Coriolis: Scalable VM Clustering in Clouds

Daniel Campello*, Carlos Crespo*, Akshat Verma†, Raju Rangaswami*, and Praveen Jayachandran†
*Florida International University, †IBM Research - India

Abstract

The growing popularity of virtualized data centers and clouds has led to virtual machine sprawl, significantly increasing system management costs. We present *Coriolis*, a scalable system that *analyzes* virtual machine images and automatically clusters them based on content and/or semantic similarity. Image similarity analysis can improve in *planning* many management activities (e.g., migration, system administration, VM placement) and reduce their *execution* cost. However, clustering images based on similarity – content or semantic – requires large scale data processing and does not scale well. *Coriolis* uses (i) asymmetric similarity semantics and (ii) a hierarchical clustering approach with a data access requirement that is linear in the number of images. This represents a significant improvement over conventional clustering approaches that incur quadratic complexity and therefore becoming prohibitively expensive in a cloud setting.

1 Introduction

Cloud computing lends a fundamental shift to how businesses view IT, from being capital-intensive to being a commodity that can be acquired on-demand and paid for as per usage. However, the growing popularity of cloud data centers has led to the problem of virtual machine sprawl. Standardization is a key principle that allows cloud providers to provide services on-demand and at a lower cost than what individual IT departments can do. System management costs reduce with standardization of software at all levels: operating systems, middleware, applications, and management tools [1, 13].

We conjecture that classifying (possibly) diverse virtualized servers in a cloud into clusters of similar virtual machines (VMs) can improve the planning of many system management activities. We classify VM similarity into two types – content similarity and semantic similarity. Content similarity refers to data similarity in the raw files that constitute virtual machines. Semantic similarity refers to the similarity in the operating system, middleware, and application software present in two virtual machines. Several management activities can be planned better to reduce their execution cost using analysis of content and/or semantic similarity.

We develop and evaluate *Coriolis*, a framework for clustering images based on any given notion of similarity. Conventional clustering techniques require at least quadratic data access or worse, prohibitive for cloud environments with a large number of VMs. Further, clustering images based on the conventional symmetric notion of similarity leads to a uniform data access pattern; consequently, caching techniques that leverage popularity or locality for optimizing index lookup in deduplication systems [15, 6] are not applicable. *Coriolis* employs a novel tree-based VM clustering algorithm that consumes time that is only linear in the number of images. The algorithm uses an asymmetric notion of similarity to avoid computing all-pairs similarity values and a hierarchical order to introduce popularity in data access.

2 VM Similarity: Types and Applications

The similarity across VMs in enterprise data centers and clouds has been studied extensively in the context of data deduplication [5, 6, 8, 9, 15]. In this section, we discuss both content and semantic similarity and then discuss how such similarity can be utilized for streamlining system management tasks.

2.1 Content Similarity

The classical notion of similarity is that of *content*, whereby a subset of the bytes contained within the images are identical. Identical content can occur either in the form of whole or partial files [11] and techniques to detect similar content have ranged from whole file and fixed size chunking to more sophisticated variable size chunking [8, 15]. Content similarity is useful in minimizing the amount of data that needs to be managed for a task involving a collection of VMs (e.g., VM backup [14] or Virtual Image Library [3]). A recent large-scale study of VM images in a production IaaS cloud investigates such content similarity [7]. This study found that the distribution of *content similarity across images is skewed* and that individual VM images tend to be similar to a small subset of images than to the entire image population leading to clusters of similar images. They also noted that *computing pair-wise similarity is very expensive* and reported results for only 30% of their image collection due to scalability issues.

2.2 Semantic Similarity

Semantic similarity characterizes the similarity of software functionality within images. Examples of semantically similar software include instances of the same application, different versions of the same application, or even different applications that accomplish the same goal (e.g., MySQL and DB2 which both implement database systems and require database expertise to manage). Causes for semantic similarity include standardization of the software stack in modern enterprises and

Use Case	Content	Semantic
Administrator Allocation	×	✓
Troubleshooting	×	✓
VM Placement	✓	✓
Migration	✓	✓

Table 1: Similarity types relevant for each use case

the popularity of specific types of programming models. As identified in previous work, when enterprises are migrated to the cloud, they are adjusted and standardized so that the same set of agents and processes can be used for management services such as backup recovery, security compliance, and patching [13]. Semantic similarity is useful for streamlined system administration, troubleshooting, and management tasks such as grouped scheduling of maintenance and upgrade engineers leading to lower personnel costs. With the growing problem of virtual image sprawl, administrators find it increasingly difficult to keep track of what software is installed on each VM. Automating the detection of VMs with semantically similar software is thus valuable. Unfortunately, the nature of semantic similarity in enterprise and cloud data centers is not well understood.

