Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 110 (2017) 159–166

Procedia Computer Science

www.elsevier.com/locate/procedia

The 14th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2017)

Helping the Performance Evaluation of an Agent Run-time Framework: the SARL Experience Index

Stéphane Galland[a]*, Ansar-Ul-Haque Yasar[b], Elhadi Shakshuki[c], Nicolas Gaud[a]

[a]*Multiagent Group, LE2I, Univ. Bourgogne Franche-Comté, UTBM, F-90010 Belfort, France*
[b]*Transportation Research Institute (IMOB), Hasselt University, 3590 Diepenbeek, Belgium*
[c]*Jodrey School of Computer Science, Acadia University, Wolfville, NS, B4P 2R6, Canada*

Abstract

The performance evaluation of an agent platform is central in the agent-based modeling field. The agent platform and the hardware as well as the operating system modules, including any virtual machines, influence this performance. The impact of these hardware and operating system modules should be understood and evaluated due to their high impact on the global performances. In order to evaluate these two components, this paper proposes the scoring approach, named SARL Experience Index. The score is based on CPU, memory and disk sub-scores. We advocate this score may be helpful for the agent platform users for determining quickly the positive or negative impacts of a new deployment platform for an agent framework.

Peer-review under responsibility of the Conference Program Chairs.

Keywords: Agent platform, Evaluation, Platform Independent, SARL

1. Introduction

Modeling the interaction between individual agents becomes progressively important in recent research. Traditional modeling tools have difficulties for handling the complexity of communication, negotiation and coordination that are required in carpooling simulations. A method that is more suited for the interaction of

* Corresponding author. Tel.: +33 384 583 418.
E-mail address: stephane.galland@utbm.fr, stephane.galland@ubfc.fr

1877-0509
Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2017.06.073

autonomous entities is Agent-Based Modeling (ABM). ABM is essentially decentralized and individual-centric approach, which allows one to understand the interactions of physical particles, and describe many problems of astronomy, biology, ecology and social sciences. ABM is used in a wide range of applications, such as mobile environments and adaptable control entities in large complex networks. Regardless of the type of application, every agent needs some basic functionality. The agent platform (or framework) is the distributed middleware providing common services required by ABM applications, such as communication, security, agent life cycle management and directory services.

When the activity and the number of agents deployed on the platform increases, the pressure and the need to distribute the shared resources is also increases while keeping up the performance. Therefore, comparisons of these aspects of agent platforms are necessary. This comparison depends on the agent platform and the hardware and software systems on which the agent platform is running. Assuming a single-agent platform is used, the following two questions arise. 1) *What is the impact of the hardware and software components (outside the agent framework) on the general performances of the agent platform,* and 2) *How this impact is evaluated***.** In this work, we provide an indicator, named SARL Experience Index that helps the agent developers to compare the hardware and software platforms on which the agent platform may be run.

The SARL Experience Index (SEI) is inspired by the Windows Experience Index. It measures the capability of your computer's hardware and software configuration, including the Java Virtual Machine, and expresses this measurement to a number called a base score. A higher base score generally means that your computer will perform better and faster than a computer with a lower base score. This index is introduced to help the SARL users and developers to compare the performances of the platform with different hardware configurations.

Section 2 describes Windows Experience Index, which is the source of inspiration for SEI. Section 3 details the computation of the SARL Experience Index. Section 4 provides a use case based on the Janus agent platform. Section 5 concludes and provides perspectives to this work.

2. Experience Index for Desktop Applications

The Windows System Assessment Tool (WinSAT) is a module of Microsoft Windows Vista, Windows 7, Windows 8 and Windows 10. It measures various performance characteristics and capabilities of the hardware it is running on and reports them as a Windows Experience Index (WEI) score.

The WEI includes the following five sub-scores: 1) processor, 2) memory, 3) 2D graphics, 4) 3D graphics, and 5) disk. The base score is equal to the lowest of the sub-scores and is not an average of the sub-scores[1,2]. WinSAT reports WEI scores on a scale from 1.0 to 5.9 for Windows Vista, 7.9 for Windows 7[3], and 9.9 for Windows 8 and Windows 10[6].

The WEI enables users to match their computer hardware performance with the performance requirements of the software. For example, the Aero graphical user interface will not automatically be enabled unless the system has a WEI score of 3 or higher[5].

The WEI can also be used to show which part of the system would be expected to provide the greatest increase in performance when upgraded. For example, a computer with the lowest sub-score being its memory would benefit more from a RAM upgrade than adding a faster hard drive (or any other component[2]).

3. Experience Index for Agent Applications

In this section, the computation of the SARL[6] Experience Index is described in detail. The SEI is basically a score named the base score. This base score is computed from sub-scores related to the following sub-components: central processor unit (CPU), memory, and disk. These components are selected because they are fundamental to the execution of agents. Graphical capabilities for the system are discarded. We believe they are outside the scope of an agent platform, and may be measured by the WinSAT, for example.

