Apa ethical guidelines for human research psychology

Title Mastering Apache Spark 2.0 Author (s) Jacek Laskowski Publisher: GitHub Books (2016) Paperback: N/Electronic Book PDF (1,141 pages, 19.6 MB) Language: English ISBN-10: N/A ISBN-13: N/A Share This: Book Description This book presents Apache Spark, an open source cluster computing system that makes data analytics write quickly and work quickly. With Spark, you can quickly solve large datasets with simple APIs in Python, Java, and Scala. This edition contains new information about the coordinates of Spark S'L, Spark Streaming, setup and Maven. About authors Jacek Laskowski is an independent consultant who is passionate about software development and teaching people how to use Apache Spark, Scala, sbt, and Apache Kafka (with a bit of Hadoop YARN, Apache Mesos, and Docker). Reviews, Ratings and Recommendations: Related Book Category: Read and Download Links: Mastering Apache Spark 2.0 (Jacek Laskowski) Related Books: For the first time I use AsciiDoc to write a document that should eventually become a book about Apache Spark. While on the written route, I also aimed to master the git (hub) stream to write a book as described in the life of the future of technical writing (with pull requests for chapters, elements of action to show the progress of each industry and such). The branching and task progress features encompass the concept of working on a branch on a chapter and using pull queries from GitHub Flavored Markdown for task lists. Once the tasks are identified, GitHub shows the progress of the request to pull with the number of completed tasks and the bar progress. Figure 1. Pull the query with 4 tasks, of which 1 completed It's all to make things harder ... ekhm... reach higher levels of zen writing. While on the written route, I also aimed to master the git (hub) stream to write a book as described in the life of the future of technical writing (with pull requests for chapters, elements of action to show the progress of each industry and such). The branching and task progress features encompass the concept of working on a branch on a chapter and using pull queries from GitHub Flavored Markdown for task lists. Once the tasks are identified, GitHub shows the progress of the request to pull with the number of completed tasks and the bar progress. Figure 1. Pull's 4-task query, of which 1 is Apache Spark completed™ 2.0 is a monumental shift in ease of use, higher performance, and smarter API merging into Spark components. It lays the groundwork for a single API interface for structured streaming, and sets the stage for how these unified APIs will be developed in Spark components in future releases. In this fourth book, we curate technical blogs and related assets. Whether or not you start with or are an experienced developer, it will equip you with the knowledge to take full advantage of Spark 2.0, including: Introduction to Apache Apache 2.0 Combined APIs for DataSet, DataFrames and SparkSessions Machine Learning MLlib's DataFrame based on the Spark S'L API, Catalyst Optimizer and Wolframa Phase II Performance Enhancement Continuous Applications: Evolution of Spark Streaming How to Use Spark 2.0 Structured Streaming API Download e-Book, Mastering Apache Spark 2.0. Get mastering Apache Spark now with O'Reilly online training. O'Reilly members experience live online learning as well as books, videos and digital content from 200 publishers. Gain experience in data processing and storage using advanced techniques with Apache SparkAbout This bookExplore integration Apache Spark with third-party applications such as H20, Databricks and TitanEvaluate, both Cassandra and Hbase can be used to storeAn an extended guide with a combination of instructions and practical examples to expand the most modern Spark features to bypass in the world of Spark, then this book is perfect for you. It is supposed to know Linux, Hadoop and Spark. Scala is expected to be knowledgeable. What you'll learn is the Extend tools available for processing and storageAxamine clustering and classification using MLlibDiscover Spark streaming processing via Flume, HDFS Create a scheme in Spark S'L, and learn how the Spark scheme can be populated by dataStudy Sparkdy based on graph-processing Graphics with H20 and Deep Learning. , Titan, HBase and CassandraUse Apache Spark in the cloud with Databricks and AWSIn DetailApache Spark is a parallel processing system based on memory clusters that provides a wide range of functionality such as graphics processing, machine learning, thread processing and SLD. It works at an unprecedented speed, is easy to use and offers a rich set of data transformations. This book aims to take your limited Spark knowledge to the next level by teaching you how to expand Spark functionality. The book begins with a review of Spark's eco-system. You'll learn how to use MLlib to create a fully functioning neural network to recognize handwriting. You'll then learn how thread processing can be configured to optimal performance and provide parallel processing. The book is distributed to show how to include H20 for machine learning, Titan for graph-based storage, Databricks for Cloud Spark. The interim scala-based code examples are given to process the Apache Spark