Universidad de la Ciudad de Aguascalientes

Maestría en Ciencia de Datos (RVOE 2727)

Materia : Big Data & Data

4 Análisis de datos en Python.
- 4.1 Conceptos básicos de Python.
  - 4.1.1 ¿Qué es y cómo instalar Python?
  - 4.1.2 Ejecución de código en Jupyter
  - 4.1.3 Sintaxis
- 4.2 Análisis de datos en Python.
  - 4.2.1 Fuentes de datos
  - 4.2.2 Tipos de datos que se pueden recopilar
  - 4.2.3 Gestión, exploración y visualización efectiva de los datos

Ejemplo de aplicaciones del análisis de datos masivos en Python.

- **Análisis de negocio**: Analizar datos de ventas, marketing, finanzas y otras áreas de negocio para obtener información valiosa que ayude a tomar mejores decisiones.
- **Análisis financiero**: Analizar datos financieros, como precios de acciones, datos de mercado y carteras de inversión, para tomar decisiones de inversión más informadas.
- **Análisis de redes sociales:** Analizar datos de redes sociales, como tweets, publicaciones y comentarios, para comprender mejor las tendencias y el comportamiento del consumidor.
- **Análisis de atención médica**: Analizar datos de atención médica, como registros de pacientes, imágenes médicas y datos de ensayos clínicos, para mejorar la calidad de la atención y la investigación médica.
- **Ciencia e ingeniería**: Analizar datos científicos y de ingeniería, como datos de experimentos, simulaciones y modelos, para obtener nuevos conocimientos y desarrollar nuevas tecnologías.

¿Qué es python?

Python es un lenguaje de programación muy que se utiliza ampliamente en el análisis de datos masivos debido a su facilidad de uso, su amplia gama de bibliotecas especializadas y su capacidad para escalar a grandes conjuntos de datos.

Entorno de desarrollo: Herramientas como Anaconda, Jupyter, Google Colaboratory, que proporciona una distribución de Python preconfigurada con las bibliotecas necesarias para el análisis de datos, o es posible instalar Python y las bibliotecas individualmente utilizando pip, el administrador de paquetes de Python.

- **Variables y tipos de datos**: Familiarizarse con las variables y los diferentes tipos de datos que Python soporta, como números, cadenas, listas y diccionarios.
- **Operadores matemáticos y lógicos:** Entender los operadores básicos para realizar cálculos y comparaciones entre datos.
- **Estructuras de control**: Dominar el uso de condicionales (if, else) y bucles (for, while) para controlar el flujo de ejecución de un programa.
- **Funciones:** Aprender a crear y usar funciones para organizar el código y hacerlo más reutilizable.

Manipulación y limpieza de datos

- **Lectura de datos**: Aprender a leer datos desde diferentes fuentes como archivos de disintos formatos, bases de datos y APIs.
- **Limpieza de datos**: Identificar y corregir errores, valores inconsistentes y valores faltantes en los datos.
- **Transformación de datos**: Transformar los datos a un formato adecuado para el análisis, como normalizar las variables y crear nuevas variables a partir de las existentes.

- **Manipulación de datos**: Python tiene varios componentes para trabajar con datos, como Pandas y NumPy. Pandas es especialmente útil para la manipulación y análisis de datos estructurados, mientras que NumPy es excelente para operaciones numéricas eficientes.

- **Procesamiento distribuido**: Cuando los conjuntos de datos son grandes para caber en la memoria de una sola máquina, se necesita procesamiento distribuido, a lo cual existen frameworks como Apache Spark, Dask o Ray, que te permiten distribuir la carga de trabajo en múltiples nodos o procesadores.

- **Visualización de datos**: Es importante poder visualizar los datos para entenderlos mejor y comunicar los resultados de manera efectiva. Para esto, existen componentes como Matplotlib, Seaborn o Plotly para crear gráficos y visualizaciones interactivas.

- **Apache Spark**: Un framework de procesamiento distribuido que proporciona APIs en Python (PySpark), Java, Scala y SQL para procesar grandes conjuntos de datos en clústeres de computadoras.

