Big Data Research 25 (2021) 100203

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Cost Optimization for Big Data Workloads Based on Dynamic Scheduling and Cluster-Size Tuning

Marek Grzegorowski [a,*], Eftim Zdravevski [b,c], Andrzej Janusz [a], Petre Lameski [b], Cas Apanowicz [c], Dominik Ślęzak [a]

[a] Institute of Informatics, University of Warsaw, Poland
[b] Faculty of Computer Science and Engineering, Sts Cyril and Methodius University, Skopje, Macedonia
[c] CogniTrek Corp., Toronto, Canada

ARTICLE INFO

Article history:
Received 26 May 2020
Received in revised form 20 September 2020
Accepted 15 January 2021
Available online 2 February 2021

Keywords:
Big Data
ETL
Cloud computing
Spot price prediction
ARIMA
Spark

ABSTRACT

Analytical data processing has become the cornerstone of today's businesses success, and it is facilitated by Big Data platforms that offer virtually limitless scalability. However, minimizing the total cost of ownership (TCO) for the infrastructure can be challenging. We propose a novel method to build resilient clusters on cloud resources that are fine-tuned to the particular data processing task. The presented architecture follows the infrastructure-as-a-code paradigm so that the cluster can be dynamically configured and managed. It first identifies the optimal cluster size to perform a job in the required time. Then, by analyzing spot instance price history and using ARIMA models, it optimizes the schedule of the job execution to leverage the discounted prices of the cloud spot market. In particular, we evaluated savings opportunities when using Amazon EC2 spot instances comparing to on-demand resources. The performed experiments confirmed that the prediction module significantly improved the cost-effectiveness of the solution – up to 80% savings compared to the on-demand prices, and at the worst-case, 1% more cost than the absolute minimum. The production deployments of the architecture show that it is invaluable for minimizing the total cost of ownership of analytical data processing solutions.

1. Introduction

The ability to analyze the available data is a valuable asset for any successful business, especially when the analysis yields meaningful knowledge. To represent the data in a format suitable for such analysis, several steps need to be performed first. Extract-Transform-Load (ETL) is recognized as the most time-consuming and expensive among them. The recurring nature of this process, usually at daily intervals, justifies the effort of cost optimization. It is a common practice in the industry to use the cloud computation resources for performing repetitive ETL tasks. On the other hand, cloud providers aim to optimize server utilization to avoid idle capacity and significant peaks [1]. This led to the emergence of cloud spot markets on which service providers and customers can trade computation power in near real-time. The number of available pricing models on the cloud markets is overwhelming, but it is worth paying special attention to two of them, in particular: the on-demand and spot markets. The first one represents the pay-as-you-go cloud model, and today is the most common way the resources are provisioned. The second one allows customers to save up to 90% of costs by using the cloud data centers' idle servers.

One of the evident concerns regarding the spot model is that prices fluctuate along with changes in supply and demand. Furthermore, cloud providers may terminate provisioned instances with a minute notice due to outbidding. The ability to forecast future spot prices in a time horizon necessary to complete ETL tasks would be a game-changer allowing to decrease total costs of operation of data processing pipelines significantly and to minimize the risk of resource terminations. In practice, typical ETL tasks have a degree of temporal flexibility - they need to be done before a specific deadline, however, it is often possible to defer the computations if it could lead to overall cost reduction due to price fluctuation. With a reliable forecasting model that provides accurate spot price prediction for a given time horizon, and reliable estimation of resources required to perform the task, one could recommend an efficient and cost-effective cluster configuration. Currently, there is no comprehensive cloud-based architecture that meets such requirements.

* Corresponding author.
E-mail addresses: m.grzegorowski@mimuw.edu.pl (M. Grzegorowski), eftim@finki.ukim.mk (E. Zdravevski), a.janusz@mimuw.edu.pl (A. Janusz), lameski@finki.ukim.mk (P. Lameski), cas@cognitrek.com (C. Apanowicz), slezak@mimuw.edu.pl (D. Ślęzak).

https://doi.org/10.1016/j.bdr.2021.100203
2214-5796/

In our study, we propose a novel method to build resilient clusters on the cloud resources that are fine-tuned to the individual data processing task. The presented architecture follows the infrastructure-as-a code paradigm so that the cluster configuration could be dynamically managed [2]. We evaluate savings opportunities due to the utilization of Amazon EC2 spot instances comparing to the on-demand ones. We also enhanced the architecture presented in [3] with the spot price prediction module. The performed experiments confirmed that our solution selects an appropriate and cost-effective cluster configuration to execute the scheduled tasks. On the one hand, accurate price forecasting allows us to estimate the required budget reliably. On the other hand, we minimize the risk of losing resources due to the potential outbidding of the provisioned cloud resources.

In this paper, we thoroughly extend our previous research on cluster-size optimization within a cloud-based ETL framework for Big Data [4]. In several production deployments of our previously introduced solution, we noticed that it is imperative to extend it with cost optimization abilities. Therefore, we investigated methods suitable for sport price prediction, we evaluated them experimentally, and we proposed an elegant way to embed them into the appropriately adjusted architecture of our solution. The main contributions of this paper, compared to the previous study, are as follows:

1. We extended the cloud architecture with a possibility to dynamically schedule resources from various cloud pricing models.
2. We trained several spot price prediction models to forecast future prices.
3. We performed an extensive experimental study on real data from the AWS cloud to confirm the feasibility of short-term spot price prediction.
4. We extended the cluster-size optimization mechanism with a module based on machine learning (ML) that recommends the most cost-effective EC2 resources, taking into account current on-demand prices and predicted spot prices.
5. We performed a broad cost analysis that confirmed the cost-effectiveness of the described solution

The rest of the paper is organized as follows. In Section 2, we review the related approaches to resource management and scheduling, topics related to ETL in cloud, big data, as well as practical price forecasting applications on various markets. In Section 3, we describe the architecture of the solution. Subsequently, we describe the methods in Section 4. Next, the experimental results are shown and discussed in Section 5. Finally, in Section 6 we conclude the paper and discuss future directions for research.

2. Related work

There are various challenges in delivering efficient, cost-effective, scalable, and reliable ETL frameworks for Big Data. The solution involves an interdisciplinary study, associated with a wide range of topics – from infrastructure and software related to data processing and machine learning applications. Therefore, in this section, we briefly review recent research results and appropriate technologies associated with the related challenges. We first discuss ETL challenges and solutions for Big Data. Then we focus on distributed storage and processing solutions. Further, we shift our focus on the cost-effectiveness challenges on the cloud. Finally, we review approaches that apply ML-based forecasting of prices.

In classical Business Intelligence (BI), the ETL process loads data into warehouse servers [5]. For reasonable data volumes, there are ETL tools that were successfully used in organizations throughout the years, such as Informatica, IBM Infosphere Datastage, Ab Initio, Microsoft SQL Server Integration Services (SSIS), Oracle Data Integrator, Talend, Pentaho Data Integration Platform (PDI), etc. Cloud-native frameworks for ETL, such as Amazon Glue, Microsoft Azure Data Factory, Panoply, and Snowflake, also emerged. Big Data ETL processes, and recently Data Lake solutions, rely on distributed storage for effective processing of often semi-structured or unstructured data. The Hadoop Distributed File System (HDFS), Amazon S3, Azure Blob Storage, and Google Cloud Storage are the most popular distributed file systems used today. Traditional ETL deployments do not consider data partitioning [6] that the distributed file systems offer, but recent cloud solutions such as Amazon Redshift [7,8], Amazon Athena [9], Azure Synapse Analytics [10], Hive [11], HBase [12], Amazon Redshift [8], or Infobright [13], leverage the distributed file systems and have the capability to execute ad-hoc queries. It is important to note that the execution of queries against such databases relies on MapReduce, Spark, or Tez [14] jobs. Furthermore, the synchronous dataflow execution model of MapReduce and Spark limits the use of asynchrony for complex analytics. Approaches, such as [15], attempt to bridge this gap by proposing asynchronous architectures.

Once the data is processed and loaded into a data warehouse, it needs to be available for reporting and interactive visualization even for large data sets [16,17]. Likewise, analytics and decision support [18,19] are critical approaches. The typical applications of the state-of-the-art machine learning techniques also require proper data pre-processing, ranging from clustering [20], and including the expensive process of representation learning or feature extraction [21–24]. A cost-effective approach to scale this process for Big Data [25] is crucial for the efficiency of applications in many domains [26,27].

Apache Spark focuses on the class of applications that reuse a working set of data across multiple parallel operations and allows all ETL processes to be distributed across various nodes and all transformations to be performed on distinct portions of data [28,29]. It includes libraries and components for data processing, machine learning, or data mining [30]. Additionally, it hides the complexities related to parallelism, fault-tolerance, and cluster setting from end-users and application developers [31].

Cloud computing has emerged as an essential paradigm offering convenient Big Data solutions related to scalable storage, processing, and sophisticated business analytics. Simultaneously, because of the distributed architecture, it also raises new challenges [32] related to synchronization, replication, scheduling, or security [33]. Due to the growth of Big Data over the cloud, cost-effective allocation of appropriate resources has emerged as a significant research problem [34].

The proper allocation of cloud resources is a challenging task, particularly for computationally cumbersome processes of data transformation. There are quite a few examples of cluster size optimizations for Big Data analytics that focus on resource management for sustainable and reliable cloud computing [35]. One of the approaches could rely on initial estimations of data stream characteristics expressed in a vector termed Characteristics of Data (CoD). Clusters of cloud resources could then be created dynamically with the help of, e.g., Self-Organizing Maps [34,36]. Another approach – presented in [37] – focuses on the optimization of short-running jobs. Authors in [38] propose a query-like environment where developers can query for the required cluster size. The proposed approach requires, however, implementation-specific details. The evaluation of historic executions and metrics, for each step of the ETL scenario, is one of the prominent methods that leads to proper cluster size optimization, resulting in the timely processing of data [3].