module in the CentOS Linux and Databricks cloud. Style and Approach This book presents its extensive guide to Apache Spark modules and tools and shows how Spark functionality can be expanded to handle and store in real time with well-established examples. Publishing September 2015 Apache Spark is a distributed and highly scalable memory data analysis system that enables applications to be developed in Java, Scala, Python, and in languages such as R. It has one of the highest contributions/participation among Apache top-level projects to date. Apache systems such as Mahout now use it as a computing engine instead of MapReduce. In addition, as shown in Chapter 4, Apache Spark S'L, you can use the Hive context to have the spark application process data directly in and out of Apache Hive.Apache Spark provides four main submodules that are S'L, MLlib, GraphX and streaming. All of them will be explained in their own chapters, but a simple overview would be useful here. The modules are compatible, so the data can be transferred between them. For example, streaming data can be transmitted to S'L, and a temporary table can be created. The next figure explains how this book will solve Apache Spark and its modules. The top two rows show Apache Spark and four of its submodules described earlier. However, where possible, I always try to show by giving an example of how functionality can be expanded with additional tools: For example, the streaming module explained in Chapter 3, Apache Spark Streaming, will run examples showing how data movement is performed using Apache Kafka and Flume. The MLlib or Machine Learning module will have its functionality reviewed in terms of the data processing features that are available, but it will also be expanded using the H2O system and deep learning. The previous figure, of course, is simplified. He represents the systemic relationships presented in this book. For example, there are many more routes between Apache Spark and HDFS than the previous chart. The Spark S'L chapter will also show how Spark can use the Hive context. Thus, the Spark app can be designed to create objects based on Hive and run Hive Evil against the Hive tables stored in HDFS. Chapter 5, Apache Spark GraphX and Chapter 6, a storage-based graph, will show how the Spark GraphX module can be used to process big data scale graphs, and how they can be stored using Titan's graph database. Titan will show that you can store big data scale graphs and shove them as graphs. For example, it will show that Titan can use both HBase and Cassandra as storage mechanisms. When using HBase, it will be shown that implicitly, Titan uses HDFS as a cheap and reliable distributed storage mechanism. So, I think this section explained that Spark is a memory processing system. When used at scale, it cannot exist alone data must somewhere. It will probably be used in alongside Hadoop's toolkit and its associated eco-system. Fortunately, Hadoop stack vendors, such as Cloudera, provide a CDH Hadoop stack and a cluster cluster integrates with Apache Spark, Hadoop and most of the current stable toolkit. During this book, I will use a small cluster of CDH 5.3 installed on The CentOS 6.5 64 bit servers. You can use an alternative configuration, but I believe CDH provides most of the tools I need and automates the configuration, leaving me more time to develop. By withering the Spark modules and the software that will be presented in this book, the next section will describe the possible design of the big data cluster. In this section, I would like to present an overview of the functionality that will be presented in this book from the perspective of Apache Spark, and the systems that will be used to expand it. I'll also try to explore the future of Apache Spark as it integrates with cloud storage. When you look at the documentation on the Apache Spark website (you will see that there are topics that cover SparkR and Bagel. While I will cover the four main Spark modules in this book, I won't cover these two topics. I have limited time and opportunity in this book, so I will leave these topics to read the investigation or on a future date. Spark MLlib offers machine learning functionality in a number of areas. The documentation available on the Spark website introduces the data types used (such as LabeledPoint vectors and structure). This module offers functionality that includes: StatisticsClassificationRegressionCollaborative filtrationClusterEdClassingDimensionality ReductionFeature ExtractionFrequent Pattern MiningOptimizationSkal based on practical examples of KMeans, Naa¨ve Bayes, and artificial neural networks have been introduced and discussed in Chapter 2, Apache Spark MLlib of this book. Thread processing is another big and popular theme for Apache Spark. It includes processing data in Spark as streams, and covers topics such as input and weekend operations, conversions, perseverance and control directions among others. Chapter 3, Apache Spark Streaming, covers this area of processing and provides practical examples of different types of thread processing. It discusses the configuration of the flow of packages and windows, as well as a practical example of checkpoint effects. It also covers various examples of thread