Let’s Stay Together: Towards Traffic Aware Virtual Machine Placement in Data Centers

Xin Li, Jie Wu, Shaojie Tang, Sanglu Lu

Nanjing University
Temple University
Outline

- Background and Scenario
- Problem Statement
- Homogeneous Case
- Heterogeneous Case
- Conclusion
Virtual machine (VM) placement

- Tenants submit their resource requirements to the cloud system, and the cloud decides how to implement the resource allocation.
- One of the primary tasks in virtualization-based cloud systems.

The cost is one of the major concerns for the cloud providers.

- PM-cost
- N-cost
Scenario

- We use *slot* to represent one basic resource unit. (CPU/memory/disk)
- Tenants submit their resource requirements, in terms of the number of VMs (slots).
  - Each slot host one VM
- For one tenant, it could be one project group, and each VM can be assigned to one group member.
  - The VMs (group members) finish the task cooperatively.
Virtual Machine Placement

- Inter-PM traffic
  - Inter-VM traffic
- The objective
  - Minimize the total inter-PM traffic.

How to determine the communication cost?

$r_{ij}$: the VMs placed on PM $j$ of request $r_i$. 

placement for 4 requests on 3 PMs: $r_1, r_2, r_3, r_4$. 

Inter-PM traffic

Inter-VM traffic
Communication Model

- Two communication models

Centralized Model

Distributed Model

$r_{ij}$: the VMs placed on PM $j$ of request $r_i$. 
Communication Cost

- The traffics between VMs are assumed to be aware in most related works.

- Here, we do NOT adopt this assumption.

- We focus on network cost, measured by the number of traffic links between VMs
  - One request may be placed on multiple servers
Problem Statement

- Given a set of requests $R = \{r_i | 0 \leq i < n\}$, and a data center that consists of $m$ uniform PMs with $c$ slots for each. There may be traffic between VMs of the same tenant. Present a VM placement such that the overall network cost is minimized.
  - $\phi_i$: the cost caused by request $i$
  - $r_i$: the requirement of request $i$
  - objective:
    $$\min \sum_{i=0}^{n-1} \phi_i$$
Cost Function

- Centralized Model Cost Function (CCF)
  \[ - \phi_i^{(1)} = K_i \]

- Distributed Model Cost Function (DCF)
  \[ - \phi_i^{(2)} = K_i^2 \]

- Enhanced Distributed Model Cost Function (E-DCF)
  \[ - \phi_i^{(3)} = \frac{1}{2} \sum_{\kappa=1}^{K_i} r_i^{(\kappa)} \cdot (r - r_i^{(\kappa)}) \]

\( K_i \): the number of fractions of request \( i \).
Classification

- Only N-cost is discussed, PM-cost is fixed as the minimal number of PMs that can host all of the required VMs.
  - Homogeneous case
    - $r_i = r_j = r$;
  - Heterogeneous case
    - otherwise.
Homogeneous Case

- **CCF**
  - Recursive algorithm
  - Optimal solution

- **DCF**
  - Algorithm based on the above recursive algorithm
  - Optimal solution

- **E-DCF**
  - Recursive algorithm
  - Optimal solution
Homogeneous Case - CCF

- Recursive algorithm

Diagram showing the structure and solution with resource pieces and their sizes.
Homogeneous Case - CCF

- Basic idea
  - Achieve the *perfect placement* as many as possible, then split the unplaced requests into *pieces*.
    - Layer
  - Perfect placement (Stay Together)
    - All of the required VMs are placed on the same PM.
      - For each layer, the perfect placement may be different, i.e. the number of required VMs varies.
  - Piece
    - TPC: *terminal-piece*
    - CPC: *continue-piece*
Homogeneous Case - CCF

- **Piece**
  - TPC, terminal piece
    - One piece is placed completely without split at some layer;
  - CPC, continue piece
    - Otherwise.

