2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 978-1-6654-3577-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/IPDPSW52791.2021.00052

Characterizing Job-Task Dependency in Cloud Workloads Using Graph Learning

Zhaochen Gu
Computer Sci. & Engg. Dept.
University of North Texas
Denton, Texas
ZhaochenGu@my.unt.edu

Sihai Tang
Computer Sci. & Engg. Dept.
University of North Texas
Denton, Texas
SihaiTang@my.unt.edu

Beilei Jiang
Computer Sci. & Engg. Dept.
University of North Texas
Denton, Texas
BeileiJiang@my.unt.edu

Song Huang
Big Data Development
Allstate
Irving, Texas
SongHuang@my.unt.edu

Qiang Guan
Computer Science Department
Kent State University
Kent, Ohio
qguan@kent.edu

Song Fu
Computer Sci. & Engg. Dept.
University of North Texas
Denton, Texas
Song.Fu@unt.edu

***Abstract*—Modeling and scheduling diverse and dynamic workloads effectively has become a crucial issue due to the ever increasing scale and complexity of systems and applications in modern data centers. A large-scale cloud system consists of a large number of computing nodes, storage nodes and networking devices, running diverse workloads. Existing works analyzed execution traces in terms of resource usage by applying statistical methods. Cloud workloads, especially batch jobs, are composed of tens to thousands of tasks with complex dependency which can be represented by directed acyclic graphs (DAGs). Those workloads and their dependencies have not been thoroughly studied. Understanding the characteristics of batch cloud workload helps us foresee resource demands and execution time of new jobs and make better decisions in job scheduling. In this paper, we investigate batch jobs in production cloud computing environments with dependencies from the perspective of topological characteristics and structural patterns. We design a graph learning approach for job classification based on jobs' *topological similarity*. We evaluate our methods using traces collected from a production data center and discover insightful properties and patterns of batch jobs in real-world scenarios. To the best of our knowledge, this is the first such work that leverages graph learning to explore the topological structures for cloud workflow for characterization and analysis.**

***Index Terms*—Cloud computing, Workload characterization, Graph learning, Job dependency, Classification.**

I. Introduction

Applications run on large-scale cloud computing platforms contain various numbers of tasks with dependencies that can be expressed by directed acyclic graphs (DAGs). DAG jobs are widely seen in big data analytics workloads. There is a dependency hierarchy in these batch jobs. However, the composition of DAG batch jobs can differ significantly. Then the question comes up: How can we characterize these various workflows in an effective manner that helps maximize the efficiency of task scheduling and resource utilization? To answer this question, we consider two aspects in cloud workload analysis: resource management and topological structure-based optimization.

There are existing works on resource utilization analysis for cloud workloads. Conventional statistical analysis approaches have been applied to characterize workload on production clouds [4], [12], [14]. However, the dependencies in batch jobs have not been analyzed in depth. Existing works do not consider the structural patterns and resource needs of multiple jobs co-run on a node. There are gaps between the traditional resource analysis and graph learning based characterization scheduling, which provides a holistic view among jobs in cloud.

In this paper, we aim to bridge this gap and provide an in-depth analysis of DAG-described batch workloads to enable effective job scheduling decisions in large-scale, co-located cloud environments. We not only explore the inner properties of task dependencies among jobs, but also explore the inter-job structural relationships based on DAG topology among various jobs in the cloud. Our main contributions are as follows.

1) We automate the construction of DAGs for more than 3 million batch jobs with dependencies from a production data center and transform these graphs into a vector representation for graph learning.
2) We perform graph learning and discover important patterns with regard to job-task-node dependency which provides crucial information for job scheduling.
3) We leverage graph clustering to group cloud jobs for workflow classification and scheduling.

The rest of the paper is organized as follows. Section 2 describes the background on cloud management. Section 3 presents the graph abstraction and data processing. Section 4 characterizes the structural patterns of batch cloud jobs and Section 5 details the graph-based clustering method and presents the analytical results. Section 6 concludes the paper with remarks on future work.

978-1-6654-3577-2/21/$31.00 ©2021 IEEE
DOI 10.1109/IPDPSW52791.2021.00052

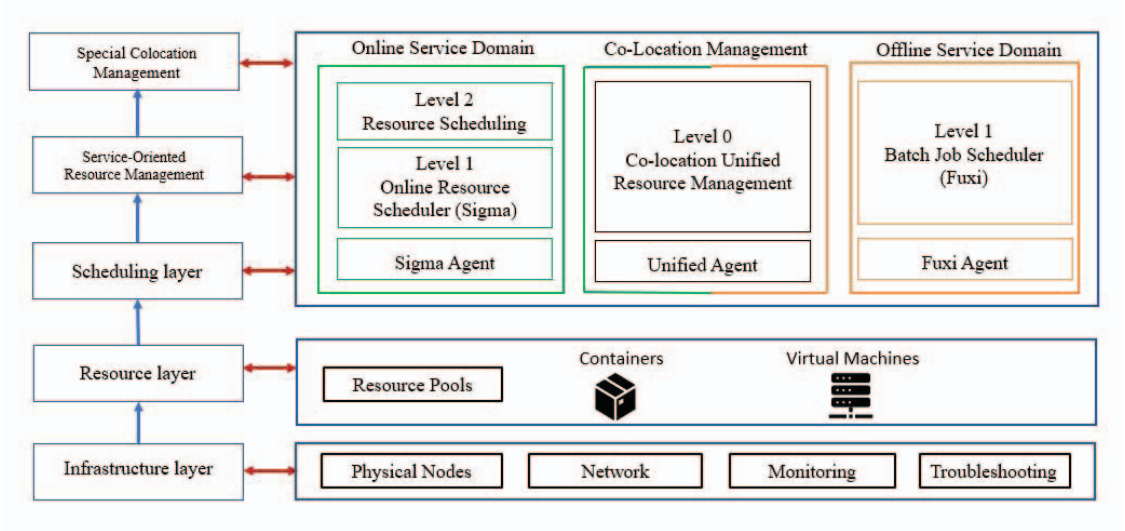

Fig. 1: System architecture. There are five layers in the cloud system. Above from the scheduling layer, there are there major components for cloud management: (1) unified controller. (2) cloud schedulers [42]. (3) upper level scheduling software, for example, Hippo Manager [43].

