

An Effective Approach For Execution Of Workloads In Clouds With Improving Scalability And Negotiation Strategy

Authors: S.Sujithkumar¹; M.Newlin Rajkumar²

Affiliation: PG Scholar, Regional Centre of Anna University Coimbatore¹;

Assistant Professor, Regional Centre of Anna University Coimbatore²

ABSTRACT

Infrastructure-as-a-Service clouds offer entire virtual infrastructures for distributed processing while concealing all physical underlying machinery. Current cloud interface abstractions restrict users from providing information regarding usage patterns of their requested virtual machines (VMs). In the existing paper, present the design, implementation, and evaluation of a cloud gateway, Nefeli. Nefeli performs intelligent placement of VMs onto physical nodes by exploiting user-provided deployment hints. Hints realize placement preferences based on knowledge only the cloud consumer has regarding the intended usage of the requested VMs. By modeling workloads as patterns of data flows, computations, control/synchronization points, and necessary network connections, users can identify favorable VM layouts. But in this paper Scalability issues present in large infrastructures is main drawback and communication overhead, decreasing resource utilization on provider's side are the disadvantages. Due to this the performance of the system is reduced. To improve the performance, we have to improve the scalability and provide negotiation. In proposed paper, to improve the scalability we use proposed algorithms called VM placement algorithm and for negotiation we use ranking algorithm. We design an approximate algorithm that efficiently solves the VM placement problem for very large problem sizes and ranking algorithm for negotiation. We use traffic traces collected from production data centers to evaluate our proposed VM placement algorithm, and we show a significant performance improvement compared to existing generic methods that do not take advantage of traffic patterns and data center network characteristics

General Terms

Ranking algorithm, and VM scheduling algorithm.

Keywords

Cloud computing, VM Scheduling, VM Placement

1. INTRODUCTION

In this paper, proposed Nefeli, a virtual infrastructure gateway that lifts this restriction. Through Nefeli, cloud consumers provide deployment hints on the possible mapping of VMs to physical nodes. Such hints include the collocation and anticolocation of VMs, the existence of potential performance bottlenecks, the presence of underlying hardware features (e.g., high availability), the proximity of certain VMs to data repositories, or any other information that would contribute in a more effective placement of VMs to physical hosting nodes. Consumers designate only properties of their virtual infrastructure and remain at all times agnostic to the cloud internal physical characteristics. The set of consumer-provided hints is augmented with high-level placement policies specified by the cloud administration. Placement policies and hints form a constraint satisfaction problem that when solved, yields the final VM-to-host placement. As workloads executed by the cloud may change over time, VM-to-host mappings must follow suit. To this end, Nefeli captures such events, changes VM deployment, helps avoid bottlenecks, and ultimately, improves the quality of the rendered services. Scalability issues present in large infrastructures is main drawback. Consumer will not be aware of the current resource allocation on provider side so, the chances of new requests getting rejected are more. Thus, it will increase communication overhead between cloud provider and consumer as well as it will decrease resource utilization on provider's side. It will also degrade the performance of a provider in managing many incoming requests due to previously rejected ones.

2. OVERVIEW OF SYSTEM

The potential benefit of optimizing Traffic-aware VM Placement Problem (TVMPP) is greater with increased traffic variance within one partition, i.e. one composite application. This is attributed to the fact that VMs with high pair wise traffic volume are placed close to each other. The potential benefit of optimizing TVMPP is greater with increased number of traffic partitions, i.e. number of isolated composite applications, or decreased partition size, i.e.

