Image Management in Geo-distributed Clouds and Edge Environments

Jad Darrous, Shadi Ibrahim, Christian Perez

Avalon/Stack team, Inria

Final Discovery Meeting

10 July 2019

Global Context

- Virtualization is an enabling technology for cloud computing.

Global Context

- Virtualization is an enabling technology for cloud computing.
- Services in the cloud are deployed as *Virtual Machines (VMs)* or *Containers*.

Global Context

- Virtualization is an enabling technology for cloud computing.
- Services in the cloud are deployed as *Virtual Machines (VMs)* or *Containers*.
- VMs and Containers are launched from *disk images*.

Global Context

- Virtualization is an enabling technology for cloud computing.
- Services in the cloud are deployed as *Virtual Machines (VMs)* or *Containers*.
- VMs and Containers are launched from *disk images*.
- Disk images (i.e., VM image / container image) contain the software and the dependencies for the service (including the OS in case of VMIs).

Global Context

- Virtualization is an enabling technology for cloud computing.
- Services in the cloud are deployed as *Virtual Machines (VMs)* or *Containers*.
- VMs and Containers are launched from *disk images*.
- Disk images (i.e., VM image / container image) contain the software and the dependencies for the service (including the OS in case of VMIs).

Our work

Image management for *efficient service provisioning* in Geo-distributed clouds and Edge environments.

PART 1: Network-aware VM Image *Retrieval* in Geo-distributed Clouds

Geo-distributed VMI management

- Clouds are going geographically distributed.

[1] Amazon AWS provides more than 19,000 public images (Nov. 2017)

Geo-distributed VMI management

- Clouds are going geographically distributed.
- VMIs are essential to build cloud services
  - Number of VMIs is continuously increasing[1]
  - VMI sizes could be up to dozens of GBs

[1] Amazon AWS provides more than 19,000 public images (Nov. 2017)

Geo-distributed VMI management

- Clouds are going geographically distributed.
- VMIs are essential to build cloud services
  - Number of VMIs is continuously increasing[1]
  - VMI sizes could be up to dozens of GBs
- Replicating VMIs on all sites is not practical
  - High cost in term of *data transfer*, *time*, and *money*
  - VMIs are updated frequently ($>$ 100 patches per week)

[1] Amazon AWS provides more than 19,000 public images (Nov. 2017)

Geo-distributed VMI management

- Clouds are going geographically distributed.
- VMIs are essential to build cloud services
  - Number of VMIs is continuously increasing[1]
  - VMI sizes could be up to dozens of GBs
- Replicating VMIs on all sites is not practical
  - High cost in term of *data transfer*, *time*, and *money*
  - VMIs are updated frequently ($>$ 100 patches per week)
- On-demand VMIs acquisition is subject to
  - Low bandwidth (35 Mbps)
  - Link Heterogeneity (X12 difference in bandwidth)

[1] Amazon AWS provides more than 19,000 public images (Nov. 2017)

Nitro

Nitro is a VMI management system that focuses on *minimizing the transfer time* of VMIs over a heterogeneous WAN.

- Reduce network overhead (by employing deduplication)
- Network-aware data retrieval (optimal chunk retrieval algorithm)
- Ensure minimal runtime overhead (runs in subsecond)

Nitro

Nitro is a VMI management system that focuses on *minimizing the transfer time* of VMIs over a heterogeneous WAN.

- Reduce network overhead (by employing deduplication)
- Network-aware data retrieval (optimal chunk retrieval algorithm)
- Ensure minimal runtime overhead (runs in subsecond)

Experimental results

Nitro outperforms state-of-the-art VMI storage systems (e.g., OpenStack Swift) by up to 77%

Nitro (Software)

Software:

- Written in Python; 1500 LoC
- Publicly available
  `https://gitlab.inria.fr/jdarrous/nitro`
- License: GPL-3.0

Nitro (Software)

Software:

- Written in Python; 1500 LoC
- Publicly available
  `https://gitlab.inria.fr/jdarrous/nitro`
- License: GPL-3.0

There is a on going discussion to deploy Nitro for the *Institut Pluridisciplinaire Hubert CURIEN (IPHC) - CNRS*

Publications

International conferences:

- J. Darrous, S. Ibrahim, A.C. Zhou, and C. Perez. "Nitro: Network-Aware Virtual Machine Image Management in Geo-Distributed Clouds". In: *18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2018)*. 2018.

