7
Prova de Conceito

7.1
Domínio

Para provar os conceitos apresentamos uma implementação do McCloud no domínio de um problema amplamente conhecido no campo da matemática. Escolhemos a aproximação do valor da constante π, que é a razão entre o perímetro e o diâmetro da circunferência.

O racional do cálculo da aproximação do π, utilizando o Método de Monte Carlo, encontra-se detalhado abaixo, ilustrada na Figura 25 [24].

1) Imagine um círculo de raio r inscrito dentro de um quadrado de lado 2 x r, ou seja, de lado duas vezes o raio do círculo ou, simplesmente, o diâmetro do círculo;

2) A geometria básica nos ensina que a área do círculo, nomeada de A, é $A = \pi.r^2$, e a área do quadrado, nomeada de B, é $B = lado^2 = (2.r)^2 = 4.r^2$.

3) Se dividirmos as áreas do círculo pela área do quadrado encontramos a relação $A / B = \pi.r^2 / 4.r^2 = \pi / 4$, então, π pode ser expresso por $\pi = 4\ A / B$.

4) Assim, se sortearmos pontos dentro da área do quadrado, este pode ou não estar dentro da área do círculo. Desta forma, sorteando uma quantidade suficientemente grande de pontos (amostra) obtemos a relação de pontos dentro do círculo e o total de pontos sorteados (obrigatoriamente dentro do quadrado), aproximando, assim, a relação A / B pelo Método de Monte Carlo e, consequentemente, o valor de π.

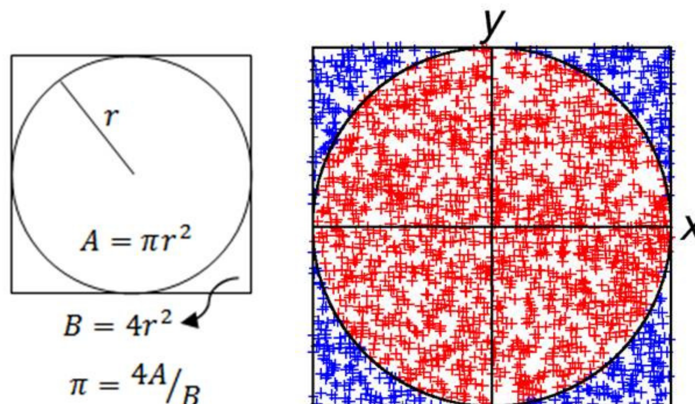

Figura 25 – Ilustração do racional da aproximação da constante pi

PUC-Rio - Certificação Digital Nº 1012652/CA

Os passos lógicos da simulação, considerando o racional descrito acima, são:

i. A incerteza envolvida é o ponto a ser sorteado, ou melhor, as duas coordenadas cartesianas x-y que definem esse ponto;

ii. Considerando o centro do quadrado como o ponto (0,0), as coordenadas podem variar entre –r e r, pois o lado do quadrado tem comprimento 2 x r e estamos interessados apenas em pontos dentro da área do quadrado;

iii. Como estamos interessados em aproximar a relação (A / B), podemos considerar, para efeito de simplificação, apenas um quarto do quadrado e consequentemente do círculo. Neste caso, escolhemos o quarto onde os eixos são positivos;

iv. Como a relação A / B se mantêm independente da escolha do tamanho do quadrado, ou seja, o valor de r. Supondo que r = 1, a distribuição de probabilidade das coordenadas x e y são os números entre 0 e r, ou melhor, entre 0 e 1;

A geração de dois valores pseudo-aleatórios, aderentes a essa distribuição, representará o resultado obtido ao se sortear um ponto, ou seja, dois valores entre 0 e 1, que representaram x e y;

i. O resultado determinístico corresponde a encontrar um resultado que indique se o ponto está ou não dentro do quadrado;

ii. Utilizando o Teorema de Pitágoras podemos extrair a relação $z = \sqrt{x^2 + y^2}$;

iii. Se o z é menor que r, i.e., o raio do círculo, no caso em questão, o número 1, o ponto está dentro da área do círculo e o resultado é positivo, caso contrário, o resultado é negativo.