3.1. Base Score

The base score is a value between 1.0 and 5.9. The base score is decomposed to the sub-scores, which are associated to each sub-component that is relevant for the evaluation of the agent platform performances. In Algorithm 1, the base score is computed from the CPU, memory and disk sub-scores. Their sub-scores are considered because they usually have a high impact upon the execution performance of any agent platform.

Algorithm 1. Global Score Computation, using the SARL syntax[†].

```
1.  def globalScore : float {
2.    val scores = #[
3.      cpuWeight * cpuScore(),
4.      memoryWeight * memoryScore(),
5.      diskWeight * diskScore()
6.    ]
7.    return average(normalize(scores))
8.  }
9.  def normalize(scores : float*) {
10.   for (var i = 0; i < scores.length; i = i + 1) {
11.     var s = scores.get(i)
12.     for (var j = 0; j < scores.length; j = j + 1) {
13.       if (j !== i) {
14.         var score = scores.get(j)
15.         if (s >= (score - TOLERANCE) && s <= score) s = score
16.       }
17.     }
18.   }
19.   scores.set(i, s)
20. }
```

The model logic is tolerant of one sub-score being below a threshold named the tolerance, which has by default the value 0.1. For example, assume that the memory score was 4.0 and the processor score 3.9. This would mean that the processor score would marginally be the only item keeping the base score below level 4. The model that addresses this issue by rounding up a single value that is below the next round level by the tolerance amount. This normalization is detailed at line 9 of Algorithm 1.

After the sub-scores are normalized, the global base score is the weighted average score of the sub-components. The different weights associated to the scores depend upon the ABM application. For example, in many applications, there is no disk access from the agents. In this case, the weight of the disk score may be lowered. By default, all the weights are equal to 1.

3.2. Central Processor Unit Score

The CPU score is created to measure the processor performances when tasked with the common usage activities of an agent platform. The processor is assessed based on the following items:

a) Compression and decompression using the Zip compression algorithm;
b) Encryption and decryption assessment;
c) Arithmetic operations;
d) Parallel execution or multiprocessor capabilities.

[†] SARL Agent programming language[8]: http://www.sarl.io

The results are normalized in order to have a 10^{-1} floating-point number precision and weight-averaged to obtain the final CPU sub-score. Algorithm 2 describes in detail the computation of the CPU sub-score based on the previously mentioned items.

Algorithm 2. CPU Score Computation, using the SARL syntax.

```
1.  def cpuScore : float {
2.    var scores = #[
3.      computeCompressionScore(), computeEncryptionScore(),
4.      computeArithmeticScore(), computeMultiProcessorScore()
5.    ]
6.    return round(average(scores), 10)
7.  }
```

The compression sub-score is selected because it requires a high number of computational resources in order to be proceeded. This sub-score is based upon the duration of the compression and decompression of an array of 6,000 Kb. This size is selected because it needs usually three seconds maximum to be processed on a modern computer. The duration of the compression and decompression is normalized and clamped in order to fit the [1, 6) range. Algorithm 3 describes the compression of the sub-score computation, using the SARL syntax.

Algorithm 3. Compression Sub-Score Computation, using the SARL syntax.

```
1.  def computeCompressionScore : float {
2.    var buffer = randomBytes( 6000 * 1024 )
3.    var zos = new ZipOutputStream(new ByteArrayOutputStream)
4.    var s = nanoTime
5.    zos.putNextEntry(new ZipEntry("test.bin"))
6.    zos.write(buffer, 0, buffer.length)
7.    var e = nanoTime
8.    return normalize(e - s, 3.secs)
9.  }
```

Encryption and decryption tasks are usually executed when the agent platform writes on and read data from a computer network in order to interact with another instance of the agent platform. Usually, the encryption and decryption algorithms are considered as huge computational activities. For this reason, they are included as sub-component of the CPU score. An array of 6,000 Kb is created and randomly filled. This array is encrypted with the standard MD5 and SHA1 algorithms. These two algorithms are non-reversible, so that the decryption is not possible. The reversible DES algorithm is used for encrypting and decrypting the byte array. As illustrated in Algorithm 4, the encryption score is the normalization of the duration of the execution of the three-encryption algorithms.

