Sistema de Scripts para monitoramento Linux

Itamar Bernardo da Silva Júnior
Ciência da computação
Universidade Federal do Agreste de Pernambuco - UFAPE
Garanhuns - PE, Brasil
itamarbernardo2013@gmail.com

Jefferson Santos Alves
Ciência da computação
Universidade Federal do Agreste de Pernambuco - UFAPE
Garanhuns - PE, Brasil
salves.jefferson@gmail.com

Jonathan Matos
Ciência da computação
Universidade Federal do Agreste de Pernambuco - UFAPE
Garanhuns - PE, Brasil
jonathan2018.matos@gmail.com

Laisy Cristina Ferreira Silva
Ciência da computação
Universidade Federal do Agreste de Pernambuco - UFAPE
Garanhuns - PE, Brasil
laisyferreira2011@gmail.com

Weverton Antonini Cordeiro Cintra
Ciência da computação
Universidade Federal do Agreste de Pernambuco - UFAPE
Garanhuns - PE, Brasil
weverton.cintra.1@gmail.com

***Abstract*—For an efficient information system, it is necessary that there is a good functioning of its resources, full communication, exchange of information and decision-making. Based on this, this article proposes the creation of a system for creating Bash, Python and Java scripts on Linux systems, for evaluating machine resources.**

***Resumo*—Para um sistema de informação eficiente, é necessário que haja um bom funcionamento dos seus recursos, comunicação plena, troca de informações e tomada de decisões. Com base nisso, este artigo propõe a criação de um sistema de criação de scripts Bash, Python e Java em sistemas Linux, para avaliação de recursos de máquina.**

***Index Terms*—Linux, Memória, CPU, Rede, Avaliação de Desempenho, Recursos de Máquina.**

I. Introdução

Atualmente vivemos a era da informação, com isso os sistemas de informação têm um papel fundamental para gestão das empresas, competitividade, tomada de decisões, bem como, no dia a dia, para usuários se manterem informados e em constante comunicação. Entretanto, nem sempre esses sistemas atendem adequadamente às necessidades dos seus usuários. Normalmente essa dificuldade ocorre devido a algum problema ou deficiência do próprio sistema. A fim de encontrar e analisar tais problemas faz-se necessário a avaliação de tais SIs.

Um sistema de informação considerado eficiente, é necessário que haja um bom funcionamento dos seus recursos, comunicação plena, troca de informações e tomada de decisões. Nesse contexto, é essencial que os administradores possam garantir que seus sistemas estejam configurados da melhor maneira possível.

Para realizar avaliações de sistemas é necessário utilizar algum modelo, ferramenta ou outro sistema, que permita identificar e obter dados necessários para verificar se o sistema está satisfazendo ou não as necessidades das empresas/usuários, se está funcionando de maneira correta e consumindo os componentes de Hardware ideais.
Assim, avaliar o desempenho constantemente desses sistemas se torna extremamente necessário. O primeiro passo para realizar as tarefas com maior eficiência é monitorar o sistema em questão [Sottile and Minnich 2002].

Já existem diversos sistemas e métodos para monitoramento disponíveis, utilizados para monitorar recursos como memória RAM, CPU, rede, entre outros. A maioria das soluções concentra-se em utilizar scripts para gerar relatórios.

Este trabalho tem como objetivo o desenvolvimento de uma aplicação para monitoramento de sistemas Linux. Neste processo, serão descritas as ferramentas, passos e resultados envolvidos. Como o presente projeto, espera-se a criação de um sistema web flexível para gerar scripts de monitoramento personalizados, usando as seguintes linguagens: Bash, Python e Java. Pretendemos, também, fazer um experimento para comparação desses scripts nas três linguagens apresentadas.

II. Fundamentação Teórica

A. Conceitos e definições

Segundo [Eranian, S. 2006] o monitoramento de desempenho é a ação de selecionar informações sobre a execução de um programa. O tipo de informação coletada depende do nível em que é coletado. Ela ainda distingue em dois níveis:
Nível do Programa:

O programa é instrumentado adicionando chamadas explícitas a rotinas que coletam determinadas métricas. Com essas ferramentas, é possível coletar, por exemplo, o número de vezes que uma função é chamada.
Nível do Hardware:

Aqui o programa não é modificado. As informações são coletadas por hardware da CPU, por exemplo. Medem o micro-comportamento arquitetural do programa, ou seja, o número de ciclos decorridos, quantos o cache de dados para, quantas perdas de TLB.

Quando falamos em scripts de monitoramento linux, via bash, por exemplo, consideramos o nível de hardware, é através dele que observamos como os recursos de hardware apresentam opções de obter melhorias consideráveis no desempenho do sistema. Para [Eranian, S. 2006] o monitoramento de desempenho baseado em hardware pode ajudar a identificar problemas em como o software usa esses novos recursos de hardware.

Dessa forma, o monitoramento de desempenho é um processo crucial para garantir que o sistema está funcionando de forma adequada. Podemos coletar informações sobre o desempenho do sistema, como utilização de recursos, atividade de rede, capacidade de armazenamento e outras métricas relevantes. Esses dados podem ser analisados para identificar possíveis problemas e garantir que o sistema esteja funcionando de acordo com as expectativas.