Some of the frameworks for cluster size optimization, to minimize the deployment cost, consider allocating server time to spot cloud resources. For that purpose, a fine-tuned heuristic to auto-

mate application deployment and a Markov model that describes the stochastic evolution of the spot price and its influence on virtual machine reliability are proposed [39]. In [40], the authors describe an integral framework for sharing time on servers between on-demand and spot services. This is one way to guarantee that on-demand users can be served quickly while spot users can stably use servers for an appropriately long period. This is a critical feature in making both on-demand and spot services accessible. However, guaranteeing timely cloud job execution on a spot instance is a very challenging task, and existing strategies may not fulfill requests in case of outbidding.

Changes in supply and demand are the primary factor that impacts the price of a given service. This behavior is well-known in the stock exchange or commodity markets [41,42]. Among many available methods for time series regression [43] – which are the most suitable for modeling the problem of price prediction – one of the most popular and broadly used are autoregressive integrated moving average (ARIMA) models [44,45]. Results obtained in this study confirmed that the ARIMA has a strong potential for short-term spot prediction.

The ability to accurately forecast future spot prices is essential to minimize the risk of resource terminations. A number of models have already been applied for that task [1]. For example, in [46], the authors evaluated a model to predict EC2 spot prices based on long/short-term memory recurrent neural networks. The problem of forecasting EC2 spot prices one day and one week ahead was also evaluated with random forest regressors [47]. Because of the similarity between cloud spots and financial markets [42], we decided to assess ARIMA models [44], which are known to be robust and efficient in short-term time series forecasting on stock exchanges or commodity markets [41,48].

The proposed cluster-size optimization algorithm works without the need for implementation details. Furthermore, our architecture facilitates data processing in different ETL scenarios, making it more versatile and applicable in many practical cases [21,49]. The proposed mechanism facilitates an optimal selection of cloud computing resources available in the spot pricing model yielding outstanding cost-savings. Our experimental study confirmed that the ARIMA forecasting models have a strong potential for short-term spot price prediction. The simulations carried out proved that any further optimization in this area could improve the achieved cost-effectiveness by at most 1%.

3. Architecture of the framework for ETL of Big Data

Fig. 1 shows the architecture of the utilized ETL system, which is capable of processing data from a variety of data sources. It handles relational database management systems (RDBMS), structured and semi-structured data from internal or third-party data providers, that generate reasonably-sized data. This kind of low-volume data can be processed using traditional Data Integration Tools to store it in the Data Warehouse (marked with light gray arrows in Fig. 1).

The traditional ETL approach is not applicable in Big Data cases with high volume, velocity, or versatility. Therefore, the architecture can utilize Distributed Streaming Platforms (DSP), such as Apache Kafka and Amazon Kinesis, to efficiently collect and process Big Data streams [50]. Because of retention policies, data on DSPs can only be present for a limited time (usually up to two weeks), which is not suitable for long-term storage. Accessing data on a DSP queue can be performed by either push or pull mechanisms [51]. With the pull mechanism, each DSP consumer manages its read pointer.

The proposed architecture allows the consumption of DSP queues by the three most common and widely used types of consumers. The first two types of consumers are redundant alternatives for reliable and permanent storage of the incoming data in raw format. First, with Push Lambda Functions (Stream-based model), event sources publish events on DSP, which trigger the lambda function multiple times per second as data arrives on the queue. The lambda function processes the events [52,53], and the unprocessed raw data can be stored in the original format on Object Storage Services (OSS), such as, Amazon S3 or Windows Azure Blob Storage. This scenario of storing the raw data is commonly referred to as Data Lake. Second, Storage Stream Pullers as consumers have more control in fetching records from DSPs because they manage their read pointer independently, and therefore, can reprocess events if needed (e.g., for recovering after failures). Third, Analytics Stream Pullers, such as Spark Streaming and Apache Storm, are typical consumers that perform stream processing and provide near real-time analytics [52].

Very complex algorithms can be implemented with these technologies to provide valuable business insights and near real-time analytics, and can also store data in the warehouse. Be that as it may, Analytics Stream Pullers can execute algorithms that use only recent data because of DSPs' data retention policies. To complement this, the proposed architecture employs dynamically provisioned Spark clusters for implementing more sophisticated algorithms for ETL and feature engineering. They can analyze dynamic trends over more extended time periods (e.g., week-by-week or month-by-month comparisons of various metrics) or find the time since some particular event happened (e.g., "the time since the last occurrence of event X", "the time since the user's last login", "last bought or viewed product", "last used service", etc.). Such metrics are not computable with Analytics Stream Pullers.

The Cluster Launcher module, located on the same instance, hosts the Data Integration Tool (DIT) and orchestrates the start of Spark clusters, which can be triggered manually, based on a predefined schedule based on a dynamic schedule leveraging spot instance price analysis. After the Spark cluster is created, it downloads the source code from a release branch of a code repository and automatically executes it. Each Spark cluster during its lifetime runs only a specific ETL job. If the organization requires multiple ETL processes of unrelated data, then multiple Spark jobs can be defined, and for each of them, a separate workflow can be managed (i.e., different code repositories, execution schedules, destination data warehouses, etc.).

Apache Spark applications process Big Data stored on the Data Lake (OSS), while also considering the dimensional data from the data warehouse. Generally, the dimensional data does not have to be processed by Spark because usually, it is with considerably lower volumes compared to the transactional data. Consequently, traditional ETL tools can be used for it. However, while processing the transactional data, which is to be stored as fact tables, the dimensional data is still required. If the fact tables are wide, then the dimensional data is a prerequisite for the denormalization. Otherwise, it is needed for setting up foreign keys to the dimensions.

The proposed architecture makes it possible to handle high-velocity Big Data thanks to its three components. First, DSPs apply sharding techniques allowing processing of thousands of events per second. For instance, each Amazon Kinesis shard can process up to 1000 writes per second, and we could add new shards without any downtime. Therefore, the high-velocity data can be transformed into static data stored in OSS with either of the two mechanisms (i.e., pull or push). Second, after the data is on the DSP stream, it is reliably retained up to a predefined period (e.g., up to a week on Amazon Kinesis). Third, different vendors offer reliable and persistent OSS, providing necessary metadata information (such as a list of files, timestamps of creation and modification, object size, etc.) through APIs.

In general, the architecture of production systems could be quite different than the described. For example, Relational Data-

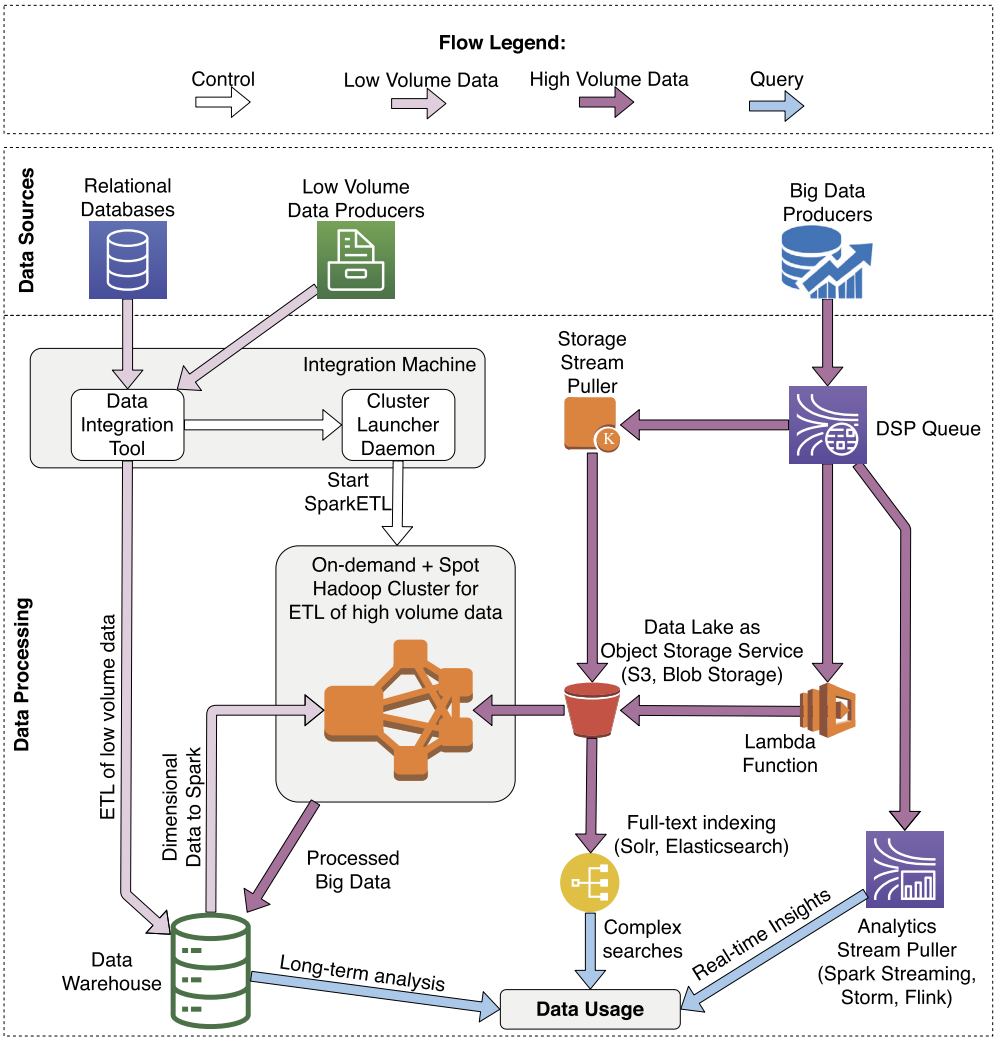


Fig. 1. Architecture of scalable Cost-optimizing cloud-based Big Data Warehouse.

bases, Low Volume Data Producers or Data Integration Tool could be irrelevant, and therefore removed from the architecture. The result of the processing, instead of being kept in a Data Warehouse, could be passed on to another system or stored again in Object Storage Service. In any case, spot prediction methods described later in this paper are independent of the used architecture. There are, however, two major considerations for any prospective architecture that uses the spot instances. First, the data storage layer should be separated from the compute resources, such as in the proposed architecture. Second, to make economical sense of using spot instance predictions, the task needs to be repetitive and divisible between multiple machines.

