação oscilaram entre 33.4 e 35.6 segundos devido a oscilações naturais no desempenho da rede e são observados na figura 4-2.

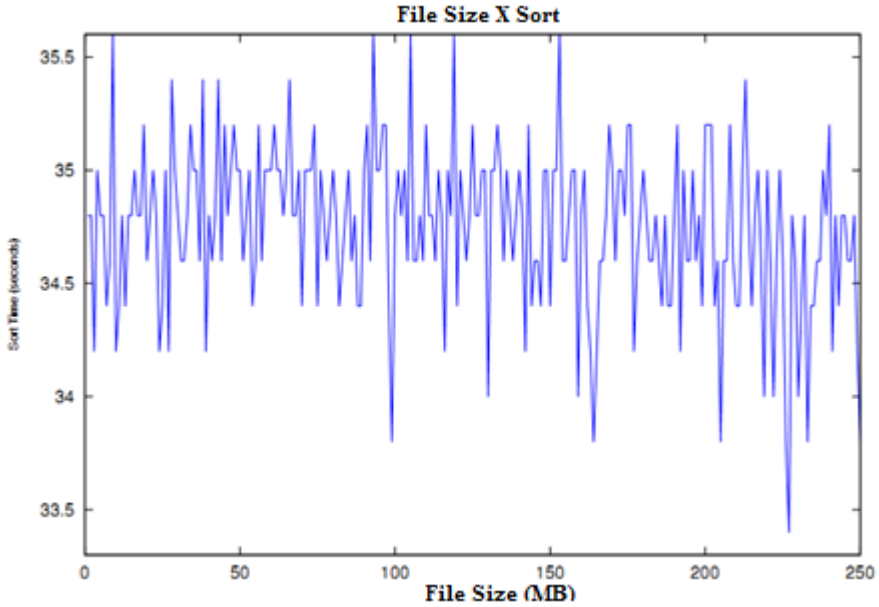


Figura 4-2 - Desempenho no processamento de dados

O tempo de ordenação é essencialmente o mesmo para os dados nesse intervalo devido ao overhead de comunicação e sincronização entre os nós da nuvem. É importante perceber que há um *overhead* constante, o qual é naturalmente esperado devido os nós que compõem o volume têm que receber e processar as mensagens de sincronização antes da verdadeira transferência dos dados. Mesmo para o caso da ordenação de um único megabyte de dado é necessário preparar os nós e o sistema de arquivos distribuído para executar a requerida tarefa de ordenação, a qual leva um tempo constante. O tempo gasto pela tarefa de ordenação em si próprio é insignificante quando comparado ao *overhead* de sincronização, mostrando que essa solução é extremamente escalável para um grande volume de dados.

Ambos os experimentos realizados nessa infraestrutura de nuvem tiveram a intenção de ser o núcleo da arquitetura criada, mostrando que é viável para armazenar e processar grandes quantidades de dados. Tanto a criação dos arquivos, o processo de transferência e o algoritmo de ordenação, foram eficientemente executados na infraestrutura de nuvem. Essas características são essenciais para a análise dos logs segurança, uma vez que os dados são coletados e a informação deve ser classificada.

De posse desses resultados, um ambiente real foi criado para verificar o comportamento e validade dessa arquitetura distribuída, comprovando que supera a propostas atuais de tratamento de logs de segurança da informação. Os sensores foram ativados e passaram a registrar os eventos de atividades na rede, como tentativas de intrusão e outros logs do

sistema. A coleta foi feita em ambientes heterogêneos buscando ser a mais abrangente e completa possível, fazendo com que os dados coletados nos dessem o maior detalhamento possível da rede e dos sistemas monitorados.

O formato dos logs é importante porque através dele foi construído scripts MapReduce para trabalhar nas tarefas de ordenação. Por exemplo, temos a seguir o formato padrão de um log de um honeypot gerado pelo sistema Honeyd, **2007-07-01-00:00:09.5428 tcp(6) S 76.97.76.42 3306 201.59.23.109 1080 [Windows XP SP1].** Dessa forma, foi mapeado os campos desse log os quais permitiram extrair e executar mais eficiente informações desses dados. Foram separados 8 campos, *timestamp*, protocolo, conexão (início (S) e fim (E)), IP de origem, porta de origem, IP de destino, porta de destino e sistema operacional do atacante. O mesmo procedimento foi feito para os logs do sistema de detecção de intrusão, porém com outros campos. A seguir temos um exemplo do formato do log do IDS, **COMMUNITY SIP TCP/IP message flooding directed to SIP proxy [Attempted Denial of Service] [Priority: 2] 02/22-09:38:18.721749 201.14.100.126:51857 -> 164.X.X.33:80**, assim foram mapeados o evento registrado, a classificação do ataque, prioridade, *timestamp*, IP de origem, porta de origem, IP de destino e porta de destino. Os demais logs tiveram o mesmo tratamento. A figura 4-3 exemplifica parte de um script criado para correlacionar os logs dos sensores IDS com os do honeypot, para extrairmos a informação de qual modalidade de ataque foi mais utilizada para atacar nossos servidores e através de qual sistema operacional, informação até então impossível de ser obtida nas abordagens atuais de análise de logs.

```
1  ids_logs = LOAD /cloud/logdoop/ids_logs AS (events, classification, priority,
2   timestamp, source_ip, source_port, dest_ip, dest_port);
3  honey_logs = LOAD /cloud/logdoop/honey_logs AS (timestamp, protocol, conection,
4   source_ip, source_port, dest_ip, dest_port, os_attacker);
5  F1 = FILTER ids_logs BY classification;
6  F2 =  FILTER honey_logs BY os_attacker;
7  J1 = JOIN F1 BY dest_ip, F2 BY dest_ip);
8  G1 = GROUP G1 BY dest_ip;
9  STORE Result INTO `correlation_os_vs_class_by_dest_ip´;
10
```

Figura 4-3 - Correlação de logs

Perceba que esse resultado tem um alto valor agregado. Gigabytes de dados foram transformados em informação extremamente útil. Manusear uma quantidade massiva de dados sempre foi um grande problema. Agora temos uma coleta, armazenamento e processamento distribuído, que agrega maior poder de transformação de dados em

informação através da eficiente e produtiva manipulação desses arquivos. Pode-se realizar grandes estudos da rede, do comportamento dos atacantes, formas de ataque, sistemas operacionais mais utilizados e assim criar mecanismos de contenção e aperfeiçoamento das técnicas atuais de prevenção a intrusão.