- **Dask**: Una biblioteca de paralelización en Python que escala desde una sola máquina hasta clústeres de tamaño considerable. Se puede integrar bien con las bibliotecas de análisis de datos de Python, como Pandas y NumPy.

- **Ray**: Un framework de computación distribuida de código abierto para aplicaciones de aprendizaje automático y procesamiento de datos en Python.

Almacenamiento de datos: Para trabajar con grandes volúmenes de datos, es pertinente utilizar un sistema de almacenamiento adecuado, para implementar gestores de datos como MongoDB, PostgreSQL o Apache Cassandra, o sistemas de archivos distribuidos como Hadoop Distributed File System (HDFS) o Protocolos como S3.

Paralelismo y concurrencia: Para trabajar con grandes volúmenes de datos, es importante aprovechar al máximo los recursos de cómputo disponibles. Python tiene herramientas como asyncio y threading para programación concurrente, así como herramientas externas como multiprocessing para ejecutar tareas en paralelo.

Optimización de código: A medida que trabajas con conjuntos de datos más grandes, es importante escribir código eficiente para minimizar el tiempo de procesamiento. Esto implica conocer técnicas de optimización de código en Python, como vectorización, uso de estructuras de datos eficientes y evitar bucles innecesarios.

Entorno para utilizar Python

Jupyter es una aplicación web de código abierto que te permite crear y compartir documentos interactivos que contienen código, visualizaciones y texto explicativo. La palabra "Jupyter" es un acrónimo que combina los nombres de tres lenguajes de programación populares: Julia, Python y R, que son los principales lenguajes que Jupyter admite inicialmente.

Los documentos de Jupyter se denominan "cuadernos" (notebooks en inglés) y pueden contener celdas de código ejecutable, texto enriquecido, imágenes, enlaces y otros elementos multimedia. Esto hace que Jupyter sea una herramienta muy útil para el análisis de datos, ya que te permite escribir y ejecutar código Python (u otros lenguajes de programación) en el contexto de un documento interactivo.

La relación de Jupyter con el análisis de grandes volúmenes de datos con Python es significativa por varias razones:

- **Interactividad:** Jupyter proporciona un entorno interactivo donde puedes escribir y ejecutar código Python en tiempo real. Esto es útil para explorar y analizar datos de manera interactiva, ya que puedes ejecutar fragmentos de código y ver los resultados de inmediato.
- **Visualización de datos**: Jupyter es compatible con bibliotecas de visualización de datos populares como Matplotlib, Seaborn y Plotly, lo que te permite crear gráficos y visualizaciones interactivas directamente en tus cuadernos. Esto facilita la exploración y la comunicación de tus resultados de análisis de datos.
- **Documentación**: Los cuadernos de Jupyter te permiten combinar código, visualizaciones y texto explicativo en un solo documento. Esto es útil para documentar tus análisis de datos, compartir tus resultados con otros y crear informes técnicos o tutoriales.
- **Soporte para diferentes lenguajes**: Aunque Python es el lenguaje más comúnmente utilizado en los cuadernos de Jupyter, también es compatible con otros lenguajes de programación como R, Julia, Scala y muchos más. Esto te permite realizar análisis de datos utilizando una variedad de herramientas y bibliotecas en el mismo entorno.

Ventajas de usar Jupyter para el análisis de datos

- **Interactividad:** Permite explorar y analizar datos de forma interactiva, ejecutando código y visualizando resultados de forma inmediata.

