The Design and Implementation of a High-performance Portfolio Optimization Platform

Yidong Chen, Zhonghua Lu and Xueying Yang

October 20, 2020

The Design and Implementation of a High-performance Portfolio Optimization Platform

1st Yidong Chen
Computer Network Information Center
Chinese Academy of Sciences
University of Chinese
academic of sciences
Beijing, China
chenyidong@cnic.cn

2nd Zhonghua Lu
Computer Network Information Center
Chinese Academy of Sciences
Beijing, China
zhlu@sccas.cn

3rd Xueying Yang
Computer Network Information Center
Chinese Academy of Sciences
University of Chinese
academic of sciences
Beijing, China
yangxueying@cnic.cn

***Abstract*—Aiming at the high complexity of parameter optimization for portfolio models, this paper designs a distributed high-performance portfolio optimization platform(HPPO) based on parallel computing framework and event driven architecture. The platform consists of the data layer, the model layer, and the excursion layer, which is built in a component, pluggable, and loosely coupled way. The platform adopts parallelization acceleration for backtesting and optimizing parameters of portfolio models in a certain historical interval. The platform is able to docking portfolio model with real-time market. Based on the HPPO platform, a parallel program is designed to optimize the parameters of the value at risk(VAR) model. The performance of the platform are summarized by analyzing the experimental results and comparing with the open source framework Zipline and Rqalpha.**

***Index Terms*—high-performance computing, portfolio optimization, portfolio backtesting, periodic data, parameter optimization.**

I. Introduction

Portfolio optimization refers to optimizing the parameters of a certain portfolio models by simulating buying and selling of underlying assets based on historical market data. The parameters of a portfolio model are optimized through a historical interval by maximizing the Sharpe ratio or minimizing the maximum retracement and so on. Portfolio optimization is conducive to assisting asset managers to build portfolios with higher profitability, risk tolerance and stability [1].

In practical application, the fund managers tend to optimize the parameters of a portfolio model dynamically according to the real-time market data, which can help fund managers adjust their portfolios as the market changes [2] [3]. For example, in dynamic risk monitoring of investment portfolios, in order to maintain the stability of returns during investment management, restrict and monitor extreme risks, investment managers usually need to perform dynamic asset allocation and portfolio management throughout the process while taking into account dynamic risk monitoring and adjust model parameters [4] [5] [6] [7]. This process requires usually minute-level data in practical applications, preferably second-level data or even tick data [8]. Existing software packages such as Zipline [9] and Rqalpha [10] usually take minutes or even hours to solve such problems, which takes too long to meet the actual needs. For another example, In the mean-variance model, the mean vector and the covariance matrix can be easily estimated by the history data, statistically. However, the parametric matrix D of the possibility distribution is much more difficult to estimate [11]. Traditional software packages require a lot of time to carry out a round of calculations, let alone the optimization of the model parameters. Therefore, it is necessary to accelerate the solution of the model in parallel to reduce the time cost by backtesting [12]. Besides, we need to develop more efficient parameter optimization methods for different portfolio models. Based on these issues, we propose an efficient portfolio optimization framework and develop a portfolio optimization platform named HPPO which can meet the following needs [13] [14]

- Requirements for solving large-scaled portfolio models efficiency with minute-level or tick data at a given moment.
- Requirements for optimizing portfolio models with history market data during a certain interval of time.
- Requirements for parallel computation for large-scaled portfolio optimization problems.

In building the HPPO platform, we focused on three key issues as listed below:

- Distributed computing for portfolio optimization. We focus on how to use Monte Carlo methods to generate approximate solutions to portfolio optimization problems [15], how to do solve large-scale portfolio optimization problems in a parallel environment, and how to design a high-performance portfolio backtesting framework. We use a three-dimensional topological structure diagram to describe the overall layout of the framework in detail.
- Inter-process communication and task scheduling deployment. We focus on how to transfer information and data between different modules effectively, how to enable the platform to call the computing cluster, and how to allocate different computing resources to different models. We

This research was supported by National Natural Science Foundation of China (61873254)

use Mesos to take charge of the management of the underlying infrastructure resources and implement the scheduling and use of resources by writing a framework to allocated resources based on Mesos.

- Simulation of transactions and docking portfolio model to the real-time market. The platform is built by event-driven architecture which can not only simulate the real trading environment but also, more importantly, integrate the optimized model directly into the real-time market without any modification and achieve the seamless connection between the historical simulation and the real-time trading.

II. The Architecture of the HPPO Platform

A. The Topography Structure of the HPPO Platform

In this section, the overall layout of the platform is introduced in detail with a three-dimensional topological structure diagram.

In the process of searching optimal parameters for portfolio models, for each group of parameters input to our platform, backtesting is needed. Different portfolio models are based on a different level of data, which will cause problems during data transmission. For instance, some portfolio models are based on daily data, some are based on hourly data or 15 minutes, 5 minutes, 1-minute data, and even tick data. The data volume required by these models is quite different, which makes it difficult for the master processor to broadcast all the market data to the slave processors at one time [16]. The platform solves the problem by slicing the history or real-time data.

