Large-Scale Causality Discovery Analytics as a Service

Xin Wang, Pei Guo, Jianwu Wang
Department of Information Systems, University of Maryland, Baltimore County, Baltimore, MD, United States
{xinwang11, peiguo1, jianwu}@umbc.edu

***Abstract*—Data-driven causality discovery is a common way to understand causal relationships among different components of a system. We study how to achieve scalable data-driven causality discovery on Amazon Web Services (AWS) and Microsoft Azure cloud and propose a causality discovery as a service (CDaaS) framework. With this framework, users can easily rerun previous causality discovery experiments or run causality discovery with different setups (such as new datasets or causality discovery parameters). Our CDaaS leverages Cloud Container Registry service and Virtual Machine service to achieve scalable causality discovery with different discovery algorithms. We further did extensive experiments and benchmarking of our CDaaS to understand the effects of seven factors (big data engine parameter setting, virtual machine instance number, type, subtype, size, cloud service, cloud provider) and how to best provision cloud resources for our causality discovery service based on certain goals including execution time, budgetary cost and cost-performance ratio. We report our findings from the benchmarking, which can help obtain optimal configurations based on each application's characteristics. The findings show proper configurations could lead to both faster execution time and less budgetary cost.**

***Index Terms*—Causality discovery, XaaS, Cloud computing, Benchmarking, Big data analytics**

I. Introduction

Causality [29] is a fundamental research topic studying cause-effect relationships among different components of a system and causality study can help explain why the system has certain behaviors. Data-driven causality learning/discovery has been widely studied [22] and applied in many disciplines including climatology [31] and neuroscience [15]. For instance, study at [31] shows ENSO phenomenon causes surface air temperature in many remote areas using a data-driven algorithm, which is consistent with climate models.

In this paper, following XaaS (Everything as a Service) principle [17], we study how to provision scalable causality discovery as a service (CDaaS) and mainly address the following challenges. First, with more and more types of cloud services available to use, it is not easy to find the best cloud services for causal discovery from large-scale datasets. Second, it is difficult to know the best configurations (including virtual machine instance type and number) based on different goals such as least budgetary cost, shortest execution time, and minimal cost-performance ratio.

978-1-6654-3902-2/21/$31.00 ©2021 IEEE

To address the above challenges, we design an extensible framework to enable efficient causality discovery from large-scale datasets on the cloud and evaluate it on both Amazon Web Services (AWS) and Microsoft Azure cloud. To the best of our knowledge, our work is the first cloud service for scalable causality discovery. The implementations of our work are open-sourced at [2] and the contributions of the work are summarized below.

- We propose an open-source and extensible causality discovery cloud computing service framework that works on AWS and Azure cloud. We make our service an end-to-end, configurable, and automated pipeline to have good usability.
- To address large-scale available datasets and complicated software dependencies for different causality discovery algorithms, our framework utilizes and integrates container service and distributed virtual cluster to achieve scalable causality discovery by dynamic deployment of the containerized software environment.
- We did extensive experiments to benchmark different configurations of our CDaaS to explore how to best provision resources based on certain goals such as shortest execution times or least budget. We show proper configurations could lead to both faster execution time and less budgetary cost.

The rest of the paper is organized as follows. In Section II, we briefly introduce the cloud services we utilize in our proposed service and our scalable and hybrid causality discovery framework. In Section III, we express the architecture and implementation of the proposed causality analytic as a service framework in detail. Experiments and benchmarking results are discussed in Section IV. We compared our work with related studies in V and conclude in Section VI.

II. Background

A. Related Cloud Services

1) Cloud Virtual Cluster: Cloud virtual cluster consists of a group of nodes hosted on virtual machines (VMs) and connected within one virtual private network. Usually, it can form a cluster from multiple common VMs connected, while managing and performing operations within the whole cluster only through a single master VM. Based on this regular service, most clouds provide their advanced VM cluster services.

As a cloud-based big data platform implemented by Amazon, Elastic MapReduce (EMR) utilizes a hosted Hadoop [33] framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (EC2). Microsoft also has a similar big data infrastructure service called Azure HDInsight [26]. Like EMR, HDInsight also provides open-source tools such as Apache Hadoop, Apache Spark [34], Apache Hive [32] and many other Apache software. One advantage of cloud virtual cluster service is that it is easy to set up, configure, and operate using both graphic user interface (GUI) and command-line interface, such as AWS CLI [11]. It is also reliable with stable software releases. Another good feature is its elasticity, which means users can request any number of instances or containers to run their applications or utilize its auto-scaling ability to let the service manage cluster sizes based on utilization.

2) Virtual Private Cloud: Virtual Private Cloud (VPC) or virtual network enables users to define the isolated, secure, and monitored virtual network for cloud resources. Users can easily set up VPC, like AWS VPC [10], to get full control of the virtual networking environment, with customized cloud network configuration, such as IP address range selection, subnet creation, the configuration of routing table and network gateways. In VPC, a subnet can be either public-facing to have access to the Internet or private-facing without Internet access. Moreover, the level of security can be separately configured in each subnet. For example, in order to connect between VPCs, cloud resources and on-premise networks, one of AWS approaches named PrivateLink [12] provides private connectivity and ensures the security of the network traffic. It is simple to use since there is no need to set up firewall rules, path definitions, or routing table. There are two types of endpoints in PrivateLink: interface VPC endpoints and gateway load balancer endpoints. The former connects to the cloud-hosted services, which is used in our work.

3) Container Registry: A container is a standard unit of software. It makes code and all its dependencies into packages, then it can be used in different computing environments rapidly and reliably without complex configurations. Cloud container registry like Docker Hub [16] and Amazon Elastic Container Registry (ECR) [7] makes it easy to store, manage, share, and deploy container images and artifacts anywhere. There are mainly two choices of cloud container registry: private or public. For private container registry, the container software is privately shared within the organization. However, the public container registry can be found and downloaded by anyone. In our framework, both public and private container registries are supported.

B. Scalable and Hybrid Ensemble Causality Discovery

To discover the cause-effect relationships in a system, many learning approaches exist such as Granger causality [19], PCMCI [30], Dynamic Bayesian Network [27], and Convergent Cross Mapping [38]. For instance, Granger causality [19] defines one-time series X Granger causes another time series Y, if and only if regression-based prediction for Y based on past values of both X and Y is statistically significant than regression-based prediction of Y only based on past values of Y. Causal graph is the most common way to model causal relationships in which vertices/nodes represent variables and directed edges represent causal relationships.

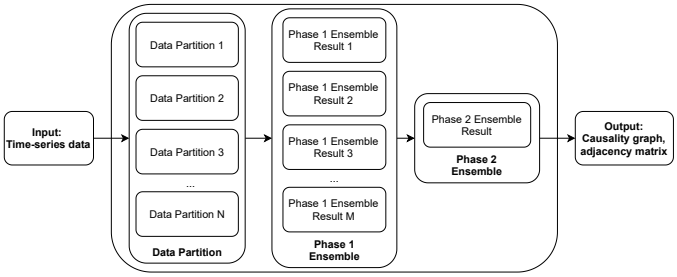


Fig. 1: Two-phase hybrid ensemble framework for causality discovery.

