Optimization of private cloud infrastructures for task scheduling applications

Danilo Oliveira

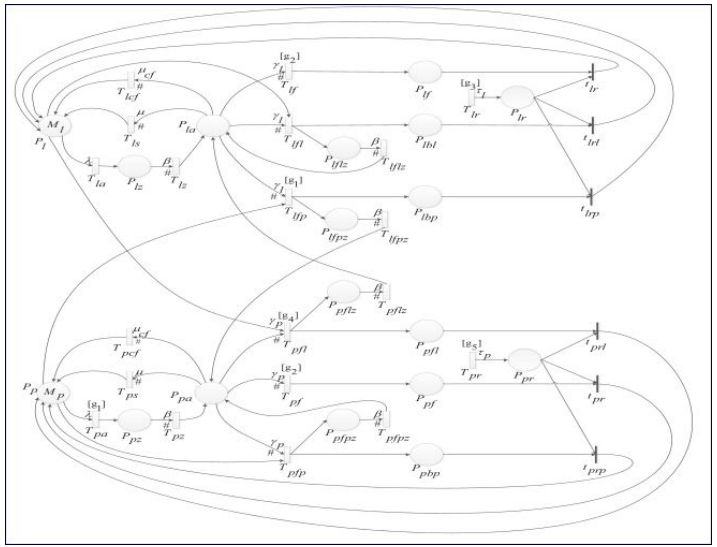
Orientador: Paulo Maciel
Co-orientador: Nelson Rosa

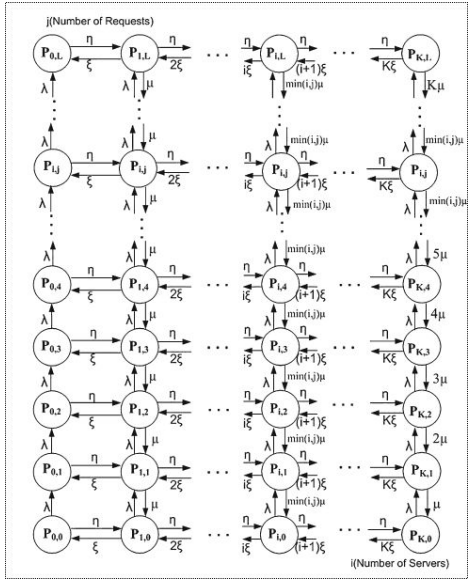
Roteiro

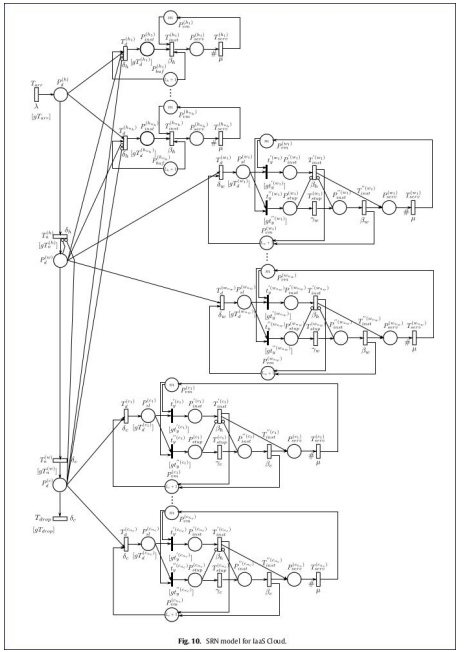
- Introdução
- Solução proposta
- Object Petri Nets
- Simulação e métodos heurísticos
- Conclusão

Introdução

Modelagem de performabilidade em nuvens computacionais por técnicas analíticas possui limitações em termos de expressividade e escalabilidade







