CloudCluster: Unearthing the Functional Structure of a Cloud Service

Weiwu Pang, *University of Southern California;* Sourav Panda, *University of California, Riverside;* Jehangir Amjad and Christophe Diot, *Google Inc.;* Ramesh Govindan, *University of Southern California*

https://www.usenix.org/conference/nsdi22/presentation/pang

This paper is included in the Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation.

April 4–6, 2022 • Renton, WA, USA

978-1-939133-27-4

CloudCluster: Unearthing the Functional Structure of a Cloud Service

Weiwu Pang
University of Southern California

Sourav Panda
University of California, Riverside

Jehangir Amjad
Google Inc.

Christophe Diot
Google Inc.

Ramesh Govindan
University of Southern California

Abstract

In their quest to provide customers with good tools to manage cloud services, cloud providers are hampered by having very little visibility into cloud service functionality; a provider often only knows where VMs of a service are placed, how the virtual networks are configured, how VMs are provisioned, and how VMs communicate with each other. In this paper, we show that, using the VM-to-VM traffic matrix, we can unearth the *functional structure* of a cloud service and use it to aid cloud service management. Leveraging the observation that cloud services use well-known design patterns for scaling (*e.g.,* replication, communication locality), we show that *clustering* the VM-to-VM traffic matrix yields the functional structure of the cloud service. Our clustering algorithm, CloudCluster, must overcome challenges imposed by scale (cloud services contain tens of thousands of VMs) and must be robust to orders-of-magnitude variability in traffic volume and measurement noise. To do this, CloudCluster uses a novel combination of feature scaling, dimensionality reduction, and hierarchical clustering to achieve clustering with over 92% homogeneity and completeness. We show that CloudCluster can be used to explore opportunities to reduce cost for customers, identify anomalous traffic and potential misconfigurations.

1 Introduction

As more online services migrate to the cloud, and as the user base of these services increases, the complexity and scale of cloud deployments has increased significantly. Today, cloud services routinely use tens of thousands of VMs, geographically dispersed for reliability and low-latency access to customers. Monitoring and managing a cloud deployment can be significantly challenging, since the performance, cost, and reliability of the service can depend on a large number of factors: how the cloud customer maps logical functionality to VMs, how the VMs are provisioned, where they are located, how well the paths between the VMs are provisioned, and so on. More generally, how well a cloud service works depends both on how well a customer designs the service, and how well the provider provisions the underlying infrastructure.

Cloud service monitoring. Cloud providers struggle to provide customers with insights on the performance and reliability of a cloud service. This is because, while a VM provides a very convenient abstraction for computing and communication, the provider has (by design) very little *visibility* into cloud service logic embedded in the VM. This lack of visibility prevents providers from being able to relate problems observed at the service level to issues in the underlying infrastructure. For a given service, a provider often only knows where the VMs are, how much compute and storage each VM is provisioned with, customer-supplied names for the VMs, and how much traffic each VM exchanges with other VMs in the service. Customers are often loath to reveal more, for business and privacy reasons.[1]

Today, major cloud providers (such as Amazon Web Services [12], Azure Cloud [2] and Google Cloud Platform [3]) provide customers with monitoring services. Their monitoring services (AWS CloudWatch [12], Azure Monitor [2] and Google Cloud Monitoring [3]) expose, using customizable dashboards, metrics capturing the state and activity of the cloud service's VM instances (*e.g.,* their CPU and disk utilization, and the volume of network traffic to and from instances) as visible to the cloud provider, as well as other measures of the underlying networking infrastructure (*e.g.,* loss rates between instances). Some of these monitoring services also provide custom alerting mechanisms. Customers can define metrics that capture user-perceived performance, and configure alerts when these metrics exceed service-level objectives that cloud customers have with *their* customers.

Goal. Given that cloud service monitoring provides a competitive advantage, cloud providers continuously seek to add

[1] **Ethical considerations:** For the 15 cloud projects we used in the evaluations in the paper (§4), we obtained explicit consent. For each project, we only used information available to the cloud provider: VM locations, VM names, and the VM-to-VM traffic matrix. We used the VM-to-VM traffic matrix to generate the clusters, and names and locations to evaluate the performance of CloudCluster. After our evaluations, we shared the results with each customer, and obtained feedback.

innovative capabilities to their monitoring systems, despite their limited visibility into cloud services. In this paper, we describe a new capability not, to our knowledge, previously considered in the literature: inferring the *functional structure* of a cloud service, *i.e.,* how a cloud service is modularized across its many VMs. Our work is inspired by a body of prior work on inferring structural relationships between components in a distributed system (§6).

To explain what we mean by functional structure, consider Figure 1(a) which shows the connectivity graph of VMs (which VMs communicate with which other VMs) of a cloud service. These VMs reside in different cloud regions (roughly, parts of a continent, see §2); most communication is within a region, but some communication exists across regions. With just the information that the cloud provider has, it can only obtain this kind of a view of the project. Now suppose that this cloud service is, in fact, architected as in Figure 1(b): it has a front-end load-balancer and a backend processing layer. With the information she has, the cloud service operator's conceptual view of the structure of the service might be as shown in Figure 1(c): its VM instances are spread across two regions, with load balancer VMs (in red on the cluster on the left, and green on the cluster on the right) communicating with the processing-layer VMs (in magenta and cyan respectively) but not with other load balancer VMs, and the processing-layer VMs in each region communicating with each other as well. In addition, one of the load-balancers communicates with processing VMs in the other region (*e.g.,* due to overload in its own region).

The focus of our paper is to *unearth* the structure in Figure 1(c) *only using information available to the provider*. Specifically, we aim to develop methods that can extract this structure in which VMs are grouped into *VM groupings* by function and location. Ultimately, this will enable the provider to represent the service by a compact inter-grouping graph abstraction (Figure 1(d)).

In deriving the representations shown in Figure 1(c-d), CloudCluster can only determine that VMs in a cluster likely perform the same function, but cannot tell *which* function they perform (for example, whether the VMs run load-balancers, or image transcoders). This mitigates any privacy concerns cloud customers might have. Even so, we expect that in an actual deployment, a cloud provider will obtain consent from the customer before applying CloudCluster to the customer's service.

Approach. We hypothesize that we should be able to infer VM groupings by *clustering the VM-to-VM traffic matrix of a cloud service*. Clustering a traffic matrix implies grouping together similar rows; two rows are similar if the traffic from their corresponding VMs to all other VMs is similar. Intuitively, two functionally similar VMs are likely to satisfy this property. For example, how two load-balancer VMs in a region are likely to communicate with all other processing layer VMs in the same region is likely to be similar, so clustering

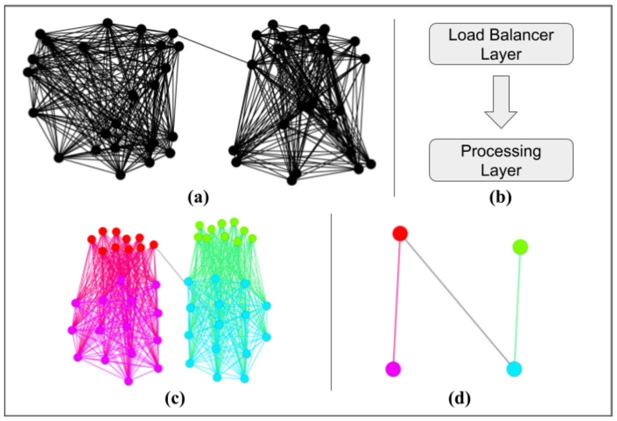

Figure 1: An example of different views of a cloud service: (a) VM connectivity graph as visible to the cloud provider; (b) The service architecture; (c) VM connectivity graph colored by function and location (the desired output of CloudCluster); (d) A compact inter-grouping graph abstraction.

will group them together. Furthermore, we expect clusters to be large because of the horizontal scaling employed by cloud services, which replicate processing or storage at a given layer using functionally identical VMs (*e.g.,* databases, in-memory stores, image transcoders *etc.*).

Challenge. Analyzing large VM-to-VM traffic matrices of real-world cloud services presents two challenges: scale, and robustness to variability and noise. At the scale of tens of thousand of VMs, any analysis must overcome the curse of dimensionality [60]; the sparsity of the traffic matrices in these higher-dimensions makes it difficult to derive insights from the data. Moreover, cloud services often vary in VM-to-VM traffic by several orders of magnitude, and methods of inferring their properties must accommodate this variability and be robust to noise introduced by the underlying measurement methodology (*e.g.,* by traffic sampling).

Contributions. This paper shows that *clustering* the VM-to-VM traffic matrix of a cloud service provider can help determine the functional organization of a cloud service, and that these clusters can be a useful abstraction for providing cloud customers with actionable insights into their service. To this end, the paper makes three contributions.

First, we develop a clustering algorithm, CloudCluster, that clusters VMs by similarity in their network communication characteristics (§3.4). CloudCluster is a novel combination of techniques, some known, and others new, to address the scaling and robustness challenges mentioned above. At its core, it uses a variant of *hierarchical clustering*, called agglomerative clustering [48] to determine clusters. This approach clusters VMs by proximity in some high-dimensional space. It requires a way to determine distance thresholds, and CloudCluster determines these thresholds dynamically in a data-driven manner. To scale better, it employs dimensionality reduction, and to be robust to variability in traffic volumes it scales traffic features (see §3 for more details).

