Portfolio Optimization using a Big Data Framework: A Passive Management, Spark and Genetic Algorithm approach

Pedro B. Caeiro

Abstract—This work analyzes the literature regarding portfolio management and proposes to perform this task automatically. A portfolio management model—based on the seminal work of Markowitz—, a metaheuristic for mathematical optimization—genetic algorithm—and a Big Data framework—Spark—are used in the system to guide stock selection. The main goal is to design a portfolio management system able to use the extensively studied genetic algorithms to solve a portfolio optimization problem while being capable of storing and processing large datasets. The secondary goal is to provide a basis for further testing of Big Data frameworks in portfolio management. Computer performance validation was done via analyzing the impact of concurrency, iteration processing and Spark memory configurations on runtime. This validation showed that the system can reduce runtime via concurrency—more than 90% latency speedup between 1 and 4 cores—and, thus, it is viable to advance with further studies regarding scalability in clusters. The system also attained adequate financial performance by beating the SPY exchange traded fund consistently, albeit not always the buy and hold strategy, in the first two weeks after optimization. In these two weeks, the solution return on investment gains reached up to 2% above SPY.

Index Terms—Big Data, Genetic Algorithm, Markowitz, Mean-Variance Analysis, Spark

I. INTRODUCTION

Managing a portfolio through mathematical optimization was first introduced by Markowitz in 1952, coining the concept of a Portfolio Optimization Problem (POP). His work is the basis of Modern Portfolio Theory (MPT) and aims at obtaining, out of a set of assets, the portfolio which minimizes risk, measured as a variance, for a specified level of minimum wanted return [1], [2]. Markowitz’s work is traditionally considered a passive approach [3], [4]. In fact, although MPT often yields results that beat the benchmark index, more active management strategies for portfolio management exist.

Since optimization problems are at the core of current portfolio management systems, methods to solve them need to be understood. These methods clearly diverge into two main categories: exact methods, which can guarantee a globally optimal solution but are usually computationally taxing, and heuristic methods, that have no mathematical guarantee of finding a globally optimal solution but are computationally less expensive [5]. Most works in literature are mainly worried with solving the optimization problems subjacent to portfolio management in a computationally efficient manner. Since more complex models of the problem are intractable, the focus of most recent related works has shifted from traditional exact methods to heuristic methods [1], [6]–[11]. Metaheuristics are a specific type of heuristic method that has seen an increase in published works, being used to solve both passive and active management models [1], [3], [6], [12], [13]. Many of the analyzed works evaluate results by how computationally efficient the system is or how accurate it is w.r.t. to exact methods. With these metrics they attain results which are very promising. But, while, this is a legitimate validation methodology in Computer Science, specifically for metaheuristics, it does not align with the original goal of financial portfolios. The true goal of managing a portfolio is to attain the highest possible return while maintaining the lowest possible perceived risk. As such, this work is motivated by the idea of validating the system via the profitability of the managed portfolio, following the trend established by [3] and [12]. Those works achieved encouraging results and their future work propositions are motivations for this work, specifically the inclusion of parallel processing and the ability to analyze historical data regarding more than one financial market.

Historical data can be found online, for example at [14] and [15], and is available in heterogeneous data formats and for an increasingly large number of assets. As these datasets become larger the need for distributed and parallel computing to process and analyze them increases, since traditional systems become inadequate to give solutions in an acceptable time window. Recent Big Data frameworks, like Hadoop or Spark, introduce a chance to improve computational performance via distributed and parallel computing while abstracting from the programmer many of the downsides of such a system [16], [17]. These frameworks present an opportunity to achieve this work motivations regarding the ability to process large datasets.

The goal of this work is to develop a software system that manages a portfolio through solving optimization problems with large datasets. This will be achieved using a domain tuned metaheuristic implemented in a Big Data framework. The portfolio will be managed from a passive management perspective.

This work is organized as follows: Section II presents the relevant literature regarding portfolio optimization, with emphasis on solutions using metaheuristics, and Big Data frameworks; Section III specifies the system’s architecture and design choices; Section IV presents the validation
methodologies used and experimental results; finally, Section V is a brief conclusion to this work.

II. RELATED WORK

In this section, the most relevant related work will be introduced, mostly inside the domain of portfolio management. Since the number of portfolio management systems which use Big Data techniques is insufficient, several works regarding Big Data frameworks will also be presented.

