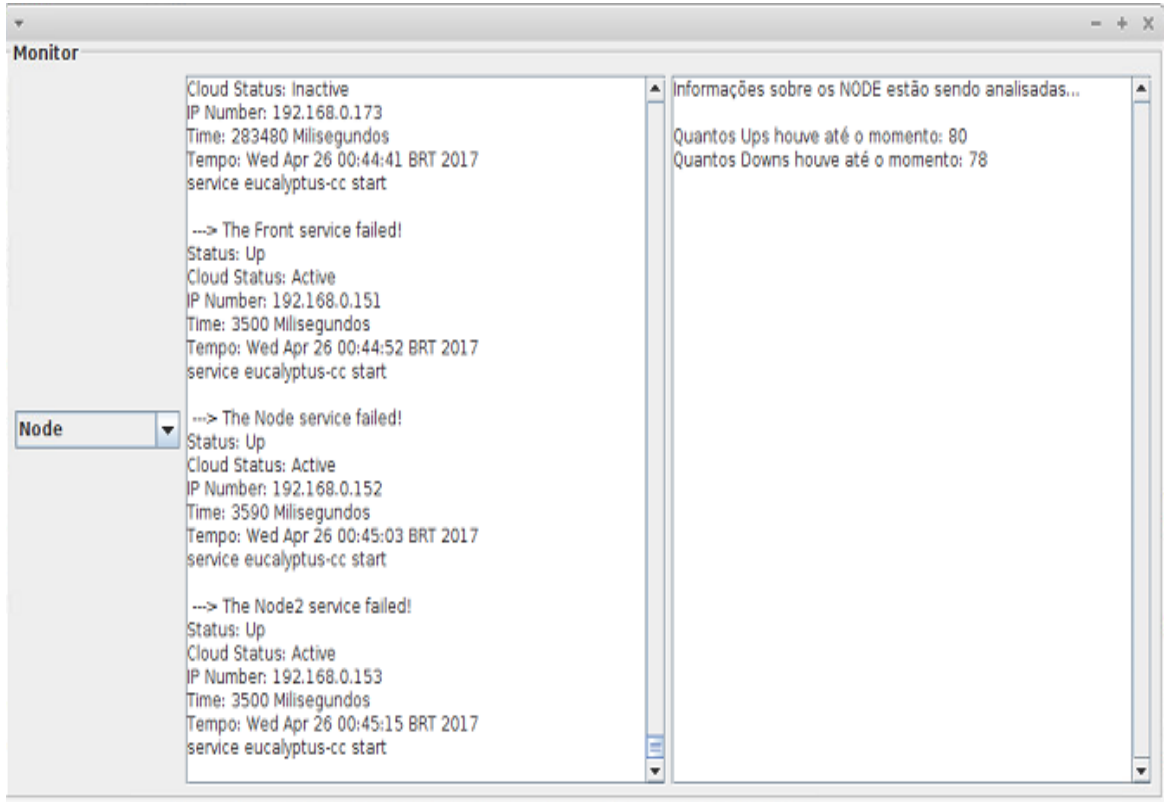
# SIMF: Framework de Injeção de Falhas e Monitoramento para Cloud utilizando SPN

Eliardo Claúdio - IC
Prof. Paulo Maciel - prmm@cin.ufpe.br

Universidade Federal de Pernambuco
Centro de Informática U·F·P·E

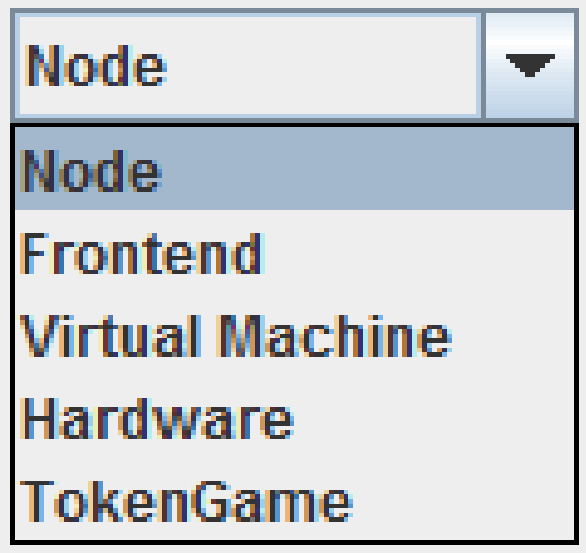
# TELA INICIAL

Como o nome já diz, o papel do monitor é mostrar em tempo de execução qualquer alteração que seja realizada dentro do arquivo de log que é gerado pelo SIMF.

