I/O Characteristics Discovery in Cloud Storage Systems

Jiang Zhou, Dong Dai, Yu Mao, Xin Chen, Yu Zhuang, and Yong Chen
Department of Computer Science, Texas Tech University, USA

***Abstract*—The data growth from many applications in clouds poses significant challenges to cloud storage systems. To deliver the best storage and I/O performance possible, it is often required to understand and leverage the I/O characteristics based on data accesses. A number of research studies have been carried out on this topic. However, most of them either utilize a limited number of data-access attributes, restricting the general applicability of the method for different applications, or heavily rely on the domain knowledge or expertise about applications' I/O behaviors to select the best representative features, introducing bias for certain workloads. To overcome these limitations, in this study, we present a new I/O characteristic discovery methodology. This method enables capturing data-access features as many as possible to eliminate human bias. It utilizes a machine-learning based strategy to derive the most important set of features automatically, and groups data objects with a clustering algorithm (DBSCAN) to reveal I/O characteristics discovered. These I/O characteristics revealed can direct I/O performance optimizations in numerous scenarios, such as in data prefeteching and data reorganization optimizations in cloud storage systems.**

***Keywords*-Cloud storage systems, file systems, I/O characteristics discovery**

I. Introduction

The ever-increasing data demand in many science and engineering domains and applications has posed significant challenges to cloud computing systems. To achieve highly optimized data-access performance of cloud storage systems understanding and leveraging data-access patterns have been proven effective in I/O optimization [1].

Arguably, the better the access pattern is understood, the better the storage and cloud system can be tuned. As a result, numerous studies have been conducted to identify, characterize, and leverage data-access patterns. There are two typical methods to analyze and discover I/O characteristics. One is to analyze spatial/temporal behaviors to identify I/O access sequence [2–6]. The other method is to analyze semantic information such as the access correlations between blocks/objects by mining I/O semantic attributes, which can potentially discover more complex patterns, especially semantic patterns [7–11]. While existing studies show the feasibility of I/O characteristics discovery through various approaches, they have two shortcomings as discussed below, which also motivates this research.

First, many of these approaches are limited to specific and predefined features. A feature refers to an attribute of a data access. For example, the object ID of an object access is a feature; the data access size is another feature. Many existing studies investigate one or more rather specific, predefined features to analyze access patterns. The resulting insights, although valuable, often lead to an incomplete view of access patterns. For example, *object ID* and *access time* help obtain temporal I/O behavior, but, if *access length* and *offset* are considered, spatial correlation of data accesses can be further derived. In general, more features indicate more I/O behaviors, i.e., read/write *operation code* reveals read/write types and *target node ID* provides object location. Given the increasing complexity of I/O behaviors, it is inadequate to attempt to discover I/O characteristics with only specific or predefined features. It is critical to analyze abundant features thoroughly for I/O characteristics discovery.

Second, existing approaches often introduce bias. Clearly, treating all features equally is not accurate because distinct features can have significantly different impacts on I/O characterization. However, selecting the desired set of features is often daunting and introduces bias, which requires domain knowledge and assumptions about storage systems and applications. Besides, we will not know whether we have a good set of features until we have completed the entire analysis process. For example, in study [7], the authors presented an iterative process to initially select some basic features, e.g., total I/O size and read-write ratio for a file, and then add new features if the analysis results leave some system design choice ambiguous. They still need to interpret the output results and derive access patterns by looking at only the relevant subset of features, again using domain knowledge. Moreover, the system may exhibit various data-access patterns for a given application. For example, the scientific computing applications in study [12] show various data-access patterns for massive data processing. Identifying representative features manually would not adapt to this scenario, and may lead to an untenable analysis. This largely limits the efficiency of using access patterns for tuning the system and optimizing the performance.

In this paper, we present a new design methodology to overcome these shortcomings of existing methods discussed above. Specifically, we propose to capture features of data accesses as many as possible (more than 20 in our test cases), including features like *object ID*, *access time*, *target node*, and others we can collect, to generalize data-access pattern discovery for various applications. Based on the rich set of features, we use access correlations among objects to identify different patterns. We utilize machine-learning based strategy (principal component analysis [13]) to find the most important "key features" automatically among all collected features in an unsupervised way. This eliminates the bias from users or domain knowledge requires for applications, and provides an automatic, extensible way to identify the dominant data-access patterns. Based on the learned key features, we apply a clustering algorithm (i.e., DBSCAN) to mine the objects' I/O similarity, particularly "key feature correlations", and group highly relevant object IDs, which can be leveraged to improve the I/O performance.

The key contributions of this study are three-fold:

- Propose a new I/O characteristics and data-access pattern discovery strategy based on a large number of collectable features of I/O accesses.
- Utilize machine-learning based strategy to identify key features and to cluster objects into highly correlated groups for I/O optimizations.
- Conduct various experimental evaluations to validate the proposed I/O characterization methodology.