Problema

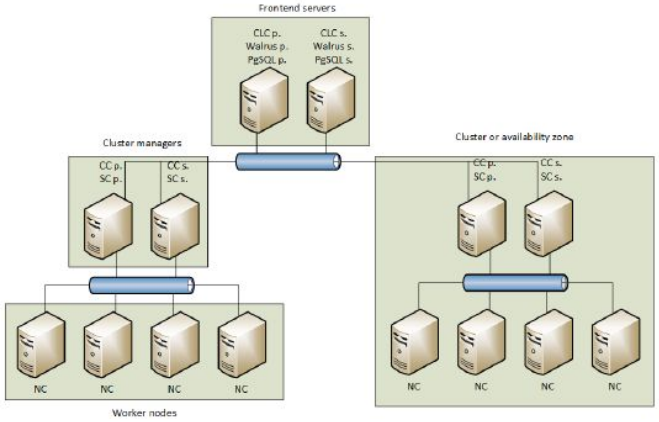
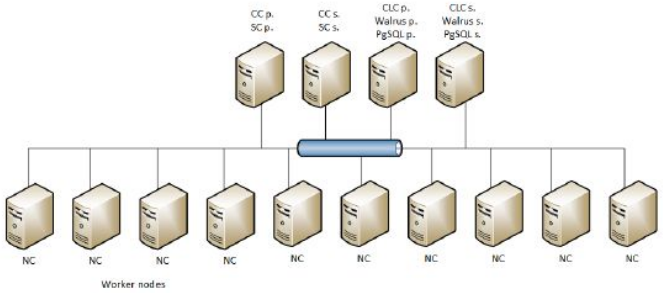
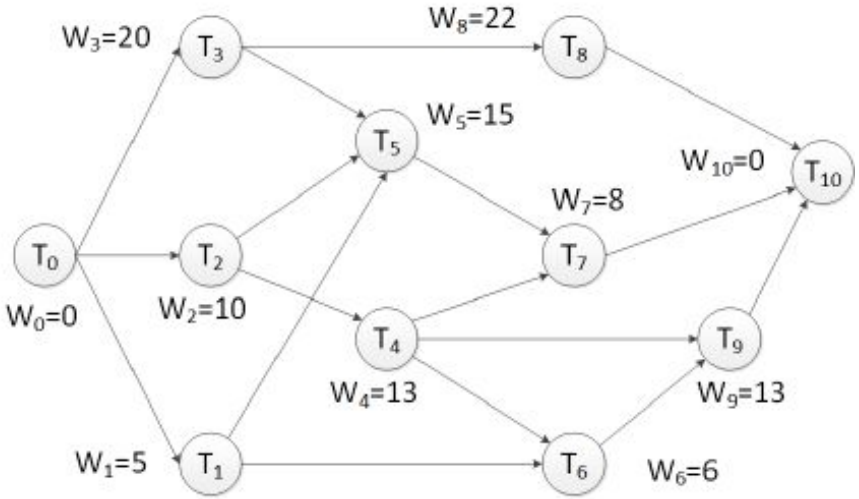


Fig. 3. Eucalyptus cloud architecture with two availability zones





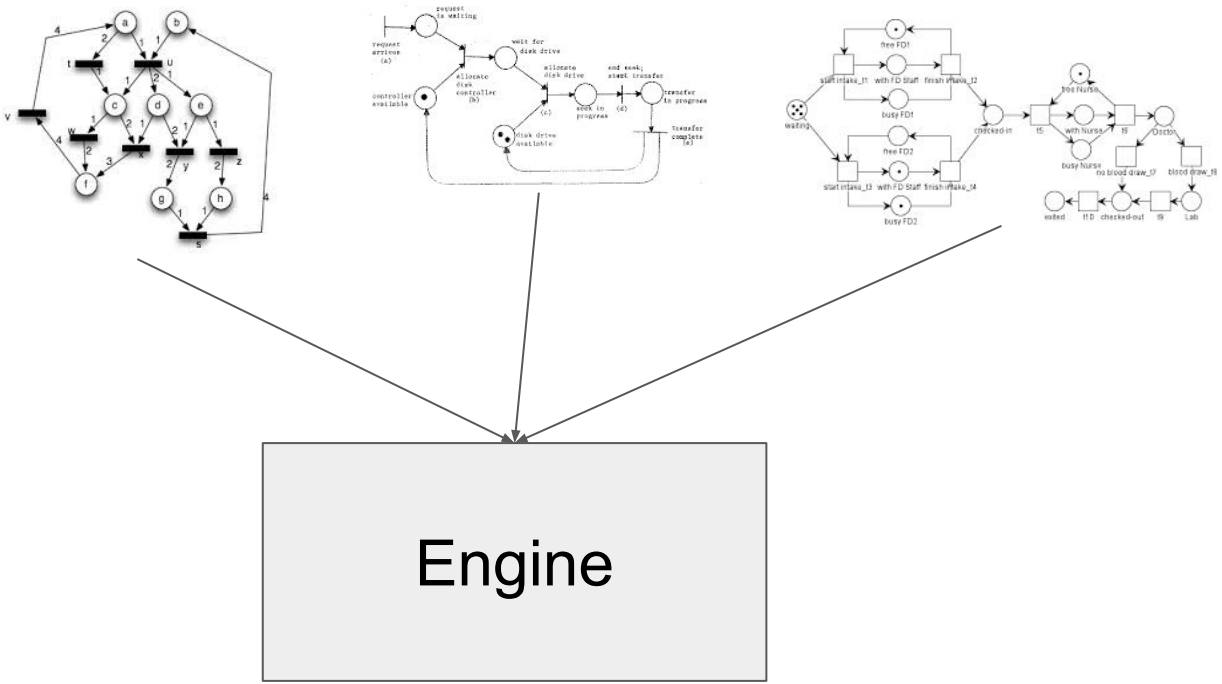
Solução

- **Object Petri Nets** para a representação do modelo
- Investigação do uso de **simulação de eventos discretos como função custo** em um algoritmo de **otimização**