Na tentativa de resolução para o tratamento de grandes volumes de logs de segurança, as abordagens utilizadas por outros autores não resolvem o problema por completo, tendo em vista que soluções adotadas para processamento não incluíam o armazenamento ou vice-versa, adicionando o problema de dificuldade de escalonar as soluções para grandes demandas.

No ambiente em nuvem proposto, o armazenamento distribuído através do HDFS provê grande capacidade de armazenamento aliado à características essenciais de tolerância a falhas, como replicação de dados e tratamento automático de falhas que juntamente com o paradigma de programação MapReduce, fornece um processamento paralelo altamente eficiente e também mais tolerante a falhas. Temos a coleta, armazenamento e processamento distribuídos e escaláveis. A gerência e execução desse sistema é um tanto complexa e buscando a melhoria dos resultados propostos no tratamento dos dados, aumento da produtividade, clareza da informação, automatização de processos, portabilidade da informação, uma interface foi criada para contribuir ainda mais com a pesquisa realizada, com um resultado extremamente aplicado aos problemas atuais e podendo servir como uma ferramenta única no mercado atual. Passa-se a ter outra visão do mundo do Big data, quanto mais dados mais informação, a um custo extremamente acessível!

Na construção dessa interface, foi pensado formas para facilitar a gerência da nuvem criada. Assim, é possível de forma prática ter acesso às informações atuais do estado da nuvem, adicionar e remover nós, definir o índice de replicação dos dados, fazer testes de desempenho, verificar os trabalhos MapReduce submetidos, a execução das funções Map e Reduce e capacidade de armazenamento, por exemplo. A seguir temos imagens que demonstram essas funcionalidades:

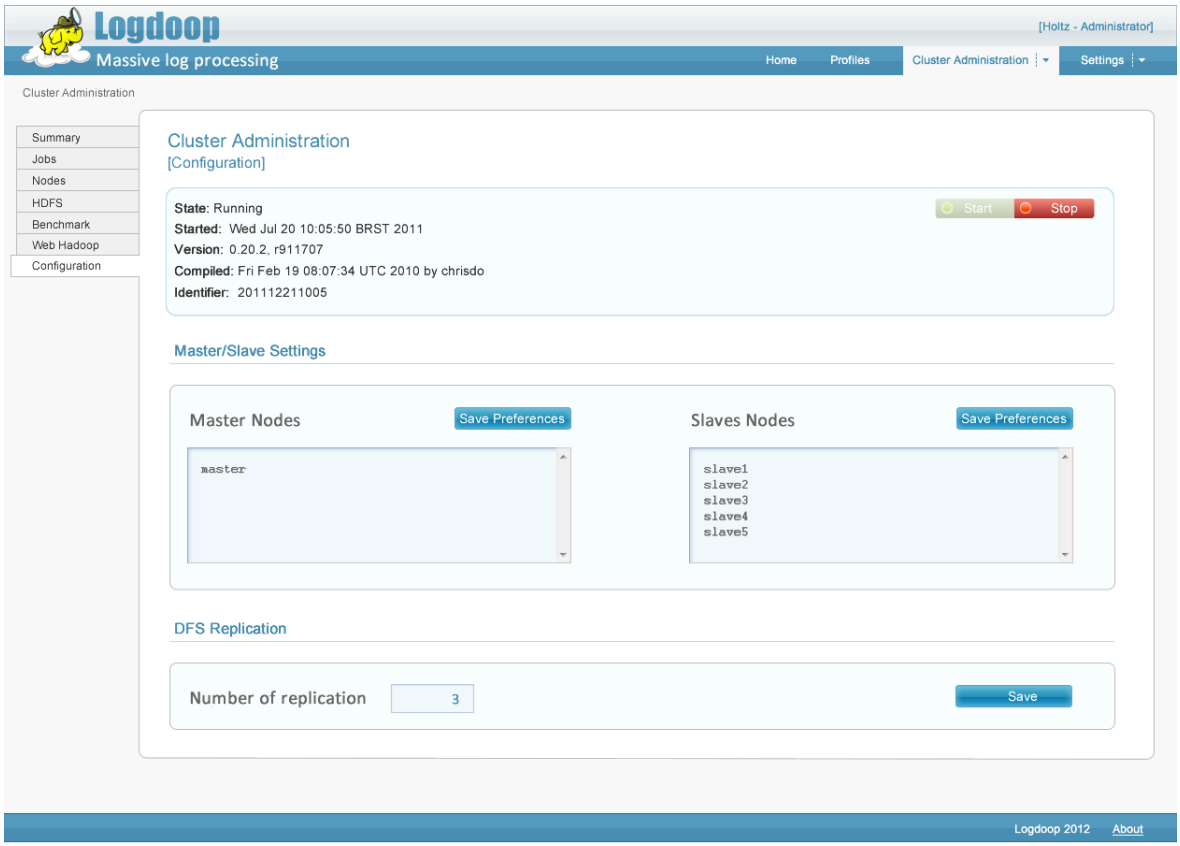


Figura 4-4 - Logdoop - Administração da nuvem

Na figura 4-4 temos a estrutura de um nó master e cinco nós slaves. O cluster está ativo, pronto para receber os dados e processá-los através dos scripts. Foi escolhido o índice de replicação três, dessa forma o mesmo dado será armazenado em três datanodes diferentes. A figura 4-5 demonstra o resultado de um teste de desempenho realizado. Foi feita a leitura de um arquivo de 1TB, onde o *throughput* obtido foi de aproximadamente 14 Gigabytes por segundo, taxa extremamente alta e que nós dá alto poder de processamento e que poderia ser maior se acrescentássemos mais máquinas à nuvem.

