Monte Carlo Service in Windows Azure

Rafael Nasser <rnasser@inf.puc-rio.br>
Karin Breitman <karin@inf.puc-rio.br>
Hélio Lopes <lopes@inf.puc-rio.br>
Agenda

- Introducing you to McCloud Service Framework, *the easiest way to develop Monte Carlo simulations in the cloud!*

- Demonstrate **two different** implementations with McCloud;

- One in **mathematical** filed with simulations written in **C#**;

- And the other, in **mechanical engineering** area in **Matlab technology**.
Monte Carlo

- **MC** is a statistical method to *approximate solutions* of problems;
- It is very useful in a wide range of fields;
- In reality **MC** simulations work out an experiment several times using *random numbers* as input to describe the statistical behavior of a problem;
- The **higher the number of samples on MC**, better is the *approximation*;
- Subsequently, **MC** has a large demand on *computational resources*, but on the other hand, it is *easy to parallelize* samples;
- Which makes it an excellent candidate to *go to the cloud*. 
Service Framework

We proposed a framework to develop simulation services

• Why a Framework?
  ➡️ Because each problem is particular in its own way. Given that our aim is to allow you to easily and quickly implement the simulation without cloud concerns.

• Why a Service?
  ➡️ Because we want Monte Carlo simulation alone to go to the cloud with minimum changes in your application.
Conceptual Architecture
This solution has been structured in three layers

**Application**
Service Client which can be in any Technology

**Software as a Service**
Monte Carlo Simulation Service implemented with McCloud

**Platform as a Service**
Cloud platform with all you need in your simulation
It is available in a friendly web page (*complexity are hidden*)
Framework Workflow

In this demonstration...

- **The framework methods are in green**
  They are available to framework clients.

- **The framework hotspots are in orange**
  They allow the developers to extend the framework.
1. Let’s imagine an application requesting a simulation to proposed framework
2. The arguments of this simulation are tested according to the “test” hotspot

3. And we estimated and got an optimal configuration to execute the simulation according to the “optimization” hotspot, possible starting new Azure worker nodes, running the “startup” hotspot to configuration each node
4. The simulation has been split in tasks according to an ideal number of tasks estimated. The tasks are saved in the queue.

5. The arguments of this simulation are saved onto the table.
6. This part of workflow we called **SPLIT STAGE**

7. In the end of stage the client receive a key to identify this simulation
8. In an asynchrony way, the worker nodes available take tasks from the queue in order, one by one.

9. The task is processed by the worker node which picked it according to the implementation of “execute” hotspot.
10. In the end of each process the result is saved in a blob

11. This is the **PROCESS STAGE**
12. The node that had processed the last piece, which can be anyone, will start the next stage.

13. All the tasks result are rescued.

14. This node aggregates all tasks result and calculate the approximation according to the “finish” hotspot.
15. The result of approximation is saved in blob and the simulation status is updated on the table.

16. This is the MERGE STAGE, the last one.
17. At any time the client can check the status of simulation with “Check” method using the simulation key;