On top of these individual algorithms, our previous work [20], [21] proposed a two-phase hybrid causality discovery ensemble framework, which utilizes data partitioning and ensemble techniques as in Figure 1 to achieve scalability and reduce result uncertainty. The ensemble is achieved by first conducting phase 1 ensemble for partitioned data and then conducting phase 2 ensemble from phase 1 ensemble results. The hybrid framework can combine learning results from different data partitions (namely data ensemble), and different algorithms (namely algorithm ensemble). Two ensemble algorithms are implemented: *data-algorithm ensemble* and *algorithm-data ensemble*. The first one conducts data ensemble in phase 1 and algorithm ensemble in phase 2. The second one conducts data ensemble and algorithm ensemble in the opposite order. To achieve scalability, we further parallelize the ensemble approaches via the Spark big data analytics engine.

III. Causality Discovery as a Service (CDaaS)

To leverage various cloud services and support scalable causal discovery in the cloud, we design a new cloud service called Causality Discovery as a Service (CDaaS). We first introduce the usage pipeline of our proposed CDaaS on AWS as an example. The proposed architecture and its main components are discussed next as our logical implementation, and the physical implementation is introduced followed. To extend CDaaS beyond one specific cloud, we also discuss its extensibility on causality algorithms and cloud platforms. This CDaaS architecture supports the two-phase hybrid ensemble framework in Section II-B well by processing partitioned data in parallel among the worker nodes of CDaaS.

A. CDaaS Usage Pipeline

The usage pipeline of CDaaS is shown in Figure 2. Introduced with AWS cloud as an example, the user first needs to upload or select some time-series datasets. Then the available causality discovery algorithms can be selected. In the next step, users can select a single causal discovery algorithm, or select multiple algorithms to produce ensemble results using

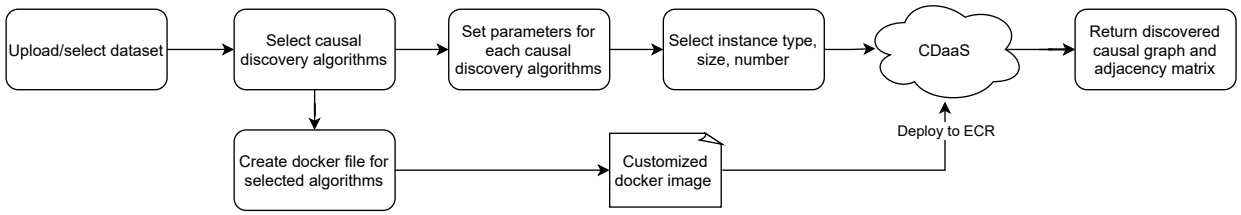


Fig. 2: The causality discovery experiment usage pipeline based on our proposed CDaaS architecture.

our ensemble model explained in Section II-B. After algorithm selection, with the container-based approach, the system will automatically create a docker file containing an essential execution environment based on the selected algorithms, then a customized docker image will be created and deployed to ECR to get prepared. With script-based approach, the pre-defined environment script will be uploaded to the cloud as a bootstrap file for virtual cluster initialization. After algorithm selection, the user is also enabled to set parameters for each selected algorithm. In the next step, the VM instance type, size, and number can also be configured by the user. After these settings are done, the CDaaS will be initialized automatically. When cloud computing of the causality discovery process is done, the results including discovered causal graph and adjacency matrix are returned to the user.

B. CDaaS Architecture

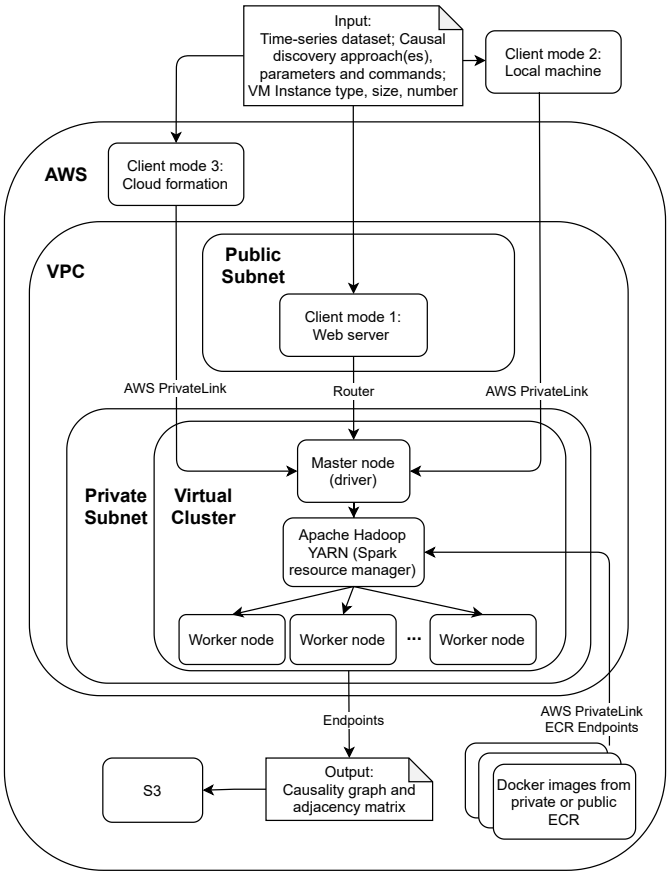


Fig. 3: The architecture of CDaaS on AWS.

We now describe the logical implementation of CDaaS followed by the usage pipeline introduced in Section III-A. Its physical implementation is further introduced in Section III-C and the cloud extensibility is discussed in Section III-D. Our CDaaS' architecture is shown in Figure 3. Its inputs include time-series datasets, causal discovery algorithm selection, and corresponding parameters, AWS virtual machine (VM) instance type, size, number, and other configurations. The final output is the discovered causality graphs and adjacency matrices.

CDaaS supports three client modes for users to interact with: 1) web server, 2) local machine and 3) cloud formation [5]. In web server mode, a web server is deployed in the public subnet to communicate with Internet. It hosts its front-end GUI to get the benchmark configurations from the user and send them to its back-end. The operations that causality will execute in virtual cluster are first defined and packaged within a Lambda function, the function then invoked from the front-end RESTful APIs and WebSocket APIs through API Gateway. After the analytics execution is finished in virtual cluster, the response will be sent to the front-end application with its detail about the causality output. Within the same VPC, web server is connected to the VM cluster through a router because the cluster is inside a private subnet. Another client mode is local machine. The user provides customized data and configurations directly to the VM cluster through an AWS private link. Local machine mode uses the cloud-specific software development toolkit (SDK) to handle the execution of data analytics, which provides flexible management and debugging interface. The local machine mode requires basic programming knowledge about Linux commands, so that program engineers or developers belong to the preferred users of this mode. Cloud formation mode allows users to code their configurations in YAML or JSON directly with sample templates, and use the cloud-specific web-based manager service to manage all causality resources based on these configurations. By doing this, the automation stacks and resources will be orderly and predictably provisioned as a running environment, which enables the version control of our service. Like the web server mode, the cloud formation mode also provides web-based manual operation, which is preferred for end-users since it does not need much knowledge of implementation and programming.