Second, by evaluating the resulting clusters on 15 different cloud service *projects*[2] (§4), we experimentally demonstrate that the resulting clusters *group together VMs by location and function*: *i.e.,* all VMs in a cluster are geographically co-located, and they perform the same function[3]. We verify this on cloud services that name VMs by function; for these projects, CloudCluster has homogeneity and completeness scores (metrics equivalent to precision and recall, respectively) of over 0.92 and 0.94 respectively.

Third, we demonstrate ways in which CloudCluster can be used to provide customers with actionable insights (§5). CloudCluster can analyze the inter-cluster graph (Figure 1(d)) to identify opportunities for reconfiguring VM placements to reduce cost: in one case, we found opportunities to reduce cost by 41.2% by provisioning an additional cluster to minimize inter-region traffic. It can also be used to detect anomalous traffic between clusters, to identify traffic shifts within a cloud project, or structural changes in the project across time. From 25 traffic anomalies reported either by an internal anomaly detector or the customer, a CloudCluster-based anomaly detector detected every anomaly, and identified the impacted clusters. CloudCluster can be used to detect potentially mislabeled VM names (names that do not reflect function) or mis-provisioned VMs. In some projects, up to 1% of VMs appear to be mis-provisioned. In others, over 7% a project's VMs appear to be mis-labeled — their traffic patterns differ from the majority of VMs that have the same labels.

2 Anatomy of a Cloud Service

In this section, we provide a brief background on the structure of cloud services. Our description focuses on Google's cloud services; different service offerings may differ from this description in the details.

Google's cloud resources are hosted in multiple locations worldwide. The network is subdivided into *regions* which are in turn divided into *zones* [9]. A region represents a part of a continent, and zones represent disjoint geographical areas within a region in which infrastructure resides. This partitioning permits cloud customers to coarsely control the placement of VMs to, for example, ensure low-latency access to customers, control cost and ensure high availability.

Customers can organize their cloud service into *projects* [4], which are granular functional groupings that simplify management of a cloud service. For example, an ad-supported social media service can have different projects for the user-facing front-end, the ads subsystem, and an analytics backend. Depending on the scale of the service, projects can be large, spanning tens of thousands of VMs across multiple regions. In this paper, we focus on the structure of projects.

VMs in a project are connected by one or more *virtual networks* [6] that provide isolation. Customers can organize VMs into *sub-networks* [7]: VMs in one sub-network must all be within the same region, and communicate over the same virtual network. Sub-networks simplify VM management tasks: *e.g.,* IP address assignment.

Customers populate projects with VMs. To create a VM, the customer: (a) selects a *configuration* for the VM (configurations differ in compute and storage), (b) specifies the VM's *name* (the name is opaque to the provider, but customers may embed hierarchical structure into a name; some customers name VMs by function, a feature we leverage in evaluating CloudCluster in §4), (c) identifies the sub-network and the virtual network the VM uses, and (d) specifies the region and zone the VM is located in. This is the *only* information a cloud customer explicitly provides to Google. In addition, if customers opt in to flow logging [10], the logging service records VM-to-VM traffic for each enabled project.

3 CloudCluster Design

In this section, we describe CloudCluster, whose goal is to discover the underlying structure of a cloud project. We discuss how it scales to large cloud projects, while being robust to noise and variability.

3.1 Goals, Approach, and Overview

Notation. The input to CloudCluster is a VM-to-VM traffic matrix for a cloud project, containing traffic volumes between each VM over a fixed aggregation window.[4] Traffic volumes are obtained by sampling flows. In §4, we discuss the actual values of the aggregation window and the sampling frequency. Formally, we denote this traffic matrix by $\mathbf{Y}$, with dimensions $n \times m$, where n is the number of source VMs (belonging to this specific project under consideration) and m is the number of destination VMs (which do not all have to belong to the same project, since VMs in a project can communicate with external clients or VMs in other projects).[5] The i, j-th entry y_{ij} of $\mathbf{Y}$ represents the volume of traffic (in bytes) from VM i to VM j, where $i \in [n], j \in [m]$.

Challenge: Noise. Since $\mathbf{Y}$ is sampled, it is bound to be noisy. Aside from the error induced by sampling, measurement errors and randomness in traffic patterns can also induce noise. To model this, we can write:

$$\mathbf{Y} = \mathbf{M} + \mathbf{E} \tag{1}$$

[2]As discussed in §2, a project is a granular functional grouping within a cloud service.

[3]In this paper, we use the term *function* to denote a long-lived heavyweight service that forms part of a cloud service; we do *not* consider services deployed using ephemeral cloud functions (*e.g.,* lambdas).

[4]CloudCluster uses minimum possible aggregated information, namely the communication volume. Other metadata (*e.g.,* port numbers, process names) might be helpful in finding the functional structure. CloudCluster does not use this information. With consent from the customer, it might be possible to use this to improve our clustering, but we have left it to future work, in part because it is not clear whether customers will consent to revealing additional information.

[5]CloudCluster does not currently model traffic to cloud native services, like traffic to Google Cloud Storage [5]). Identifying traffic volumes from these services requires using other instrumentation services (*e.g.,* storage service logs), and we have left this to future work.

where $\mathbf{M}$ is the unobserved noise-free, *true* traffic matrix and $\mathbf{E}$ is a noise matrix. We assume e_{ij} is independent of all other entries and $\mathbb{E}[e_{ij}] = 0$ and $Var[e_{ij}] < \infty, \forall i \in [n], \forall j \in [m]$. This implies that $\mathbf{M} = \mathbb{E}[\mathbf{Y}]$.

Challenge: Scale. $\mathbf{Y}$ can be large, since projects can have tens of thousands of VMs. We have observed, through manual inspection of cloud projects, that to enable projects to scale, designers often group VMs that perform similar functions. At the front-end, load-balancers redirect requests to VMs that scale with the request load; all these VMs perform the same function (*e.g.,* handle requests). In turn, at the back-end, these VMs may invoke other services that may be replicated across several identical VMs, or may send the request to a coordinator VM that invokes an iterative distributed computation spread across several identical VMs. Such structures result in *VM groupings*. We hypothesize that VMs in a group have similar traffic patterns (in terms of which VMs they communicate with, and the volume of traffic). If this hypothesis is true, then $\mathbf{M}$ must be a *low rank matrix*.

We can formalize a VM grouping as:

Definition 1 ***VM Grouping***. *Let a VM Grouping be denoted by $\mathbf{S}_i$. Let m_u denote a row of matrix, $\mathbf{M}$. m_u belongs to $\mathbf{S}_i$, if*

$$d(m_u, m_v) < \min_r \{d(m_u, m_r)\}, \forall v \in \mathbf{S}_i, \forall r \notin \mathbf{S}_i$$
$$\wedge d(m_u, m_v) < \delta, \forall v \in \mathbf{S}_i$$

$d(\cdot,\cdot)$ is some distance function and δ is a distance threshold.

Definition 1 implies that *similar* rows will be grouped together if they are most similar to each other and their similarity, quantified via a distance function, $d(\cdot,\cdot)$, is less than the distance threshold. In practice, this threshold can be different for different cloud projects.

Goal and Approach. Our goal is to discover all VM Groupings present in a cloud project. To do this, CloudCluster (a) estimates $\mathbf{M}$ and then (b) clusters VMs (rows of $\mathbf{M}$ with similar traffic patterns) to find the VM groupings.

To estimate $\mathbf{M}$, we leverage prior work, such as [27] and [21], which show that in a setting like ours, we can estimate well and with consistency the low-rank and noise-free, but unobserved, matrix, $\mathbf{M}$, from a random observation of the noisy matrix $\mathbf{Y}$, where $\mathbf{M} = \mathbb{E}[\mathbf{Y}]$.

Clustering algorithm: Overview. Using the estimate $\hat{\mathbf{M}}$, CloudCluster's clustering algorithm[6] seeks to extract VM Groupings according to Definition 1, with $\mathbf{M}$ replaced by $\hat{\mathbf{M}}$. It must also address the scalability and robustness challenges identified above. To do this, CloudCluster's algorithm has four components (Algorithm 1): 1) Feature scaling to transform the input traffic matrix. 2) Matrix estimation to estimate $\mathbf{M}$. 3) Hierarchical clustering to group similar VMs. 4) Cluster merging to fuse similar clusters. We describe each component in the following subsections.

[6]This is orthogonal to prior work that has explored clustering to group similar traffic matrices [59].

Algorithm 1: Steps in VM clustering

```
input  : Y, threshold θ to merge similar clusters
output: Clusters merged_clusters
1 scaled_Y = feature_scaling(Y);
2 scaled_M̂ = TruncatedSVD(scaled_Y);
3 clusters = hierarchical_clustering(scaled_M̂);
4 merged_clusters = merging(clusters, scaled_M̂, θ);
```

3.2 Feature Scaling

What is a feature and why scaling is necessary. Each row of $\mathbf{Y}$ can be treated as a (high-dimensional) feature. Then, identifying similarity in this feature space is equivalent to identifying VMs that have similar traffic patterns.

In practice, even within a single project, traffic volumes between VMs can span several orders of magnitude. This can make it difficult to discriminate between low and medium volume traffic patterns. Clustering relies on a distance metric, and many applicable distance metrics are disproportionately sensitive to larger values.

Log Scaling. Feature scaling normalizes the range of each feature to enable clustering algorithms to be robust to highly variable traffic volumes. Of the existing feature scaling methodologies, standardization and minmax scaling cannot handle the range of traffic volumes we see in cloud projects. Standardization replaces each feature's value by how many standard deviations it is above or below the mean [1]. Minmax scaling transforms each individual feature value into the ratio between that value's distance from the minimum to the range of values [1]. Traffic in cloud projects can span several orders of magnitude (from 0 to 10^9) and have skewed distributions; linear transformations like minmax scaling, or those that assume Gaussianity, like standardization, do not work well. For this reason, we choose *log scaling*, which uses the natural logarithm of the traffic instead of the original values; this handles volume variability much better (we demonstrate this experimentally in §4.4).