O monitoramento de desempenho tem várias vantagens, uma delas é a capacidade de prever e evitar problemas antes que eles ocorram. Ao analisar os dados de desempenho do sistema, os administradores podem identificar padrões e tendências que podem indicar possíveis problemas no futuro. Isso permite que os administradores tomem medidas preventivas para evitar possíveis interrupções no sistema.

Outra vantagem do monitoramento de desempenho é a capacidade de otimizar o desempenho do sistema. Ao monitorar a utilização de recursos e outras métricas, os administradores podem identificar áreas onde o sistema está subutilizado ou super utilizado. Eles podem, então, fazer ajustes no sistema para garantir que ele esteja funcionando de forma ideal.

Alguns comandos de monitoramento no sistema Linux são apresentados na Tabela I [Andresen 2004, sys 2015].

Na tabela II descrevemos alguns comandos úteis utilizados na construção dos scripts bash.

Python oferece umas algumas bibliotecas para esse tipo de monitoramento. Consideramos a psutil, pois é uma biblioteca de plataforma cruzada para recuperar informações sobre processos em execução e utilização do sistema. É útil principalmente para monitoramento de sistemas, imitação de recursos de processo e gerenciamento de processos em execução. É possível monitorar CPU, memória, discos, redes e sensores através de seu uso. Na Tabela III, podemos ver algumas funções úteis na construção dos scripts em Python.

Java possui uma interface chamada OperatingSystemMX-Bean para colher informações do sistema. Tambem foi preciso utilizar a biblioteca Sigar para colher outros recursos que a interface não era capaz de fornecer. Na Tabela IV, vemos algumas funções úteis usadas na construção dos scripts em Java

III. Trabalhos Relacionados

Encontramos alguns trabalhos que apresentam características semelhantes às nossas, em [Araújo, Melo and Cordeiro 2021] os autores propõem uma abordagem para desenvolvimento de scripts de monitoramento flexíveis, utilizando a linguagem Shell-Script. O objetivo é a geração de relatórios personalizados e geração de gráficos dos resultados obtidos.

Em [Tongen and Stinson 2010], os autores exploram o monitoramento de dispositivos de rede, a fim de detectar inatividade em organizações. Isso possibilita identificar e resolver problemas de forma mais rápida. O objetivo é oferecer um melhor monitoramento de dispositivos de rede com representação visual aprimorada dos resultados.

Também existem alguns programas de monitoramento de sistemas que estão relacionados ao nosso trabalho. O Nagios é um software de código aberto que faz o monitoramento de rede e sistema amplamente utilizado, que ajuda a monitorar a disponibilidade e o desempenho dos serviços, dispositivos e aplicativos de uma rede. Ele fornece uma interface web para visualizar o status de dispositivos e serviços em tempo real, e também envia alertas quando problemas são detectados. O Nagios pode ser executado em uma variedade de plataformas, incluindo Linux, Windows e Unix. Também é possível fazer uso de plugins para monitorar os dispositivos e serviços da rede, e esses plugins podem ser personalizados para atender às necessidades específicas de monitoramento de uma organização.

O Zabbix é uma solução de software de monitoramento de rede e sistema de código aberto, projetada para monitorar a disponibilidade e o desempenho de serviços, aplicativos e dispositivos de rede. Normalmente, ele é usado por empresas para monitorar seus sistemas e garantir a disponibilidade contínua de seus serviços. Essa ferramenta oferece uma variedade de recursos de monitoramento, incluindo a capacidade de monitorar métricas de desempenho, como o uso da CPU, memória, espaço em disco e uso de rede. Ele também oferece a capacidade de monitorar serviços, como bancos de dados, servidores web e aplicativos de negócios personalizados. O Zabbix é altamente personalizável e pode ser estendido usando plugins e scripts personalizados. Ele também oferece uma API aberta para integração com outras ferramentas e sistemas de gerenciamento.

Tabela I
COMANDOS DE MONITORAMENTO BASH

Comando	Objetivo
top	Lista de processos ativos. Geralmente estão em ordem decrescente de uso da CPU.
uptime	A quanto tempo o sistema está ligado e a carga de trabalho média.
free	Informações sobre o estado da memória RAM e swap.
ps	Mostra o status dos processos dependendo dos parâmetros utilizados e do privilégio do usuário solicitante.
pidstat	Estatísticas para processos Linux.
/proc/cpuinfo	Mostra informações da CPU.
/proc/meminfo	Mostra informações da Memória RAM(Memória Total, Livre, Buffers, Cache, etc).

Tabela II
COMANDOS ÚTEIS BASH

Comando	Objetivo
echo	mostra texto na saída padrão.
grep	Faz buscas no conteúdo dos arquivos procurando linhas que encontrem a expressão regular mencionada.
awk	permite a manipulação de textos a partir de uma sequência de padrões.
sleep	pausa na execução do script.
while	comando de repetição.

