Conference Paper

Empirical Study Between Compiled, Interpreted, and Dynamic Programming Languages Applying Stable Ordering Algorithms (Case Study: Java, Python, Jython, Jpype and Py4J)

Estudio empírico comparativo entre lenguajes de programación compilados, interpretados y dinámicos aplicando algoritmos de ordenamiento estables (Caso de estudio: Java, Python, Jython, Jpype y Py4J)

Milton Labanda-Jaramillo[1], José Guamán Quinche[2], Luis Chamba-Eras[1], Edison Coronel-Romero[1], Jose-Luis Granda[1], Lorena Elizabeth Conde Zhingre[3,4]

[1]Grupo de Investigación en Tecnologías de la Información y Comunicación (GITIC), Carrera de Ingeniería en Sistemas, Facultad de Energía, Universidad Nacional de Loja, Av. Pío Jaramillo Alvarado, Loja, Ecuador
[2]Carrera de Ingeniería en Sistemas, Facultad de Energía, Universidad Nacional de Loja, Av. Pío Jaramillo Alvarado, Loja, Ecuador
[3]Universidad Internacional Del Ecuador, Escuela de Informática y Multimedia, Titulación Ingeniería en Tecnologías de la Información, Quito, Ecuador
[4]Instituto Tecnológico Superior Daniel Álvarez Burneo, Carrera de Análisis de Sistemas, Loja, Ecuador

Corresponding Author:
Milton Labanda-Jaramillo
miltonlab@unl.edu.ec

Received: 4 December 2018
Accepted: 5 December 2018
Published: 27 December 2018

Publishing services provided by Knowledge E

Selection and Peer-review under the responsibility of the SIIPRIN-CITEGC Conference Committee.

Abstract

This article allows to investigate benchmark between programming languages, with the objective of identifying the performance between the execution time and the memory use between the Java and Python languages, as well as, in three implementations of dynamic languages that combine the two aforementioned languages: Jython, Jpype, Py4J. According to the results, it is concluded that the language that obtains the best performance is Py4J.

Resumen

El presente trabajo permite investigar sobre pruebas de rendimiento entre lenguajes de programación, con el objetivo de identificar el rendimiento entre al tiempo de ejecución y el uso de memoria entre los lenguajes Java y Python, así como, en tres implementaciones de lenguajes dinámicos que combinan los dos lenguajes antes mencionados: Jython, Jpype, Py4J. De acuerdo a los resultados, se concluye que el lenguaje que obtiene el mejor rendimiento es Py4J.

OPEN ACCESS

Keywords: programming languages, benchmark, algorithms, compilers

Palabras clave: lenguajes de programación, prueba de rendimiento, algoritmos, compiladores

1. Introducción

Muchos son los aspectos que a primera vista un programador puede considerar al momento de elegir un lenguaje de programación, el presente trabajo pretende ser una guía de referencia que aporte datos al desempeño de los lenguajes de programación Java, Python, así como las implementaciones intermedias o dinámicas entre estos dos lenguajes con el fin de aportar información para la caracterización del rendimiento de estos lenguajes en cuanto a tiempos de ejecución y consumo de memoria. De la revisión bibliográfica previa que se ha realizado se ha detectado que existen dos tipos de trabajos relacionados con la temática: aquellos trabajos sobre lenguajes de programación indistintamente sean compilados o interpretados y aquellos trabajos que comparan Java con lenguajes interpretados y hospedados sobe JVMs.

Respecto del primer grupo de trabajos existen publicados diversos estudios, entre ellos tenemos el trabajo de tesis de Goyal [1] en el cual se presenta un estudio comparativo de los lenguajes de programación C, Java, C# y Jython en relación a los criterios de consumo de memoria, utilización de CPU y tiempo de ejecución en un escenario distribuido. Es común que el estudio del rendimiento de los lenguajes de programación se los delimite de acuerdo al campo de aplicación en la ciencia, así dentro de la computación científica, se ha realizado recientemente un estudio [2] con el objetivo encontrar la idoneidad de un subconjunto de lenguajes de programación desde el punto de vista de cálculo numérico, enfocado en algunos lenguajes populares como FORTRAN, C ++, Python, Java, MATLAB, Mathematica y Julia. En [3] se determina que Python es un lenguaje poseedor de un completo conjunto de características: compacidad, estructuras de datos de alto nivel, portabilidad automática, desarrollo tipo y ejecución, recolección de basura, comprobación de errores en tiempo de ejecución, tipado dinámico y extensibilidad; las mismas que ninguno de los lenguajes como Fortran, C, C++, Java ni Perl las poseen en conjunto razones por las cuales combinar Python con otros lenguajes da como resultado desarrollos y ejecuciones más rápidas [3]. Una

