Um *Framework* de Otimização para Aplicações de Alto Desempenho em *Cloud*

Danilo Oliveira

Orientador: Paulo Maciel
Co-orientador: Nelson Rosa

Roteiro

1 - Introdução

2 - Modelo de aplicação HPC mestre/escravo

3 - Adicionando mecanismos de tolerância a falhas (checkpoint e restart)

4 - Estudos de caso preliminares

5 - Panorama do doutorado

6 - Conclusões

Introdução

O que é a High Performance Computing

e qual a relação dela com a *cloud computing*?

Introdução

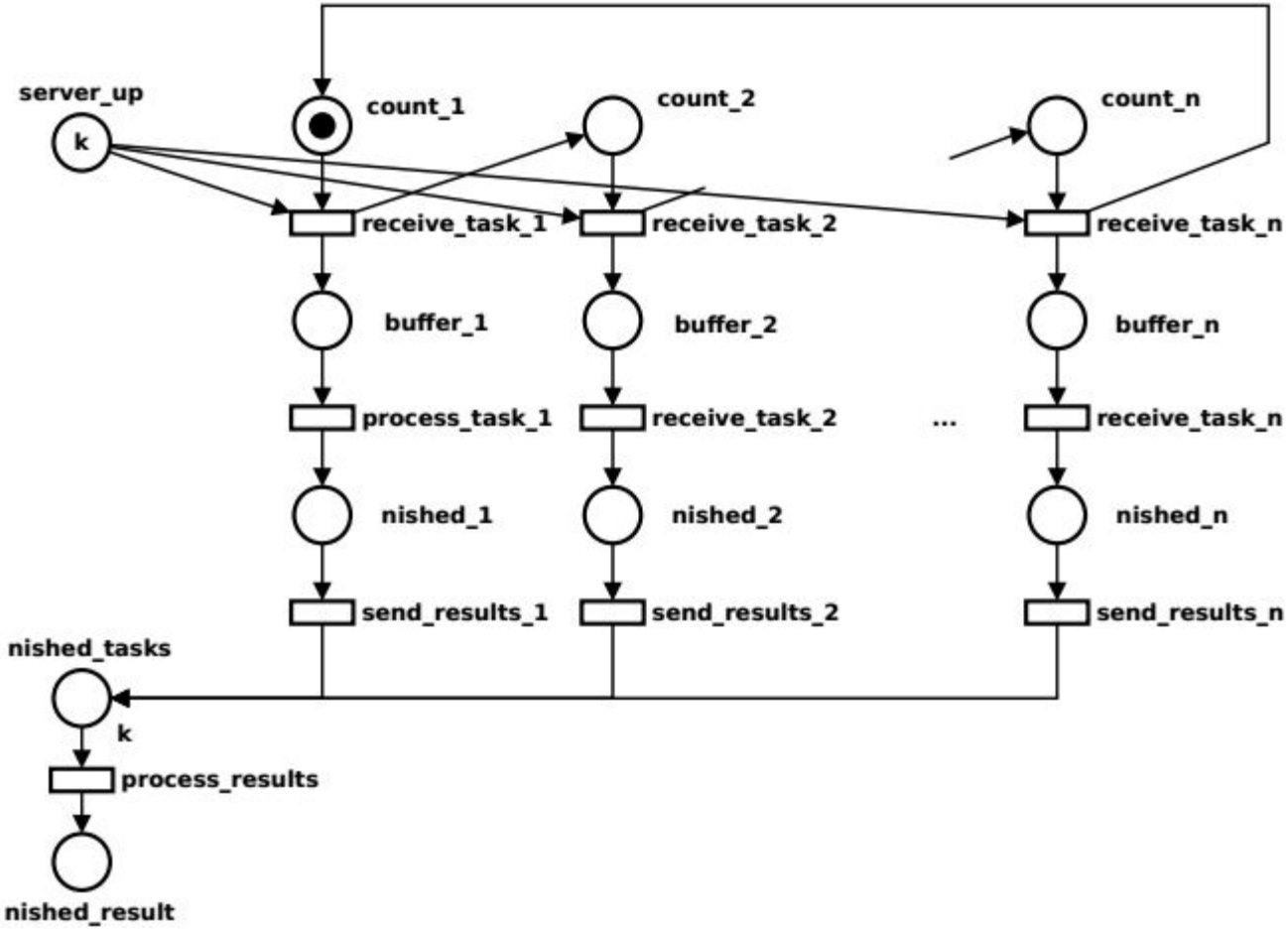
Qual a importância da tolerância a falhas em aplicações de HPC?