Tabela III
COMANDOS DE MONITORAMENTO PYTHON

Comando	Objetivo
disk io counters	recupera as estatísticas atuais de E/S do disco.
cpu percent	utilização da CPU em porcentagem.
cpu count	número de CPUs lógicas no sistema.
cpu times(percpu=False)	retorna tempos de CPU para cada CPU lógica no sistema.
cpu times.user	retorna tempo gasto por processos normais executando no modo de usuário.
cpu times.system	retorna o tempo gasto pelos processos em execução no modo kernel.
cpu times.idle	retorna o tempo ocioso.
read bytes	velocidade de leitura.
write bytes	velocidade de escrita.
disk usage	retorna as estatísticas de uso do disco.
virtual memory	retorna estatísticas sobre o uso da memória do sistema.
Speedtest	realizar os teste de teste de conexão.
get best server	seleciona o servidor de origem da conexão.
net io counters	retorna as estatísticas de entrada/saída da rede em todo o sistema na forma de contadores.

Tabela IV
COMANDOS DE MONITORAMENTO JAVA

Comando	Objetivo
getCpuInfoList	Recupera informações da CPU, como quantidade de CPUs no sistema.
getCpuPerc	Utilização da CPU em porcentagem.
getCpu	Disponibiliza informações como: tempo gasto executando no modo usuário, tempo em modo kernel e tempo ocioso.
getTotalPhysicalMemorySize	Obtém o total de memória no sistema.
getFreePhysicalMemorySize	Obtém a quantidade de memória livre no sistema.
getFileStores	Obtém informações referentes ao disco, como Disco Usado e Livre.
getNetInterfaceStat	Obtém informações da rede, como bytes recebidos, bytes enviados, erros ao receber e enviar.

O SolarWinds Network Performance Monitor (NPM), é um software de monitoramento de rede que ajuda a monitorar a disponibilidade e o desempenho da rede em tempo real. Ele oferece recursos como mapeamento de rede, alertas personalizados, relatórios personalizados e análise de tendências. O SolarWinds NPM é escalável e pode monitorar milhares de dispositivos de rede em uma única instalação.
Incentivados pelos trabalhos relacionados, este artigo vai além, propondo uma plataforma flexível de geração de scripts de monitoramento, oferecendo ao usuário a possibilidade de gerar scripts para monitorar os seguintes recursos: Memória, CPU, Disco e Rede. Nosso trabalho também possibilita ao usuário a escolha da linguagem do script: Bash, Python e Java.

IV. IMPLEMENTAÇÃO

A. Descrição da estratégia proposta

Propomos o desenvolvimento de um software online que possibilite a geração de scripts de monitoramento no nível do hardware, simples e intuitivo para que, mesmo usuários com pouca experiência em linguagens de programação, consigam monitorar recursos importantes no seu computador. Como opção de monitoramento, propomos os recursos a seguir:

CPU, Memória, Disco rígido e Rede.

B. Descrição da implementação

O projeto foi criado por meio de uma interface web, utilizando de uma lib da linguagem Javascript, o ReactJS. Nosso software foi desenvolvido de forma que, para montar o script de monitoramento, o usuário só precisa clicar e arrastar o recurso desejado para a área indicada, selecionar a tecnologia em que o script será criado e o intervalo de tempo para o monitoramento. O usuário poderá escolher entre as tecnologias Bash, Python e Java. Para o intervalo de tempo entre as medições, poderá ser selecionado 1 min, 10 min e 1h.

Após as seleções do usuário, é necessário confirmar a geração dos scripts apertando um botão, e em seguida é mostrado os scripts de monitoramento selecionado pelo usuário.

C. Módulo Python

O módulo desenvolvido na linguagem Python também faz o monitoramento de recursos do sistema. Com a implementação deste módulo e do subsequente, os usuários podem executar os scripts em sistemas operacionais Windows, não sendo necessário a instalação de um WSL (Windows Subsystem for Linux). Em todos nossos scripts, utilizamos a biblioteca psutil. Esta biblioteca permite acessar informações sobre o sistema operacional e os recursos do sistema. A seguir, vemos os scripts utilizados no nosso sistema:

```python
def teste_memoria():

    mem = psutil.virtual_memory()
    mem_total = mem.total / (1024*1024*1024)
    mem_available = mem.available / (1024*1024*1024)
    mem_used = mem.used / (1024*1024*1024)
    mem_percent = mem.percent

    data_atual = datetime.now().strftime('%d/%m/%Y')
    hora_atual = datetime.now().strftime('%H:%M')

    return mem_total, mem_available, mem_used, mem_percent, data_atual, hora_atua
```

Figura 1. Script em Python para monitoramento de Memória.

No script da Figura 1, podemos ver o script de monitoramento da memória. Neste, utilizamos a função virtual_memory() da biblioteca “psutil” para colher informações relacionadas à memória no sistema.

```python
def teste_cpu():

    cpu_percent = psutil.cpu_percent(interval=1)

    cpu_count = psutil.cpu_count(logical=True)

    cpu_times = psutil.cpu_times(percpu=False)
    cpu_times_user = cpu_times.user
    cpu_times_system = cpu_times.system
    cpu_times_idle = cpu_times.idle

    data_atual = datetime.now().strftime('%d/%m/%Y')
    hora_atual = datetime.now().strftime('%H:%M')

    return cpu_percent, cpu_count, cpu_times_user, cpu_times_system, cpu_times_id
```

Figura 2. Script em Python para monitoramento de CPU.