II. Background and Motivation

Cloud computing has become a mainstream investment in the modern business market. Enterprises adopt various state-of-the-art technologies to improve their competitiveness in a reliable economic-friendly manner and to better adapt to the rapidly changing business environments. The higher level of virtualization, automation, and security of cloud data centers provides larger capacity and greater convenience of management compared with conventional data centers. According to the prediction from Cisco [1], 94% of workloads and compute instances will be processed by cloud data centers in 2021.

The increasing scale of cloud services in data centers has led to the development of advanced systems and infrastructures. Meanwhile, the complexity is rising dramatically. Workloads in production cloud environments are heterogeneous and dynamic. A wide variety of applications are serviced by large-scale data centers. These applications include but not limited to live streaming services, Internet of things, transactions on e-commerce platforms, machine learning, data management and storage services. Due to miscellaneous capacity requirements and diverse performance characteristics of resources consolidation in data centers, the efficient management of these common shared infrastructures remains an important challenge. Cloud services providers, such as Alibaba cloud, Amazon Web Service (AWS), Google Cloud, IBM cloud and Microsoft Azure, are seeking solutions to improve their computing services for customers from different technical perspectives: infrastructures, platforms, and software. Cloud computing systems co-allocate online services and batch jobs to improve the server utilization as well as reduce the cost of energy and management. [2]

A. System Architecture

Before service co-location, different types of workloads ran on separate subsystems, which caused more financial cost and energy cost for data centers. The co-location architecture in cloud management systems is designed based on the needs of serving multiple different types of services, such as Online user-interactive services and offline computing services. Client-end job submissions such as transactions or searching requests are examples of online services. Online services require real-time responses with low latency and high performance. These jobs are usually hosted on containers. Offline services aim for executing large scale batch jobs which can support big data processing, computing services and statistical analysis. These server-end services are latency insensitive. Due to the characteristics of these two types of services, online jobs usually have a higher priority than offline services to ensure the quality of service (QoS) at run time. Co-location technology integrates the resource pool for both services in order to properly handle the resource allocation and competition in the scheduling layers. In a co-located cloud, resources are shared by online and offline jobs.

A hierarchical structure exists in the overall co-location architecture. Figure 1 presents a five-layer structure. In the infrastructure layer, servers, networking equipment, and storage resources needs to be well-planned and executed. On top of that, a centralized manager for resource pools is deployed at the resource layer. Furthermore, there are three levels of schedulers: Level-0 scheduler is responsible for allocating resources for both online and offline jobs, Level-1 schedulers are dedicated to managing online and offline services individually. Moreover, upper level scheduling software is deployed in the level-2 scheduler for specific resource needs.

B. Batch Workload

Batch jobs run without user interaction and involve job scheduling on multiple nodes and run tasks in parallel. Unlike online stream processing, a large volume of data needs to be processed for a long time and mostly follow diurnal patterns. In addition, the latency is insensitive for batch jobs. However, the lower priority of offline batch processing leads to uncertainty in execution. For example, when resource competition happens, the running batch jobs may be either suspended or killed to retain more computational resources for online streaming jobs. They are then rescheduled to run on other nodes.

Moreover, dependency is common in batch jobs. An increasing proportion of jobs with dependencies play an important role in the batch workloads. There are around 50% of batch jobs have dependencies in the production data center that we study and they consume 70% to 80% resources among all batch jobs.

Dependency of batch jobs exist among their tasks. Batch jobs with dependencies follow a job-task-instance hierarchical paradigm that each job consists of one or more tasks. A task is a computation unit of manifold distributed computing models such as MapReduce, Spark, and SQL. There is at least one instance for each task within a job. Resource requests can be similar among instances of a task, and their input data are usually distinct [2]. We use Directed Acyclic Graphs (DAGs) to present the dependency relationship among tasks in batch jobs. The completion time of a job is the total amount of time it spends from the earliest time of starting the first task(s) to the latest time of finishing the last task(s). Resulting from the increasing complexity of varying jobs in the batch workload, it becomes challenging to manage these long-running jobs with dependencies in multiplexing cloud environments.

III. Cloud and Workload Data

The data of Alibaba's cloud trace was released in 2018 and contains co-located jobs from about 4000 nodes over 8 days. It provides comprehensive data and information collected from machines, containers and batch jobs. Machines' meta and usage files were collected from servers. Configuration and runtime resource usage are included. Each server ran multiple containers that enabled distinct services to co-locate different type of jobs.

Our study focuses on analyzing batch cloud jobs. There are more than 4 million batch jobs run in the cloud during the 8-day period. Batch workloads were generated by internal users [12] to run jobs of MapReduce, SQL, and Flink ([6], [46], [45]), and machine learning. Batch jobs follow the job-task-instance hierarchical paradigm that each job includes one or more tasks and each task has multiple instances. Our work concentrates on job-level analysis of batch cloud workload. Task dependency in a job is represented by DAGs. We create a graph for each job and the vertices in the job graph represent tasks. In our traces, batch job data is stored in two files: ***batch_task*** and ***batch_instance***. Batch task data contains information at the task level, including task names under their lead jobs, instance number for each task, duration and planned resource usage. This dataset helps us determine the dependencies among various types of tasks and is used to build job DAGs for graph learning. Batch instance data provides the detail about instances' execution information of all tasks. It also contains temporal records for each instance and the actual resource consumption information (CPU and DRAM).

IV. Graph Representation and Data Processing

In this section, we create a graph to represent batch jobs from raw data and analyze the characteristics of DAG jobs with dependencies.

A. Batch Job DAGs Graph

DAG is an unique form to represent a batch job with dependencies. It provides an intuitive view of job representations and their interrelationships. In our study, most of the batch workloads constitute DAGs while others are independent. In the workload, task dependency is denoted in the field of Task_Name. For instance, the DAG of a job with job ID 1001388 in figure 8(a) consists of 5 tasks (M1, M3, R2, R4, R5) with some dependencies. In accordance with the dependency, each task named differently in the dataset. M1 and M3 are "Map" tasks that can start individually without waiting for other tasks to be finished. R2 and R4 are two "Reduce" tasks that are named with R2_1 and R4_3, which indicate that task 2 (R2) has dependency with task 1 (M1) and task 2 can only start running when task 1 is finished. Similar to task 4 (R4), it can only run after task 3 (M3) is completed. Moreover, the last task (R5) with name R5_4_3_2_1 denotes that task R5 can only start running after all previous tasks are accomplished.

Batch workloads graph representation enables us to perceive the scale of jobs that running in the cluster from the overall collection and it provides us with an intuitive view of topological structure based on tasks dependency. Moreover, it allows us to detect the relative distribution of jobs according to their volumes and concurrency degree.