Proposta

- Extensões na linguagem MSL para modelar uma aplicação de HPC com arquitetura mestre escravo:
  - Comandos de repetição e condicional na declaração de modelos
  - **Simulação de SPN com programação baseada em eventos (NEW!)**

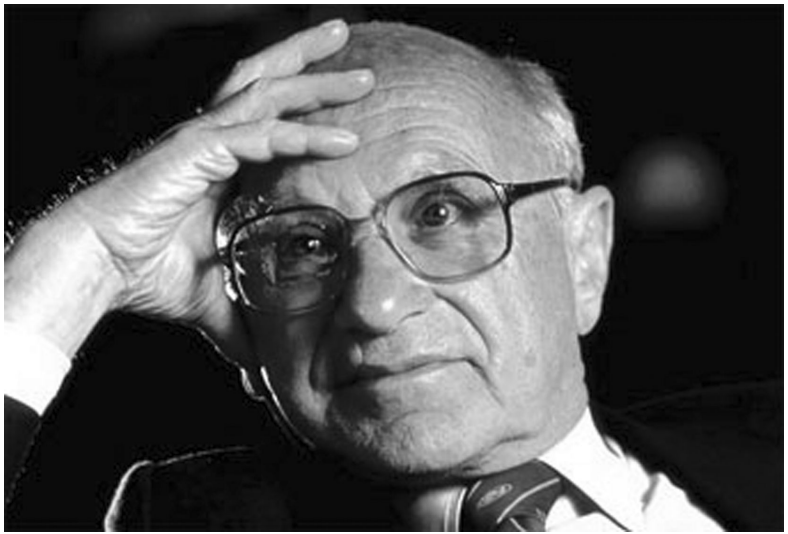
Exemplo (Pseudo código MPI):

```
int main(int argc, char **argv) {
    int my_rank;
    int size;
    MPI_Init(&argc, &argv);
    Task* tasks;
    long tasksLength;

    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if( my_rank == 0 ){
        initializeTasks(tasks, &tasksLength);
        int rank = 1;
        int counter = 0;
        while( rank < (size - 1) ){
            MPI_send(...);
            rank = counter % size;
        }
        while( ... ){
            receiveResults();
        }
        processResults();
    }else{

        while(hasMoreTasks()){
            MPI_Recv(...);
            processTask( task );
            sendResults( results );
        }
    }

    MPI_Finalize();
}
```


server_up
k
count_1
count_2
count_n
receive_task_1
receive_task_2
receive_task_n
buffer_1
buffer_2
buffer_n
process_task_1
receive_task_2
...
receive_task_n
nished_1
nished_2
nished_n
send_results_1
send_results_2
send_results_n
nished_tasks
k
process_results
nished_result

Problemas com essa abordagem

“Não existe almoço grátis em modelos baseados em espaço de estados”