For example, if we create a portfolio model based on tick data and the parameters of the model are required to be optimized by using all the tick data of the A-share market in 2019. Then the total volume of the required data is about 45GB. We slice the whole data into weekly data and the master processor broadcasts weekly data to the slave processors, with a data volume of about 860MB per broadcast. For another example, if a portfolio model is based on daily data, we can broadcast the data on an annual basis. The data we broadcast each round is called periodic data.

The parallel computing topology model of HPPO platform is shown in Fig. 1. $P_j^{(k)}$ represents the model parameters stored on the jth processor at the kth round of optimization for $j = 1, \ldots, p$. $T_{ij}^{(k)}$ represents a backtesting task $(D_i, P_j^{(k)})$ when the periodic data is D_i for $i = 1, \ldots, m$ and the model parameter is $P_j^{(k)}$. $F_j^{(k)}$ is the calculation result of the portfolio model in the kth round of optimization with the periodic data D_j.

In the first round of calculations, the platform completes the initialization work F_0 and the main processor assigns task group $T_{11}^{(1)}, T_{12}^{(1)}, \ldots, T_{1p}^{(1)}$ to each slave processors. Each of the slave processors receives the task and performs backtesting. After all the slave processors complete the calculation, the master processor collects the calculation results and updates the periodic data and then the task group $T_{21}^{(1)}, T_{22}^{(1)}, \ldots, T_{2p}^{(1)}$ are broadcast to the slave processors. The

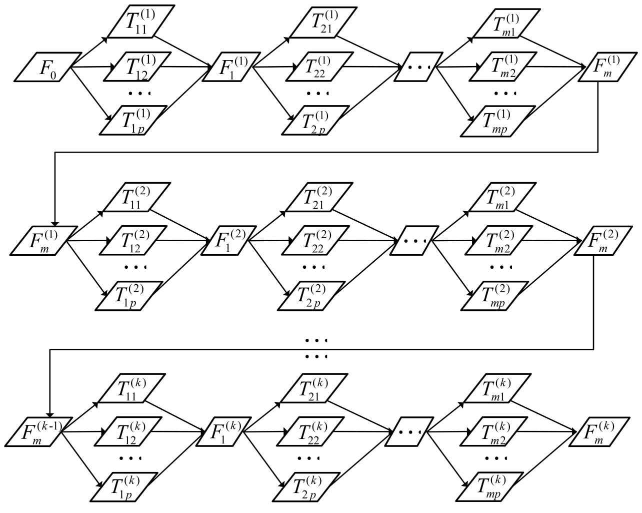

Fig. 1. The parallel computing topology model of the HPPO platform

salve processors continue the backtesting until the task group $T_{m1}^{(1)}, T_{m2}^{(1)}, \ldots, T_{mp}^{(1)}$ is completed, which means a round of parallel computing is completed. The main processor collects the results and saves them as $F_m^{(1)}$. After that, the model parameters are optimized according to the optimization algorithm. Afterward the master processor generates task group $T_{11}^{(2)}, T_{12}^{(2)}, \ldots, T_{1p}^{(2)}$ and broadcasts it to the slave processors for a new round of parallel backtesting. The process above is repeated until the optimization algorithm converges or all parameters are searched.

It is worth noting that when we do parallel parameter optimization for portfolio optimization, we can not parallel the data on the time axis. We parallel the set of model parameters or parallel the model itself. The HPPO platform focuses on the parallelism of parameters rather than the parallelism of the portfolio model itself.

B. The Architecture of the HPPO Platform

The overall architecture of the HPPO platform is mainly divided into three parts: the data layer, the model layer and the execution layer. The data layer is managed by the master processor, which is responsible for data storage, data management and providing data for other layers. The model layer and the execution layer are managed by the slave processors. The model layer is responsible for portfolio model building, backtesting and parallelization. And the execution layer is responsible for executing trading orders. The three parts are independent of each other, which minimizes the degree of coupling between the layers.

As shown in Fig. 2, the communication protocol between the data layer and the model layer is designed with a message passing interface (MPI). Messaging communication involves two basic communication models: point-to-point communication and collective communication. After the slave processors complete the backtesting task, the results are sent to the master processor through point-to-point communication. MPI_Send, MPI_Recv, MPI_Isend and MPI_Irecv all provide point-to-point communication, the first two based on the blocking communication mode and the second two based on the non-blocking communication mode. Because each slave processor needs to wait for the master processor to update periodic data after passing backtesting results to the master processor, the blocking communication model is adopted. Collective communication can be divided into data mobility and data aggregation. The master processor broadcasts the periodic data and model parameters to the slave processors by one-to-many data scatter communication, which is realized by calling function MPI_Scatter. Each slave processor sends the results to the master processor by using many-to-one data protocol communication, which is realized by calling function MPI_Reduce. The master processor is responsible for task allocation and the judgment of algorithm convergence.