We display the result in Figure 2. Each vertex represents a task that indicates stage of a running job. We applied direct edges to demonstrate the link dependency between stages. The dependency indicated that the succeeding task can only start running after its precedent task finished. We labeled nodes using the combination of their job and task name to distinguish tasks from different jobs. In addition, we take account the resource usage of CPU and memory and instances information including amounts, running time and periods as attributes to the running tasks.

B. Feature Vector Selection

In the subsequent experiments, we sample 100 DAG batch jobs arbitrarily from the overall data set to illustrate our method and display the result. To maintain the fairness of data

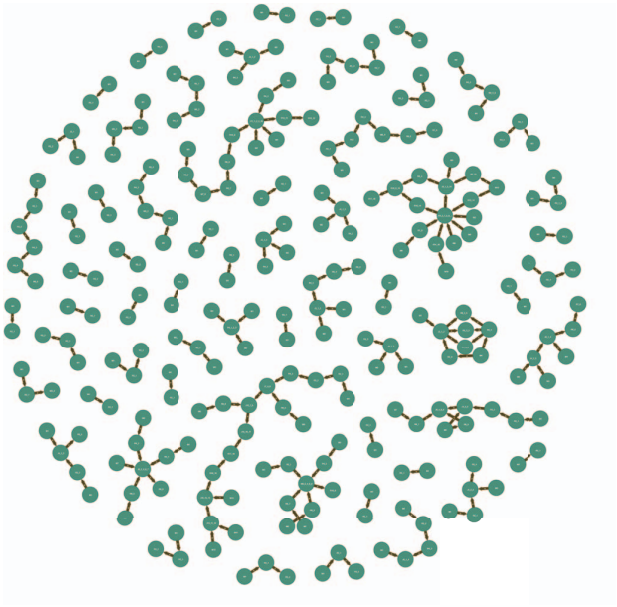

Fig. 2: Sample of job-level abstraction of DAG batch workload

sampling, we select our job entities based on the following criteria:

Integrity. Not all batch jobs in the workloads are end completely without any interruption. Some jobs still run during the process of data acquisition cut off. Besides, some jobs are canceled resulting from resource competition or erratic hardware issues. Thus, we are intention to filtrate the terminated jobs within the active interval of overall data to remain the job completeness.

Availability. Despite the fact that selected jobs with terminated status uphold the integrity of job graphs relatively, the evidence of existing running tasks collected before the starting point is missing. Original data source acquisition by administrators my happen while cluster servers provide normal services to the public. In this case, the actual running period of a job is no longer reliable which would affect the authenticity and effectiveness of the follow up analysis and the resultant model. We aims to keep the sampling data with effectual resource information in addition to the structural completeness of DAG jobs.

Variability. It is important to preserve the variability of DAG jobs by having manifold topological structures and sizes. Data-intensive jobs running in the cluster exhibit strong features on dependency. The dependency associated with tasks presents temporal and spatial information in the workloads. Additionally, all jobs vary in patterns on the basis of parallelism and temporal distribution. Overall, we have 17 different size types in our experimental set where the number of tasks ranging from 2 to 31 nodes.

C. Node Conflation

In the large-scale batch workload, jobs with smaller size are more likely to appear repetitively in terms of their simple topological structure. However, recurrent structure occurred partially in larger jobs. Some tasks perform the same kind of operations without sophisticated dependency to other nodes. Hence, we can consolidate them together to reduce the complexity of large jobs in the workload. In this way, we can improve the efficiency of estimating the DAG jobs's structure for further operations. As shown in Figure 3, the ratio of smaller jobs increases compared before doing merge operation.

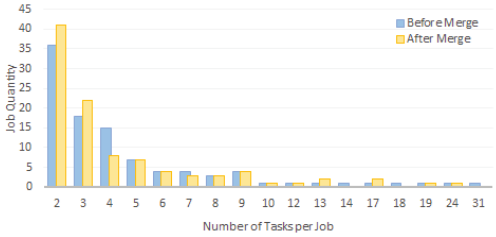

Fig. 3: Size of DAG jobs before and after node conflation

V. Characterization of Job Graphs

A. Structural Quantification

To better understand the characteristics of batch jobs, we measure our data by taking account the following features: 1) batch job size, 2) job critical path, and 3) job maximum width. The number of tasks each job has determines batch job size. In our experiment set, we obtain 17 different job volumes from raw data. The results are shown in Figure 4 and Figure 5. The amount of jobs in each size group are decreasing as the batch size increases. For example, the larger size jobs have appeared fewer times than the smaller size jobs.

Fig. 4: Job features before node conflation

The job critical path defined by the longest depth of a job DAG graph which also illustrate the level that each job has. We calculate the depth from each individual job and obtain the maximum one from distinct dimensional groups. The length of critical path varies from 2 to 8 in the experimental set. The

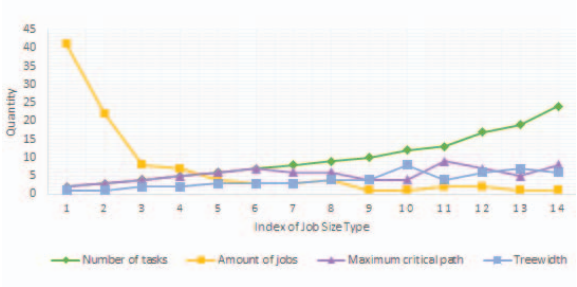

Fig. 5: Job features after node conflation

result shows that the maximum critical path does not increase linearly as the job size grows. In other words, some larger jobs have more tasks running in parallel. The overall length of maximum critical path of each size group are relatively small.

Furthermore, we compute the largest width of every single DAG graphs and select the maximum one from each size group. From our investigation, the parallelism of a job is quite positively correlated to the size of jobs. As shown in Figure 4, the extreme case occurs when there are 30 out of 31 tasks running in parallel and only one existing node perform the reduce operation. Even though jobs with larger size has higher parallelism, there is no guarantee that smaller jobs must have lower parallelism. For instance, the maximum width of a job from group 10, which consists of 12 nodes in the graph, is larger than the values of a job from group 14, which has a total 18 tasks.