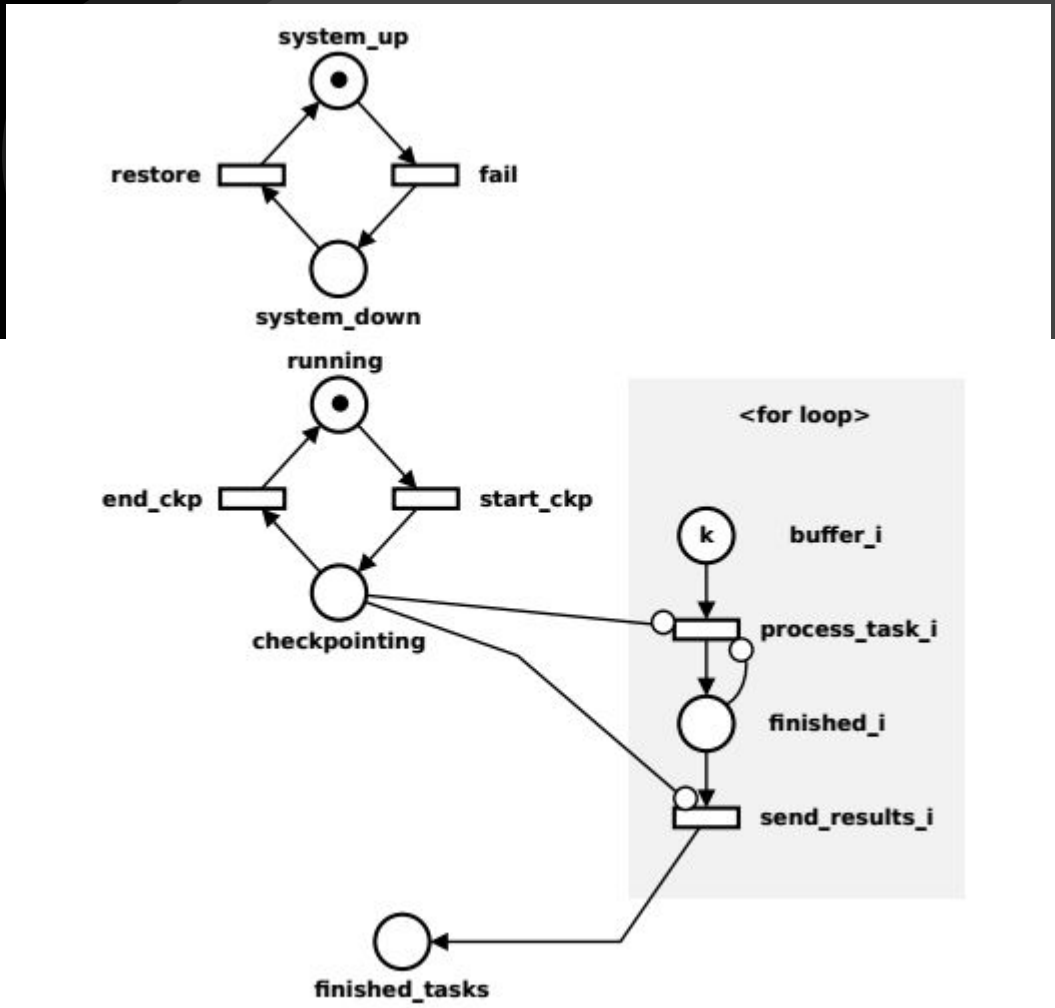


Solução

Usar simulação de eventos discretos.

Uma nova engine de simulação foi criada em virtude da necessidade da modelagem dos mecanismos de checkpoint e restart

Checkpoint e Restart



Código MSL

```
place running( tokens= 1 );
place checkpointing;

timedTransition start_ckp(
  inputs = [running],
  outputs = [checkpointing],
  delay = ( type = "Deterministic",
   checkpointInterval ) ),
  onFiring = onCheckpointStart()
);
```

Código Java

```
@Function( name = "onCheckpointStart")
public class OnCheckpointing  extends AbstractFunction{

    @Override
    public double execute() {

        ApplicationState state_ = (ApplicationState) getRuntime()
                .getApplicationState().get("checkpoint");

        Map<String, Integer> marking = getSimulation().getTokenGame()
   .getMarking();

        state_.setMarking(marking);
        return 0.0;
    }

}
```

Simulação de Petri nets + Programação orientada a eventos

Neste exemplo, usamos um código Java para salvar a marcação atual. A função de tratamento do evento do reparo irá restaurar essa marcação salva.