4. Methods

4.1. Algorithm for cluster size cost-optimization

The proposed architecture consists of multiple components. Some of them, including database machines, Distributed Streaming Platforms and Data Lake storage capacity, mostly depend on the volume of data, so arguably there is little room for cost optimization there. However, the inappropriate choice of Hadoop clusters can result in considerably higher costs. To minimize this cost, two aspects of the ETL process are used. First, we are considering the fact the data volume and maximum time by when the process has to successfully complete are known upfront before the ETL starts. This assumption is realistic because the OSS APIs can be used to estimate the number and size of objects to be processed. Likewise, the end time by which the ETL process has to finish is a business requirement, and it is also known upfront. Second, Hadoop clusters can be dynamically launched when they are needed. In particular, there are two ways of doing that – using on-demand clusters with known up-front cost, or using spot clusters with dynamic cost. The choice depends on customer bids and machine availability, as described in subsection 4.2. This paper primarily focuses on predicting future spot prices in a short time horizon and finding the optimal time to execute the ETL job for overall cost optimization.

Regarding the estimation of the right cluster size for a specific Hadoop job, we refer to our algorithm introduced in [3]. In a nutshell, it considers two parameters: the size of data that needs to be processed in one run, and the maximum time in which this data needs to be fully processed and loaded into the warehouse. The main output of the algorithm is the estimated duration, the number and the type of nodes that the cluster needs. In other words, this can be translated to the number of optimal vCPUs and RAM capacity, and the corresponding duration. To make this optimization, the algorithm analyzes collected logs about processed data volumes and the duration of previous ETL jobs. Again, having such logs is also a reasonable assumption for the following reason. Any organization adopting any Hadoop-based ETL architecture will need to verify that the setup can successfully process the required data. During this verification, and even production use of the archi-

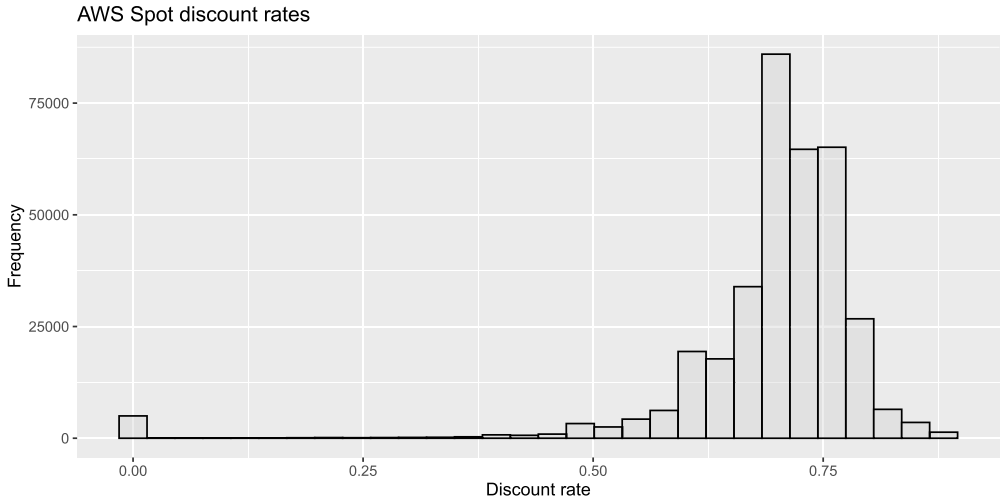


Fig. 2. Histogram of discount rates for the Linux/UNIX spot machines compared to on-demand pricing. The values are computed based on the data described in Section 5.1.

tecture (but without cluster-size optimization), these logs will be collected. Considering that the details of the cluster-size optimization algorithm and the experimental results are described in [3], the remainder of this article focuses on the second aspect, namely, predicting when the spot prices will be minimal for the suitable workload.

4.2. Spot instances

Spot instances can be regarded as spare compute capacity in the cloud. They are offered as one of the three ways cloud providers sell their computing capacity – the other two are on-demand and reserved instances. In terms of the servers, there is no difference between the three. The difference is in the business model. On-demand instances represent the pay-as-you-go model, while reserved instances facilitate long-term renting of computing resources with a discount. However, spot instances allow customers to save up to 90% of costs by using the cloud's unused servers (see Fig. 2, described in section 5.1). The two most popular cloud providers, Amazon AWS[1] and Windows Azure,[2] have such spot instance offerings. Even though both Windows Azure and Amazon AWS offer spot instances, there are years of spot instance price history available for AWS. Therefore, most of the discussion in this paper revolves around AWS types of spot instances.

With spot instances, customers never pay more than the maximum price specified in the bid. However, the evident concern with the spot model is that the cloud provider may terminate these instances with literally last-minute notice. To understand how spot instances can be utilized for ETL workloads, the reasons for interruptions and behavior during them need to be understood first. The following are the possible reasons that Amazon EC2 might interrupt Spot Instances[3]: *Price*, the Spot price is greater than the customer's maximum price; *capacity*, if there are not enough unused EC2 instances to meet the demand for Spot Instances, Amazon EC2 interrupts Spot Instances; and *constraints*, if the customer's request includes a constraint such as a launch group or an Availability Zone group, these Spot Instances are terminated as a group when the constraint can no longer be met.

AWS offers various options to configure interruption behavior of Spot instances and Spot Fleets (a set of spot instances), including hibernation and automatic restarting. When the AWS Spot service determines to hibernate a Spot Instance, an interruption notice is issued as a CloudWatch event, but the customer does not have time before the Spot Instance is interrupted, and hibernation begins immediately. To prevent interruptions, the best practices suggest using the on-demand price for bidding, storing necessary data regularly at persistent storage (e.g., Amazon S3, Amazon EBS, or DynamoDB), and dividing the work into small tasks while using checkpoints.

For the current use case, we are interested in performing ETL with deterministic latency. In other words, the assumption is that even though ETL can be delayed later, there is a deadline by which the ETL has to successfully complete. Therefore, the proposed framework can tune the cluster size and predict the optimal start time of a cluster, but once a cluster starts, we want to avoid interruptions, even at a higher cost (up to the cost of on-demand instances). In addition to the proper budgeting, this is one of the reasons that make an accurate price prediction in the upcoming periods very important.

4.3. Spot price prediction

Naive predictions Considering the use-cases of interest, a reasonable approach in many business scenarios would be to run the ETL process daily. As in many short-term forecast problems, the last known value is a reasonably good indicator of the next value. Thus, such predictions are commonly used as a baseline. In this section, we refer to it as Naive prediction model – that refers to the last known spot price of a particular instance type in the given availability zone. With this approach, at the time when we need to predict future spot price of a particular instance, we simply use the current price and predict that all future prices are going to be equal to it. Obviously, this is a very naive assumption, completely ignoring the dynamic demand for spot instances. As the prediction is about values further away in the future, the expectation is that the quality of such a forecast would significantly decrease. Still, the motivation to include this approach in the evaluation is derived from the preliminary exploratory analysis of the dataset showing that the spot prices of some instances were infrequently changing.

Autoregressive integrated moving average ARIMA form a class of time series models that are widely applicable in the field of time series forecasting. ARIMA models are known to be robust and efficient in short-term time series forecasting, with some prominent results in financial and commodity markets [44,48]. In the ARIMA

[1] aws.amazon.com/ec2/spot/.
[2] azure.microsoft.com/en-us/pricing/spot/.
[3] docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-interruptions.html.

model, the future value of a variable is a linear combination of past values and errors after removing the trend – by differencing. Given a time series data Y_t where t is an integer index, an ARMA(p,q) model is given by:

$$Y_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i Y_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$$

where Y_t and ε_t are the actual value and error at time period t, respectively. Whereas, c is a constant, θ_i and φ_i are model parameters to be estimated in the process of model training. ARIMA(p,d,q) model is an extension of ARMA that aims to model non-stationary processes. When the observed time series has a trend, the difference between consecutive observations is computed d times until the observed process becomes stationary.

To provide high-quality spot price forecasts, we trained ARIMA models separately for each AWS instance type in each availability zone. To adjust price prediction to still changing environment on the AWS spot market, hence minimizing the effect of so-called concept drifts [54], models were iteratively re-trained after each day in data. The more detailed analysis of spot price prediction is provided further in Section 5.3.

4.4. Cluster configuration strategy

The configuration strategy should consider several factors. First, the deadline by when the ETL process has to successfully complete (e.g., 5 AM EST, each day). Second, the estimated duration and required CPUs (n_vCPU) and memory capacity (RAM), with can be done with the approach described in section 4.1. Third, the ability to obtain the desired cluster capacity by picking different instance types from the same instance family and availability zone before the ETL process is initiated.