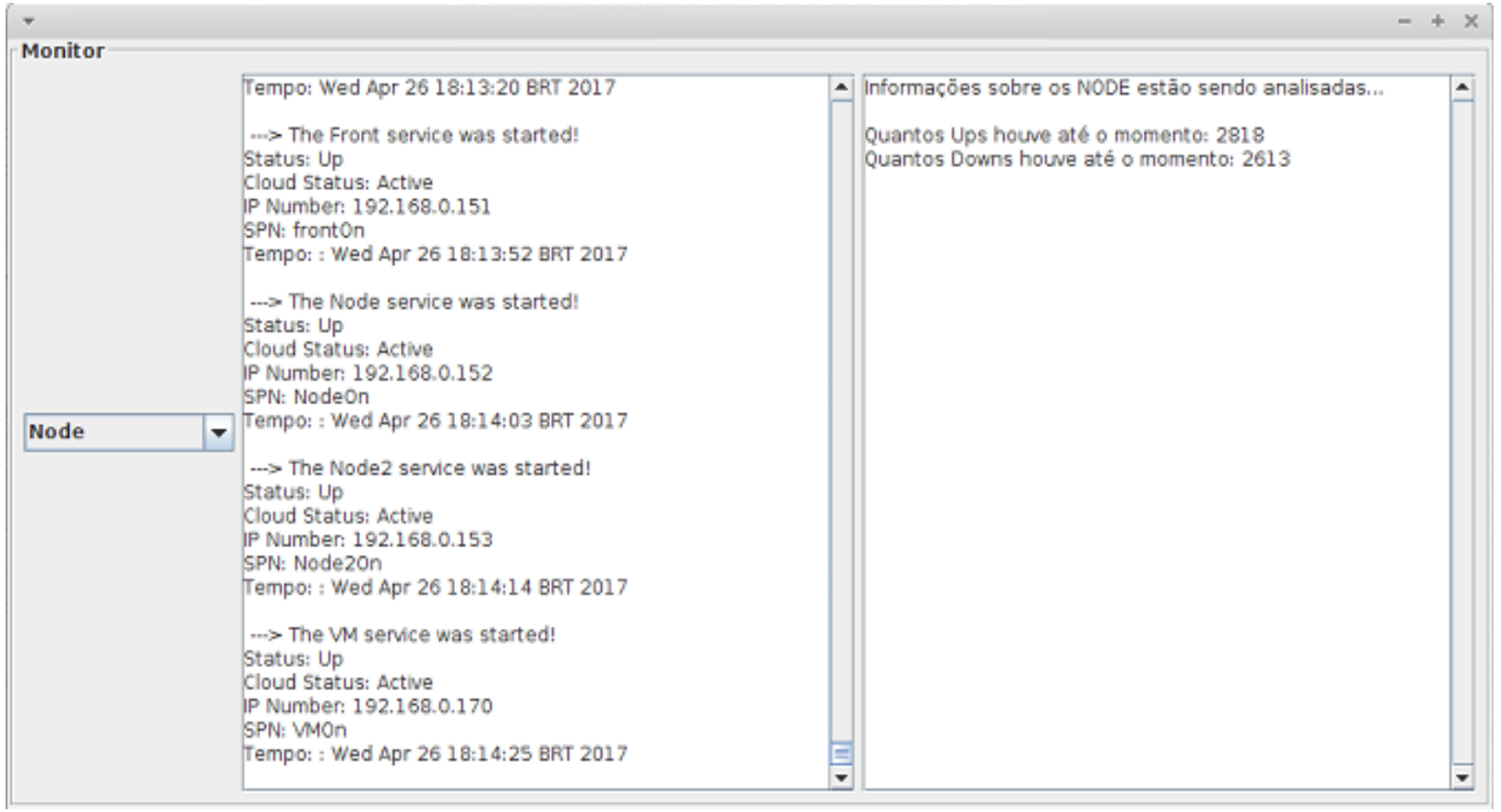


# FUNCIONALIDADES

Ele funciona da seguinte maneira, após o usuário iniciar o processo de Injeção de falhas através do SIMF, o usuário contará com uma aba que possui todas as opções de injeção que podem ser realizados pelo SIMF.



# MONITORAMENTO DUPLO



# CÓDIGO

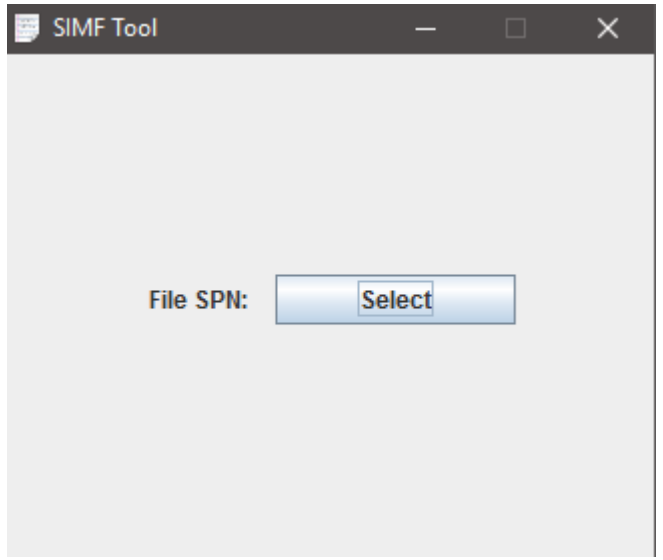
Dentro da classe código, é utilizado um switch para pegar a opção que o usuário deseja monitorar, a partir disso o programa inicia o processo de monitoramento em tempo real.

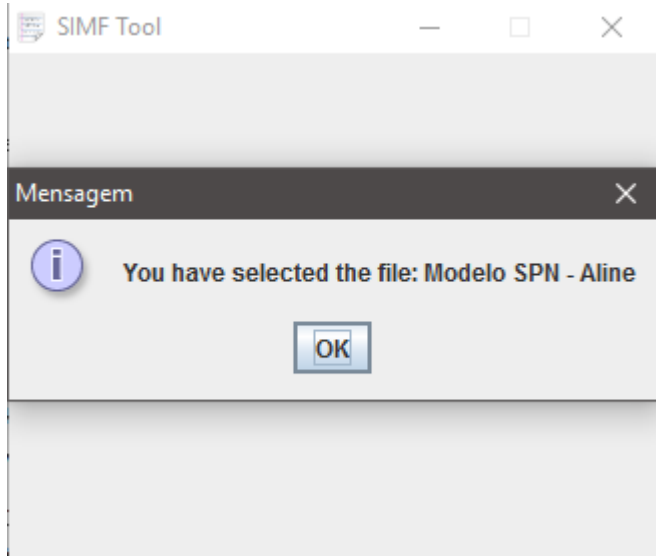
```
public void codigo() {
    try {
         String opcao = String.valueOf(JComboBox1.getSelectedItem());
         opcao = opcao.toUpperCase();

         switch(opcao)
         {
            case "NODE":
                display.setText("");
                BufferedReader leitor = new BufferedReader(new FileReader ("Node.txt"));
                max = leitor.readLine();
                display.append("O teste "+opcao+" está sendo Monitorado...");
                display.append("\n"+"\n");
                while (max != null){
                    display.append(max);
                    max = leitor.readLine();
                    display.append("\n");
                    display.setCaretPosition(display.getText().length());
             }
                leitor.close();
                    break;

            case "FRONTEND":
                display.setText("");
                BufferedReader leitor2 = new BufferedReader(new FileReader ("Frontend.txt"));
                max = leitor2.readLine();
                display.append("O teste "+opcao+" está sendo Monitorado...");
                display.append("\n"+"\n");
                while (max != null){
                    display.append(max);
```