A. Modern Portfolio Theory

In Markowitz’s work [2], the market is modeled by n tradeable securities and the rate of return is \( r_i \) and the expected return is \( E[r_i] \). The error for each security is labeled \( \sigma_i \). A portfolio \( W \) is composed by \( n \) weights \( (w_1, ..., w_n) \). Each \( w_i \) represents the proportion of the available budget invested in the security \( i \). There are two typical constraints to the value of \( w_i \), as seen in Equations 1 and 2.

\[
\sum_{i=1}^{n} w_i = 1 \quad \text{(1)}
\]
\[
0 \leq w_i \leq 1 \quad \text{(2)}
\]

Equation 2 establishes that no short-selling, that is selling of borrowed securities, is permitted and that an asset can occupy, at most, the whole portfolio [2], [18], [19]. Any asset has an expected return denoted \( R_i \). The expected return of the portfolio is defined in Equation 3 [2].

\[
R_p = \sum_{i=1}^{n} R_i w_i \quad \text{(3)}
\]

Each asset also has a risk \( \sigma_i \) expressed as the variance of its returns over time. The portfolio’s risk is formulated as the covariance between its assets, as formalized in Equation 4 [2], [18], [19].

\[
\sigma_p = \sum_{i=1}^{n} \sum_{j=0}^{n} \sigma_{ij} w_i w_j \quad \text{(4)}
\]

In which \( \sigma_{ij}, j \neq i \) is the covariance between \( i \) and \( j \) and \( \sigma_{ii} \) is the variance of security \( i \). The POP can then be expressed as seen in Equation 5 [3], [18], [19]. In this optimization problem a \( R_p \) must be chosen, which represents the intended constant level of expected return [20].

\[
\min_{w_i} \sigma_p, \text{ subject to (1), (2) and (3)} \quad \text{(5)}
\]

Even though additional constraints to the introduced Markowitz’s formulation create a more computationally complex problem, they are needed to create a more realistic model of the POP. This is true because Markowitz’s makes some naïve assumptions, such as: a market without taxes, no transactions costs and no short selling [44]. There is also the consideration of single-period, where the portfolio is defined once, or multi-period, where the portfolio is managed and readjusted periodically [47].

In the classical MPT, the POP is single-objective (SO), it has only one objective function, despite taking into consideration both risk and return. This is achieved by setting a constant level of return and having the objective function minimize risk. This classical Markowitz SO POP can be solved using an exact method like quadratic programming (QP), however the addition of a simple cardinality constraint, limiting the number of assets in the portfolio, renders the problem NP-Hard [10], [11]. This makes the use of heuristic methods, particularly metaheuristics, more attractive compared to traditional exact methods [9], [13].

B. Optimization Techniques for Portfolio Management

Metaheuristics can solve the classical POP extended with several constraints as well as technical or fundamental analysis-based optimization. The most preeminent metaheuristics should be known:

- Genetic algorithms (GAs);
- Tabu Search (TS);
- Simulated Annealing (SA);
- Particle Swarm Optimization (PSO).

One of the foremost works that applied metaheuristics to solve a POP was Chang et al. [21]. The authors use GAs, SA and TS and their findings indicated that the Genetic Algorithm (GA) implementation was the best alternative when trying to approximate the efficient frontier of the original Markowitz’s model. Nevertheless, when adding a simple cardinality constraint, the difference in performance was found to be negligible between the three methods. The authors depicted the POP as a SO problem using a lambda trade-off function between return and variance. In the same year, Busetti [8] stated that in a POP with nonlinear transaction costs, cardinality, floor and ceiling constraints the GA method outperformed the TS, detailed as a SO optimization problem with a lambda trade-off return variance function. In 2009, Soleimani et al. [9] showed that, for their specific dataset, GA could solve a SO POP with an objective function which minimizes variance within a 3% margin of error to the exact global optimum solution.

In 2011, Woodside et al. [6] continued the work of Chang et al. [21] by implementing a SO POP with GA, TS and SA and comparing the metaheuristics. GA and TS were the best individual solutions, with TS showing less mean error but larger median error and a much larger computation time when compared to GA. The main conclusion of Woodside et al. [6] is that their metaheuristics implementations were more efficient than in Chang et al. [21] and that pooling the results of different metaheuristics further improves their performance. The considered constraints were cardinality, floor and ceiling. A new solution using EAs was proposed by Lwin and Qu [45] during 2013. It used a combination of Population Based Incremental Learning (PBIL) and Differential Evolution (DE), two variants of EAs. This hybrid algorithm was named PBILDE. While PBILDE mostly outperformed all the compared metaheuristics, it failed to outperform, on many tests, the GA implemented by Woodside et al. [11].

While the start of metaheuristic usage to solve the POP might have been dominated by SO formulations, multi-objective (MO) formulations developed parallelly to the SO formulations. MO formulations evaluate solutions based on distinct objectives, like mean return and variance for example, that are not in single objective function, as in SO formulations. There is no major indicator that MO is superior to SO, or vice-versa. Nevertheless, when dealing with EAs, MO EAs have the advantage of generating an approximate efficient frontier in a single run [3]. Skolpadungket et al. [22] benchmarked the performance of several MO EAs with cardinality, floor and minimum transactions lots constraints.