To illustrate the third factor, let us consider the following example. Let us assume that the total required capacity is denoted by ($n\_vCPU = 96$) and memory capacity ($RAM = 768$ GB). To achieve this, there are multiple options. On AWS, as the instance size increases, both the RAM and vCPUs also increase proportionally to the size multiplier. For example, an *r5.2xlarge* instance has 64 GB RAM and 8 vCPUs, whereas an *r5.12xlarge* instance has 384 GB RAM and 48 vCPUs, or exactly 6 times more computing capacity for 6 times the on-demand cost while having the same network performance. Therefore, to achieve the same computing capacity of 768 GB RAM and 96 vCPUs, within the r5 instance family the are 6 options (i.e., 24 × *r5.xlarge*, 12 × *r5.2xlarge*, 6 × *r5.4xlarge*, 3 × *r5.8xlarge*, 2 × *r5.12xlarge*, or 1 × *r5.xlarge*). This assumes all instances are of the same type, which is a recommended practice for Hadoop clusters. In [4] the experimental evaluation showed that in most cases, the total computing capacity is what matters and that the ETL duration is marginally affected by choice of instance types within the same family and network speed.

To formalize this, let us first define the following parameters. An instance family F_i consists of all instance defined by AWS, except the ones that have smaller network speed. In some instance families, the smallest instances have lower network speed, in which case, they are excluded. This assumption makes sure that the network does not impact the subsequent estimates. For each instance $x \in F_i$, we denote the on-demand cost $cost_{ondemand}(x)$, spot instance cost at time t as $cost_{spot}(x,t)$ and normalized spot cost as

$$cost^n_{spot}(x,t) = cost_{spot}(x,t)/cost_{ondemand}(x)$$

Next, we can compare two arbitrary instances x and y that have the same network speed from the same instance family F_i in terms of computing capacity and cost with

$$capacity\_ratio(x,y) = \frac{compute\_power(x)}{compute\_power(y)}$$

and

$$cost\_ratio_{ondemand}(x,y) = \frac{cost_{ondemand}(x)}{cost_{ondemand}(y)}$$

Similarly for *spot* instances, we can define

$$cost\_ratio_{spot}(x,y,t) = \frac{cost_{spot}(x,t)}{cost_{spot}(y,t)}$$

$$scaleup(x) = compute\_power(x)/\min_F compute\_power(y)$$

For on-demand prices, the following holds

$$capacity\_ratio(x,y) = cost\_ratio_{ondemand}(x,y), \forall x \in F_i, \forall y \in F_i.$$

This means that the ratios of price and computational capacity of different instances are equal – for example – the twice more powerful machine, costs twice more. However, our exploratory analysis showed that this is not always true for spot instances, meaning that in general the following holds

$$\exists t, capacity\_ratio(x,y) \neq cost\_ratio_{spot}(x,y,t)$$

If we can predict which instance is the cheapest at those times t, this provides an opportunity to create a cluster with the same capacity composed of cheaper instances. We can always chose the instance with lowest $cost^n_{spot}(x,t)$ for the same instance family F_i at time t. This means that for the same computing power that we need, we'll always pay the cheapest price. The highest price we would pay, by choosing spot instances would be defined as

$$max\_cost(t) = \max cost^n_{spot}(x,t) \ for\, x \in F_i.$$

We can also define the predicted cost using the proposed prediction methods as $predicted\_cost_{spot}(x,t)$. So for each time period t we can either chose a machine

$$min^x_t = \min_x predicted\_cost_{spot}(x,t) \ for\, x \in F_i$$

By choosing the appropriate x_t, we would actually get a

$$predicted\_instance\_cost(t) = cost^n_{spot}(x = min^x_t, t)$$

If we chose a random spot machine, we would risk having the

$$worst\_case\_cost(t) = max\_cost(t)$$

In order to evaluate the success of the strategy, we compare the $cost^n_{spot}(x,t)$ with the $worst\_case\_cost(t)$ which is the worst case cost for spot instances and with the $cost_{ondemand}(x)$ which is the same for the needed processing power, regardless of which machine within the same family we choose. If we define the Dynamically-selected spot instance price as the $predicted\_cost_{spot}(x,t)$ then we can calculate the ratio between this and the worst case cost and the on demand cost.

$$on\_demand\_cost\_ratio = \frac{cost\_ondemand(x)}{predicted\_cost_{spot}(x,t)}$$

and

$$most\_expensive\_spot = \frac{worst\_case\_cost(t)}{predicted\_cost_{spot}(x,t)}.$$

To get a better estimation of the prices difference we set the dynamically-selected spot price as

$$dynamically\_selected\_cost(t) = \frac{predicted\_cost_{spot}(x,t)}{predicted\_cost_{spot}(x,t)} = 1$$

Table 1
The most popular spot machines.

Region	AZ	Machine	N	Bid freq. (h)		Bid price ($)	
				Avg	StDev	Avg	StDev
ap-northeast-2	a	r4.8xlarge	513	5.609858	1.850305	0.61898109	0.032847797
ap-south-1	c	m5.4xlarge	511	5.646654	2.098600	0.26793268	0.049972972
ap-south-1	a	m4.10xlarge	510	5.664042	1.919549	0.64866667	0.065327196
us-west-1	a	r4.8xlarge	509	5.679289	1.691382	0.68227269	0.042938466
us-west-1	a	r5.4xlarge	507	5.697051	1.577393	0.56153195	0.082851776
sa-east-1	c	m4.4xlarge	505	5.718381	2.680653	0.33331168	0.022590959
us-west-2	b	c5n.4xlarge	504	5.720918	1.442650	0.40579861	0.031436667
us-west-1	b	m5d.4xlarge	504	5.730931	1.736511	0.37877837	0.071149562
eu-central-1	c	c5d.9xlarge	504	5.732888	1.632358	0.64598294	0.032545920
ap-south-1	b	c5.2xlarge	504	5.740978	1.712241	0.19892956	0.022324986

5. Results and discussion

Contributing to the popularity in industry and research community of the Amazon Web Services (AWS), and the hardware heterogeneity offered in various instance types [52], AWS was used for the experimental evaluation of the proposed architecture, including the experiments with spot instances. AWS cloud consists of geographically dispersed regions around the world, each with multiple availability zones (AZ's).[4] In each of the regions, AWS offers a broad number of cloud services, among which the Elastic Cloud Compute (EC2) is the essential one.

5.1. Dataset

The analyzed spot price data set was collected from 11 AWS regions[5]: *ap-northeast-1* (427309), *ap-northeast-2* (303113), *ap-south-1* (322593), *ap-southeast-1* (459592), *ap-southeast-2* (382703), *ca-central-1* (206218), *eu-central-1* (453457), *eu-west-1* (546474), *sa-east-1* (280076), *us-east-1* (1061429), *us-west-1* (285975), *us-west-2* (661369) over the period between November 11, 2019 and March 11, 2020. It consisted of a total of 5.4*M* records, each corresponding to a particular bid for one of the AWS spot instances. After the preliminary data filtering, we left only those records which referred to the EMR compatible machine types[6] working with *Linux/UNIX* operating system. In particular, we were only interested in ETL related servers, that is: the memory oriented machine types – *m* and *r* series; the computation oriented – *c* series, and *g* and *p* series which are popular for the data analysis and machine learning. The remaining instance families were ignored. For spot instances there is also a constraint that the root volume must be an Elastic Block Store (EBS) volume, not an instance store volume, which eliminated some of the instances for this study. The spot price time series for each machine type may differ between regions and availability zones.

Concerning the savings that could be made with spot instances compared to the corresponding on-demand prices, the experimental results indeed show the advertised claim of savings up to 90% is indeed true, as shown in Fig. 2, based on the analysis of the time period and instance types described in section 5.1.

In Table 1, we present a brief overview of 10 spot price time series for the most popular (i.e., with the most frequent spot price changes) machine types.

5.2. Spot prices exploratory analysis

To verify the feasibility of short term spot price prediction, we decided to limit the scope of analysis further and to focus on the time series with non-trivial price change characteristics. Therefore, we discarded machines with infrequent bids (less than 100 bids in the entire data set), as well as the time series with almost constant prices in the analyzed time range (with the standard deviation of prices $\sigma < 0.01$). The final data set contained 854 time series for 85 different machine types aggregated in non-overlapping candlesticks – a standard tool in financial stock market analysis [55]. Candlestick charts are often used together with various machine learning models, like SVM or DNN [56]. In the performed experiments, each candlestick contained volume of operations as well as open, high, low and close price during one day. The exemplary candlestick charts for two popular machine types in North Virginia (*us-east-1*) region are depicted in Fig. 3.

5.3. Spot price predictions

Having the data aggregated in one-day candlesticks, for a give day (t_x), we aim to predict the highest price during the next day (t_{x+1}). The models are evaluated with two commonly used error metrics, namely, root mean square error (RMSE) and mean absolute percentage error (MAPE). However, to make the prediction and obtained error rates comparable between various machines, the prices were scaled – they were divided by the on-demand price of the same machine type in the corresponding region. This allows providing an estimation of a budget needed for the data processing task. The preliminary analysis showed that even the *Naive* model, which used as a prediction the last day price, achieved a relatively good quality. The highest MAPE of 5.49% was recorded for the *m5d.16xlarge* machine type in *ap-northeast-2* region (see Fig. 5 (a)). The median and macro average of MAPE over all 854 evaluated time series was 0.66% and 0.87%, respectively. These results mean that in the worst case, we can expect a budget overrun of ca. 5.49% in the event of a rapid price change (as shown in Fig. 5). However, on average, the error will be much smaller. The results of Naive model performance aggregated over all 854 time series are presented in Table 2 in the row signed *Naive*. The median RMSE, in Table 2, refers to median value of 854 experiments. Similarly, in the table, we also report macro average, 3rd quantile, and max value for RMSE and MAPE.