# Porque o Monitor é importante

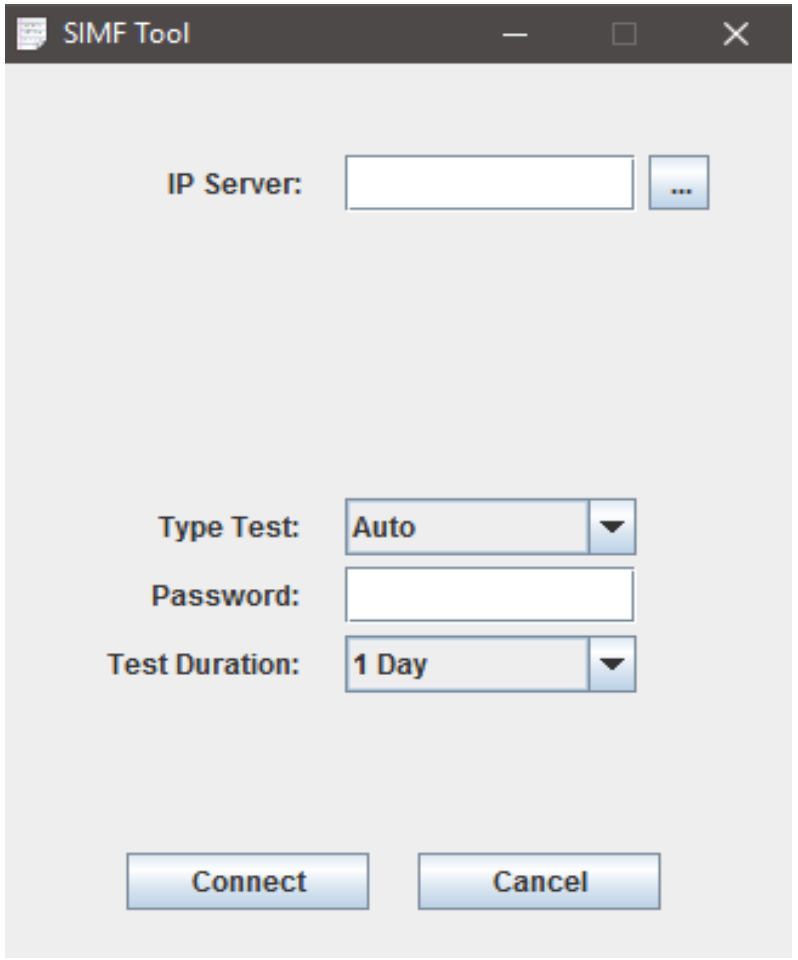
# SIMF

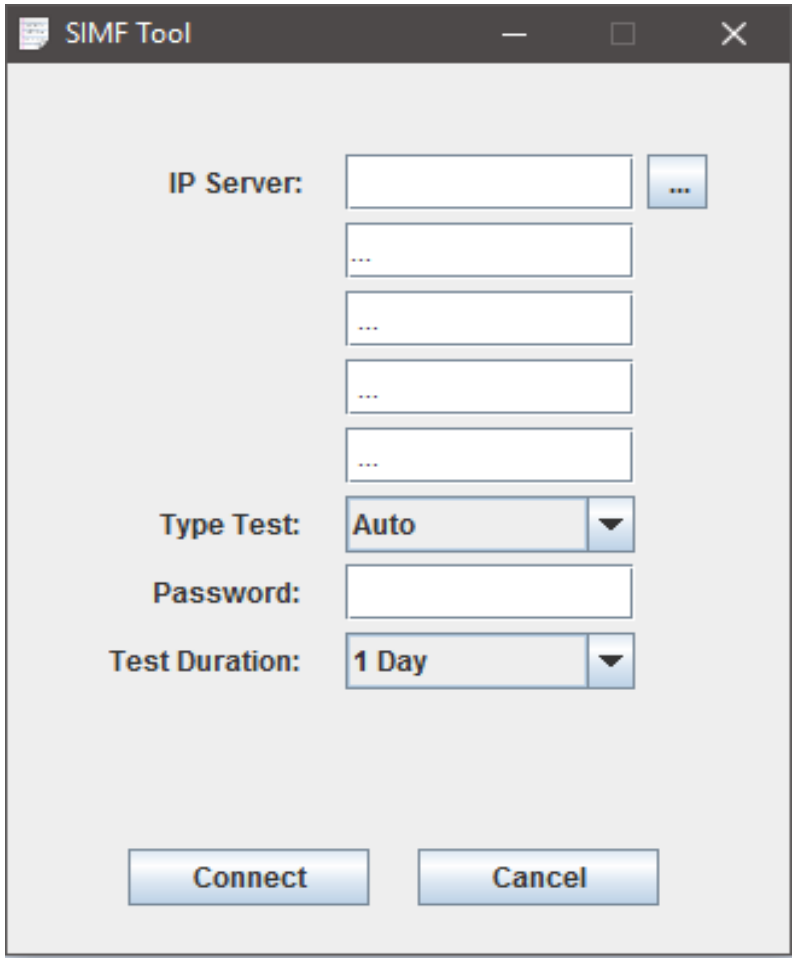
O diferencial do SIMF está na sua integração com o Script SPN gerado pelo Mercury. Ao clicar em File SPN o usuário irar selecionar o arquivo SPN.mry que é gerado pelo Mercury, feito isso o programa irá fazer uma leitura e tradução do script e só após isso liberar a tela para iniciar o processo de injeção de falhas.