3.3 Estimating M

Estimating singular values. Singular value thresholding can produce a good estimate, $\hat{\mathbf{M}}$, of the low-rank $\mathbf{M}$, using only observations from $\mathbf{Y}$ (see [27], [21], [20]). However, estimating the number of singular values to keep cannot be determined exactly. After performing Singular Value Decomposition of the matrix, we choose the number of singular values to retain based on an *elbow* finding heuristic such as one introduced by [51]. The *elbow* suggests the approximate number of singular values to retain because most of the singular values after the *elbow* contribute little to the spectrum of the matrix. Figure 2 shows the spectrum of singular values for a traffic matrix for a project with over 3000 VMs. The sharp decline in the spectrum after about 50 singular values is indicative of a low-rank structure. Singular values in the tail which don't quite decay to 0 indicate random noise (which tends to spread

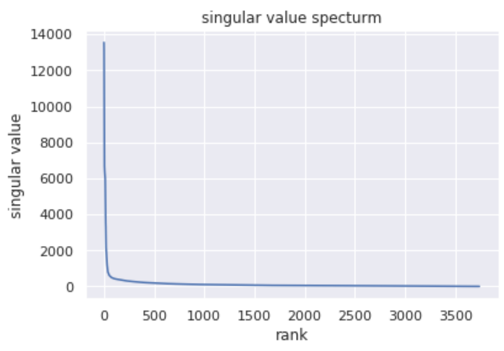

Figure 2: Singular Value spectrum of a traffic matrix with dimension (3742x3271)

across all orthogonal directions) with small finite variance (indicated by the small magnitude).

Extracting an r-rank approximation of M. Once the number of singular values r is heuristically determined, performing an SVD produces the reduced rank estimate of the original matrix. Specifically, given the $n \times m$ original traffic matrix $\mathbf{Y}$, SVD produces the reduced dimension matrix $\hat{\mathbf{M}}$, such that:

$$\hat{\mathbf{M}} = \mathbf{U_r}\mathbf{\Sigma}_r\mathbf{V_r^T},$$

where $\mathbf{\Sigma}_r$ is an $r \times r$ diagonal matrix of the singular values of $\mathbf{Y}$, $\mathbf{U_r}$ and $\mathbf{V_r}$ are orthonormal bases of dimensions $n \times r$ and $m \times r$, respectively. $\hat{\mathbf{M}}$ is a low-rank, *i.e.,* rank = $r \ll \min\{n, m\}$, approximation to the original matrix. However, $\hat{\mathbf{M}}$ is of dimensions $n \times m$. We need to project this matrix to an $n \times r$ subspace which will allow us to retain all the rows (associated individually to VMs), each of r-dimensional feature (columns). We denote this desired matrix by $\hat{\mathbf{M}}_\mathbf{r}$, determined by:

$$\hat{\mathbf{M}}_\mathbf{r} = \mathbf{U_r}\mathbf{\Sigma}_r$$

Effectively, the retained r singular values of the original matrix $\mathbf{Y}$ determine how to scale each of the $r-$dimensional orthonormal vectors in $\mathbf{U_r}$. $\hat{\mathbf{M}}_\mathbf{r}$ remains a good approximation of $\mathbf{M}$ (in a reduced dimensional subspace) because it is simply a projection of each of the rows in $\hat{\mathbf{M}}$ (which is the best rank-r approximation of $\mathbf{M}$) on to a r-dimensional subspace. Both $\hat{\mathbf{M}}$ and $\hat{\mathbf{M}}_\mathbf{r}$ are of rank r, and have the same norm.

The key benefit of SVD. Traffic matrices obtained from large cloud projects can have tens of thousand of rows and columns. The distance functions (used to compute row-similarity) scale exponentially in the number of features/columns. Moreover, in high dimensional feature spaces and with noisy data, distance metrics are unreliable [60] (the curse of dimensionality). Given this, a reduced-rank estimation of $\mathbf{M}$, and projection on to a feature-space of reduced dimensions allows our algorithm to remain robust to scale while retaining much of its structure.

3.4 Hierarchical Clustering

Infeasible clustering methods. Given the original matrix $\mathbf{Y}$, or the rank-r estimate $\hat{\mathbf{M}}_\mathbf{r}$, we can use traditional clustering techniques to find VM Groupings. For instance, prior approaches like [28] have established links between dimensionality reduction and K-Means clustering. However, for our use-case we would like to use dimensionality reduction for robustness to scale and noise but maintain fine control over the number of clusters to produce. Therefore, given that we do not know the number of clusters to produce, much of the existing work around K-Means [41] does not suffice for our needs. Density-based approaches such as DBSCAN [32] and OPTICS [22] do not require the number of clusters as input. However, they rely on other threshold parameters, estimating which requires domain knowledge (*e.g.,* information about a project beyond the sampled traffic volumes we have available) and maybe hard with high-dimensional data. MeanShift [29] and Affinity Propagation [33] also don't require the number of clusters, but their main drawback is time complexity, which depends on the number of iterations until convergence. We also show that MeanShift and Affinity Propagation don't perform well in the context of VM clustering in section 4.4.

Agglomerative clustering. Similar to density-based approaches, hierarchical clustering does not require the number of clusters *a priori*. CloudCluster uses agglomerative clustering [48], a bottom-up hierarchical clustering approach: each VM (row) starts in its own cluster, and clusters are recursively merged together. It uses Ward linkage [56] to determine which two clusters should be merged: at each iteration, this technique selects two clusters that minimize the increase in total within-cluster sum of squared error [44]. In the context of clustering VMs, doing this produces clusters of VMs with homogeneous traffic patterns, and this variance-minimizing property is similar to the K-Means objective function. The output of agglomerative clustering is a dendrogram (tree) of VMs (rows); the leaf nodes are the VMs (rows) and the non-leaf nodes are the nested clusters. Each non-leaf node has a value ("height"), which is the Ward distance [44] between the two entities merging at that node.

From hierarchical to flat clustering. In a dendrogram, each non-leaf node represents a potential cluster (containing all the leaf nodes in its sub-tree). CloudCluster must extract disjoint clusters from this dendrogram. To do this, it can use a static height threshold: each non-leaf node higher than this threshold is a distinct cluster. But, determining the threshold requires domain knowledge for each project. Instead, CloudCluster cuts the dendrogram based on *cluster inconsistency* [54]. For a given non-leaf node in the dendrogram with height h, if its sub-tree contains nodes with heights $H = \{h_0, h_1, \ldots\}$, and mean of the heights is $\overline{H}$, and the standard deviation is $\sigma(H)$, the *inconsistency* of the node is: $inc = \frac{h-\overline{H}}{\sigma(H)}$.

When deciding whether to merge two sub-trees (or nested clusters), the inconsistency metric quantifies how different the new merged cluster would be compared to the nested clusters within it. A low value means that the merged cluster would be similar to the nested clusters under it. Conversely, a high

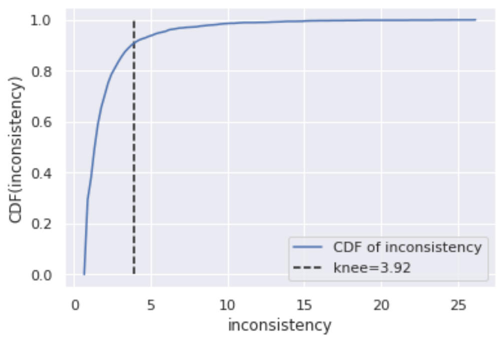

Figure 3: CDF of inconsistency value and the knee

inconsistency means that the merged cluster contains nested clusters which are fairly different. Therefore, the algorithm merges nested clusters when the inconsistency score is less than a threshold, μ.

Estimating μ. Instead of manually selecting the threshold μ, we use the following technique to estimate it. Closer to the leaves of the dendrogram, inconsistency values will be small. They will increase at non-leaf nodes higher in the dendrogram. For many projects, the distribution of inconsistency values is similar to Figure 3. This suggests that the *knee* of this curve is a good choice for μ because it identifies a transition between low and high inconsistency values. We use the knee locator implemented by [51] to determine μ. Then, we cut the dendrogram based on the threshold μ, resulting in a set of clusters.

3.5 Cluster Merging

In practice, we have found that our approach produces, for projects with thousands of VMs, tens or hundreds of clusters with small internal variation in terms of VM traffic patterns. However, it is too aggressive, and we find we can merge some of these clusters in a fast post-processing step. For this, we determine the centroid of each cluster produced by hierarchical clustering. Each centroid can be viewed as a *feature* of the candidate clusters. We treat each of these centroids as a new entity and cluster these entities. Inspired by MeanShift [29] which fuses clusters that are close to each other by comparing the distances to a threshold, we calculate the pairwise cosine distances of the clusters centroids and recursively merge pairs of clusters until no two clusters have a centroid distance less than a fixed merging threshold θ.