In the performed study, we trained ARIMA models for each of 854 time series in data. Similar to the Naive model's case, the evaluation was performed on the last two months in data (60 days). Before each assessment (at time t_x) of next day price, the ARIMA model was re-trained on all available historical data ($t_0 \ldots t_x$). The aggregated performance of the ARIMA model trained on all available history is presented in Table 2 in the row marked as "ARIMA(All)". To further verify the optimal history size for the estimation of model's parameters, that allows more dynamically

[4] See AWS global cloud infrastructure at aws.amazon.com/about-aws/global-infrastructure.

[5] Numbers in brackets indicate the amount of bids. We may notice that in the most active region – North Virginia (us-east-1) – there were recorded over two times more bids than in the second most active one.

[6] See docs.aws.amazon.com/emr/latest/ManagementGuide/emr-supported-instance-types.

(a) r5.12xlarge (b) r4.16xlarge

Fig. 3. Candlestick charts for two popular machine types in N. Virginia region, both in AZ: 'b'.

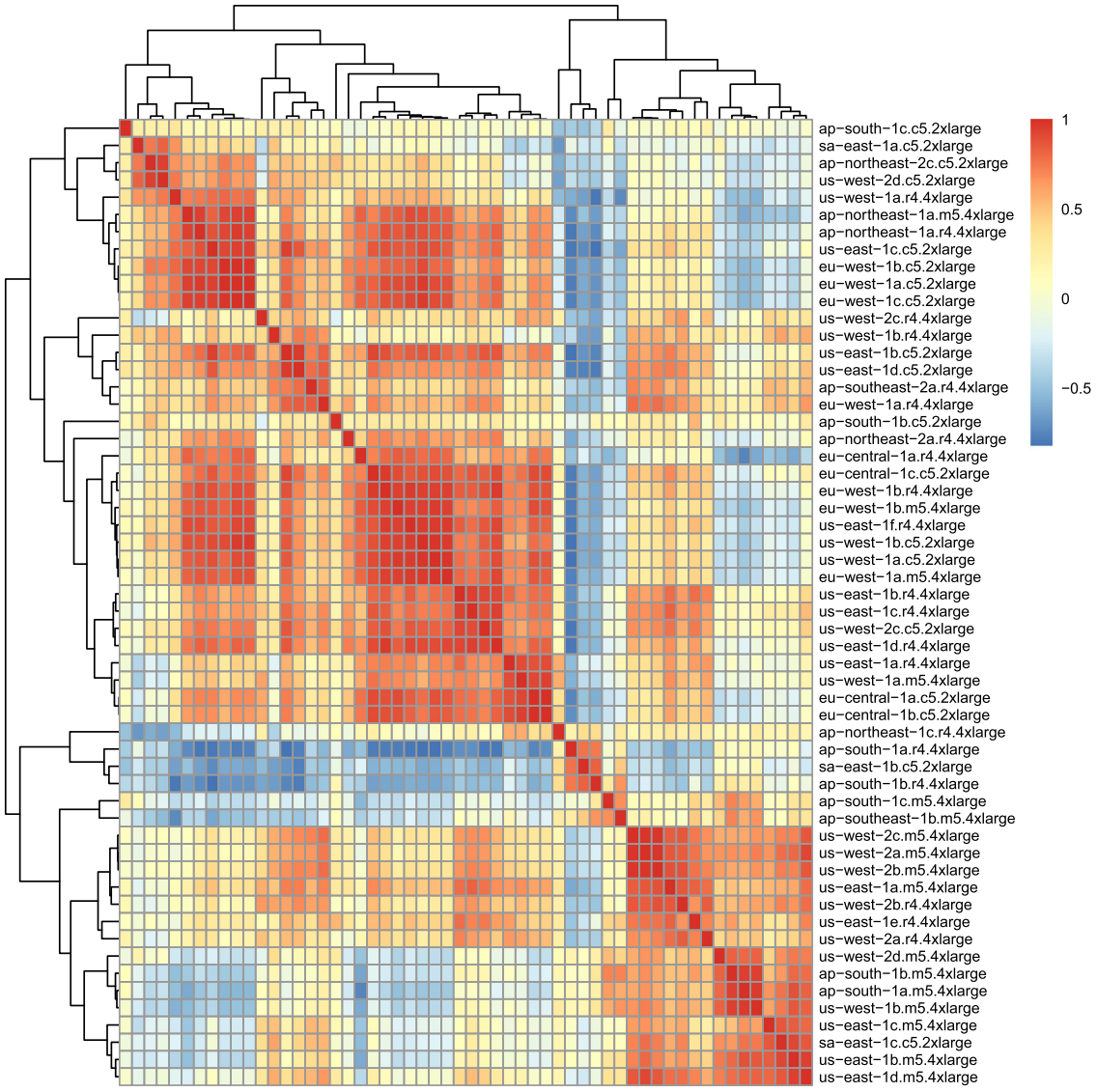
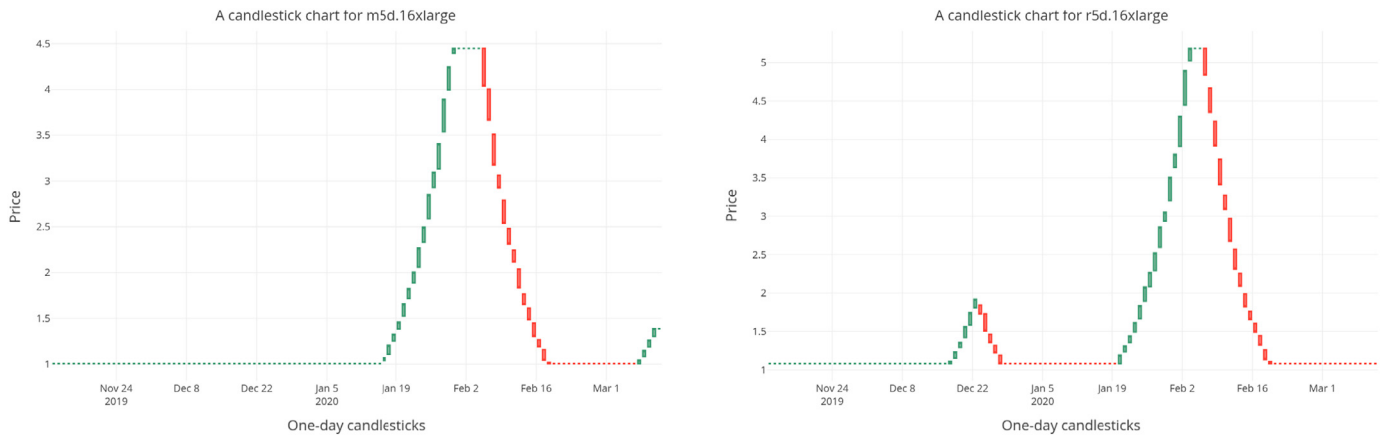


Fig. 4. Correlation heatmap for the three popular spot machine types: r4.4xlarge, m5.4xlarge, c5.2xlarge. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)



(a) Reg.(AZ):ap-northeast-2(b), machine:m5d.16xl (b) Reg.(AZ):us-west-1(b), machine:r5d.16xl

Fig. 5. Candlestick charts of two machines troublesome for prediction.

Table 2
Spot price prediction in one day horizon. ARIMA with various sliding window settings (training history length in brackets) and the naive model with last known value. The minimal error for all classifiers was equal to zero. The 1st quantile was always smaller than 0.001 and 0.15 for R2 and MAPE, respectively. Prediction evaluated on the last two months in data (60 days).

Model	Hist. size	RMSE				MAPE			
		Median	MacAvg	3rdQu.	Max	Median	MacAvg	3rdQu.	Max
ARIMA	10	0.003541	0.006239	0.006510	**0.041954**	0.7069	0.9088	1.2520	5.1401
	15	0.003147	**0.00547**	0.00713	0.06168	0.6486	0.8008	1.1552	3.8915
	20	0.003252	0.006428	0.007588	0.059637	0.6620	0.8884	1.2644	5.1581
	25	0.003125	0.006296	0.007502	0.059256	0.6322	0.8597	1.1928	6.4112
	30	0.003045	0.005978	0.007135	0.059093	0.6199	0.8240	1.2005	5.5383
	35	0.002986	0.006060	0.007151	0.080333	0.6101	0.8197	1.1622	4.7605
	40	0.002928	0.005767	0.006876	0.059086	0.5987	0.8020	1.1433	**3.8416**
	45	0.002864	0.005771	0.007044	0.075629	0.5954	0.8032	1.1157	4.0203
	50	0.002861	0.005567	0.006923	0.102551	0.5893	**0.789**	1.0645	4.9812
	All	**0.00281**	0.005506	0.007388	0.078838	**0.5866**	0.8051	**1.0335**	6.5449
Naive	1	0.003135	0.006071	**0.005902**	0.046476	0.6640	0.8782	1.2556	5.4873

respond to the shifts in characteristics of AWS spot prices, we repeated the experiments for various length of training data history to fit the ARIMA model (from 10 days to 50 days). In the case of 40 day long history the maximal MAPE error of 3.84% was recorded in *us-west-1* region for *r5d.16xlarge* machine (see Fig. 5(b)). This provides us with the worst-case estimation of cost under- or over-run of spot resource allocation. Still, in the (macro) averaged or mean case, we would be far more accurate. In Table 2, the aggregated analysis for the Naive model and all ARIMA settings is presented.

For the more in-depth analysis, we decided to select Naive, ARIMA(All), ARIMA(40) models. The first one presents a baseline, the second achieved lowest average errors, whereas the ARIMA(40) minimized the maximal MAPE error, which assures the lowest worst-case budget misestimation. The three main parameters to be estimated in the ARIMA(p, d, q) model are the number of time lags of the auto-regressive model p, degree of differencing d, and the order of the moving average model (q). In our experiments, these parameters were estimated using the Box–Jenkins approach. The analysis of the selected models revealed that for various time series and length of available training data, different p and q parameter values were chosen. In the performed experiments, the series were most often differenced once - trend components for the trained ARIMA models were usually $d = 1$. The AR parameters were typically equal to 1 or 2, whereas MA parameters varied from 0 to 4. The seasonality test was negative in all examined cases. Hence, the trained ARIMA models were of a form ARIMA$(1-2, 1, 0-4)$.