2.3 Harnessing Image Similarity

We identify four common system management scenarios that can leverage image similarity to reduce data center costs. The most natural use case is allocation of servers to system administrators for routine maintenance. It has been shown that system administrators can be more efficient and manage up to 80% more servers if the servers have a similar software stack [1]. A second use case is troubleshooting system errors during regular updates in data centers. Troubleshooting in data centers is often akin to manual outlier detection where the engineer attempts to identify servers that responded similarly to the update. Once similar servers are identified, the engineer identifies the difference between the failed server and the successful server to fix the issue. Automated clustering of servers based on semantic similarity can aid such identification. Third, placement of VMs to hosts or to management systems often leverage content for efficiency. Images with high semantic similarity are likely to exhibit higher number of duplicate pages in main memory, which can be deduplicated. Similarly, images with higher content similarity can benefit more from deduplication performed at a shared management server (e.g., vSphere [14]).

The final use case is migration of enterprise applications from one data center to another. Migration is performed in batches or waves, where a certain number of images (e.g., 25) are migrated in one weekend [13]. Migrating images with similar content together can reduce migration time using deduplication. Further, images with similar applications can be reconfigured with fewer application experts, reducing migration cost. Identifying image clusters with both high content and semantic similarity and using them to create waves can help reduce both migration time and cost. Table 1 summarizes the type of similarity relevant for all the use cases.

3 Similarity-based VM Clustering

Clustering is a well-studied problem in computer science. While the problem is NP-hard, various heuristics exist with acceptable clustering performance.

3.1 A Representative Clustering Algorithm

k-means is one of the most popular clustering techniques employed in the real world. The algorithm starts with an initial set of *k*-clusters and refines them iteratively. Even though multiple variants of the algorithm exist, they all apply two canonical operations in each iteration:

- *Assignment Step:* Assign each element to the cluster with the closest mean. *Distance* computation is the core internal operation, performed k times for each element. If there are N elements to cluster, this requires kN *Distance* operations.
- *Update step:* Calculate the new mean for each cluster. The core step is a *Merge* operation which computes the average for 2 elements along each of the D dimensions. In each iteration, across the k clusters, $N-1$ merge operations are performed.

The worst case time for *k*-means is exponential in N. For arbitrary set of points in $[0,1]^D$, if each point is independently perturbed by a normal distribution with variance σ^2, then the expected running time of k-means algorithm is bounded by $O(N^3 4k^3 4D^8 \log^4(N)/\sigma^6)$ [4]. Even for simple cases, the best known bounds on average running time are at least $O(N^4)$.

3.2 A *Similarity* Function for Images

In spite of its high computational complexity in number of elements, *k*-means is popular in practice because the time taken for each *Distance* and *Merge* operation is usually very small. Even for problems with 100 dimensions, *Distance* and *Merge* operations require only about 100 addition and division operations. However, these operations are not very well-defined for VM images. We first define a natural definition of these operations and then present the time taken for each operation.

For VM images, it is more natural to define a *similarity* measure than a distance measure. Two images are similar if they contain a large number of identical elements (files or software). Given a pair of images I_i, I_j, similarity between the images can be defined as

$$SIM(I_i, I_j) = \frac{wt(I_i \cap I_j)}{wt(I_i \cup I_j)} \quad (1)$$

Image Size	Similarity	Merge
8.8 GB	45.5 sec	14.7 sec
12.3 GB	75.2 sec	24.1 sec
13.6 GB	98.5 sec	31.2 sec
16.3 GB	142.3 sec	44.2 sec
19.7 GB	172.2 sec	53.5 sec
22.1 GB	232.7 sec	64.9 sec

Table 2: Time for *Similarity* and *Merge* operations. Images and file are stored in a database making use of appropriate indices for these operations.

where $I_i \cup I_j$ is a meta-image that consists of the union of I_i and I_j, $I_i \cap I_j$ is a meta-image that consists of the intersection of I_i and I_j. The weight (wt) function is defined based on the type of similarity that needs to be computed. To estimate content similarity, the wt function is the sum of all files in the image, weighted by the sizes of the files. To estimate semantic similarity, the wt function is the sum of all software deployed in the image weighted by the complexity of the software. Adopting other notions of similarity is straightforward (e.g., a weighted composition of content and semantic similarity). *Distance* can now be calculated simply as $1 - SIM(I_i, I_j)$. The *Merge* operation would create a new image that constitutes the set of all unique elements across the images.