comparación empírica entre un conjunto de 7 lenguajes de programación agrupados en lenguajes de scripting y lenguajes de no-scripting se realiza a través del extenso trabajo de Prechelt [4] en donde se realizan 80 implementaciones de un mismo reto de programación participando 74 programadores diferentes y midiendo el tiempo de ejecución junto con el consumo de memoria para cada lenguaje, en contraposición a este estudio, en [5] en lugar de insistir en las diferencias entre los lenguajes tipados dinámicamente y estáticos, se propone buscar una integración del aspecto estático y dinámico en el mismo lenguaje de programación tipeo estático donde sea posible y digitación dinámica cuando sea necesario.

El área de la educación no está exenta de estudios sobre de evaluación de características y desempeños los lenguajes de programación, así por ejemplo en [6] se revelan resultados que demuestran que el uso de un lenguaje sintácticamente simple (Python) en lugar de uno más complejo (Java) facilita el aprendizaje de los conceptos de programación por parte de los estudiantes.

Si bien existen alternativas para evaluar el rendimiento de varias plataformas o lenguajes de programación "hasta ahora, ningún conjunto cohesivo de métricas ha cubierto adecuadamente las propiedades que varían sistemáticamente entre las cargas de trabajo de plataformas Java y las que no son Java" [7], sin embargo en [8] se presenta una propuesta de mejora y optimización a algunos problemas identificados con respecto a intérpretes alojados en Máquinas Virtuales Java, la misma que se evalúa con tres intérpretes: Jruby (Ruby), Jython (Python), y Rhino (Java Script). De manera similar en [9] se caracteriza el comportamiento dinámico de los lenguajes interpretados Clojure, Python y Ruby sobre la Máquina Virtual de Java discutiendo brevemente sus implicaciones. Frente a la necesidad de evaluar casos de uso de lenguajes Java y aquellos que no son Java existen propuestas de desarrollo de infraestructuras de herramientas como la presentada en [7] para poder como medir el desempeño en diferentes escenarios. En [10] se realiza un extenso estudio con 75 métricas para determinar el comportamiento de un conjunto de programas escritos en Java, Clojure, JRuby, Jython y Scala determinándose a través de técnicas de análisis exploratorio de datos que los comportamientos observados pueden variar de acuerdo a la madurez de los lenguajes interpretados sobre Máquinas Virtuales de Java, así como las nuevas características agregadas a dichas Máquinas Virtuales. Así como existen varios aportes para evaluar lenguajes dinámicos alojados sobre un a MVJ existe un trabajo de tesis de master [11] en donde se mide el rendimiento con respecto al tiempo de ejecución de los intérpretes Cpython, Cython, Jython y Pypi a través de un banco de pruebas con problemas de tamaño variable creados por el autor.

Luego de haber realizado la respectiva revisión bibliográfica se determinó que no existen estudios o trabajos de comparación o evaluación de lenguajes de programación que involucren lenguajes estáticos, dinámicos e híbridos o combinados. La intención del presente trabajo es realizar un estudio empírico de comparación entre los lenguajes y/o plataformas de programación Python, Java, Jython y Jpype para responder a la pregunta de investigación *¿Cuál es del desempeño de lenguajes compilados vs interpretados vs híbridos respecto del tiempo de ejecución y del consumo de memoria?*

2. Revisión de literatura

2.1. Python

Python es un lenguaje de programación que soporta paradigmas de programación imperativos, funcionales y orientados a objetos con enfoque en la calidad del software, la sintaxis simple y la productividad del desarrollador, es además un lenguaje interpretado de forma dinámica, lo que significa que no es necesario declarar un tipo de variables en el código y que no es necesario compilarlo previamente, a cambio un intérprete compila el código de Python en bytecode que luego lo ejecuta la Máquina Virtual de Python (PVM) [11], en resumen la estructura del entorno de ejecución de Python se resume en dos tareas: el código fuente de Python se compila a bytecode el cual es ejecutado por la Máquina virtual de Python.