To validate the statistical significance of observed differences between the performance of the selected models, we decided to employ the Wilcoxon signed rank test – due to a very low p-value observed during Shapiro-Wilk normality test on both RMSE and MAPE distributions achieved during the tests. In all the cases, p-value of Shapiro-Wilk normality test was: $p\text{-}value < 1.0e\text{-}15$. In the case of RMSE, the Wilcoxon signed rank test, with the null hypothesis that the errors of the Naive model are not greater than those of ARIMA(40) did not allow to reject this hypothesis ($p\text{-}value = 0.1954$). However, when the ARIMA(All) model was compared to ARIMA(40) and Naive, the p-values of both tests were very low, i.e., $3.786e\text{-}08$ for Naive and $4.649e-06$ for ARIMA(40), respectively. It allowed us to reject the null hypothesis, hence showing the statistical significance of differences between the models. Slightly different observations were made for the MAPE measure. In this case, the Wilcoxon test revealed that ARIMA(40) model was significantly better than Naive ($p\text{-}value = 3.053e\text{-}07$). However, the ARIMA(All) again performed significantly better than both ARIMA(40) ($p\text{-}value = 0.005515$) and Naive ($p\text{-}value = 3.396e\text{-}11$) models.

In the last part of our study, we attempted to validate the feasibility of spot price prediction in a bit longer horizon of two and three days ahead. We examined the performance of the three selected models from our previous test: Naive, ARIMA(All) and ARIMA(40). The results - presented in Table 3 - showed that the observed drop of each model performance is significant, and the maximal MAPE error exceeds 16% for both ARIMA(All) and Naive models. However, we may conclude that prediction is still feasible two and three days ahead, with a median of MAPE errors only slightly exceeding 1.6% for ARIMA(All). Further research on

Table 3
A summary of the prediction errors of the selected models for various forecasting horizons (1-3 days).

Model	Pred. day	RMSE				MAPE			
		Median	MacAvg	3rdQu.	Max	Median	MacAvg	3rdQu.	Max
ARIMA (All)	1	0.0028	0.0055	0.0074	0.0788	0.5866	0.8051	1.034	6.545
	2	0.0053	0.01	0.0118	0.095	1.15	1.537	2.029	10.73
	3	0.0076	0.0146	0.0152	0.142	1.647	2.28	2.948	16.32
ARIMA (40)	1	0.0029	0.0058	0.0069	0.0591	0.599	0.802	1.143	3.842
	2	0.0056	0.0105	0.0123	0.079	1.187	1.57	2.18	7.96
	3	0.0079	0.015	0.017	0.136	1.72	2.36	3.24	13.98
Naive	1	0.00314	0.0061	0.006	0.0465	0.664	0.878	1.256	5.487
	2	0.00547	0.0112	0.011	0.0887	1.1841	1.6459	2.335	11.01
	3	0.0073	0.0157	0.014	0.126	1.609	2.345	3.317	16.48

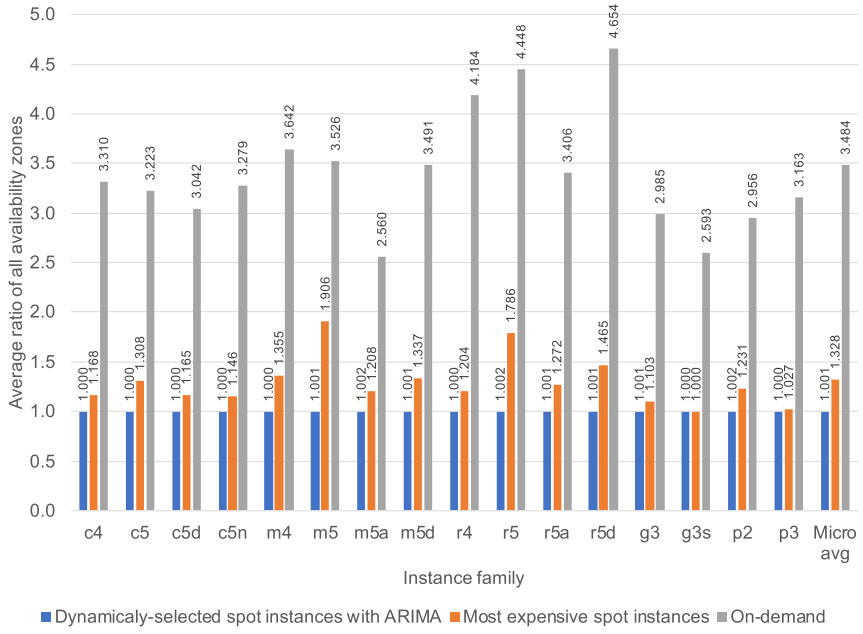


Fig. 6. Cost ratio compared to the minimum cost per instance families. The last bar is the Micro average of the ratio considering all instances and all availability zones.

more sophisticated regression methods could bring more accurate predictions. An interesting approach would be to use multivariate methods, mainly due to the observed correlations between various time series in multiple regions and availability zones, as shown in Fig. 4. This figure is a heatmap with a dendrogram added to the left side and to the top where the color of each cell represents the correlation between the price of a pair of instance types in different availability zones. A dendrogram is a tree-structured graph that visualizes the result of a hierarchical clustering calculation. For the dendrogram on the left side of the heatmap, the individual rows in the clustered data are represented by the right-most nodes (i.e., the leaf nodes). Each node in the dendrogram represents a cluster of all rows from the connected leafs. The left-most node in the dendrogram is therefore a cluster that contains all rows.

5.4. Cost-effectiveness analysis of dynamic cluster selection

We performed the following experiments to illustrate the practical benefit of the proposed method for dynamic cluster configuration. Using the same data as described in Section 5.1, we made predictions of the spot prices of all instances for the last 60 days in data with the best of the analyzed models in the previous section, i.e., ARIMA(all). When this model is deployed in a production setting to make predictions of the price, it dynamically guides the cluster configuration. At the end of each billing period, the implications related to actual incurred costs are known, as well as whether the predictions were the most optimal. Therefore, Fig. 6 and 7 relate to the actual cost and not the predicted cost. Also, the ARIMA-based spot prices cost, the on-demand prices, and the most expensive spot-configuration are ratios compared to the cheapest instances that could be used for the dame day. Considering the parameters defined in section 4.4, lets take for example the *ap.northeast.1c* region and the *c5d* instance types. Let us assume that the task lasts less than 24 hours to complete and that we need to repeat this task for the following 60 days. We need one 4*x.large* instance or two 2*x.large* instances to complete the task each day. We could choose between several strategies – use one 4*x.large* instance each day, use two 2*x.large* instances every day or use the best option using the ARIMAs prediction. The normalized cost we would pay for using the on-demand machine would be 60 as we pay the full price for the desired processing power every day for 60 days. If we chose two 2*x.large* every day, we would pay normalized spot cost of

$$total\_cost(2x.large) = \sum_{t=1}^{60} cost^{n}_{spot}(x = 2x.large, t) = 17.46$$

and if we opt in for one 4*x.large*, we would pay normalized spot cost of

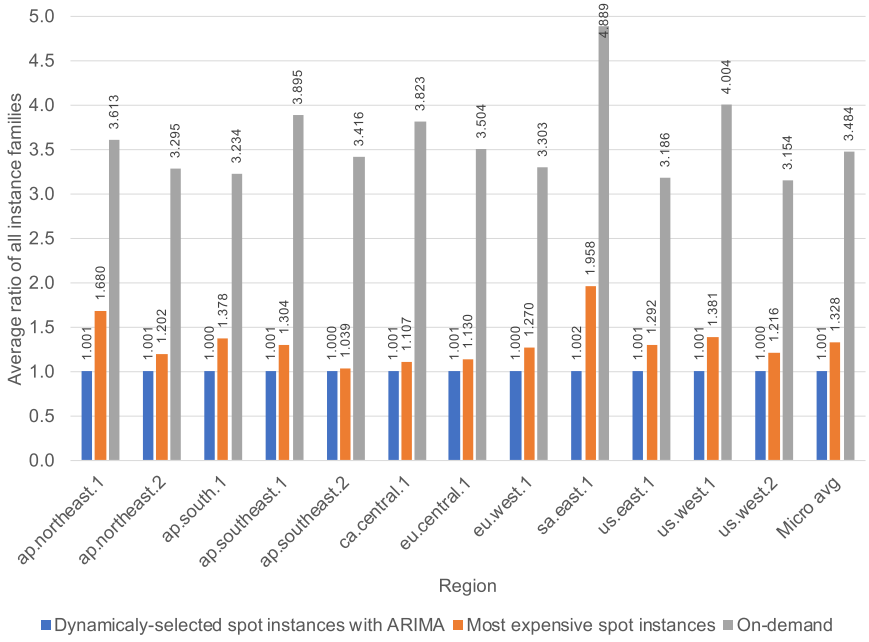


Fig. 7. Cost ratio compared to minimum cost per region. The last bar is the Micro average of the ratio considering all instances in a region.

$$total\_cost(4x.large) = \sum_{t=1}^{60} cost^{n}_{spot}(x = 4x.large, t) = 18.18$$

These represent the sums of normalized costs. By using the ARIMA method and the proposed strategy, the normalized spot cost for the full 60 day period would be:

$$total\_cost(ARIMA) = \sum_{t=1}^{60} cost^{n}_{spot}(x = predicted\_spot, t) = 17.23$$

where

$$predicted\_spot = \min_{x}(predicted\_cost_{spot}(x,t)) \;\; for\; x \in F_i$$

If we use the actual minimum price by knowing it up-front somehow, we would pay 17.23 for this specific instance type which is very close to the ARIMA prediction and if we chose always the most expensive option the cost would be 18.41. Based on these findings, for the 60 days period we would have:

$$dynamically\_selected\_cost = \frac{17.23}{17.23} = 1$$

$$most\_expensive\_spot\_ratio = \frac{18.41}{17.23} = 1.068$$

$$on\_demand\_cost\_ratio = \frac{60}{17.23} = 3.48$$

In Fig. 6 and 7, lower is better, and the minimum value is 1, meaning the average cost is the same as the absolute minimum cost that could be paid for those days.