# SIMF


SIMF Tool
IP Server:
...
Type Test:
Auto
Password:
Test Duration:
1 Day
Connect
Cancel


SIMF Tool
IP Server:
...
...
...
...
...
Type Test:
Auto
Password:
Test Duration:
1 Day
Connect
Cancel

# TRADUÇÃO DO SCRIPT SPN

**Utilizando a classe tokengametest**

# O QUE A CLASSE FAZ

# APÓS A TRADUÇÃO DO SCRIPT

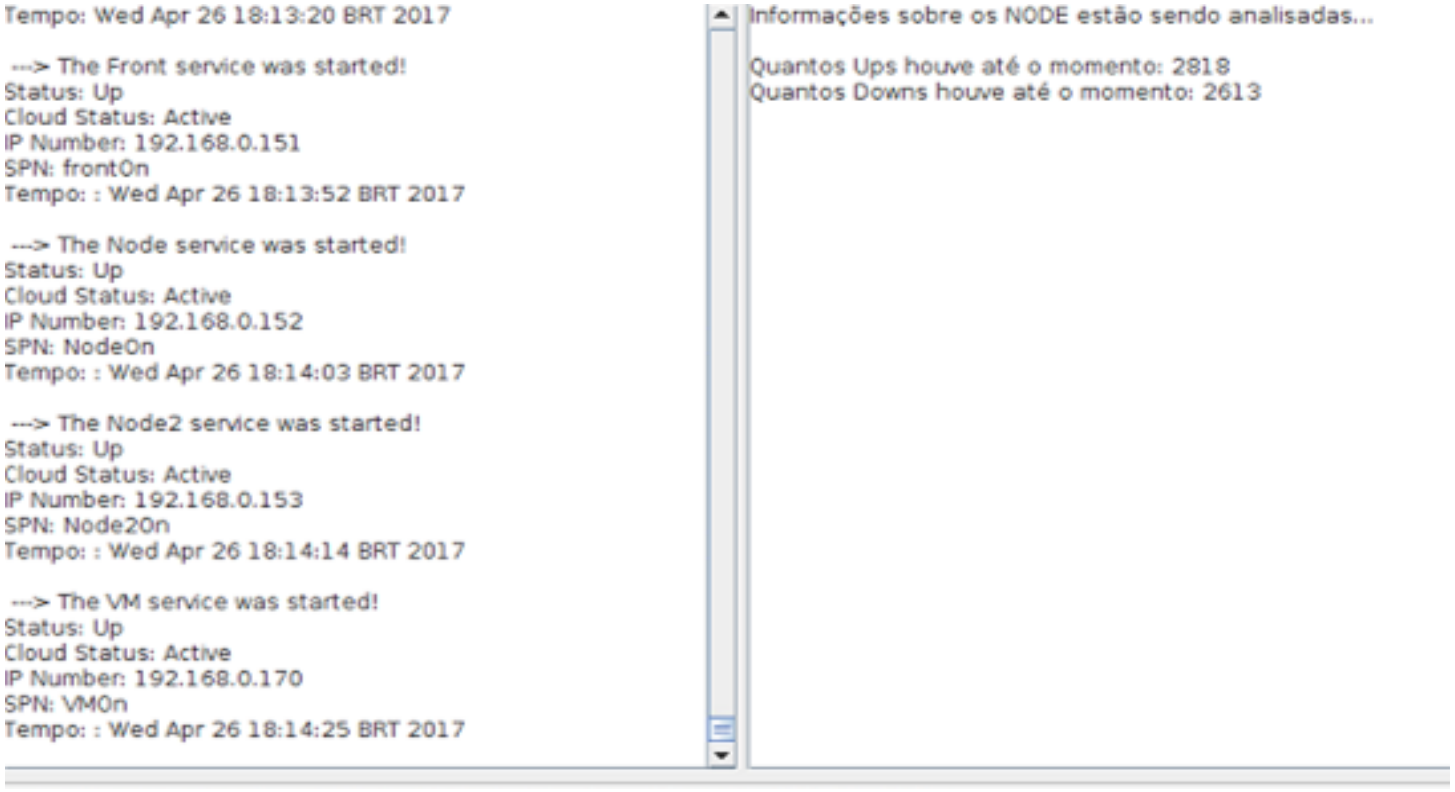
```
 ---> Test started on: Fri Apr 07 00:36:56 GFT 2017
{nodeOn=0, vmOn=0, nodeOn2=0, vm2Off=2, vmOff=2, frontOff=1, nodeOff=1, nodeOff2=1, vm2On=0, frontOn=0}
Transition: repairNode Fired!
Transition: repairNode2 Fired!
Transition: repairVm2 Fired!
Transition: repairVm Fired!
Transition: failVm2 Fired!
Transition: failVm Fired!
Transition: failNode2 Fired!
Transition: repairNode2 Fired!
Transition: repairVm2 Fired!
Transition: failNode2 Fired!
Transition: TI2 Fired!
Reached Marking!
Transition: repairFront Fired!
Reached Marking!
Transition: repairNode2 Fired!
```