II. Related Work

A number of tools have been developed to profile and trace I/O activities, such as Darshan [14], LANL-Trace [15], RIOT I/O [16], etc. Existing tools record I/O behaviors for user applications. However, most of them focus on the collection of I/O statistical information without providing effective ways to understand data-access patterns.

There is a rich set of literature on the topic of I/O features analysis and pattern discovery. These approaches mainly focus on two categories: I/O access sequence analysis and I/O semantic attribute analysis. The I/O sequence analysis is based on various parameters of data accesses, including spatial locality, temporal sequence, and repeating operations [2–6]. However, these analyses were performed to look for the periodicity of an application's I/O behavior based on prior workload expectations. It lacks of consideration on analyzing the I/O behaviors that have no knowledge to assume any application to possesses certain patterns.

On the other hand, by extracting semantic attributes from file systems, semantic attribute mining approaches can analyze more complex I/O patterns and get the correlations among data accesses, such as C-Miner [9], Farmer [8] Block2Vec [17], and many others [7, 10, 11]. Although these methods look at trace for I/O characteristics discovery, they perform the pattern analysis with only one or few specific features at a time. Chen et al. [7] proposed a multi-dimensional, statistical correlation trace analysis with K-means data clustering algorithm to identify access patterns. It can obtain comprehensive data access behavior, but require domain knowledge for selecting the set of descriptive and interpret the output results. Different from them, we introduces the *principal component* concept to automatically select key features from a vast number of access features. It reduces the bias introduced by domain knowledge or priori information of the applications. In addition, we also utilize DBSCAN, a multi-dimensional statistical data clustering algorithm to analyze I/O similarity and dynamically adjust the distance threshold for clustering. It can identify groups of highly similar I/O accesses without any assumption the number or shape of result clusters and achieve I/O characterization.

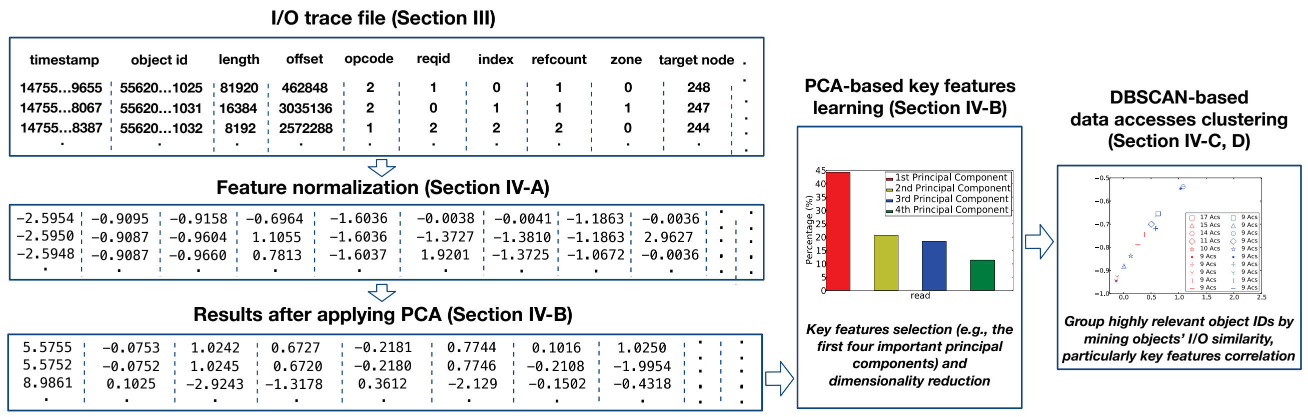
With the analysis on data-access patterns, the storage sources can be better leveraged to boost the performance of applications. It has motivated various I/O optimizations including prefetching [6], data layout [8], and scheduling techniques [18]. Model-based algorithms, such as using neural network [3], Markov models [19], grammar-based model [5] and so on [20], have been studied and proven their efficiency for prefetching in many cases. However, existing preftching strategies mainly focus on spatial/temporal I/O behaviors or specific access features to prefetch future data and achieve performance optimization. On contrast, We address the limitations of current prediction systems for data accesses with high I/O similarity. We use PCA-based method and data clustering algorithm to analyze key feature correlations among objects from I/O behavior. These correlations and characterizations discovered can be used to optimize systems.

III. I/O Trace in Storage Systems

In most storage models, applications access data files through the POSIX interface. Under such model, three I/O software stack layers can be identified [14, 21]. The top layer runs on compute nodes (i.e., clients) within user applications, where the I/O accesses are issued from. The second layer includes both the client-side file system libraries and runtime supports. The storage system needs to implement the functions of file read and write operations to support the POSIX interface. Data accesses will be mapped to storage objects in this layer. At the third layer, these mapped object requests will be dispatched to different storage nodes for accessing data.