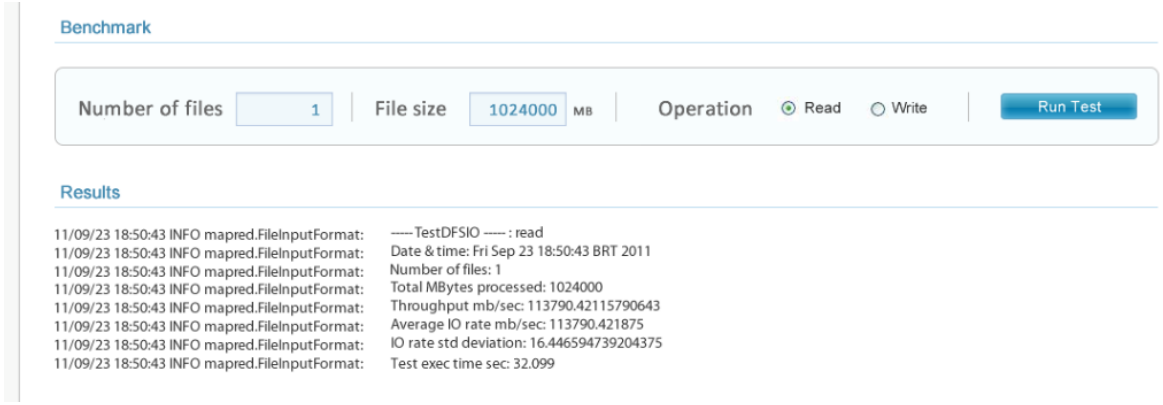


Figura 4-5 - Logdoop -Teste de performance

A figura 4-6 exemplifica o estado dos trabalhos MapReduce e como as funções são executadas.

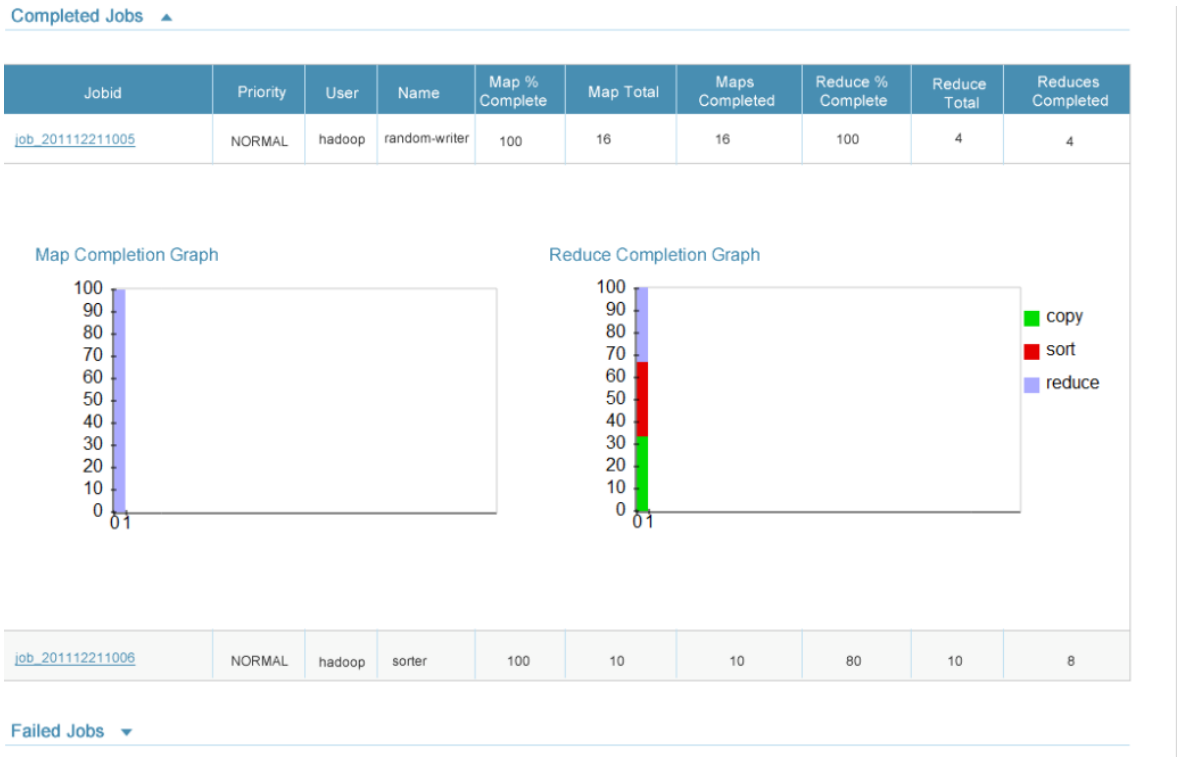


Figura 4-6 - Execução dos trabalhos MapReduce

Para a manipulação dos logs coletados, foram criados perfis que serão responsáveis pelo tratamento da informação. Através deles é possível analisar os dados e extrair informações minuciosas de acordo com um determinado período, hosts mais atacados, tipos de ataques além de correlação de dados para obter dados cruzados e de difícil acesso quando comparado a outros sistemas de análise. Na figura 4-7 temos a definição dos perfis de acordo com o formato dos logs, sendo possível ainda analisar um dado externo inserido ao sistema e não somente o que é coletado pelo próprio sistema.