Repetindo o passo B e C conseguiremos obter uma amostra suficientemente grande, ou seja, com n resultados positivos ou negativos.

Com base na amostra podemos contar quantas vezes ocorreu o Sim e dividindo pelo tamanho da amostra (n) aproximamos a relação A / B, que multiplicado por 4, que resulta na aproximação de π.

Na Figura 26 ilustramos três diferentes momentos da geração de pontos para o quarto do quadrado, onde alguns se encontram dentro do círculo e outros não.

PUC-Rio - Certificação Digital Nº 1012652/CA

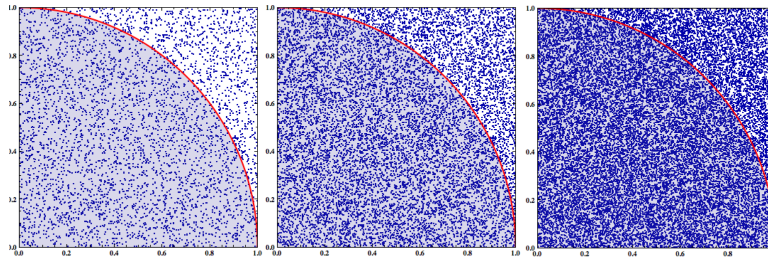
Figura 26 – Ilustração de 3 momentos da geração de pontos

Note que quanto maior o número de pontos (amostra), mais próximo estaremos de cobrir toda a área e, consequentemente, de uma aproximação mais precisa. Evidente, portanto, que quanto maior o esforço computacional melhor a aproximação obtida.

Apresentamos abaixo, na Figura 27, a implementação desta simulação para aproximação do π na linguagem C#.Net e para computação tradicional.

```
public decimal algoritmo(double n)
{
    Random a = new Random();
    int min = 0; int max = 1; double countYes = 0;
    for (double i = 0; i < n; i++)
    {
        double x = min + (a.NextDouble() * (max - min));
        double y = min + (a.NextDouble() * (max - min));
        double z = Math.Sqrt(Math.Pow(x, 2) + Math.Pow(y, 2));
        if (z < 1) countYes++;
    }
    decimal pi = ((decimal)4.0 * (decimal)countYes / (decimal)n);
    return pi;
}
```

Figura 27 – Algoritmo simulação do pi em computação tradicional

PUC-Rio - Certificação Digital Nº 1012652/CA

7.2
Implementação

Diante do entendimento do domínio optamos pela implementação da solução utilizando o McCloud e com as seguintes características:

• Envio de código textual na linguagem C#.Net pelos parâmetros *codein* e *codeout* do método *Run*, para ser compilado em tempo de execução, visando permitir o atendimento de outros domínios com essa implementação;

• Aplicação cliente na linguagem PHP5 conectando ao serviço através dos métodos *Run, Check* e *Result* para provar a independência de tecnologia;

• Não otimizar a simulação, trabalhando com uma quantidade fixa de nós, para que seja tratada essa questão em um segundo momento, no caso, no capítulo 6.