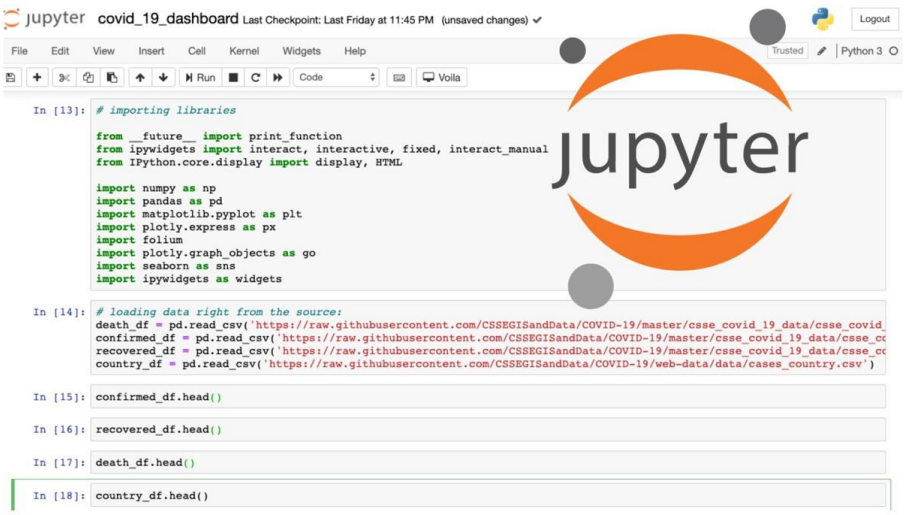
- **Facilidad de uso**: Es una herramienta intuitiva y fácil de aprender, incluso para principiantes en Python.

- **Reproducibilidad:** Los notebooks de Jupyter son documentos que contienen código, resultados y explicaciones, lo que permite compartir y reproducir análisis de forma sencilla.

- **Flexibilidad**: Permite combinar código, texto, imágenes y visualizaciones en un solo documento, lo que facilita la creación de informes y presentaciones.

Ejemplos de uso de Jupyter para el análisis de datos

- **Limpieza y transformación de datos**: Usar Jupyter para cargar datos, limpiarlos, transformarlos y prepararlos para el análisis.
- **Análisis estadístico**: Realizar análisis estadísticos descriptivos e inferenciales utilizando las bibliotecas de Python.
- **Visualización de datos**: Crear gráficos y visualizaciones para explorar los datos y comunicar tus hallazgos.
- **Aprendizaje automático**: Usar Jupyter para desarrollar modelos de aprendizaje automático para predecir resultados o clasificar datos.



```
import pandas as pd

# Cargar datos del INEGI
datos_inegi = pd.read_csv("https://www.inegi.org.mx/", compression="zip")

# Seleccionar variables de interés
datos_inegi = datos_inegi[["Entidad", "Sexo", "Poblacion_total"]]

# Calcular la población total por sexo y entidad
poblacion_total = datos_inegi.groupby(["Entidad",
"Sexo"])["Poblacion_total"].sum()

# Visualizar la población total por sexo en un gráfico de barras

import matplotlib.pyplot as plt plt.bar(datos_inegi["Entidad"],
poblacion_total)
plt.xlabel("Entidad")
plt.ylabel("Población total")
plt.show()
```

En Python, hay varios tipos de datos que son comúnmente utilizados para el análisis de datos.

- Números enteros (int): Representan números enteros, como 1, 2, -5, entre otros
- Números de punto flotante (float): Representan números decimales, como 3.14, -0.001, entre otros
- Cadenas de caracteres (str): Representan texto, como "hola mundo", "Python es genial", entre otros
- Listas (list): Son colecciones ordenadas de elementos que pueden ser de diferentes tipos de datos. Por ejemplo, [1, 2, 3], ["a", "b", "c"], entre otros
- Tuplas (tuple): Son similares a las listas, pero son inmutables, es decir, no se pueden modificar una vez creadas. Por ejemplo, (1, 2, 3), ("a", "b", "c"), entre otros
- Diccionarios (dict): Son colecciones de pares clave-valor, donde cada valor está asociado con una clave única. Por ejemplo, {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}, entre otros
- Conjuntos (set): Son colecciones no ordenadas de elementos únicos. Por ejemplo, {1, 2, 3}, {"a", "b", "c"}, entre otros
- Booleanos (bool): Representan valores de verdad, es decir, verdadero (True) o falso (False).

Existen numerosas fuentes de datos masivos que pueden ser analizadas con Python para extraer información valiosa.