Object Petri Nets

- Características:
  - Tokens podem ser objetos contendo atributos (como nas CPNs), e também comportamento
  - Transições, lugares, arcos e a própria rede também são objetos e suas classes podem ser customizadas
  - Permite o uso de herança, encapsulamento e polimorfismo em Petri nets e, além disso vários padrões de projeto podem ser usados (método fábrica, observer, command, etc.)
- A seguir veremos algumas funcionalidades do simulador desenvolvido

Simulação de várias redes ao mesmo tempo



Modificação do comportamento dos elementos da rede

```
OPN Model{

    place P0;
    place P1;

    timedTransition mu1(
        outputs = [P0( t( class = "A" ) )],
        delay = 2
    );

    timedTransition mu2(
        outputs = [P0( t( class = "B" ) )],
        delay = 2
    );

    timedTransition mu3(
        outputs = [P0( t( class = "C" ) )],
        delay = 2
    );

    timedTransition lambda extends MultiClassTrans(
        inputs = [P0(t)],
        outputs = [P1(t)]
    );
}
```

```
public class MultiClasTrans extends OPNTimedTransition {


    public MultiClasTrans(String name) {
        super(name);
    }

    @Override
    public double getRandomDelay(Map<String, OPNObject> binding) {
        OPNToken t = (OPNToken) binding.get("t");
        String clazz = t.getStringAttribute("class");

        switch (clazz) {
            case "A":
                return serviceTime(1);
            case "B":

                return serviceTime(2);
            case "C":

                return serviceTime(3);
        }

        throw new RuntimeException("Invalid class");
    }
```

```
SPN Model{

    place P0;
    place P1;
    place P2( tokens= 10 );


    immediateTransition TI0(
        inputs = [P0, P2],
        outputs = [P1]
    );

    immediateTransition TI1(
        inputs = [P0],
        inhibitors = [P2]
    );

    timedTransition TE0(
        outputs = [P0],
        delay = 1.8
    );

    timedTransition TE1(
        inputs = [P1],
        outputs = [P2],
        delay = 1.3
    );

    metric m = stationaryAnalysis( expression =
}

OPN OPNModel extends org.modcs.cloud.distributed

    reward r(
        true -> # P1
    );

}
```

Mais features...

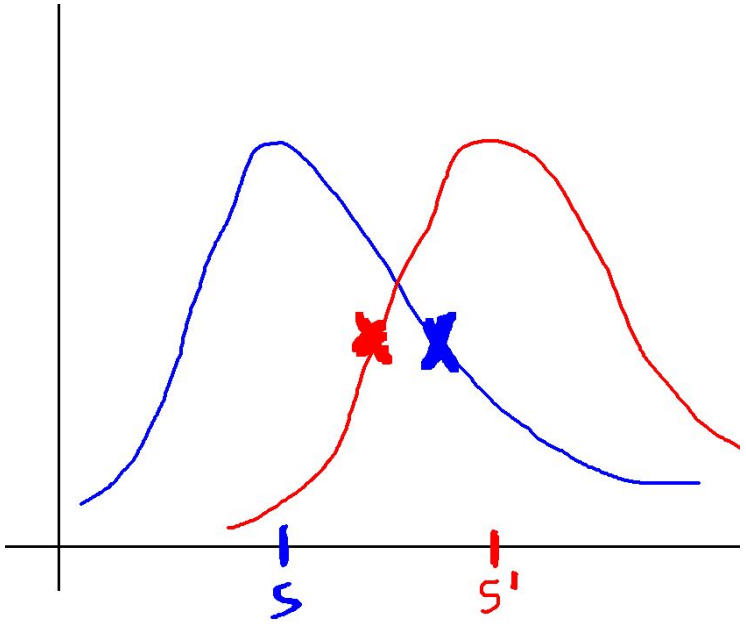
- Customização da classe de uma OPN
- Importando uma SPN dentro de uma OPN como módulo
- Taxas de recompensa em OPNs

```
public class MM1K extends OPNNet {

    @Override
    public void preCreate() {
        SPNModel model = (SPNModel) getRuntime().getModel("Model");

        model.convertToOPN(this);
    }

}
```