The previously mentioned works all revolve around Markowitz’s MPT to model the portfolio management, representing it as a POP. This is a passive approach towards portfolio management. Works that focus on a different model
of the problem and on a more active management perspective will now be introduced. Gorgulho et al. [3] detail a work that uses TA to model the portfolio management problem. The work allows for short selling and uses a chromosome representation which includes TA indicators and the strength of the buy and short signals towards an asset. The optimization problem in Gorgulho et al. [3] is SO and does not aim at optimizing the weights of each asset owned in the portfolio. Rather, it aims at optimizing the weight of each TA indicator so that the portfolio gives the best Return on Investment (ROI) possible during a back-testing period. The authors showed that the system could effectively beat the buy and hold and random strategies in some time periods.

C. Big Data Frameworks

There are very few portfolio management systems using Big Data frameworks. The proposal of Jothimani et al. [23] is a good example of a suggested software architecture using these tools, although theoretical. This solution suggests Data Envelopment Analysis (DEA), a non-parametric linear programming tool, in addition to the Hadoop framework to perform sentiment analysis and obtain perceived efficient stocks. It then uses machine learning techniques, k-means clustering and Artificial Neural Networks (ANNs), for stock diversification and ranking of assets. Finally, it suggests a MPT formulation for optimization using metaheuristics, like GAs, PSO and Ant Colony Optimization (ACO).

Since we cannot compare Big Data frameworks in the context of portfolio management domain, it is important to better understand these frameworks to properly choose the most adequate for this work’s solution. In the work of Inoubli et al. [24] a survey on Big Data frameworks is done, which encompasses the five most used frameworks: Hadoop, Spark, Storm, Samza and Flink [24]. The first step in these authors’ analysis is to conduct empirical experiments comparing scalability, resource usage and effect of different configurations of parameters. The authors remark that in the case of small datasets Spark is the fastest framework, followed by Flink and then Hadoop. When the dataset size is big, Spark remains the fastest, followed by Hadoop and, finally, Flink [24]. The next relevant experiment Inoubli et al. [24] performed was aimed at studying the scalability and processing time of these frameworks. The authors achieved this by varying the number of nodes in the cluster. Spark showed to be the more reliable framework. Another experiment relevant to this work’s scope in the work of Inoubli et al. [24] analyses how the number of iterations impacts average processing time of the K-Means algorithm with a dataset containing 10 million learning examples. In this experiment Flink caught up to Spark performance-wise and even surpassed it with higher iteration counts.

D. Overview and Discussion

Analyzing the literature in Table I, it is noticeable that it is very hard to compare existing solutions via benchmarks, w.r.t. accuracy, computational performance or profitability, due to the heterogeneity of evaluation metrics, datasets and time periods used in back-testing. This makes it hard to analyze which is the best overall main optimization method and remains true even if we take into consideration that a substantial number of works,

<table>
<thead>
<tr>
<th>Reference</th>
<th>Year</th>
<th>Main methods</th>
<th>Supplementary traits</th>
<th>Big Data</th>
<th>Constraints</th>
<th>Model</th>
<th>MO or SO?</th>
<th>Objective functions</th>
<th>Dataset</th>
<th>Validation Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jothimani et al. [23]</td>
<td>2014</td>
<td>GA, PSO, ACO</td>
<td>Data Envelopment Analysis (DEA), Sentiment Analysis, Diversification through clustering: K-Means, Louvain, Ranking: ANN</td>
<td>Hadoop</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Di Tollo et al. [26]</td>
<td>2014</td>
<td>TS, SD, FD</td>
<td>QP used to determine weights of assets</td>
<td>-</td>
<td>Cardinality, Floor, Ceiling</td>
<td>MPT and index tracking</td>
<td>SO</td>
<td>Min. risk using QP solver</td>
<td>Weekly prices. Indexes: Hang Seng, DAX100, FTSE100, S&amp;P100, Nikkei225</td>
<td>Time, Coverage Measure</td>
</tr>
<tr>
<td>Maguire et al. [27]</td>
<td>2012</td>
<td>GA</td>
<td>Parallel populations for each objective</td>
<td>Parallelization</td>
<td>Floor, Multi-period consideration</td>
<td>MPT</td>
<td>MO</td>
<td>Return to standard deviation ratio; Return to variance ratio</td>
<td>Daily prices. Indexes: S&amp;P500</td>
<td>Same as objective functions</td>
</tr>
</tbody>
</table>
like Woodside et al. [11] or Lwin and Qu [50], try to create a platform for performance comparison between metaheuristic algorithms by using the dataset of Chang et al. [49]. The most frequently used main optimization method is the GA metaheuristic, which presents good overall results in every work that implements it and in benchmarks. As such, GA will be the core of the main optimization method considered in this work.