- **Redes Sociales**: Plataformas como X-Twitter, Facebook, LinkedIn y Reddit generan enormes cantidades de datos en forma de publicaciones, comentarios, conexiones de red, entre otros. Estos datos pueden ser analizados para comprender tendencias, opiniones de usuarios, patrones de comportamiento y más.
- **Sensores IoT (Internet de las Cosas)**: Dispositivos IoT como sensores ambientales, medidores inteligentes, cámaras de vigilancia, entre otros, generan grandes volúmenes de datos en tiempo real. Estos datos pueden utilizarse para monitorear y predecir eventos, optimizar procesos industriales, mejorar la eficiencia energética y mucho más.
- **Aplicaciones y Plataformas Web:** Las aplicaciones y plataformas web recopilan datos de usuarios, como registros de acceso, clics, compras, búsquedas, entre otros. Estos datos pueden ser analizados para comprender el comportamiento del usuario, optimizar la experiencia del usuario, realizar análisis de mercado y más.

- **Datos Geoespaciales:** Datos geoespaciales, como imágenes de satélite, datos de GPS, mapas digitales, entre otros, pueden ser analizados para realizar análisis de ubicación, detección de cambios en el medio ambiente, planificación urbana, navegación y más.
- **Datos de Transacciones Financieras:** Los datos de transacciones financieras, como transacciones bancarias, transacciones con tarjetas de crédito, datos bursátiles, entre otros, pueden ser analizados para detectar fraudes, predecir tendencias del mercado, realizar análisis de riesgos, entre otros.
- **Datos de Salud**: Los datos de salud, que incluyen registros médicos electrónicos, datos de dispositivos médicos, datos de genómica, entre otros, pueden ser analizados para identificar patrones de enfermedades, descubrir tratamientos efectivos, realizar diagnósticos predictivos y más.
- **Datos de Medios de Comunicación**: Los datos de medios de comunicación, como artículos de noticias, transmisiones de televisión, transmisiones de radio, entre otros, pueden ser analizados para realizar análisis de sentimientos, detectar tendencias de noticias, identificar temas de interés público y más.

Ejemplos de fuentes de grandes volúmenes de datos para análisis con Python:

Datos gubernamentales:

- https://en.www.inegi.org.mx/ (Instituto Nacional de Estadística y Geografía): Ofrece información estadística sobre México, incluyendo datos demográficos, económicos, sociales y ambientales.
- https://datos.gob.mx/ : Portal de datos abiertos del gobierno de México, con información de diversas entidades públicas.

Datos financieros:

- https://finance.yahoo.com/ : Ofrece datos históricos y en tiempo real sobre acciones, bonos, divisas y otros instrumentos financieros.
- https://data.nasdaq.com/publishers/QDL : Base de datos de series temporales sobre diversos temas, incluyendo economía, finanzas y energía.

Datos de redes sociales:

- https://developer.twitter.com/en/docs/twitter-api : Permite acceder a datos de tweets, usuarios y hashtags.
- https://developers.facebook.com/docs/graph-api/ : Permite acceder a datos de usuarios, publicaciones, páginas y grupos de Facebook.

Datos de sensores:

- https://www.kaggle.com/ : Ofrece una variedad de conjuntos de datos para análisis, incluyendo datos de sensores.
- https://opendatacommons.org/ : Catálogo de conjuntos de datos abiertos, incluyendo datos de sensores.

Datos científicos:

- https://www.genome.gov/human-genome-project : Proyecto Genoma Humano, con información sobre la secuencia de ADN del genoma humano.
- https://www.sdss.org/ : Encuesta digital del cielo, con información sobre millones de objetos celestes.