4 CloudCluster Evaluation

The goal of the evaluation is to demonstrate that CloudCluster produces clusters that are consistent with VMs grouped by location and function. In other words, *in each cluster, all VMs are in the same zone, and perform the same function.*

4.1 Methodology and Metrics

Dataset. We use anonymized, aggregated flow logs (specifically, Google VPC logs [10], please see footnote on page 1 for a statement of the ethical use of customer data.) from cloud customers to generate our evaluation dataset. Our dataset includes projects ranging from a few thousand VMs to those with tens of thousands of VMs. We do not consider smaller (10-20 VMs) projects in our analysis; at these scales, less sophisticated tools can provide actionable insights. The dataset includes projects of VMs with various type of workloads (*e.g.*, web servers, load balancers, image transcoders, key-value stores *etc.*). It includes projects that are internal to Google and those belonging to external customers. Each traffic matrix in the dataset contains uniformly sampled VM-to-VM traffic aggregated over a 1-hour window. We use sampled data; sampling is necessary to scale measurement systems, and, as long as the sampling mechanism is uniform, we expect our clustering algorithm to work just as well as it would have on un-sampled data given that uniform sampling ought to preserve traffic volume relationships between VMs.

Implementation. Customer flow logs are stored in Google's Colossus file system [30]. CloudCluster loads the flow logs into Dremel [43] and uses Dremel's SQL-like queries to *select* data within the aggregation window, *group by* src-dst VM pairs and *aggregate* by volume. CloudCluster runs on a single VM with 128G memory, loads the aggregated result from Dremel into a dataframe, extracts the VM-to-VM traffic matrix, and then runs the algorithm described in §3. Traffic matrices for the projects we evaluate fit comfortably into a single VM.

Methodology. To demonstrate that CloudCluster produces clusters consistent with VMs grouped by location and function, we conduct two experiments on disjoint sets of the fifteen projects in our dataset:

i. Carefully-Named Group. The first experiment uses data belonging to eight projects. These eight projects (called the *Carefully-Named Group*) are different from the other seven projects *because we have information about the location and function of each VM*. For these projects, the customers have carefully named each VM based on function, likely to simplify manageability of the project. For example, VM naming schemes contain strings identifying well-known services (e.g. "redis" [25], "cassandra" [39], or "nginx" [53]). We call these strings *VM labels* (in addition to labels, VM names may contain, for example, instance identifiers). For projects in this group, we show that CloudCluster's clusters, when further sub-grouped by the VM location (the VM's zone, §2) match well with VM groupings by location and VM labels, *i.e.,* functions.

ii. Coarsely-Named Group. The second experiment uses data belonging to the remaining seven projects. For these, we have location information for each VM, but the VM naming scheme does not always indicate the function, or indicates function coarsely (we explain later precisely what this means). For this group of projects, we show that CloudCluster's clusters, when further sub-grouped by the VM location, *do not* match well with VM groupings by location (zone) and VM labels.

We emphasize that the cloud provider will always know a VM's location, but cannot always know the VM's function, since function-based naming is not a requirement of any cloud-service API that we are aware of.

Metrics. We use two standard measures of clustering goodness, *homogeneity* and *completeness* [49]. These are both scalar real-valued metrics in the range $[0,1]$. In the context of VM clustering, *homogeneity* is the fraction of VMs in a same cluster that have the same location and VM label. Conversely, *completeness* is the fraction of VMs that have the same location and VM label that are in a single cluster. These are the analogs of precision and recall used in classification.

4.2 The Carefully-Named Group

The Carefully-Named Group refers to the eight projects where the VM's are carefully named to reflect their function, in addition to the location (zone) information.

High homogeneity and completeness. We cluster the VMs in each of the eight projects in the Carefully-Named Group. As noted earlier, we further sub-group the clusters produced by location, *i.e.,* zone. Table 1's third and fourth columns show the homogeneity and completeness for all the projects in this group. Across these projects, CloudCluster has high homogeneity: all projects have a homogeneity of 0.92 or higher, and for six of them the score is higher than 0.96. Completeness scores are also high: all projects have a completeness of 0.94 or higher. High completeness and homogeneity scores indicate good matching in the clustering results, and substantiate our central assertion: that CloudCluster's clusters, when augmented with zone information, match VMs grouped by location and function.

What values of homogeneity and completeness are acceptable? Recall that these measures are the equivalent of precision and recall (respectively), for which acceptable thresholds depend upon the specific use case. Similarly, acceptable values of homogeneity and completeness depend upon what clustering is used for; we discuss this in §5.5. Also, as with precision and recall, we can trade-off homogeneity for completeness and vice versa; see §4.4 for an example.

CloudCluster works well for projects at different scales. Projects range in size from 500 VMs to over 10,000 VMs (second column of Table 1). The number of clusters (third column of Table 1) varies from a handful to around 200. CloudCluster also discovers clusters at different scales. Within project A, some clusters have more than 900 VMs, and some clusters have dozens of VMs or sometimes one. Moreover, CloudCluster can handle projects with varying functional and geographical diversity. Projects A, B, E and F each run more than 20 different kinds of software and span across a number of zones across the globe. This also explains why they have many clusters (recall that clusters are distinguished both by function and location). Projects C and D are functionally homogeneous and scoped to a single continent; and projects G and H are moderately functionally diverse (5-6 different types of functions) but scoped within North America. This explains why C, D, G and H have only a handful of clusters.

Why location is important. Clustering groups VMs with a similar traffic pattern. Our hypothesis was that VMs that perform the same function will have similar traffic patterns. However, consider two VMs that perform the same function, but are located in zones on different continents. Although their traffic distributions to other VMs will be similar, they will likely send traffic to completely different sets of VMs (*e.g.,* load-balancers, other services) because they are located in different zones. Thus, their *rows* in the traffic matrix will be different, and CloudCluster will be unable to cluster them.

To illustrate the importance of location, for projects in the Carefully-Named Group, we compare completeness and homogeneity scores *without using location information*. This means that we no longer sub-group CloudCluster's clusters by location (zone) *and* we do not use the location information when computing homogeneity and completeness scores. Table 1's 5th and 6th columns show that, in this case, while homogeneity is reasonably high (all VMs in a cluster tend to have the same label, *i.e., function*), completeness drops significantly for about half of the projects (*i.e.,* VMs with the same label do not all fall into the same cluster).

Label and cluster conflicts. Prior work has also explored a different way to characterize the quality of clustering [46]. Consider any pair of VMs. These VMs can either be in different clusters, or they can be in the same cluster. If clustering is perfect, then (a) if the VMs belong to different clusters, they must have different labels,[7] and (b) if they belong to the same cluster, they must have the same labels. Conversely, clustering can fail in two ways: (a) the VMs belong to different clusters, but they have the same label (we call this a ***cluster conflict***, which results in a completeness score lower than 1.0) and (b) the VMs belong to the same cluster, but have different labels (we call this a ***label conflict***, which results in a homogeneity score less than 1.0).

To understand the magnitude of these conflicts, Table 1's 7th and 8th column show the *rate* of cluster and label conflicts in each of our projects in the Carefully-Named Group. Following [46], we compute the rate of cluster conflicts as the fraction of all VM pairs in different clusters that have the same label, and the rate of label conflicts as the fraction of VM pairs in each cluster that have different labels. Table 1's 7th and 8th column show that these numbers are negligibly small (less than half a percent) across all projects, and represents another way of viewing the results in Table 1's 3rd and 4th column. For instance, for project *D*, label conflicts are zero, so its homogeneity is 1. Similarly, project *C* has high homogeneity because its label conflict rate is very small and

[7] More precisely, different labels or locations; we use labels to simplify the explanation

			CloudCluster w/ location info		CloudCluster w/o location info		Percentage of Conflict	
Project	#VM	#Cluster	Homogeneity	Completeness	Homogeneity	Completeness	Cluster conflict	Label conflict
A	10000+	72	0.984	0.966	0.942	0.312	0.020%	0.197%
B	5000+	206	0.919	0.951	0.888	0.706	0.046%	0.532%
C	500+	4	0.999	0.964	0.989	0.865	0.001%	0.614%
D	500+	3	1.000	0.938	0.989	0.831	0.000%	0.427%
E	5000+	60	0.966	0.940	0.929	0.901	0.212%	0.386%
F	5000+	177	0.937	0.949	0.873	0.929	0.127%	0.188%
G	5000+	8	0.996	0.971	0.992	0.971	0.001%	0.169%
H	1000+	6	0.997	0.997	0.997	0.997	0.000%	0.028%

Table 1: Homogeneity and completeness score (with and without location information) and percentage of conflict for projects in the Carefully-Named Group

project *H* has highest completeness and the lowest cluster conflict rate. (As an aside, these rates are defined on VM-pairs, so the actual rates cannot directly be matched to imperfections in homogeneity and completeness.).

Why CloudCluster is less than perfect. Given the diversity of project in the Carefully-Named Group, CloudCluster's agreement with customer-provided functional groupings is impressive. However, it is less than perfect for several reasons.

Feature scaling compresses the range of each feature, which changes the relative feature distances of all VMs. Dimensionality reduction step removes information from all feature vectors. TruncatedSVD [34] only keeps the information of the specified number of dimensions. Merging might also induce errors. We use an approach similar to MeanShift's postprocessing [29] in that we merge clusters that are similar to each others by a specified distance. Even though we choose a rather aggressive merging threshold, it is still possible to merge two groups of VMs that have different traffic patterns. Similarly, the merging threshold can also be so high that it breaks other sub-clusters which should be merged.

Finally, some of these label and cluster conflicts can be caused by inconsistently assigned VM labels. For instance, in project *F*, which has high homogeneity and completeness, we found some VMs labeled `default` or `pool`. Table 1 suggests that mis-naming of VMs in our Carefully-Named Group is small. As we discuss in §4.3, CloudCluster works less well for our Coarsely-Named Group because VM naming does not reflect function (*i.e.,* from the perspective of this analysis, the VMs are mis-labeled). Equally important, the non-zero rate of label and cluster conflicts suggests that, even for well-named projects, labels may be mis-configured; in §5.3 we discuss techniques to detect such misconfigurations.