# CÓDIGO

```
public void actionPerformed(ActionEvent e){
     JFileChooser fc = new JFileChooser();
     int res = fc.showOpenDialog(null);

     if(res == JFileChooser.APPROVE_OPTION){
         File arquivo = fc.getSelectedFile();
         String arq = arquivo.toString();
         JOptionPane.showMessageDialog(null, "Você selecionou o arquivo: " + arquivo.getName());
         caminhoarq.setText(arq); //// Setando o valor da Label automaticamente, esse valor será o valor escolhido

         try{
             int cont = 1;

             BufferedReader leitor = new BufferedReader(new FileReader ("TokenGame.txt"));   //Lendo Arquivo Trans
             max = leitor.readLine();
             while (max != null){
                 cont +=1;
                 max = leitor.readLine();
             }
             ArrayList<String> lista = new ArrayList<String>(cont);
             leitor.close();

             BufferedReader leitor2 = new BufferedReader(new FileReader ("TokenGame.txt"));   //Lendo Arquivo Tran
             max = leitor2.readLine();
             lista.add(max);
             while (max != null){
                 max = leitor2.readLine();
                 lista.add(max);
            }
             System.out.println(lista);
```

# UNIVERSIFICAÇÃO

# CARACTERÍSTICAS DO SIMF



# COMO É FEITO O PROCESSO DE MONITORAMENTO DO SIMF – EX:VIRTUAL MACHINE (VM)

```
public void runCLCStateMachine() throws InterruptedException, IOException {
switch (this.getState()) {
case RUNNING:
    if (this.isAlive()) {
        WriteFile.logger("\n ---> The service was started!" , "CLCController_log.txt");
        WriteFile.logger("Started. CLC: " + this.getSshConnection().getHost(),"CLCController_log.txt");
        WriteFile.logger(new Date().toString(), "CLCController_log.txt");
        int waitingTime = this.generateRandomFailureTime();
        this.setTimer(new MyTimer(waitingTime));
        this.setState(StateMachineEnum.TIMER_INJECT_FAILURE);
        WriteFile.logger("Generated Failure Time: " + waitingTime, "CLCController_log.txt");
        WriteFile.logger(new Date().toString(), "CLCController_log.txt");
        } else {
        // Sleep again until the CLC starts
        }
        break;

case TIMER_INJECT_FAILURE:
    if (this.getTimer().isExpired()) {
        this.stopCLC();
        WriteFile.logger("\n ---> The service stopped!" , "CLCController_log.txt");
        WriteFile.logger("Failed. CLC: " + this.getSshConnection().getHost(), "CLCController_log.txt");
        WriteFile.logger(new Date().toString(), "CLCController_log.txt");
        Thread.sleep(30000); // 30 seconds sleep for the CLC to actually stops
        this.setState(StateMachineEnum.FAILED);
        } else {
        // Sleep again until the timer expires
        }
        break;
```

# SIMF: Framework de Injeção de Falhas e Monitoramento para Cloud utilizando SPN

Aline S Oliveira Valente - aso2@cin.ufpe.br
Prof. Paulo Maciel - prmm@cin.ufpe.b

Universidade Federal de Pernambuco
Centro de Informática U·F·P·E

# AGENDA

- Contextualização;
- Motivação;
- Objetivo;
- SIMF *baseline*;
- Estudo de caso;
- Arquitetura;
- Contribuições;
- Próximos Passos.

# CONTEXTUALIZAÇÃO

Evolução de sistemas computacionais visam oferecer:

- Usabilidade;
- Funcionalidades diversificadas;
- Confiáveis;
- Baixo custo; e
- Alto desempenho.



# MOTIVAÇÃO

**Desenvolver uma ferramenta única, capaz de inserir falhas através de rede de Petri (SPN) em plataformas de *cloud*.**

- Necessidade do pesquisador em estudar a disponibilidade e confiabilidade em ambientes de computação em nuvem;
- Carência de ferramentas de avaliação de disponibilidade através de SPN;
- Diminuição do retrabalho por partes dos pesquisadores.