In the virtual private cloud, both the public and private subnet are used in CDaaS architecture. Within our design, public subnet is used to host the web server and private subnet

is created to launch the virtual cluster. We choose private subnet because it is more secure by not exposing the traffic to Internet. Within the private subnet, the VM cannot receive traffic from Internet directly since they do not been allocated the elastic IP addresses [13], which are the static public IPv4 addresses for dynamic cloud computing. Besides, each public and private subnet can be assigned to a security group. The security group of public subnet allows HTTP and HTTPS inbound rules from everywhere through IPv4 or IPv6. Instead, the security group of private subnet only enables all TCP and UDP connections for EMR cluster computation. Specifically, the master instance can be set an additional SSH inbound rule for operation debugging. The outbound rule for both security groups allows all traffic from everywhere as our default setting.

Within the virtual cluster, the application execution environment can be loaded as a docker image saved in the container registry as either the public or private repository. For example in AWS, since the docker image is in ECR and outside the AWS VPC, the interface VPC endpoints, powered by AWS PrivateLink, act as the connector, which enables the resource manager of the cluster to download docker images from the chosen ECR registry. After a container is downloaded, the YARN NodeManager launches it directly inside one docker container on the host machine of the cluster. By using a container-based approach, it is easy to deploy libraries and runtime dependencies for our causality discovery algorithms. Additionally, due to the different resource capabilities among cloud providers, we also provide a general script-based approach. Rather than launching the YARN NodeManager inside the docker, the script-based approach installs the defined software environment in all cluster VMs. These bash files include master and worker-specific scripts, which consisted of the command-line interface for interacting with operating systems so that it can be compatible with the virtual cluster almost in any cloud provider.

The standard storage S3 [9] are used to save input data, execution log, and output on the cloud, and it requires endpoints to communicate with the virtual cluster. The execution measurements can be retrieved for execution evaluation. To download the output to a local machine through the private subnet of the cluster, a network address translation (NAT) [8] gateway is set up for communication between the private subnet and Internet. Finally, the output, which includes causality graphs and adjacency matrix discovered by the causality algorithm, is returned to the end-user.

C. Implementation of CDaaS

Following what is shown in Figure 3, we first achieve the first client mode (web server) by using the AWS Amplify, which helps users setting up continuous deployment and hosting, creating an app back-end in the admin GUI, and managing full-stack environments. Besides the dynamic API calls over HTTP from Amplify, we also provide RESTful APIs in Amazon API Gateway which are optimized for serverless workloads and HTTP back-ends using HTTP APIs. The back-end Lambda function is then invoked by the bound event of API Gateway proxy integration. With our implementation, this Node.js Lambda function enables virtual cluster initialization, software/docker environment setup, and causality execution based on the commands delivered by body field of the HTTP requests. After the causality execution is done, Lambda function will send a callback of the returned status code. Especially, the docker image in the current implementation is one general image containing all supported causality discovery algorithms. It could be further improved by supporting multiple docker images so each causality discovery algorithm can be packed in its own docker image.

We also support end-to-end automatic causality discovery on AWS cloud using client mode 2, including logging of execution time and cost. The automation is built on top of Boto (v2) [1] library, which is an integrated interface to current and future cloud services offered by Amazon Web Services. Specifically, our implementation consists of a set of functions that are specific to AWS services as handy scripts by using Boto, which can be invoked to perform necessary cloud automation operations. For example, in order to perform actions on cloud services and resources, CDaaS connects AWS by creating a Boto session using authentication credentials.

For client mode 3 (cloud formation), we implement AWS CloudFormation templates both in its JSON and YAML specifications. The template file contains configuration information about the cloud resources and pipeline during causality execution. By manually operations on cloud console web page with this template, the virtual machine cluster can be deployed and perform the causality execution.

Within these three client modes, some implementations can be used directly on Azure and other clouds, like VM software/docker environment setup scripts, docker images and files, and the RESTful APIs in front-end environments. Thus for CDaaS in Azure, we only provide the client mode 3, which is implemented by using Azure Resource Manager service. By building the specified JSON template, which includes a description of the resources and their deployment settings, Resource Manager enables the user to deploy and manage resources as a group, and execute causality execution when initialization is finished.

The docker image in the current implementation is one general image containing all supported causality discovery algorithms. It could be further improved by supporting multiple docker images so each causality discovery algorithm can be packed in its own docker image.

D. Extensibility of CDaaS

To achieve easy extensible service, CDaaS extensibility should be considered from two parts, including causality discovery algorithms and cloud platforms. The detail is discussed as follows.

1) Extensibility on causality discovery algorithms: Since our CDaaS is modularized, it is highly extensible. As shown in the pipeline in Figure 2, different causality discovery algorithms can be easily selected separately to conduct causality learning. Regarding our ensemble model explained in Section

TABLE I: Extensibility of CDaaS to other cloud platforms.

Service category	Service description	Amazon AWS	Microsoft Azure	Google Cloud
Big data cluster	Cloud-based big data platform that hosting Hadoop and Spark cluster.	EC2/EMR	VM Scale Set/HDInsight	Compute Engine/Dataproc
Network security	Provide managed and monitored virtual network for resources.	VPC	Virtual Network	Virtual Private Cloud
Container registry	Storing and managing container images.	ECR	Azure Container Registry	Artifact Registry
Object storage	Secure Cloud storage.	S3	Blob Storage	Cloud Storage
Web server	Building, deploying and scaling web applications and services.	Amplify	App Service	App Engine
Python SDK	Easy-to-use interface to access cloud services.	Boto/Boto3	.NET Core	Cloud SDK

II-B, if more than one algorithm is selected, both causal graphs of individual causal discovery algorithms and the causal graph ensemble results from all selected algorithms will be generated. Also, the parameters of specific causality discovery algorithms can also be easily changed and tuned.

Besides, new causality discovery algorithms could be easily added to the system. A new container and corresponding scripts will be set up to support the additional algorithms, then be deployed to container registry and directly run on the virtual cluster.

2) Extensibility on cloud platforms: CDaaS can be extensible to other cloud platforms. Most cloud services provided by AWS, Azure and Google Cloud can be mapped to each other. We mainly talk about the CDaaS architecture and pipeline on AWS in this section, but our work is not tightly coupled with AWS definitely. To prove the extensibility, we also implement the partial CDaaS architecture in Azure cloud, and its implementation is already introduced in Section III-C. For cloud services we used in CDaaS, other providers also have very similar services. For each cloud service CDaaS used, we list its corresponding services in different cloud platforms in Table I.

IV. Experiments and Benchmarking

In this section, we conduct various experiments to benchmark how different factors affect the execution time, budgetary cost, and cost-performance ratio of our CDaaS. The factors we investigated include Spark submission parameters, VM instance type and sub-type, VM instance number, VM instance size, cloud service and cloud provider. We summarize our findings at the end of each comparison. The findings are useful to find the optimal configurations based on each application's characteristics.

A. Environment Setup

1) Software and dataset: The algorithm in our benchmarking is the Spark-based scalable two-phase hybrid *algorithm-data ensemble* causality discovery algorithm described in Section II-B. The ensemble contains three data-driven causality discovery algorithms: Multivariate Granger causality, Dynamic Bayesian Network, and PCMCI. The data in our experiment is 10M rows of simulated five variable time-series records, which is the same as the one used in our previous work [21]. Its data size is around 500 MB.