B. Common Graph Patterns

In practice. many DAG batch jobs have complex dataflow intermixing with multi-hierarchical levels and various parallel tasks in the clusters. It made resource allocation and scheduling in a more efficient way become a challenge. Sometimes, the system needs to adopt a greedy approach to allot resources which could cause long waiting time and over provision issues on hardware. To better reveal the problems of batch jobs under co-located environment, we find several prevalent graphical patterns of DAG batch jobs. These components are fundamental structure to form a job. Also, some jobs are the synthesis of multiple components together. We categorize them into shape-based fundamental patterns: *inverted triangle*, *straight chain*, *diamond*.

Inverted Triangle: Jobs with inverted triangle structure are convergent from input vertices to the terminated node. Input vertices are nodes with in-degree of zero. Some of these tasks may running simultaneously while others may work separately. The number of inputs vertices are larger than the existing nodes. Most inverted triangle jobs end with a single node that perform a summary operation. A very easy example of this type of job is a simple MapReduce job. In the beginning, there are two map tasks and they finally merge to the reduce task. This type of job has second highest frequency (37%) appeared in the dataset.

Straight Chain: There are 58% of DAGs are straight chain jobs which means that all the tasks are running one after another. The number of inputs and output nodes are identical and restricted to one. Parallel tasks are not existing in this type of job. Tasks in the straight chain jobs need to wait to start after its parent tasks finish running. Every single task within the job is strongly reliant upon the performance of the previous task.

Diamond: Jobs in diamond topological structure have relatively low frequency than inverted triangle- and straight chain-type of jobs appeared in our data set. These jobs have same amount of input and output nodes such that they usually starting and ending with a single task. However, their intermediate level has multiple tasks running in parallel and the width is larger than the number of two edge nodes.

In addition, we investigate other pattern components and they also play an important role in the overall cluster workload. *hourglass* and *trapezium* style jobs have also been detected. They are basically the combination of formerly mentioned patterns or the alternatives with minor changes from fundamental components. The hourglass type of jobs have similar numbers of nodes in the beginning and ending stage but only have a few tasks running in the intermediate stage. The trapezium-type jobs have more ending tasks than input ones. There are also jobs with combination style such as having inverted triangle in the beginning, but following tasks are running sequentially with long tail. Learning job-based topological shape can help us understand the job structure according to their distribution and trend of how the tasks are arranged. This would further help with clustering and incoming job predictions.

C. Exploratory Investigation of Task Types

The subsequent question is how assigned tasks are organized in the DAG jobs? To answer this, we investigate the internal organization along with the structure of DAG jobs to explore the pattern of tasks. Large scale computing usually uses multi-level pipeline based parallel computing framework [4], for instance, Apache Hadoop [38]. Alibaba cloud developed their own platform for multi-tenancy data processing called MaxCompute [37]. It is also supporting computation jobs such as SQL, MapReduce, and Spark. We observe that there are some common batch programming modes has appeared in our data: **map-reduce** [6], **map-join-reduce** [7], and **map-reduce-merge** [8].

Map-Reduce is a programming model and best used for handling homogeneous data in the cloud environment. Most search-engine related jobs and some machine learning based applications use this model to process data with large volumes. Map function in the Map-Reduce framework splits input data into smaller blocks and then processes and releases them to an intermediate phase called shuffle, which could operate on different machines in the cluster. The framework shuffles and sorts the results to the reducer in the reduce phase. Additionally, Map-Join-Reduce is an extended version of MapReduce that improves the runtime efficiency on processing

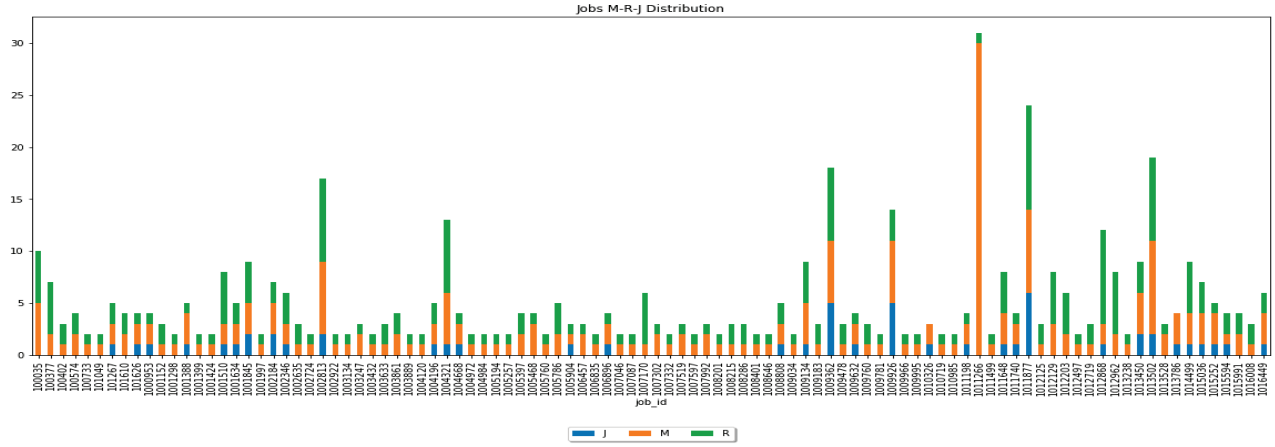

Fig. 6: Distribution of Map-Join-Reduce tasks

heterogeneous data analytic. It is designed for multi-way join and allows to partition multiple data sets to the reducers in one pass [7]. Map-Reduce-Merge [8] adds a Merge phase after applying the map and reduce function and it reads data in an organized manner. This programming model can also support join operation on multiple dataset and it is helpful for processing relational data.

To understand the relationship between task type and the structure of DAG jobs, we measure the amount of different task types among each job. As shown in figure 6. There are three types of tasks appearing in our data: M, J, or R. There might be other types of tasks in the original dataset. However, those tasks are either jobs without DAG structure or they are not fully terminated jobs. So we did not conclude them in our test. M represents the tasks are either Map or Merge. R indicates that the reduce function is applied in the task. J is the join operations. The original cluster trace does not provide the detailed description on specific task type, unlike trace data from other organizations, like Google [10]. Regardless, these jobs follow some common patterns that we can infer their types based on the framework infrastructure. Many smaller sized jobs that their length of critical path is less or equal to two which exploit fundamental Map-Reduce framework that only map and reduce functions deployed. Note that the job id with 1011266 has the largest number of M tasks and just a single R task.