A implementação da classe com os pontos de extensão *execute*, *test*, *optimizaton* e *finish* é ilustrada na Figura 28. Repare que incluímos e utilizamos para suportar a execução de código textual da linguagem C#.Net o Mono [25], uma implementação em código aberto do Microsoft .Net Framework, que permite compilar em tempo de execução essa linguagem.

```
public class PoC : McHotspootI
{

    public string execute(string key, double n, double index, string codein, out string message)
    {

        Mono.CSharp.Evaluator.Run("using System;");
        Mono.CSharp.Evaluator.Run("double n = " + n + ";");
        string r = (string)Mono.CSharp.Evaluator.Evaluate(codein.TrimEnd());
        message = "Code: " + codein + "\r\n" + "Result: " + r;
        return r;
    }

    public string finish(string key, double n, string r, string codein, string codeout, out string message)
    {
        Mono.CSharp.Evaluator.Run("using System;");
        Mono.CSharp.Evaluator.Run("using System.Text;");
        Mono.CSharp.Evaluator.Run("using System.IO;");
        Mono.CSharp.Evaluator.Run("double n = " + n.ToString() + ";");
        Mono.CSharp.Evaluator.Run("string r = @" + "\"" + r + "\"" + ";");
        string m = (string)Mono.CSharp.Evaluator.Evaluate(codeout.TrimEnd());
        message = "Code: " + codeout + "\r\n" + "Result: " + m;
        return m;
    }

    public string test(string key, double n, string codein, string codeout)
    {
        try{
            string message = "";
            double x = 5;
            string r = execute(key, x, 1, codein, out message);
            string filesRoot = RoleEnvironment.GetLocalResource("McCloudStorage").RootPath;
            string file = Path.Combine(filesRoot, key + ".temp");
            FileStream fileStream = new FileStream(file, FileMode.Create);
            byte[] bytes = Encoding.Default.GetBytes(r);
            fileStream.Write(bytes, 0, bytes.Length);
            fileStream.Close();
            string m = finish(key, x, file, codein, codeout, out message);
            if (m != "") return "";
            else return "Error: Null Result";
        }
        catch(Exception e) { return "Error: "+e.ToString(); }
    }

    public void optimization(double n, string codein, string codeout,
        out double ninstancesadded, out double ntasks, out int timeoutInSeconds)
    {
        ninstancesadded = 0;
        timeoutInSeconds = 2 * 60 * 60;
        ntasks = 16;
    }

}
```

Figura 28 – Implementação *execute* e *finish* para a simulação do pi

Repare que o *execute* apenas passa *n* e executa o *codein* com o Mono [25], retornando o resultado desta execução para o arcabouço. O mesmo ocorre com o *finish*, onde o *codeout* é executado. No entanto, neste a entrada, é o endereço do arquivo onde estão todas as saídas do *execute* concatenadas. Esta entrada é feita via arquivo, pois o volume de dados pode não permitir a alocação total em memória. O *test* executa a simulação para um número pequeno de realizações, a

PUC-Rio - Certificação Digital Nº 1012652/CA

fim de validar os parâmetros de entrada. Finalmente o *optimazation* apenas define, de forma fixa, o número de tarefas e o tempo de espera de cada processamento, pois no contexto deste capítulo, ainda não estamos interessados em buscar uma simulação ótima, portanto, nenhum nó é levantado em tempo de execução para a simulação, sendo o número de nós fixo, no caso, está em 16 nós. Optamos por realizar uma tarefa por nó, ou seja, 16 tarefas também. O tempo máximo de execução de uma tarefa foi configurado com o máximo permitido (2 horas).

O parâmetro *codein* em formato textual é apresentado na Figura 29 é praticamente igual à parte intensiva em processamento do algoritmo, apresentado para computação tradicional na Figura 27. Repare que a mudança consiste no retorno em formato *string* (textual), e em apresentar apenas a parte de geração da amostra, não realizando a aproximação final. A última variável é o valor retornado.

```
Random a = new Random();
int min = 0;
int max = 1;
double c = 0;
for (double i = 0; i < n; i++)
{
      double x1 = min + (a.NextDouble() * (max - min));
      double x2 = min + (a.NextDouble() * (max - min));
      double x = Math.Sqrt(Math.Pow(x1 - 1, 2) + Math.Pow(x2 - 1, 2));
      if (x < 1) c = c + 1;
}
string r = c.ToString() + ";";
r;
```

Figura 29 – Parâmetro codein para aproximação do pi

O parâmetro *codeout* em formato textual é apresentado na Figura 30.

```
FileStream fileStream = new FileStream(r, FileMode.Open);
StreamReader reader = new StreamReader(fileStream);
string content = reader.ReadToEnd();
fileStream.Close();

string[] rAux = content.Split(\';\');
double sum = 0;
for (int i = 0; i < (rAux.Length-1); i++)
{
      sum = sum + double.Parse(rAux[i]);
}
decimal estPi = ((decimal)4.0 * (decimal)sum / (decimal)n);
string f = estPi.ToString();
f;
```