- There is exactly one TPC for each request.
- There is at most one CPC on each PM.
Solution Structure

c = α \cdot r + u
r = β \cdot u + v
TPC: r
CPC: u

\[ c \leftarrow u \]
\[ r \leftarrow v \]
TPC: v
CPC: w

\[ \phi_i^{(1)} = K_i \]
Homogeneous Case - CCF

- **Swap operation**
  - \( s_i \): a set of pieces placed on PM \( i \).
  - \( \text{swap}(s_i, s_j) \)
    - \( s_i = s_j \)
    - \( s_i > s_j \)
      - Split \( s_i \) into two parts, \( s_i^* \) and \( s_i^\Lambda \), such that \( s_i^* = s_j \), then swap \( s_i^* \) and \( s_j \).
      - It is easy to get \( s_i^* \) by splitting ONLY one piece into two parts.
    - \( s_i < s_j \)
Optimality

- Theorem
  - The recursive algorithm gives the optimal solution when $\forall i, r_i = r \leq c$, and $\phi_i = \phi_i^{(1)} = K_i$, i.e., the CCF cost function.

- Proof
  - Case $\Omega$
    - For any PM, the sum of the sizes of the fragments is more than $r$.
  - Fragment
There is no case $\Omega$ in our solution.

- In the optimal solution, we can remove all of the case $\Omega$.
  - Let $r_{ij}$ be one of the fragment, and $s_j$ be the union of the other fragments of PM $j$.
  - There must be another fragment $r_{ij}$, on PM $j'$, and we have $s_j > r_{ij}$, since $s_j + r_{ij} > r$.
  - Swap operation: $\text{swap}(s_j, r_{ij})$.
  - The swap operation will not change the fact.
Repeat the swap operation until there is only one piece for \( r_i \), and the sum of the size of fragments on PM \( j \) can be reduced by \( r \).

- There can be no case \( \Omega \) in the optimal solution.
  - Reduce the optimal solution to our solution.
  - There are \( \alpha \) perfect placement in the layer 0.
  - For the remaining pieces, we can do swap operation to gather the pieces of the same tenant as close as possible.
Solution Structure

\[ c = \alpha \cdot r + u \]
\[ r = \beta \cdot u + v \]
TPC: \( r \)
CPC: \( u \)

\[ c \leftarrow u \]
\[ r \leftarrow v \]
TPC: \( v \)
CPC: \( w \)

recursively

\[ \phi^{(1)}_i = K_i \]
Homogeneous Case - DCF

- DCF: $\phi_i^{(2)} = K_i^2$
- CCF: the sum of the pieces is minimal.
- The basic idea
  - To minimize the objective function, we should achieve the $K$ distribution like this: 1,1,...,1,2,...,2
    - Swap operation
  - For given number of items, to minimize the sum of the square of items, its sum should be minimized, and it achieves the minimal value when all the
Homogeneous Case - Example

(a) Placement given by Algorithm 1. There are 2 layers, and $K_8 = 3$.

(b) The TPC of $r_8$ is located (red rectangle), and $s_4$ (red dashed rectangle) is selected.

(c) Do $swap(r_{85}, s_4)$, then we have $K_3 = 2, K_8 = 2$. $s_3$ (blue dashed rectangle) is selected.

(d) Do $swap(r_{84}, s_3)$. We achieve the final optimal placement.
Optimality

- Feasibility of the swap operation.
  - There must be at least 1 perfectly placed request on the PMs that contains CPC of $r_i$.
  - The perfectly placed request will provide its part to be swapped out of the PM.

(b) The TPC of $r_8$ is located (red rectangle), and $s_4$ (red dashed rectangle) is selected.

(c) Do $\text{swap}(r_{85}, s_4)$, then we have $K_3 = 2, K_8 = 2$. $s_3$ (blue dashed rectangle) is selected.
Optimality (cont.)

- Let the swap operation start from the TPC of $r_i$, so it is unnecessary for the PM that contains TPC of $r_i$.
  - Only one perfectly placed pieces on each PM is enough.
    - There is at most one CPC on each PM.
  - In fact, we have $\alpha (\alpha > 1)$ perfectly placed pieces on each PM.
After the swap operations for all requests that have more than 2 pieces, their piece number becomes to 1.