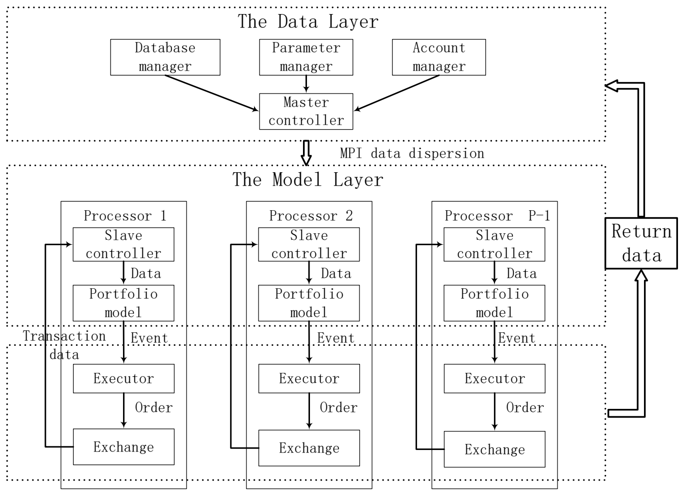

Fig. 2. The framework of the HPPO platform

In terms of the underlying architecture, Event-Driven Architecture (EDA) is adopted. EDA is a way to achieve maximum loose coupling between components or services with events as the medium. Traditional interface-oriented programming uses interfaces as the medium to realize the decoupling between the caller and the interface implementer, but this degree of decoupling is not very high. If the interface changes, the code on both sides needs to change. Event-driven is that the caller and the callee do not know each other, and they are only coupled with the intermediate message queue. The event-driven architecture has three key components: the event producer, the event router, and the event consumer. The producer publishes events to the router, which filters them and pushes them to the consumer. Producer services and consumer services are separate, which allows them to scale, update, and deploy independently.

C. The Operating Mechanism

This section we introduce an example to show the operating mechanism of the HPPO platform. The value at risk(VAR) model [17] calculates the optimal portfolio weight by solving the following equation with the given parameters α and μ:

$$\begin{aligned} \min_{x} \quad & VaR_{\alpha}(w^T x) \\ s.t. \quad & E(w^T x) \geq \mu \\ & \sum_{i=1}^{n} w_i = 1 \\ & w_i \geq 0, i = 1, 2, ..., n \end{aligned}$$

where $w := (w_1, \cdots, w_n)$ is the weight of n risky assets.

The investor seeks a group of parameters α and μ to maximum the absolute return of his portfolio during a certain history period, such as January 1, 2015 through December 31, 2015. The model of this problem can be build by HPPO through the following step:

1) Model parameters setting: The following table shows the parameters of the optimal problem. Those parameters are set by the users which define the optimal problem. The covariance matrix is estimated by the closing price data of the past 30 trading days, which means the data period of the covariance matrix is 20 days.

TABLE I
MODEL PARAMETERS SETTING

Parameter	Setting
portfolio model	VaR
optimization goals	maximum Sharpe Ratio
parameters to be optimized	α, μ
Optimization algorithm	grid search
the range of α	[0.1,0.2]
the range of μ	[0.1,0.2]
investment target	SSE 50
benchmark	Shanghai Stock Index
date range	from 2015/01/01 to 2015/12/31
position adjustment interval	20 trading days
data period of the covariance matrix	20 trading days

2) Periodic data broadcasting and parallel backtesting: Fig. 3 shows how the platform work after defining the optimal problem.

In this example, the volume of daily data is relatively small, we use one year's data as periodic data and assume that the number of slave processors is 2. After selecting the periodic data, the parameters (α, μ) are initialized to $(0.10, 0.10), (0.11, 0.10)$ and broadcasted to the two slave processors, respectively.

The controller 1 receives the periodic data and generates a data update event signal daily. After the event is updated, the model layer receives the data, solves the model and calculates the weight of the portfolio. After the model layer completes the calculation, a position adjustment event is generated and sent to the execution layer. The execution layer places orders with the simulated stock exchange, which returns the trading results. We continue to repeat the above process until all the data in a given time interval (in this example, to December 31, 2015) is updated, the backtesting is completed and the performance index such as Sharpe ratio is obtained.