A very straightforward observation regarding the state of the art in portfolio management is the notorious lack of approaches using modern Big Data frameworks, like in Jothimani et al. [58], and even more traditional parallel or distributed approaches, such as the work of Maguire et al. [56]. The lack of Big Data techniques for portfolio management is very puzzling, since the ability to feed more historical data to a portfolio management system enables the analysis of an increased number of assets to choose from. The larger number of available assets can increase the likelihood of finding undervalued ones, which provide higher portfolio returns. Although a Big Data system would have to deal with larger quantities of data, it can achieve results in a reasonable amount of time due to the extensive use of parallel and distributed computing techniques, taking advantage of the highly available and affordable cloud computing paradigm, as described by Yee [59]. Considering the comparative analysis provided by the work of Inoubli et al. [34] regarding batch processing Big Data frameworks, Spark seems to generally outperform Hadoop and Flink in scalability. Nevertheless, it is interesting to note that Flink outperforms Spark in iterative processing. Since GA is an iterative algorithm, this fact could suggest that Flink would be the better alternative. Despite this, the scalability of Spark allied with its maturity in terms of libraries development and strong presence in production environments make Spark the Big Data framework chosen for this work. While Spark is still not efficient enough to beat dedicated domain-specific systems using parallelization technologies like OpenMP and MPI, as shown by the work of Reyes-Ortiz et al. [60], the ease of implementation coupled with the existence of the cloud computing paradigm make it a suitable choice.

The most frequent portfolio management model is MPT, which indicates a strong dominance of passive management approaches. Since there is no clear better management approach, this work will implement the predominant passive management based on Markowitz’s MPT. Regarding constraints in the passive management approach, the most prevalent, as well as the ones that will be adopted in this work are: cardinality, floor and ceiling. Considering attained results, it is not clear which number of objective functions is best in the underlying optimization problem: SO or MO. It is apparent that MO optimization offers a more suitable overview of trade-offs, since it returns the whole Pareto curve. However, SO can achieve the same level of information with multiple runs. Another apparent difference is that in SO optimization the user defines trade-off preferences before running the algorithm, while in MO preferences are expressed after the run. Since this work will already make use of Spark as its Big Data framework, which is scarcely studied in portfolio management literature, the solution will use the more traditional SO problem formulation. As for what objective functions to use in the underlying optimization problems, maximization of mean return and minimization of variance, as well as the linear combination of both, seem to be the norm in passive management.

Lastly, regarding evaluation metrics, this work will deviate from the norm by adopting a financial performance metric, namely ROI, as opposed to accuracy w.r.t. exact methods. It will also employ the most prevalent metric, which is computation time.

III. PORTFOLIO MANAGEMENT SYSTEM

This section describes the achieved solution: a portfolio management system which suggests a portfolio composition for a certain time window.

A. Overview and Architecture

In this section the proposed solution will be introduced. The features of this work are the use of a Big Data framework – Spark – in the data Extract, Transform, Load (ETL) process and in the GA used to solve a MPT-based passive portfolio management optimization problem.

Considering the established goals, the proposed architecture will have to be able to accommodate them via satisfying the following requirements:

1. Retrieve and store large amounts of heterogeneous historical market data available online;
2. Prepare extracted data for system use via an ETL process;
3. Solve an optimization problem modeled with MPT using cardinality, floor and ceiling constraints using a SO GA;
4. Evaluate the attained solutions;
5. Be scalable and minimize amount of performance deterioration with scalability.

An overview of the components composing the system is depicted in Figure 2. Different components are responsible for satisfying different requirements: Download Script and HDFS are responsible for 1., Data Module for 2., Optimization Module

![Fig. 1. Portfolio Management Data Flow and Interaction Overview. The boxes represent the system’s components. The black arrows represent user interaction via inputs and outputs. The blue arrows represent interaction between system modules.](image-url)
for 3. and, finally Portfolio Module for 4. Requirement 5 is satisfied by using the Spark framework alongside good programming practices.

To achieve this work’s goals, the portfolio management system must be able to construct a valid portfolio (respecting constraints) step-by-step. Each system component, or module, combines a set of actions and represents a step in the construction of the solution portfolio. Every module is unique in its goals and responsibilities, coming together to form the whole portfolio management system.

The complete list of system components, as can be seen in Figure 1, is composed by five modules that fall into three different tiers:

- Hadoop HDFS (File System): The file system component of the system corresponds to Hadoop HDFS. This component is responsible for persistent data storage in a distributed fashion. For testing purposes HDFS can be interchangeable with a local file system. Belongs to the Data Access Tier.
- Download Script: This system component is charged with periodically extracting historical data from the Yahoo! Finance online database and performing some minor transformations. Belongs to the Data Access Tier.
- Data Module: One of the most fundamental system components. It reads historical data from the file system and performs transformations in it. Additionally, it saves processed data to the file system and is also responsible for loading it to the main memory. Belongs to the Data Access Tier.
- Optimization Module: The core of the portfolio management system. This module is responsible for the SO GA iterative process that generates the solution portfolio. Belongs to the Optimization Tier.
- Portfolio Module: An auxiliary module responsible for keeping portfolio evaluation functions and generating comparison portfolios for testing. Belongs to the Portfolio Management System.
- User Module: A very straightforward module, which allows the user to specify input parameters to the portfolio management system and abstracts the system complexity.