applications running on less VMs. This is attributed to the fact that VMs belonging to the same composite application are placed close to each other. The potential benefit of optimizing TVMPP depends on the network architecture. The benefit is greater for a multi-layer architecture, such as BCube; the benefit is minimal for an architecture that employs network load balancing techniques, such as in VL2. The TVMPP problem is NP-hard and it belongs to the general QAP problem, for which no existing exact solutions can scale to the size of current data centers. Therefore, in this work we describe an approximation algorithm. According to this algorithm, the first one is solving TVMPP is intuitively equivalent to finding a mapping of VMs to slots such that *VM pairs with heavy mutual traffic be assigned to slot pairs with low-cost connections*. The second design principle is divide-and-conquer: we partition VMs into VM-clusters and partition slots into slot clusters. Then we first map each VM-cluster to a slot-cluster. For each VM-cluster and its associated slot-cluster, we further map VMs to slots by solving another TVMPP problem, yet with a much smaller problem size. VM-clusters are obtained via classical min-cut graph algorithm which ensures that VM pairs with high mutual traffic rate are within the same VM cluster. Such a feature is consistent with an early observation that traffic generated from a small group of VMs comprises a large fraction of the total traffic. Slot-clusters are obtained via standard clustering techniques which ensure slot pairs with low-cost connections belong to the same slot-cluster. Proposed algorithm will handle each advance reservation request. Input parameter of algorithm is request means offer from consumer which contains start_time, duration, number of virtual nodes, CPU amount and memory amount. Output parameter is the list of counter offers generated by algorithm if offer is not acceptable. Algorithm first tries to map the resource requirement of a request with current resource allocation at provider. If requested slot will be found then it will schedule advance reservation of a request. Otherwise it will generate counter offer list as explained below. It will try to find duration from starting time up to which requested resource amount i.e. nodes, CPU and memory amount, can be satisfied. It will add this slot in counter offer list. Next it will try to find a slot by varying start time which can satisfy all other parameters i.e. duration and resource amount and add it to counter offer list. A proposed algorithm that efficiently solves the VM placement problem for very large problem sizes. Improve the performance by improving scalability using a VM placement algorithm. To reduce communication overhead between consumer and provider of cloud and increase resource utilization on cloud provider side, negotiation is necessary. The algorithm to generate counter offers provides counter offers considering constraints' flexibilities to maximize the chances of their acceptance

3. IMPROVING THE SCALABILITY OF DATA CENTER NETWORKS

A proposed algorithm for VM placement which leverages unique features of traffic patterns

and network topologies in data centers. The pseudo-code for the algorithm is described in Algorithm. *SlotClustering*: n slots are partitioned into k clusters by using the cost between slots as the partition criterion. There are two approaches in implementing this function. One is a manual procedure by the operators, who can leverage their *prior* knowledge on network configurations. This approach may give better results but could be labor intensive. The other approach is running classical clustering algorithms based on the cost matrix. Note that our cost definition is the number of switches on the path between two slots, so it satisfies the triangle inequality. Thus this becomes the Minimum k -clustering Problem which is NP-hard. We solve this problem by the algorithm, with an approximation ratio. The output from SlotClustering is a set of slot-clusters, sorted in decreasing order of the total outgoing and incoming cost.

VM placement algorithm

Require: D (Traffic matrix), C (Cost matrix), k (Number of clusters, a parameter used in clustering and min-cut components), G (Graph weight matrix), $\{b_1, \dots, b_k\}$ (size of each cluster)

- 1: $n \leftarrow$ size of D {Find out VM count}
- 2: SlotClustering(C, k) {Partition slots into k clusters: $\{r_i\}$ }
- 3: Sort $\{r_i\}$, in decreasing order of the cost of edges having one endpoint in r_i . Each edge only counts once
- 4: $l \leftarrow$ size of G , Compute Gomory-Hu tree for G and obtain $l - 1$ cuts $\{g_i\}$ {These $l - 1$ cuts contains the minimum weight cuts for all server pairs}
- 5: Sort $\{g_i\}$ by increasing weight
- 6: for $i = 1$ to k do
- 7: Clear s_i
- 8: Find the minimum j such that removing $\{g_1, \dots, g_j\}$ will partition G into two components: c_1 with size $|b_i|$ and c_2 with size $l - |b_i|$
- 9: $s_i \leftarrow c_1$
- 10: $G \leftarrow c_2$
- 11: $l \leftarrow l - |b_i|$
- 12: end for
- 13: Return $\{s_i\}$
- 14: Assign s_i to r_i , $\forall i = 1, \dots, n$ {One-to-one mapping between slot-cluster and VM-cluster}
- 15: for $i = 1$ to k do
- 16: if $|s_i| > 1$ then {Multiple VMs in s_i }
- 17: Cluster-and-cut($D(s_i)$, $C(r_i)$, $|s_i|$){ $D(s_i)$: traffic matrix for s_i . $C(r_i)$: cost matrix for r_i . $|s_i|$: recursively call Cluster-and-Cut}
- 18: end if
- 19: end for