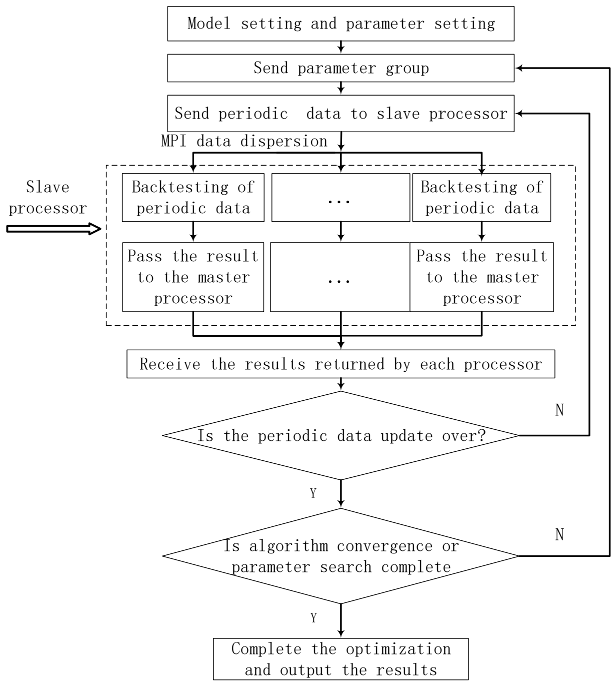
Model setting and parameter setting
Send parameter group
Send periodic data to slave processor
MPI data dispersion
Slave processor
Backtesting of periodic data
...
Backtesting of periodic data
Pass the result to the master processor
...
Pass the result to the master processor
Receive the results returned by each processor
Is the periodic data update over?
N
Y
Is algorithm convergence or parameter search complete
N
Y
Complete the optimization and output the results

Fig. 3. The operational mechanism of the HPPO platform

3) Parameter optimization: After completing the parallel backtesting for a given parameter group, the platform optimizes the parameters. In this example, the grid search is used. In the next round, the two slave processes calculate the performance index of the VaR model when the parameters (α, μ) are $(0.12, 0.10)$, $(0.13, 0.10)$, $\cdots$, $(0.2, 0.10)$, $(0.1, 0.11)$, $(0.11, 0.11)$,$\cdots$, $(0.2, 0.2)$, respectively. The optimal parameters are obtained when all the parameters are searched or the optimization algorithm adopted by the user converges.

III. Implementation of HPPO Platform

A. The Data layer

The data layer is responsible for data management, including data acquisition, data cleaning, data storage and providing data services to other layers. The data layer is divided into historical data module and real-time data module. This distinction is made because the memory management model between real-time data and historical data is quite different. The historical data module needs to extract data from the database and transfer the data to other modules. The real-time data module needs to take into account the real-time data storage and avoid the extra performance overhead caused by it.

1) The Historical Data Module: The historical data module retrieves the historical data from the Tushare or Wind database and then stores the historical data into the Mysql database (row database), and finally compresses the data in Mysql database into a columnar database. Columnar databases are more efficient I/O for read-only queries because they only

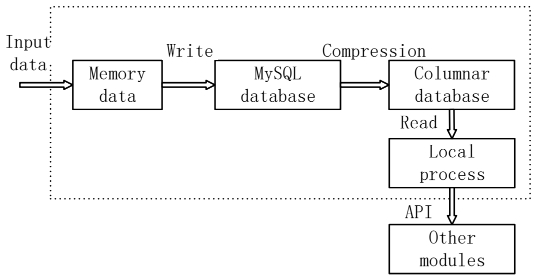

Fig. 4. The historical data module

need to read the attributes accessed by the query from disk (or from memory). Columnar databases are mainly suitable for batch data processing and instant query. Columnar databases have the following advantages: a) Extremely high load speed (up to the sum of all hard disk IO's) b) Suitable for large amounts of data, not small ones. c) Suitable for loading data in real time but limited to increase (deletion and update need to uncompress Block and then recompress storage).

The columnar databases can significantly reduce the performance overhead of other modules when reading data and avoid the entire platform spending a lot of time on data I/O at runtime. One thing to note here is that we do not directly compress the obtained data into a columnar database. The reason is that columnar databases are not suitable for real-time operations that include deletions and updates, that is, columnar databases are not easy to horizontally expand.

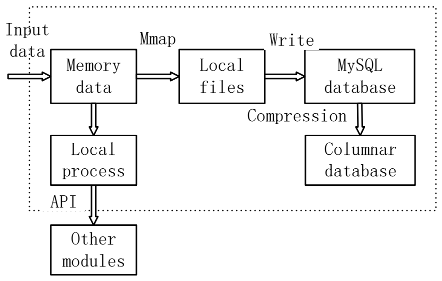

Fig. 5. The real time data module

2) The Real Time Data Module: The real-time data module needs to consider two issues: the management of data in memory and the management of data on disk. We need not only to store the data in memory, but to store the data on disk. One method is to start two threads to write data to memory and local disk respectively, but starting a thread specifically to store data will cause unnecessary performance overhead for the system. To avoid this problem, we use memory-mapped file.

A memory-mapped file associates the contents of a file with a portion of the process's virtual address space, as if the entire file was loaded from disk to memory. Therefore, when using memory-mapped files to process files stored on disk, we no longer have to perform I/O operations on the file, which means it is no longer necessary to apply for and allocate cache for files when processing files, all file cache operations are directly managed by the system. Memory-mapped file plays an important role in handling large data volumes by eliminating steps such as loading file data into memory, writing back data from memory to file, and freeing blocks of memory. We use memory-mapped files to establish a one-to-one correspondence between memory data and local files. Since the memory block and the data on the local disk file are created at the same time, there is no performance overhead in storing the memory data to disk. Besides, we do not process any data on the disk during the operation of the platform. Instead, we write local files to the database after the platform is finished. This avoids the performance overhead caused by storing real-time data. The same data structure are built between the historical data module and the real-time data module to keep the data consistent.