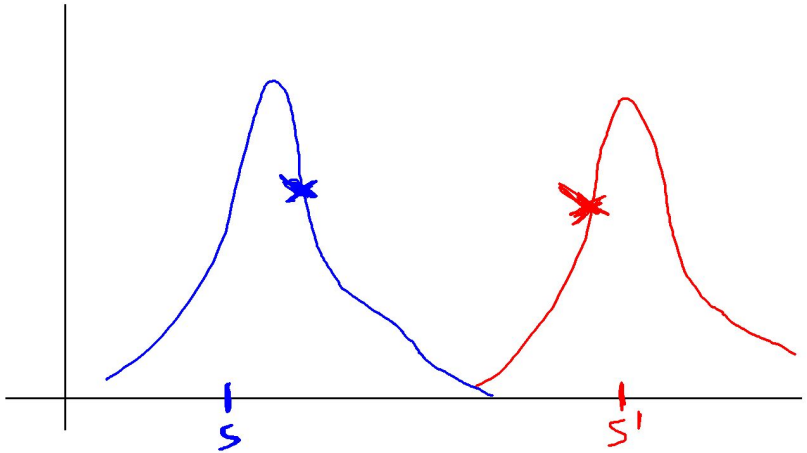
Otimização e simulação

- Ao utilizar um simulador como função custo em um algoritmo de otimização, estamos utilizando uma amostra de uma variável aleatória para comparar as soluções
- Queremos verificar o tradeoff entre:
  - Mais replicações/batches - mais tempo numa solução individual, menor probabilidade de trocar uma solução melhor por outra pior
  - Menos replicações - o contrário

S é a solução atual, S’ é a nova solução gerada aleatoriamente. O valor esperado de S’ é maior, porém, se o simulador gerar uma amostra de S maior que a de S’, estaremos trocando uma solução boa por uma pior.



Contudo, se as médias forem afastadas o suficiente, o problema anterior não ocorre,

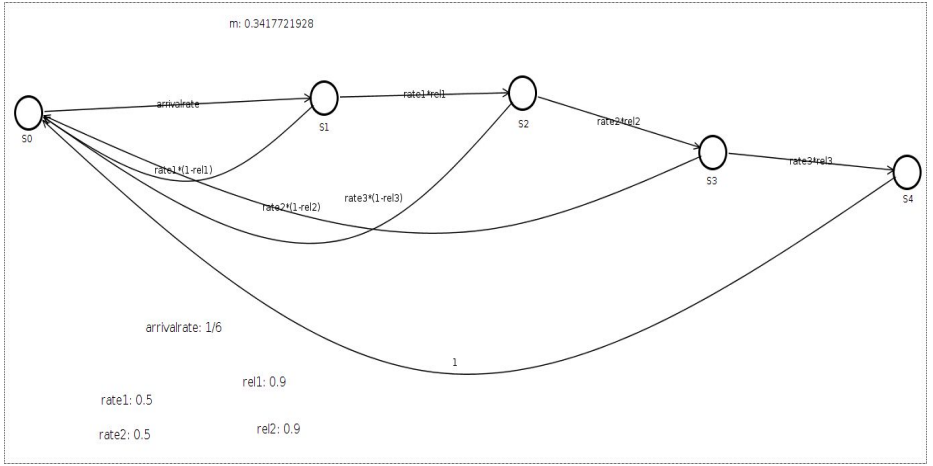


Modelo

Inspirado no modelo usado por Rubens de composição de web services, com três serviços em série.

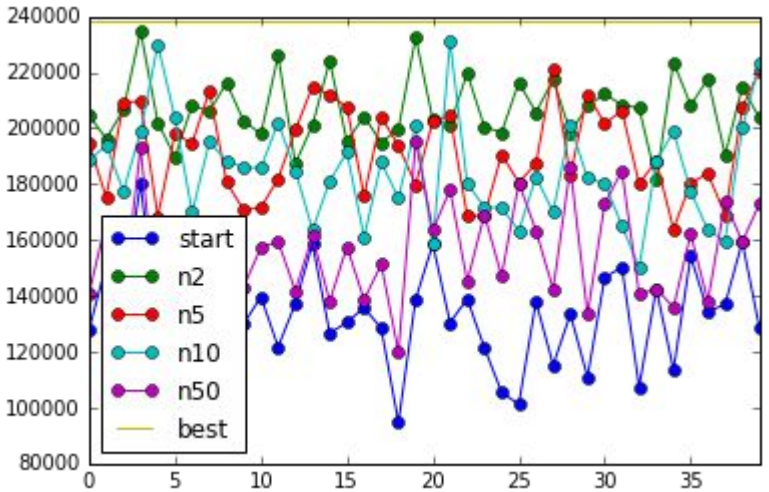
Cada serviço pode ser provido por uma lista de provedores com tempos de resposta e confiabilidades distintos.