18. When the simulation is over, the client can download the results with “Result” method.
Framework Methods

The three methods available in the framework are

```csharp
/// <summary>
/// Runs this simulation in the cloud
/// </summary>
/// <param name="n">Total samples you want in this simulation</param>
/// <param name="codein">Code fragment to use in 'execute' hotspot</param>
/// <param name="codeout">Code fragment to use in 'finish' hotspot</param>
/// <param name="npt">Maximum samples by task</param>
/// <returns>One key to identify this simulation</returns>
[OperationContract]
string Run(double n, string codein, string codeout, double npt);

/// <summary>
/// Checks the status of this simulation
/// </summary>
/// <param name="key">One key to identify one simulation</param>
/// <returns>Status of this simulation</returns>
[OperationContract]
string Check(string key);

/// <summary>
/// Gets link to download result of this simulation
/// </summary>
/// <param name="key">One key to identify one simulation</param>
/// <returns>Link to download the result</returns>
[OperationContract]
string Result(string key);
```
The main framework hotspots are implemented extending the class `McHotspot`.

```csharp
namespace McHotspoot
{
    public interface McHotspotI
    {
        /// <summary>
        /// Generates samples of one task
        /// </summary>
        /// <param name="key">Key generated on 'Run' method</param>
        /// <param name="n">Total samples of this task defined by framework</param>
        /// <param name="index">Index of this task defined by framework</param>
        /// <param name="codein">Same code fragment received on 'Run' method</param>
        /// <param name="message">Returns message log to framework</param>
        /// <returns>String with task results processed</returns>
        string execute(string key, double n, double index, string codein, out string message);

        /// <summary>
        /// Calculates the approximation of simulation
        /// </summary>
        /// <param name="key">Key generated on 'Run' method</param>
        /// <param name="n">Total samples received on 'Run' method</param>
        /// <param name="r">File address with all tasks results</param>
        /// <param name="codein">Same code fragment received on 'Run' method</param>
        /// <param name="message">Returns message log to framework</param>
        /// <returns>String with simulation approximation calculated</returns>
        string finish(string key, double n, string r, string codein, string codeout, out string message);
    }
}
```
Framework Hotspots

```csharp
/// <summary>
/// Tests the parameter received on 'Run' method
/// </summary>
/// <param name="key">Key generated on 'Run' method</param>
/// <param name="n">Total samples received on 'Run' method</param>
/// <param name="codein">Same code fragment received on 'Run' method</param>
/// <param name="codeout">Same code fragment received on 'Run' method</param>
/// <param name="npt">Same maximum samples by task received on 'Run' method</param>
/// <returns>Returns a message if a problem comes up</returns>
string test(string key, double n, string codein, string codeout, double npt);

/// <summary>
/// Estimates the best framework configuration to run this simulation
/// </summary>
/// <param name="n">Total samples received on 'Run' method</param>
/// <param name="codein">Same code fragment received on 'Run' method</param>
/// <param name="codeout">Same code fragment received on 'Run' method</param>
/// <param name="npt">Same maximum samples by task received on 'Run' method</param>
/// <param name="ninstancesadded">Amount of new worker nodes to initialize</param>
/// <param name="ntasks">Amount of tasks to split this simulation</param>
/// <param name="timeoutInSeconds">Maximum time to wait the execute of task</param>
void optimization(double n, string codein, string codeout, double npt, 
                   out double ninstancesadded, out double ntasks, out int timeoutInSeconds);
```
The last hotspot, **startup**, is different...

- It needs to be implemented writing the Windows command line file `startup.cmd` and updating files to `onstart` container.

- The framework **runs** this commands and **copies** the files in this container when each node starts.

- Therefore you can **download** and **install** everything you need to implement the other hotspots supporting **any other technology** in simulations!
It’s easy to notice that π can be represented by a division of circle area by square area, if one is exactly inside the other.

Classic Example

The approximation of π number with Monte Carlo

It’s also easy to see that you can approximate the square and circle area generating many random points inside that.

Consequentially, π can be approximated by:

\[ \pi \approx \frac{4 \times \text{Total of red points}}{\text{Total points generated}} \]
With McCloud we implemented a service to run a C# textual code received in `codein` and `codeout` of Run method.

**Traditional Algorithm**

```csharp
public decimal traditionalAlgorithm(double n)
{
    // Initializes System.Random class, using the specified seed value
    Random a = new Random(1982 + n - index + 2);
    int min = 0; int max = 1; double countYes = 0;

    // Generates n samples of points
    for (double i = 0; i < n; i++)
    {
        // Generates random coordinates (x-y) of one point
        double x = min + (a.NextDouble() * (max - min));
        double y = min + (a.NextDouble() * (max - min));

        // Increments the counter if the point inside the circle
        if (Math.Sqrt(Math.Pow(x - 1, 2) + Math.Pow(y - 1, 2)) < 1)
            countYes++;
    }

    // Calculates the approximation
    return (decimal)4.0 * (decimal)countYes / (decimal)n;
}
```

**Bear in mind that you must control the seed!**

We use Mono CSharp, a free compiler runtime of C#, to run this codes in hotspots.