Em relação à Figura 2, podemos ver o script de monitoramento da CPU. Neste, utilizamos a função cpu_percent(), cpu_count() e cpu_times() da biblioteca “psutil” para colher informações relacionadas à CPU no sistema.

```python
def teste_disco():
    disk_io_infos = psutil.disk_io_counters()

    disk_read_speed = (disk_io_infos.read_bytes / (1024*1024)) / (disk_io_infos.r
    disk_write_speed = (disk_io_infos.write_bytes / (1024*1024)) / (disk_io_infos

    disk = psutil.disk_usage('/')
    disk_used = disk.used / (1024*1024*1024)
    disk_total = disk.total / (1024*1024*1024)
    disk_percent = disk.percent

    data_atual = datetime.now().strftime('%d/%m/%Y')
    hora_atual = datetime.now().strftime('%H:%M')

    return disk_read_speed, disk_write_speed, disk_used, disk_total, disk_percent
```

Figura 3. Script em Python para monitoramento de Disco.

Em relação à Figura 3, podemos ver o script de monitoramento do disco. Neste, utilizamos a função disk_io_counters(), disk_usage() da biblioteca “psutil” para colher informações relacionadas ao Disco no sistema.

```python
def teste_internet():
    s = sp.Speedtest()
    s.get_servers()
    s.get_best_server()
    velocidade_download = round(s.download(threads=None)*(10**-6))
    velocidade_upload = round(s.upload(threads=None)*(10**-6))

    net_io_counters = psutil.net_io_counters()
    bytes_sent = net_io_counters.bytes_sent
    bytes_recv = net_io_counters.bytes_recv
    err_sent = net_io_counters.errout
    err_recv = net_io_counters.errin
    data_atual = datetime.now().strftime('%d/%m/%Y')
    hora_atual = datetime.now().strftime('%H:%M')

    return data_atual, hora_atual, velocidade_download, velocidade_upload, bytes_
```

Figura 4. Script em Python para monitoramento de Rede.

Já na Figura 4, podemos ver o script de monitoramento da Rede. Aqui fazemos uso da biblioteca speedtest que permite medir a velocidade da conexão de internet do usuário. Essa biblioteca faz uso da API do Speedtest.net e obtém resultados precisos de velocidade de upload e download. Como podemos observar na Figura 4, fazemos uso da função download() e upload() da biblioteca “speedtest” e também a função net_io_counters() da biblioteca “psutil” para obter informações da rede.

D. Módulo Bash

O módulo desenvolvido na linguagem Bash também faz o monitoramento de recursos do sistema. Nos nossos scripts, utilizamos os próprios comandos da linguagem para realizar os monitoramentos. A seguir, vemos um dos scripts utilizado no nosso sistema:

No script da Figura 5, podemos ver o script de monitoramento de Memória. Neste, fazemos uso dos seguintes comandos em Bash: “grep”, “awk” e “free” para coletar as informações da Memória. O comando “free” mostra informações sobre o uso de memória em sistemas baseados em Unix e Linux.

Já no script da Figura 6, podemos ver o script de monitoramento de CPU. Neste, foi usado o comando

```
memoriaRam(){

    #total de memoria ram
    mem=`free | grep  Mem  | awk '{print $2}'`
    memoriaTotal=$(($mem/1024))

    #memoria livre
    memFree=`free | grep  Mem  | awk '{print $4}'`
    memoriaFree=$(($memFree/1024))

    #memoria em Uso
    memUso=`free | grep  Mem  | awk '{print $3}'`
    memoriaUsada=$(($memUso/1024))

    #memoria em buff/cache
    memBC=`free | grep  Mem  | awk '{print $6}'`
    memoriaBC=$(($memBC/1024))

    #porcentagem de memoria usada
    porcentagem=$((($memUso*100)/$mem))

    echo $memoriaTotal $memoriaUsada $memoriaFree $memoriaBC $porcentagem

}
```

Figura 5. Script em Bash para monitoramento de Memória.

```
cpuUsada(){

    cpusTotal=`grep -c processor /proc/cpuinfo`
    #porcentagem de uso da cpu pora aplicações, kernel e tempo ociosa
    porcentoUSR=`mpstat | grep all | awk '{print $3, $5, $12}'`
    #echo $porcentoUSR >> monitoramento.txt
    echo $porcentoUSR $cpusTotal

}
```

Figura 6. Script em Bash para monitoramento de CPU.

“mpstat” com o objetivo de obter as estatísticas da cpu, juntamente com ele também foi utilizado o “awk” para manipular o texto obtido pelo comando “print”, este sendo manipulado para pegar apenas 3 parâmetros, cpu usada, cpu usada pelo sistema e cpu livre.

```
#recolher dados caso seja um ssd
diskSSD(){

    diskTotal=`df | grep /dev/nvme0n1p6 | awk '{print $2}'`
    total=$(((diskTotal/1024)/1024)) #converte os dados para GB

    diskLivre=`df | grep /dev/nvme0n1p6 | awk '{print $4}'`
    totalLivre=$(((diskLivre/1024)/1024)) #converte os dados para GB

    diskUsado=`df | grep /dev/nvme0n1p6 | awk '{print $3}'`
    totalUsado=$(((diskUsado/1024)/1024)) #converte os dados para GB

    echo $total $totalUsado $totalLivre
}

#recolher dados caso seja HD
diskHD(){

    diskTotal=`df | grep /dev/sda | awk '{print $2}'`
    total=$(((diskTotal/1024)/1024))

    diskLivre=`df | grep /dev/sda| awk '{print $4}'`
    totalLivre=$(((diskLivre/1024)/1024))

    diskUsado=`df | grep /dev/sda | awk '{print $3}'`
    totalUsado=$(((diskUsado/1024)/1024))

    echo $total $totalUsado $totalLivre
}
```

Figura 7. Script em Bash para monitoramento de Disco.