4.3 Coarsely-Named Group

In §4.2, we showed that (a) CloudCluster has high homogeneity and completeness for projects where labels reflect functions, and VM location is taken into account, and (b) it has high homogeneity and low completeness when VM location is omitted. The Coarsely-Named Group contains projects where VM labels do not reflect function well. We expect CloudCluster to perform poorly in this case; we use this group of projects to rule out the possibility of other factors contributing to high completeness and homogeneity for the Carefully-Named Group.

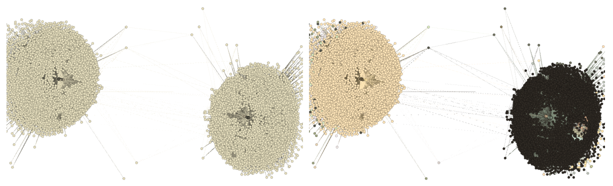
(a) Cluster by customer labels **(b)** Predicted clusters

Figure 4: Same set of VMs clustered by (a) the customer label and (b) our algorithm. This figure shows VMs with generic labels like "default", "pool" or "farm".

As Table 2 shows, for projects I through O (which have comparable functional, size and spatial diversity as projects A-H), homogeneity is high, but completeness is low (for most projects in the 0.6-0.8 range, but in one case as low as 0.25). These results indicate that, in these projects VM labeling has the following property: if two VMs are similar in traffic characteristics, they are likely to have the same labels, but if they are different by traffic characteristics (so are in different clusters) they may still have the same labels. In other words, labels in these projects lack functional *specificity*.

Labeling specificity. Table 2 suggests that, if VM grouping by labels and location should match well with CloudCluster, labels have to have functional *specificity*. Some projects have less specific (or generic) functional labels, as we illustrate in the following examples.

Project	Homogeneity	Completeness
I	0.988	0.825
J	0.988	0.740
K	0.952	0.782
L	0.936	0.786
M	0.978	0.250
N	0.983	0.758
O	0.993	0.603

Table 2: Homogeneity and completeness scores for projects in the Coarsely-Named Group.

Figure 4 shows a group of VMs where the VM labels are generic (`default`, `pool`, or `farm`). In Figure 4(a), the dots are colored by the VM group they belong to by customer label. In this case, all VMs belong to the same group because they are assigned a generic label, so in Figure 4(a) they all have the same color (green). However, if we closely examine the functional structure of this project, we see two distinct groups densely connected internally, but sparsely connected externally. Figure 4(b) shows that CloudCluster is able to correctly distinguish between the two groups (yellow and black).

Sometimes, making labels specific enough requires careful thought. Figure 5 illustrates a case where *sharding* may require generic labels. In Figure 5(a), the customer has labeled all nodes within the circled ellipses as "loadbalancer". However, from Figure 5(b), we observe that there is an internal structure to these load-balancers. They can be further separated into three groups where each has a distinctive connection pattern. The clustering algorithm captures this difference and puts them into different clusters.

Customers are not required to provide specific functional labels, but these examples give some insight into how CloudCluster's clustering might differ from a customer's notion of functional labels. At the same time, many projects *do* label VMs by function. For these, being able to identify generic customer labeling (or, more generally, mis-labeling) can help identify configuration errors (see §5).

4.4 Impact of Design Choices

We now quantify the importance of various design choices.

Dimensionality reduction. Dimensionality reduction reduces the runtime of the pipeline and improves the clustering accuracy. In the absence of dimensionality reduction, the distance metric can be unreliable for data with high-dimensional feature spaces [60]. Moreover, distance computation does not scale well for projects with more than 10,000 VMs; on our largest project, without dimensionality reduction, the pipeline takes more than ***40 minutes*** to finish. With dimensionality reduction, CloudCluster's pipeline completed in ***150 seconds*** for the same project. CloudCluster is not latency-sensitive, but lower computational complexity is important for reducing the overhead or cost of executing CloudCluster's algorithms on the cloud.

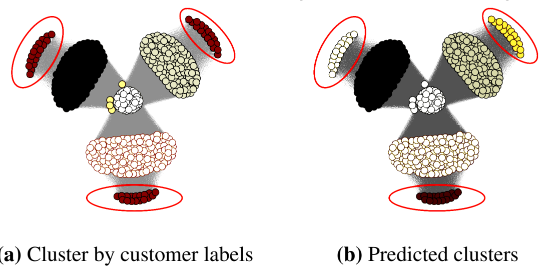
(a) Cluster by customer labels **(b)** Predicted clusters

Figure 5: Same set of VMs clustered by (a) the customer and (b) our algorithm. This figure shows that customer-defined VM groups contains customer specific sharding.

Feature scaling. Feature scaling approaches influence CloudCluster's performance. If we disable feature scaling, CloudCluster produces lower homogeneity (0.812) and completeness (0.822) scores for project A, our largest project. By contrast, log-scaling is able to achieve 0.984 homogeneity and 0.966 completeness. Using other forms of feature scaling result in slightly lower homogeneity and completeness scores. Figure 3 shows that using standardized and minmax scaling reduces both homogeneity and completeness.

Hierarchical clustering. To validate our choice of our clustering algorithm, we compare with other plausible clustering approaches. We used OPTICS [22], Affinity Propagation [33] and MeanShift [29] to produce another set of clusters, and compared the clusters with project A's labels. To be fair to these alternative clustering approaches, we performed the same feature scaling and dimensionality reduction before feeding the data into the algorithms. We used default parameters for these other clustering algorithms. We show that OPTICS results in significantly lower homogeneity (0.471) and completeness (0.163). Affinity Propagation produces slightly better homogeneity (0.994) by producing more than 3000 clusters in the project with 10,000+ VMs. This comes at the cost of a significantly lower completeness (0.559). Conversely, MeanShift achieves a higher completeness score (0.989) by having giant, noisy clusters, but with lower homogeneity (0.701).

Merging. Without merging, we achieve a slightly better homogeneity score (0.996), but a much worse completeness score (0.547). The high homogeneity is due to the fact that we produce clusters with small internal variation in the process of hierarchical clustering. The requirement of small internal variation divides VMs with similar traffic patterns into different clusters and lowers the completeness score. Merging combines similar clusters to significantly improve completeness for project A (from 0.547 to 0.966) at the expense of a small drop in homogeneity (from 0.996 to 0.984). This benefit of merging is evident across all projects in the Carefully-Named Group (Table 4). In some cases, the improvements in completeness are even more dramatic, increasing from 0.442 to 0.996 for project H.

	Homog.	**Compl.**
CloudCluster	0.984	0.966
Without feature scaling	0.812	0.822
Feature scaling: standardizer	0.939	0.953
Feature scaling: minmax scaler	0.974	0.948
Clustering: OPTICS [22]	0.471	0.163
Clustering: Affinity Prop [33]	0.994	0.559
Clustering: MeanShift [29]	0.701	0.989
Disable merging	0.996	0.547

Table 3: Compares the impact of different design choice on project A's result

	Without Merging		With Merging	
	Homog.	**Compl.**	**Homog.**	**Compl.**
A	0.996	0.547	0.984	0.966
B	0.932	0.896	0.919	0.950
C	0.998	0.522	0.998	0.963
D	1.000	0.442	1.000	0.938
E	0.985	0.698	0.965	0.940
F	0.978	0.781	0.937	0.949
G	0.996	0.386	0.996	0.971
H	0.997	0.442	0.996	0.996

Table 4: Effect of merging on the Carefully-Named Group group

5 CloudCluster For Project Management

In this section, we describe several proof-of-concept ways in which the output of CloudCluster can help cloud providers provide their customers with actionable insights about the configuration and management of their services.

5.1 Reconfiguration to Reduce Cost

Cloud providers often price traffic in multiple tiers: traffic within the same cloud zone typically costs less than traffic between VMs from different zones, regions or continents. Customers engineer VM placements to reduce cost while balancing availability and proximity to customers. CloudCluster can help identify opportunities for reconfiguring VM placements to reduce costs. In this section, we discuss three examples that illustrate these opportunities; future work can develop systematic tools to discover such opportunities.

Figure 6 shows the distribution of traffic to other zones from VMs of project A belonging to VM label L. CloudCluster detects that VMs with this label belong to two different clusters: one which sends traffic more-or-less uniformly to VMs in 8 different zones (first cluster in Figure 6) and the other which sends over 80% of its traffic to a single zone (second cluster). A customer can potentially reconfigure the placement of VMs of the latter cluster to avoid inter-zone traffic. Although the traffic skew is visible across all VMs (so the customer might have been able to detect it using the VM label), CloudCluster is able to identify the precise set of VMs to re-configure.

Using CloudCluster, the cloud provider can determine the volume of intra-cluster and inter-cluster traffic, and determine how much of this traffic crosses zone, region, or continent boundaries. Using this, it can estimate cost savings resulting from reconfigured VM placements. Figure 7 and Figure 8 illustrate cost savings from reconfiguration in two cases.

The first case is a cluster C from project A of VMs located in different zones of a single cloud region. Almost 90% of traffic in C is intra-zone, which is free or relatively cheap on most cloud providers ([15], [8], [11]). However, the remaining traffic traverses continental boundaries, and accounts for a significant fraction of total cost charged to C. If the customer were to provision a small cluster in the zone on the other continent where the traffic comes from, it can reduce the cost attributable to this cluster by 41.2% (Figure 7).