3.3 Scaling Challenge

We measured the running time for a single *Similarity* and a single *Merge* operation on a dual-core 2 GHz Intel Xeon with 4GB memory and images stored on a 5-disk RAID5 SATA array. Table 2 lists run times for real images of different sizes. While the actual times seem small, in aggregate, the costs of these operations present a significant challenge. For example, a data center with 1000 images would have to perform 1000^3 similarity computations (even for the best special cases on average complexity), and would need about 2000 years.

In-memory data structures can reduce the cost of these operations. We conducted experiments by enabling the in-memory feature in MySQL. We observed that the maximum time taken for one similarity computation is 5 seconds (a reduction of $50X$), which though significant only brings down the similarity computation in our previous example to 40 years. Further, this requires the entire index to be memory resident which is not practical. One could envision computing similarity based on only files that are larger than a certain threshold size in each image, but that again would bring down the running time only by a constant factor, while compromising accuracy.

An alternate approach to speed up clustering is to perform approximate clustering based on pair-wise similarity information. The k-medoids clustering algorithm [12] does exactly that by restricting the cluster center in an iteration to one of the existing points (images). Hence, both assignment and update steps in each iteration can leverage pair-wise similarity values that are computed in advance. This simplifying approximation, however, still requires pair-wise similarity computation for all images. Since individual similarity operations are expensive for VM images, this approach becomes un-affordable in practice for moderate to large numbers of VMs as is typical in a cloud, as we shall demonstrate later (§4.3). Anecdotally, in a recent study on VM image similarity, the authors reported pair-wise similarity only for a fraction of their image corpus citing scalability challenges [7]. With 1000 images, this would take 2 years with the file systems on disk and 15 days with an in-memory system. Clearly, there is a need to reduce the number of operations even further. Unfortunately, k-medoids suffers from an additional challenge, that of determining k a priori. The value of k should ideally be the minimum number of clusters required subject to cluster size constraints dictated by the application. However, this information is not always known a priori. In the next section, we discuss an approach that successfully overcomes the core limitations of existing clustering approaches.

4 *Coriolis*

Coriolis uses a novel approach to VM clustering. We discuss this approach and evaluate its scalability relative to the state-of-the-art k-medoids clustering in this section.

4.1 Solution Idea: Asymmetric Clustering

To solve the computational and memory challenge in VM clustering, we draw on a key insight in *Coriolis*. First, we observe that to significantly speed-up the *Distance* and *Merge* operations, caching only a small subset of the image manifest and hash index of image content must be able to satisfy a large fraction of operations. Enabling cache effectiveness requires introducing asymmetry into the clustering algorithm, that is, the algorithm cannot afford to consider all content from all images as equally important. The *Coriolis* clustering approach involves constructing a tree, where each node in the tree is either a cluster of images or a single image, such that each level in the tree from the root node represents a minimum extent of similarity within images in a cluster. The salient aspects of this approach are:

- **Hierarchical multi-level similarity**: Use multiple levels of similarity to quickly find most relevant clusters. By design, restrict comparisons only with clusters that are similar, reducing the total number of *Similarity* operations.
- **Ordered Index Lookup**: Clusters at low similarity levels are more popular than leaf nodes. Images with popular content will require more accesses and can be cached.
- **Online Clustering:** Add a new node to existing clusters. Allows addition/deletion of images with only incremental computation.

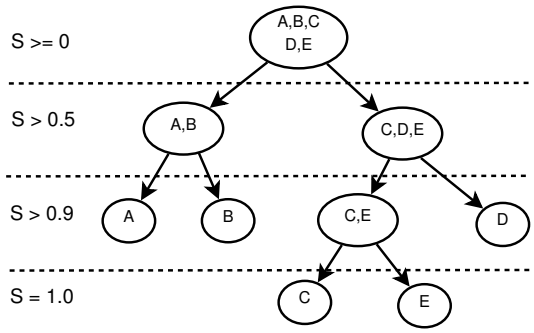

Figure 1: Tree-based clustering. Computed Similarity Values {(A,B):0.75, (C,E):0.95, (CE, D):0.8}