We present causality discovery analytics in both AWS and Azure platform. For causality on AWS, all VM instances in the cluster utilize the EMR release version *emr-6.0.0 with Spark application*, which includes Spark 2.4.4 on Hadoop 3.2.1 YARN with Ganglia 3.7.2 and Zeppelin 0.9.0-SNAPSHOT. For causality on Azure, since HDInsight service does not support docker-based Spark computation, we instead run causality in the script-based approach on both clouds for performance comparison.

The docker environment we implemented is hosted on ECR in public repository and Docker Hub, with Python 3.7 and R 3.4. The driver's memory is utilized as much as we can, around 80% of the instance memory. The deploy mode of Spark is the *cluster* with YARN resource manager. The data partition level is 240. The same metric used in our previous work, namely hamming distance, is employed to measure the quality of the results. All experiments produce the same causal graphs as output because all experiments use the same data partition number, which will generate the same number of Spark tasks. The differences of the experiments are in how and where the tasks are executed.

2) Hardware: We mainly introduce the hardware of experiments on AWS. On Azure, we use the VM cluster with the same computational capability for different cloud evaluations. Since both docker and script-based approaches are run on top of EC2, we discuss EC2 VM cluster types in our benchmarking experiments. The prices and resources of the VM instance types are shown in Table II. The prices in the table are on-demand rates. The experiments are executed on EC2 general purpose instances $m5$, memory optimized instances $r5$ and compute optimized instances $c4$, $c5$ and $c5n$, since EC2 clusters are designed with several focused purposes. The instances sizes we selected are $xlarge$, $2xlarge$, and $4xlarge$. All instances utilize Elastic Block Store (EBS) [6] as storage and initiated in US-West-2 (Oregon) region. Enhanced networking is enabled to get significantly higher packet per second (PPS) performance, lower network jitter and lower latency.

B. Measurement

Like related work such as [25], we also use the following three metrics to measure our CDaaS.

1) Budgetary cost: When computing the budgetary cost, we break up the cost of each component and compute cost into four main parts: VM cost, big data service cost, VPC cost, and EBS cost. Let C indicate cost, t be the execution time, p be the price of the service, k be the instance number, and s as data size, the cost function will be as follows.

$$\begin{aligned} C_{total} = C_{VM}(t \times p \times k) + C_{BigData}(t \times p \times k) \\ + C_{VPC}(p \times t) + C_{EBS}(s \times p) \end{aligned} \quad (1)$$

In our cost function, namely Equation (1), the hourly rate of VM (EC2) and big data service (EMR) on AWS already

TABLE II: The prices and resources of different instance types and sizes utilized on AWS.

Instance type	Instance sub-type	Instance size	EC2 hourly price ($)	EMR hourly price ($)	vCPU	Memory (GB)
General purpose	m5	xlarge	0.192	0.048	4	16
Memory optimized	r5	xlarge	0.252	0.063	4	32
Compute optimized	c5	xlarge	0.170	0.043	4	8
		2xlarge	0.340	0.085	8	16
		4xlarge	0.680	0.170	16	32
	c4	xlarge	0.199	0.052	4	7.5
		2xlarge	0.398	0.105	8	15
		4xlarge	0.796	0.210	16	30
	c5n	xlarge	0.216	0.054	4	10.5
		2xlarge	0.432	0.108	8	21
		4xlarge	0.864	0.216	16	42

shown on Table II, respectively. We note that even the cloud uses hourly prices, the bills we received show the calculation is done based on exact hour time in fraction (e.g., 3.14), not rounded-up integer hour numbers (e.g., 4). The costs of these two parts are related to the hour and the instance number. The VPC price is $0.01 per virtual private cloud endpoint hour. EBS is a high-performance and block-storage service for VM cluster, with the cost of $0.10 per GB-month of general purpose SSD (gp2) provisioned storage.

The above formula only contains the main costs of cloud services. The minor costs are negligible compared to the main costs and many of them are charged by accumulated usage, not the individual one. Specifically, the minor costs we did not count are: 1) the regional data transfer bandwidth cost, which includes in/out/between VM availability zones or using elastic IPs or elastic load balancing ($0.01 per GB); 2) the container registry storage cost, calculated by the number of GB-hours that data was stored in the storage ($0.10 per GB-month); 3) all API request cost, including $0.01 per Cost Explorer API request, $0.03 per 10000 Key Management Service API request, $0.05 per 10000 Secrets Manager API request, and $0.50 per 1,000,000 Simple Notification Service API request; and 4) standard storage cost, calculated by the number of GB-hours that data was stored in the storage ($0.023 per GB-month).

2) Execution time: When we record the total CDaaS execution time, we use wall-clock time, which includes four steps: 1) VPC endpoints creation, 2) VM cluster initialization, 3) VM software preparation, and 4) execution of causality discovery algorithms. VPC endpoints creation step runs the creation command and waits for all required endpoints are ready to be used. VM cluster initialization time starts from creating the cluster to its status becoming waiting. VM software preparation step includes downloading data to cluster, updating built-in software libraries used by the cluster, installing additional required packages, and getting connected to cloud container registry to get ready for Spark job submission. Causality discovery execution time starts from the submission of the Spark job to the end of the execution. After that, the logs are downloaded then the cluster and all endpoints are shut down. From all experiments we conducted, we draw the box-plots of time taken for these four steps in Figure 4. The figure shows the time taken by the first three steps is relatively constant and much shorter than the time for the fourth step. The time range of the first three steps accounts for 8.3%, 3.3%, and 1.7% of the total, respectively. In contrast, the range of algorithm execution is around 0.5 hours, which accounts for around 83.3% of the total, which is very large compared to the previous three steps. So the causality discovery execution step is the main reason for the wall-clock time differences.

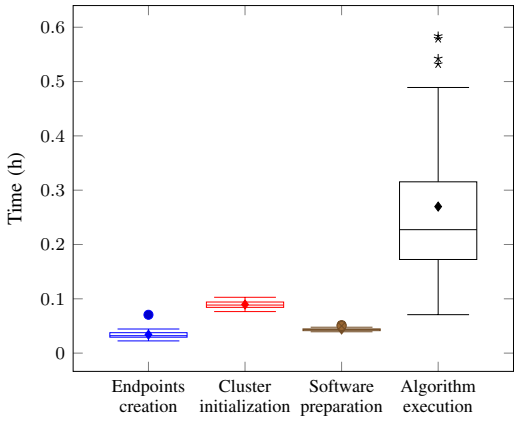


Fig. 4: The box-plots for the time taken by different steps of our CDaaS.

3) Cost-performance ratio: Regarding the cost-performance ratio computation, suppose the cost-performance ratio is α, the wall-clock time of the whole experiment is t, the performance of the program is p, the computed cost is c, then the ratio equation is below. It is more desirable with the lower α, excluding other factors.

$$\alpha = \frac{c}{p} = \frac{c}{\frac{1}{t}} = c \times t \tag{2}$$

C. Benchmarking Results

We did benchmark to understand the effects of the following seven factors: 1) Spark parameter setting, 2) VM instance number, 3) VM instance type, 4) VM instance subtype, 5) VM instance size, 6) cloud service, and 7) cloud providers. The first five factors are evaluated only on AWS cloud with container-based approach, and the last two are evaluated on both AWS and Azure with script-based approach. For each factor, we measure from three perspectives: execution time, budgetary

cost, and cost-performance ratio. We report our findings at the end of each comparison.