The second case is a cluster C' of project A whose traffic is largely inter-region (intra-zone traffic is $< 0.1\%$). 92.3% of egress traffic from C' goes to zones in another region R, and 95.9% of its ingress traffic comes from VMs in a single zone in R. Moving VMs in C' to R (an *egress-favored* placement) reduces cost by 21.1%, while moving these VMs to the zone in R from which they receive most traffic (an *ingress-favored* placement) reduces cost by 15.1% (Figure 8).

These are simplified examples; in practice, tools that suggest re-configuration of VM placements will need to consider other customer objectives such as availability and latency. We have left development of such tools to future work, but CloudCluster's clustering can be a valuable input to such tools.

5.2 Anomaly Detection

Large-scale cloud project outages are sometimes caused by rapid increases in service workload, management operations by the customer (incorrect service configuration), by the provider (VM migration), or failures in the provider's network. These are often accompanied by sudden shifts in traffic between VMs in the service or traffic to and from external entities (*e.g.,* customers of the cloud service). Such traffic shifts may often be visible in the aggregate traffic between clusters. Because our clusters correspond to functionally homogeneous VMs, if one VM in cluster A starts communicating more with a VM in cluster B, it is likely that all other VMs in A will also start communicating more with VMs in B.

In this section, we present a preliminary evaluation of an anomaly detector that tracks significant deviations in aggregate inter-cluster traffic on each link in the inter-cluster graph (Figure 1(e)). Such a detector can also help localize anomalies, as we discuss below. In practice, we expect our anomaly detector to complement other approaches used by cloud providers.

The anomaly detector works as follows. For each edge in the inter-cluster graph (an edge exists between two clusters if their VMs communicate), it tracks at each aggregation window, the total volume in bytes, the total flow count, and the number of communicating VM pairs between each pair of clusters. When, for a given edge, any of these quantities deviates significantly from a windowed moving median [57], we flag that deviation as an anomaly (we omit the details for brevity). Because the inter-cluster graph is sparser than the inter-VM graph (*e.g.,* Figure 1(a)), we are able to scalably identify correlated anomalies, where two or more communicating cluster pairs exhibit anomalous traffic at the same time.

Trace Analysis and Results. To quantify the effectiveness of this detector, we identified 25 time windows across different projects where either (a) an internal anomaly detector that uses a different methodology flagged anomalous traffic in the project during the corresponding time window (17 instances) or (b) the customer filed a trouble ticket (8 instances).

We then ran the CloudCluster-based anomaly detector on these 25 time windows, and, in each case, were able to con-

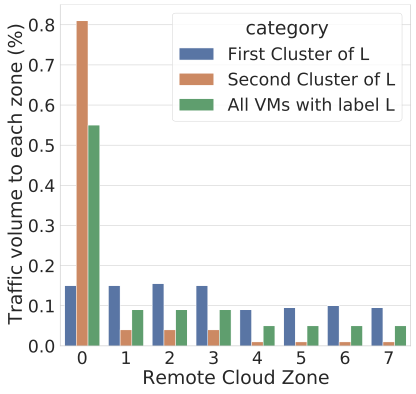

Figure 6: CloudCluster finds VMs with same label but different traffic patterns.

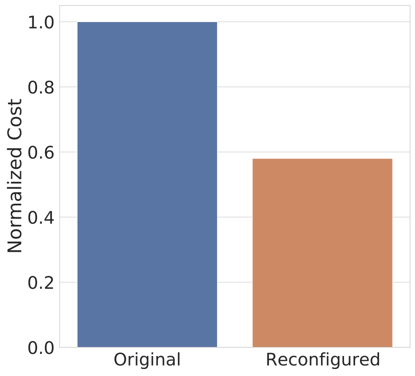

Figure 7: Original vs. Reconfigured placement cost

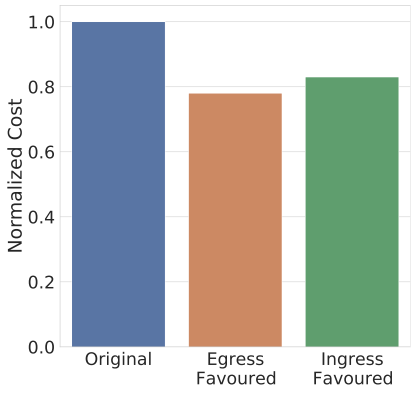

Figure 8: Original vs. Egress-favored vs. Ingress-favored placement cost

firm the existence of the anomaly[8], and also to pinpoint which cluster-pairs were responsible for the deviations. We have not analyzed false positive rates; since we started with known anomalies flagged by other systems. For the 8 customer-reported incidents, our detector was able to correctly identify the offending cluster-pairs (as determined in the post-mortem reports). We identified two broad classes of anomalies: *traffic shifts* and *structural changes*. In the first class, the inter-cluster graph does not change, but traffic on some subset of links changes significantly. In the second, new nodes and or edges are added to the graph or nodes and edges are removed. Of the 25, three were traffic shifts and the rest were structural changes.[9]

Our detector is fast: the maximum processing latency to compute the deviation scores, across all projects, was 92.3 milliseconds per time window.

The following paragraphs briefly describe some qualitatively different anomalies that we were able to detect; §A contains a more detailed description.

Correlated traffic shift due to peering router failure. This anomaly was reported by the network operator in reaction to a peering router failure. Our detector observed that a cluster in the region nearest the peering router saw a sudden reduction in flow and byte counts. Concurrently, a cluster in another region, (which, from label names, we determined was functionally identical to the first cluster), saw an increase in traffic. We suspect that the peering router failure diverted external traffic to enter the cloud provider's network at a different location, but don't have the instrumentation to confirm this.

Structural change due to VM migration. This anomaly was reported by the internal anomaly detector. Our CloudCluster-based anomaly detector identified a sequence of structural changes across successive aggregation windows. Recall that clusters are distinguished both by function and location (§4.2). In this case, the structural changes were caused by a migration of VMs from one server to another due to scheduler-driven evictions. The migration was spread out over multiple aggregation windows, so our detector noticed a sequence of structural changes corresponding to progressive migration of VMs from one server to another.

Structural change due to project reconfiguration. Our internal anomaly detector flagged anomalous traffic for a cloud provider. The CloudCluster-based anomaly detector identified a structural change: two clusters were removed from the graph and one was added. The two initial clusters corresponded to a singleton cluster containing a leader VM and another containing 120 worker VMs. The new cluster contained the 121 VMs, encompassing both the leader and the workers. In this case, it turns out that the customer had initiated the structural change, decommissioning the older VMs in favor of another set of VMs as part of an upgrade.

5.3 Potential Label Misconfiguration

As discussed in §4.2, several customers label VMs with precise function names and location information. We conjecture that they use this to simplify project management. These VM labels are often configured, either by hand or by a script. *Label misconfigurations* can occur, and CloudCluster can be used to detect the *likely* candidates. When a label misconfiguration occurs in a project whose VMs appear to be named by function and location, *i.e.,* when the project has a high homogeneity and completeness, it manifests either as a label conflict or a cluster conflict (§4.2).

Cluster Conflict. A cluster conflict occurs when VMs belong to different clusters, but have the same labels. Such a conflict can either result from a misconfigured label, or from a clustering error. To distinguish between those two cases, we use a technique inspired by prior work in clustering on *silhouette analysis* [50], which attempts to measure the intrinsic performance of clustering. This analysis assigns each item (or VM, in our context) a score in the range $[-1,1]$ that measures how similar the VM is to its own cluster, compared to other

[8] A more detailed analysis of the detector performance, and comparisons with other detection techniques, is beyond the scope of this paper.

[9] Some of these structural changes or traffic shifts might be intentional, even though our approach flags them as (statistical) anomalies.

clusters.

We modified this idea to derive a metric that quantifies whether a customer label is too generic (*i.e.,* spans multiple clusters) or too specific. Let V_l be the set of VMs that has a customer-defined label l, but CloudCluster splits it up into n clusters $\{C_1, C_2, ... C_n\}$. Let $\bar{c_l}$ be the centroid, in feature space, of the traffic features of all VMs in V_l. Let $\bar{c_i}$ be the centroid of the traffic features of all VMs in C_i (the C_is might contain VMs not in V_l). For each cluster C_i, let a_i be the average distance of each VM in this cluster to $\bar{c_l}$ and b_i be the average distance of each VM in this cluster to $\bar{c_i}$. Then, consider the following metric: $ms(i) = \frac{b(i)-a(i)}{max(a(i),b(i))}$.

Intuitively, if $ms(i) < 0$, each VM in the cluster is closer to the cluster center than to the label's center, so the labeling is too generic. Conversely, if $ms(i) > 0$, then the label is too specific. Either way, this indicates a mismatch between clustering and customer-provided labeling, which can be used in some cases to identify potential mis-labeled VMs.

To detect mis-labeling VMs using this technique, we apply the following algorithm. Without loss of generality, assume that C_1 has the largest number of VMs with label l. For all $i > 1$, if $ms(i) < \psi$ (a conservative threshold < 0, we use -0.5), we mark all VMs in C_i with label l as mis-named.

The output of this analysis is a list of *potentially* (we use this term to indicate that, ultimately, any such mis-labeling would have be be verified by a customer, since the customer understands the *intent* behind the naming) mis-labeled VMs that cause cluster conflicts.

Table 5 lists the fraction of potentially mis-labeled VMs for four of our projects. These four projects belonged to a customer who gave us feedback on our clustering results. The fraction of mis-labeled VMs range from negligible amounts (*e.g.,* project H has 0.1% mis-named VMs) to a few percent (for projects A, E and F). For these projects, we were able to verify with the customer that our identification of mis-labeled VMs was accurate. In these cases, the customer had changed the functions in some VMs but forgot to update the VM labels.