Data accesses can be traced at any I/O stack layer. For example, Darshan [22] collects I/O trace statistics at the first layer via instrumenting I/O calls made by applications. In this study, we focus on tracing accesses at the third layer, i.e., on the storage nodes (server-side). There are two reasons for this design decision. First, all first- and second-layer I/O behaviors will be ultimately turned into accesses at the third layer. I/O accesses can be fully collected through tracing on the server side. Second, tracing server-side accesses does not assume any domain knowledge or priori information of applications, which is critical to support a system-level I/O characteristics discovery methodology.

We focus on a distributed object-based storage system, Sheepdog, which follows a decentralized design. I/O requests are directly sent to the storage nodes (e.g., via a consistent hashing algorithm [23]). Specifically, we trace data accesses in Sheepdog via *gateway nodes* that receive or forward all I/O requests, and use them to design and


I/O trace file (Section III)
Feature normalization (Section IV-A)
Results after applying PCA (Section IV-B)
PCA-based key features learning (Section IV-B)
1st Principal Component
2nd Principal Component
3rd Principal Component
4th Principal Component
Percentage (%)
read
Key features selection (e.g., the first four important principal components) and dimensionality reduction
DBSCAN-based data accesses clustering (Section IV-C, D)
Group highly relevant object IDs by mining objects' I/O similarity, particularly key features correlation

timestamp	object id	length	offset	opcode	reqid	index	refcount	zone	target node	.
14755...9655	55620...1025	81920	462848	2	1	0	1	0	248	.
14755...8067	55620...1031	16384	3035136	2	0	1	1	1	247	.
14755...8387	55620...1032	8192	2572288	1	2	2	2	0	244	.
.	.	.	.	.	.	.	.	.	.	.

-2.5954	-0.9095	-0.9158	-0.6964	-1.6036	-0.0038	-0.0041	-1.1863	-0.0036	.	.
-2.5950	-0.9087	-0.9604	1.1055	-1.6036	-1.3727	-1.3810	-1.1863	2.9627	.	.
-2.5948	-0.9087	-0.9660	0.7813	-1.6037	1.9201	-1.3725	-1.0672	-0.0036	.	.
.	.	.	.	.	.	.	.	.	.	.

5.5755	-0.0753	1.0242	0.6727	-0.2181	0.7744	0.1016	1.0250	.	.
5.5752	-0.0752	1.0245	0.6720	-0.2180	0.7746	-0.2108	-1.9954	.	.
8.9861	0.1025	-2.9243	-1.3178	0.3612	-2.129	-0.1502	-0.4318	.	.
.	.	.	.	.	.	.	.	.	.

Figure 1: Overview of I/O characteristics discovery (the real trace data and features are collected on Sheepdog)

implement the server-side object access tracing mechanism. By analyzing I/O trace on gateway nodes, I/O characteristics can be mined for the entire storage system.

IV. I/O Characteristics Discovery

In this section, we introduce the design and implementation of the proposed I/O characteristics discovery methodology. Figure 1 shows an overview of this method. It begins with a stream of I/O traces, which can be collected periodically based on data accesses of cloud storage systems. Each line/record in the trace file indicates one object access with various features. Figure 1 illustrates part of the real trace data and features we have collected from Sheepdog. In this specific example, due to the space limit, we show the following features: 1) access time, 2) object ID, 3) access length, 4) access offset, 5) operation code, 6) request ID, 7) object index, 8) object reference count, 9) zone (one object copy per zone), and 10) target node ID. The trace data will be pre-processed for feature normalization and formatted to training datasets. These training datasets are then used by PCA (principal component analysis) module [13] to derive key features (i.e., the important, principal features) from their original features. Each feature represents one dimension describing an attribute of data accesses. Learning key features is essentially the process of dimensionality reduction. Utilizing key features and training datasets, a DBSCAN-based clustering algorithm [24] is used to group relevant objects by calculating the correlation of key features. Based on these I/O characterization results, storage systems can be tuned and optimized for better performance. We will describe the feature normalization, learning key features, clustering, and discuss the efficacy in this section.

A. Feature Normalization

To discover I/O characteristics, traces need to go through a feature normalization pre-processing stage to lay out a foundation for the comparison and key feature selection. The reason is that, in I/O traces, features have values in different units. For example, in Figure 1, the *access time* feature is a 64-bit integer in milliseconds, while the *access offset* feature is a 32-bit integer in bytes. Such a large range of values make it difficult to compare different features and identify their importance. To solve this issue, feature normalization is performed to reduce the range and variance of different feature values. Further, these features with same values are ignored because they do not help distinguish I/O behaviors. In the current study, we also do not consider features that are not in numerical values.