Algorithm 4. Encryption Sub-Score Computation, using the SARL syntax.

```
1.  def computeEncryptionScore : float {
2.    var buffer = randomBytes( 6000 * 1024 )
3.    var seed = this.userName+this.hostName+this.javaBersion+this.osName+this.osVersion
4.    var original = md5(seed).bytes
5.    var bkey = randomByteArray( DESKeySpec::DES_KEY_LEN )
6.    var kkey = new SecretKeySpec(bkey, "DES")
7.    var cipher = Cipher::getInstance("DES")
8.    var s = System::nanoTime
9.    MessageDigest::getInstance("MD5").digest(buffer)
10.   MessageDigest::getInstance("SHA").digest(buffer)
11.   cipher.init(Cipher::ENCRYPT_MODE, kkey)
12.   var output = cipher.doFinal(buffer)
13.   cipher.init(Cipher::DECRYPT_MODE, kkey)
14.   cipher.doFinal(output)
15.   var e = nanoTime
16.   return normalize(e - s, 4.secs)
17. }
```

Arithmetic operations are at the core of the agent-based models. Estimating the performances of the hardware and software layers regarding the arithmetic computations cannot be avoided. It is part of the CPU score as a sub-component. As illustrated by Algorithm 5, we assumed 100 million calls to the arctan operation to enable the estimation of the arithmetic performances of the system, with a maximal computation time of 30 seconds on a modern computer.

Algorithm 5. Arithmetic Sub-Score Computation, using the SARL syntax.

```
1.  def computeArithmeticScore : float {
2.    var s = nanoTime
3.    for (var i = 0; i < 100000000; i = i + 1) atan2(123, 456)
4.    var e = nanoTime
5.    return normalize(e - s, 30.secs)
6.  }
```

Algorithm 6 provides the score related to the number of processors within the system. In the provided function, it is assumed that the maximal number of processors within the system is 16.

Algorithm 6. Multiprocessor Sub-Score Computation, using the SARL syntax.

```
1.  def computeMultiProcessorScore : float {
2.    var n = Runtime::runtime.availableProcessors
3.    return normalize(n + 1, 16)
4.  }
```

3.3. Memory Score

The memory score measures the bandwidth of moving data into and out of the central memory in megabytes per second, such that the higher the bandwidth is, the better the memory is. We believe that not having enough memory is a limiting factor to the agent framework performance. The measurement of the memory performances is achieved by assessing the following:

a) The access time to memory variable;
b) The duration of the object construction statement.

As illustrated by Algorithm 7, the first criterion is evaluated by creating a byte array of 10 Mb, and accessing to all the array components. The second point is based on 10,000 calls to the string-of-character constructor.

Algorithm 7. Memory Score Computation.

```
1.  def memoryScore : float {
2.    var s = nanoTime
3.    var tab = new byte[10.megaBytes]
4.    for (var i = 0; i < tab.length; i = i + 1) {
5.      tab.set(i, 123)
6.      if (i > 0) var t = tab.get(i - 1)
7.      if (i < (tab.length - 1)) var t = tab.get(i + 1)
8.    }
9.    for (var i = 0; i < 10000; i = i + 1) new String("ABSD")
10.   var e = System::nanoTime
11.   return normalize(e - s, 1.secs)
12. }
```

As a result, the amount of memory in the system constrains the score value. The amounts of the memory limits are given in Table 1. Indeed, the amount of available memory in the system, e.g. in the Java virtual machine, has an impact upon the run-time performance of the memory garbage collector, such that lower the amount of available memory is, higher the run-time spent by the garbage collector to do its activities.

Table 1. Standard Limits to Memory Scores.

Amount of memory	Highest possible score
64 MB or less	1.0
Less than 128 MB	2.0
Less than 256 MB	3.0
Less than 512 MB	4.0
Less than 1 GB	5.0

3.4. Disk Score

The disk score measures the disk bandwidth (in megabytes per second). The conversion to an index number is set in such away that all modern disks will score at least 2.0. Algorithm 8 provides the function that is executed for computing the disk score.

Algorithm 8. Disk Score Computation.

```
1.  def diskScore : float {
2.    var tempFile = File::createTempFile("SEI", ".bin")
3.    var s = nanoTime
4.    var fw = new FileWriter(tempFile)
5.    for (var i = 0; i < 1024 * 1024 * 20; i = i + 1) fw.write('A')
6.    fw.flush
7.    var fr = new FileReader(tempFile)
8.    while (fr.read != -1) ;
9.    var e = nanoTime
10.   return normalize(e - s, 5.secs)
11. }
```

4. SEI Evaluation and Comparison with the Janus Platform Performances

To illustrate the use of the scoring mechanisms proposed in this paper, two computers are evaluated with the SEI. Additionally, the major components of the Janus platform (http://www.janusproject.io) are executed and their performances are evaluated. The Janus platform is the run-time environment that supports agent applications, which are written with the SARL agent programming language[8] (http://www.sarl.io).