Label conflicts. Mis-labeling can also cause label conflicts: different labels within the same cluster. Table 5 also shows the rate of occurrence of these. They happen less frequently, and often fall into two categories. VMs labeled generically such as "default" fall into the same cluster as VMs with more specific labels (*e.g.,* "app-server"). A second cause of mis-labeling is inconsistent hyphenation (*e.g.,* "appserver" vs. "app-server"), or inconsistent abbreviations (*e.g.,* using "es" instead of "east"). We identified examples in the second category using manual inspection; future work can automate the detection of mis-labeling in this category using edit-distance based string similarity analysis [55].

5.4 Potentially Mis-provisioned VMs

When configuring a VM, project owners can *provision* VM resources by specifying the *machine type* for each VM. Machine types determine the capacity of the VM instances in terms of CPU cores, memory and egress network bandwidth [35]. Different machine types are priced differently, so over-provisioning a VM can have cost implications. Mis-provisioning can also impact performance: under-provisioned VMs can result in stragglers, causing services to violate their latency SLOs.

Project	**Cluster Conflict** (%)	**Label Conflict** (%)	**Mis-provisioning Rate** (%)
A	4.62	0	1.59
E	5.75	3.15	0
F	7.26	0.10	0.80
H	0.09	0.04	0.38

Table 5: The percentage of VMs that are mis-labeled in each project (§5.3), the rate of misprovisioning (§5.4).

CloudCluster can identify mis-provisioned VMs by determining outlier machine types in a cluster. Since CloudCluster's clusters identify VMs performing a similar function, if most VMs in a cluster are of machine type a, but a small number are of machine type b, we can identify the latter set as mis-labeled VMs. In determining the rate of mis-labeling, we must filter out mis-labeled VMs. To be more robust to clustering errors, we flag a VM as mis-labeled if it does not lie at the edge of the cluster (as determined by distance from the cluster centroid in feature space).

Table 5 shows the rate of mis-provisioning in 4 of the projects in the Carefully-Named Group. A small number, 1%, appear to be mis-provisioned. We say "appear to be" because the operator cannot know the intent of the customer; they may have deliberately provisioned these machines differently to run additional tasks (*e.g.,* compute bound jobs whose footprint is not visible in the VM-to-VM traffic matrix). Any mis-provisioning will ultimately have to be verified as such by manual inspection by the customer.

5.5 Discussion

In §4.2, we said that acceptable values of homogeneity and completeness depend upon what clustering is used for. We conclude this section with a brief qualitative discussion of this issue, leaving quantitative analysis to future work.

We have described two types of use cases in this section. Reconfiguration and anomaly detection are based upon the inter-cluster graph abstraction, and specifically upon inter-cluster traffic volumes. For these cases, if *most* VMs (*e.g.,* 90%) in a cluster are functionally similar, the reconfiguration decision, or the anomaly detection is likely to be correct.

Detecting mis-labeled or mis-provisioned VMs requires comparing attributes of VMs within a cluster. This can be more susceptible to false positives and false negatives, unless homogeneity and completeness are very high. Because a cloud provider cannot always know the homogeneity and completeness a priori, using clustering for these tasks requires additional filtering steps to minimize false positives. For mis-provisioned VMs, we filter candidates at the edge of the cluster (§5.4). For mis-labeling, we use silhouette analysis (§5.3).

6 Related Work

Inferring Structure from Traffic. Complementary to CloudCluster, others have explored inferring host behavior and distributed system properties from network traffic. The closest prior work [58] groups Internet hosts within each IP prefix by traffic similarity, and explores how this can be used to detect malicious behavior. Other work has modeled host-to-host communication as a graph to understand properties of inter-host communication [13, 14, 36], to infer botnet structure [45], or the logical structure of enterprise networks [16]. The body of work on tracing in distributed systems seeks to infer the causal structure as well as other properties of distributed systems from RPC traces to aid performance debugging (*e.g.,* [19,47,52]). Other work has used traffic to infer specific characteristics of VMs in cloud settings: strongly connected groups of VMs as candidates for migration [26], or compromised VMs [23]. Some of these use clustering [26,58], but do not consider scale and robustness to range of traffic volumes.

Data Clustering. Clustering is a mature area of research, with many established techniques such as K-Means [41], DBSCAN [32], OPTICS [22], AffinityPropogations [33], Hierarchical Clustering [48], *etc.* That clustering is susceptible to the curse of dimensionality is well-known [60]. Clustering in high dimensions has been explored extensively either by: (a) using heuristics to determine attributes of sub-spaces (*e.g.,* CLIQUE [18] or SUBCLU [37]) or (b) designing special distance measures (*e.g.,* projected clustering, as in PreDeCon [24] or PROCLUS [17]). In contrast, CloudCluster explicitly reduces the dimension of the VM-to-VM traffic to the point where conventional clustering techniques and similarity measures are applicable.

Cloud Monitoring and Workload Characterization. Tangentially relevant prior work has used CPU and memory utilization traces to infer properties of VMs [31, 38, 42].

7 Conclusion

CloudCluster performs clustering on the VM-to-VM traffic of cloud projects and yields the functional structure of the cloud service. It overcomes the challenges imposed by scale (cloud services contain tens of thousands of VMs), by orders-of-magnitude variability in traffic volume and measurement noise, and by the lack of prior knowledge of the cloud projects (for number of clusters). The output of CloudCluster can help detect potentially mis-provisioned or mis-labeled VMs, identify opportunities to reduce cost, and detect anomalies.

Future work. Several directions of future work remain, including: identifying the frequency at which to apply CloudCluster to projects; incrementally adjusting clusters when VMs leave or join; supporting traffic to cloud native services such as storage; exploring better methods for determining the cluster merging threshold; more thoroughly evaluating the accuracy of cost reconfiguration, anomaly detection, or misconfiguration determination, and comparing their performance against other alternatives; determining whether additional information from customers, obtained with their consent, can improve the quality of the resulting functional structure.

References

[1] 6.3. preprocessing data. https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing.

[2] Azure monitor overview - azure monitor. https://docs.microsoft.com/en-us/azure/azure-monitor/overview.

[3] Cloud monitoring | google cloud. https://cloud.google.com/monitoring.

[4] Creating and managing projects. https://cloud.google.com/resource-manager/docs/creating-managing-projects.

[5] Google cloud storage. https://cloud.google.com/storage.

[6] Google vpc. https://cloud.google.com/vpc/docs.

[7] Network and subnetwork terminology. https://cloud.google.com/vpc/docs/vpc#subnets_vs_subnetworks.

[8] Network pricing|compute engine documentation|google cloud. https://cloud.google.com/compute/network-pricing.

[9] Regions and zones | compute engine documentation | google cloud. https://cloud.google.com/compute/docs/regions-zones.

[10] Using vpc flow logs. https://cloud.google.com/vpc/docs/using-flow-logs.

[11] Virtual network pricing: Microsoft azure. https://azure.microsoft.com/en-us/pricing/details/virtual-network/.

[12] Clouds project cloudwatch. https://aws.amazon.com/cloudwatch/, 2000.

[13] Blinc. *ACM SIGCOMM Computer Communication Review*, 35(4):229–240, 2005.

[14] Network monitoring using traffic dispersion graphs (TDGs). *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, (c):315–320, 2007.

[15] Aws site-to-site vpn and accelerated site-to-site vpn connection pricing. https://aws.amazon.com/vpn/pricing/, 2020.

[16] Role classification of hosts within enterprise networks based on connection patterns. *Proceedings of the General Track: 2003 USENIX Annual Technical Conference*, pages 15–28, 2020.

[17] Charu C Aggarwal, Joel L Wolf, Philip S Yu, Cecilia Procopiuc, and Jong Soo Park. Fast algorithms for projected clustering. *ACM SIGMoD Record*, 28(2):61–72, 1999.

[18] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data. *Data Mining and Knowledge Discovery*, 11(1):5–33, 2005.

[19] Marcos K. Aguilera, Jeffrey C. Mogul, Janet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP ’03, page 74–89, New York, NY, USA, 2003. Association for Computing Machinery.

[20] Muhammad Amjad, Vishal Misra, Devavrat Shah, and Dennis Shen. Mrsc: Multi-dimensional robust synthetic control. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(2), June 2019.

[21] Muhammad Amjad, Devavrat Shah, and Dennis Shen. Robust synthetic control. *Journal of Machine Learning Research*, 19(22):1–51, 2018.

[22] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod record*, volume 28, pages 49–60. ACM, 1999.

[23] Behnaz Arzani, Selim Ciraci, Stefan Saroiu, Alec Wolman, Jack Stokes, Geoff Outhred, and Lechao Diwu. Privateeye: Scalable and privacy-preserving compromise detection in the cloud. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 797–815, Santa Clara, CA, February 2020. USENIX Association.

[24] Christian Bohm, K Railing, H-P Kriegel, and Peer Kroger. Density connected clustering with local subspace preferences. In *Fourth IEEE International Conference on Data Mining (ICDM’04)*, pages 27–34. IEEE, 2004.

[25] Josiah L. Carlson. *Redis in Action*. Manning Publications Co., USA, 2013.

[26] Marco Cello, Kang Xi, Jonathan H Chao, and Mario Marchese. Traffic-aware clustering and vm migration in distributed data center. In *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing*, pages 41–42, 2014.

[27] Sourav Chatterjee. Matrix estimation by universal singular value thresholding. *The Annals of Statistics*, 43(1):177–214, Feb 2015.