There are many data normalization methods, such as min-max, z-score and decimal scaling normalization methods [25]. We use the *z-score* method for feature normalization as the trials of these methods confirm the z-score delivers the best promise to reduce the value variance across different features and remove outliers [25]. The z-score method subtracts the mean from each feature and further divides it by its standard deviation. It transforms the data set to a distribution with zero mean and unit variance. The conversion function for feature normalization of each data access is described as:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

where x is one value of the feature (e.g., 81920 in *length* feature in Figure 1), μ and σ are the average and standard deviation of all values for the feature. A sample result of feature normalization is shown in Figure 1. It can be seen that the features with large variance are transformed to the same scale, which lays out the foundation for further key feature selection.

B. PCA-based Key Feature Selection

After performing feature normalization, I/O access features are normalized. However, applications often exhibit complex behaviors. It is challenging to select a set of features for I/O characteristics discovery without any domain knowledge of applications [7]. To solve this problem, we introduce the concept of *principal component* to describe and identify I/O behaviors. Assume all data accesses construct a multi-dimensional space/coordinate. Each point in the coordinate represents one object access and each feature represents one dimensionality that describes an attribute of data access (e.g., the *access time* feature represents a dimension that describes when an object is accessed and the *object ID* is another dimension describing which object is accessed). The principal component analysis learns the "main direction" of data accesses with a dimensionality reduction process. This "main direction" represents the dominant data-access pattern of I/O behaviors. They keep the key set of descriptive features and reduce the noises in data accesses without requiring any expertise or domain knowledge.

More specifically, PCA (principal component analysis) method [13], an unsupervised machine learning analysis is used to learn the key features. PCA is a statistical method that captures patterns in multi-dimensional dataset by choosing a set of important dimensionality automatically, the principal components or key features, to reflect covariation among the original coordinate. The input of PCA is the set of normalized features, and the output of PCA is a new subset of features defined by the principal components, usually with less dimensionality. Each principal component has an eigenvector, which indicates the importance of this component. These eigenvector values can be calculated from the covariance matrix in the PCA analysis. Assume the eigenvectors for n principal components are $\lambda_1, \lambda_2, ..., \lambda_n$, respectively, the eigenvector proportion of principal component i is:

$$\frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}. \tag{2}$$

We define the first k principal components as "key features", if the below proportion formula is larger than a threshold, such as 90%.

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{n} \lambda_j} \tag{3}$$

C. Object Clustering

With key features of data accesses, a clustering stage is performed to identify the access similarity among objects for discovering I/O characteristics. As the example in Figure 1 shows, the result of PCA is a new multi-dimensional data set, where each row/record corresponds to one original object access, and each column indicates a principal component. We use the formula (3) to select the first k principal components as the key features. Then, clustering is performed to group data accesses based on their distances calculated by the key features. As each data access has an object ID, we can consider the objects in the same group have high I/O similarity. If there are two or more data accesses for one object in the same group, we will remove duplicate objects.

We have tried three clustering algorithms, nearest neighbor (NN) [24], K-means [26] and DBSCAN [24], and selected the DBSCAN. The reasons are two-fold. First, DBSCAN is simple in terms of the algorithm complexity. It allows fast processing of large data sets with the average time complexity of $O(n \log(n))$, where n is the number of data points. On the contrast, K-means and NN are much more time consuming. K-means has the time complexity of $O(n * k * t)$, where k is number of clusters, t is number of iterative calculations. To find the k closest points, the time complexity of NN is $O(nd + kn)$, where d is the feature number of each data point.

Second, DBSCAN is a density-based clustering algorithm that is very robust and handles noisy data well. In fact, according to our observations, the output data set of PCA stage shows irregular shapes (e.g., most data points reside close to a straight line, as seen in Figure 3). In this case, K-means has low efficiency because it is used to identify a set of data points that congregate around a region in multi-dimensional space (spherical distribution) [26]. Figure 2 shows an example of K-means clustering results after PCA for real traces in Sheepdog (such a small number of data points is used for an easy illustration). The x axis and y axis represent two key features (the results have no units after the PCA stage). It can be seen that K-means algorithm will group data sets into *three* clusters, where *cluster1* and *cluster2* cross two lines. This is counter-intuitive because data points in the same line have better similarity. Instead, DBSCAN clusters these data points along the line (data points with different colors and shapes means different clusters on the line), and the result is much more accurate. NN groups the points from two different lines into the same cluster and does not generate the accurate result.