Then we need to partition n VMs into k VM-clusters with minimum inter-cluster traffic. More importantly, we must ensure that for any already formed slotcluster, there is a corresponding VM-cluster with the equal size. The partition method used here is adapted from the minimum k -cut algorithm. This algorithm is originally applied to balanced Minimum k -cut problems in which the k clusters have equal size. Besides, the cut sorting and removal procedure ensures that smaller cost cuts have higher chance to be removed earlier. As a result, VM-clusters with low outgoing/incoming traffic more likely

correspond to slotclusters with low-cost outgoing/incoming connections. This complies with the aforementioned first design principle. In Algorithm, after establishing a one-to-one assignment between slot-clusters and VM-clusters, the rest of the code solves the VM-to-slot assignment problem within each cluster.

4. IMPROVING THE NEGOTIATION STRATEGY

Algorithm to generate counter offers for advance reservation is implemented in Haizea as a component. Through experiments, it is demonstrated that our algorithm adapts to more allocation requests and ensures maximum resource utilization with better capacity planning. Cloud provider will give set of counter offers. To choose best suitable counter offer is tedious task for consumer. Proposed algorithm for user selection policy takes flexibility parameters from consumer and ranks all the counter offers. Rank will be used to sort counter offers. After that suitability test will be performed on each counter offer. Thus it provides suitable counter offers in a decreasing order of suitability. This algorithm will be provided at consumer side by provider to select best suitable offer.

RANKING ALGORITHM: GENERATE – COUNTER – OFFERS

INPUT:

Request from consumer: Offer (start_time, duration, nodes, CPU, mem)

Start time of a reservation: start_time

Duration of a reservation request: duration

No. of virtual nodes requested: nodes

CPU amount requested: CPU

Memory amount requested: mem

OUTPUT:

List of counter offers generated: counterOffer_List []

1:begin

2:if resources are available for requested slot then

3:Accept the requested offer

4:else

5:Find duration up to which requested resources will be available from starting time and add this slot in to counterOffer_List[]

6:Find next slot fulfilling all the requested parameters by varying starting time and add this slot in to counterOffer_List []. Consider this slot's start_time as end_time

7:Find all the freeslots within window (start_time, end_time)

8:for each slot in free slots

9:do

10: if slot duration $\geq 50\%$ requested duration, slot CPU $\geq 80\%$ requested CPU and slot mem $\geq 50\%$ requested mem then

11: Add it to counteroffer_List []

12: end if

13:done

14: for each slot in freeslots

15: do

16: if slot CPU is less than requested CPU and slot duration is greater than requested duration then

17: Add this slot in to counterOffer_List []

18: end if

19: if slot duration is less than requested duration and slot CPU is greater than requested CPU then

20: Add this slot in to counterOffer_List []

21: end if

22: done

23: if No. of counter offers generated ≤ 1 then

24: Repeat above process by considering CPU constraint as 70% of requested CPU

25: end if

26: if No. of counter offers generated > 0 then

27: Return counterOffer_List [] to consumer

28: end if

29: end if

30: end

Proposed algorithm will handle each advance reservation request. Input parameter of algorithm is request means offer from consumer which contains start_time, duration, number of virtual nodes, CPU amount and memory amount. Output parameter is the list of counter offers generated by algorithm if offer is not acceptable. Algorithm first tries to map the resource requirement of a request with current resource allocation at provider. If requested slot will be found then it will schedule advance reservation of a request. Otherwise it will generate counter offer list as explained below. It will try to find duration from starting time up to which requested resource amount i.e. nodes, CPU and memory amount, can be satisfied. It will add this slot in counter offer list. Next it will try to find a slot by varying start time which can satisfy all other parameters i.e. duration and resource amount and add it to counter offer list. This slot can be called as exact match slot because it is matching the entire resource requirement including duration. Starting time of exact match slot will be considered as end_Time variable. Starting time of a request will be considered as start_Time variable. Other counter offers will be generated by looking at resource allocation existing in between start_Time and end_Time.