processing, including Kafka and Flume.There are many other ways in which data flow can be used. You can use other Spark features (such as S'L, MLlib, and GraphX) to process the flow. Spark streaming can be used using systems such as Kinesis or zero MH. You can even create custom receivers for your own user data sources. From the Spark 1.3 version, data frames were entered into Apache Spark, so the data can be processed in tablicular form and table functions (e.g. select, filter, groupBy) can be used to process data. The Spark S'L module integrates with the Parket and JSON formats for data that will be stored in formats that better represent data. It also offers more opportunities to integrate with external systems. The idea of integrating Apache Spark into the Hadoop Hive big data database can also be introduced. Context-based Hive Spark applications can be used to manage table data based on Hive. This results in rapid distributed processing in Spark memory to Hive's big data storage capabilities. This effectively allows Hive to use Spark as a processing engine. The Apache Spark GraphX module allows Spark to offer quick, big data when processing memory graphs. The graph is represented by a list of vertices and edges (lines that connect vertices). GraphX is able to create and manipulate graphs using property, structuring, joining, aggregation, cache and operators' sky. It introduces two new types of data to support graph processing in Spark: VertexRDD and EdgeRDD to represent graph tops and edges. It also introduces graph example features such as PageRank and triangle processing. Many of these features will be addressed in Chapter 5, Apache Spark GraphX. When studying big data processing systems, I think it's important to look not only at the system itself, but also how it can be expanded, and how it integrates with external systems so that a higher level of functionality can be offered. In a book of this size, I can't cover every option, but hopefully by presenting a theme, I can stimulate the reader's interest so that they can explore further. I used the H2O library machine learning system to expand the Apache Spark machine learning module. Using an example based on the deep learning of H2O Scala, I showed how neural processing can be introduced into Apache Spark. I am, however, aware that I have just scratched the surface of the H2O function. I used only a small neural cluster and one type of functionality classification. In addition, there is much more to H2O than deep learning. As the chart treatment is more accepted and used in the coming years, so there is a chart based storage. I researched the use of Spark with the NoS'L Neo4J database using a prototype mazerunner application. I also researched the use of titan's Aurelius (Datastax) database for graph-based storage. Again, Titan is a database in its infancy that needs both community support and further development. But I'd like to explore future options for integrating Apache Spark. The next section will show that the Apache Spark release contains scripts to create a Spark cluster on the AWS EC2 vault. There are a number of options that allow the cluster creator to identify attributes such as cluster size and storage type. But this type of cluster is difficult making it difficult to manage changing requirements. If the amount of data changes or grows over time, you may need a larger cluster with a larger the people who developed Apache Spark have created a new startup called Databricks that offers control of the Spark cluster based on web consoles, as well as a host of other features. It offers the idea of working organized laptops, managing user access, security, and a host of other features. It is described at the end of this book. This service is in its infancy, currently only offering cloud storage on Amazon AWS, but this is likely to extend to Google and Microsoft Azure in the future. Other cloud providers, i.e. Google and Microsoft Azure, are also expanding their services so they can offer Apache Spark processing in the cloud. As I mentioned, Apache Spark is a distributed, memory-based, parallel processing system that needs a connected storage mechanism. So when you build a big data cluster, you're likely to use a distributed storage system like Hadoop, as well as data-moving tools such as Sqoop, Flume, and Kafka.I, that would like to present the idea of edged nodes in a big data cluster. These nodes in the cluster will collide with the customer on which the components facing the customer, such as Hadoop NameNode or possibly Master Spark, are located. Most of the big data cluster can be behind the firewall. The edge nodes will then reduce the complexity of the firewall because they will be the only nodes that will be available. The next digit shows a simplified cluster of big data: it shows four simplified cluster racks with switches and edge node computers facing the customer through the firewall. It's certainly stylized and simplified, but you get the idea. Common processing nodes are hidden behind the firewall (dotted line) and are available for general processing, from the perspective of Hadoop, Apache Spark, Animal Protector, Flume and/or Kafka. The next digit is several nodes at the edge of the big data cluster and tries to show which