Would ChatGPT-facilitated programming mode impact college students' programming behaviors, performances, and perceptions? An empirical study

Dan Sun[1], Azzeddine Boudouaia[2], Chengcong Zhu[1] and Yan Li[3*]

*Correspondence:
yanli@zju.edu.cn

[1] Chinese Education Modernization Research Institute, Hangzhou Normal University, Yu Hang Tang Rd. 2318, 311121 Hangzhou, China
[2] Zhejiang University, College of Education, Yu Hang Tang Rd 866, 310058 Hangzhou, Zhejiang, China
[3] Xiaoshan High School, , Gongxu Rd 538, 311201 Hangzhou, China

Abstract

ChatGPT, an AI-based chatbot with automatic code generation abilities, has shown its promise in improving the quality of programming education by providing learners with opportunities to better understand the principles of programming. However, limited empirical studies have explored the impact of ChatGPT on learners' programming processes. This study employed a quasi-experimental design to explore the possible impact of ChatGPT-facilitated programming mode on college students' programming behaviors, performances, and perceptions. 82 college students were randomly divided into two classes. One class employed ChatGPT-facilitated programming (CFP) practice and the other class utilized self-directed programming (SDP) mode. Mixed methods were utilized to collect multidimensional data. Data analysis uncovered some intriguing results. Firstly, students in the CFP mode had more frequent behaviors of debugging and receiving error messages, as well as pasting console messages on the website and reading feedback. At the same time, students in the CFP mode had more frequent behaviors of copying and pasting codes from ChatGPT and debugging, as well as pasting codes to ChatGPT and reading feedback from ChatGPT. Secondly, CFP practice would improve college students' programming performance, while the results indicated that there was no statistically significant difference between the students in CFP mode and the SDP mode. Thirdly, student interviews revealed three highly concerned themes from students' user experience about ChatGPT: the services offered by ChatGPT, the stages of ChatGPT usage, and experience with ChatGPT. Finally, college students' perceptions toward ChatGPT significantly changed after CFP practice, including its perceived usefulness, perceived ease of use, and intention to use. Based on these findings, the study proposes implications for future instructional design and the development of AI-powered tools like ChatGPT.

Keywords: ChatGPT, Programming learning, Behavioral analysis, Perception, College student

Array databases: concepts, standards, implementations

Peter Baumann, Dimitar Misev, Vlad Merticariu and Bang Pham Huu*

*Correspondence:
b.phamhuu@jacobs-university.de
Large-Scale Scientific Information Systems Research Group, Jacobs University, Bremen, Germany

Abstract

Multi-dimensional arrays (also known as raster data or gridded data) play a key role in many, if not all science and engineering domains where they typically represent spatio-temporal sensor, image, simulation output, or statistics "datacubes". As classic database technology does not support arrays adequately, such data today are maintained mostly in silo solutions, with architectures that tend to erode and not keep up with the increasing requirements on performance and service quality. Array Database systems attempt to close this gap by providing declarative query support for flexible ad-hoc analytics on large n-D arrays, similar to what SQL offers on set-oriented data, XQuery on hierarchical data, and SPARQL and CIPHER on graph data. Today, Petascale Array Database installations exist, employing massive parallelism and distributed processing. Hence, questions arise about technology and standards available, usability, and overall maturity. Several papers have compared models and formalisms, and benchmarks have been undertaken as well, typically comparing two systems against each other. While each of these represent valuable research to the best of our knowledge there is no comprehensive survey combining model, query language, architecture, and practical usability, and performance aspects. The size of this comparison differentiates our study as well with 19 systems compared, four benchmarked to an extent and depth clearly exceeding previous papers in the field; for example, subsetting tests were designed in a way that systems cannot be tuned to specifically these queries. It is hoped that this gives a representative overview to all who want to immerse into the field as well as a clear guidance to those who need to choose the best suited datacube tool for their application. This article presents results of the Research Data Alliance (RDA) Array Database Assessment Working Group (ADA:WG), a subgroup of the Big Data Interest Group. It has elicited the state of the art in Array Databases, technically supported by IEEE GRSS and CODATA Germany, to answer the question: how can data scientists and engineers benefit from Array Database technology? As it turns out, Array Databases can offer significant advantages in terms of flexibility, functionality, extensibility, as well as performance and scalability—in total, the database approach of offering "datacubes" analysis-ready heralds a new level of service quality. Investigation shows that there is a lively ecosystem of technology with increasing uptake, and proven array analytics standards are in place. Consequently, such approaches have to be considered a serious option for datacube services in science, engineering and beyond. Tools, though, vary greatly in functionality and performance as it turns out.