- For the other request that participate the swap operation (the perfectly placed request), their piece number becomes to 2.
- For the other, their piece number remains unchanged.
- We achieve the optimal $K$ distribution.
Homogeneous Case – E-DCF

- The same algorithm as the case CCF.
  - Recursive algorithm
    \[ \phi_i^{(3)} = \frac{1}{2} \sum_{\kappa=1}^{K_i} r_i^{(\kappa)} \cdot (r - r_i^{(\kappa)}) \]

- We assume that \( r_{iu}, r_{iv}, r_{ju}, r_{jv} \) are four pieces.
  - The four piece will not coexist in the optimal placement, because we can do \( \text{swap}(r_{iu}, r_{jv}) \) or \( \text{swap}(r_{iv}, r_{ju}) \).
  - If \( r_{iu} \geq r_{iv} \) and \( r_{iu} + r_{iv} > r_{ju} \), then \( r_{iu}, r_{iv}, r_{ju} \) will not coexist, since we can do \( \text{swap}(r_{ju}, r_{iv}) \).
Optimality

- From the two facts, we can construct the optimal solution from any given placement.
  - (1) Mark the pieces that have the size equal to $r$ as red; otherwise, black.
  - (2) Select the piece with largest size among the black pieces. (Assume that $r_{iu}$ is selected)
  - (3) Do $\text{swap}(r_{ju}, r_{iv})$, as shown above, until no $r_{ju}$ or $r_{iv}$ can be selected. Then mark the new $r_{iu}$ red.
Optimality (cont.)

- The impact of the swap operation (step 3).
  - The piece $r_{iu}$ will be larger.

- Feasibility of the swap operation.
  - $r_{iu}$ has the largest size among the black pieces.

- When the swap operation will be terminated.
  - $K_i = 1 \ (r_{iu} = r)$
    - The other pieces on PM $u$ are all marked as red.
      - If it still have black piece, the swap operation can continue.
Optimality (cont.)

- The red piece will not participate the swap operation.
  - The red piece has the size equal to r; (step 1)
  - There are no black pieces on the PM it located.
- From the construction process, there will be $\alpha$ perfect placement on each PM, and other requests will occupy as fewer PM as possible.
- The result matches the recursive solution.
Solution Structure

\[ c = \alpha \cdot r + u \]
\[ r = \beta \cdot u + v \]
TPC: \( r \)
CPC: \( u \)

\[ c \Leftarrow u \]
\[ r \Leftarrow v \]
TPC: \( v \)
CPC: \( w \)

recursively

solution structure
Heterogeneous Case

- SBP: Sorting-based Placement
- Basic idea: place the requests with larger VM requirements first.
  - Sorting
    - According the number of VMs that tenants require
    - Ascending order
  - Place the first item of the sequence ($r_0$)
    - Case 1: perfect placement
    - Case 2: split $r_0$ into two pieces
An Example

- The inputs:
  - \( r_1 = 3, r_2 = 6, r_3 = 4, r_4 = 5, r_5 = 7, r_6 = 2, r_7 = 5 \)
  - Different color
- Sorting: 7, 6, 5, 5, 4, 3, 2
Greedy Algorithm

- Basic idea
  - The basic idea of GBP is that, for each request, place the required VMs on the current PM as much as possible; when the current PM is fully loaded, then place the part that exceeds the PM capacity to the next PM. Hence, there are at most 2 pieces for each request. In fact, the total number of pieces will not exceed $m + n$, since there are at most $m$ requests that are split into two pieces.
Approximation Ratio of GA

\[
\sum_{i=0}^{n-1} \phi_i^{(1)} < m + n \leq 2 \cdot n \leq 2 \cdot OPT
\]

\[
\sum_{i=0}^{n-1} \phi_i^{(2)} \leq 4 \cdot n \leq 4 \cdot OPT
\]

\[
\sum_{i=0}^{n-1} \phi_i^{(3)} \leq \sum_{i=0}^{n-1} \frac{r_i^2}{4} \leq \frac{c^2}{4} \cdot n \leq \frac{c^2}{4} \cdot OPT
\]
Comparison

(a) CCF
(b) DCF
(c) E-DCF
Impact of Number of PMs
Conclusion

- VM placement for network cost minimization.
- Homogeneous case
  - Optimal solutions for 3 cost functions
  - CCF, DCF, E-DCF
- Heterogeneous case
  - Approximation algorithm
  - 2-approximation ratio for CCF.
Thank You!

Let’s Stay Together: Towards Traffic Aware Virtual Machine Placement in Data Centers

Xin Li

Email: lixin@dislab.nju.edu.cn