applications might be on them. The applications of the edge site will be master applications similar to Hadoop NameNode, or Apache Spark master server. These will be components that bring data in and out of the cluster, such as Flume, Sqoop and Kafka. This could be any component that makes the user interface accessible to the customer's user similar to Hive: Typically, firewalls, adding security to a cluster, also increase complexity. Ports between the components of the system should be open so that they can talk to each other. For example, a zookeeper is used by many components for configuration. Apache Kafka, a messaging system with a subscription to publish, uses a messaging system to customize its themes, groups, consumers and manufacturers. Thus, client ports for the zookeeper, through the firewall, must be opened. Finally, you need to take into account the distribution of systems to cluster nodes. For example, if Apache Spark uses Flume or Kafka, channels will be used. You need to take into account the size of these channels and the memory used caused by the data flow. Apache Spark should not compete with other Apache components for memory use. Depending on data streams and memory usage, you may want to have Spark, Hadoop, Animal Protector, Flume, and other tools on individual cluster nodes. Typically, edge nodes that act as NameNode cluster servers, or Spark Master servers, will need more resources than cluster processing nodes in the firewall. For example, a CDH cluster site manager's server would need additional memory, as did the main Spark server. You need to monitor the edge nodes to use the resources and adjust their location if necessary in terms of the resources and/or location of the application. This section summarizes the scene for the big data cluster from the perspective of Apache Spark, Hadoop, and other tools. However, how can you set up the Apache Spark cluster itself in a big data cluster? For example, you can have many types of Spark cluster manager. The next section will address this issue and describe each type of Apache Spark cluster manager. The following diagram, borrowed from the spark.apache.org website, demonstrates the role of Apache Spark cluster manager in terms of wizard, slave(employee, artist, and client applications: Spark context, as you'll see from many examples in this book, can be determined by the Spark configuration object and Spark URL. The Spark context connects to the Spark cluster manager, who then allocates resources to work nodes for the application. The cluster manager distributes the performers to the cluster workers' nodes. He copies the application cans file for employees, and finally he distributes the tasks. The following subsections describe the possible options for the Apache Spark cluster manager currently available. If you specify a local URL of the Spark configuration, you may start your app locally. By specifying a local one, you can use Spark threads to run your app locally. This is a useful development and testing option. The offline mode uses a basic cluster manager that comes with Apache Spark. The url of the Spark Master will be as follows: Here is the name of the host on which The Spark Master works. I pointed to the 7077 as a port, which is the default, but it's customizable. This simple cluster manager, currently, only supports FIFO (first in the first of) planning. You can think of simultaneous application planning by setting resource configuration settings for each application. For example, use spark.core.max to exchange kernels between apps. On a larger scale, when integrating with Hadoop YARN, the Apache Spark cluster manager may be YARN, and can work in one of two modes. If the main value of Spark is set to be the main value of the zlt;/hostname, it's not a big deal. the application can then be filed in a cluster and then terminated. The cluster will take care of the allocation of resources and tasks. However, if the master of the application is presented as a client yarn, the application stays alive during the processing lifecycle, and requests resources from YARN. Apache Mesos is an open source system for sharing resources in a cluster. This allows multiple frameworks to share clusters, managing and planning resources. It's a cluster manager that provides insulation with Linux containers, allowing multiple systems such as Hadoop, Spark, Kafka, Storm, and more to safely share the cluster. It is very scaled for thousands of knots. It is a master of the slave-owning system, and the wine is tolerant, using the zookeeper to control the configuration. For one master node Mesos cluster, the Spark master URL will be in this form: Where is the host name of the main server, the port is defined as 5050, which is the default port of the Mesos Master (this is customizable). If there are several Mesos servers in the large-scale Mesos high-availability cluster, the Spark Master URL will look like this: so the choice of the main Mesos server will be controlled by the zookeeper. It will be the name of the host in the zookeepers quorum. In addition, port number 2181 is the default master port for zookeeper release. The Apache Spark contains scripts to run Spark in the cloud against Amazon AWS EC2 servers. The following list, as an example, shows