B. The Model Layer

The model layer is responsible for solving the portfolio model with different parameters. The model layer is the core module of HPPO platform which takes up the most computing resources. The model layer not only provides the interface for the user to design the customized portfolio model but also realizes the parameter parallelization and calls the high-performance computing resources to solve the portfolio efficiently.

1) Customized Portfolio Model: HPPO platform allows users to write their portfolio models by providing interfaces such as data interfaces and ordering interfaces. First, the data interface is called to obtain the data; Then the portfolio model is solved according to the data and the portfolio weight is obtained. Finally, the trading interfaces are called to convert the calculation results into order event to complete the placing of orders.

For example, to adjust the weight of the risky assets, the slave processors obtained portfolio data through the interface history_bars and calculate portfolio weight with certain parameters. Then the latest_price is called to query the latest transaction price of the portfolio and get_positions is called to query the position. Finally, the interfaces such as order_target_percent, buy_stock, sell_stock are called to complete the order.

2) Parameter Parallelization: Parameter parallelization is one of the ways for the HPPO platform to optimize parameters. The HPPO platform uses different parallel algorithms to optimize for different portfolio optimization models. For example, for the grid search method, we perform the parallel calculation on different model parameters to reduce the solution time and improve the efficiency of the solution. For more complicated models, we also use a variety of parallel optimization algorithms, such as the parallel genetic algorithm (PGA).

3) High-performance Computing of the Model: The high-performance computing of the model refers to the use of high-performance computing resources for acceleration when solving portfolio models. This requires different high-performance computing solutions tailored to different models. For example, the multi-objective portfolio optimization model is solved by parallelization and a dynamic portfolio hedging model based on the LSTM algorithm can be accelerated by CUDA.

C. The Execution Layer

The execution layer is responsible for executing all trading orders, including buying and selling investment targets, canceling orders and so on. Similar to the data layer, the execution layer is divided into the simulated execution module and the real-time execution module. This division is made because of the difference between simulated trading and real-time trading.

1) The Simulated Execution module: The simulated execution module mainly includes simulated executor and simulated exchange (see Fig. 6 below).

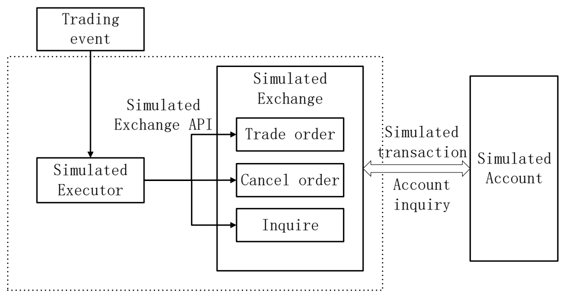

Fig. 6. The simulated execution module

The simulated executor module is responsible for processing order event and the simulated exchange module provides an interface for other modules, which is responsible for simulating order placing, cancellation, query and account operation. Take an order event transfer as an example. The model layer completes the model calculation and generates an order event. The simulated execution module processes the order event and the interface of the simulated exchange module is called to deal with the order event. The simulated exchange module performs simulated actions such as transactions, account inquiries and account operations according to the corresponding event and returns the results to the simulated executor module.

2) The Real-time Execution Module: The real-time execution module is responsible for real-time trading. The real-time execution module is composed of the executor (see Fig. 7 below). The real-time execution module is realized by docking the executor with the exchange. The executor is responsible for processing order events and calling trading interfaces, such as the Shanghai Futures Exchange CTP interface and the Zhongtai Securities XTP interface. If order events are passed from the model layer, the executor calls the interface provided by the exchange to deal with order operations such as placing orders, withdrawing orders and inquiring positions.

In the process of building the HPPO platform, it is necessary to keep the consistency between the simulated exchange interface and the exchange interface.

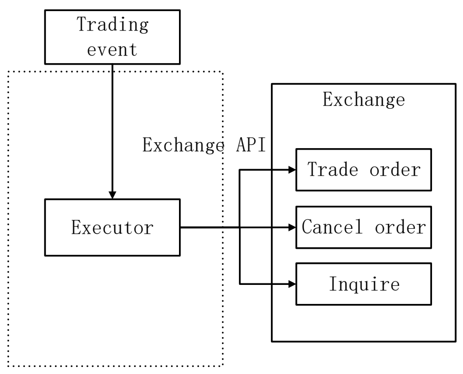

Fig. 7. The real-time excution module

IV. Experiments

A. Experimental Environment and Examples

In this section, experiments were conducted on Sugon's high-performance computer. The server is configured with 24 processors of Intel Xeon CPU E5-2620 v3, 256G memory, and network connection is 10 Gigabit Ethernet. The deployment environment is Ubuntu 16.04.1 LTS, Python 3.7. Based on the HPC environment and the content above, we design the HPPO platform and run tests to compare the performance with Zipline and Rqalpha.