Other than that, the majority of jobs present Map-Reduce with join operations. Interestingly, there are different kinds of Joins in the jobs. To distinguish the joins in different algorithms, we find out that "join" in the general MapReudce can be performed in either map or reduce side. The join occurs before reaching the map function is the map-side join. The shuffle phase sorts and merges the intermediate file from mapper to reducer. It's a one-to-one strategy that the data in the intermediate node can only pass data to another node [9]. Basically, if the join operations are performed in the map or reduce phases are implementing the general MapReduce. However, "join" in Map-Join-Reduce is an independent stage that introduces a filtering-join-aggregation programming model which allows the one-to-many shuffling activated [9]. The reduce functions read the input from the output of the last join function [7].

In our observation, nearly all chain-structured jobs performed MapReduce without join operations. The amount of R tasks in most of the chain-structure cases are deployed more than M tasks except those jobs who have less than four task nodes in job graphs. In relatively larger jobs, the form of task deployed is even more complicated. Map-Reduce and Map-Join-Reduce framework are used in combination. We also find that jobs with higher parallelism have more reduced tasks after the node conflation process.

Performing clustering analysis is a critical step for exploring the topological features of batch jobs DAG. Before applying classification techniques, a similarity measurement is needed for graph-structured data. A conventional idea is to calculate the edit distance of transforming one graph to another. However, the computational cost is exponential depending on the number of nodes, which is less effective. We adopt graph kernels, in particular Weisfeiler-Lehman kernel [18] in our method to conduct the graph clustering tasks for the batch jobs.

D. The Kernel-Based Approach

The Weisfeiler-Lehman (WL) graph kernels break down a connected graph into subtrees. and then, a similarity function is defined according to the number of common substructures across pairs of graphs. The common patterns are also discovered in our batch job DAGs as we introduced in section 5.

A graph kernel is a kernel function over a set of graphs. It is similar to an inner product of the embedding. Weisfeiler-Lehman can effectively compare if two graphs are isomorphic. It builds on top of the substructure-based kernel such as subtree kernel or shortest path kernel. The Weisfeiler-Lehman is defined as follows:

Let G and G' be graphs for comparison. Assume G^1, G^2, ..., G^n are graphs emerging from G at the iteration 1, 2, ..., n of the Weisfeiler-Lehman algorithm. We define the Weisfeiler-Lehman kernel as follows:

$$k_{wl}^{n}(G, G') = \sum_{i,j=0}^{n} k(G^n, G'^n) \tag{1}$$

In equation (1), k represents a base kernel function, such as subtree or shortest path kernel. The base kernels, k(G, G'), compare the substructure of two graphs that usually takes polynomial time. We define that Σ represents the set of labels of graph G and G'. The set of original labels of both graphs is represented by Σ_0, whereas, Σ_n is the set of labels after n^{th} iteration of the Weisfeiler-Lehman algorithm. Assume that all Σ_n are pairwise disjoint and all elements (σ_{n1}, ..., $\sigma_{i|\Sigma_n|}$) inside Σ_n is ordered. The subtree kernel on graph G and G' with n iterations is defined as:

$$k_{wlsubtree}^{(n)}(G, G') = \langle \phi_{wlsubtree}^{(n)}(G), \phi_{wlsubtree}^{(n)}(G') \rangle \tag{2}$$

where $\phi_{wlsubtree}^{(n)}(G)$ and $\phi_{wlsubtree}^{(n)}(G')$ are sets of maps that count the number of occurrences of the label (σ_{ij}) in the graph G and G' respectively, such that

$$\phi_{wlsubtree}^{(n)}(G) = (m_0(G, \sigma_{01}), ..., m_0(G, \sigma_{0|\Sigma_0|}), \\ ..., (m_n(G, \sigma_{n1}), ..., m_n(G, \sigma_{n|\Sigma_n|}))$$

and

$$\phi_{wlsubtree}^{(n)}(G') = (m_0(G', \sigma_{01}), ..., m_0(G', \sigma_{0|\Sigma_0|}), \\ ..., (m_n(G', \sigma_{n1}), ..., m_n(G', \sigma_{n|\Sigma_n|}))$$

Graph kernels employ re-labeling technique to the graph in each iteration and obtain the new kernel values. Then, the current nodes in both graphs will be re-labeled based on the new values. When iterations end, if vertices in two graphs get the same labels, the two graphs are isomorphic. If they are not completely same, similarity scores are calculated. We have shown the correlation map of our experimental data based on the similarity score in Fig 7. The x- and y-axis of the map are index values from 100 job samples randomly selected from the dataset for demonstration purpose. Values of similarity scores are float numbers ranging from 0 (darker blue) to +1 (red), where 1 indicates that two job graphs are identical in terms of the topological structure. In other words, the smaller the value, the less similar the two graphs are. We ignore the results of the red solid block along the diagonal since they are self-comparison of the same job graphs. We found out that smaller graphs with short tails and low-level parallelism usually have higher similarity scores. For larger graphs, parallelism, tail length, and degree of nodes in subgraphs among jobs are comprehensively examined based on the calculation using proposed method. We then use them for clustering to have a deeper insight into the implication of outcomes.

VI. Graph Similarity and Classification Analysis

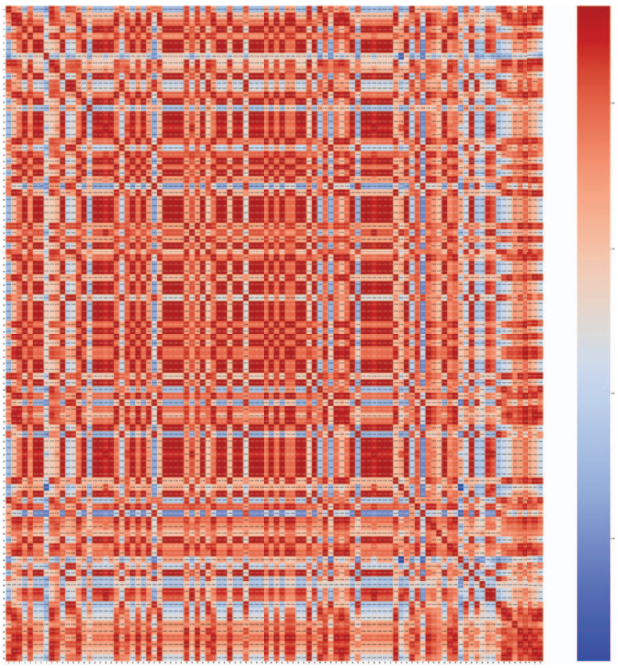

Fig. 7: Similarity score map formed by pairwise comparison between batch job DAGs.