2.2. Java y la Máquina Virtual

La introducción del soporte de scripting y el soporte para lenguajes dinámicamente tipados a la plataforma Java permite la creación de scripts en programas Java y simplifica el desarrollo de entornos de ejecución le lenguajes dinámicos, con lo cual los desarrolladores de literalmente cientos de lenguajes de programación eligen la Java Virtual Machine (JVM) o Máquina Virtual de Java (MVJ) como la plataforma anfitrión para sus lenguajes, tanto para evitar desarrollar un nuevo entorno de ejecución desde cero como para beneficiarse de la madurez, ubicuidad y rendimiento de la MVJ. En la actualidad, programas escritos en lenguajes dinámicos populares como Ruby, Python o Clojure (un dialecto de Lisp) se pueden ejecutar en la JVM, creando un ecosistema que aumenta la productividad de los desarrolladores [9].

A pesar de las bondades de la MVJ, dado que se concibió originalmente para un lenguaje de tipo estático, el rendimiento de la JVM y su compilador JIT con lenguajes

de tipado dinámico a menudo no produce los resultados deseados, es por ello que trabajos como el presente son de relevancia para buscar mejorar.

2.3. Jython

Jython es una implementación del lenguaje de programación Python sobre la Máquina Virtual de Java. Durante la ejecución de los programas Jython, el código fuente de Jython se traduce en bytecode de Java que se puede ejecutar en cualquier computadora que admita la máquina virtual de Java [12]. "Jython trae el poder del lenguaje Python a la MVJ. Proporciona a los desarrolladores Java la capacidad de escribir código productivo y dinámico utilizando una elegante sintaxis. Asimismo, permite a los desarrolladores de Python aprovechar la gran cantidad de bibliotecas y API útiles de Java que la JVM tiene para ofrecer" [13]. A pesar de las bondades de esta alternativa Jython no admite extensiones nativas escritas para CPython como NumPy o SciPy y debido a que la mayoría del código científico de Python depende fundamentalmente de dichas extensiones nativas directa o indirectamente, por lo general no se pueden ejecutar con Jython [14], es por ello que aparecen alternativas como JyNI (Jython Native Interface) que permite cargar extensiones de CPython (la implementación nativa de Python) nativas y acceder a ellas desde Jython como si se lo hiciera desde CPython.

2.4. La librería JPype

Otra alternativa candidata a cubrir las debilidades de Jython para acceder a código nativo es JPype. "JPype es un esfuerzo para permitir a los programas de Python el acceso completo a las bibliotecas de clases de Java. Esto se logra no mediante la reimplementación de Python, como en Jython/JPython, sino mediante la interconexión a nivel nativo en ambas Máquinas Virtuales" [15]. Técnicamente casi toda la ciencia de datos, el acceso a dispositivos de hardware, el acceso a interfaces nativas y más temas cuyo soporte no está completo en Jython se intentan resolver con JPype al usar los módulos típicos que a la final se ejecutan sobre CPython y de todas formas se accede a las funciones de Java vía JNI (Java Native Interface).

2.5. La librería Py4J

Py4J es un puente entre Python y Java, que permite que los programas de Python que se ejecutan en un intérprete de Python accedan dinámicamente a los objetos Java en

una máquina virtual Java [16]. "Py4j permite a Python acceder a objetos desde una instancia Java en ejecución y también permite a los programas Java llamar a scripts de Python. Lo hace creando una conexión TCP entre un objeto en el lado Java (llamado un GatewayServer) y otro en el lado de Python (llamado JavaGateway). La ventaja de usar esta tecnología es que se pueden usar todas las bibliotecas de Python existentes. Esto no sería posible si uno, por ejemplo, usa Jython, que es una implementación de Python que se ejecuta en la máquina virtual Java" [17].