The same metric used in our previous work [20], namely hamming distance, is employed to measure the quality of the results. All experiments produce the same causal graphs as output because all experiments use the same data partition number (240), which will generate the same number of Spark tasks. The differences of the experiments are at how and where the tasks are executed.

1) Comparison of Spark dynamic vs static parameter settings: We experimented with three groups of Spark parameter settings: 1) dynamic allocated executor, 2) static with 2 executors per worker nodes, and 3) static with 4 executors per worker nodes. Dynamic allocated executor means the number of executors and the resource is dynamically scaled up and down based on the workload. The upper bound and lower bound of dynamic executor numbers are infinity and zero, respectively. Static settings keep the executor number, memory, and cores fixed during the experiments. The memory of each driver/worker node are static in each set of experiments. Each parameter setting is executed on 4, 6 and 8 worker nodes using m5.xlarge cluster (Figure 5), r5.xlarge cluster (Figure 6) and c5.xlarge cluster Figure 7), respectively. In each figure, the cost comparison, time comparison and cost-performance ratio comparison are plotted.

For m5.xlarge instance, as shown in Figure 5a and Figure 5b, dynamic executor always has the lowest cost and the shortest wall-clock time, compared to the static of both 2 and 4 executors per node. The results show the same outcome for experiments we have done using different worker instance numbers (4, 6 and 8 worker instances). Thus, the cost-performance ratio is lowest for dynamic executors as shown in Figure 5c, since it is both cheapest and fastest. The case is also the same with r5.xlarge and c5.xlarge experiments. **Finding 1: our experiments conclude dynamic executor allocation is always the best for our Spark-based big data applications in all three metrics (budgetary cost, execution time and cost-performance ratio).** Because of this, in the following experiments, all jobs are executed with dynamic allocated executors.

2) Comparison of VM instance numbers: When comparing how CDaaS performs with different number of VM instances, we still refer to figures as in the previous experiment: m5.xlarge cluster (Figure 5), r5.xlarge cluster (Figure 6) and c5.xlarge cluster (Figure 7), respectively. The focus is on dynamic executor settings. When looking at m5.xlarge clusters, the lowest cost is with 6 work nodes, while the shortest wall-clock time and lowest cost-performance ratio all appear to be with 8 worker nodes. And the worst performance and wall-clock time is with 4 worker nodes. So in our experiments, with a general purpose cluster, the more nodes we use, the faster it executes with a lower cost-performance ratio.

Previous studies such as [37] have shown it is difficult to improve both execution time and budgetary cost of cloud services. In our experiment, comparing using 4 worker nodes and 8 worker nodes, not only the execution time, the cost decreases. To analyze this seemingly impossible phenomenon, we break down the cost and time of m5.xlarge instance with 4 and 8 worker nodes in Table III and Table IV respectively. Because the data partition level in our program is 240, it means our execution can still achieve a high level of parallelization when the worker node number increases from 4 to 8. Table IV shows the algorithm execution time difference is about 2.5 times (0.479 hours for 4 nodes and 0.188 hours for 8 nodes). It means the improvement in the performance of scaling up from 4 to 8 worker nodes is better than linear. Because the algorithm execution time is dominant in total time, the speedup of the execution results in less cost of EC2, EMR, and VPC (as shown in Table III). EBS cost is the same because it does not depend on time. In summary, the speedup of the execution overweighs the additional cost with more worker nodes, which results in less cost, better execution time, and thus better cost-performance.

TABLE III: Total cost and the components of m5.xlarge instance with dynamic executor allocation (unit: $).

Worker instance number	EC2	EMR	VPC	EBS	Total
4	0.592	0.148	0.006	0.140	0.887
8	0.564	0.141	0.003	0.140	0.848

TABLE IV: Total time and the components of m5.xlarge instance with dynamic executor allocation (unit: hour).

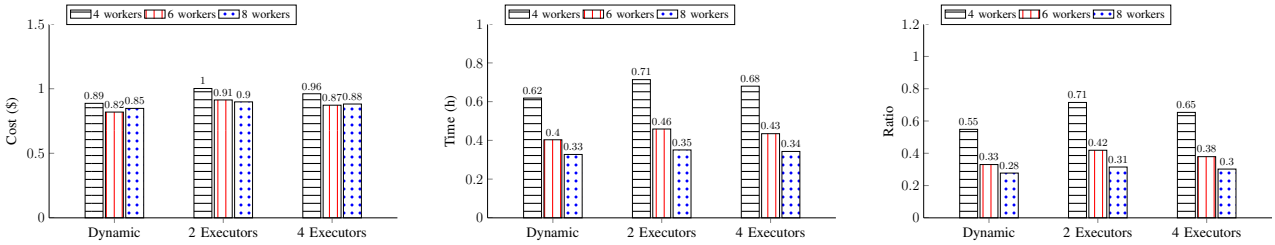
Worker instance number	Endpoints creation	Instance initialization	Instance preparation	Algorithm execution	Wall-clock time
4	0.035	0.095	0.043	0.479	0.618
8	0.032	0.093	0.045	0.188	0.327

Similarly, in r5.xlarge and c5.xlarge instance experiments, the lowest cost is with 6 worker nodes. But the best wall-clock time and the best cost-performance ratio are with 8 worker experiments. So with these purpose-optimized clusters, the trade-off between a lower cost and a lower cost-performance ratio also exists. The decision of the number of instances will be made differently based on a budget-focused view or a cost-performance-focused view. **Finding 2: within a sufficient level of parallelization on worker nodes, our experiments show both execution time and cost-performance ratio improve with a larger number of instances. A larger instance number could also lead to less budgetary cost for certain instance types.**

3) Comparison of general different instance types: Our next experiments focused on the performance comparison across the different instance types. The results of costs, wall-clock time and cost-performance ratio can be found in Figure 8.

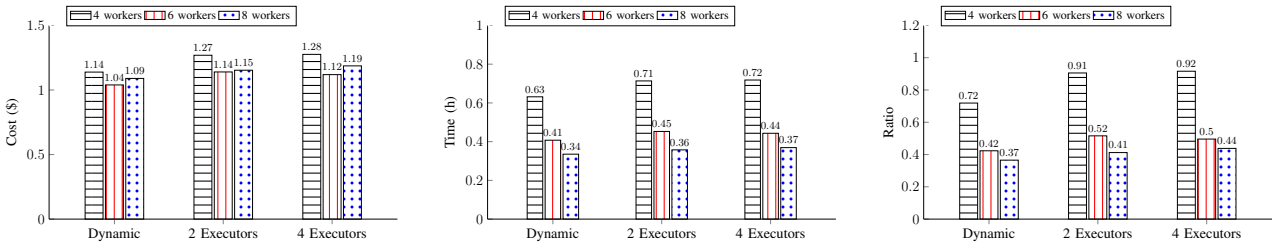
When comparing the cost from Figure 8a, the lowest one appears to be with 6 workers in c5.xlarge. Figure 8b shows that the shortest wall-clock time is c5.xlarge with 8 worker nodes. And the best cost-performance ratio is c5.xlarge with 8 worker nodes.