The data flow within the system, how information is transferred between components, can also be consulted in Figure 1. The system, with the exception of the Download Script module which uses the popular Python Pandas library, was developed in Scala using the DataFrame abstraction and making use of SparkSQL and SparkMLlib functions.

B. Hadoop HDFS

To implement this component a distribution of Hadoop must be installed and configured so it can communicate with the Portfolio Management System. Despite being open-source and readily available online, alongside proper documentation, the Hadoop ecosystem is highly complex to implement and configure. For this reason, there are several Hadoop distributions which were developed by companies to facilitate the deployment and creation of a Hadoop production environment [63]. Erraissi et al. [63] performed a comparative study of these distribution where the five most preeminent ones were presented: Cloudera, HortonWorks, MapR, IBM BigInsights and Pivotal HD. From this selection, Cloudera and HortonWorks are the most popular [63]. From these two, Cloudera is the one which presents the best open source distribution regarding utility and functionality. This distribution is called Cloudera Distribution for Hadoop (CDH) [64]. This was the distribution chosen for the Portfolio Management System, namely CDH version 5.13.0. CDH has the bonus of coming with a QuickStart Virtual Machine (VM) with the Linux CentOS [65] Operating System (OS) and CDH pre-installed to facilitate beginning of production. CDH comes with Hadoop 2.6.0, which includes HDFS, and includes a Spark installation. Nevertheless, the Spark version used was 2.3.0 and not the included 1.6.0 version, since after Spark 2.0 the main Spark API changed from RDD to DataFrame and DataSet JVM objects [66]. Both are higher level representations of RDD, but DataFrame is more interesting for this work due to its inbuilt SQL utilities. Regarding further Hadoop HDFS configurations, in line with the discoveries made by Inoubli et al. [34], regarding the impact of HDFS block size on the runtime of an iterative process w.r.t. Spark, the chosen block size was 16MB.

C. Data Module

The Data Module is responsible for providing access to the data files stored in Hadoop HDFS and to their main memory distributed representation, the Spark DataFrame object. The DataFrame abstraction was chosen over the low-level RDD representation since its data is in a tabular format, with named columns, imposing structure in the distributed dataset. Additionally, it provides easy access to SparkSQL functions [66].

The Data Module starts by loading the CSV files containing the financial assets and currency exchange rates historical prices, after being processed by the Download Script, into two Spark DataFrame objects: stocksDF and currencyDF, respectively. At this point, the Data Module starts performing Spark transformations in the stocksDF, since the currencyDF data can be used as is. It starts by dropping the columns Open, Close, High and Low, as AdjClose presents all the price information need by the Portfolio Management System. The Rate of Return (ROR) is the gain or loss of an asset over a specified time window, expressed as percentage of the asset’s cost. Formally, in the case of financial assets, it is defined by Equation 4.1:

\[ ROR = \frac{A_t - A_0}{A_0} \times 100 \]  

(III.1)

Where \( A_0 \) is the initial value of the financial asset and \( A_t \) is the current value of the financial asset.

This attribute is used to compute an asset’s expected return by calculating its mean over an historical time window. The \( A_0 \) is obtained for a preset of six time windows in trading days: 5, 10, 20, 60, 120 and 252. Considering that \( x \) can take the values of the preset time windows, the Data Module takes advantage of SparkSQL Window functions, which allow calculations on a group of rows while still returning a single value for every input row [69], to make several Spark transformations that add the columns:

- \( ROR_x \): which represents the ROR for an asset on the entry’s date, considering that \( x \) is the number of trading days in the time window;
- \( Mean_x \): which represents the average ROR, or the expected return, for an asset on the entry’s date, considering that \( x \) is the number of trading days in the time window.

This can be seen in Table II. Finally, it saves the resulting DataFrame in a Parquet file on HDFS. Apache Parquet is a structured column-oriented file format specially designed for
The crossover, or recombination, operator is similar to how biological reproduction works. With the parents already selected by selection operator new individuals are generated by combining the chromosomes of two parents. There are several types of crossover operators. In this work two were considered due to their straightforwardness: One-Point Crossover, where a random crossover point is chosen and the tails of the two parents are switched and Multi-Point Crossover, which is a generalization of the one-point crossover where several alternating segments of parents are swapped to generate a new individual. After analyzing the work of Jorge Magalhães-Mendes [75], it was apparent that the one-point crossover operator tends to display the best results, as corroborated by Gorgulho et al in their GA work [2]. For this reason, the chosen operator was the one-point crossover.