Al ser Py4J un componente parte de Spark, es uno de los núcleos de tecnologías utilizadas de hoy en día en trabajos con gran demanda de procesamiento sobre aprendizaje de máquina distribuida [18], en personalización de herramientas relacionadas con semántica de datos como [19] o en análisis de datos con la utilización de Big Data [20].

3. Metodología y experimentación

Para dar respuesta a la pregunta de investigación se realizaron pruebas de experimentación, con el objetivo de medir el rendimiento y desempeño - registrar el tiempo de ejecución en segundos requerido por el intérprete o plataforma para completar la ordenación así como la cantidad de memoria ocupada por el proceso ejecutado en memoria principal - de los lenguajes de programación Java (1.8.0) y Python (3.5.2), así como, sus implementaciones intermedias entre esos lenguajes (Jython, Jpipe, Py4J). El proceso para la experimentación fue el siguiente: 1) elección de los algoritmos de ordenamiento a usar en los casos de pruebas unitarias: burbuja (buble Sort), por inserción (Insertion Sort) y por mezcla (Merge Sort); 2) implementación de los algoritmos - desarrollados en la "Huazhong University of Science & Technology (https://github.com/hustcc/JS-Sorting-Algorithm)" -, en los dos lenguajes de programación usando un conjunto de datos de 10000 números flotantes generados aleatoriamente; 3) recolección de datos en formato CSV sobre el tiempo de ejecución y del consumo de memoria.

Código previo e intermedio de los test para persistencia de los datos en los algoritmos de codificados en Java.

```
@Before
public void setUp() {
    sourceArray = SortsTest.array;
    expResult = SortsTest.orderedArray;
    t0 = System.currentTimeMillis();
    m0   =   Runtime.getRuntime().totalMemory()   -
     Runtime.getRuntime().freeMemory();
}
public void befforeAssert() {
    t1 = System.currentTimeMillis();
    m1   =   Runtime.getRuntime().totalMemory()   -
     Runtime.getRuntime().freeMemory();
    pw.printf(Locale.US,"%s, java, %s, %f, %s%n",
     new  Timestamp(t1),  this.method_name,  (t1-
     t0)/1000.0, (m1-m0));
}
```

Test para el algoritmo de ordenación burbuja codificado en Python.

```
def setUp(self):
      self.t0 = time()
      # get the residente set size memory
      self.m0 =
       psutil.Process(os.getpid()).memory_info().rss

  def beforeAssert(self):
      self.t1 = time()
      # get the resident set size memory
      self.m1 =
       psutil.Process(os.getpid()).memory_info().rss
      self.t = self.t1 - self.t0
      self.m = self.m1 - self.m0
      timestamp = datetime.today().isoformat()
      self.fw.write('{0},  python,  {1},  {2},  {3}
       \n'.format(timestamp,       self.method_name,
       self.t, self.m))
```

La descripción de las variables analizadas en las pruebas de benchmark se detalla en la Tabla 1.

4. Resultados y discusión

Se ejecutaron las pruebas con 1000 iteraciones en cada uno de los escenarios (Java, Python, Jython, Jpype, Py4J, ver Tabla 3), donde se demuestra que el uso de los lenguajes de programación Java y Python combinadas a través de las librerías o plugins, presenta variaciones tanto positivas como negativas que difieren de los resultados

Tabla 1: Ejemplo de recopilación de métricas sobre tiempo de ejecución y memoria principal ocupada.

Variable	Descripción
Marca de tiempo	Instante en el que se toma los datos
Lenguaje/Plataforma	Lenguaje o librería utilizados
Algoritmo	Uno de tres algoritmos de ordenación
Tiempo	Tiempo en segundos que demora la ejecución del algoritmo
Memoria	Cantidad de memoria que utiliza el proceso residente en la memoria principal

cuando se usan por separado. Dentro de los resultados obtenidos se puede apreciar la variación por cada lenguaje y cada algoritmo respecto de la media aritmética del tiempo y de la memoria consumidos (ver Tabla 2).

Tabla 2: Ejemplo de recopilación de métricas sobre tiempo de ejecución y memoria principal ocupada.

