Constraint-based Data Center Resource Allocation

Nodir Kodirov
Computer Science department, University of British Columbia

***Abstract*—Cloud providers should be able to quickly and efficiently map customers virtual network requests to their physical infrastructure. Doing so *quickly* improves responsiveness of the provider - ability to start service faster, and doing so *correctly* makes efficient use of provider resources, giving higher return on investment. This work explores the use of Satisfiability Modulo Theories (SMT) solvers to solve the resource allocation problem. The SMT-based solution guarantees correctness, and SMT solver with special characteristics, such as monotonicity, can be used to further improve solver performance. In this project, we implement the constraint-based resource allocation algorithm using a general-purpose SMT solver, Z3, and then explore opportunity to use a special featured SMT solver, SAT Modulo Monotonic Theories, to compute the solution in even shorter time. Such performance improvements bring us one step closer to deploy SMT-solver to real data center environment, where quick allocation of physical resources is highly desired to reduce service provisioning delay.**

I. Introduction

Today, most of internet services rely on Infrastructure-as-a-Service providers to host their compute resources. Service providers request the virtual network which has to be mapped to the physical infrastructure of the cloud provider. Virtual networks consist of several (sometimes hundreds to thousands) virtual machines and connection between them. Cloud provider's task is to quickly and efficiently map the requested virtual network to the provider's physical network.

This work explores opportunities of using SMT solvers to efficiently allocate data center resources. Recent work [8] shows feasability of using SMT solver for resource allocation with bandwidth guarantees. In that work, resource allocaion is driven by introducing contraints, which mandate requirements to be satisfied for the mapping to be correct. Set of constraints are fed into SMT solver, Z3 [4]. Z3 does complete solution space exploration to find virtual machine (VM) to server assignment satisfying all mandated constraints. Output of the Z3 can be directly used to deploy virtual network to the physical infrastructure of the cloud provider. Z3's failure to find satisfying solution guarantees infeasibility of resource allocation for given virtual network.

The next section discusses the scope of this work, followed by a description of each constraint given to the SMT-solver. We conclude by discussing results, limitations, and future work.

II. Scope

Initial scope of this project was to compare performance of two SMT solvers. Previous work [8] used Z3, a publicly available, general-purpose SMT solver, and reported its performance. In that work it took around 200 seconds to map virtual network consisting of 15 VMs to physical network of 200 servers. This runtime was further reduced to 2 seconds by introducing abstraction and refinement to Z3 constraints. In this project we intend to solve the allocation problem using SAT Modulo Monotonic Theories (SMMT) [1], another SMT solver with specific characteristics to solve the allocation problem faster. Because SMMT is able to reason about graph edges, it can guarantee adjacency of the physical links allocated to a virtual link. Therefore, SMMT is expected to solve mapping problem within shorter time.

However, due to time constraints, only the first stage of the initial scope was achieved. Original paper [8] we planned to reimplement to experience with SMT solver was (largely) reproduces. This work is referred as an *original paper* hereafter. We also got an intuition to which of original paper's constraints could be solved faster by SMMT.

III. Implementation

This chapter describes constraints fed to the SMT solver to map virtual network (VN) to physical network (PN). Since all of these constraints were actually designed and explained by authors of the original paper, we only provide intuiton behind each constraint in the context of our sample allocation as in figure 1. Interested readers can refer to [8] for a formal and complete description, which also includes abstraction and refinement of these constraints, too.

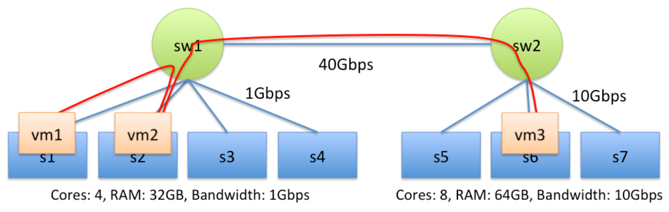
sw1
sw2
40Gbps
1Gbps
10Gbps
vm1
vm2
vm3
s1
s2
s3
s4
s5
s6
s7
Cores: 4, RAM: 32GB, Bandwidth: 1Gbps
Cores: 8, RAM: 64GB, Bandwidth: 10Gbps

Fig. 1: Virtual network to physical network mapping.