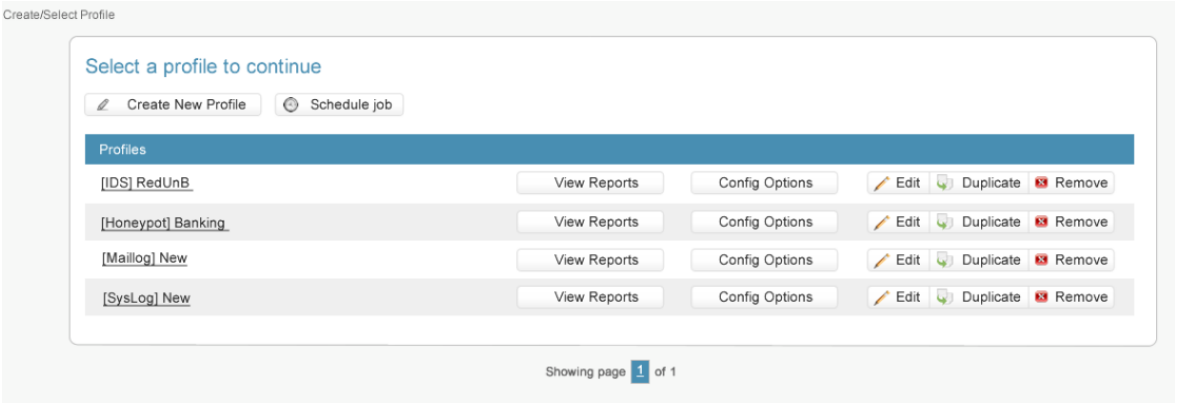


Figura 4-7 - Logdoop - Escolha de perfis de análise

Os dados podem ser analisados de acordo com uma data específica se assim for desejado, trazendo maior controle e filtragem da informação.

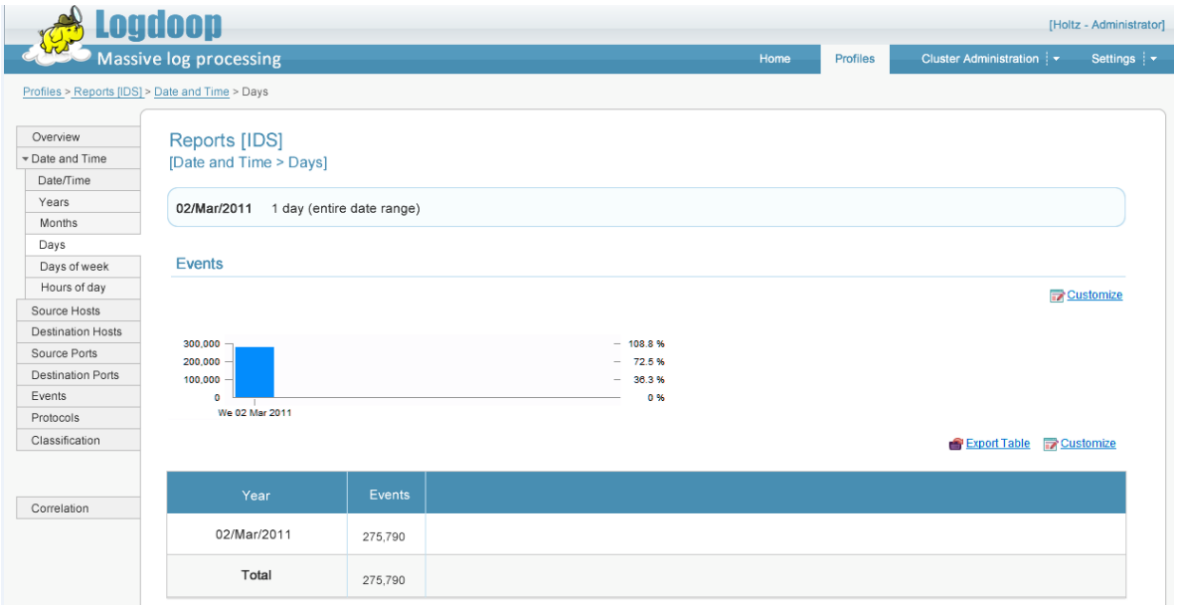


Figura 4-8 - Logdoop – Eventos por dia

Nesse exemplo, demonstrado pela figura 4-8, temos que apenas em único dia de coleta, tivemos 275,790 eventos coletados pelo sistema IDS. Informações mais específicas e aprofundadas podem ser extraídas através de outros campos do sistema para análise de logs.

Podemos obter a classificação dos principais tipos de ataques sofridos pelo sistema, o que nos ajuda a trabalhar de forma preventiva e corretivamente criando ações e políticas de segurança que venham a ser mais eficazes no combate dessas atividades maliciosas. Na figura 4-9 temos a classificação dos principais ataques sofridos em uma determinada data.

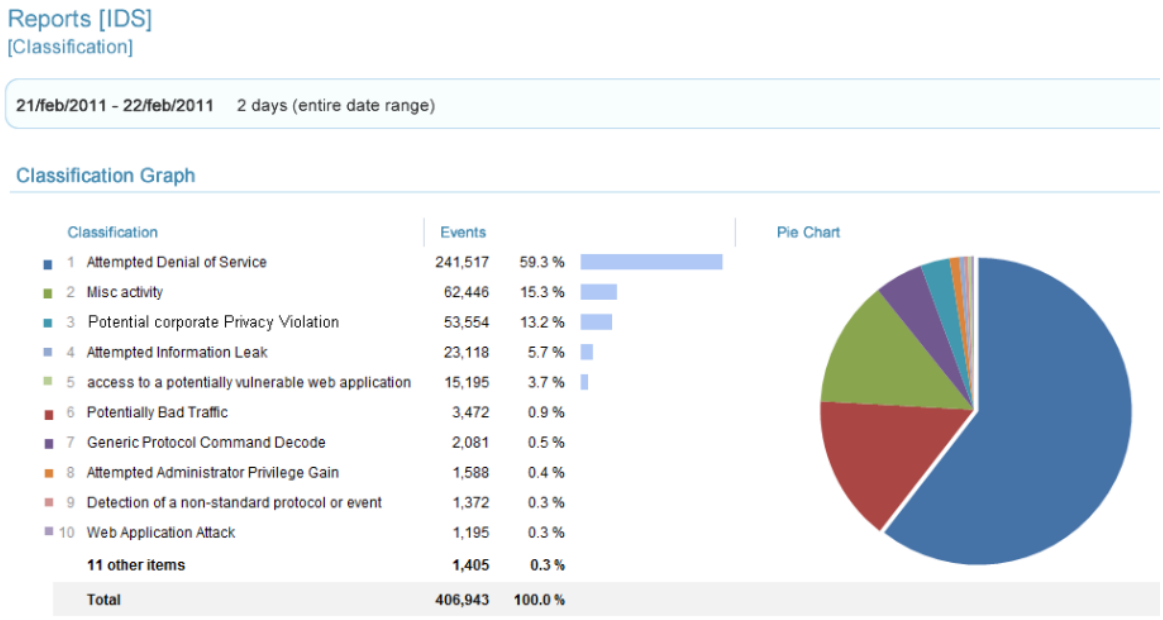


Figura 4-9- Top 10 Classificação de ataques

Através da correlação dos logs conseguimos extrair informações valiosas, como por exemplo, o comportamento de atacantes, quais sistemas operacionais eles mais utilizam, qual tipo de ataque mais executado, quais ferramentas utilizadas e etc. Tais informações não seriam possíveis de serem obtidas através de uma simples análise, e essa plataforma nos permite obter esses dados de alto valor agregado. A imagem 4-10 exemplifica algumas formas de correlação e alguns os resultados que podem ser obtidos.