A. Job Graph Clustering

In order to discover the representative pattern from existing DAG jobs, we perform an exploratively analysis by applying unsupervised learning methods on the received similarity map to cluster jobs with multiple topological characteristics into groups. In particular, we explore spectral clustering method based on the generated jobs correlation map. Spectral clustering aims to classify items into clusters through the eigen decomposition of a similarity matrix. Due to the nature of our circumstance, we implement spectral clustering because it can capture the characteristics of geometrics from graph-based data and can apply affinity measures directly from the formatted input matrix. In a typical spectral clustering algorithm, it needs to construct a similarity graph defined by each pair of data points as input and it will then calculate the clusters based on the affinity matrix instead of defining specific attributes to train the model for clusters.

Conventional spectral clustering algorithm usually performs on the task-level (or node-level in graph theory). In our experiment, we apply the spectral clustering algorithm to a pairwise graph comparison of sampling jobs and discover the clusters based on the similarity scores between each pair of jobs. In the experimental results, job graphs are clustered into five groups in terms of DAG topological structure. We display statistical analysis in Fig 9 according to the clustering results. Various job sizes based on the number of tasks appear in

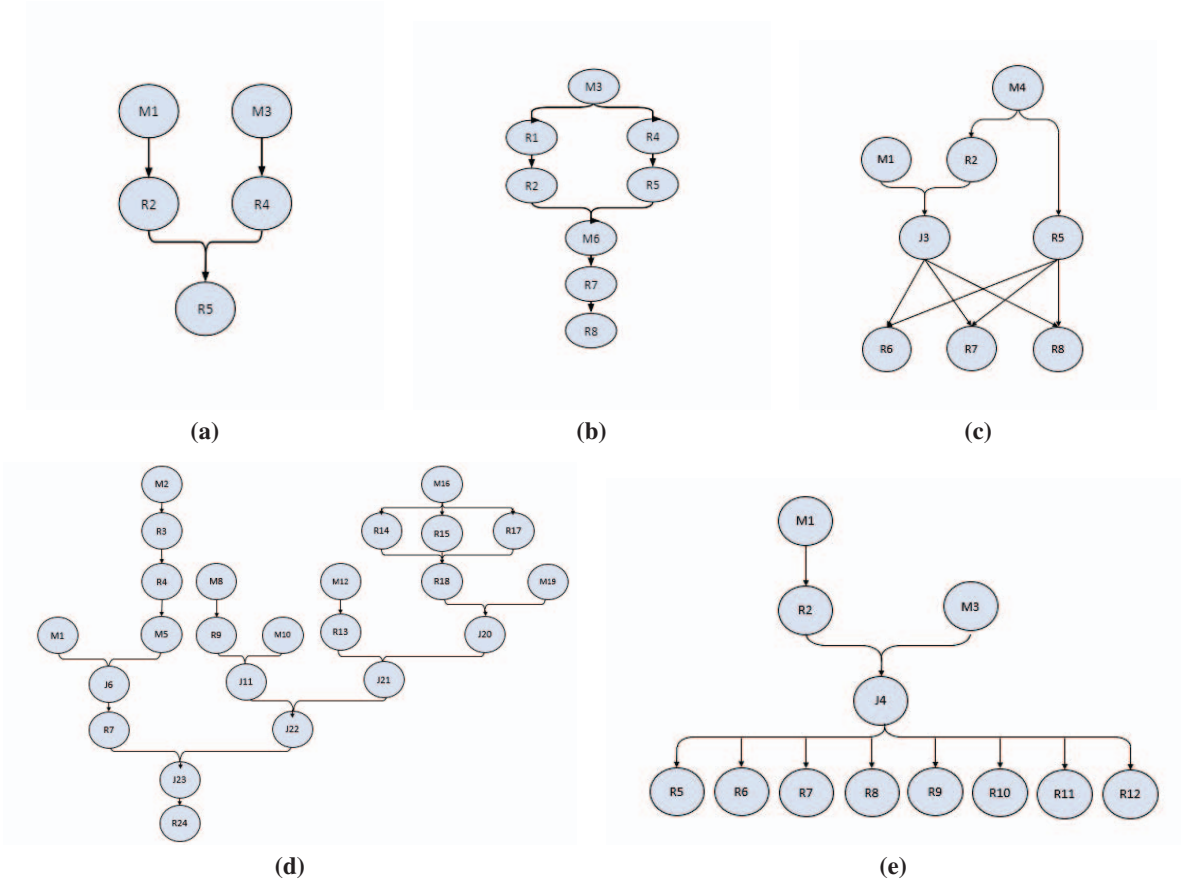

Fig. 8: **Clustering Groups.** Five different groups have been detected. Some representative jobs from each group are selected to show above. **(a)** Group A with job id 1001496 **(b)** Group B with job id 1012962 **(c)** Group C with job id 1001510 **(d)** Group D with job id 1011877 **(e)** Group E with job id 1012867

each cluster group which is shown in Figure 9(a). 75% of jobs are clustered in the group A. The majority (90.6%) of jobs inside group A are short jobs which have less than three nodes (tasks) in each job DAG. Their length of critical path and parallelism are smaller compared to other groups. Small chain-structure jobs are frequently seen (91%) in this group. This is because smaller jobs have relatively simple structure. In addition, the intermediate parallel running tasks of non-chained jobs in group A are more likely to converge into one ending node in the last step. The topological shape of jobs in Group A involves inverted triangle, straight chain, and diamonds.

The scale of jobs in the clustered group B increases as the average size raises about approximately 1.55 times in respect of group A. The overall extent of critical path and parallelism of jobs enhances as well. It appears that more jobs contain chain-structured subgraphs closely tied to its distributed tasks within the graph. These jobs still follow the style of convergence but with longer tails in their hybrid structure.

Furthermore, due to the increasing complexity, jobs in group D have higher average values among the metrics. Subgraph after distributed tasks in each job not only has chains but also successive serial tasks appear before the aggregation. Group C and Group E have a few different properties than other groups. Unlike jobs in the other clusters follow a convergence structure, Jobs in group C and E are diffuse. Based on the depth of graphs, the number of tasks in the last level are more than its precedent level. In group E, the extreme case, 8 tasks are released from a single node at the ending level. For the jobs in Group C, task intersection exists, for instance, all the ending nodes are fully connected to every node in their previous level.