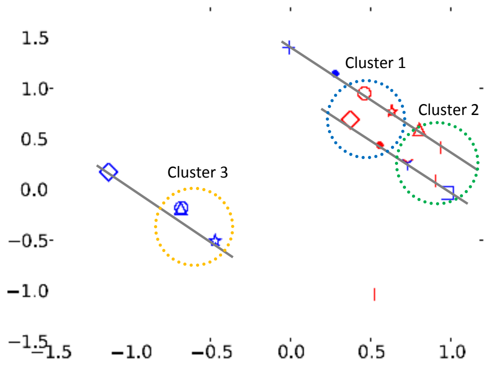
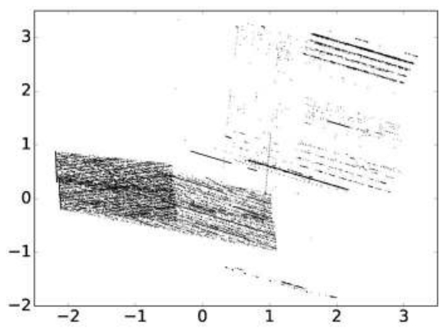
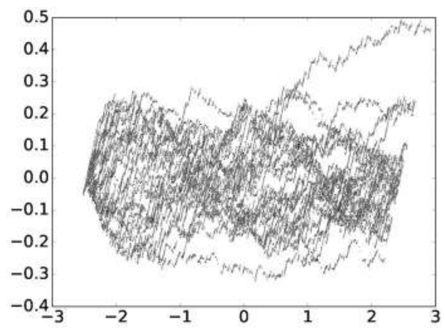
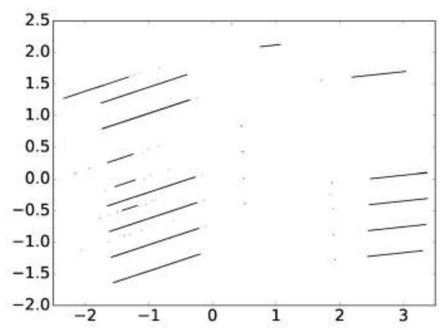


Figure 2: Object clustering with K-means

For clustering objects, DBSCAN uses a distance function to calculate the distance among data accesses to decide whether objects are in the same group. We use the Euclid space distance [24] of key features (key features correlation) to calculate the distance between two accesses. There are two parameters highly relevant with the clustering results of


(a) FIO sequential read


(b) FIO sequential write


(c) FIO 3 randread 3 sequential read

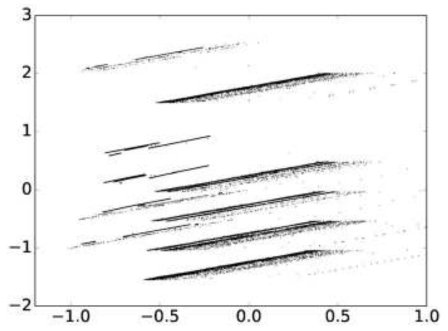

(d) FIO 3 randwrite 3 sequential write

Figure 3: Two key features after PCA method with FIO on one or more VMs. The x axis is the 1st PC and the y axis is the 2nd PC. There are no units for key features after PCA.

DBSCAN. One is a distance threshold *dis_thr* that indicates the maximum distance between two objects allowed in one group. It is actually difficult to obtain an accurate value. In this study, we present a dynamic method to adjust the distance threshold *dis_thr*, as discussed in Section IV-D. The second parameter is *min_samples*, the minimum number of objects in a group. We set the value to 2 to ensure each group at least has two data access points.

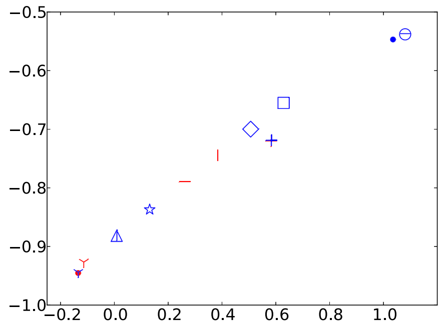
D. Distance Threshold Adjustment

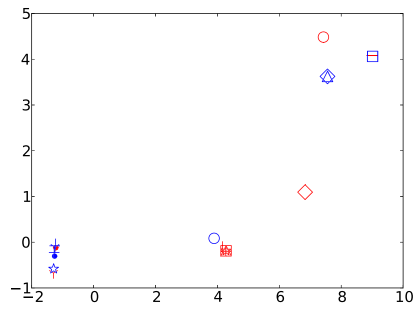
In this section, we discuss how to tune DBSCAN to conduct an appropriate number of clusters for grouping objects. As we have described in the previous section, DBSCAN controls the clustering results through the distance threshold *dis_thr*. If this value is too small, then the average cluster size (the number of accesses in a cluster) will be very large, which cannot distinguish I/O similarity well. On the other hand, if this threshold is too large, the average cluster size will be very small, e.g., one access in the cluster, which leads to no similar objects in the system. To address this challenge, we dynamically adjust the distance threshold *dis_thr* until selecting an appropriate value. Specifically, we use the "elbow" method [27], which examines the variance of the average cluster size for different thresholds (*dis_thr*), and chooses appropriate values for both the threshold value and average cluster size.