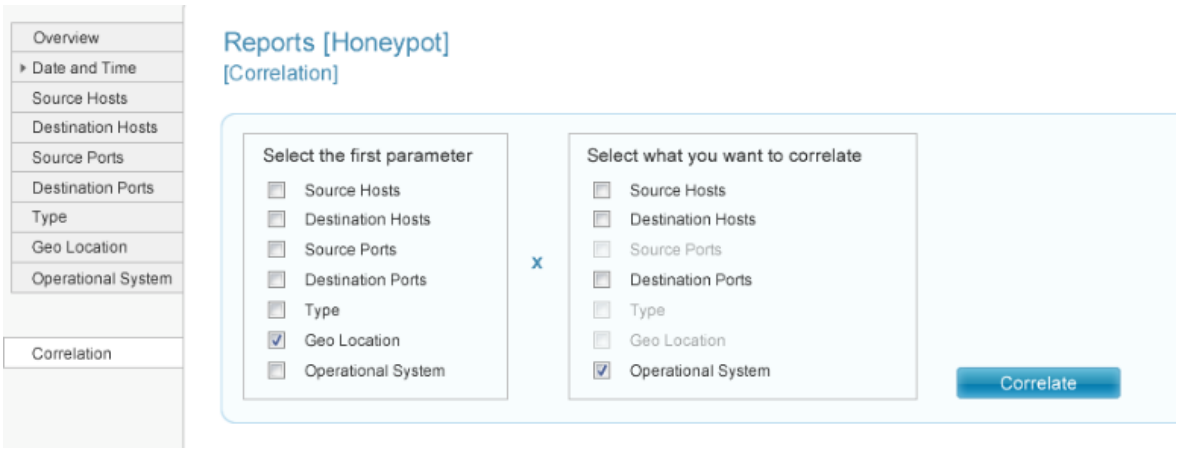


Figura 4-10- Logdoop – Correlação de informação

Gráficos obtidos através dessa correlação.são demostrados através da figura 4-11.