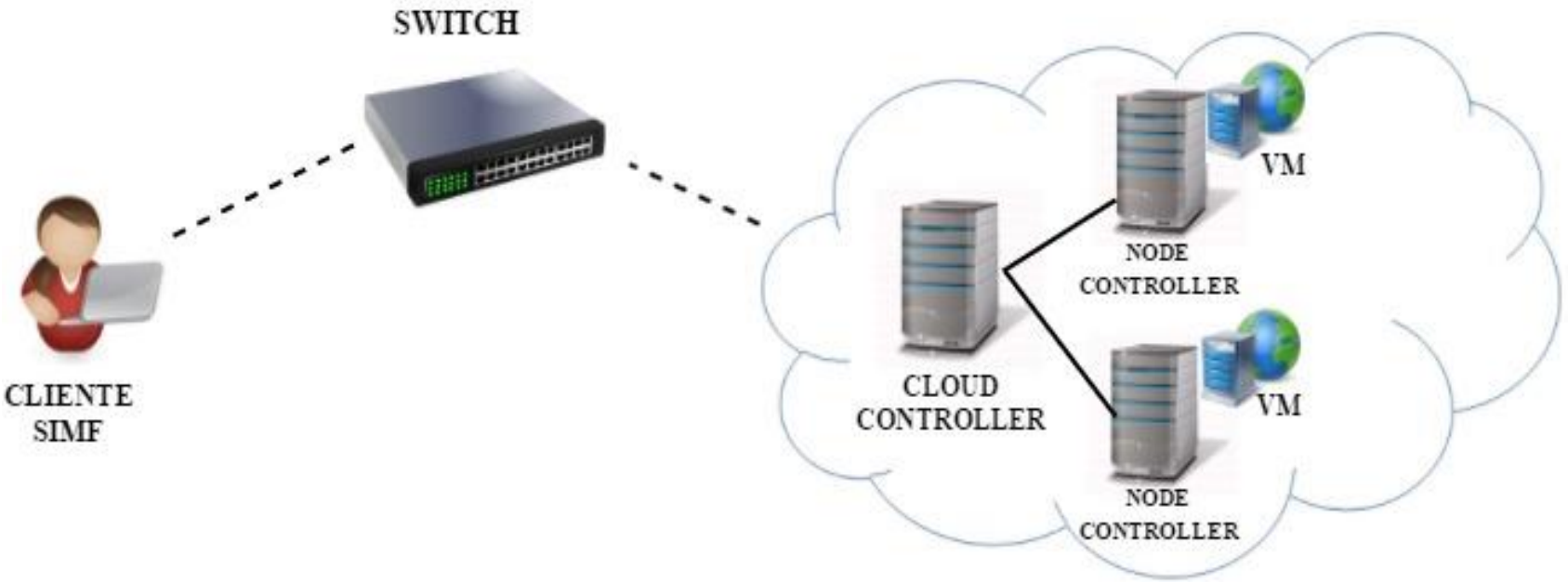
# OBJETIVO

- Propor um *framework* que ofereça suporte ao estudo de disponibilidade em ambientes de nuvem;
- Propor funcionalidades que ofereçam suporte à injeção de falhas e monitoramento de infraestrutura de *cloud* com base em modelos de Rede de Petri;
- Avaliar a disponibilidade de sistemas de computação em nuvem com base em experimentos feitos a partir de uma modelo SPN;
- Propor possíveis melhorias nos sistemas analisados com base nos resultados obtidos.

# SIMF

- Injetor de falhas em SPN, injeta falhas com opção de reparo, e simula a ausência de serviços em uma plataforma de computação de nuvem.
- Seu diferencial é utilizar a **SPN** como parâmetro para injeção de falhas.
- Possui um monitor que exibe ao usuário status do serviço
- O Monitor que informa (Up, Down, se os componentes estão funcionais e se a injeção de falha está ocorrendo.)