Outras funcionalidades possíveis:

- Modificar os parâmetros da rede “on the fly”
- Modificar variáveis que são usadas para habilitar ou desabilitar transições
- Modificar variáveis que são usadas como métricas da simulação
- Modificar a **estrutura** da rede “on the fly”, adicionar/remover lugares/transições/arcos dinamicamente

Estudo de caso

```
k = 3000;
nprocs = 500;
taskTime = 1;
send = 0.01;
mtbf = 25;
mttr = 0.5;
checkpointInterval = 10;
checkpointLength = 0.25;
nSamples = 1000;

SPN Model {
  ...
  metric m1 = variableMetric( variable = TIME,
              numberOfSamples = nSamples,
              onInit = onSimulationStart(),
              onFinish = onSimulationStop()
  );
}
main {
    m1 = solve( Model, m1 );
    println( "Wallclock time: " .. m1);
}
```

Estudo de caso #1

Tempo para terminar uma replicação individual X número de tarefas (tokens no lugar “tasks”)

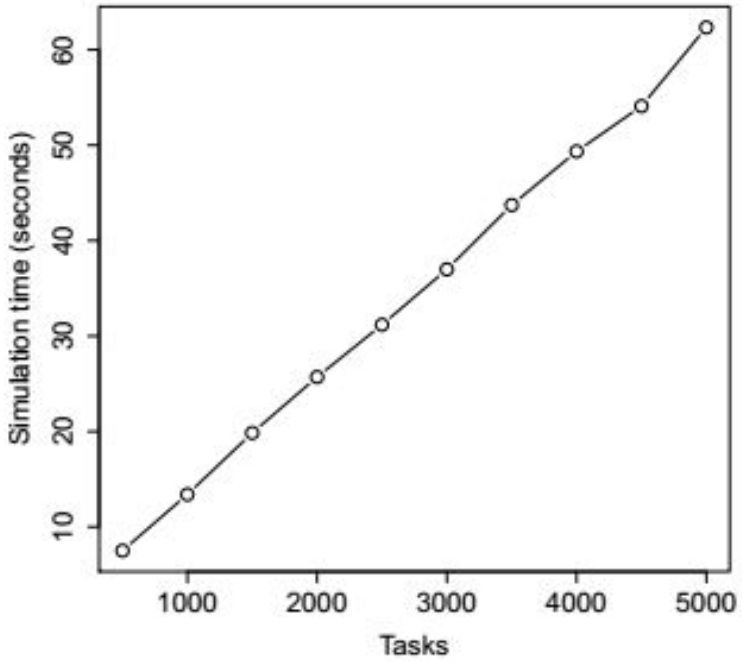


Figure 5.4: Time to finish a single run

Estudo de caso #2

Número de replicações

X

Erro

Table 5.2: Number of runs, confidence interval, and error

Number of runs	Result	C.I. low	C.I. high	Error
5	25.7417667734	24.1030054407	27.3805281062	0.6375059957
50	28.6564409209	27.5883877581	29.7244940837	0.5317509235
100	30.3995496867	29.4076970371	31.3914023363	0.4999329075
500	29.5057903589	29.1125106662	29.8990700515	0.2001708769
1000	29.610009891	29.3267266814	29.8932931005	0.1443599693
5000	29.6759362081	29.5487584987	29.8031139176	0.0648720713
10000	29.6937914034	29.6045094342	29.7830733725	0.0455473486