Figure 1 shows a sample VN to PN mapping, where VN consisting of three VMs: vm1, vm2, vm3 ($v \in V$) is deployed to PN with seven servers ($s \in A$) and two switches (sw1, sw2 $\in B$). Here, physical network of the cloud provider is encoded as $PN = (A \cup B, L)$, where A is set of servers, B is set of network nodes (L2 switches), and L is set of physical links that connect servers/switches with other switches. Virtual network $VN = (V, E)$ consists of the set of virtual machines $v \in V$, and virtual links $e \in E$ between them. In figure 1, there are two virtual links, vm1 to vm2, and vm2 to vm3.

The first constraint ensures each virtual machine v is mapped to the server s, and it is mapped only once. In this case $X(v, s)$ is integer variable equal to 1 if virtual machine v is assigned to server s, 0 otherwise.

$$\alpha_s : \bigwedge_{v \in V} \left(\sum X(v,s) = 1 \right).$$

Virtual links are laid over two or more physical links, i.e., each virtual link can include multiple physical links. Each such hop of the virtual link is indexes with k, and only one physical link can be k-th hop of the virtual link e. Following constraint encodes this requirement. $R(l,e,k)$ is assigned integer value 1 if physical link l is k-th hop of the virtual link e. Note that $0 \le k \le k_{max}$, where k_{max} is the maximum possible length of adjacent physical links (excluding loops). In figure 1 $k_{max} = 3$.

$$\alpha_r : \bigwedge_{e,k} \left(\sum_{l \in L} R(l,e,k) \le 1 \right).$$

If virtual link has more than one hop, we should ensure two consequent hops are actually made of adjacent physical links. The third constraint below encodes this requirement.

$$\alpha_c : \bigwedge_{e,k} \left(\bigvee_{l_1,l_2 : l_1,l_2 \ are \ adjacent} R(l_1,e,k) \wedge R(l_2,e,k+1) \right).$$

Now we introduce an integer variable to represent physical bandwidth allocated to virtual link. If physical link l lies on k-th hop of the virtual link e, $Y(l,e)$ is assigned value equal to bandwidth of the virtual link $r(e)$, 0 otherwise. Following constraint encodes such requirement. For example in figure 1 all three hops – s2 to sw1, sw1 to sw2, and sw2 to s6 – of the virtual link vm2 to vm3 get assigned value equal to this virtual link's bandwidth, i.e., $Y(l_{1,2,3}, vm1\_to\_vm2) = r(vm1\_to\_vm2)$.

$$\alpha_y : Y(l,e) = r(e) \Leftrightarrow \bigvee_k R(l,e,k) = 1.$$

For each virtual link, we need to make sure two host servers VMs get mapped to are physically connected. The next constraint encodes this requirement using previously defined variables $X(v,s)$ and $Y(l,e)$. Note that although $Y(l,e)$ in current formula represents physical links directly connected to the host servers, when combined with α_y and α_c constraints this representation extends to all hops between two servers.

$$\alpha_v : \bigwedge_{\substack{(v_1,v_2) \in E, \\ s_1,s_2 \in A, s_1 \neq s_2}} ((X(v_1,s_1) = 1 \wedge X(v_2,s_2) = 1) \rightarrow \bigvee_{\substack{l_1 : s_1 \in l_1 \\ l_2 : s_2 \in l_2}} (Y(l_1,e) = r(e) \wedge Y(l_2,e) = r(e))).$$

We need two more formulas to encode compute and link capacity of the servers. β_{server} ensures sum of VMs' compute resources mapped to a particular server does not exceed capacity of that server.

$$\beta_{server} : \bigwedge_{s \in A} \left(\sum_v X(v,s) \le c(s) \right).$$

β_{link} encodes link capacity of the server, i.e., sum of virtual links' bandwidth passing through a particular physical link must not exceed actual bandwidth $b(l)$ of that physical link. For example, for a mapping shown in figure 1, where both virtual links are allocated as an overlay to physical link sw1 to s2, following requirement $b(vm1\_to\_vm2) + b(vm2\_to\_vm3) \le b(sw1\_to\_s2)$ has to be satisfied.