The first computer is a Windows computer with the following configuration:

- Processor: Intel Core i7 CPU 960 @ 3.20GHz, 4 cores (Windows Experience Index: 7.5)
- Memory: 12 Gb (Windows Experience Index: 7.5)
- Operating System: Windows Seven (64 bits)
- Java Runtime Environment: 1.6.0_31-b05
- Java command line options: -Xmx1024m

The second computer is a Linux computer with the following configuration:

- Processor: Intel Core 2 Duo CPU P8700 @ 2.53GHz
- Memory: 4 Gb
- Operating System: Ubuntu Linux 11.10 (64 bits)
- Java Runtime Environment: 1.6.0_26-b03
- Java command line options: -Xmx1024m

Table 2 provides the SEI evaluations for these two computers. Table 2 also provides the performances of the Janus platform related to the emitting of messages.

Table 2. SEI Evaluation and the Janus Platform Performances.

Evaluation Item	Windows Computer	Linux Computer
SEI Base Score:	4.4	4.0
- CPU Sub-score	4.8	3.4
- Memory Sub-score	5.0	5.0
- Disk Sub-score	3.5	3.7
Janus Platform:		
- Atomic message sending	1,728 ns ± 14	6,446 ns ± 128
- Message sending with threaded agent	182,437 msg/s	124,275 msg/s
- Atomic message broadcasting	124,075 ns ± 2,842	116,986 ns ± 1,500
- Message broadcasting with threaded agent	6,855 msg/s	5,610 msg/s

The atomic message sending and broadcasting lines correspond to the evaluation of the message routing module of the Janus platform from one agent to other agents. The message sending is a one-to-one agent communication, and broadcasting is a one-to-many agent communications. The number of messages that could be sent by a threaded agent is evaluated from the sending function call in the sending agent to the mailbox addition function for the receiving agent.

According to the derived results during our first experiment, a correlation between the SEI base score and the real performances of the agent platform exists. We advocate that the SEI is helpful to the agent platform users for determining quickly if a given hardware and software platform (operating system and virtual machine) increases or decreases the global performances of the agent application.

5. Conclusion and Perspectives

In this paper, we considered the problem of performance evaluation of an agent platform. The agent platform and the hardware as well as the operating system modules, including any virtual machines, influenced this performance. Therefore, the question related to the impact of the hardware and operating system modules arose. In this paper, we proposed a simple and easy-to-use scoring approach in order to evaluate these components that are independent of the agent platform. A score, named the SARL Experience Index or SEI, based on the CPU, memory and disk sub-scores, is computed for evaluating and qualifying the performances of the computer in the context of the execution of an agent application. We advocated this scoring approach helps the agent platform users for determining quickly the positive or negative impacts of a new deployment environment for an agent platform. SEI is included into the official development environment of the SARL agent programming language (http://www.sarl.io), and its code may be found on Github[‡].

The determination of the correlation factor between the general performance of an agent platform and the SEI is a perspective in this work. More types of computer configurations will be included in the next test cases. Moreover, the correlation between the SEI and other agent platforms, e.g. GAMA[7] will be studied.

References

1. Softpedia. “Windows 7 WEI Scores 6.0 through 7.9 Explained”. Online: http://news.softpedia.com/news/Windows-7-WEI-Scores-6-0-through-7-9-Explained-172277.shtml. Retrieved May 21, 2017.
2. Microsoft. “Earning the top Windows Experience Index score”. Online: http://windows.microsoft.com/en-gb/windows7/achieving-a-perfect-windows-experience-index-score-in-windows-7. Retrieved May 21, 2017.
3. Marco Chiappetta. “How to Max Out Your Windows Performance for $1000”. PC World. September, 2011.
4. MSDN. “WinSAT Comprehensive”. Online: https://msdn.microsoft.com/en-us/library/windows/hardware/hh825488.aspx. Retrieved May 21, 2017.
5. Microsoft. “Full screen previews have got disabled. How do I re-enable them?”. Online: http://answers.microsoft.com/en-us/windows/forum/windows_7-desktop/full-screen-previews-have-got-disabled-how-do-i-re/d44193c0-7e51-4a2f-8ec0-4ac115c710c0. Retrieved May 21, 2017.
6. Rodriguez, S., Gaud, N., & Galland, S. (2014). SARL: a general-purpose agent-oriented programming language. In The 2014 IEEE/WIC/ACM International Conference on Intelligent Agent Technology. Warsaw, Poland: IEEE Computer Society Press.
7. Grignard, A., Taillandier, P., Gaudou, B., Vo, D-A., Huynh, N-Q., Drogoul, A. (2013), GAMA 1.6: Advancing the Art of Complex Agent-Based Modeling and Simulation. In PRIMA 2013: Principles and Practice of Multi-Agent Systems', Lecture Notes in Computer Science, Vol. 8291, Springer, pp. 117-131.

‡ https://github.com/sarl/sarl/tree/master/contribs/io.sarl.experienceindex/io.sarl.experienceindex.plugin