VII. Related Work

Workload analysis and resource management. Cluster workload analysis can provide valuable insight to the system performance and utilization. [25] characterized the resource usage of both online and offline production jobs from Alibaba's cluster data of 2017. [26] also analyzed the same sets of data but they focused on discovering the characteristic

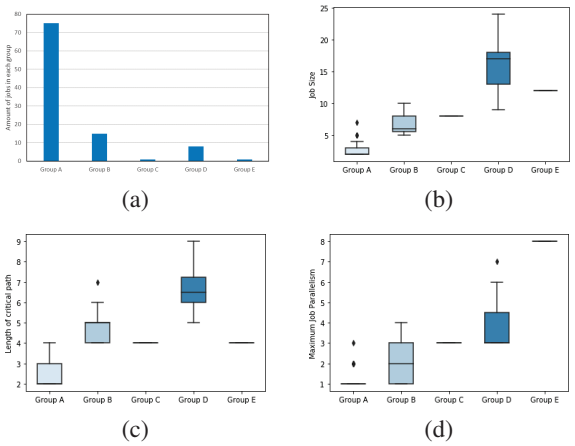

Fig. 9: Properties of job DAGs in cluster groups. (a) Population of jobs in groups. (b) Distribution of job size across groups. (c) Distribution of the length of the longest critical path of jobs across groups. (d) Distribution of the maximum parallelism among jobs across groups.

of imbalance from overall workload in terms of resource demands and job duration. [14] provided statistical analysis on workload behavior and applied k-Means clustering algorithm to grouping the similar job with similar properties on resource demands and duration. [12] provided an overall analysis on newly released data from Alibaba. [13] performed a character analysis on DAG jobs and innovated a synthesis method to generate simulated workloads. In addition, [34] introduced a fractal-based model to investigate and characterize the self-similarity and non-stationarity property of cloud workloads for optimal control.

Algorithm design of dependency-aware scheduling in distributed clusters is another prevalent topic to improve efficiency of data center. [16] developed a tool, called Decima, to automatically learn the scheduling policies in the cluster using reinforcement learning technique. Another group [15] implemented the deep reinforcement learning approach to optimize the scheduling of dependency-based jobs. However, their job only focused on the task level optimization. [27]applied a task duplication strategy to improve the performance of solving multi-clusters DAG mapping problem.

Graph-based learning. Moreover, the literature of graph-based learning provides a variety of intriguing approaches and applications. In terms of graph similarity analysis, [20] proposed a new graph matching networks model based on graph neural networks. It can effectively learn the similarity reasoning and identify differences among graphs by computing similarity scores through cross-graph attention mechanisms. SimGNN [19] is a neural network-based approach to solve the graph similarity search problems by combining embedding function with attention mechanism and pairwise node comparison in graph-level embedding. On top of that, instead of using the fixed-dimensional embedding in graph-level representations, node embedding matching technique was applied to obtain the fine-grained differences between graphs for similarity computation. [21]

More specifically, the following works adopted the Weisfeiler-Lehman algorithm [18] in their design to improve the performance of graph learning. [22] developed a graph neural network-based framework in accordance with transform-sum-concatenation pattern which considered the continuous similarity in the neighborhood aggregation. [23] presented two algorithms for both labeled and unlabeled graphs aiming to capture the global properties of graph comparison. The first algorithm employed indefinite kernels based on SVM classification and the second algorithm consolidated Pyramid Match Kernel and Weisfeiler-Lehman framework to enhance the accuracy of graph classification. In addition, [24] proposed a graph kernel method on the basis of topological properties that leverages iterative variants of persistent Weisfeiler-Lehman procedure to improve model generalizability and predictive performance by capturing the features of connected components and cycles in non-attributed graphs.

Furthermore, a wide range of applications of graph learning have been explored in real-world scenarios. [28] and [29] applied graph similarity analysis in storage workload and cloud migration pattern generation, respectively. [30] developed a graph-kernel based structure feature selection method to classify connectivity networks for brain disease. Similarly, [31] applied WL-align technique to evaluate the brain atlases based on topological structure similarity in connectome. Besides, [32] and [33] also adopted the principles from WL kernel to detect program resemblance and improve clustering outcome of source code, correspondingly.

VIII. Conclusions

In this work, we perform a job-level graph-based batch workload analysis with an aim towards understanding the hidden information of job topological structure in the large-scale cluster. We execute a comparative character analysis on dependable jobs to study the performance in terms of common patterns and task type. Finally, we contribute a graph similarity approach that learns from the sub-patterns of each job and clustering them into multiple groups. The result shows that our method bridges the gap that can effectively capture the properties of topological structure of batch jobs in a co-located cloud environment. In the future, we plan to extend the analysis by combining resource analysis techniques for job scheduling optimization.

Acknowledgment

This work has been supported in part by the National Science Foundation grants CCF-1563750, CNS-1828105, CNS-1852134, OAC-2017564, and CNS-2037982. We thank the anonymous reviewers for their constructive comments, which helped us improve this paper.

REFERENCES

[1] Global Cloud Index Projects Cloud Traffic to Represent 95 Percent of Total Data Center Traffic by 2021 https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1908858

[2] C. Jiang, G. Han, J. Lin, G. Jia, W. Shi and J. Wan, "Characteristics of Co-Allocated Online Services and Batch Jobs in Internet Data Centers: A Case Study From Alibaba Cloud," in IEEE Access, vol. 7, pp. 22495-22508, 2019.

[3] Malte Schwarzkopf. 2017. Cluster Scheduling for Data Centers: Expert-curated Guides to the Best of CS Research: Distributed Cluster Scheduling. Queue 15, 5, pp. 78–89, 2017.

[4] Y. Cheng, A. Anwar and X. Duan, "Analyzing Alibaba's Co-located Datacenter Workloads," IEEE International Conference on Big Data (Big Data), 2018.

[5] https://github.com/alibaba/clusterdata

[6] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1, pp. 107–113, 2008.

[7] D. Jiang, A. K. H. Tung and G. Chen, "MAP-JOIN-REDUCE: Toward Scalable and Efficient Data Analysis on Large Clusters," IEEE Transactions on Knowledge and Data Engineering, vol. 23, no. 9, pp. 1299-1311, 2011.

[8] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D. Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (SIGMOD). 2007.

[9] J. Ren, L. Liu, F. Liu, W. Zhou and S. Lü, "An Executable Specification of Map-Join-Reduce Using Haskell," in IEEE Access, vol. 7, pp. 10892-10904, 2019.

[10] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In Proceedings of the Third ACM Symposium on Cloud Computing (SoCC). 2012.