The mutation operator is responsible for generating a small random tweak in the chromosome genotype to get a new solution. It is applied with probability equal to the defined mutation rate, which should be low, or the GA will simply be a random search. As with the selection and crossover operators there are several types of mutation operators. In this work the chosen was the simple Random Resetting Mutation which randomly chooses a gene and changes its value to a random permissible value [76]. The solution also implements Elitism, which saves the current generation best individual for the next generation, unmutated [2].

The initial population, or generation, is generated randomly. To avoid infeasible solutions w.r.t. Equation 1 the sum of all asset weights must be equal to one. This is achieved by normalizing the weights as follows in Equation 6 [2]:

$$w_i = \frac{w_i}{\sum_{i=0}^{N} w_i}$$

Fitness, or evaluation, functions are used to quantify the quality of a chromosome. They are essential in the selection operator, since only the fittest ones are chosen for crossover. There are two fitness functions:

Evaluate Portfolio Return: creates a fitness score equal to the portfolio’s expected return $R_P$, calculated using Equation 3;

$$Fitness(p) = R_P$$

(7)

Evaluate Portfolio Linear Combination: the fitness score is a linear combination of the portfolio’s expected return $R_P$ and the portfolio’s variance $\sigma_P$ as seen in Equation 4. The linear combination’s default values for coefficients ($a_1$ and $a_2$) for return and variance, respectively, are 0.5.

$$Fitness(p) = a_1R_P + a_2\sigma_P$$

(8)

To satisfy the cardinality, floor and ceiling constraints the simple death penalty methodology for dealing with constraints was implemented. The Optimization Module is the main system component, as it implements the SO GA algorithm to attain an optimized portfolio.

### Table II

<table>
<thead>
<tr>
<th>Asset DataFrame initialization – partial snippet</th>
</tr>
</thead>
<tbody>
<tr>
<td>windowsList = [5, 10, 20, 60, 120, 252]]</td>
</tr>
<tr>
<td>w:WindowSpec = Window.partitionBy(&quot;symbol&quot;).orderBy(col(&quot;date&quot;))</td>
</tr>
<tr>
<td>for (i ← windowsList)</td>
</tr>
<tr>
<td>df = df.withColumn(&quot;adjClose Lag Si&quot;, lag(&quot;adjClose&quot;, i).over(w))</td>
</tr>
<tr>
<td>.withColumn(&quot;ror Si&quot;, (&quot;adjClose&quot; - col(&quot;adjClose Lag Si&quot;)) / col(&quot;adjClose Lag Si&quot;)*100)</td>
</tr>
<tr>
<td>.drop(&quot;adjClose Lag Si&quot;)</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>w:WindowSpec = Window.partitionBy(&quot;symbol&quot;).orderBy(col(&quot;date&quot;)).rowsBetween(-i, 0)</td>
</tr>
<tr>
<td>df = df.withColumn(&quot;mean_Si&quot;, mean(col(&quot;ror Si&quot;))).over(w)</td>
</tr>
<tr>
<td>end for</td>
</tr>
</tbody>
</table>

The Hadoop ecosystem, inspired by Google’s Dremel [70]. It was designed to be query efficient as it reads only needed cells on the DataFrame by using vertical and horizontal partitioning [71]. This persistence on the file system is needed for two reasons: to speed up further runs of the application by skipping the transformation process and to clear the DataFrame’s underlying RDD lineage.

### D. Optimization Module

The Optimization Module is the core of the Portfolio Management System. It is responsible for solving a MPT-based optimization problem with the following constraints: cardinality, floor and ceiling. The chosen chromosome representation for the problem dictates that an individual representing a portfolio consists in an array of $n$ assets and their respective $W$ weights. This chromosome representation is depicted in Table III. The use of every attribute might seem to impact memory management negatively. Nevertheless, the impact is negligible since the complete dataset of assets is already cached in a DataFrame in memory by the Data Module.

The selection operator is responsible for defining how the GA will choose the individuals who will be the parents of the next generation. There are several selection operators available for a GA but truncation selection is the fastest. The tradeoff is that it is the selection operator that disallows the most amount of information variation, which could make the solution a local maximum instead of a global maximum [72]. Truncation Selection sorts the population by fitness and then drops the lowest ones according to the truncation threshold. In this system the truncation threshold is set to 0.5, which considers the top half of the population as parents. After obtaining the set of parents to choose from, pairs of parents must be chosen to generate new offspring via crossover. The selection of parent pairs is done via Roulette Wheel Selection, or Fitness Proportionate Selection. This selection method maps the chromosomes fitness values to a probability, generates a random probability – the roulette’s fixed point – and then iterates over the chromosomes map calculating the cumulative probability. When the cumulative probability is bigger or equal to the random probability, the iteration stops, and that chromosome is chosen [72].
IV. EXPERIMENTAL VALIDATION

In this Section the performance metrics used to validate the Portfolio Management System are explained. System results need to be understood considering previous solutions and to verify that the goals of this work were achieved. The Portfolio Management System was evaluated according the following performance metrics:

- Return: the solution portfolio return was evaluated via the indicator ROI;
- Computation time/runtime: the processing time needed to obtain the portfolio solution. The validation comprehends studying how several configurations impact runtime, namely regarding:
  - Concurrency: how varying the number of used cores impacts portfolio computation time;
  - Iterative processing: how varying the number of iterations impacts runtime;
  - Memory: how changing the amount of main memory available to the Spark Portfolio Management Application impacts runtime.