B. Experiment and Analysis

1) Performance Analysis of Serial Computing: Zipline and Rqalpha are two open frameworks that allow users to design portfolio models and run backtesting. However, none of them provides a parallel computing environment. We design a serial program for parameter optimization and compare the running time of Zipline, Rqalpha and HPPO platform. In this experiment, we optimize the parameters μ and α discussed above. The search size of the parameter α is 11 and the search size of the parameter μ is 11, too. The total search size of this optimization problem is $11 \times 11 = 121$.

Table II shows some basic results calculated by the HPPO platform with different parameters (α, μ), which makes it easy to find the maximum shape ratio. The results are the same as those calculated by Zipline and Rqalpha.

The table III shows that the total calculation time of Zipline is 3113.33s and the total calculation time of Rqalpha is 2516.80s. Each calculation takes 25.73s and 20.80s, respectively. The calculation time of the HPPO platform is 1318.90s. Each calculation takes 10.90s. The HPPO platform has significantly reduced the computing time for serial programs compared to Zipline and Rqalpha.

2) Performance Analysis of Parallel Computing: Based on the high-performance computing platform and MPI parallel programming environment, a parallel computing program for multi-parameter optimization of the VaR model is designed and implemented in the HPPO platform. Table IV lists the calculation results of some nodes. In the calculation, we performed 121 parameter groups in parallel and calculated the acceleration ratio and parallel efficiency when the number of nodes was 2, 4, 8, 12 respectively. The acceleration ratio S_p and parallel efficiency η are calculated according to $S_p = T_s/T_p$ and $\eta = S_p/p$ (T_s is the serial calculation time, T_p is the parallel calculation time and p is the number of parallel nodes).

As shown in Table IV, under the current scale of 24 processors, such a smaller-scale portfolio optimization problem in the example achieves an acceleration ratio of more than 8 times and a parallel efficiency of more than 69%. According to Gustafson's law, with the further expansion of