PART 2: Network-aware Container Image *Placement* in the Edge

Table of Contents

Containers in Fog/Edge

- Containers are gaining popularity due to their lightweight overhead[2].

[2] Google launches more than 2 billion containers a day

Containers in Fog/Edge

- Containers are gaining popularity due to their lightweight overhead[2].
- Moreover, container are widely accepted as the virtualization technology for Edge.

[2] Google launches more than 2 billion containers a day

Containers in Fog/Edge

- Containers are gaining popularity due to their lightweight overhead[2].
- Moreover, container are widely accepted as the virtualization technology for Edge.
- However, downloading images from distant remote repository is time consuming.
  $\hookrightarrow$ 500 MB image over 5 MB/s link takes 100s to be downloaded.

[2] Google launches more than 2 billion containers a day

Containers in Fog/Edge

- Containers are gaining popularity due to their lightweight overhead[2].
- Moreover, container are widely accepted as the virtualization technology for Edge.
- However, downloading images from distant remote repository is time consuming.
  $\hookrightarrow$ 500 MB image over 5 MB/s link takes 100s to be downloaded.

What we propose

Placing container images across Edge servers!

[2] Google launches more than 2 billion containers a day

Service provisioning

- **Goal**: providing *fast* and *predictable* retrieving times for a set of images on the entire network.

- **Challenges**:
  - **Heterogeneity of the network (bandwidth)**
  - **Avoiding data loss**
  - **Limited storage capacities**

Service provisioning

- **Goal**: providing *fast* and *predictable* retrieving times for a set of images on the entire network.
  $\hookrightarrow$ Reduce the maximum time to retrieve an image to any Edge-server.
- **Challenges**:
  - **Heterogeneity of the network (bandwidth)**
  - **Avoiding data loss**
  - **Limited storage capacities**

Service provisioning

- **Goal**: providing *fast* and *predictable* retrieving times for a set of images on the entire network.
  $\hookrightarrow$ Reduce the maximum time to retrieve an image to any Edge-server.
- **Challenges**:
  - **Heterogeneity of the network (bandwidth)**
    $\hookrightarrow$ Network awareness during placement and retrieval
  - **Avoiding data loss**
  - **Limited storage capacities**

Service provisioning

- **Goal**: providing *fast* and *predictable* retrieving times for a set of images on the entire network.
  $\hookrightarrow$ Reduce the maximum time to retrieve an image to any Edge-server.

- **Challenges**:
  - **Heterogeneity of the network (bandwidth)**
    $\hookrightarrow$ Network awareness during placement and retrieval
  - **Avoiding data loss**
    $\hookrightarrow$ Replication of images
  - **Limited storage capacities**

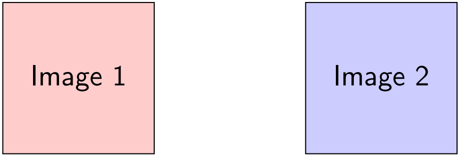
Service provisioning

- **Goal**: providing *fast* and *predictable* retrieving times for a set of images on the entire network.
  $\hookrightarrow$ Reduce the maximum time to retrieve an image to any Edge-server.
- **Challenges**:
  - **Heterogeneity of the network (bandwidth)**
    $\hookrightarrow$ Network awareness during placement and retrieval
  - **Avoiding data loss**
    $\hookrightarrow$ Replication of images
  - **Limited storage capacities**
    $\hookrightarrow$ Not too much replications!