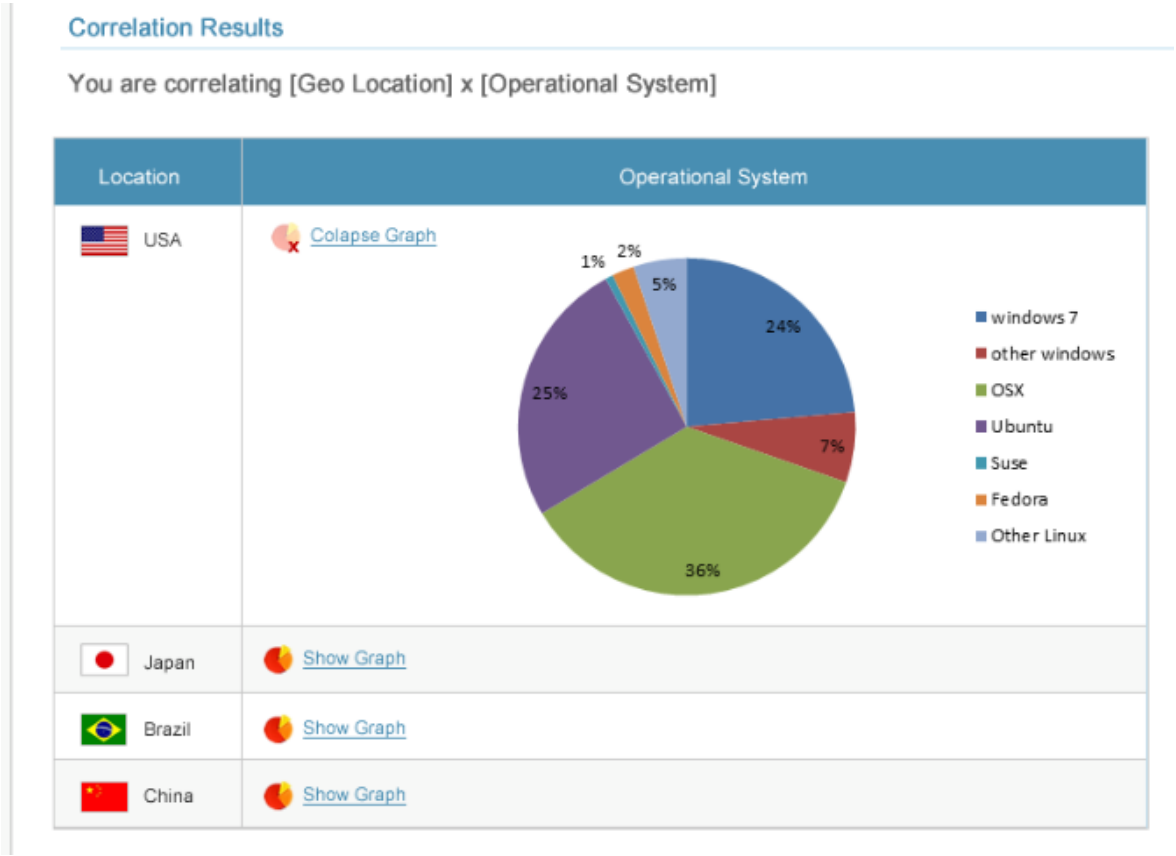


Figura 4-11 - Resultado - Correlação

Pesquisas realizadas pelo instituto Gartner (Pettey, 2011) mostram que nos EUA, o sistema operacional mais utilizado é o Windows 7, sendo o MacOSX apenas o terceiro na preferência. Porém, ao correlacionarmos os logs dos ataques coletados pelo honeypot, percebemos uma variação na popularidade dos sistemas operacionais, sendo o MacOSX o sistema mais utilizado pelos atacantes e sistemas operacionais *Unix/Unix like* a grande maioria utilizada nas tentativas de perpetrar ataques, chegando a quase 70%. Em usuários comuns, as estatísticas mostram que nos EUA, por exemplo, a preferência de sistemas operacionais Windows chega a ser cerca de 85%. Esses resultados são extremamente interessantes e úteis para o estudo de perfis dos atacantes, informações que podem ser utilizadas pra um estudo mais aprofundado sobre os ataques realizados e perfis de atacantes. Conhecer bem o comportamento dos atacantes é fundamental. Com todos os logs coletados, temos informações para entender melhor como nosso sistema está se comportando e como nos proteger.

5. CONCLUSÃO

A ineficiência no tratamento da informação gerada por sistemas de detecção de intrusão e honeypots aliada ao crescimento massivo da quantidade de dados trafegado em redes de larga escala, juntamente com necessidade de mecanismos mais eficazes na mitigação de ataques cada vez mais complexos, foi o fato motivador da pesquisa realizada. O presente trabalho mostrou de forma inovadora mecanismos distribuídos de coleta, armazenamento e processamento de dados, utilizando a computação em nuvem e o paradigma de computação MapReduce. A pesquisa realizada atacou problemas antes abordados por vários autores, os quais não conseguiram resolver por completo problemas no tratamento de grandes massas de dados aplicados aos problemas de segurança da informação.

A arquitetura proposta na pesquisa resultou num sistema de tratamento de dados totalmente distribuído e escalável, capaz de atender às altas demandas de tráfego, processamento e armazenamento exigidos pelas redes atuais. A coleta distribuída através dos sensores em vários níveis gera informações importantes na detecção de ataques complexos, o que era um trabalho inviável computacionalmente em outras abordagens. Todos os dados coletados são transformados em informação útil no trabalho de proteção aos sistemas, análise de tendências comportamentais de atacantes, criação de novas assinaturas de ataques, auxiliando na criação de contramedidas que irão fortalecer as entidades nas batalhas travadas contra invasores.

Toda a informação gerada nos dá subsídios para entender melhor como nosso sistema atua e qual o comportamento dos atacantes. Como menciona Sun Tzu, “Se você conhece seu inimigo e conhece a si mesmo, não precisa temer o resultado de cem batalhas. Se você se conhece mas não conhece o inimigo, para cada vitória ganha sofrerá também uma derrota. Se você não conhece nem o inimigo nem a si mesmo, perderá todas as batalhas.” (Tzu, 1994).

Como trabalhos futuros espera-se trabalhar na melhoria dos próprios sistemas de detecção de intrusão através do uso de técnicas de PCA promovendo a paralelização dos algoritmos de RADOI, melhorando consideravelmente e de forma automática a detecção de novos ataques sem a necessidade prévia de assinaturas.

REFERÊNCIAS BIBLIOGRÁFICAS

Alata, E., M. Dacier, M., Y. Deswarte, Y., and others, “Collection and analysis of attack data based on honeypots deployed on the Internet” in Quality of Protection, Advances in Information Security, D. Gollmann, F. Massacci, and A. Yautsiukhin, Eds. Springer US, 2006, vol. 23, pp. 79–91.

Almotairi, S., Clark, A., Mohay, G. and Zimmermann, J. “Characterization of attackers’ activities in honeypot traffic using principal component analysis,” in Proceedings of the 2008 IFIP International Conference on Network and Parallel Computing. Washington, DC, USA: IEEE Computer Society, 2008, pp. 147–154.

Almotairi, S., Clark, A., Mohay, G. and Zimmermann, J. “A technique for detecting new attacks in low-interaction honeypot traffic” in Proceedings of the 2009 Fourth International Conference on Internet Monitoring and Protection. Washington, DC, USA: IEEE Computer Society, 2009, pp. 7–13.

Andrew, J. “The Open Group Base Specifications Issue 7, IEEE Std 1003.1TM – 2008, in http:// http://pubs.opengroup.org/onlinepubs/9699919799/” acessado em 17/08/2011.

Anderson, J.P. “Computer security threat monitoring and surveillance”, Technical Report, Fort Washington, Pennsylvania, 1980.

Anil, R “Apache Mahout” http://lucene.apache.org/mahout acessado em 12/08/2011.

Axelsson, S. “Intrusion detection systems: A survey and taxonomy,” Tech. Rep. 2000.

Balasubramanuyan, J., Garcia-Fernandez, J., Isacoff, D., Spafford, E. and Zamboni, D. “An architecture for intrusion detection using autonomous agentes” in Computer Security Applications Conference, 1998, Proceedings, 14th Annual, december 1998, pp 13-24.

Bray, T., Paoli, J., Sperberg-McQueen, C.M. and others, “Extensible mark-up language (XML) 1.0”, in W3C, 2000.

Bu, L. and Chandy, J.A. “FPGA based network intrusion detection using content addressable memories,” Field-Programmable Custom Comput-ing Machines, Annual IEEE Symposium on, vol. 0, pp. 316–317, 2004.

Buya R. and others, “High Performance Cluster Computing: Architectures and Systems, volume 1, Prentice Hall, PTR, pages 327-350, 1999.

Cutting, D. “Ten Common-Hadoopable Problems” – White paper – Cloudera 2011.

David, B.M., Costa, J.P.C.L., Nascimento, A.C.A., Holtz, M.D., Amaral, D. and Sousa Jr., R. T. "Blind Automatic Malicious Activity Detection in Honeypot Data," The International Conference on Forensic Computer Science (ICoFCS) 2011, Florianópolis, Brazil.

David, B.M., Costa, J.P.C.L., Nascimento, A.C.A., Holtz, M.D., Amaral, D. and Sousa Jr., R. T. "A Parallel Approach to PCA Based Malicious Activity Detection in Distributed Honeypot Data," in International Journal of Forensic Computer Science (IJoFCS), 2012.