Figura 30 – Parâmetro codeout para aproximação do pi

PUC-Rio - Certificação Digital Nº 1012652/CA

Nesse caso, não teremos um único resultado de entrada, mas a concatenação do resultado de todas as execuções do *codein*, recebido concatenado em arquivo. Dessa forma, é necessário ler o arquivo, formatar, e somar as partes, antes de executar o mesmo cálculo final do algoritmo apresentado para computação tradicional ilustrado pela Figura 27. De forma similar ao parâmetro *codein*, a última variável é o valor retornado. Repare que o código tanto para o *codein* quanto para o *codeout* são similares ao código proposto para a computação tradicional, com as diferenças destacadas a seguir.

1. O n é enviado como parâmetro ao *codein* pelo McCloud e se refere à execução de responsabilidade da tarefa e não da simulação total;

2. O resultado do *codein* deve ser enviado em formato textual, pois é concatenado aos demais resultados das tarefas antes de ser enviado ao *codeout*;

3. Os resultados das tarefas já concatenado em um arquivo é enviado ao *codeout*, que deve tratá-lo, bem como calcular a aproximação do π com essa amostra total.

A implementação da aplicação cliente em PHP5 para consumir o método *Run* é apresentada na Figura 31. O *wsdl* refere-se ao endereço do nó que exerce o papel WCF Role, publicado no Portal do Azure. O *codein* e *codeout* devem ser preenchidos conforme apresentado anteriormente para a aproximação do π. O *n* deve ser preenchido com o número de realizações desejado para a simulação. Neste caso, está sendo recebido como parâmetro na URL da página.

```
<?php
set_time_limit (0);
$wsdl = 'http://maccloudcsharp.cloudapp.net/Service.svc?wsdl';
$mcc = new SoapClient($wsdl);
$obj->n = $_GET['n'];
$obj->codein = '...';
$obj->codeout = '...';
$result = $mcc->Run($obj);
$key = $result->RunResult;
echo $key
?>
```

Figura 31 – Consumo do método Run em PHP5

A implementação da aplicação cliente em PHP5 para consumir o método *Check* e *Result* é apresentada na Figura 32. O *key* deve ser preenchido com a identificação que se deseja checar o andamento. No caso, a retornada pela página apresentada acima.

PUC-Rio - Certificação Digital Nº 1012652/CA

```php
<?php
set_time_limit (0);
$wsdl = 'http://maccloudcsharp.cloudapp.net/Service.svc?wsdl';
$mcc = new SoapClient($wsdl);
$obj->key = '...';
$result = $mcc->Check($obj);
$status = $result->CheckResult;
if($status != "finished") echo $status;
else {
	$result = $mcc->Result($obj);
	$url = $result->ResultResult;
	$pi = file_get_contents($url);
	echo "PI = ".$pi;
}
?>
```

Figura 32 – Consumo do método Check e Result na aproximação do pi

7.3 Performance

Executamos o código proposto para aproximação do π em computação tradicional (Figura 27) e a implementação do o McCloud que acabamos de apresentar (seção 7.2) com diferentes tamanhos de amostra como entrada. Os resultados são apresentados na Tabela 3.

A coluna “N” apresenta o tamanho da amostra (realizações). A coluna “Precisão” informa com quantas casas decimais conseguimos aproximar corretamente π. A coluna “W” e “Tarefas” indica o número de nós no papel de *Worker Role* e de tarefas do experimento, enquanto a coluna “Tar./W” indica o número arredondado de tarefas por nó. As colunas do conjunto “Tempo” apresentam o tempo total e de cada etapa dos experimentos, considerando a implementação proposta como prova de conceito. Neste tempo não foi considerado o intervalo necessário para instanciar e desativar os nós, que levou, em média, em torno de 7 minutos. A coluna “Blob>Interno” indica a quantidade de dados armazenadas dentro da nuvem e a “Blob>Saída” a quantidade de dados retornado ao solicitante da simulação. A coluna “Comparação>Tempo” apresenta o tempo da computação tradicional, enquanto a coluna “Comparação>Razão” apresenta a razão do tempo tradicional em relação ao tempo com o McCloud do experimento (*speedup*).