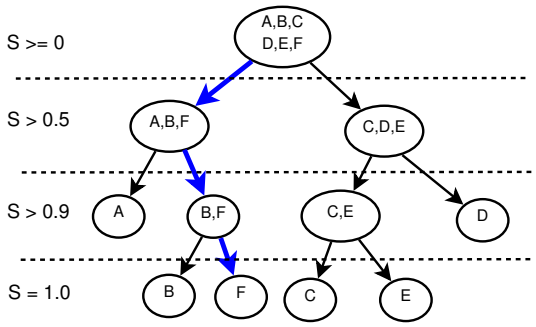

Figure 2: Clustering a new image F. Computed Similarity Values are {(AB,F):0.95, (CDE,F):0.3, (A,F):0.75, (B:F):0.95}

4.2 *Coriolis*' Tree-based Clustering

Coriolis's tree-based clustering approach is outlined below and it is based on two key ideas. The most common operation in clustering is to identify the cluster most similar to a given element and the first idea focuses on speeding up this operation. Since clusters can grow to become very large whereas individual images are typically small, we define and use an asymmetric similarity function S within *Coriolis* that runs in time proportional to the smaller of the two. In particular, we define similarity as the coverage offered by a larger node B (typically a cluster) to a new node A that is being added to the cluster by replacing the *union* operator in the denominator by the *min* operator.

$$S = \frac{wt(A \cap B)}{min(wt(A), wt(B))} \tag{2}$$

Our second key idea is to ensure skew in the usage of images and image clusters allowing effective caching. Further, we reuse the similarity computations done for an image when computing similarity for other images. *Coriolis* uses a tree-based partitioning of the images to achieve both these goals. Each level of the tree represents a predefined minimum level (extent) of similarity. The root of the tree captures a similarity level $S \geq 0$. Thus, all images can be clustered in this meta-node. The last level of the tree captures a similarity level $S = 1$; it consists of either single images or a collection of duplicate images. Intermediate levels represent predefined similarity levels, $0 < S < 1$, which increases with the depth of the tree. We elaborate our representation using the example in Figure 1. Consider 5 images A, B, C, D, E. The tree has 4 levels representing similarity of $0, 0.5, 0.9$ and 1 respectively. A and B have a similarity measure of 0.75. Hence, they are clustered at level $S > 0.5$ but are independent nodes at level $S > 0.9$. Similarly, C and E have a similarity of 0.95 and are grouped together up to all levels $S > 0.9$ but are independent nodes at level $S = 1$.

Given a new image v_i, our goal is to find similar nodes (or meta-nodes) with as few *Similarity* operations as possible. *Coriolis*'s grouping of VM image clusters within a hierarchical tree structure allows early pruning of images that are not similar to the new image v_i. Adding a new image to the *Coriolis* VM image tree, the new image is first added to the root meta-node. Once an image is added to a node, we compute the similarity of the new node with each of its children to determine if it can be added to any child. If the similarity S level is found adequate with more than one child, the new image is only added to the child node with which the similarity is the greatest. If no such child node exists, we create a new child node and add v_i to the node. This process terminates when we reach a leaf node.

Figure 2 illustrates a new image F as it traverses the tree. It is important to note here that the number of *Similarity* and *Merge* operations executed for an image is proportional to the depth of the tree. The depth of the tree is a pre-defined constant, bound by the *log* of the number of images inserted. Hence, the approach allows us to create a tree in time no more than $O(N \log N)$, where N is the number of images. And given the similarity levels at various tree depths, the tree can then be queried in linear time for clusters with specific properties.

4.3 Scalability Evaluation

To evaluate *Coriolis*, we used VM images from 2 production data centers. The first set of 9 images is from a large-scale enterprise data center at IBM. The latter set of 12 images is from the CS department's small-scale data center at Florida International University. The former set of images are diverse compared to the latter set reflecting the needs typical of a large-scale enterprise data center. Next, we created increasingly larger sets of images from these initial set of 21 production images. We did this by separating out 3 of the 21 images and randomly sampling files contained within these to generate synthetic images. The net effect is that the synthetic images contain a random combination of files from these 3 source images. We performed clustering experiments in a Linux VM configured with 16 GB RAM on an 6-core AMD Opteron processor virtualized using the VMware ESX hypervisor.