A. Simulations and Environment

There are several input parameters which the user must decide upon before starting the Portfolio Management System. For each simulation a table presenting input parameters is presented. Additionally, the fitness function used is stated.

The environment created to support the simulations encompasses two machines. The first machine launches a VM with CDH 5.13.0 installed. This machine is used keep the Hadoop ecosystem services running, namely Hadoop HDFS 2.6.0 and Spark 2.3.0. Machine details can be seen in Table IV.

The second machine is where the Portfolio Management System is run. It corresponds to the Spark Driver. All simulations were performed locally in this pseudo-cluster of a single machine, while the CDH machine provided access to the needed Hadoop environment libraries and HDFS, acting as a server. Simulations were run using just the Spark Driver machine detailed in Table V using Spark local mode, that is, all computations are performed at the Driver.

The dataset used for every simulation can be seen in Table VI. This specific dataset was chosen to represent stocks from different continents and countries, traded in different currencies. The addition of volatility ETFs is there to give the algorithm a chance at better results when markets are devaluing (bearish), since the Portfolio Management System does not perform short-selling. The dataset size in the Parquet file format is 324.7MB and has a total of 582 assets. All historical data existent, for each symbol, from 01-06-2006 until 28-06-2018 was considered in this dataset.

1) Computer Performance Simulations

While Tables VII and VIII present the general configuration for testing. To perform the following studies some of these parameters need to vary on a study by study basis, in order to understand their impact on overall system performance. Specific GA parameters like population size and number of generations are proposedly relatively low to avoid capping the machine’s resources, since reaching convergence is not very important for computational performance tests. Additionally, shown results are the average of five executions for each particular study configuration.
a) Concurrency

In the first simulation scenario, the system concurrency, or ability to perform parallel tasks, was evaluated. Since the number of available CPU cores directly affects the system’s ability to perform concurrent tasks, the parameter of interest in this scenario is Spark’s spark.driver.cores parameter. By varying this parameter, the impact of the number of available cores on runtime can be studied. To better evaluate concurrency, the Speedup in latency, or just speedup, of multi-core executions when compared to single-core executions was calculated, as expressed in Equation 9, Where $L_i$ is the latency, or runtime, for each execution. Results are depicted in Table IX.

$$S_{\text{latency}} = \frac{L_{\text{old}}}{L_{\text{new}}}$$ (9)

Fig. 2. Graphic depicting how the number of generations (iterations) impacts average system runtime

b) Iterative Processing

In the second simulation scenario, an evaluation is performed on the system regarding iterative processing with the implemented SO GA. Since each generation represents a new GA iteration, the system parameter of interest in this study is Generations. By varying the Generations parameter, the impact of the number of iterations on runtime can be studied. The results are depicted in Figure 2. Another interesting study which was performed with this second scenario – analyzing the evolution each iteration runtime for 50 iterations (generations). These results can be seen in Figure 3.

c) Memory

The third and last computer performance study scenario evaluates how varying the amount of main memory available to the Spark Driver affects the system’s runtime. Spark memory allocation has two main usages: execution, where data being processed is buffered such as, for shuffles, joins, aggregations and sorts; and storage, used to cache recurrently used RDDs. How contention between these two memory usages is managed and overall memory size can have an impact on Spark’s performance, according to Inoubli et al. [34]. Thus, the parameter of interest in this simulation is Spark’s spark.driver.memory parameter which defines the amount of memory allocated to the Spark Driver in the JVM heap. The impact of varying the spark.driver.memory parameter on runtime is depicted in Figure 4.

2) Financial Performance Simulations

In this Section, several tests regarding how Portfolio Management System and Spark parameter configurations affect the ROI metric are detailed. Two simulations will be made: one during a bullish market period and the last one in a sideways market. Bearish markets are omitted, since the only mechanism the system has to deal with these markets are volatility ETFs which were unavailable in the last global bear market, in the 2008 crisis.

For the following studies, the system general system configuration is detailed in Table X and the Spark configurations remain the ones shown in Table XI. Additionally, shown results are the average of five executions for each case study configuration. It is also worthy to note that the user profile is of a risk-taker, since the weights of the linear combination of return and variance are 90% and 10%, respectively.