Table of Contents

Docker, Images and Layers

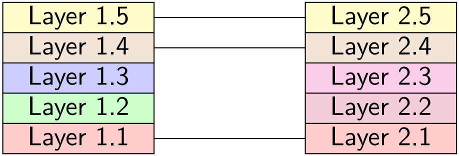
- We base our model on the Docker structure of container images.
- Each image is composed of several layers (Libraries, software. . .).
- A layer can be shared by several images.



- Layers are replicated, not images
  $\hookrightarrow$ Gain in term of storage cost.

Docker, Images and Layers

- We base our model on the Docker structure of container images.
- Each image is composed of several layers (Libraries, software. . .).
- A layer can be shared by several images.



- Layers are replicated, not images
  $\hookrightarrow$ Gain in term of storage cost.

Retrieving assumptions

- We focus on placement here but we need to define the retrieving policy.
- **Policy**: If an image is requested on one node, each layer is individually retrieved from the node that owns a replica that has the *largest* bandwidth.
- **The retrieving time of an image is determined by the longest retrieving time among the ones of its layers.**

MaxLayerRetrievalTime

Problem (*MaxLayerRetrievalTime*)

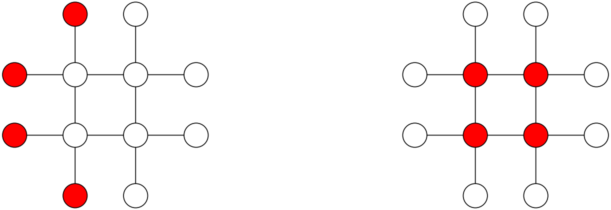
Let V be a set of nodes with storage capacity c and $\mathcal{L}$ be a set of layers. Return a valid placement that minimizes: $\max_{u \in V,\ l_i \in \mathcal{L}} T_i^u$.

- V: set of nodes of the network (seen as a complete graph).
- c: storage capacity of a node (equal for all nodes).
  $\hookrightarrow$ The sum of the sizes of layers stored on each node has to be lower than c.
- T_i^u: retrieving time of layer l_i on node u.
  $\hookrightarrow$ Depends on the size of l_i and on the bandwidth between u and the chosen node.

k-Center

Problem (k-Center)

Placing k facilities on a graph such that the maximum distance from any node to any facility is minimized.



- Popular model for Content Delivery Networks (CDNs).

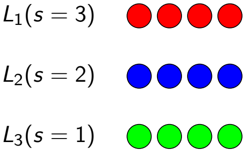
k-Center

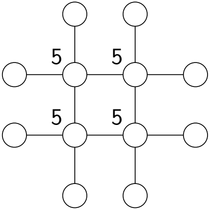
- *k*-Center is NP-complete.
- The best possible approximation ratio is 2 (worst case scenario).
- Some algorithms with good average ratio exist (1.058 on a classic benchmark).
- With only one layer, replicated *k* times, *MaxLayerRetrievalTime* is equivalent to *K-center*.
  $\hookrightarrow$ *MaxLayerRetrievalTime* is NP-complete.
- Because of limited storage capacities, all layers cannot be placed on the *k* most central nodes.

k-Center Based Placement

- Our solution: iterating a k-Center approximation algorithm.

- Sort the layers by decreasing sizes
- For each layer L_i with size s_i:
  - Use a k-Center solver (k number of replicas) on the subgraph with all nodes with remaining storage capacities $c_j \geq s_i$

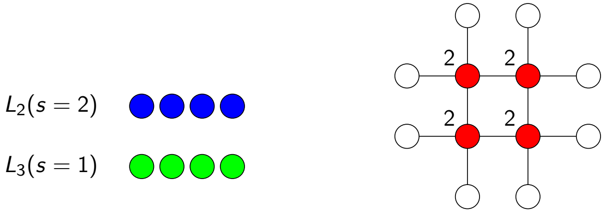




k-Center Based Placement

- Our solution: iterating a k-Center approximation algorithm.