In Fig. 6, the cost ratios per instance family are shown. The metrics average the cost ratios per instance family, regardless of the availability zone and region. Notably, the average on-demand cost ranges from 2.5 to almost 5 times more than the absolute minimum that could be obtained with a dynamic choice of spot instances for the same instance family. Similarly, by choosing sub-optimal spot instances, the cost ratio ranges from 1.1 to 1.9, meaning one would pay 10% to 90% more. The proposed prediction method minimizes this cost, which in most cases, is the optimal one, and always is less than 0.2%. It is also worth noting the memory-optimized instance families benefit from spot instances mostly. Likewise, the GPU instance types have substantial cost reduction compared to on-demand instances, but see the smallest benefit from the dynamic instance size selection.

In Fig. 7, we show the cost ratios per region. The metrics average the cost ratios per region, regardless of the availability zone and instance family. Notably, the average on-demand cost ranges from 3.1 to almost 5 times more than the absolute minimum that could be obtained with a dynamic choice of spot instances from the same instance family in the same availability zone. Similarly, by choosing sub-optimal spot instances, the cost ratio ranges from 1.04 to 1.9, meaning one would pay 4% to 90% more. The proposed prediction method minimizes this cost, which in most cases, is the optimal one, and always is <0.1%. It is also worth noting that the clusters deployed in the *sa.east.1* region would mostly benefit from spot instances. Likewise, except for the *ap.southeast.2* region, in other regions, considerable cost reduction can be achieved with dynamic instance size selection compared to having a cluster with predefined instance sizes.

5.5. Limitations

One factor worth considering when applying the proposed approach is the geographical location of the data, and the clusters that are ought to process it. Preferably, the data and compute resources should be in the same data center to leverage the co-location for reducing the latency and avoiding unnecessary expenses related to data transfer out of the data center. That being said, in some cases, the optimization strategy should be only limited to consider clusters in the data center where the data is kept. However, even in such cases, using the proposed algorithm, it is possible to identify the optimal cluster configuration by selecting the cheapest instance types in the location of interest.

Another consideration is the sudden termination of VMs by the cloud provider. Amazon discloses that it can happen at two-minute notice before termination, providing a chance to interrupt running jobs. As described in Section 4.2, there are various bidding strategies, and practical guidelines provided by Cloud operators that aim at minimizing that risk. Still, it would be valuable to thoroughly evaluate those strategies through extensive testing. Cloud providers also recommend saving the progress of running jobs in small increments, so in the case of interruptions, only a small part of in-

complete work is lost. This last recommendation is in line with the builtin Spark fault tolerance mechanism, i.e., D-Streams in a system called Spark Streaming [57], or Resilient Distributed Datasets [58].

In theory, if multiple cloud customers use the proposed bidding strategy, their actions could result in a spike in demand for spot instances, causing the predictions to be invalid. Situations like that often occur in the stock markets and are considered as particularly challenging to address and predict. In that regard, any experimental verification of our approach would also be complicated and would require significant resources. The validation of the proposed algorithms in real-life scenarios would involve simultaneous requests of hundreds, if not thousands, of VMs in a single availability zone. With appropriate budgeting, however, this is an exciting topic for future research.

6. Conclusions

There are exceptionally reliable and efficient tools that are well-architected by the cloud providers are easy to combine into highly effective processes. By doing so, architects and developers create data pipelines and business processes much faster and with better quality than the more traditional ways did it. At the same time, the resources available are not always the most cost-effective. A given process, such as Spark session, may be performed at the prime time, when the cost is higher even though, that actual results are not needed at that specific time.

This paper proposed a cloud-based architecture for processing of Big Data. The proposed framework first identifies the optimal cluster configuration to perform the processing. Then, by analyzing the history of spot prices of the needed resources, performs intelligent dynamic scheduling so that the cost of the resources are minimized. Our experiments showed that we could optimize the cluster costs with a reduction of up to 80% compared to on-demand clusters. Furthermore, the dynamically selected configuration costs only up to 1% more than the absolute minimum. The proposed method could be a valuable method for optimizing cloud-computing costs of organizations, while also providing accurate budget planning predictions. In the future, the dynamically scheduling system needs to be evaluated in a production setting and investigate how often interruptions of spot instances occur and whether they can be avoided.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was co-financed by the EU Smart Growth Operational Programme 2014-2020 under the project “Development and validation of CogniMarts - an artificial intelligence-based system capable of independent, automatic (without human involvement) ordering and combining large data sets”, POIR.01.01.01-00-0742/19. This work was co-financed by Polish National Science Centre (NCN) grant no. 2018/31/N/ST6/00610.

References

[1] R. Keller, L. Häfner, T. Sachs, G. Fridgen, Scheduling flexible demand in cloud computing spot markets, Bus. Inf. Syst. Eng. 62 (1) (2020) 25–39, https://doi.org/10.1007/s12599-019-00592-5.

[2] J. Sandobalin, E. Insfrán, S. Abrahão, On the effectiveness of tools to support infrastructure as code: model-driven versus code-centric, IEEE Access 8 (2020) 17734–17761, https://doi.org/10.1109/ACCESS.2020.2966597.

[3] E. Zdravevski, P. Lameski, A. Dimitrievski, M. Grzegorowski, C. Apanowicz, Cluster-size optimization within a cloud-based ETL framework for big data, in: 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, December 9–12, 2019, IEEE, 2019, pp. 3754–3763.

[4] E. Zdravevski, P. Lameski, C. Apanowicz, D. Ślęzak, From big data to business analytics: the case study of churn prediction, Appl. Soft Comput. 90 (2020) 106164, https://doi.org/10.1016/j.asoc.2020.106164.

[5] S. Chaudhuri, U. Dayal, V. Narasayya, An overview of business intelligence technology, Commun. ACM 54 (8) (2011) 88–98, https://doi.org/10.1145/1978542.1978562.

[6] M. Bala, O. Boussaid, Z. Alimazighi P-etl, Parallel-etl based on the mapreduce paradigm, in: 2014 IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA), 2014, pp. 42–49.

[7] M. Cai, M. Grund, A. Gupta, F. Nagel, I. Pandis, Y. Papakonstantinou, M. Petropoulos, Integrated querying of sql database data and s3 data in Amazon redshift, IEEE Data Eng. Bull. 41 (2) (2018) 82–90.

[8] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, V. Srinivasan, Amazon redshift and the case for simpler data warehouses, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data – SIGMOD '15 2015-May, 2015, pp. 1917–1923.

[9] Amazon Athena, https://docs.aws.amazon.com/athena/. (Accessed 5 May 2020).

[10] K. Feasel, Polybase in azure synapse analytics, in: PolyBase Revealed, Springer, 2020, pp. 233–249.

[11] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, R. Murthy, Hive - a petabyte scale data warehouse using hadoop, https://doi.org/10.1109/ICDE.2010.5447738, March 2010.

[12] E. Zdravevski, P. Lameski, A. Kulakov, Row key designs of NoSQL database tables and their impact on write performance, in: Proceedings – 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2016, 2016, pp. 10–17.

[13] D. Ślęzak, R. Glick, P. Betlinski, P. Synak, A new approximate query engine based on intelligent capture and fast transformations of granulated data summaries, J. Intell. Inf. Syst. 50 (2018) 385–414.

[14] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, C. Curino, Apache tez: a unifying framework for modeling and building data processing applications, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, ACM, New York, NY, USA, 2015, pp. 1357–1369.

[15] J.E. Gonzalez, P. Bailis, M.I. Jordan, M.J. Franklin, J.M. Hellerstein, A. Ghodsi, I. Stoica, Asynchronous complex analytics in a distributed dataflow architecture, preprint, arXiv:1510.07092.

[16] P. Godfrey, J. Gryz, P. Lasek, Interactive visualization of large data sets, IEEE Trans. Knowl. Data Eng. 28 (8) (2016) 2142–2157.

[17] P. Godfrey, J. Gryz, P. Lasek, N. Razavi, Interactive visualization of big data, in: S. Kozielski, D. Mrozek, P. Kasprowski, B. Małysiak-Mrozek, D. Kostrzewa (Eds.), Beyond Databases, Architectures and Structures. Advanced Technologies for Data Mining and Knowledge Discovery, Springer International Publishing, Cham, 2016, pp. 3–22.

[18] H. Chen, R. Chiang, V. Storey, Business intelligence and analytics: from big data to big impact, Manag. Inf. Syst. Q. 36 (4) (2012) 1165–1188.

[19] M. Ceci, R. Corizzo, F. Fumarola, M. Ianni, D. Malerba, G. Maria, E. Masciari, M. Oliverio, A. Rashkovska, Big data techniques for supporting accurate predictions of energy production from renewable sources, in: Proceedings of the 19th International Database Engineering and Applications Symposium, IDEAS '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 62–71.

[20] P. Lasek, J. Gryz, Density-based clustering with constraints, Comput. Sci. Inf. Syst. 16 (2) (2019) 469–489.