Dean, J., Ghemawat, S. “MapReduce: a flexible data processing tool”, in Communications of the ACM, volume 56, janeiro 2010.

Debar, H., Dacier, M., and Wespi, A. “Towards a taxonomy of intrusion-detection systems,” Computer Networks, vol. 31, no. 8, pp. 805–822, 1999.

Denning, D.E. “Intrusion Detection Model”, IEEE Transactions On Software Engineering, vol SE-13, n^{o} 2, pp 222-232, fevereiro 1987.

El-Sonbaty, Y., Ismail, M.A. and Farouk, M. ”An Efficient Density Based Clustering Algorithm for Large databases,” ICTAI 2004.16th IEEE International Conference, pp673 - 677.

Ester, M., Kriegel, H., Sander, J, Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996.

Ghemawat,. S. and Dean, J. “MapReduce: Simplified data processing on large clusters”, in Symposium on Operating System Design and Implementation OSDI’04, San Francisco, CA, USA, 2004.

Ghemawat, S., Gobioff, S., Leung, T. "The Google file system", ACM SIGOPS Operating Systems Review, vol. 37, n. 5, pp. 29-43, 2003.

Ghosh A. and Schwartzbard, A. “A study in using neural networks for anomaly and misuse detection,” in Proceedings of the 8th conference on USENIX Security Symposium-Volume 8 . USENIX Association, 1999, p. 12.

Ghourabi, A., Abbes, T., and Bouhoula, B. “Data analyzer based on data mining for honeypot router,” in Computer Systems and Applications, ACS/IEEE International Con-ference on, vol. 0, pp. 1–6, 2010.

Gosling, J., Joy, B., Steele, G. and Bracha, G. “JavaTM Language Specification, The 3rd Edition”, 2005.

Graefe, G., Kuno, H. “Modern B-tree techniques” in ICDE IEEE, pages 1370 – 1373, maio 2011.

Guangchun, L. Xianliang, Jiong, L. and Jun, Z. “Madids: a novel distributed ids based on mobile agente”. SIGOPS Oper. Syst. Ver., vol 37, pp. 46 – 53, january 2003.

He, W., Hu, G., Yao, X., Kan, G., Wang, H., and Xiang, H. “Applying multiple time series data mining to large-scale network traffic analysis,” in Cybernetics and Intelligent Systems, 2008 IEEE Conference on, September 2008, pp.394 –399.

Holtz, M.D., David, B.M., Sousa Jr., R.T “Building Scalable Distributed Intrusion Detection Systems Based on the MapReduce Framework” in Revista Telecomunicações, vol. 13, nº 02, Dezembro de 2011.

Holtz, M.D., David, B.M., Peotta, L. and Sousa Jr., R.T, “An architecture for distributed network intrusion detection based on the map-reduce framework,” in Proceedings of the

International Workshop on Telecommunications - IWT 2011, C. A.Ynoguti and M. C. de Paiva, Eds., 2011, pp. 106–110.

Korpela, E., Werthimer, Anderson, D. and others, "SETI@home-massively distributed computing for SETI", Comuting in Science & Engineering, january 2001.

Larson, S., Snow, C. and others, "Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology", Citeseer, 2002.

Lee, W. and Stolfo, S.J. "Data mining approaches for intrusion detection," in Proceedings of the 7th conference on USENIX Security Symposium – Volume 7. Berkeley, CA, USA: USENIX Association, 1998, pp. 6-6.

Lee, Y., Kang, W. and Son, H. "An internet traffic analysis method with mapreduce," in Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP , April 2010, pp. 357 –361.

Maheswari, V. and Sankaranarayanan, P. E. "Honeypots: Deployment and data forensic analysis," in Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007) -Volume 04, ser. ICCIMA '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 129–131.

Mell, P., Grance, T. "The NIST Definition of Cloud Computing", NIST Special Publication 800-145, september 2011.

McCarthy, J. "History of LISP", in History of Programming languages I", pages 173-185, ACM, 1978.

Morris, R. Truskowski, B. "The evolution of storage systems", IBM Systems Journal, vol 42, pages 205 – 217, 2003.

Mokube, I., Adams, M. Honeypots: concepts, approaches, and challenges in ACM-SE 45 Proceedings of the 45th annual southeast regional conference ACM, New York, NY, 2007.

Ordonez, C. Cereghini, P. “SQLEM: fast clustering in SQL using the EM algorithm.” in SIGMOD’00, 2000.

Pettey, C. http://www.gartner.com/DisplayDocument?ref=clientFriendlyUrl&id=1634314, acessado em 20/11/2011.

Phillips, T. “ET, phone SETI@home”, Nasa, may 1999.

Pollari-Malmi, K., Soisalon-Soininen, E., Yloren, T. “ Concurrency control in B-trees with batch updates” in IEEE transactions, Knowledge and Data Engineering, volo 8, pages 975-984, dezembro 1996.

Porras, P. and Neumann, P. “EMERALD: Event monitoring enabling responses to anomalous live disturbances,” in Proceedings of the 20th National Information Systems Security Conference, 1997, pp.353–365

Sang, Y., Yi, Z. “Motion Determination Using Non-Uniform Sampling Based Density Clustering.”, Fifth International Conference on Fuzzy Systems and Knowledge Discovery, 2008.

Snir, M. “MPI – The complete Reference: The MPI-2 extentions”, vol 2, MIT press, 1998.

Sourdis, I. and Pnevmatikatos, D. “Fast, large-scale string match for a 10gbps fpga-based network intrusion,” FPL , vol. 2003, pp. 880–889, 2003.

Spitzner, L. “Honeypots: Catching the insider threat” in Proceedings of the 19th Annual Computer Security Applications Conference, ser. ACSAC ’03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 170

Spitzner, L. “Honeypots: tracking hackers". Addison-Wesley Professional, 2003

Spitzner, L. “Honeytokens: The other honeypot” in Security Focus, 2003

Stoncbrakcr, M “Inclusion of new types in relational data base systems” in Readings in database systems, The MIT Press, vol 39, number 82-C, pages 459, 2005

Sun Tzu, “Art of War”, Basic Books, 1994.