No script da Figura 7, podemos ver o script de monitoramento de Disco. Usa-se o comando “df” para pegar dados de espaço do disco, em seguida usamos o “grep” para pesquisa por um padrão, no caso, o padrão é o caminho para o disco, então selecionamos os dados usando desejados com o “awk” e “print”.

No script da Figura 8, podemos ver o script de monitoramento de Rede. Para coletar os dados de rede, acessamos por meio do comando “cat/proc/net/dev” as informações. Este diretório fornece uma visão abrangente

```
rede(){
    num=0 #variável para usar como contador para o laço while
    redeR=0 #variável para requisições recebidas em bytes
    redeT=0 #variável para requisições transmitidas em bytes

    while [ $num -le 1 ]
    do
        if [ $redeR -eq 0 -a $redeT -eq 0 ]; then

            redeR=`cat /proc/net/dev | grep wlo1 | awk '{print $2}'`
            redeT=`cat /proc/net/dev | grep wlo1 | awk '{print $10}'`

        else
            rede1=`cat /proc/net/dev | grep wlo1 | awk '{print $2}'`
            rede2=`cat /proc/net/dev | grep wlo1 | awk '{print $10}'`

            receive=$(($rede1 - $redeR))
            transmit=$(($rede2 - $redeT))
            echo $transmit $receive

            redeR=$rede1
            redeT=$rede2

        fi
        sleep $1
        num=$(($num + 1))

    done
}
```

Figura 8. Script em Bash para monitoramento de Rede.

de vários parâmetros e estatísticas de rede. Cada diretório e arquivo virtual dentro deste, descreve aspectos da configuração de rede do sistema. Então selecionamos os dados desejados usando “grep”, “awk” e “print”. Como os dados obtidos do arquivo são cumulativos, foi necessário acessar o arquivo duas vezes a primeira para obter os dados acumulados, é uma segunda para pegar a mudança que ocorreu nesses dados. De início temos as variáveis “redeR” e “redeT”, estas iniciadas com zero, elas vão armazenar as requisições recebidas e transmitidas que estão acumuladas no arquivo, usando de um “if” verificamos se as variáveis citadas tem valores 0, caso tenha estamos na primeira execução do algoritmo, caso não então entramos no “else”, onde pegamos dados é armazenamos nas variáveis “rede1” e “rede2” que iram armazenar os dados recebidos e transmitidos durante um certo intervalo, agora para obter a informação de mudança, fazemos uma subtração dos dados da última execução do “while”, subtraindo os dados da primeira, então temos os dados recebidos e transmitidos por segundo, então retornamos estes para serem avaliados.

E. Módulo Java

Assim como os dois módulos anteriores, o módulo desenvolvido na linguagem Java também faz o monitoramento de recursos do sistema. Nos nossos scripts, utilizamos uma interface de gerenciamento de sistema que faz parte do pacote java.lang.management, chamada OperatingSystemMXBean. Também fazemos uso da biblioteca Sigar, desenvolvida pela Hyperic Inc., para acessar informações adicionais do sistema operacional. A seguir, vemos um dos scripts utilizado no nosso sistema:

```
OperatingSystemMXBean osBean = ManagementFactory.getPlatformMXBean(OperatingSystemMXBean.class);

memTotal = osBean.getTotalPhysicalMemorySize()/1024.0/1024.0/1024.0; //Em GB
memFree = osBean.getFreePhysicalMemorySize()/1024.0/1024.0/1024.0;
memUsed = (memTotal - memFree);
memPercent = ((double) memUsed / memTotal) * 100.0;
```

Figura 9. Script em Java para monitoramento de Memória.

No script da Figura 9, podemos ver o script de monitoramento de Memória. Neste, utilizamos as seguintes funções da interface OperatingSystemMXBean: getTotalPhysicalMemorySize( ) e getFreePhysicalMemorySize( ) para obter, respectivamente, as memórias total e livre no sistema.

```
sigar = new Sigar();
cpuInfos = sigar.getCpuInfoList();
qntCpus = cpuInfos.length;

cpuPerc = sigar.getCpuPerc().getCombined() * 100.0;

cpu = sigar.getCpu();
cpuTimesUser = cpu.getUser();
cpuTimesSystem = cpu.getSys();
cpuIdle = cpu.getIdle();
```

Figura 10. Script em Java para monitoramento de CPU.