The experiments indicate that the compute optimized instances, namely c5 clusters, achieve the best cost-performance ratio compared to general purpose instances (m5) and memory optimized instances (r5). The reason is that our CDaaS is



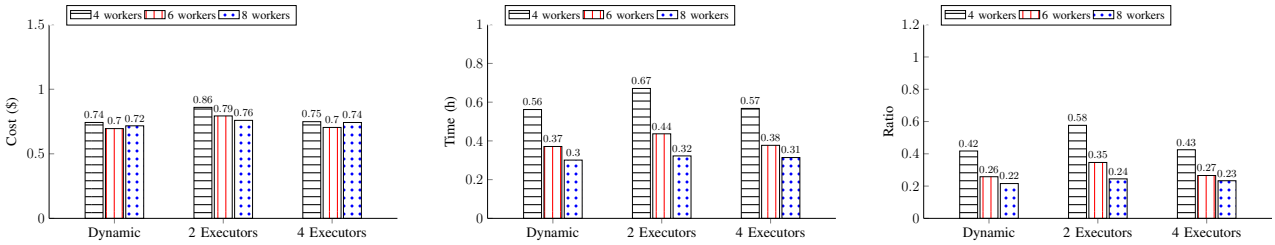
(a) Cost comparison. (b) Time comparison. (c) Ratio comparison.

Fig. 5: m5.xlarge: Comparison of Spark dynamic vs static parameter settings.



(a) Cost comparison. (b) Time comparison. (c) Ratio comparison.

Fig. 6: r5.xlarge: Comparison of Spark dynamic vs static parameter settings.



(a) Cost comparison. (b) Time comparison. (c) Ratio comparison.

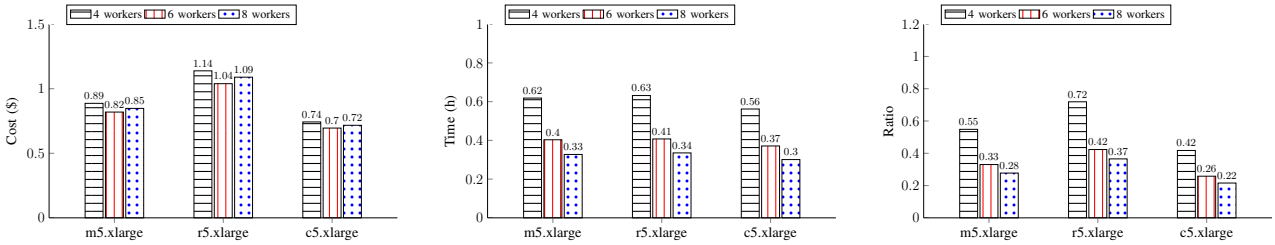
Fig. 7: c5.xlarge: Comparison of Spark dynamic vs static parameter settings.

compute-intensive. It achieves better performance with a more advanced CPU. General purpose m5 clusters perform a little bit better than memory optimized r5 clusters here, the reason is the CPU of m5 clusters is slightly better than that of r5 clusters. Also, our service is not memory intensive, so that larger memory did not help much. **Finding 3: results show that for compute-intensive applications, compute-intensive instance type gives better execution performance, lower cost, therefore also better cost-performance ratio.**

4) Comparison of different instance sub-types: The experiments of comparison of different instance sub-types are executed on compute-optimized clusters since it performs the best in the previous comparison to general purpose and memory optimized clusters. We select three instance sub-types in compute-optimized instances: c4, c5, and c5n. The experiments are executed on these three sub-types, with three different instance sizes (xlarge, 2xlarge, and 4xlarge) with 8 worker nodes. The results are shown in Figure 9. The lowest cost appears on c5.xlarge, and the highest performance is achieved by c5n.4xlarge. Here c5n performs better because this sub-type has a larger bandwidth than c5 instances. Also, c5n and c5 have better CPUs than c4 instances. The best cost-performance ratio is achieved by c5.xlarge instance. Generally speaking, in our experiments, the c4 instances are relatively more expensive and slower than the c5 instances. **Finding 4: sub-types with faster CPU and larger bandwidth can speed up the experiments, but the cost is also higher. The trade-off between cost and performance exists among VM instances sub-types.**

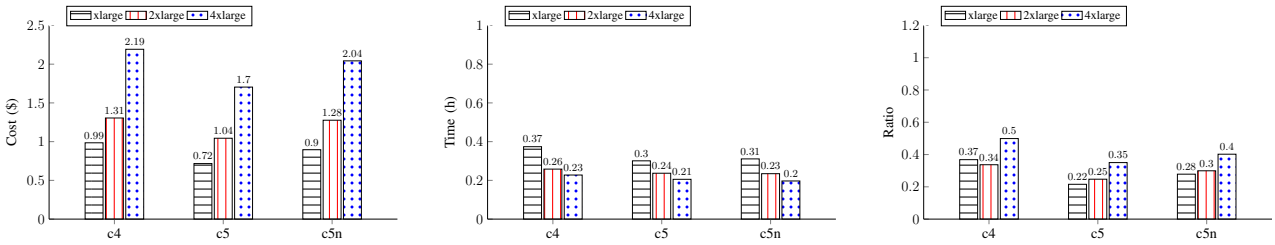
5) Comparison of different VM instance sizes: Since c5 instance gets best the cost-performance ratio in the previous experiment, we further analyze the influence of different VM instance sizes on cost, time, and cost-performance ratio focusing on three different sizes (xlarge, 2xlarge, and 4xlarge), as shown in Figure 10. For looking into the cost components, we break down the costs as in Equation (1) illustrated in Figure 10a. The cost components indicate that the EC2 cost is the largest part. Moreover, since the cost of VPC is less than $0.01 so the plot shows 0. The detailed costs of each component in different instance sizes can be found in Table V.

The results show that c5.xlarge has the lowest cost, and c5.4xlarge has the lowest wall-clock time. The best cost-performance ratio is c5.xlarge, which shows that 4 times less resources brought around 30% improvement comparing with



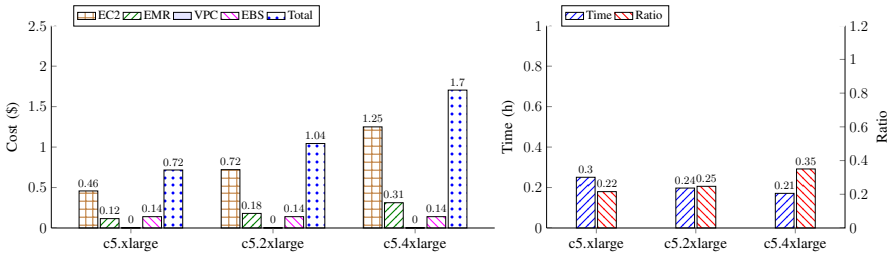
(a) Cost comparison. (b) Time comparison. (c) Ratio comparison.

Fig. 8: Comparison of different instance types.



(a) Cost comparison. (b) Time comparison. (c) Ratio comparison.

Fig. 9: Comparison of different compute optimized instances.



(a) Cost breakdown comparison. (b) Time and ratio comparison.

Fig. 10: Comparison of different instance sizes.