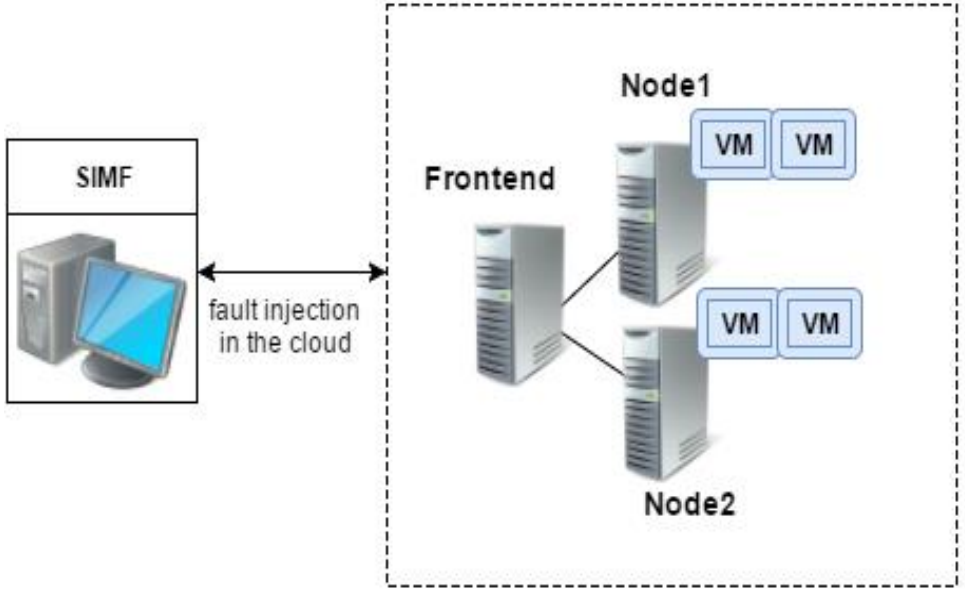
# SIMF *BASELINE*

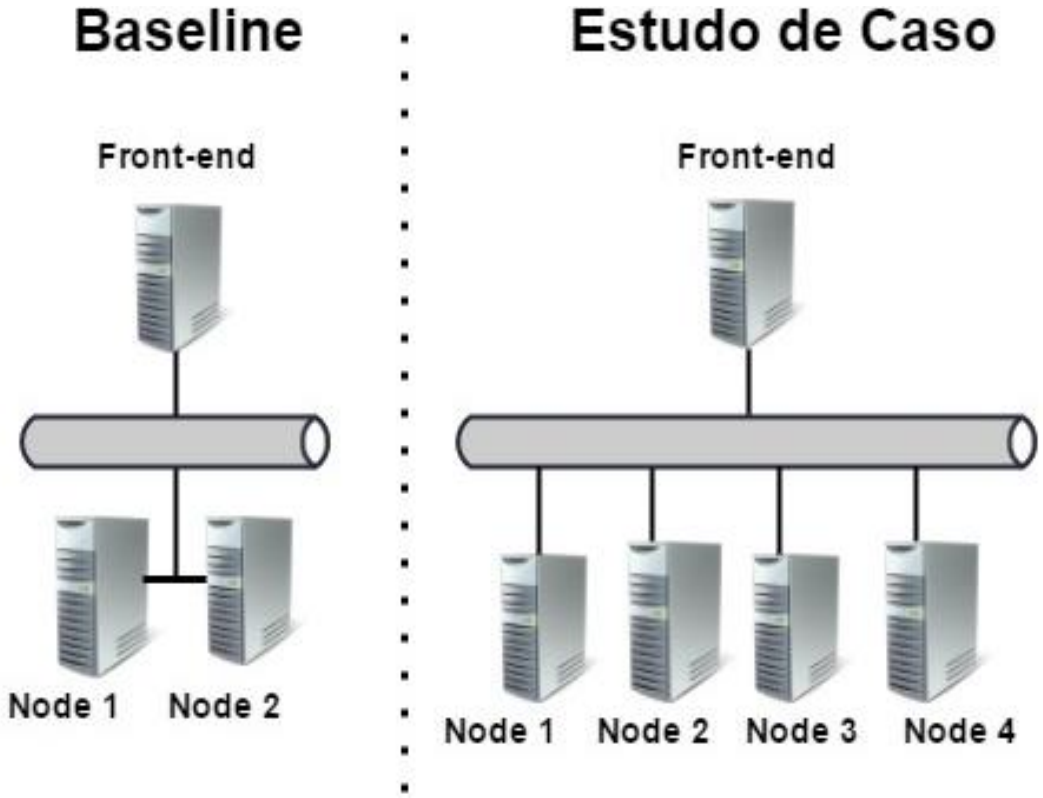



**Como usar o Framework SIMF:**

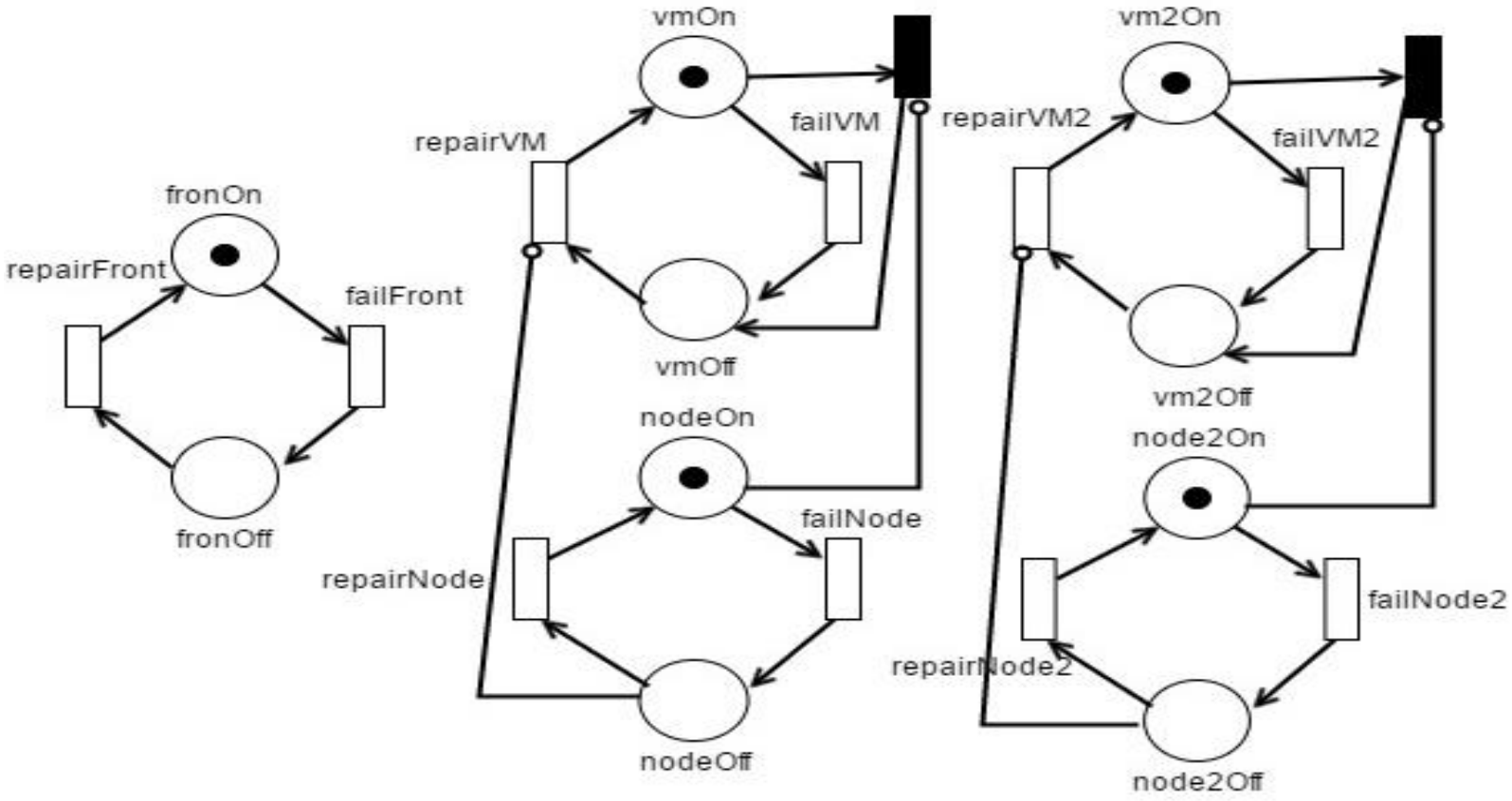
- Criar um modelo SPN no Mercury;
- Colocar o modelo dentro do injetor;
- Possuir infraestrutura de nuvem Eucalyptus.

# SIMF: Resultados previstos



# MODELO SPN *BASELINE*



# ESTUDO DE CASO I: VALIDAÇÃO DA ARQUITETURA *BASELINE*

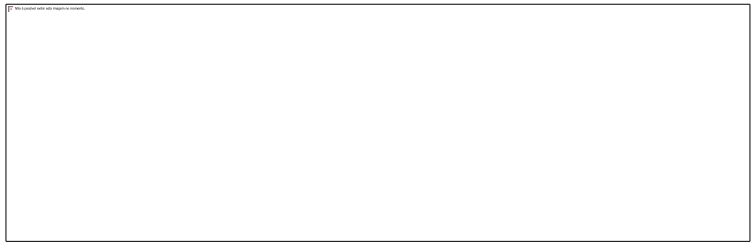
**Objetivo**

- Criar um modelo SPN de uma arquitetura básica (*baseline*);
- Analisar a SPN criada através do sistema SIMF com atividades de injeção de falhas a partir do modelo gerado anteriormente;
- Validar o modelo gerado com o experimento realizado pelo SIMF.