Lenguaje	Algoritmo	Tiempo (s)	Memoria (bytes)
java	bubblesort	0.268818	0
java	insertionsort	0.050218	0
java	mergesort	0.180907	28948131.096
jpype	bubblesort	318.226742513876	963633.595441595
jpype	insertionsort	0.0390298172958895	30801.6866096866
jpype	mergesort	0.279384418770l4	605192.752136752
jython	bubblesort	53.3371398810832	126606333.357143
jython	insertionsort	18.3358947394967	149497364.003972
jython	mergesort	0.362856154876587	27013689.1269841
py4j	bubblesort	0.21276976728439	65536
py4j	Insertionsort	0.0262142233848562	0
py4j	mergesort	0.078377597808843	0
python	bubblesort	14.9533103532885	622569.031775701
python	insertionsort	0.00385150664320593	77850.7962616822
python	mergesort	0.128628573685051	93235.6785046729

Para lograr una apreciación general se procedió a obtener la media general del tiempo y la memoria por cada lenguaje (ver Tabla 3). La Figura 1 muestra la relación (regresión lineal) entre las variables tiempo y memoria.

5. Conclusiones

En la implementación de los test y benchmarks con cada uno de los escenarios se determinó que una de las variables adicionales que puede incidir directamente en los resultados lo constituye el tipo concreto de estructuras de arreglos utilizados para

Tabla 3: Media aritmética del tiempo y memoria consumidos en cada lenguaje.

Lenguaje/Plataforma	Tiempo (s)	Memoria (bytes)
java	0.1666477	9649377.03
python	5.0285968	264551.84
jython	24.0138412	101023098.98
jpype	106.1817189	533209.34
py4j	0.1057872	21845.33

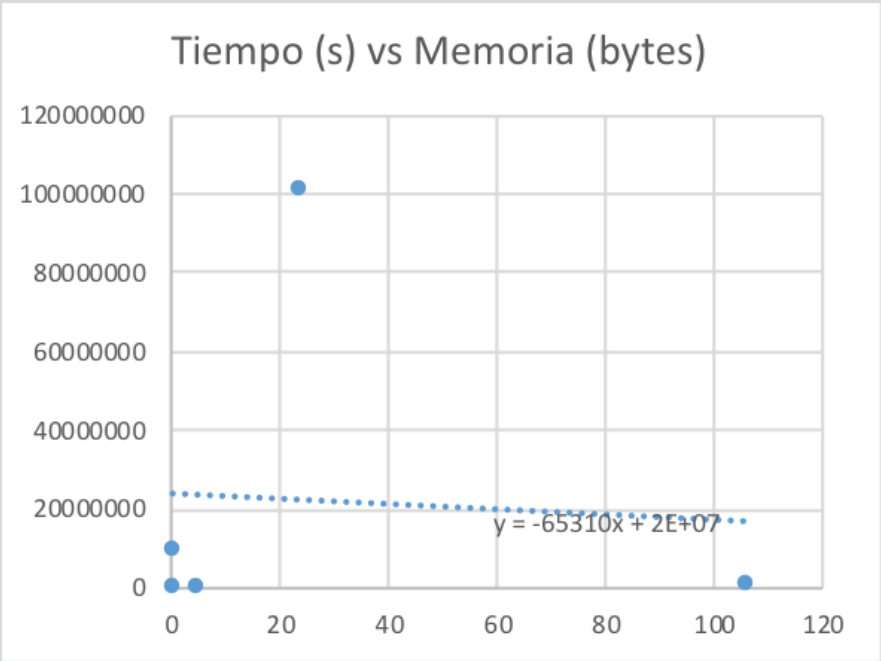

Figura 1: Diagrama de dispersión entre el tiempo utilizado en segundos y memoria ocupada en bytes.

manejar los arreglos de datos (ver Tabla 2). El mejor tiempo y memoria se obtiene al utilizar el esquema cliente servidor usado por la librería Py4J el cual implementó el servidor de métodos de búsqueda en Java y el cliente de pruebas en Python, permitiendo con ello contestar la pregunta de investigación. Como trabajos futuros se propone realizar un análisis de varianza ANOVA sobre las medias aritméticas obtenidas por cada escenario, o replicar la misma experimentación haciendo variaciones por ejemplo en el conjunto de datos a ordenar, o aumentando la cantidad de tests para realizar su ejecución en una infraestructura de computación de alto rendimiento.