- Sort the layers by decreasing sizes
- For each layer L_i with size s_i:
  - Use a k-Center solver (k number of replicas) on the subgraph with all nodes with remaining storage capacities $c_j \geq s_i$

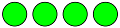


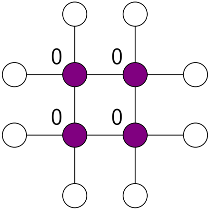
k-Center Based Placement

- Our solution: iterating a k-Center approximation algorithm.

- Sort the layers by decreasing sizes
- For each layer L_i with size s_i:
  - Use a k-Center solver (k number of replicas) on the subgraph with all nodes with remaining storage capacities $c_j \geq s_i$

$L_3(s = 1)$

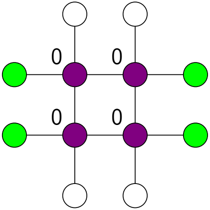




k-Center Based Placement

- Our solution: iterating a k-Center approximation algorithm.

- Sort the layers by decreasing sizes
- For each layer L_i with size s_i:
  - Use a k-Center solver (k number of replicas) on the subgraph with all nodes with remaining storage capacities $c_j \geq s_i$



MaxImageRetrievalTime

- An image is a set of layers.
- Several layers downloads from the same node may degrade the bandwidth.
- Layer-level placement may be too optimistic.
- **New rule**: if several layers are retrieved from the same node, these downloads are done sequentially.

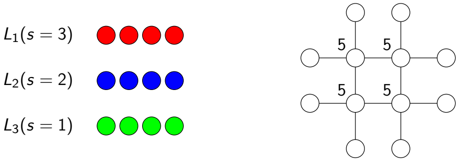
Problem (*MaxImageRetrievalTime*)

Let V be a set of nodes with storage capacity c and $\mathcal{I}$ be a set of images. Return a valid placement that minimizes: $\max_{u \in V, I_j \in \mathcal{I}} T^u_{I_j}$.

k-Center Based Placement-Without Conflict

- KCBP tends to gather many layers on same nodes $\rightarrow$ higher chance to have two layers of an image on the same nodes.

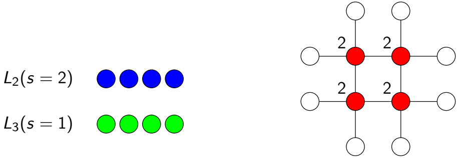
- Sort the layers by decreasing sizes
- For each layer L_i with size s_i:
  - Use a k-Center solver (k number of replicas) on the subgraph with all nodes with remaining storage capacities $c_j \geq s_i$ **and that do not own layers that share an image with this layer**



k-Center Based Placement-Without Conflict

- KCBP tends to gather many layers on same nodes $\rightarrow$ higher chance to have two layers of an image on the same nodes.

- Sort the layers by decreasing sizes
- For each layer L_i with size s_i:
  - Use a k-Center solver (k number of replicas) on the subgraph with all nodes with remaining storage capacities $c_j \geq s_i$ **and that do not own layers that share an image with this layer**

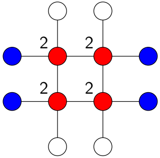


k-Center Based Placement-Without Conflict

- KCBP tends to gather many layers on same nodes $\rightarrow$ higher chance to have two layers of an image on the same nodes.

- Sort the layers by decreasing sizes
- For each layer L_i with size s_i:
  - Use a k-Center solver (k number of replicas) on the subgraph with all nodes with remaining storage capacities $c_j \geq s_i$ **and that do not own layers that share an image with this layer**

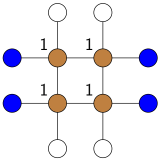
$L_3(s = 1)$



k-Center Based Placement-Without Conflict

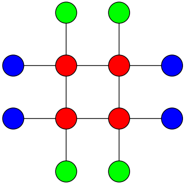
- KCBP tends to gather many layers on same nodes $\rightarrow$ higher chance to have two layers of an image on the same nodes.