Já no script da Figura 10, podemos ver o script de monitoramento de CPU. Aqui, fazemos uso da biblioteca Sigar para obter as informações necessárias. O método getCpuInfoList() é usado para obter a quantidade de CPUs no sistema. Já o método getCpuPerc() colhe o percentual utilizado de CPU e o método getCpu() é necessário para obter o tempo gasto em processos no modo usuário, no sistema e o tempo ocioso.

```
for (FileStore store : FileSystems.getDefault().getFileStores()) {

    diskTotal = store.getTotalSpace() /1024.0/1024.0/1024.0;
    diskFree = store.getUnallocatedSpace()/1024.0/1024.0/1024.0;
    diskUsed = diskTotal - diskFree;
    diskPercent = ((double) diskUsed / diskTotal) * 100.0;


}
```

Figura 11. Script em Java para monitoramento de Disco.

No script da Figura 11, podemos ver o script de monitoramento de Disco. Neste script, fazemos uso da classe FileStore. Ela fornece métodos para obter informações sobre o espaço livre e utilizado no dispositivo, bem como a capacidade total do dispositivo.

```
Sigar sigar = new Sigar();
NetInterfaceStat stat = sigar.getNetInterfaceStat(interfacePrincipal);
bytesRecv = stat.getRxBytes();
packetsRecv = stat.getRxPackets();
errorsRecv = stat.getRxErrors();
bytesSent = stat.getTxBytes();
packetsSent = stat.getTxPackets();
errorsSent = stat.getTxErrors();
```

Figura 12. Script em Java para monitoramento de Rede.

No script da Figura 12, podemos ver o script de monitoramento de Rede. Aqui fazemos uso da classe NetInterfaceStat, que faz parte do pacote Sigar. Usamos essa classe para obter estatísticas de interface de rede, como bytes enviados e recebidos, pacotes enviados e recebidos, erros ao enviar e ao receber.

F. Demonstração do Sistema

Na figura 13, podemos ver a interface do software de monitoramento, onde é possível fazer a seleção dos parâmetros para geração do script de monitoramento.

Na figura 14, podemos ver os recursos de CPU e RAM sendo selecionado, assim como foi escolhido o Python como linguagem do script, e sem tempo de parada para essa amostra.

Na figura 15, podemos ver o resultado final da geração do script, onde, o usuário pode copiar e utilizar para o monitoramento.

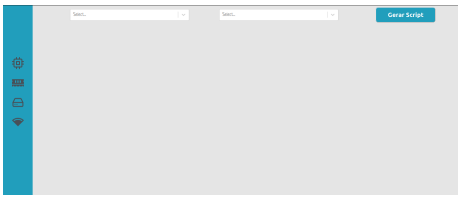

Figura 13. Interface Software de monitoramento.

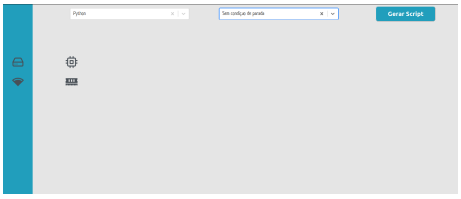

Figura 14. Interface Software de monitoramento.

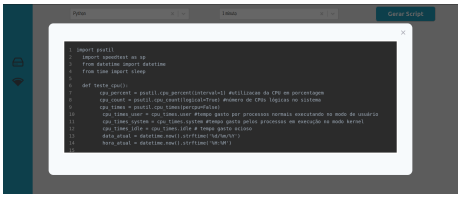

Figura 15. Interface Software de monitoramento.

V. Experimento

O experimento realizado durante os testes consistiu na utilização de uma aplicação que pudesse gerar uma sobrecarga no sistema e com isso, poderia ser feito uma coleta de dados através dos scripts de monitoramento simulando uma situação de uso da máquina.

A aplicação utilizada para se fazer os testes de sobrecarga foi a Stress-ng. É uma ferramenta para carregar, testar o estresse e comparar um sistema de computador. Ela pode sobrecarregar vários subsistemas de um computador como: CPU, cache, disco, memória, soquete e I/O de pipe, agendamento e muito mais. Foi projetado para exercitar vários subsistemas físicos de um computador, bem como as várias interfaces do kernel do sistema operacional.

Possui uma instalação bem simples, com poucas linhas de comando, além de uma fácil utilização.

Para o experimento em questão, foram feitos testes utilizando sobrecarga na memória e a CPU do computador.

Testes utilizados:
CPU - stress-ng –cpu 4. Este teste utilizava os núcleos do computador para realizar o teste.
Memória - stress-ng –vm 1 –vm-bytes 1G. Utilizava um estressor de memória virtual utilizando 1G dessa memória virtual.

VI. Análise dos Resultados

A. Metodologia da Análise

Utilizamos as bibliotecas Pandas, MatPlot e NumPy da linguagem Python para coletar os dados fornecidos dos experimentos, analisá-los e gerar gráficos. Demonstraremos dois cenários:

Pré Stress: rodada de scripts antes da utilização do stress na máquina.
Pós Stress: rodada durante e pós a utilição do stress.

Os dados são subdividos em: Porcentagem de CPU e Memória coletadas da máquina pré e pós stress; Tempo de execução, porcentagem de CPU e Memória consumidos pelo processo de cada script de cada linguagem.