Agradecimientos

Este trabajo forma parte de las actividades académicas-investigación del Grupo de Investigación GITIC, adscrito a la Carrera de Ingeniería en Sistemas, de la Facultad de Energía de la Universidad Nacional de Loja.

Referencias

[1] P. Goyal, "Comparative Study of C, Java, C # and Jython," University of North Florida, 2014.

[2] D. Singh, "An Empirical Study of Programming Languages from the Point of View of Scientific Computing," *Int. J. Innov. Sci. Eng. Technol*, vol. 4, no. 6, pp. 367–371, 2017.

[3] C. Seberino, "Python: Faster and Easier Software Development," *Annu. Conf. Calif. San Diego*, pp. 3–5, 2012.

[4] Prechelt, "An empirical comparison of c, c++, java, perl, python, rexx, and tcl for a search/string-processing program," 2000.

[5] E. Meijer and P. Drayton, "Static Typing Where Possible, Dynamic Typing When Needed: The End of the Cold War Between Programming Languages," 2004.

[6] T. Koulouri, S. Lauria, and R. D. Macredie, "Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches," *Trans. Comput. Educ.*, vol. 14, no. 4, p. 26:1–26:28, 2014.

[7] A. Sarimbekov, A. Sewe, and S. Kell, "A Comprehensive Toolchain for Workload Characterization Across JVM Languages," *Proc. 11th ...*, pp. 9–16, 2013.

[8] G. Savrun-Yeniçeri, W. Zhang, H. Zhang, E. Seckler, C. Li, S. Brunthaler, P. Larsen, and M. Franz, "Efficient hosted interpreters on the JVM," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 1, pp. 1–24, 2014.

[9] A. Sarimbekov, A. Podzimek, L. Bulej, Y. Zheng, N. Ricci, and W. Binder, "Characteristics of dynamic JVM languages," *Proc. 7th ACM Work. Virtual Mach. Intermed. Lang. - VMIL '13*, pp. 11–20, 2013.

[10] W. H. Li, D. R. White, and J. Singer, "JVM-hosted languages," *Proc. 2013 Int. Conf. Princ. Pract. Program. Java Platf. Virtual Mach. Lang. Tools - PPPJ '13*, pp. 101–112, 2013.

[11] A. Roghult, "Benchmarking Python Interpreters," KTH Royal Institute of Technology School, 2016.

[12] S. V. Chekanov, "Numeric Computation and Statistical Data Analysis on the Java Platform - Chapter," 2016.

[13] J. Juneau, J. Baker, V. Ng, L. Soto, and F. Wierzbicki, *The Definitive Guide to Jython*. Apress, 2010.

[14] S. Richthofer, "JyNI - Using native CPython-Extensions in Jython," *Proc. 6th EUR. CONF. PYTHON Sci. (EUROSCIPY 2013)*, no. Euroscipy, pp. 59–64, 2014.

[15] Jpype, "JPype - Java to Python integration." [Online]. Available: http://jpype.sourceforge.net/.

[16] B. Dagenais, "Py4j - a bridge between python and java." [Online]. Available: https: %0A//www.py4j.org/.

[17] K. Fuchsberger and Y. I. L. Cern, "PyMad – INTEGRATION OF MadX IN PYTHON," *Proc. IPAC2011, San Sebastián, Spain*, pp. 2289–2291, 2011.

[18] M. Sewak and S. Singh, "Optimal State Recommender Solution," *2016 Int. Conf. Internet Things Appl. Maharashtra Inst. Technol. Pune, India 22 Jan - 24 Jan, 2016*, pp. 101–106, 2016.

[19] H. O. Hatzel, B. J. Kunkel, E. J. Kunkel, Z. Prof, and T. Ludwig, "Extracting Semantic Relations from Wikipedia using at Fachbereich Informatik," Universitat Hamburg, 2017.

[20] M. E. Verjaga, "Análisis de datos y extracción de conocimiento utilizando BigData," Universidad de Jaén, 2018