- Sort the layers by decreasing sizes
- For each layer L_i with size s_i:
  - Use a k-Center solver (k number of replicas) on the subgraph with all nodes with remaining storage capacities $c_j \geq s_i$ **and that do not own layers that share an image with this layer**



k-Center Based Placement-Without Conflict

- We do not want to spread too much!



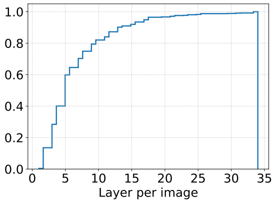
- What if another layer share an image with the three previous ones?
- We only apply the criterion "not sharing an image" on the α% *largest* layers ($\alpha = 10$ here).

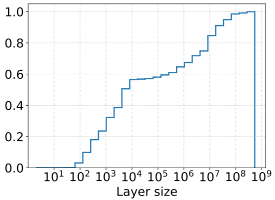
Table of Contents

Container Images

- IBM cloud traces from Frankfort data centers.

Total #images	996
Total size of images	93.76 GB
Total #layers	5672
Total size of unique layers	74.25 GB





Synthetic Networks

- Complete graphs with random bandwidths on edges.
- **Homogeneous**: same bandwidth for all.
- **Low**: most of the edges have low bandwidth.
- **High**: most of the edges have high bandwidth.
- **Uniform**: edges bandwidths follow a uniform distribution.

Network	**Number of nodes**	**Links bandwidths (bps)**				
		min	**25th**	**median**	**75th**	**max**
Homogeneous	50	4G	4G	4G	4G	4G
Low	50	8M	763M	1G	2G	8G
High	50	478M	5G	6G	7G	8G
Uniform	50	8M	2G	4G	6G	8G

Real Networks

- France and Slovakia national networks (retrieved from `www.topology-zoo.org`).
- Graph are made complete: The bandwidth between two nodes is the minimum bandwidth of the shortest path time 0.95^n where n is the size of the shortest path.

Network	Number of nodes	Links bandwidths (bps)				
		min	25th	median	75th	max
Renater	38	102M	126M	132M	139M	155M
Sanet	35	63M	6G	8G	8G	10G

Strategies

- Our placement strategies:
  - KCBP
  - KCBP-WC
- Comparison strategies:
  - Best-Fit (round-robin distribution of layers)
  - Random
  - 50 runs for each.

- All layers are replicated 3 times.
- Storage capacity: $f \times \frac{\text{size of total dataset}}{\text{number of nodes}}$, $f \in \{1.1, 2, INF\}$.

Impact of Conflicts

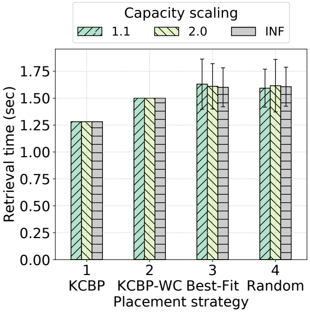


Figure: Layers Retrieval Times (High Network)

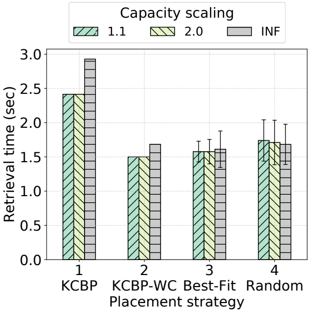


Figure: Images Retrieval Times (High Network)

- Conflicts are not negligible at all.
- "Extra space effect": having more storage capacity increase retrieving time.