Key words: Arrays, Array databases, Datacubes, SQL/MDA, OGC WCPS

Programming big data analysis: principles and solutions

Loris Belcastro[1,2], Riccardo Cantini[1], Fabrizio Marozzo[1,2*], Alessio Orsino[1], Domenico Talia[1,2] and Paolo Trunfio[1,2]

*Correspondence: fmarozzo@dimes.unical.it
[1] University of Calabria, Rende, Italy
Full list of author information is available at the end of the article

Abstract

In the age of the Internet of Things and social media platforms, huge amounts of digital data are generated by and collected from many sources, including sensors, mobile devices, wearable trackers and security cameras. This data, commonly referred to as Big Data, is challenging current storage, processing, and analysis capabilities. New models, languages, systems and algorithms continue to be developed to effectively collect, store, analyze and learn from Big Data. Most of the recent surveys provide a global analysis of the tools that are used in the main phases of Big Data management (generation, acquisition, storage, querying and visualization of data). Differently, this work analyzes and reviews parallel and distributed paradigms, languages and systems used today to analyze and learn from Big Data on scalable computers. In particular, we provide an in-depth analysis of the properties of the main parallel programming paradigms (MapReduce, workflow, BSP, message passing, and SQL-like) and, through programming examples, we describe the most used systems for Big Data analysis (e.g., Hadoop, Spark, and Storm). Furthermore, we discuss and compare the different systems by highlighting the main features of each of them, their diffusion (community of developers and users) and the main advantages and disadvantages of using them to implement Big Data analysis applications. The final goal of this work is to help designers and developers in identifying and selecting the best/appropriate programming solution based on their skills, hardware availability, application domains and purposes, and also considering the support provided by the developer community.

Keywords: Parallel Programming models, Programming systems, Big Data analysis, MapReduce, Workflow, Message Passing, Bulk Synchronous Parallel, SQL-like

Introduction

Over the last years, with the development of the Internet of Things, the growth of social networks and the widespread diffusion of mobile devices, enormous amounts of digital data are being generated by and gathered from several sources. For instance, data from sensors, webcams, in-vehicle infotainment, mobile devices, GPS devices, wearable trackers, social networks and web services is drastically rising. This huge amount of data, commonly referred to as Big Data, is characterized by the complexity, by the variety in terms of format [1], and is produced at a speed that is challenging the current storage, processing and analysis capabilities. In fact, if on the one hand it opens up to several

Investigating the effects of learning activities in a mobile Python tutor for targeting multiple coding skills

Geela Venise Firmalo Fabic, Antonija Mitrovic* and Kourosh Neshatian

* Correspondence: tanja.mitrovic@canterbury.ac.nz
Department of Computer Science and Software Engineering, University of Canterbury, Private Bag 4800, Christchurch 8041, New Zealand

Abstract

Mobile devices are increasingly being utilized for learning due to their unique features including portability for providing ubiquitous experiences. In this paper, we present PyKinetic, a mobile tutor we developed for Python programming, aimed to serve as a supplement to traditional courses. The overarching goal of our work is to design coding activities that maximize learning. As we work towards our goal, we first focus on the learning effectiveness of the activities within PyKinetic, rather than evaluating the effectiveness of PyKinetic as a supplement resource for an introductory programming course. The version of PyKinetic (PyKinetic_DbgOut) used in the study contains five types of learning activities aimed at supporting debugging, code-tracing, and code writing skills. We evaluated PyKinetic in a controlled lab study with quantitative and qualitative results to address the following research questions: (R1) Is the combination of coding activities effective for learning programming? (R2) How do the activities affect the skills of students with lower prior knowledge (novices) compared to those who had higher prior knowledge (advanced)? (R3) How can we improve the usability of PyKinetic? Results revealed that PyKinetic_DbgOut was more beneficial for advanced students. Furthermore, we found how coding skills are interrelated differently for novices compared to advanced learners. Lastly, we acquired sufficient feedback from the participants to improve the tutor.