O objetivo é descobrir a combinação provedores que maximiza a vazão de requisições

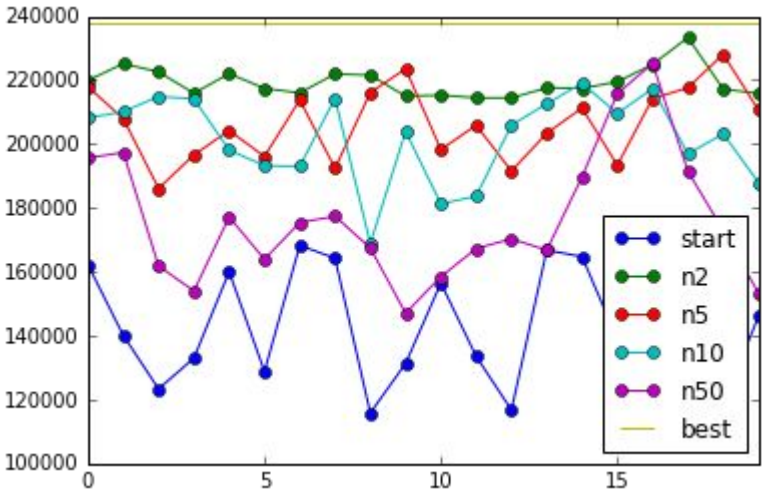


Melhor solução encontrada pelo algoritmo (hill climbing)

5 minutos

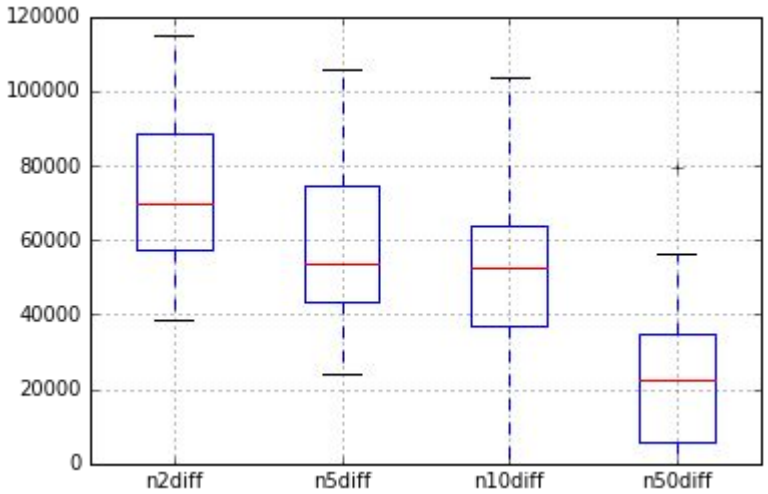


10 minutos

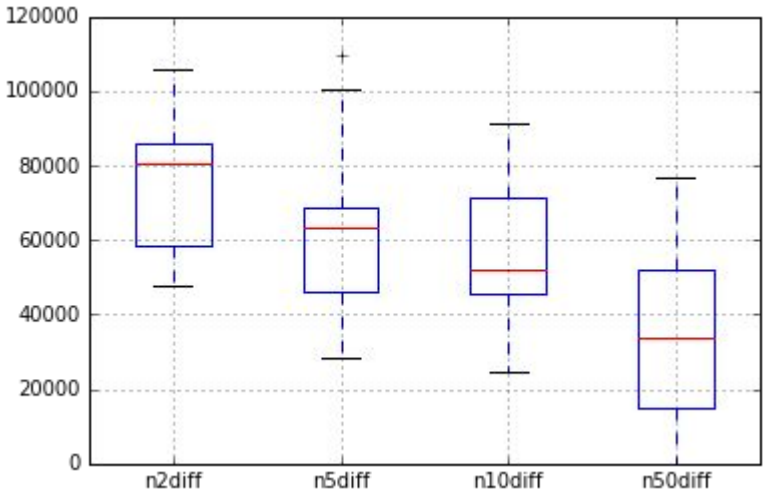


Diferença entre a solução inicial e a encontrada pelo algoritmo

5 minutos



10 minutos



Conclusões

O modelo de performabilidade de uma aplicação de workflow por meio de simulação pode gerar um número excessivo de eventos, portanto, reduzir fazer mais replicações pode atrasar o algoritmo de busca.

Os próximos passos consistem na implementação do modelo, verificação, validação e aplicação do método de otimização propriamente dito.