PUC-Rio - Certificação Digital Nº 1012652/CA

PROVA DE CONCEITO PERFORMANCE															
Entrada			Configuração			Tempo (segundos)				Blob		CUSTO	Comparação	
#	N	Precisão	W	Tarefas	Tar./W	Split	Process	Merge	Total	Interno	Saída	USD$	Tempo	Razão
1	100	1	16	16	1	1,59	6,83	1,84	**10,26**	160 Byte	13 Byte	**2,19**	0,016	0,00
2	1.000	1	16	16	1	1,44	1,87	1,92	**5,23**	160 Byte	13 Byte	**2,19**	0,109	0,02
3	10.000	2	16	16	1	3,78	13,00	1,81	**18,59**	160 Byte	13 Byte	**2,19**	0,375	0,02
4	100.000	3	16	16	1	1,45	10,53	1,77	**13,75**	160 Byte	13 Byte	**2,19**	0,641	0,05
5	1.000.000	3	16	16	1	1,42	4,90	1,75	**8,07**	160 Byte	13 Byte	**2,19**	1,719	0,21
6	10.000.000	3	16	16	1	1,44	10,12	1,95	**13,51**	160 Byte	13 Byte	**2,19**	11,531	0,85
7	100.000.000	3	16	16	1	1,38	9,92	1,78	**13,08**	160 Byte	13 Byte	**2,19**	109,438	8,37
8	1.000.000.000	4	19	19	1	1,80	33,03	1,89	**36,72**	190 Byte	13 Byte	**2,55**	1.234,047	33,61
9	10.000.000.000	4	19	19	1	1,92	179,81	2,00	**183,73**	190 Byte	13 Byte	**2,55**	9.857,360	53,65
10	100.000.000.000	5	19	19	1	4,22	1696,69	6,39	**1707,30**	190 Byte	13 Byte	**2,55**	98.482,969	57,68

Tabela 3 – Teste de performance da prova de conceito

Optamos pelo número máximo de 19 nós no papel *Worker Role*, que adicionado ao nó que exerce o papel *WCF Role*, somam 20 instâncias, limite máximo da conta gratuita. Portanto, todos esses experimentos, foram realizados de forma gratuita, pois respeitamos o limite de instâncias, e não atingimos o consumo das 750 horas de computação permitidas.

7.4 Análise dos Resultados

Quando N é pequeno, o tempo de gerenciamento das tarefas é maior que o ganho com o paralelismo. No entanto, para um erro de aproximação adequado, temos um n suficientemente grande, onde a solução que propomos apresenta significativos ganhos de performance, chegando a ser cinquenta e sete vezes mais rápido no teste de maior número de realizações (N = 100.000.000).

Com toda certeza podem existir pequenas diferenças de capacidade de processamento das instâncias em relação à máquina utilizada para computação tradicional, assim como performance distintas dos armazenamentos e rede na nuvem. Contudo, nessa ordem de ganho, tal questão se torna secundária.

O custo de execução para esta prova de conceito com o maior tamanho de amostra correspondeu a 2 horas de computação pequena em 20 máquinas (20 x 1 x USD$0,12), mais os custos mínimos de armazenamento (USD$0,14), e transações (USD$0,01), totalizando USD$2,55.

Este pequeno exemplo serve para demonstrar o potencial que a computação na nuvem tem para alavancar a adoção de simulações baseadas no Método de Monte Carllo, que no momento tem sido limitada pela infraestrutura computacional disponível.

PUC-Rio - Certificação Digital Nº 1012652/CA

Quanto à implantação realizada nessa prova de conceito, o serviço disponibilizado não se restringe ao caso proposto, pois aceita qualquer código textual na linguagem C#.Net pode ser enviado através dos métodos, ou seja, é possível implementar diversas outras simulações. Além disso, o usuário tem liberdade para implementar a aplicação cliente na complexidade e utilizando a tecnologia que deseja.

PUC-Rio - Certificação Digital Nº 1012652/CA