Consideramos um cenário próximo para todos os testes, não utilizando outros recursos para que não tenha interferência nos testes e resultados. Foram 100 iterações durante 300 segundos, ou seja, 1 iteração a cada 3 segundos. Para a rodada de stress, utilizamos o mesmo tempo e o mesmo número de iterações, sendo adicionado 3 picos de stress.

B. Python

Durante a rodada do script de python foi registrado um máximo de aproximadamente 11% de consumo de CPU e mínimo de 1% (figura 16). Já o consumo de memória atingiu aproximadamente 31,5% e mínimo de 29,75% durante o mesmo tempo (Figura 17).

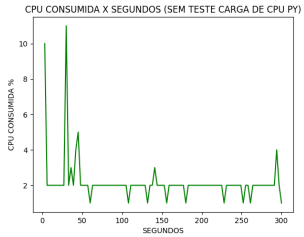

Figura 16. Avaliação de consumo de CPU Python pré Stress

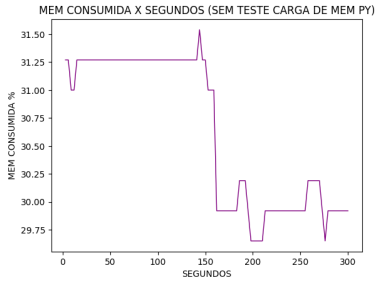

Figura 17. Avaliação de consumo de Memória Python pré Stress

Os dados coletados durante e após a rodada de sobrecarga na máquin mpstram que a CPU teve um aumento de 90% em seu consumo, chegando a um pico de 100% de uso em sobrecarga (Figura 18). Já para a memória houve um máximo de 76% de consumo detectado, um aumento de aproximadamente 45% (Figura 19).

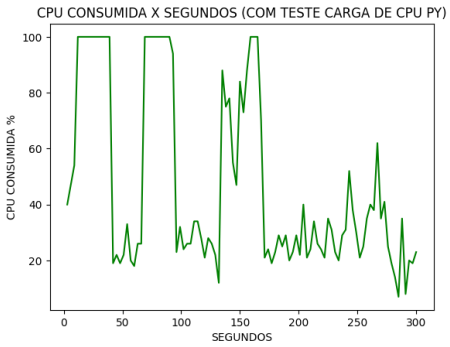

Figura 18. Avaliação de consumo de CPU Python pós Stress

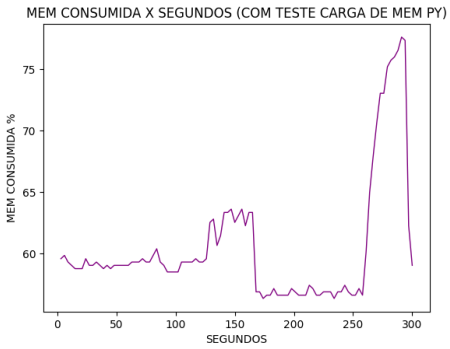

Figura 19. Avaliação de consumo de Memória Python pós Stress

Ao analisarmos os dados de consumo do processo do script de python observamos que o consumo de memória permaneceu estável, o consumo de CPU teve um aumento de 0.03% e um gasto de 4 segundos a mais para processar durante a sobrecarga no sistema.

Tabela V
COMPARATIVO DE CONSUMO DO PROCESSO PYTHON

CARGA	%CPU	%MEM	TEMPO
ANTES	0.18	0.3	00:04:52
DEPOIS	0.21	0.3	00:04:56

C. Bash

O script bash detectou um consumo constante de 7% de CPU durante os 300 segundos de análise, já para memória houve variações de consumo entre 43.1% a 43.8% .

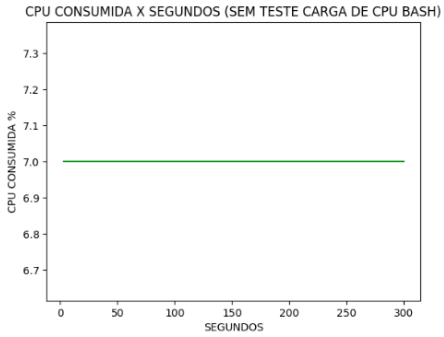

Figura 20. Avaliação de consumo de CPU Bash pré Stress

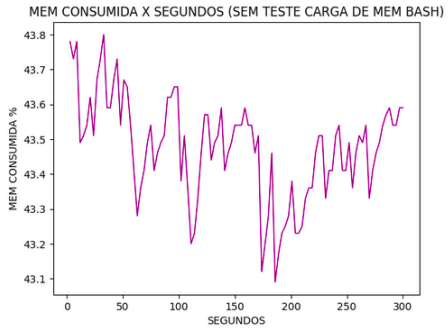

Figura 21. Avaliação de consumo de MEM Bash pré Stress

Aos a sobrecarga, o script de bash detectou um consumo constante de 8% de CPU, apenas 1% a mais que o cenário anterior, que não possuia nenhuma sobrecarga. Para memória, percebemos uma constante de 45% a 50% durante os primeiros 250 segundos e ao final, picos de 70% de consumo.

Ao analisarmos os dados de consumo do processo do script de python observamos que o consumo de memória permaneceu estável, o consumo de CPU teve um aumento de 0.03% e um gasto de 4 segundos a mais para processar durante a sobrecarga no sistema.