While Table X presents the general configuration for testing, to perform the following studies Date and Time Window need to vary on a study by study basis, in order to understand their impact on system performance. Both metrics will be evaluated
at the moment of portfolio creation and then after the following four time windows in trading days: 5, 10, 20 and 60.

System results will be compared with a random portfolio (assets and weights chosen randomly), an equally distributed buy and hold portfolio (comprised of the biggest earners in the analyzed time window) and with SPY, an ETF that mimics the growth of S&P500.

a) Case Study 1 – Sideways Market

The system’s portfolio is created at 02-01-2018, and then analyzed overall several windows until 02-04-2018. A ROI comparison between mean of achieved portfolios, best portfolio, SPY, Buy and Hold and Random portfolios is depicted in Figure 5. Finally, a graphic showing how fitness evolution behaved in the best GA portfolio attained in this case study can be consulted in Figure 6.

b) Case Study 2 – Bullish Market

The system’s portfolio is created at 03-01-2017, and then analyzed overall several windows until 03-04-2017. Next, a ROI comparison between mean of achieved portfolios, best portfolio, SPY, Buy and Hold and Random portfolios is depicted in Figure 7. Finally, a graphic showing how fitness evolution behaved in the best GA portfolio attained in this case study can be consulted in Figure 8.

3) Overview and Discussion

Computer performance was first validated regarding concurrency. There is a clear gain in increasing the number of processing cores available to the system, with a particular large increase from two to three processing cores, yielding more than 60% speedup with that increase, as seen in Table 5.6. Secondly, iterative processing performance was validated. It is apparent that the number of iterations has no significant influence on runtime, since Figure 2 shows that the system runtime is characterized by a linear slope. Concerning each iteration’s runtime, the evolution study in Figure 3 shows that it remains relatively constant, with exception of the initial two generations. This might be due to random population initiation in the first generation. Finally, regarding memory size available to the launched Spark application, results can be seen in Figure 4. Despite Inoubli et al. [34] results, which imply that memory size has a dramatic effect on runtime, our study demonstrates that in, for this application design and dataset size, it does not. The memory size appears to have no impactful influence on runtime in local mode, as long as it is sufficient to cache and process the dataset.

As for financial performance, in the ROI analysis, seen in Figures 5 and 7, the best portfolio manages to beat both the Buy and Hold strategy and SPY in the first two weeks after optimization, reaching ROIs up to 2% above those of SPY. This suggest that this period, between the first and second week, is ideal for a re-optimization of asset allocation. The mean portfolio also attains promising results, beating SPY in both case studies and buy and hold in case study 2. These results show that, although financial performance is adequate, there is still margin for growth. Future studies should include other management models, like TA and FA, additional constraints and multi-period considerations, with periodic rebalancing.
Results are influenced by the used GA configurations and the used user profile: a user who favors returns over volatility. Another fact that might influence these results is convergence, or how close the algorithm is w.r.t. the globally optimal solution. In Figures 6 and 8 fitness grows at a small rate towards the end of the 100 generations. Nevertheless, there are big jumps in the graph in terms of fitness. These jumps represent the GA getting outside the scope of a local maximum and finding a new best candidate to explore. Since 100 generations is a traditional low number of generations, results could improve as the number of generations increases, possibly reaching the global maximum. Larger population sizes could also lead to faster convergence.

V. CONCLUSION

The experimental validation described in Section 5 denotes that portfolio management using a GA can be successfully implemented in a Big Data framework, while attaining average computer performance results – regarding runtime in a single machine – but with great concurrency, which is very promising for scalability. Financial performance results were promising with the best run attaining higher ROI than both the SPY ETF and buy and hold strategy in the first 10 trading days after optimization, with falling performance thereafter. The mean of every run also has higher ROI than SPY, consistently, in the first 10 trading days.

Despite the results attained with this system, there is a lot of studying left to do in this type of approach to portfolio management, namely regarding the use of distinct portfolio management models and regarding scalability and optimization of computation time. However, this work provides the reader with a suitable knowledge base regarding portfolio management, genetic algorithms and Big Data frameworks and, as such, it is a good starting point for further exploration in this field.

There could be contention regarding the pertinence of using Big Data frameworks for historical price analysis due to traditional dataset sizes. However, with a current estimation of 630 thousand tradable companies worldwide, excluding commodities, bonds and other asset types, and knowing that there will be increasingly more available historical data, traditional systems will eventually be unable to cope with the scale of computations and memory required. Simple calculations show us that considering 90 bytes per CSV file data entry and 30 years of data (252 trading days per year), a dataset considering the aforementioned amount of companies would amount to approximately 400GB. The use of a Big Data framework, when hardware resources to mount a cluster are available, essentially future proof a portfolio management system, regarding increasing historical dataset sizes, while maintaining moderate execution times. Additionally, it allows for easy transition into real-time applications due to stream processing capabilities.

REFERENCES