[21] D. Ślęzak, M. Grzegorowski, A. Janusz, M. Kozielski, S.H. Nguyen, M. Sikora, S. Stawicki, Ł. Wróbel, A framework for learning and embedding multi-sensor forecasting models into a decision support system: a case study of methane concentration in coal mines, Inf. Sci. 451–452 (2018) 112–133, https://doi.org/10.1016/j.ins.2018.04.026.

[22] R. Corizzo, M. Ceci, E. Zdravevski, N. Japkowicz, Scalable auto-encoders for gravitational waves detection from time series data, Expert Syst. Appl. 151 (2020) 113378.

[23] B. Petrovska, T. Atanasova-Pacemska, R. Corizzo, P. Mignone, P. Lameski, E. Zdravevski, Aerial scene classification through fine-tuning with adaptive learning rates and label smoothing, Appl. Sci. 10 (17) (2020) 5792.

[24] R. Corizzo, M. Ceci, H. Fanaee-T, J. Gama, Multi-aspect renewable energy forecasting, Inf. Sci. 546 (2021) 701–722.

[25] M. Grzegorowski, A. Janusz, D. Ślęzak, M.S. Szczuka, On the role of feature space granulation in feature selection processes, in: J. Nie, Z. Obradovic, T. Suzumura, R. Ghosh, R. Nambiar, C. Wang, H. Zang, R. Baeza-Yates, X. Hu, J. Kepner, A. Cuzzocrea, J. Tang, M. Toyoda (Eds.), International Conference on Big Data, IEEE, BigData 2017, Boston, MA, USA, December 11–14, 2017, pp. 1806–1815.

[26] F.A. Batarseh, E.A. Latif, Assessing the quality of service using big data analytics: with application to healthcare, Big Data Res. 4 (2016) 13–24, https://doi.org/10.1016/j.bdr.2015.10.001.

[27] A. Janusz, M. Grzegorowski, M. Michalak, L. Wróbel, M. Sikora, D. Ślęzak, Predicting seismic events in coal mines based on underground sensor mea-

surements, Eng. Appl. Artif. Intell. 64 (2017) 83–94, https://doi.org/10.1016/j.engappai.2017.06.002.
[28] E. Zdravevski, P. Lameski, A. Kulakov, B. Jakimovski, S. Filiposka, D. Trajanov, Feature ranking based on information gain for large classification problems with mapreduce, in: Proceedings of the 9th IEEE International Conference on Big Data Science and Engineering, IEEE Computer Society Conference Publishing, 2015, pp. 186–191.
[29] E. Zdravevski, P. Lameski, A. Kulakov, S. Filiposka, D. Trajanov, B. Jakimovski, Parallel computation of information gain using hadoop and mapreduce, in: M.P.M. Ganzha, L. Maciaszek (Eds.), Proceedings of the 2015 Federated Conference on Computer Science and Information Systems, in: Annals of Computer Science and Information Systems, vol. 5, IEEE, 2015, pp. 181–192.
[30] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M.J. Franklin, R. Zadeh, M. Zaharia, A. Talwalkar, Mllib: machine learning in Apache Spark, J. Mach. Learn. Res. 17 (1) (2016) 1235–1241.
[31] A. Gounaris, J. Torres, A methodology for spark parameter tuning, Big Data Res. 11 (2018) 22–32, https://doi.org/10.1016/j.bdr.2017.05.001.
[32] R.E. Shawi, S. Sakr, D. Talia, P. Trunfio, Big data systems meet machine learning challenges: towards big data science as a service, Big Data Res. 14 (2018) 1–11, https://doi.org/10.1016/j.bdr.2018.04.004.
[33] I.A.T. Hashem, I. Yaqoob, N.B. Anuar, S. Mokhtar, A. Gani, S. Ullah Khan, The rise of "big data" on cloud computing: review and open research issues, Inf. Sci. 47 (2015) 98–115, https://doi.org/10.1016/j.is.2014.07.006.
[34] N. Kaur, S.K. Sood, Efficient resource management system based on 4vs of big data streams, Big Data Res. 9 (2017) 98–106, https://doi.org/10.1016/j.bdr.2017.02.002.
[35] S.S. Gill, P. Garraghan, V. Stankovski, G. Casale, R.K. Thulasiram, S.K. Ghosh, K. Ramamohanarao, R. Buyya, Holistic resource management for sustainable and reliable cloud computing: an innovative solution to global challenge, J. Syst. Softw. 155 (2019) 104–129, https://doi.org/10.1016/j.jss.2019.05.025.
[36] A. Malondkar, R. Corizzo, I. Kiringa, M. Ceci, N. Japkowicz, Spark-ghsom: growing hierarchical self-organizing map for large scale mixed attribute datasets, Inf. Sci. 496 (2019) 572–591, https://doi.org/10.1016/j.ins.2018.12.007, http://www.sciencedirect.com/science/article/pii/S0020025518309496.
[37] K. Elmeleegy, Piranha: optimizing short jobs in hadoop, Proc. VLDB Endow. 6 (11) (2013) 985–996.
[38] H. Herodotou, F. Dong, S. Babu, No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics, in: Proceedings of the 2nd ACM Symposium on Cloud Computing, ACM, 2011, p. 18.
[39] D.J. Dubois, G. Casale, Optispot: minimizing application deployment cost using spot cloud resources, Clust. Comput. 19 (2) (2016) 893–909, https://doi.org/10.1007/s10586-016-0568-7.
[40] X. Wu, F.D. Pellegrini, G. Gao, G. Casale, A framework for allocating server time to spot and on-demand services in cloud computing, ACM Trans. Model. Perform. Evaluation Comput. Syst. 4 (4) (2019) 20:1–20:31, https://doi.org/10.1145/3366682.
[41] Z. Cen, J. Wang, Crude oil price prediction model with long short term memory deep learning based on prior knowledge data transfer, Energy 169 (2019) 160–171, https://doi.org/10.1016/j.energy.2018.12.016.
[42] T. Fischer, C. Krauss, Deep learning with long short-term memory networks for financial market predictions, Eur. J. Oper. Res. 270 (2) (2018) 654–669, https://doi.org/10.1016/j.ejor.2017.11.054.
[43] D. Shah, H. Isah, F. Zulkernine, Stock market analysis: a review and taxonomy of prediction techniques, Int. J. Financ. Stud. 7 (2) (2019), https://doi.org/10.3390/ijfs7020026.
[44] A.A. Ariyo, A.O. Adewumi, C.K. Ayo, Stock price prediction using the ARIMA model, in: 2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation, 2014, pp. 106–112.
[45] A.A. Adebiyi, A.O. Adewumi, C.K. Ayo, Comparison of ARIMA and artificial neural networks models for stock price prediction, J. Appl. Math. 2014 (2014) 614342:1–614342:7, https://doi.org/10.1155/2014/614342.
[46] M. Baughman, C. Haas, R. Wolski, I. Foster, K. Chard, Predicting Amazon spot prices with lstm networks, in: Proceedings of the 9th Workshop on Scientific Cloud Computing, ScienceCloud'18, Association for Computing Machinery, New York, NY, USA, 2018, p. 7.
[47] V. Khandelwal, A.K. Chaturvedi, C.P. Gupta, Amazon ec2 spot price prediction using regression random forests, IEEE Trans. Cloud Comput. 8 (1) (2020) 59–72.
[48] S.A. David, J.A.T. Machado, L.R. Trevisan, C.M.C. Inácio, A.M. Lopes, Dynamics of commodities prices: integer and fractional models, Fundam. Inform. 151 (1–4) (2017) 389–408, https://doi.org/10.3233/FI-2017-1499.
[49] A. Neilson, Indratmo, B. Daniel, S. Tjandra, Systematic review of the literature on big data in the transportation domain: concepts and applications, Big Data Res. 17 (2019) 35–44, https://doi.org/10.1016/j.bdr.2019.03.001.
[50] R. Ranjan, Streaming big data processing in datacenter clouds, IEEE Cloud Comput. 1 (1) (2014) 78–83, https://doi.org/10.1109/MCC.2014.22.
[51] H. Hu, Y. Wen, T.S. Chua, X. Li, Toward scalable systems for big data analytics: a technology tutorial, IEEE Access 2 (2014) 652–687, https://doi.org/10.1109/ACCESS.2014.2332453.
[52] S. Mathew, Overview of Amazon Web Services, April 2017, accessed: 2019-06-04.
[53] M. Kiran, P. Murphy, I. Monga, J. Dugan, S.S. Baveja, Lambda architecture for cost-effective batch and speed big data processing, in: 2015 IEEE International Conference on Big Data (Big Data), 2015, pp. 2785–2792.
[54] A. Liu, J. Lu, F. Liu, G. Zhang, Accumulating regional density dissimilarity for concept drift detection in data streams, Pattern Recognit. 76 (2018) 256–272, https://doi.org/10.1016/j.patcog.2017.11.009.
[55] E. Ahmadi, M. Jasemi, L. Monplaisir, M.A. Nabavi, A. Mahmoodi, P.A. Jam, New efficient hybrid candlestick technical analysis model for stock market timing on the basis of the support vector machine and heuristic algorithms of imperialist competition and genetic, Expert Syst. Appl. 94 (2018) 21–31, https://doi.org/10.1016/j.eswa.2017.10.023.
[56] R.M.I. Kusuma, T.-T. Ho, W.-C. Kao, Y.-Y. Ou, K.-L. Hua, Using deep learning neural networks and candlestick chart representation to predict stock market, arXiv:1903.12258, 2019.
[57] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, I. Stoica, Discretized streams: fault-tolerant streaming computation at scale, in: M. Kaminsky, M. Dahlin (Eds.), ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3–6, 2013, ACM, 2013, pp. 423–438.
[58] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, in: S.D. Gribble, D. Katabi (Eds.), Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25–27, 2012, USENIX Association, 2012, pp. 15–28.