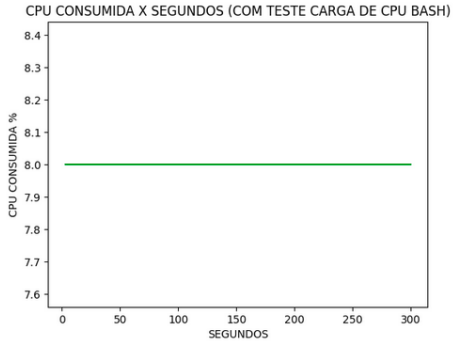

Figura 22. Avaliação de consumo de CPU Bash pós Stress

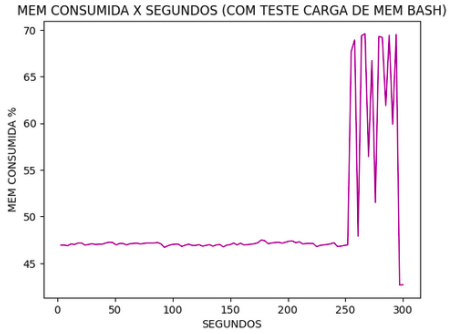

Figura 23. Avaliação de consumo de Memória Bash pós Stress

D. Java

E. Comparativos

VII. IMPLICAÇÕES

Alguns problemas e dificuldades enfrentados surgiram, como: Para linguagem Node.js parece não existir métodos ou bibliotecas que avalie dados de redes e disco. Diante disso, tomamos a decisão de substituir o Node.js pela linguagem Java, que oferece suporte para os recursos que nos propomos a monitorar.

VIII. CONSIDERAÇÕES FINAIS

REFERÊNCIAS

[1] Sottile, M. J. and Minnich, R. G. (2002). Supermon: A high-speed cluster monitoring sys-tem. In Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on, pages 39–46. IEEE.

[2] Araujo, J. Melo, C. Cordeiro, C. UNAME Tool: Automatic Generation of Computer Resources Monitoring Scripts. 2021 16th Iberian Conference on Information Systems and Technologies (CISTI). Junho, 2021.

[3] Morais, L. Angelo, M. et al. Sistema leve de monitoramento de servidores Linux por meio de scripts. Retc Revista de tecnologias. SP. janeiro, 2018.

Tabela VI
COMPARATIVO DE CONSUMO DO PROCESSO BASH

CARGA	%CPU	%MEM	TEMPO
ANTES	0.18	0.3	00:04:52
DEPOIS	0.21	0.3	00:04:56

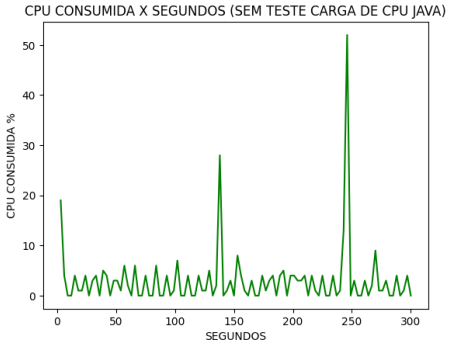

Figura 24. Avaliação de consumo de CPU Java pré Stress

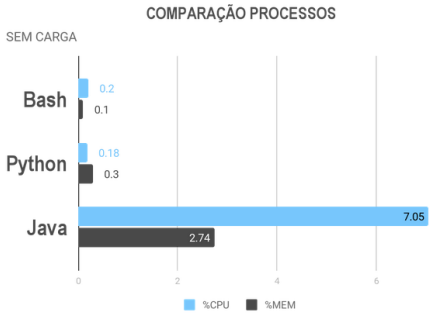

Figura 25. Comparativo de consumo CPU/MEM pré stress

[4] Durieux, A. Monitoramento de servidores com scripts. Universidade Federal de Santa Catarina, Departamento de informática e estatística.. Florianópolis - SC. Fevereiro, 2012.
[5] Andresen, R. (2004). Monitoring linux with native tools. In Int. CMG Conference, pages 345–354.
[6] Eranian, S. (2006). Perfmon2: a flexible performance monitoring interface for linux. In Proc. of the 2006 Ottawa Linux Symposium, pages 269–288. Citeseer.
[7] Tongen, Christopher ; Stinson, Michael. Improving the general monitoring system. Proceedings of the 2010 ACM conference on Information technology education, 2010.
[8] Simmy, T. Sgar, K. Node.js OS. 2021. Disponível em ¡https://www.geeksforgeeks.org/node-js-os/?ref=lbp¿
[9] psutil 5.9.4 pypi.org. 2022. Disponível em ¡https://pypi.org/project/psutil

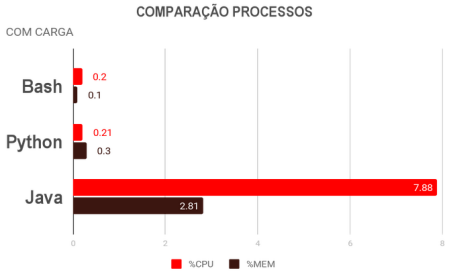

Figura 26. Comparativo de consumo CPU/MEM pós stress