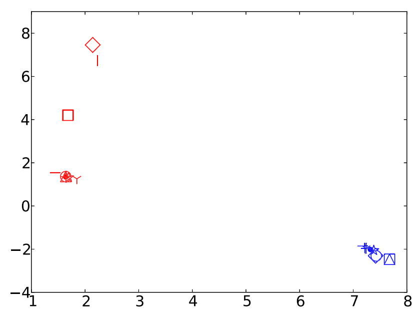
V. Evaluation

In this section, we present experimental evaluations of the proposed I/O characterization methodology. We implemented a lightweight I/O tracing layer in Sheepdog to collect server-side traces, which include more than 20 features such as access time, object ID, length, offset, target node, etc. The experiments were conducted on a local 26-node cluster, including 20 storage nodes and 6 compute nodes hosting VMs. Each storage node has dual 2.5 GHz Xeon 8-core processors, 64GB memory, a 500GB Seagate SATA HDD and a 200GB Intel SSD. The compute nodes are used for running VM clients, where each client is emulated by KVM/QEMU and configured with 2 vCPUs and 8GB RAM. We conducted the experiments using both standard file system benchmark FIO and application workload [28]. We used one storage node as the gateway node to collect all I/O accesses and analyze access patterns.

In the tests, we dynamically adjusted the threshold from 0.000025 to 0.005 to obtain an appropriate threshold for clustering. First, we report the detailed results of I/O characteristics discovery. We conducted the evaluations with FIO benchmarks on one or multiples VMs. For the test on one VM, we launched FIO with data size of $50GB$ and request size of $4MB$. For the test on multiple VMs, we launched FIO with 128 jobs, where each job accessed an independent


(a) FIO sequential read


(b) FIO sequential write


(c) FIO rand read

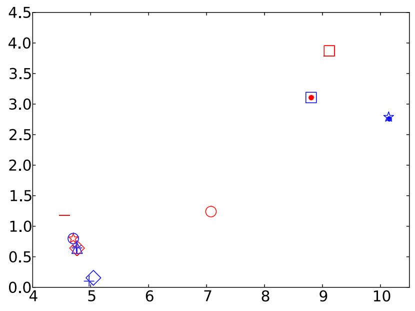
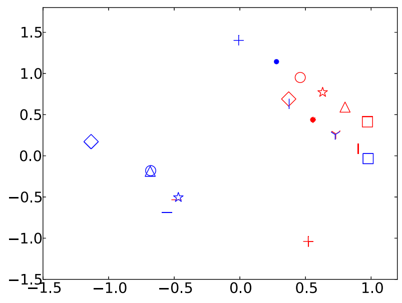
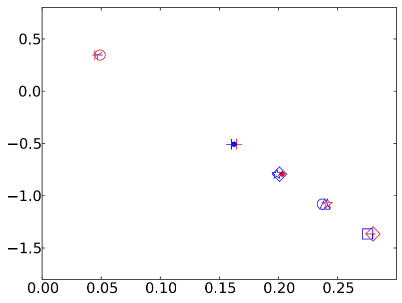

(d) FIO rand write

Figure 4: Object clusters with 20 largest-size with FIO on 1 VM. The cluster sizes vary from 12 to 23. The points with the same color and shape mean they are in the same group. The results show strong object similarities for access patterns.


(a) FIO 3 randread 3 sequential read


(b) FIO 3 randwrite 3 sequential write

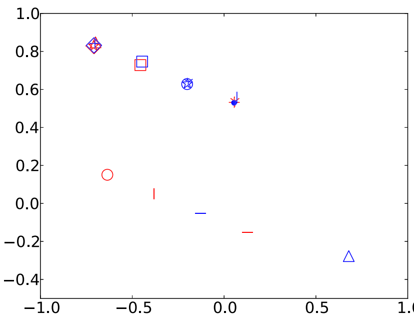

(c) FIO 3 readwrite 3 randreadwrite

Figure 5: Object clusters with 20 largest-size with FIO on 6 VMs. The cluster sizes vary from 11 to 29. The points with the same color and shape mean they are in the same group. The results show strong object similarities for access patterns.

file with $100MB$ in an asynchronous way with the request size of $4MB$.

Figure 3 shows the results of key features learning from I/O accesses with FIO running on 1 VM. Each point in the coordinate indicates an object access. To be intuitive, we plot $40K$ accesses based on the first principal component (1st PC) and second principal component (2nd PC) after PCA. Supposing the data accesses as points distribution in a multi-dimensional basis coordinate, the two principal components (x axis and y axis) reflect the dominant I/O behavior for I/O trace in the new 2-dimensional basis coordinate.