Yang, H., Dasdan, A., Hsiao, R., Parker, S. “Map-reduce-merge: simplified relation data processing on large clusters” in SIMOD’07, 2007.

Yang, S., Chen, W. and. Wang, Y. “Icas: An inter-vm ids log cloud analysis system” in Proceedings of the International Conference on Cloud Computing and Intelligence Systems - CCIS 2011.

Woltman, G., Kurowski, S. “The great internet prime search, journal http://www.mersenne.org acessado em 15/09/2012.

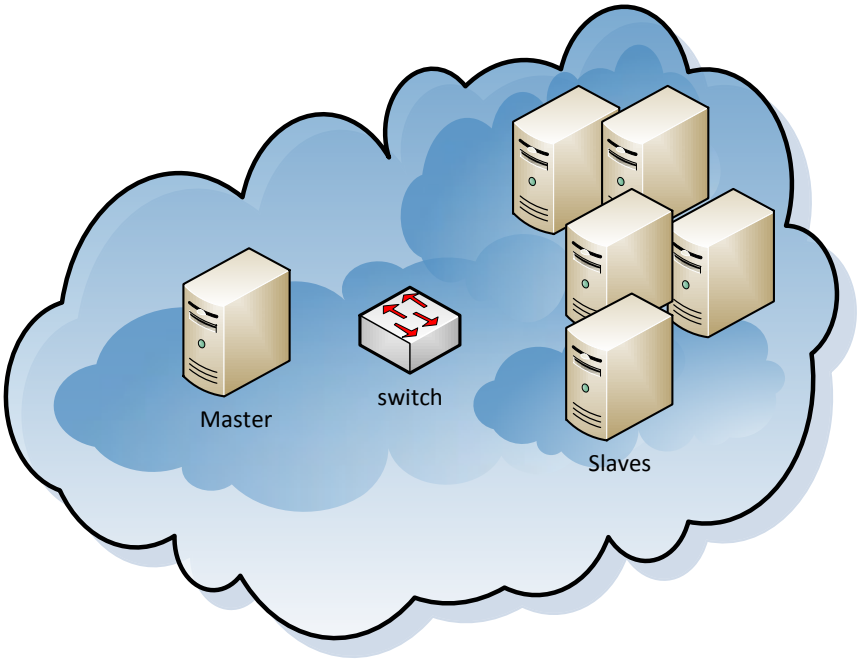
White, T. “Hadoop: The Definitive Guide”, first edition .,M. Loukides, Ed. O’Reilly, june 2009.

APÊNDICES

A INFRAESTRUTURA DO CLUSTER

O cluster foi formado com 6 máquinas obedecendo a seguinte configuração:

- Intel® Core 2 Duo CPU E7500 2.93 GHz
- 4 GB DDR 667 RAM
- 300 GB de disco rígido
- Switch ethernet 10/100



Nuvem - Hadoop

B INSTALAÇÃO E CONFIGURAÇÃO DO CLUSTER

Para a preparação do ambiente Hadoop, foi instalado nas máquinas que formaram o cluster, o sistema operacional Debian 5.0 Lenny. Para a instalação e configuração do ambiente os seguintes passos foram realizados:

- Instalação da JVM:

Adicionar o seguinte repositório em */etc/apt/sources.list*

```
deb http://archive.canonical.com/ lucid partner
```

Após feito isso, deve-se atualizar a lista dos repositórios:

```
apt-get update
```

Instalação do Java Development Kit (JDK)

```
apt-get install sun-java6-jdk
```

Verificar após a instalação, se o ambiente jabá foi corretamente instalado e encontra-se funcional:

java –version

Esse comando deverá retornar a resposta da versão instalada. Após isso deve-se criar um usuário específico e exclusivo para o Hadoop;

```
# addgroup hadoop
# adduser --ingroup hadoop hadoop
```

- Instalação do Hadoop

Baixar a última versão do Hadoop no site http://hadoop.apache.org/.

Descompactar em /usr/local/hadoop/ e dar as permissões de usuário e grupo para hadoop.

```
# cd /usr/local
# tar xzf hadoop-0.20.2.tar.gz
# mv hadoop-0.20.2 hadoop
# chown -R hadoop:hadoop hadoop
```

- Configuração do Hadoop

Em /conf/ dentro do diretório de instalação, deve se alterar os arquivos:

- hadoop-env.sh
- core-site.xml
- mapred-site.xml
- hdfs-site.xml

Alterando a variável de ambiente em hadoop-env.sh:

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

Em core-site.xml:

```
<!-- In: conf/core-site.xml -->
<property>
  <name>fs.default.name</name>
  <value>hdfs://master:54310</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
  uri's scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uri's authority is used to
  determine the host, port, etc. for a filesystem.</description>
</property>
```

Em mapred-site.xml:

```
<!-- In: conf/mapred-site.xml -->
<property>
  <name>mapred.job.tracker</name>
  <value>master:54311</value>
```

```
  <description>The host and port that the MapReduce job
tracker runs
  at.  If "local", then jobs are run in-process as a single
map
  and reduce task.
  </description>
</property>
```

Em hdfs-site.xml:

```
<!-- In: conf/hdfs-site.xml -->
<property>
  <name>dfs.replication</name>
  <value>3</value>
  <description>Default block replication.
  The actual number of replications can be specified when the
file is created.
  The default is used if replication is not specified in
create time.
  </description>
</property>
```

Nó nó mestre, em /conf/slaves, deve-se definir os nós que serão os escravos:

```
slave1
slave2
slave3
slave4
slave5
```

E em /conf/master:

```
master
```

Definido os arquivos com o nó mestre e os escravos, deve-se gerar as chaves de acesso ssh. (continuar)

Após isso, deve-se formatar o HDFS (Hadoop's distributed filesystem) para dar início ao funcionamento do cluster:

```
hadoop@master:/usr/local/hadoop$ bin/hadoop namenode –format
```

O cluster poderá então ser inicializado:

```
/bin/start-all.sh
```