Keywords: Python tutor, Mobile learning, Programming skills, Code writing, Code tracing, Novice students, Advanced students

Introduction

Mobile handheld devices provide distinctive features which are increasingly being utilized for learning. Pea and Maldonado (2006) summarized the unique attributes of mobile devices for learning into seven features: size and portability, small screen size, computing power and modular platform, communication ability, wide range of applications, synchronization and back-up abilities, and stylus-driven interface. These attributes are largely still relevant at this day and age, but nowadays most handheld devices, like smartphones, provide touchscreen surfaces where users can interact with directly using their bare hands without using a stylus or a thumb-pad keyboard. Smartphones are used to access course material, listen to podcasts, watch instructional videos, and communicate with peers (Dukic et al. 2015). Smartphones are also being used

Upgrading a high performance computing environment for massive data processing

Lucas M. Ponce[1*], Walter dos Santos[1], Wagner Meira Jr.[1], Dorgival Guedes[1], Daniele Lezzi[2] and Rosa M. Badia[2,3]

Abstract

High-performance computing (HPC) and massive data processing (Big Data) are two trends that are beginning to converge. In that process, aspects of hardware architectures, systems support and programming paradigms are being revisited from both perspectives. This paper presents our experience on this path of convergence with the proposal of a framework that addresses some of the programming issues derived from such integration. Our contribution is the development of an integrated environment that integretes (*i*) COMPSs, a programming framework for the development and execution of parallel applications for distributed infrastructures; (*ii*) Lemonade, a data mining and analysis tool; and (*iii*) HDFS, the most widely used distributed file system for Big Data systems. To validate our framework, we used Lemonade to create COMPSs applications that access data through HDFS, and compared them with equivalent applications built with Spark, a popular Big Data framework. The results show that the HDFS integration benefits COMPSs by simplifying data access and by rearranging data transfer, reducing execution time. The integration with Lemonade facilitates COMPSs's use and may help its popularization in the Data Science community, by providing efficient algorithm implementations for experts from the data domain that want to develop applications with a higher level abstraction.

Keywords: COMPSs, High-performance computing, Big data, HDFS, Lemonade

1 Introduction

Parallel and distributed computing frameworks have proven to be essential for applications that require high performance, usually associated with the processing of large volumes of data. Originally, efforts in that area originated from two different areas, High-Performance Computing (HPC) and Big Data. More recently there has been a tendency to combine efforts from both areas to merge their contributions. This work fits in that direction.

HPC applications are those that explore high-level parallelism and high-performance hardware, including low latency networks, to process mostly structured data with scientific algorithms. On the other hand, Big Data scenarios involve the processing of massive data volumes (usually unstructured), leveraging the use of conventional hardware and exploiting data parallelism. In this case, data could be processed as multiple individual streams and analyzed collectively in stream or in batch, for the discovery of knowledge. In such scenarios, data mining in big data has become one of the key tasks in many fields of Science [1].

Considering the convergence of HPC and Big Data, several proposals have emerged to address the requirements of those two areas [2–4]. HPC environments generally provide better interfaces for regular data and scientific algorithms based on bag-of-tasks models such as matrix computation. Despite the good performance in those scenarios, it is often hard to implement applications that handle irregular data and complex data structures in HPC frameworks. Big Data environments offer good solutions to address such kind of data, as well as to facilitate the development of applications by experts in the application

*Correspondence: lucasmp@dcc.ufmg.br
[1]Departamento de Ciência da Computação, Universidade Federal de Minas Gerais (UFMG), 31270-901 Belo Horizonte, Minas Gerais, Brazil
Full list of author information is available at the end of the article

D2O: a distributed data object for parallel high-performance computing in Python

Theo Steininger[1,2*], Maksim Greiner[1,2], Frederik Beaujean[2,3] and Torsten Enßlin[1,2]

*Correspondence: theos@mpa-garching.mpg.de
[1] Max-Planck-Institut für Astrophysik, Karl-Schwarzschild-Strasse 1, 85741 Garching, Germany
Full list of author information is available at the end of the article