Two observations can be made from the PCA results. First, the results show that data-access patterns vary with different workloads (mean different shapes in the coordinates). But we see most data points reside close to different straight lines. All of them formed linear clusters locally and located in certain regions. The I/O similarity can be accurately identified as DBSCAN works well for this distribution. Specially, the dark region with a large number of points means “high-density cluster”, which will be re-clustered.

Second, besides *FIO sequential read*, other three benchmarks (sequential write/rand read/rand write) also have strong object similarities. One reason we infer is that the operation system in VM can also have its behaviors and affect the access patterns (e.g., the operation system call, I/O scheduling).

Similar to the test results on 1 VM, most of the data accesses construct line distribution in different regions for multiple VMs tests, as shown in Figure 3. Among them, *FIO 3 randread 3 sequential read* means tests on 6 VMs, in which three FIO for rand read and three FIO for sequential read.

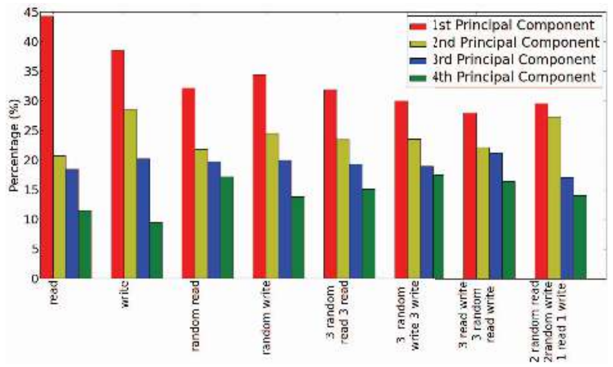


Figure 6: The proportion of variance for principal components with different FIO tests running on 1 VM and 6 VMs

To show the exact eigenvector proportion accounted for each principal component (which is also used for key features selection), we gave the values of first four principal components in Figure 6. It can be seen that the first principal component in the tests accounts for a large eigenvector proportion. Specially, the 1st PC of FIO *read* accounts for up to 44.3% eigenvector proportion. Other trace can account from 27.9% to 38.2%, thus more principal components can be used until getting the major proportion. PCA does return a less dimensional data set in most cases. However, each dimensionality actually corresponds to a combination of multiple features/attributes of the original dataset, instead of representing a single manually selected feature. In our evaluations, although the first four principal components reach more than 90% of the variance, they do not mean only four features of the original dataset take effect. They represent the most important characteristics of I/O accesses that can be used as key features for pattern analysis. Also, Figure 6 represents only one case. In fact, if the features have large variations, more principal components will be selected as key features.

As the number of clusters is large, we choose the first 20 largest clusters for each test (excluding "high-density clusters", indeed the number of "high-density clusters" are less than 3 in each test), as shown in Figure 4 and Figure 5. The results are beneficial for data I/O optimization (e.g., data prefetching) in two-fold. One is that they identify the objects with high similarity in I/O behaviors in the same group. The other is the cluster sizes are appropriate for data I/O optimization (The average cluster size is from 2 to 25, the maximum cluster size is less than 30).

VI. Conclusion

In the big data era, the I/O performance has become a critical issue for many data-intensive applications on clouds. Discovering patterns of I/O access behaviors and using them for performance improvement is a promising technique for cloud storage systems. Numerous studies have been conducted in this space. However, most of them are either limited to specific, user-defined features for pattern analysis or heavily rely on domain knowledge about the applications, limiting their usage.

In this paper, we have introduced a new method for I/O characteristics and pattern discovery in cloud storage systems. Different from existing approaches, this method intends to capture data-access features as many as possible to eliminate the bias for specific workloads. It utilizes the principal component analysis to retrieve key features from traces automatically. Based on learned key features, a density-based clustering, i.e., DBSCAN, is performed to mine objects correlation and to group objects for revealing I/O characteristics. In this way, the I/O characteristics and patterns are analyzed and discovered without any domain knowledge. We also conducted extensive experimental evaluations to validate the proposed I/O characterization methodology. In the future, we plan to implement use cases for I/O optimizations, especially for heterogeneous cloud storage systems.

Acknowledgment

We are thankful to the anonymous reviewers for their valuable feedback. This research is supported in part by the National Science Foundation under grant CNS-1338078, CNS-1526055, IIP-1362134, CCF-1409946, and CCF-1718336. This research is also supported by the International Cooperation Project of Institute of Information Engineering, Chinese Academy of Sciences under Grant No. Y7Z0511101.

References

[1] Y. Yin, J. Li, J. He, X. Sun, and R. Thakur, "Pattern-direct and layout-aware replication scheme for parallel I/O systems," in *Proceedings of the IPDPS'13*.