Impact of Heterogeneity of Bandwidths

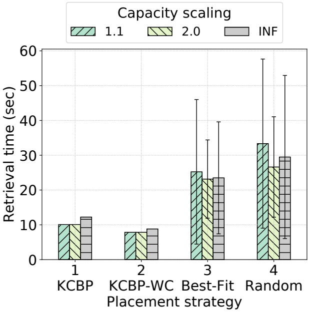


Figure: Low Network

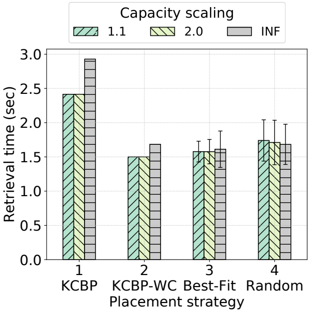


Figure: High Network

- Low Network: many "low connectivity nodes" $\rightarrow$ centrality of layers placement is important.
- High Network: few "low connectivity nodes".

Distribution of image retrieval times

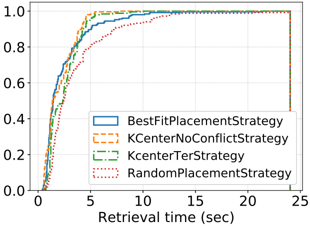


Figure: Low Network

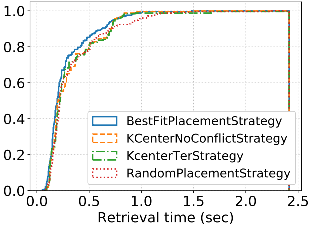


Figure: High Network

- Best-Fit has the best retrieval time for 20% of the largest images on High Network.
- For Low Network, KCBP-WC has the lead on these images.

Table of Contents

Contributions and Perspectives

- Contributions:
  - A formal model for container image placement on Fog/Edge networks.
  - Two placement strategies (i.e., KCBP and KCBP-WC).
  - A simulation-based evaluation with two state-of-the-art techniques.

Contributions and Perspectives

- Contributions:
  - A formal model for container image placement on Fog/Edge networks.
  - Two placement strategies (i.e., KCBP and KCBP-WC).
  - A simulation-based evaluation with two state-of-the-art techniques.
- Perspectives:
  - Improve placement strategies.
  - Add several level of replication.
  - Improve retrieving techniques.

Contributions and Perspectives

- Contributions:
  - A formal model for container image placement on Fog/Edge networks.
  - Two placement strategies (i.e., KCBP and KCBP-WC).
  - A simulation-based evaluation with two state-of-the-art techniques.
- Perspectives:
  - Improve placement strategies.
  - Add several level of replication.
  - Improve retrieving techniques.

Simulator code is publicly available at `https://gitlab.inria.fr/jdarrous/image-placement-edge`.

Publications

International conferences:

- J. Darrous, T. Lambert, and S. Ibrahim. "On the Importance of container images placement for service provisioning in the Edge". In: *28th International Conference on Computer Communications and Networks (ICCCN 2019)*. 2019.

Scientific contributions

Publications in international conferences:

- J. Darrous, S. Ibrahim, A.C. Zhou, and C. Perez. "Nitro: Network-Aware Virtual Machine Image Management in Geo-Distributed Clouds". In: *CCGrid'18*. 2018. (Core ranking A)
- J. Darrous, T. Lambert, and S. Ibrahim. "On the Importance of container images placement for service provisioning in the Edge". In: *ICCCN'19*. 2019. (Core ranking A)

Software:

- Nitro (GPL-3.0)

Open source code:

- Nitro, available at `https://gitlab.inria.fr/jdarrous/nitro`
- Container image placement simulator, available at `https://gitlab.inria.fr/jdarrous/image-placement-edge`

Backup slides

The relationship with VM placement

- Image management is critical for efficient service provisioning.
- VM/container schedulers could take the availability of (VM/container) images into account.
- Moreover, a cost function representing the retrieval time of an image can be integrated into the VM/container scheduler.

Integration with OpenStack

Nitro

- Nitro can be implemented as a backend storage for Glance.
- The backend storage is selected in configuration files.
- The modification on OpenStack source code is *Zero*.