$$\beta_{link} : \bigwedge_{l \in E} \left(\sum_e Y(l,e) \le b(l) \right).$$

Combinning all constraints together, we get VN allocation problem represented as an input to the SMT solver.

$$\Phi_{PN,VN} = \alpha_s \wedge \alpha_r \wedge \alpha_c \wedge \alpha_y \wedge \alpha_v \wedge \beta_{server} \wedge \beta_{link}$$

Output of the SMT solver is value assigned to X, Y, R variables, from which VM to server mapping, and virtual link overlays can be easily derived. Z3 outputs ünsatẅhen it is not possible to map given virtual network to the physical network. Because Z3 does complete solution space exploration, correctness is guaranteed, i.e., ünsatc̈onfirms infeasibility of the allocation (for example if capacity required by the VN exceeds actual capacity of the PN).

IV. Results, Limitations and Future Work

Our effort to replicate the original paper was largely successfull. Our current implementation maps virtual networks consisting of 3 VMs to physical networks consisting of seven servers and two switches within 300 milli-seconds. Although current implementation is able to handle simple topologies as shown in 1, it fails to provide correct virtual network overlays for more complex topologies. For example in figure 1, if all 3 VMs are mapped under the same switch, 2 VMs into the same server and one to different server, produced solution contains redundant hops. We believe this is due to a minor bug in our constraint encodings.

We plan to extend this work beyond class project. Immidiate future work is to fix aforementioned bug so that it would be possible handle all topologies. Source code with continuous update is available at Github project repository [2]. We also made a frozen release for the scope of this class project (which can no longer be modified) accessible at [3]. This report is based on latter source code.

Longer term future work consists of implementing abstraction and refinement introduced in the original paper, and comparing performance of Z3-based solution against SMMT-based one. Comparison could be further extended to heuristics, and Integer Linear Programming based solutions. These comparisons are indeed very useful since most of the papers published in major conferences use heuristics and ILP-based solutions [5], [6], [7]. SMT-based solutions are superior to heuristics-based conterparts not only because SMT-based approach is guaranteed to terminate, but it also guarantees correctness, i.e., solution which honours all mandated constraints [8].

Although we did not implement SMMT-based solution and have no empirical evaluation, yet, we expect it to be faster than Z3-based one. This improvement is the result of SMMT's ability to reason about adjacency of connected physical links. This means in SMMT implementation constraint α_c, which ensures consequent physical hops assigned to virtual link to be adjacent, can be solved much quickly. We intend to explore this in future work, too.

V. Acknowledgement

I am grateful to Sam Bayless, who introduced me SMMT solver and mentioned its applicability to solve data center resource allocation problem. Sam also shared his expertise on encoding Z3 constraints, saving me from spending hours on Z3 tutorial, and helped to debug my code when I got *really* stuck. I am also thankful to Microsoft Research for making Z3 publicly available.

This project has been mostly educational for myself, to get a hands-on experience with SMT sovers. Two other contributions are an open source implementation [2] of the original paper [8], and attempt to use SMMT to improve performance. All resource allocation constraints mentioned in chapter III are contribution of the original paper authors [8].

References

[1] S. Bayless, N. Bayless, H. H. Hoos, and A. J. Hu. Sat modulo monotonic theories. In *Work in progress*, Dec. 2014.

[2] N. Kodirov. Continuously updated github source code repository. https://github.com/knodir/mapper. [Online; accessed 24-Dec-2014].

[3] N. Kodirov. Frozen code for the class project. https://github.com/knodir/mapper/releases/tag/v1.0. [Online; accessed 24-Dec-2014].

[4] M. Research. Z3: Smt solver. http://z3.codeplex.com/. [Online; accessed 19-Dec-2014].

[5] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 351–364, New York, NY, USA, 2013. ACM.

[6] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 323–336, San Jose, CA, 2012. USENIX.

[7] A. Tumanov, T. Zhu, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger. TetriSched: Space-time scheduling for heterogeneous datacenters. Technical Report CMU-PDL-13-112, Carnegie Mellon University, Dec 2013.

[8] Y. Yuan, A. Wang, R. Alur, and B. T. Loo. On the feasibility of automation for bandwidth allocation problems in data centers. In *FMCAD'13*, pages 42–45, 2013.