- Fator de redução: falha e reparo 1000
- Duração: 24 horas (1 dia)
- 3758 requisições enviadas e 1889 registradas como falhas
- A = uptime / (uptime + downtime) = 48,38% (Modelo SPN)

***Valor real que foi reduzido**

| COMPONENTES | MTTR | MTTF |
|---|---|---|
| CLC | 788,4 | 1 |
| CC | 788,4 | 1 |
| SC | 788,4 | 1 |
| WALRUS | 788,4 | 1 |
| NC | 788,4 | 1 |
| VM | 2880 | 1 |



# VALIDAÇÃO DO MODELO

Para o processo de validação do modelo foi utilizado o injetor SIMF no ambiente real, e posteriormente foi realizado a validação a partir de cálculos estatísticos proposto por Keesee;

Distribuição F o valor obtido foi 0,9380 mínimo e máximo de 1,066.

| **Intervalo de Confiança *A* e *p* - *Keesee*** | | |
|---|---|---|
| ***P*** | $P_L$ | 0,94410933 |
| | $P_U$ | 1,072943013 |
| ***A*** | $A_L$ | 0,514374364 |
| | ***Au*** | 0,482405929 |

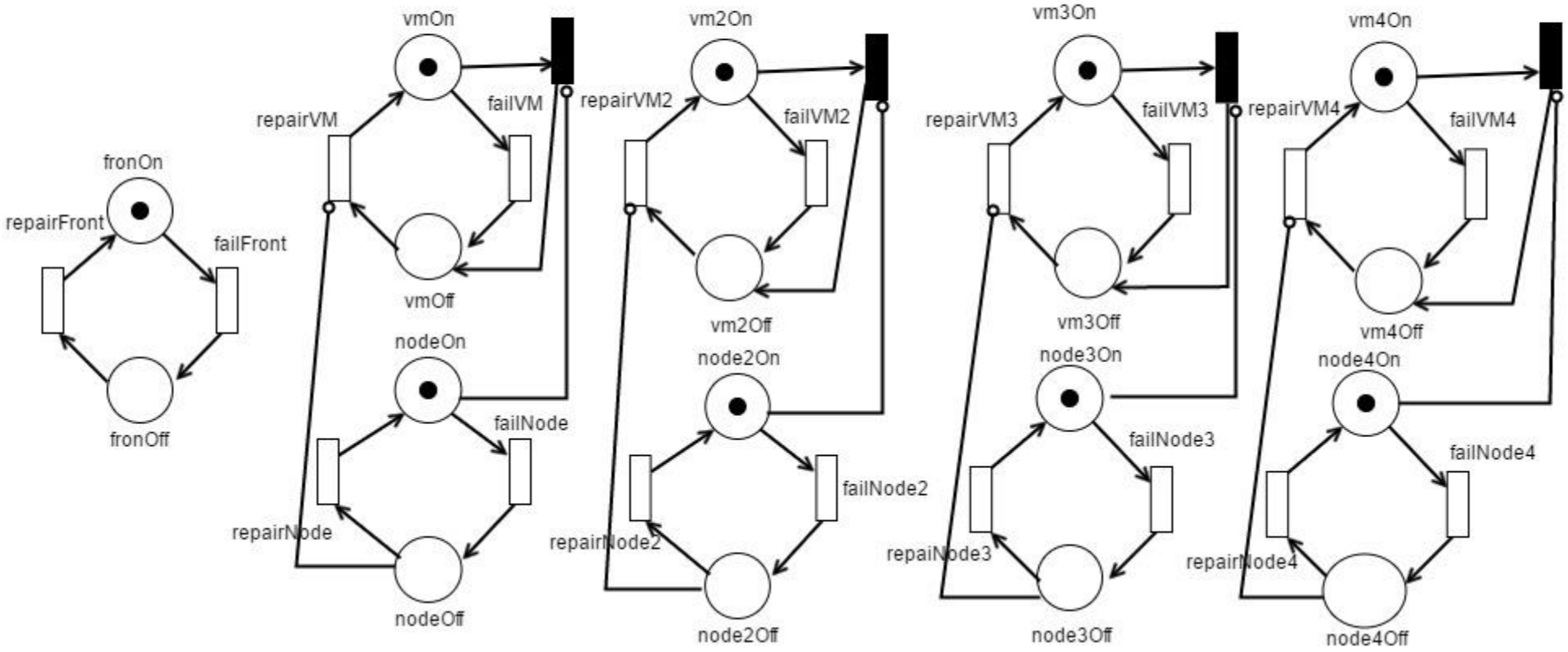
| | |
|---|---|
| **Disponibilidade do Injetor** | **50,16%** |
| **Disponibilidade do Modelo** | **48,38%** |

**O Injetor forneceu evidências que proporcionam inferir que o modelo da arquitetura base é de fato válido sob o aspecto da disponibilidade da infraestrutura analisada**

# SIMF: Estudo de Caso - Resultado

Assim, através do cálculo da disponibilidade do modelo analítico que representa o funcionamento da cloud, observamos que a disponibilidade alcançada do serviço foi de **0,4838**.
Esse valor que se encontra dentro do intervalo da disponibilidade, onde podemos afirmar a validação do modelo, visto que a disponibilidade do sistema real encontra-se dentro dos limites das disponibilidades identificadas no processo de validação.

# SIMF: Estudo de Caso



# SIMF-Próximos Passos

- Avaliar a disponibilidade da arquitetura proposta com o SIMF;
- Validar o modelo proposto;
- Propor melhorias para os sistemas analisados;
- Terminar a escrita;
- Publicação de artigo;