We have done minimal changes to the code!
## Performance Comparison

<table>
<thead>
<tr>
<th>Amount of Points</th>
<th>Decimal Precision</th>
<th>Duration in 1 Computer</th>
<th>Worker Nodes</th>
<th>Duration McCloud</th>
<th>USD$ McCloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>0.016 s.</td>
<td>16</td>
<td>10.26 s.</td>
<td>2.19</td>
</tr>
<tr>
<td>1,000</td>
<td>1</td>
<td>0.109 s.</td>
<td>16</td>
<td>5.23 s.</td>
<td>2.19</td>
</tr>
<tr>
<td>10,000</td>
<td>2</td>
<td>0.375 s.</td>
<td>16</td>
<td>18.59 s.</td>
<td>2.19</td>
</tr>
<tr>
<td>100,000</td>
<td>3</td>
<td>0.641 s.</td>
<td>16</td>
<td>13.75 s.</td>
<td>2.19</td>
</tr>
<tr>
<td>1,000,000</td>
<td>3</td>
<td>1.719 s.</td>
<td>16</td>
<td>8.07 s.</td>
<td>2.19</td>
</tr>
<tr>
<td>10,000,000</td>
<td>3</td>
<td>11.531 s.</td>
<td>19</td>
<td>13.51 s.</td>
<td>2.55</td>
</tr>
<tr>
<td>100,000,000</td>
<td>3</td>
<td>109.438 s.</td>
<td>19</td>
<td>13.08 s.</td>
<td>2.55</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>4</td>
<td>20.57 min.</td>
<td>19</td>
<td>36.72 s.</td>
<td>2.55</td>
</tr>
<tr>
<td>10,000,000,000</td>
<td>5</td>
<td>2.74 hours</td>
<td>19</td>
<td>3.06 min.</td>
<td>2.55</td>
</tr>
<tr>
<td>100,000,000,000</td>
<td>5</td>
<td>1.14 days</td>
<td>19</td>
<td>28.46 min.</td>
<td>2.55</td>
</tr>
<tr>
<td>1,000,000,000,000</td>
<td>5</td>
<td>11.2 days</td>
<td>99</td>
<td>5.71 min.</td>
<td>12.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99</td>
<td>54.64 min.</td>
<td>12.15</td>
</tr>
</tbody>
</table>

300 speedup

Working with small amount of points the McCloud isn´t a good solution.

But, if you want a better precision, McCloud speedup results with insignificant costs.
Case Study

We apply the framework in a real problem of Mechanical Engineering, the approximation of bar displacement.

Thanks to PUC-Rio Mechanical Department.
They model equations and program this systems in Matlab executable.
Contrasting π example, in this case we can’t run this executable with many samples, because MatLab do an overflow memory.

To solve this, we split it in two pieces, one to generate samples and the other to aggregate samples and calculate the approximation. Similar to what we did in π with codein/codeout. Therefore, we can run more samples...

With McCloud we implement a service to run MATLAB executables uploaded in onstart container.

Besides, on startup.cmd we download and install the Matlab Compiler Runtime in all nodes to support this technology in Azure.
1. With 64 nodes we see a great speedup in all system with low cost;

2. I’d like to stress the researchers have never succeed in executing more 1,024 samples (we performed 262.144);

3. As pointed out on this case we have big-data inside the cloud but small outside data transfer.
The yellow region on right, with more samples, fits better the mean in blue. It represents the more precision of standard deviation.

Also, the graphic on the right with more density, better represents the statistical behavior of the problem.
McCloud allow scientists to \textit{scale up} their experiments in the cloud, with \textit{minimal changes} to the code they are used to, with very \textit{low cost}. 