TABLE II
Basic results with different parameters

(μ, α)	**Shape Ratio**	**Alpha**	**Annual Return**	(μ, α)	**Shape Ratio**	**Alpha**	**Annual Return**
(0.1,0.1)	0.2560	0.0113	0.0558	(0.15,0.16)	0.1624	-0.0349	0.0144
(0.1,0.11)	0.2696	0.0171	0.0614	(0.15,0.17)	0.1275	-0.0454	0.0024
(0.1,0.12)	0.1357	-0.0307	0.0136	(0.15,0.18)	-0.1002	-0.1290	-0.0805
(0.1,0.13)	-0.0094	-0.0827	-0.0381	(0.15,0.19)	0.0377	-0.0823	-0.0340
(0.1,0.14)	0.0397	-0.0791	-0.0311	(0.15,0.2)	0.3849	0.0590	0.1051
(0.1,0.15)	0.0026	-0.0877	-0.0406	(0.16,0.1)	-0.1105	-0.1356	-0.0870
(0.1,0.16)	0.0843	-0.0514	-0.0057	(0.16,0.11)	0.0494	-0.0764	-0.0285
(0.1,0.17)	0.1492	-0.0294	0.0161	(0.16,0.12)	0.1772	-0.0309	0.0189
(0.1,0.18)	0.1127	-0.0423	0.0027	(0.16,0.13)	0.0451	-0.0792	-0.0297
(0.1,0.19)	0.2402	0.0054	0.0504	(0.16,0.14)	0.0132	-0.0891	-0.0400
(0.1,0.2)	-0.0253	-0.0916	-0.0455	(0.16,0.15)	-0.0599	-0.1173	-0.0687
(0.11,0.1)	-0.0692	-0.1056	-0.0597	(0.16,0.16)	0.1322	-0.0490	0.0007
(0.11,0.11)	0.1314	-0.0316	0.0128	(0.16,0.17)	0.1562	-0.0389	0.0110
(0.11,0.12)	0.0500	-0.0644	-0.0189	(0.16,0.18)	0.1265	-0.0459	0.0015
(0.11,0.13)	0.0648	-0.0604	-0.0142	(0.16,0.19)	0.1559	-0.0399	0.0104
(0.11,0.14)	0.0947	-0.0497	-0.0038	(0.16,0.2)	-0.0497	-0.1191	-0.0685
(0.11,0.15)	0.1310	-0.0354	0.0093	(0.17,0.1)	0.0433	-0.0821	-0.0325
(0.11,0.16)	0.1340	-0.0381	0.0083	(0.17,0.11)	0.0400	-0.0808	-0.0325
(0.11,0.17)	0.2399	0.0067	0.0503	(0.17,0.12)	-0.0833	-0.1290	-0.0794
(0.11,0.18)	-0.0489	-0.1087	-0.0605	(0.17,0.13)	0.0692	-0.0783	-0.0267
(0.11,0.19)	0.0505	-0.0704	-0.0229	(0.17,0.14)	-0.1146	-0.1475	-0.0957
(0.11,0.2)	0.2586	0.0124	0.0574	(0.17,0.15)	0.2494	0.0001	0.0484
(0.12,0.1)	0.0857	-0.0527	-0.0064	(0.17,0.16)	0.0203	-0.0887	-0.0399
(0.12,0.11)	0.3622	0.0511	0.0962	(0.17,0.17)	0.1135	-0.0533	-0.0042
(0.12,0.12)	0.2024	-0.0095	0.0360	(0.17,0.18)	0.0885	-0.0690	-0.0185
(0.12,0.13)	0.2913	0.0240	0.0689	(0.17,0.19)	0.1504	-0.0409	0.0078
(0.12,0.14)	0.3126	0.0306	0.0764	(0.17,0.2)	0.1621	-0.0364	0.0130
(0.12,0.15)	0.1842	-0.0184	0.0281	(0.18,0.1)	0.0405	-0.0875	-0.0366
(0.12,0.16)	0.2262	-0.0027	0.0437	(0.18,0.11)	0.0941	-0.0643	-0.0143
(0.12,0.17)	0.0568	-0.0634	-0.0173	(0.18,0.12)	0.1281	-0.0434	0.0043
(0.12,0.18)	0.0705	-0.0561	-0.0107	(0.18,0.13)	0.2200	-0.0134	0.0361
(0.12,0.19)	0.0937	-0.0510	-0.0042	(0.18,0.14)	0.0585	-0.0677	-0.0208
(0.12,0.2)	0.2232	-0.0020	0.0435	(0.18,0.15)	0.0996	-0.0563	-0.0081
(0.13,0.1)	0.0219	-0.0840	-0.0362	(0.18,0.16)	0.1562	-0.0360	0.0129
(0.13,0.11)	0.3174	0.0345	0.0790	(0.18,0.17)	0.3035	0.0243	0.0721
(0.13,0.12)	0.0249	-0.0795	-0.0321	(0.18,0.18)	0.1869	-0.0226	0.0250
(0.13,0.13)	0.1103	-0.0476	-0.0012	(0.18,0.19)	0.1896	-0.0280	0.0224
(0.13,0.14)	0.2257	-0.0022	0.0440	(0.18,0.2)	0.0968	-0.0573	-0.0093
(0.13,0.15)	0.0409	-0.0702	-0.0247	(0.19,0.1)	0.3483	0.0428	0.0899
(0.13,0.16)	0.3279	0.0345	0.0822	(0.19,0.11)	0.2341	-0.0029	0.0443
(0.13,0.17)	0.2593	0.0099	0.0560	(0.19,0.12)	0.1830	-0.0268	0.0215
(0.13,0.18)	0.2314	0.0013	0.0462	(0.19,0.13)	0.1479	-0.0393	0.0093
(0.13,0.19)	0.1943	-0.0143	0.0319	(0.19,0.14)	0.0383	-0.0800	-0.0318
(0.13,0.2)	0.0675	-0.0635	-0.0158	(0.19,0.15)	0.2991	0.0214	0.0692
(0.14,0.1)	0.2298	-0.0004	0.0454	(0.19,0.16)	0.0106	-0.0926	-0.0442
(0.14,0.11)	0.0077	-0.0916	-0.0429	(0.19,0.17)	0.3120	0.0248	0.0743
(0.14,0.12)	0.0575	-0.0603	-0.0159	(0.19,0.18)	0.0236	-0.0799	-0.0339
(0.14,0.13)	0.1000	-0.0514	-0.0043	(0.19,0.19)	-0.0513	-0.1169	-0.0677
(0.14,0.14)	0.0873	-0.0596	-0.0120	(0.19,0.2)	0.1412	-0.0382	0.0096
(0.14,0.15)	-0.1153	-0.1348	-0.0868	(0.2,0.1)	0.2579	0.0044	0.0523
(0.14,0.16)	0.0077	-0.0953	-0.0452	(0.2,0.11)	0.1524	-0.0365	0.0112
(0.14,0.17)	0.2368	0.0008	0.0474	(0.2,0.12)	0.3010	0.0249	0.0710
(0.14,0.18)	0.1363	-0.0410	0.0064	(0.2,0.13)	0.0035	-0.0979	-0.0489
(0.14,0.19)	0.0832	-0.0654	-0.0171	(0.2,0.14)	0.1413	-0.0408	0.0073
(0.14,0.2)	-0.0177	-0.1009	-0.0536	(0.2,0.15)	0.2794	0.0143	0.0618
(0.15,0.1)	0.0021	-0.0943	-0.0454	(0.2,0.16)	0.2241	-0.0080	0.0396
(0.15,0.11)	0.0818	-0.0590	-0.0124	(0.2,0.17)	0.2449	0.0005	0.0482
(0.15,0.12)	0.2165	-0.0101	0.0376	(0.2,0.18)	0.2248	-0.0073	0.0409
(0.15,0.13)	-0.0490	-0.1104	-0.0624	(0.2,0.19)	-0.0019	-0.0966	-0.0490
(0.15,0.14)	-0.0281	-0.1055	-0.0571	(0.2,0.2)	0.2340	-0.0053	0.0433
(0.15,0.15)	-0.1251	-0.1399	-0.0910				

TABLE III
The Serial Result

Experiment Platform	**Parameter Optimization Scale**	**T/s**
HPPO	121	1318.90
Rqalpha	121	2516.80
Zipline	121	3113.33

the optimization scale, the acceleration ratio and the increase in parallel efficiency will become more significant.