[28] Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC ’15, page 163–172, New York, NY, USA, 2015. Association for Computing Machinery.

[29] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

[30] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Dale Woodford, Yasushi Saito, Christopher Taylor, Michal Szymaniak, and Ruth Wang. Spanner: Google’s globally-distributed database. In *OSDI*, 2012.

[31] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP ’17, page 153–167, New York, NY, USA, 2017. Association for Computing Machinery.

[32] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD’96, page 226–231. AAAI Press, 1996.

[33] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[34] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, 2009.

[35] Google Inc. Machine types | compute engine documentation | google cloud. https://cloud.google.com/compute/docs/machine-types.

[36] Yu Jin, Esam Sharafuddin, and Zhi-Li Zhang. Unveiling core network-wide communication patterns through application traffic activity graph decomposition. *SIGMETRICS Perform. Eval. Rev.*, 37(1):49–60, June 2009.

[37] Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. Density-connected subspace clustering for high-dimensional data. In *Proceedings of the 2004 SIAM international conference on data mining*, pages 246–256. SIAM, 2004.

[38] Arijit Khan, Xifeng Yan, Shu Tao, and Nikos Anerousis. Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium*, pages 1287–1294. IEEE, 2012.

[39] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.

[40] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.

[41] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[42] Shruti Mahambre, Purushottam Kulkarni, Umesh Bellur, Girish Chafle, and Deepak Deshpande. Workload characterization for capacity planning and performance management in iaas cloud. In *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 1–7. IEEE, 2012.

[43] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of web-scale datasets. *Proc. VLDB Endow.*, 3(1–2):330–339, September 2010.

[44] Fionn Murtagh and Pierre Legendre. Ward's hierarchical clustering method: clustering criterion and agglomerative algorithm. *arXiv preprint arXiv:1111.6285*, 2011.

[45] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, page 7, USA, 2010. USENIX Association.

[46] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[47] Patrick Reynolds, Janet L. Wiener, Jeffrey C. Mogul, Marcos K. Aguilera, and Amin Vahdat. WAP5: Black-box performance debugging for wide-area systems. *Proceedings of the 15th International Conference on World Wide Web*, pages 347–356, 2006.

[48] Lior Rokach and Oded Maimon. *Clustering Methods*, pages 321–352. Springer US, Boston, MA, 2005.

[49] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 410–420, 2007.

[50] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65, 1987.

[51] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, pages 166–171. IEEE, 2011.

[52] Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Google, Inc., 2010.

[53] Igor Sysoev et al. Nginx. *Inc.,"nginx," https://www. nginx. com*, 2004.

[54] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv e-prints*, page arXiv:1907.10121, Jul 2019.

[55] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[56] Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[57] Eric W Weisstein. Moving median. https://mathworld.wolfram.com/MovingMedian.html.

[58] Kuai Xu, Feng Wang, and Lin Gu. Network-aware behavior clustering of internet end hosts. In *2011 Proceedings IEEE INFOCOM*, pages 2078–2086. IEEE, 2011.

[59] Y. Zhang and Z. Ge. Finding critical traffic matrices. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 188–197, 2005.

[60] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(5):363–387, 2012.

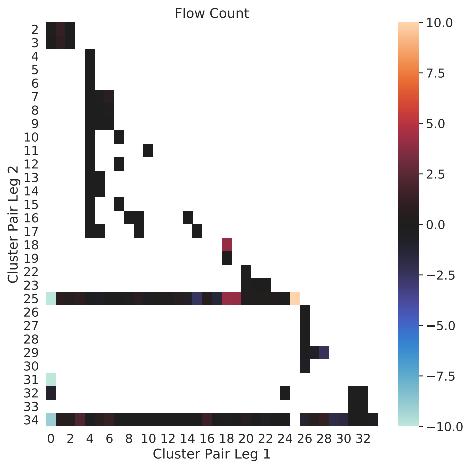

Figure 9: Cluster to Cluster flow count deviation scores

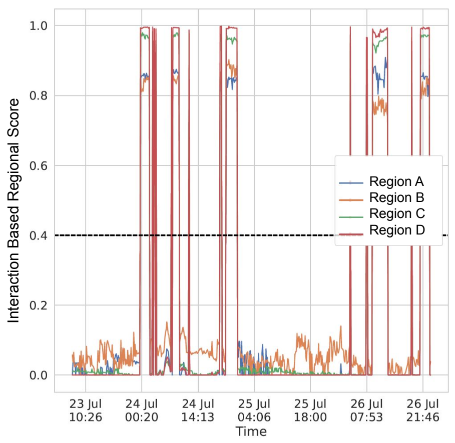

Figure 10: Regional anomaly scores during VM eviction/migration

Appendix

A Detailed Explanation of Anomalies

Correlated traffic shift due to peering router failure. This anomaly was reported by the network operator in reaction to a peering router failure. Our detector observed that a cluster in the region nearest the peering router saw a sudden reduction in flow and byte counts. Concurrently, a cluster in another region, (which, from label names, we determined was functionally identical to the first cluster), saw an increase in traffic. We suspect that the peering router failure diverted external traffic to enter the cloud provider's network at a different location, but don't have the instrumentation to confirm this. Figure 9 depicts the cluster to cluster deviations scores expressed in terms of median absolute deviations (MAD) [40] with the sign indicating whether the upper (e.g. positive) or lower (e.g. negative) anomaly detection boundary was crossed. The x-axis and y-axis represent either the source or destination of the cluster pairs whose interactions are studied. The values of the heat map show the MAD deviations observed between the interacting cluster pair and is assigned a color based on the color map where the color black indicates no deviation and warmer (calmer) colors represent anomalous traffic characteristic deviating above (below) the median traffic volume. In the observed traffic shift, we were quickly able to identify the cluster pairs impacted by the anomaly (e.g. $|dev| > 5$ in Figure 9) that appear as the red, orange, and blue cells in the heatmap, filter away clusters not impacted by the anomaly that appear as black cells in the heatmap, and provide a project wide summary of the anomaly.

Structural change due to VM migrations. This anomaly was reported by the internal anomaly detector. Our CloudCluster-based anomaly detector identified a sequence of structural changes across successive aggregation window. Recall that clusters are distinguished both by function and location (§4.2). In this case, the structural changes were caused by a migration of VMs from one server to another due to scheduler-driven evictions. The migration was spread out over multiple aggregation window, so our detector noticed a sequence of structural changes corresponding to progressive migration of VMs from one server to another. Figure 10, shows the regional-score, an average of all the cluster to cluster deviation scores weighted by the number of VM communication pairs, computed for every region. The recurring structural changes manifest as plateaus and valleys in the regional score observed. The valleys represent the time windows where the new cluster behaviour is learnt and clusters behave as *normal*, while the plateau's illustrate the time windows where there is a migration storm (e.g. significant number of VM migrations).

Load increase. A customer reported, to the cloud provider, a trouble ticket asking to root cause a missed service-level agreement. The CloudCluster-based anomaly detector identified a sudden increase of external traffic to clusters with memcached servers in one of the customer's projects. (CloudCluster models external traffic as a single node in the inter-cluster graph). This was also concluded in the manual postmortem of the event. Similar to Figure 9 where we characterize the deviation in cluster pair interactions, here we observed a drift in interactions against logical clusters (e.g. may not contain VMs such as client IP) representing connections external to the project. Using this, we identified the culprit traffic flows through heavy hitter analysis and their origin, which happens to be a project that auto scaled to keep up with traffic demand and subsequently flooded the memcached projects with requests.

Structural change due to project reconfiguration. Our internal anomaly detector flagged anomalous traffic for a cloud provider. The CloudCluster-based anomaly detector identified a structural change: two clusters were removed from the graph and one was added. The two initial clusters corresponded to a singleton cluster containing a master VM and another containing 120 worker VMs. The new cluster contained the 121 VMs, encompassing both the master and the workers. In this case, it turns out that the customer had initiated the structural

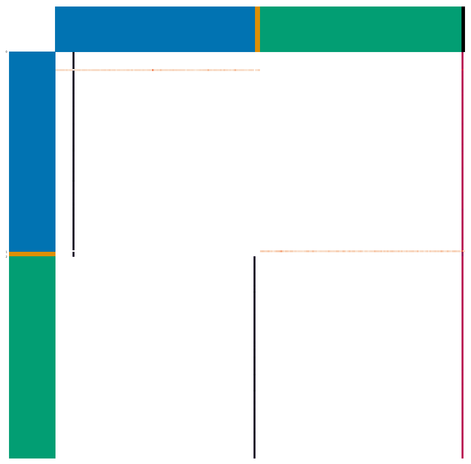

Figure 11: VMxVM Communication Graph (grouped by cluster assignment)

change, decommissioning the older VMs in favor of another set of VMs as part of an upgrade. Figure 11 shows the project communication structure using a VMxVM matrix where the values of the heatmap show the number of bytes sent between VMs log scaled (e.g. white represents no communication and warmer colors depict higher bandwidth consumption). The VMs in the rows and columns are sorted by their cluster assignments wherein they are grouped and identified by the color strip (e.g. cluster id) that appear on the top and left side of the heatmap (e.g. blue, yellow and green). On visualizing the project structure using this heatmap, the aforementioned project re-instantiation evolves in the following manner: a) Initially the top-left sub-structure (e.g. one master, 120 workers) operated devoid of the bottom-right substructure. b) After a downtime where the cluster-to-cluster deviation scores exceed a predefined threshold, we observed the bottom-right structure, which had displaced the other sub-structure. Therefore, by analyzing the traffic patterns, the network-engineer can identify changes to the project structure.