Conclusion
Hotspots (C# Textual Code)

```csharp
public class CSharp : MHotspot
{
    public string execute(string key, double n, double index, string codein, out string message)
    {
        Mono.CSharp.Evaluator.Run("using System;\n");
        Mono.CSharp.Evaluator.Run("double index = " + index + ";\n");
        Mono.CSharp.Evaluator.Run("double n = " + 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;\n");
        Mono.CSharp.Evaluator.Run("using System.Text;\n");
        Mono.CSharp.Evaluator.Run("using System.IO;\n");
        Mono.CSharp.Evaluator.Run("double n = " + n.ToString() + ";\n");
        Mono.CSharp.Evaluator.Run("string r = @" + "\n" + r + "\n" + ";\n");
        string m = (string)Mono.CSharp.Evaluator.Evaluate(codeout.TrimEnd());
        message = "Code: " + codeout + "\r\n" + "Result: " + m;
        return m;
    }
}
```
public class Matlab : McHotspot1
{
    public string execute(string key, double n, double index, string codein, out string message)
    {
        string filename = "process";
        string filesRoot = RoleEnvironment.GetLocalResource("McCloudStorage").RootPath;
        string matlab1 = Path.Combine(filesRoot, filename + ".exe");
        string output1 = "P_" + key + "index + ".csv";
        exec(matlab1, new string[] { n.ToString(), index.ToString(), codein, output1 }, out message);
        FileStream fileStream = new FileStream(output1, FileMode.Open);
        StreamReader reader = new StreamReader(fileStream);
        string r = reader.ReadToEnd();
        fileStream.Close();
        return r;
    }

    public string finish(string key, double n, string r, string codein, string codeout, out string message)
    {
        string filename = "merge";
        string filesRoot = RoleEnvironment.GetLocalResource("McCloudStorage").RootPath;
        string matlab2 = Path.Combine(filesRoot, filename + ".exe");
        string output1 = r;
        string output2 = "M_" + key + " + ".csv";
        exec(matlab2, new string[] { n.ToString(), codeout, output1, output2 }, out message);
        string m = "";
        FileStream fileStream2 = new FileStream(output2, FileMode.OpenOrCreate);
        StreamReader reader = new StreamReader(fileStream2);
        m = reader.ReadToEnd();
        fileStream2.Close();
        return m;
    }
}
Hotspots (Matlab)

```Matlab
C:\Sharp.cs  Matlab.cs  Startup.cmd  X

 "$MATLAB"
 REG QUERY %regpath% /v vcredist_x86
 IF errorlevel 1 (  
  BootStrapper.exe -get %MyBlobStorageEndpoint%/startup/vcredist_x86.exe -1r C:\cloud\blob -run C:\cloud\vcredist_x86.exe -args /w /s /v/qn -block
  REG ADD %regpath% /v vcredist_x86 /t REG_SZ /d 1 /f
 )

 REG QUERY %regpath% /v MCRInstaller
 IF errorlevel 1 (  
  BootStrapper.exe -get %MyBlobStorageEndpoint%/startup/MCRInstaller.exe -1r C:\cloud\ -block
  net start "task scheduler"
  net user McCloud m4c2d4h49 /add
  net localgroup Administrators McCloud /add
  schtasks /create /SC ONCE /ST 00:00 /SD 01/01/2100 /TN task_MCRInstaller /TR "C:\cloud\MCRInstaller.exe /w /s /v/qn" /F /RU McCloud /RP m4c2d4h49 /RL HIGHEST
  schtasks /Run /TN task_MCRInstaller
  REG ADD %regpath% /v MCRInstaller /t REG_SZ /d 1 /f
  shutdown -r -t 300
 )
```
function process(n, index, codein, output)
    tic
    N = str2double(n);
    rng(str2double(codein)+N*2*str2double(index));
    min = 0; max = 1; c = 0;
    for i=1:N
        x1 = min + (rand(1,1) * (max - min));
        x2 = min + (rand(1,1) * (max - min));
        x = sqrt( (x1 - 1)^2 + (x2 - 1)^2 );
        if (x < 1)
            c = c + 1;
        end
    end
    fid=fopen(output, 'w');
    fprintf(fid,'%u;','c);
    fclose(fid);
end

function merge( n, codeout, input, output)
    tic
    N = str2double(n);
    fid1=fopen(input,'r');
    r = fscanf(fid1, '%u;');
    sum = 0;
    [lines, cols] = size(r);
    for i=1:lines
        sum = sum + r(i,1);
    end
    estPi = 4 * sum / N;
    fid2=fopen(output,'w');
    fprintf(fid2,'%f',estPi);
    fclose(fid2);
toc
end
<?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
?>
Essa aplicação cliente se conectou ao serviço disponível no endereço: http://127.0.0.1:84/Service.svc

Digite o número de realizações desejado: 10

Digite o número de realizações máximo por tarefa (máximo de 2h por tarefa):
Thank you!
http://mccloud.codeplex.com