Abstract

We introduce D2O, a Python module for cluster-distributed multi-dimensional numerical arrays. It acts as a layer of abstraction between the algorithm code and the data-distribution logic. The main goal is to achieve usability without losing numerical performance and scalability. D2O's global interface is similar to the one of a `numpy.ndarray`, whereas the cluster node's local data is directly accessible for use in customized high-performance modules. D2O is written in pure Python which makes it portable and easy to use and modify. Expensive operations are carried out by dedicated external libraries like *numpy* and *mpi4py*. The performance of D2O is on a par with numpy for serial applications and scales well when moving to an MPI cluster. D2O is open-source software available under the GNU General Public License v3 (GPL-3) at https://gitlab.mpcdf.mpg.de/ift/D2O.

Keywords: Parallelization, Numerics, MPI, Python, Numpy, Open source

Introduction

Background

Data sets in simulation and signal-reconstruction applications easily reach sizes too large for a single computer's random access memory (RAM). A reasonable grid size for such tasks like galactic density reconstructions [1] or multi-frequency imaging in radio astronomy [2] is a cube with a side resolution of 2048. Such a cube contains $2048^3 \approx 8.6 \cdot 10^9$ voxels. Storing a 64-bit double for every voxel therefore consumes 64 GiB. In practice one has to handle several or even many instances of those arrays which ultimately prohibits the use of single shared memory machines. Apart from merely holding the arrays' data in memory, parallelization is needed to process those huge arrays within reasonable time. This applies to basic arithmetics like addition and multiplication as well as to complex operations like Fourier transformation and advanced linear algebra, e.g. operator inversions or singular value decompositions. Thus parallelization is highly advisable for code projects that must be scaled to high resolutions.

To be specific, the initial purpose of D2O was to provide parallelization to the package for Numerical Information Field Theory (NIFTy)[3], which permits the abstract and efficient implementation of sophisticated signal processing methods. Typically, those methods are so complex on their own that a NIFTy user should not need to bother with parallelization details in addition to that. It turned out that providing a generic encapsulation for parallelization to NIFTy is not straightforward as the applications NIFTy is

A survey of open source tools for machine learning with big data in the Hadoop ecosystem

Sara Landset, Taghi M. Khoshgoftaar, Aaron N. Richter* and Tawfiq Hasanin

*Correspondence:
anichter@fau.edu
Florida Atlantic University,
777 Glades Road, Boca Raton,
FL 33431, USA

Abstract

With an ever-increasing amount of options, the task of selecting machine learning tools for big data can be difficult. The available tools have advantages and drawbacks, and many have overlapping uses. The world's data is growing rapidly, and traditional tools for machine learning are becoming insufficient as we move towards distributed and real-time processing. This paper is intended to aid the researcher or professional who understands machine learning but is inexperienced with big data. In order to evaluate tools, one should have a thorough understanding of what to look for. To that end, this paper provides a list of criteria for making selections along with an analysis of the advantages and drawbacks of each. We do this by starting from the beginning, and looking at what exactly the term "big data" means. From there, we go on to the Hadoop ecosystem for a look at many of the projects that are part of a typical machine learning architecture and an understanding of how everything might fit together. We discuss the advantages and disadvantages of three different processing paradigms along with a comparison of engines that implement them, including MapReduce, Spark, Flink, Storm, and H_2O. We then look at machine learning libraries and frameworks including Mahout, MLlib, SAMOA, and evaluate them based on criteria such as scalability, ease of use, and extensibility. There is no single toolkit that truly embodies a one-size-fits-all solution, so this paper aims to help make decisions smoother by providing as much information as possible and quantifying what the tradeoffs will be. Additionally, throughout this paper, we review recent research in the field using these tools and talk about possible future directions for toolkit-based learning.

Keywords: Machine learning, Big data, Hadoop, Mahout, MLlib, SAMOA, H2O, Spark, Flink, Storm

Background

As the price of data storage has gone down and high performance computers have become more widely accessible, we have seen an expansion of machine learning (ML) into a host of industries including finance, law enforcement, entertainment, commerce, and healthcare. As theoretical research is leveraged into practical tasks, machine learning tools are increasingly seen as not just useful, but integral to many business operations.