TABLE IV
THE PARALLEL RESULT

Number of Nodes	**T/s**	S_p	$\eta/(\%)$
1	1318.90	1.00	100.0
2	679.85	1.94	96.96
4	387.91	3.40	84.99
8	221.29	5.96	74.52
12	158.33	8.33	69.38

V. CONCLUSION

This paper designs a distributed high-performance portfolio optimization platform(HPPO) based on a parallel computing framework and event-driven architecture with the aim of parameter optimization for portfolio models. The platform is built in a way that is high-performance, loosely coupled, and scalable. We take parallel computing and high-performance computing resources as the core and build the HPPO platform. HPPO platform can realize the parameter optimization for complex portfolio models with high efficiency, meeting the actual needs of fund managers for adapting portfolio models to real-time market. This paper takes the VaR model as an example to design and implement a serial program and a parameter program for parameter optimization and compares the performance of the HPPO platform with the open frameworks Zipline and Rqalpha. The results show that the HPPO platform not only has a significant improvement in serial programs compared with traditional platforms but also has a better performance when running in a parallel environment. At present, the optimization algorithm for parameter optimization on our platform mainly includes grid search, random search and parallel genetic algorithms. In the future, more optimization algorithms will be integrated into the HPPO platform for parameter optimization such as Particle Swarm Optimization(PSO), Grey Wolf Optimizer (GWO) and Bayesian optimization and so on.

REFERENCES

[1] R. O. Michaud, "Efficient asset management: A practical guide to stock portfolio optimization and asset allocation," *Oup Catalogue*, vol. 14, no. 3, pp. 901–904, 2008.

[2] A. Duran and G. Caginalp, "Parameter optimization for differential equations in asset price forecasting," *Optimization Methods and Software*, vol. 23, no. 4, pp. 551–574, 2008.

[3] S. T. Rachev, J. S. J. Hsu, B. S. Bagasheva, and F. J. Fabozzi, *6. Bayesian Framework for Portfolio Allocation: The Mean-Variance Setting*. John Wiley and Sons, Inc., 2015.

[4] D. K. Saxena and K. Deb, "Non-linear dimensionality reduction procedures for certain large-dimensional multi-objective optimization problems: Employing correntropy and a novel maximum variance unfolding," vol. 4403, pp. 772–787, 2007.

[5] Y. Xia, "Learning about predictability: The effects of parameter uncertainty on dynamic asset allocation," *Journal of Finance*, vol. 56, no. 1, pp. 205–246, 2001.

[6] N. Vol., "Dynamic asset allocation with event risk," *Journal of Finance*, vol. 58, no. 1, pp. 231–259, 2003.

[7] N. Hibiki, "Multi-period stochastic optimization models for dynamic asset allocation," *Journal of Banking and Finance*, vol. 30, no. 2, pp. p.365–390, 2006.

[8] M. Mnif, "Portfolio optimization with stochastic volatilities and constraints: An application in high dimension," vol. 56, no. 2, pp. 243–264.

[9] J. Jevnik, "Zipline, a pythonic algorithmic trading library," https://github.com/quantopian/zipline.

[10] Ricequant, "Rqalpha, a extendable, replaceable python algorithmic backtest and trading framework supporting multiple securitie," https://github.com/ricequant/rqalpha.

[11] Y. Fang, K. K. Lai, and S. Y. Wang, *Fuzzy portfolio optimization*, 2008.

[12] A. F. Perold, "Large-scale portfolio optimization," *Management Science*, vol. 30, no. 10, pp. 1143–1160.

[13] L. H. Chen and L. Huang, "Portfolio optimization of equity mutual funds with fuzzy return rates and risks," *Expert Systems with Applications*, vol. 36, no. 2-part-P2, pp. 3720–3727.

[14] H. Davari-Ardakani, M. Aminnayeri, and A. Seifi, "Multistage portfolio optimization with stocks and options," *International Transactions in Operational Research*, vol. 23, no. 3, pp. 593–622.

[15] W. A. Lodwick and J. Kacprzyk, *Fuzzy Optimization*, 2009.

[16] E. Laure and H. Moritsch, "Portable parallel portfolio optimization in the aurora financial management system," vol. 4528, pp. 193–204, 2001.

[17] S. Alexander, T. Coleman, and Y. Li, "Minimizing cvar and var for a portfolio of derivatives," vol. 30, no. 2, pp. 0–605.