Spark 1.3.1 installed on the Linux CentOS server, under a catalog called /usr/local/spark/. EC2 resources are available in the Spark EC2 sub-direction: you need to set up an Amazon AWS account to use Apache Spark on EC2. You can set up your initial free account to try it out here: you look at Chapter 8, Spark Databricks you'll see that such an account was created and used to access . The next thing you need to do is access your AWS IAM console, and choose a user option. You create or choose a user. Select the user's action option and then select Access Control Keys. Then select the creation access key and then download the credentials. Make sure the downloaded key file is secure, assuming you are on a Linux chmod file with permissions of No. 600 for users-only access. You will now have an access key, a secret access key, a key file, and the key pair's name. Now you can create the Spark EC2 cluster using the spark-ec2 script as follows: Here's the name of the key pair you gave when creating access details; This is the file you downloaded. Name that you're going to create, it's called. The region chosen here is in the western part of the United States, us-west-1. If you live in the Pacific Ocean, like me, you're living in the Pacific Ocean, you're living in the Pacific Ocean. it may be wiser to choose a closer region like the ap-southeast-2. However, if you have problems accessing benefits, you will need to try another zone. Remember also that using Cloud Clustering Spark as it will have a higher latency and poorer VIOs overall. You share your clusters with multiple users, and the cluster may be in a remote region. You can use a number of options for this core team to set up the Spark cloud cluster you're creating. The option can be used: it allows you to determine how many nodes of workers to create in the Spark EC2 cluster, i.e., 5 euros for six node cluster, one master and five slave owners. You can identify the Spark version that the cluster is running, not the latest version by default. The next option is to launch a cluster with Spark 1.3.1: The type of instance used to create the cluster will determine how much memory is used and how many cores are available. For example, the following setting will set the type of instance as m3.large: Current instances types for Amazon AWS can be found using the following data: shows current (July 2015) types of AWS M3 instances, model parts, kernels, memory, and storage. There are many types of instances available now; for example, T2, M4, M3, C4, C3, R3 and more. Examine current availability and choose accordingly: Pricing is also very important. Current prices such as AWS storage can be found by house prices are displayed in the region with the menu falling, and price after hour. Keep in mind that each type of storage is defined by cores, memory, and physical storage. Prices are also determined by the type of operating system, i.e. Linux, RHEL and Windows. Simply select the OS through the top-level menu. Here's an example of pricing at the time of writing (July 2015); it's just provided to give an idea. Prices will vary over time, and the service provider. They will vary in size of storage that you need and the length of time you are willing to commit. Keep up to date with the cost of moving data from any storage platform. Try to think long term. Check to see if you need to move all or some of your cloud data to the next system in, say, five years. Check the process to move the data and include these costs in the planning. As described, the previous figure shows the cost of AWS storage types across the operating system, region, storage type, and hour. Costs are measured per unit of hours, so systems such as don't stop ec2 instances until the full hour has passed. These costs will change over time and should be controlled by (for AWS) AWS payment console. you can also have problems if you want to change the size of the Spark EC2 cluster, so you need to be sure of the configuration of the main slave slave You're starting. Be sure how many workers you are going to require and how much memory you need. If you feel that your requirements will change over time, then you may want to consider using if you definitely want to work with Spark in the cloud. Go to Chapter 8, Spark Databricks and see how you can customize, and use in the next section, I'll explore the Apache Spark Performance Cluster, and the issues that might affect it. Before I go to the rest of the chapters covering the functional areas of Apache Spark and expanding to it, I'd like to explore the performance area. What issues and areas should be considered? What can affect the performance of the Spark app, from the cluster level to the actual Scala code? I don't want to just repeat what Spark says, so look at the following URL: amp;lt;version>/tuning.html.Here, zlt'gt;refers to the Spark version that you use, that is, the latter, or 1.3.1 for a specific version. So, looking at this page, I'll briefly mention some of the thematic areas. I'm going to list