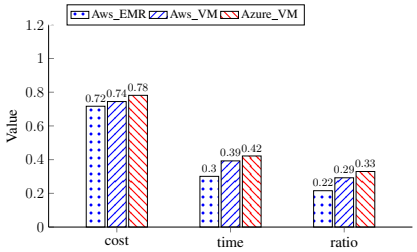


Fig. 11: Comparison of different cloud services and providers.

TABLE V: Total cost and the components of c5 clusters with xlarge, 2xlarge and 4xlarge sizes (unit: $).

Instance size	EC2	EMR	VPC	EBS	Total
xlarge	0.458	0.116	0.003	0.140	0.718
2xlarge	0.721	0.180	0.002	0.140	1.044
4xlarge	1.250	0.312	0.002	0.140	1.704

c5.4xlarge scenario. So the trade-off still exists in the instance size selection that a larger size of instance gives better performance, but it is more expensive. **Finding 5: our experiment indicates that for compute-intensive applications, a larger instance size performs better with higher cost. The trade-off between cost and performance exists.**

6) Comparison of different cloud services and providers: We compare two additional factors in this experiment: 1) differences between EMR and EC2 based virtual environment in AWS, 2) differences between AWS and Azure. Since Azure big data service HDInsight currently does not support docker-based Spark computation, we instead run the experiments on both AWS and Azure platform using script-based approach. According to previous findings, we use 8 workers with c5.xlarge VM on AWS and Standard_F4s_v2 VM on Azure, and the Spark application only runs with dynamic executor setting. The VM used in both AWS and Azure is allocated in 4vCPUs with 8GIB Memory. Figure 11 shows the comparison between the three methods: EMR based approach for AWS, and script-based approaches for both AWS and Azure. The figure shows using EMR service achieves better cost, time and ratio compared to benchmarking without EMR. We note EMR service involves extra costs because of its additional capabilities on top of EC2 service. We believe the reason EMR achieves less total cost is because the costs saved by not using EMR service were less than the costs increased by additional time caused by script-based approach. One advantage of using script-based approach is the same scripts can be used in different clouds (such as AWS and Azure). Additionally, when comparing with different cloud providers among AWS and Azure, experiments in AWS achieve lower cost, time and cost-performance ratio, but did not cause much difference, whose cost, time and ratio achieving accounts for 4.9%, 7.4% and 12.8% better performance, respectively. **Finding 6: for our CDaaS, the execution performs a little better on AWS cloud compared with Azure and using AWS EMR performs better than script-based approach in AWS.**

D. Benchmarking Summary

The benchmarking experiments show that the dynamic executor allocation is better than static executor configurations in running Spark-based applications in cost, time, and cost-performance ratio. With sufficient parallelism, a larger number of instances could also help all three measurements. So we could achieve both faster execution time and less budgetary cost with proper configuration of Spark executor allocation option and VM instance number. For compute-intensive models, compute-intensive instance gives better performance and lower cost. However, the compute optimized sub-type with better CPU and larger bandwidth performs better with higher cost. In addition, a larger instance size performs better with higher costs. The trade-off between better cost and better performance exists in these two scenarios. Also, CDaaS in AWS with EMR performs better than it without EMR, and AWS performs a little better in experiments compared with Azure.

Our experiments also show different factors could have significant effects on execution time, budgetary cost, and cost-performance ratio and no configuration can achieve the best for all three measurements. In our experiments, the worst execution time (0.718 hours with configuration of 4 executors per node, 4 worker nodes on r5.xlarge instances) is 3.64 times as long as the shortest one (0.197 hours with configuration of dynamic executors, 8 worker nodes on c5n.4xlarge instances). In terms of budgetary cost, the most expensive cost ($2.192 with configuration of dynamic executors, 8 worker nodes on c4.4xlarge instances) is 3.15 times as much as the least expensive one ($0.695 with configuration of dynamic executors, 6 worker nodes on c5.xlarge instances). In terms of cost-performance ratio, the worst ratio (0.916 with configuration of 4 executors per node, 4 worker nodes on r5.xlarge instances) is 4.24 times as large as the best one (0.216 with configuration of dynamic executors, 8 worker nodes on c5.xlarge instances).

V. Related Work

A. Causality Discovery Services and Platforms

One causality service [4], called Causal Web, is developed to help discover valid, novel, and significant causal relationships from big biomedical data that lead to new biomedical insights. Another causality discovery benchmark platform is called CauseMe [3] and allows users to test their causal discovery algorithms accuracy against various datasets provided by the platform. The difference between these services and ours are: 1) they are not cloud services, which makes them hard to reproduce specific causal discovery processes; 2) their supports for big data are limited because they do not leverage parallel computing or big data platforms for scalable execution; 3) they only support individual causal algorithms while ours supports both individual causal discovery algorithms and the ensemble of multiple causal algorithms.

B. Cloud-based Benchmarking for Big Data Analytics

Because of the on-demand and elasticity feature of cloud computing, the cloud has become a popular environment to conduct computational science [28] including big data analytics [24], [35], [39], [40]. Users can easily spin up a virtual big data cluster in the cloud, run their big data applications, and then dismantle the environment. There have been also some studies about big data analytics benchmarking in the cloud. In paper [36], for considering both hot/cold startup and hot/cold shutdown of VMs, the authors establish a multi-objective model to benchmark both performance and cost of cloud computing platform. VISCERAL [23] developed a cloud-based framework for experimentation of big data analytics focusing on the quality of prediction, which is more aiming to reduce the complexities and barriers to running experiments on huge representative datasets. PRIMEBALL [18] achieved a complete and unified benchmark for measuring parallel cloud processing frameworks for big data applications, whose strengths are parallelization capabilities supporting cloud features and big data properties. The differences between our work and these related studies can be summarized as follows. First, our architecture uses and integrates existing cloud services as much as possible so that our implementation is much easier to maintain. It can benefit from the automatic upgrades of the cloud services such as software library version upgrades. Second, we conducted more extensive benchmarking experiments by investigating seven factors (big data engine parameter setting, virtual machine instance number, type, subtype, size, cloud service, cloud provider) and the findings from our benchmarking can help users find the best configuration for their own big data application.

VI. Conclusion

To fill the gap between causality discovery and XaaS on the cloud, we develop our causality discovery analytics as a service (CDaaS) on the cloud. We propose an end-to-end framework fully utilizing cloud services and implement an automation process for users to execute scalable and hybrid causality discovery on the cloud. Further, we conducted extensive experiments to understand how to best provision our CDaaS on clouds. For the same causality discovery application from the same dataset, different configurations result in a 3-5 times difference in terms of execution time, budgetary cost, and cost-performance ratio. Previous studies such as [37] have shown execution time and budgetary cost of cloud services often conflict with each other and it is difficult to improve both. Our benchmarking shows that we could improve execution time and budgetary cost with proper configurations.

For future work, we will mainly focus on the following two aspects. First, we will extend our work to easily publish causal discovery application executions as public records following the Research Object framework [14] so they can be referred via DOI identifiers later. Second, we will study how to utilize cloud service scheduling techniques [37] to achieve automatic optimization based on users' objectives, dataset, and algorithm selection based on our benchmarking findings.

ACKNOWLEDGMENT

This work is supported by grant CAREER: Big Data Climate Causality Analytics (OAC–1942714) and grant Cyber-Training: DSE: Cross-Training of Researchers in Computing, Applied Mathematics and Atmospheric Sciences using Advanced Cyberinfrastructure Resources (OAC–1730250) from the National Science Foundation.