We choose k-medoids for this comparison as it is sig-

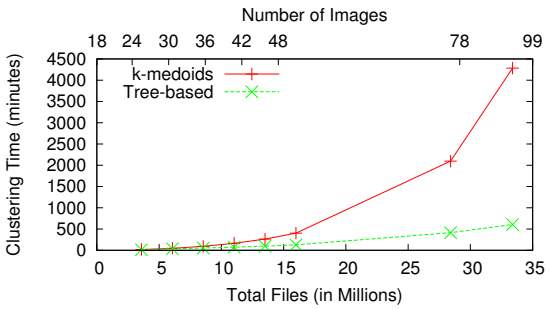

Figure 3: Scalability of k-medoids and tree-based clustering algorithms.

nificantly faster than *k*-means. Fig 3 presents the time taken by the k-medoids algorithm and the tree-based clustering algorithm as the problem size is increased. The time includes the time taken to read file metadata and store it in a database, where similarity and merge operations are performed. The *k*-medoids algorithm takes significantly longer and displays a quadratic increase in clustering time as the number of images is increased. We observed that more than 95% of the time is spent in computing similarity as the cluster size is increased. For clustering 99 images, it takes nearly 3 days, which is clearly unacceptable. In contrast, our tree-based clustering algorithm reduces the number of similarity computations by a factor of 8 and is able to cluster the images within 10 hours; an acceptable window of time even for heavyweight management VM tasks carried out over weekends.

5 Related Work

Redundancy elimination based on identifying duplicate data is a popular topic of research [10, 15]. Finding similar clusters is a related problem but is more data intensive because it requires processing over the entire index of the data as well as a manifest linking images to their contents. Further, the data access for this problem does not have inherent data popularity and locality, which is used extensively by deduplication techniques for scaling.

The research work closest to ours is VMFlocks which applies standard de-duplication techniques for images that are migrated together across data centers [2]. Given a batch of images, It eliminates raw data duplicates across the given set of VM images. However, it does not tackle identifying images with high redundancy or leveraging semantic similarity.

6 Conclusions

We described the *Coriolis* framework and system that was specifically designed for scalable clustering of VM images so as to counter the negative effects of VM sprawl in cloud data centers. We argued that the state-of-the-art k-medoids clustering algorithm incurs quadratic complexity which we demonstrated as infeasible for cloud scale data centers. *Coriolis*'s distinguishing strength lies in its scalable tree-based image clustering technique that supports an arbitrary similarity metric. This novel technique allows clustering to be performed in $O(N \log N)$ time for a data center with N images, allowing it to scale to large data centers. Our future work will explore the utility of *Coriolis* for data center administrator allocation, troubleshooting, and large-scale VM migration.

Acknowledgments

We thank the paper's reviewers and shepherd, Mustafa Uysal, for helpful feedback. This work was supported by NSF grant CNS-0747038 and an IBM faculty award.

References

[1] IDC Linux Standardization White Paper: Executive Summary. http://www.redhat.com/f/pdf/IDC_Standardize_RHEL_1118_Exec_summary.pdf, 2011.

[2] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu. VMFlock: Virtual Machine Co-migration for the Cloud. In *Proc. of the IEEE/ACM HPDC*, June 2011.

[3] G. Ammons, V. Bala, T. Mummert, D. Reimer, and X. Zhang. Virtual machine images as structured data: the Mirage image library. In *Proc. HotCloud*, 2011.

[4] D. Arthur, B. Manthey, and H. Roeglin. k-means has polynomial smoothed complexity. In *IEEE FOCS*, 2009.

[5] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li. Decentralized Deduplication in SAN Cluster File Systems. In *Proc. of the USENIX ATC*, June 2009.

[6] B. Debnath, S. Sengupta, and J. Li. ChunkStash: speeding up inline storage deduplication using flash memory. In *Usenix ATC*, 2010.

[7] K. R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, and H. Lei. An empirical analysis of similarity in virtual machine images. In *ACM Middleware*, 2011.

[8] K. Jin and E. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proc. of SysStor*, 2009.

[9] R. Koller and R. Rangaswami. I/O Deduplication: Utilizing Content Similarity to Improve I/ O Performance. In *Proc. of USENIX FAST*, 2010.

[10] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse indexing: large scale, inline deduplication using sampling and locality. In *USENIX FAST*, 2009.

[11] D. T. Meyer and W. J. Bolosky. A Study of Practical Deduplication. In *Proc. of USENIX FAST*, February 2011.

[12] L. Rousseeuw and L. Kaufman. Clustering by means of medoids. *Statistical data analysis based on the L1-norm and related methods*, 405, 1987.

[13] B. Viswanathan, A. Verma, B. Krishnamurthy, P. Jayachandran, K. Bhattacharya, and R. Ananthanarayanan. Rapid adjustment and adoption to MIaaS clouds. In *ACM Middleware, Industry track*, 2012.

[14] VMWare. VMWare vSphere Data Protection. Technical White Paper, June 2012.

[15] B. Zhu, K. Li, and H. Patterson. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. *USENIX FAST*, 2008.