some common points in this section without implying an order of importance. The size and structure of the big data cluster will affect performance. If you have a cloud cluster, your IO and latency will suffer compared to an undivided hardware cluster. You will share basic equipment with multiple customers and that cluster equipment can be remote. In addition, positioning cluster components on servers can cause a resource dispute. For example, if possible, think carefully about searching Hadoop NameNodes, Spark, zookeeper, Flume, and Kafka servers in large clusters. If you're working hard, you might want to consider separating servers into separate systems. You can also use An Apache, such as Mesos, to share resources. In addition, consider potential concurrency. The more employees in the Spark cluster for large datasets, the more opportunities for concurrency. You might want to consider using an HDFS alternative based on the cluster requirements. For example, MapR has a mapR-FS NFS based on read write file system to improve performance. This file system has the full ability to record reading, while HDFS is designed to write once, read a lot of file system. It offers better performance compared to HDFS. It also integrates with hadoop and Spark. Bruce Penn, architect of MapR, wrote an interesting article describing its features on: search a blog called MapR-FS Comparison and HDFS NFS and Snapshots. article describes mapR architecture and possible performance gains. Localization of data or the location of the data being processed and the processing of the spark. Are the data from AWS S3, HDFS, local file system/network or remote source? As a mention is the previous link setting, if the data is removed, then the functionality and data should be put together to be processed. Spark will try to use the best level of data locality to handle tasks. To avoid OOM (Out of Memory) messages for tasks, you can look at a number of areas in the Apache Spark cluster: Consider the level of physical memory available on Spark employee nodes. Can it be enlarged? Let's look at the separation of data. Can you increase the number of sections in the data used by the Spark app code? Can you increase the storage fraction, the memory used by JVM to store and caching RDD?? Let's look at setting up the data structures used to reduce memory. Consider serializing RDD storage to reduce memory usage. Try setting up the code to improve the performance of the Spark app. For example, filter app-based data at the beginning of the ETL cycle. Set up your degree of concurrency, try to find resource-expensive parts of the code and find alternatives. Although, much of this book will focus on Apache Spark examples set on physically server clusters (except for , I'd like to do what there are a few cloud options out there. There are cloud systems that use Apache Spark as an integrated component, and cloud systems that offer Spark as a service. While this book may not cover all of them in depth, I thought it would be helpful to mention some of them: Databricks covered by two chapters in this book. It offers the Spark cloud service, which currently uses AWS EC2. It is planned to extend the service to other cloud service providers (. At the time of writing (July 2015), Microsoft Azure has been expanded to offer Spark support. Apache Spark and Hadoop can be installed on Google Cloud.The Oryx system was built at the top of Spark and Kafka in real time, large-scale machine learning (. The velox system for predicting machine learning is based on Spark and KeystoneML (. PredictionIO is an open source machine learning service built on Spark, HBase and Spray (. SeldonIO is an open source predictive analysis platform based on Spark, Kafka, and Hadoop (. When I close this chapter, I would like to invite you to work out each of the Scala-based examples in the following chapters. I was impressed with the speed with which Apache Spark was developing, and I was also impressed with the frequency of releases. So At the time of writing, Spark has reached 1.4, I'm sure you'll use more than 2nd version. If you're in trouble, problems, them logically. Try contacting Spark's user group for help(or check the Spark website in I'm always interested to hear from people and connect with people on sites such as LinkedIn. I really want to hear about the projects that people are involved in and new opportunities. I'm interested to hear about Apache Spark, the ways that you use it and the systems that you build are used in scale. I can be contacted by LinkedIn on linkedin.com/profile/view?id=73219349.Or: linkedin.com/profile/view?id=73219349.Or, I can be contacted via my website via email or finally by emailing at qlt; qgt; More Unlock this book with FREE 10-day trial Mike Frampton is an IT contractor, blogger, and IT author with a keen interest in new technologies and great data. He has been working in the IT industry since 1990 in various roles (tester, developer, support and author). He has also worked in many other sectors (energy, banking, telecommunications and insurance). He currently lives on a beach in Pa-Raparaumu, New York, with his wife and teenage son. Being married to a Thai citizen, he divides his time between Paraparaumu and their home in Roi Et, Thailand, between writing and IT consulting. He's always keen to hear about new ideas and technologies in big data, AI, IT and hardware, so look it out on LinkedIn (or his website (/pageHome) to ask questions or just say hello. View this author's publications