REFERENCES

[1] Boto: A python interface to amazon web services. https://github.com/boto/boto, 2018. Accessed: 2018-07-11.

[2] Causality discovery as a service. https://github.com/big-data-lab-umbc/Causality_Discovery_as_a_Service, 2021. The corresponding Docker image: https://gallery.ecr.aws/e0a3g4z6/causality-ensemble and https://hub.docker.com/r/starlyxxx/causality-ensemble-dockerhub. Accessed: 2021-10-20.

[3] Causeme: A platform to benchmark causal discovery methods. https://causeme.uv.es/, 2021. Accessed: 2021-04-13.

[4] Welcome to ccd docs. https://bd2kccd.github.io/docs/tetrad/, 2021. Accessed: 2021-04-13.

[5] Amazon Web Services, Inc. Amazon CloudFormation. https://aws.amazon.com/cloudformation/, 2021. Accessed: 2021-04-13.

[6] Amazon Web Services, Inc. Amazon Elastic Block Store (EBS). https://aws.amazon.com/ebs/, 2021. Accessed: 2021-4-13.

[7] Amazon Web Services, Inc. Amazon Elastic Container Registry (ECR). https://aws.amazon.com/ecr/, 2021. Accessed: 2021-4-13.

[8] Amazon Web Services, Inc. Amazon network address translation (NAT) gateways. https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html, 2021. Accessed: 2021-4-13.

[9] Amazon Web Services, Inc. Amazon Simple Storage Service (S3). https://aws.amazon.com/s3/, 2021. Accessed: 2021-4-13.

[10] Amazon Web Services, Inc. Amazon Virtual Private Cloud (VPC). https://aws.amazon.com/vpc/, 2021. Accessed: 2021-4-13.

[11] Amazon Web Services, Inc. AWS Command Line Interface. https://aws.amazon.com/cli/, 2021. Accessed: 2021-4-13.

[12] Amazon Web Services, Inc. AWS PrivateLink. https://aws.amazon.com/privatelink/, 2021. Accessed: 2021-4-13.

[13] Amazon Web Services, Inc. Elastic IP addresses . https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html, 2021. Accessed: 2021-4-13.

[14] S. Bechhofer, I. Buchan, D. De Roure, P. Missier, J. Ainsworth, J. Bhagat, P. Couch, D. Cruickshank, M. Delderfield, I. Dunlop, et al. Why linked data is not enough for scientists. *Future Generation Computer Systems*, 29(2):599–611, 2013.

[15] M. Ding, Y. Chen, and S. L. Bressler. Granger causality: Basic theory and application to neuroscience. In B. Schelter, M. Winterhalder, and J. Timmer, editors, *Handbook of Time Series Analysis*, pages 437–460. Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim, Germany, Sept. 2006.

[16] Docker, Inc. Docker Hub. https://www.docker.com/products/docker-hub, 2021. Accessed: 2021-4-13.

[17] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu. Everything as a service (xaas) on the cloud: origins, current and future trends. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 621–628. IEEE, 2015.

[18] J. Ferrarons, M. Adhana, C. Colmenares, S. Pietrowska, F. Bentayeb, and J. Darmont. Primeball: a parallel processing framework benchmark for big data applications in the cloud. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 109–124. Springer, 2013.

[19] C. W. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3):424–438, 1969.

[20] P. Guo, Y. Huang, and J. Wang. Scalable and flexible two-phase ensemble algorithms for causality discovery. *Big Data Research*, 26:100252, 2021.

[21] P. Guo, A. Ofonedu, and J. Wang. Scalable and hybrid ensemble-based causality discovery. In *2020 IEEE International Conference on Smart Data Services (SMDS)*, pages 72–80, 2020.

[22] R. Guo, L. Cheng, J. Li, P. R. Hahn, and H. Liu. A survey of learning causality with data: Problems and methods. *ACM Computing Surveys (CSUR)*, 53(4):1–37, 2020.

[23] A. Hanbury, H. Müller, G. Langs, and B. H. Menze. Cloud–based evaluation framework for big data. In *The Future Internet Assembly*, pages 104–114. Springer, 2013.

[24] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li. Big data processing in cloud computing environments. In *2012 12th international symposium on pervasive systems, algorithms and networks*, pages 17–23. IEEE, 2012.

[25] K. Li. Quantitative modeling and analytical calculation of elasticity in cloud computing. *IEEE Transactions on Cloud Computing*, 8(4):1135–1148, 2017.

[26] Microsoft Azure Cloud, Inc. Microsoft Azure HDInsight. https://azure.microsoft.com/en-us/services/hdinsight/, 2021. Accessed: 2021-4-13.

[27] K. P. Murphy and S. Russell. Dynamic bayesian networks: representation, inference and learning. 2002.

[28] National Academies of Sciences, Engineering, and Medicine. *Reproducibility and Replicability in Science*. The National Academies Press, Washington, DC, 2019.

[29] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[30] J. Runge, P. Nowack, M. Kretschmer, S. Flaxman, and D. Sejdinovic. Detecting causal associations in large nonlinear time series datasets. https://arxiv.org/abs/1702.07007v2, 2018. Accessed: 2018-06-28.

[31] H. Song, J. Tian, J. Huang, P. Guo, Z. Zhang, and J. Wang. Hybrid causality analysis of enso's global impacts on climate variables based on data-driven analytics and climate model simulation. *Frontiers in Earth Science*, 7:233, 2019.

[32] The Apache Software Foundation. APACHE HIVE TM. https://hive.apache.org/, 2021. Accessed: 2021-04-13.

[33] The Apache Software Foundation. Homepage — Apache Hadoop Project. http://hadoop.apache.org, 2021. Accessed: 2021-04-13.

[34] The Apache Software Foundation. Homepage — Apache Spark Project. http://spark.apache.org, 2021. Accessed: 2021-04-13.

[35] S. Tsuchiya, Y. Sakamoto, Y. Tsuchimoto, and V. Lee. Big data processing in cloud environments. *Fujitsu Sci. Tech. J*, 48(2):159–168, 2012.

[36] B. Wan, J. Dang, Z. Li, H. Gong, F. Zhang, and S. Oh. Modeling analysis and cost-performance ratio optimization of virtual machine scheduling in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 31(7):1518–1532, 2020.

[37] J. Wang, P. Korambath, I. Altintas, J. Davis, and D. Crawl. Workflow as a service in the cloud: architecture and scheduling algorithms. *Procedia computer science*, 29:546–556, 2014.

[38] H. Ye, E. R. Deyle, L. J. Gilarranz, and G. Sugihara. Distinguishing time-delayed causal interactions using convergent cross mapping. *Scientific reports*, 5:14750, 2015.

[39] E. Zdravevski, P. Lameski, A. Dimitrievski, M. Grzegorowski, and C. Apanowicz. Cluster-size optimization within a cloud-based etl framework for big data. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3754–3763. IEEE, 2019.

[40] F. Zulkernine, P. Martin, Y. Zou, M. Bauer, F. Gwadry-Sridhar, and A. Aboulnaga. Towards cloud-based analytics-as-a-service (claaas) for big data analytics in the cloud. In *2013 IEEE International Congress on Big Data*, pages 62–69. IEEE, 2013.