Using change points, it will get free slots available between start_Time and end_Time. After getting free slots available, it will try to get slots which are satisfying 50% of required duration, 80% of required CPU amount and 50% of required memory amount. It will add all this kind of slots in counter offers list. Next it will try to get slots having CPU amount less than required CPU amount. Suppose CPU amount is x% less than required CPU amount then check if the duration of that slot is greater than or equal to $((100 / x) * \text{required duration})$. If that is the case then add this slot in counter offer list.

But this slot must have memory amount $\geq 50\%$ of required memory amount and CPU amount $\geq 50\%$ of required CPU amount. If it is not the case then check if the duration of found slot is greater than required duration. If it is then added this slot in counter offers list. Next it will try to get slots having duration less than the required duration. Suppose duration is x% less than required duration then check if the CPU amount of that slot is greater than or equal to $((100 / x) * \text{required_CPU})$. If that is the case then add this slot in counter offer list. But this slot must have memory amount $\geq 50\%$ of required memory

amount and duration $\geq 30\%$ of required duration. If it is not the case then check if the CPU amount of found slot is greater than required CPU amount. If it is then add this slot in counter offers list.

5. SYSTEM ARCHITECTURE

Through Nefeli, cloud consumers provide deployment hints on the possible mapping of VMs to physical nodes. We partition VMs into VM-clusters and partition slots into slot clusters. Then we first map each VM-cluster to a slot-cluster. The output from Slot Clustering is a set of slot-clusters, sorted in decreasing order of the total outgoing and incoming cost. After forming slot clusters, we need to partition n VMs into k VM-clusters with minimum inter-cluster

traffic. More importantly, we must ensure that for any already formed slot cluster, there is a corresponding VM-cluster with the equal size. This process continues until all VM-clusters with requested size are formed. As a result, VM-clusters with low outgoing/incoming traffic more likely correspond to slot clusters with low-cost outgoing/incoming connections. For negotiation strategy the proposed algorithm first tries to map the resource requirement of a request with current resource allocation at provider. If requested slot will be found then it will schedule advance reservation of a request. Otherwise it will generate counter offer list