[11] C. Lu, K. Ye, G. Xu, C. Xu and T. Bai, "Imbalance in the cloud: An analysis on Alibaba cluster trace," 2017 IEEE International Conference on Big Data (Big Data), 2017.

[12] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. 2019. Who limits the resource efficiency of my datacenter: an analysis of Alibaba datacenter traces. In International Symposium on Quality of Service (IWQoS), 2019.

[13] Huangshi Tian, Yunchuan Zheng, and Wei Wang. 2019. Characterizing and Synthesizing Task Dependencies of Data-Parallel Jobs in Alibaba Cloud. In ACM Symposium on Cloud Computing (SoCC '19). 2019.

[14] W. Chen, K. Ye, Y. Wang, G. Xu and C. Xu, "How Does the Workload Look Like in Production Cloud? Analysis and Clustering of Workloads on Alibaba Cluster Trace," in IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2018.

[15] Z. Hu, J. Tu and B. Li, "Spear: Optimized Dependency-Aware Task Scheduling with Deep Reinforcement Learning," in IEEE International Conference on Distributed Computing Systems (ICDCS), 2019.

[16] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In ACM Special Interest Group on Data Communication (SIGCOMM), 2019.

[17] Z. Hu, D. Li, Y. Zhang, D. Guo and Z. Li, "Branch Scheduling: DAG-Aware Scheduling for Speeding up Data-Parallel Jobs," in IEEE International Symposium on Quality of Service (IWQoS), 2019.

[18] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt, Weisfeiler-Lehman Graph Kernels. Journal of Machine Learning Research, 12(77), pp. 2539-2561, 2011.

[19] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. 2019. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. In Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining (WSDM '19). Association for Computing Machinery, 2019.

[20] Li, Y., Gu, C., Dullien, T., Vinyals, O. amp; Kohli, P.. (2019). "Graph Matching Networks for Learning the Similarity of Graph Structured Objects." Proceedings of the 36th International Conference on Machine Learning, in Proceedings of Machine Learning Research.

[21] Bai, Y., Ding, H., Gu, K., Sun, Y., Wang, W. "Learning-Based Efficient Graph Similarity Computation via Multi-Scale Convolutional Set Matching". Proceedings of the AAAI Conference on Artificial Intelligence, 2020.

[22] Ok, Seongmin. "A Graph Similarity for Deep Learning." In Advances in Neural Information Processing Systems 2020 (NeurIPS 2020).

[23] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In Proceedings of International Conference on Artificial Intelligence (AAAI), 2017.

[24] Rieck B, Bock C, Borgwardt K. A persistent weisfeiler-lehman procedure for graph classification. In International Conference on Machine Learning, 2019.

[25] Cheng, Yue, Ali Anwar, and Xuejing Duan. "Analyzing alibaba's co-located datacenter workloads." In 2018 IEEE International Conference on Big Data (Big Data), 2018.

[26] C. Lu, K. Ye, G. Xu, C. Xu and T. Bai, "Imbalance in the cloud: An analysis on Alibaba cluster trace," 2017 IEEE International Conference on Big Data (Big Data), 2017.

[27] Chaudhuri, P., Elcock, J. (2010). Scheduling DAG-based applications in multicluster environments with background workload using task duplication. International Journal of Computer Mathematics, 87(11), 2387–2397. 4

[28] Y. Zhou, L. Liu, S. Seshadri and L. Chiu, "Analyzing Enterprise Storage Workloads With Graph Modeling and Clustering," in IEEE Journal on Selected Areas in Communications, vol. 34, no. 3, pp. 551-574, 2016.

[29] Z. Wan, F. J. Meng, J. M. Xu and P. Wang, "Service Composition Pattern Generation for Cloud Migration: A Graph Similarity Analysis Approach," 2014 IEEE International Conference on Web Services, 2014.

[30] M. Wang et al., "Graph-Kernel Based Structured Feature Selection for Brain Disease Classification Using Functional Connectivity Networks," in IEEE Access, vol. 7, pp. 35001-35011, 2019.

[31] Matteo Frigo, Emilio Cruciani, David Coudert, Rachid Deriche, Emanuele Natale, Samuel Deslauriers-Gauthier, "Network alignment and similarity reveal atlas-based topological differences in structural connectomes", bioRxiv 2020.12.16.422501;

[32] Wenchao Li, Hassen Saidi, Huascar Sanchez, Martin Schäf, and Pascal Schweitzer. 2016. Detecting Similar Programs via The Weisfeiler-Leman Graph Kernel. In Proceedings of the 15th International Conference on Software Reuse: Bridging with Social-Awareness - Volume 9679 (ICSR 2016). Springer-Verlag, Berlin, Heidelberg, 315–330.

[33] Höppner F., Jahnke M. (2020) Enriched Weisfeiler-Lehman Kernel for Improved Graph Clustering of Source Code. In: Berthold M., Feelders A., Krempl G. (eds) Advances in Intelligent Data Analysis XVIII. IDA 2020. Lecture Notes in Computer Science, vol 12080. Springer, Cham.

[34] Chen, M. Ghorbani, Y. Wang, P. Bogdan and M. Pedram, "Trace-Based Analysis and Prediction of Cloud Computing User Behavior Using the Fractal Modeling Technique," 2014 IEEE International Congress on Big Data, Anchorage, AK, USA, 2014, pp. 733-739, doi: 10.1109/BigData.Congress.2014.108.

[35] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: analysis and an algorithm. In Proceedings of International Conference on Neural Information Processing Systems (NIPS). 2001.

[36] Google Omega: Schwarzkopf, Malte, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. "Omega: flexible, scalable schedulers for large compute clusters." In Proceedings of the 8th ACM European Conference on Computer Systems, pp. 351-364. 2013.

[37] Alibaba Maxcompute data processing platform: https://www.alibabacloud.com/product/maxcompute

[38] Apache Hadoop Framework: https://hadoop.apache.org/

[39] Yarn: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[40] Mesos: http://mesos.apache.org/

[41] Docker Swarm: https://docs.docker.com/engine/swarm/swarm-tutorial/

[42] Alibaba Cloud: https://us.alibabacloud.com/

[43] Hippo Manager: https://www.hippomanager.com/

[44] Apollo: https://www.apollographql.com/docs/

[45] Flink: https://flink.apache.org/

[46] Spark SQL: https://spark.apache.org/sql/

[47] Apache Spark: https://spark.apache.org/