[2] Y. Yin, S. Byna, H. Song, X. H. Sun, and R. Thakur, "Boosting application-specific parallel I/O optimization using IOSIG," in *Proc. of the CCGrid'12*.

[3] T. M. Madhyastha and D. A. Read, "Learning to classify parallel input/output access patterns," *IEEE Transactions on TPDS*, vol. 13, no. 8, pp. 802–813, 2002.

[4] T. M. Madhyastha and D. Read, "Exploiting global input output access pattern classification," in *Proc. of the ACM/IEEE Conference in Supercomputing*, 1997.

[5] M. Dorier, S. Ibrahim, G. Antoniu, and R. Ross, "Omnisc'IO: a grammar-based approach to spatial and temporal I/O patterns prediction," in *Proc. of the SC'14*.

[6] J. He, J. Bent, A. Torres, G. Grider, G. Gibson, C. Maltzahn, and X. Sun, "I/O acceleration with pattern detection," in *Proc. of the HPDC'13*, 2013, pp. 25–36.

[7] Y. Chen, K. Srinivasan, G. Goodson, and R. Katz, "Design implications for enterprise storage systems via multi-dimensional trace analysis," in *Proc. of SOSP'11*, 2011, pp. 43–56.

[8] P. Xia, D. Feng, H. Jiang, L. Tian, and F. Wang, "Farmer: a novel approach to file access correlation mining and evaluation reference model for optimizing peta-scale file system performance," in *Proc. of the HPDC'08*, 2008.

[9] Z. Li, Z. Chen, and Y. Zhou, "Mining block correlations to improve storage performance," *ACM Trans. on Storage*, vol. 1, no. 2, pp. 213–245, 2005.

[10] M. Sivathanu, V. Prabhakaran, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Semantically-smart disk systems," *Proc. of the FAST'03*, vol. 3, pp. 73–88, 2003.

[11] D. Dai, Y. Chen, D. Kimpe, and R. Ross, "Provenance-based object storage prediction scheme for scientific big data applications," in *Proc. of the IEEE International Conference on Conference on Big Data*, 2014.

[12] L. Wang, Y. Ma, A. Y. Zomaya, R. Ranjan, and D. Chen, "A parallel file system with application-aware data layout policies for massive remote sensing image processing in digital earth," *IEEE Transactions on TPDS*, 2015.

[13] I. Jolliffe, *Principal component analysis*. John Wiley and Sons, Ltd, 2002.

[14] H. Luu, B. Behzad, R. Aydt, and M. Winslett, "A multi-level approach for understanding I/O activity in HPC applications," in *Proc. of the Cluster'13*.

[15] "HPC open source software projects: LANL-Trace," 2017. [Online]. Available: http://institute.lanl.gov/data/software/lanl-trace.

[16] S. A. Wright, S. D. Hammond, S. J. Pennycook, R. F. Bird, J. A. Herdman, I. Miller, A. Vadgama, A. Bhalerao, and S. A. Jarvis, "Parallel file system analysis through application I/O tracing," *The Computer Journal*, 2013.

[17] D. Dai, F. S. Bao, J. Zhou, and Y. Chen, "Block2vec: A deep learning strategy on mining block correlations in storage systems," in *Proc. of the 45th International Conference on Parallel Processing Workshops*, 2016.

[18] Y. Liu, R. Gunasekaran, X. S. Ma, and S. S. Vazhkudai, "Server-side log data analytics for I/O workload characterization and coordination on large shared storage systems," in *Proc. of the SC'16*, 2016.

[19] J. Oly and D. Reed, "Markov model prediction of I/O requests for scientific applications," in *Proc. of the international conference on Supercomputing*, 2002.

[20] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed prefetching and caching," in *Proc. of the SOSP*, 1995, pp. 79–95.

[21] K. H. P. MCarns, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," *ACM Transactions on Storage*, 2011.

[22] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale I/O workloads," in *Proc. of Cluster Computing and Workshops*, 2009.

[23] "Sheepdog project," 2017. [Online]. Available: https://github.com/sheepdog/.

[24] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," *Kdd*, 1996.

[25] L. A. Shalabi, Z. Shaaban, and B. Kasasbeh, "Data mining: A preprocessing engine," *Journal of Computer Science*, vol. 2, no. 9, pp. 735–739, 2006.

[26] E. Alpaydin, "Introduction to machine learning," *MIT Press, Cambridge*, 2004.

[27] J. D. J. Ketchen and C. L. Shook, "The application of cluster analysis in strategic management research: An analysis and critique," *Strategic Management Journal*, vol. 17, no. 6, pp. 441–458, 1996.

[28] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "Bigdatabench: a big data benchmark suite from internet services," in *Proc. of HPCA*, 2014.