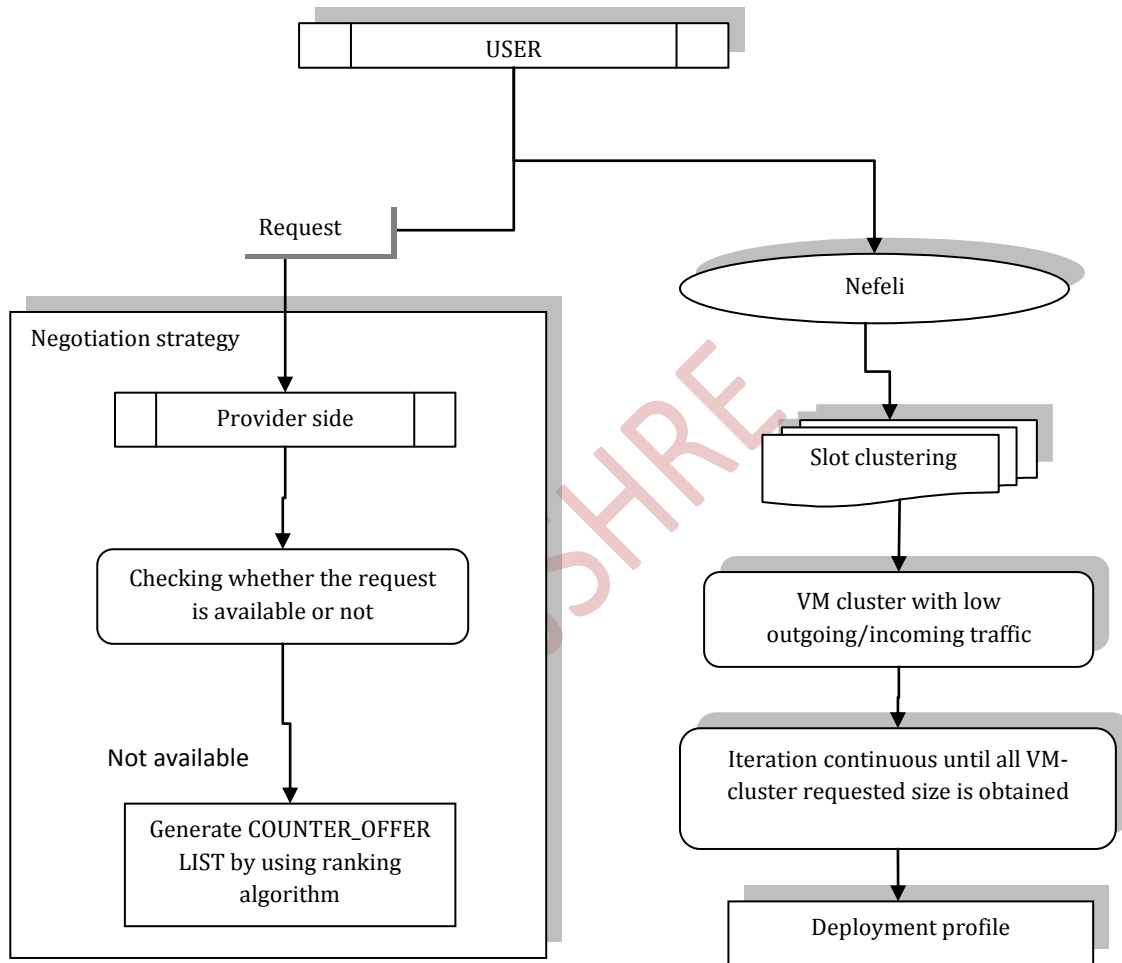


Fig 1: Overall System Architecture Diagram

6. CONCLUSION

In the existing paper, Nefeli is proposed for cloud consumers to provide deployment hints on the possible mapping of VMs to physical nodes. Such hints include the collocation and anticolocation of VMs, the existence of potential performance bottlenecks, the presence of underlying hardware features (e.g, high availability), the proximity of certain VMs to data repositories, or any other information that would contribute in a more effective placement of VMs to physical hosting nodes. But in this

scalability issue is main drawback and communication overhead, decreasing resource utilization on provider's side are the disadvantages. Due to this performance of the system is reduced. To overcome this problem, we introduced VM placement algorithms for improving the scalability in the network and ranking algorithm for negotiation strategy. This paper presents an approach of manipulating VM placement to address the scalability concern. Using ranking algorithm, consumers will get suitable offers sorted according to their needs. It will reduce consumers' efforts to go through all the

provided counter offers and choose best suitable one. By the analysis we can get the value and limit of using VM placement to improve network scalability under different network topologies and traffic patterns.

7. ACKNOWLEDGMENTS

Home is the backbone of every success and our humble salutations to my beloved Parents who inspired, motivated and supported us throughout the course of the project.

I also extend my sincere thank to all the Faculty Members of Department of CSE, Anna University Regional Centre, Coimbatore and Friends who have render their valuable help in completing this paper successful.

8. REFERENCES

- [1] Konstantinos Tsakalozos, Mema Roussopoulos, and Alex Delis, "Hint-Based Execution of Workloads In Clouds with Nefeli" IEEE transactions on parallel and distributed systems, vol. 24, no. 7, july 2013
- [2] Amazon, "Elastic Cloud," <http://aws.amazon.com/ec2/>, 2006.
- [3] F. Hermenier, J. Lawall, J.-M. Menaud, and G. Muller, "Dynamic Consolidation of Highly Available Web Application," RIA, Technical Report Research Report RR-7545, 2011
- [4] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whalley, and E. Snible, "Improving Performance and Availability of Services Hosted on IaaS Clouds with Structural Constraint-Aware Virtual Machine Placement," Proc. IEEE Int'l Conf. Services Computing (SCC), July 2011.
- [5] G. Jung, K.R. Joshi, M.A. Hiltunen, R.D. Schlichting, and C. Pu, "Performance and Availability Aware Regeneration For Cloud Based Multitier Applications," Proc. IEEE/IFIP Int'l Conf. Dependable Systems and Networks (DSN), June 2011.

IJSHRE