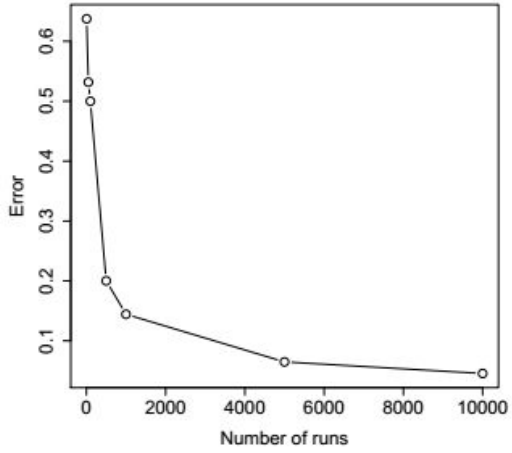
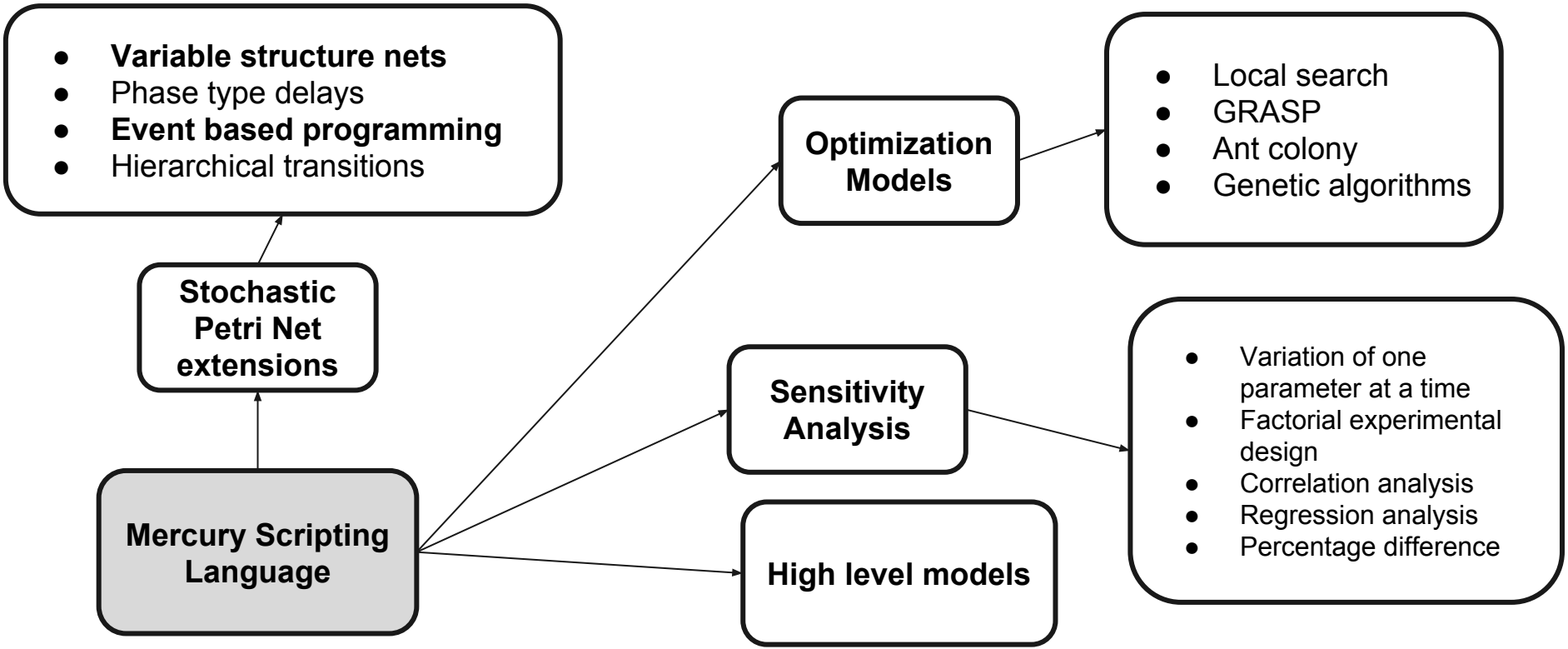


Figure 5.5: Number of runs X error

Panorama geral do doutorado /próximos passos



Considerações Finais

- A otimização de um problema de HPC seguindo essa abordagem pode ser, por si só, outro problema de HPC
- O modelo e os parâmetros deve ser cuidadosamente ajustados a fim de diminuir a complexidade da simulação e tornar o problema tratável
    - Isso gera a necessidade de criar um mecanismo para salvar o tempo dos clocks das transições