This paper was presented as part of the Mini-Conference at IEEE INFOCOM 2011

Application-aware Virtual Machine Migration in Data Centers

Vivek Shrivastava*, Petros Zerfos†, Kang-won Lee†, Hani Jamjoom†, Yew-Huey Liu†, Suman Banerjee*

*University of Wisconsin-Madison, Madison, WI, USA
†IBM T. J. Watson Research Center, Hawthorne, NY, USA

Abstract—**While virtual machine (VM) migration is allowing data centers to rebalance workloads across physical machines, the promise of a maximally utilized infrastructure is yet to be realized. Part of the challenge is due to the inherent dependencies between VMs comprising a multi-tier application, which introduce complex load interactions between the underlying physical servers. For example, simply moving an overloaded VM to a (random) underloaded physical machine can inadvertently overload the network. We introduce *AppAware*—a novel, computationally efficient scheme for incorporating (1) inter-VM dependencies and (2) the underlying network topology into VM migration decisions. Using simulations, we show that our proposed method decreases network traffic by up to 81% compared to a well known alternative VM migration method that is not application-aware.**

I. Introduction

Virtualization has become an indispensable practice in the design and operation of modern data centers. By packaging applications into virtual machines (VMs), virtualization decouples the deployed applications from the physical servers—allowing administrators to migrate or reshuffle VMs to accommodate workload spikes and/or resource shortages [1]. When deploying multi-tier enterprise applications, straightforward reshuffling can introduce undesirable effects on the underlying infrastructure, due to the inherent coupling between VMs that comprise the multi-tier application. For example, migrating a database VM that communicates with an application server VM may increase traffic on certain links of the underlying data center network.

Most current approaches (e.g. [2], [3], [4], [5]) to VM migration primarily consider only server-side constraints (such as CPU, memory, and storage capacity) when choosing the new physical location for an overloaded VM, while interdependencies and inter-communication patterns among VMs are overlooked. A recent proposal [6] for VM placement considers network traffic among virtual machines but it does not attempt to simultaneously accommodate server resource constraints.

In this paper, we propose *application-aware* virtual machine migration, where the complete application context running on top of the VM is considered when choosing the most appropriate physical server to host that virtual machine. Our novel VM migration algorithm—called *AppAware*— takes into account the measured (real-time) communication dependencies among VMs of a multi-tier enterprise application, the underlying data center network topology, as well as the capacity limits of the physical servers of the data center. The goal is to minimize the data center network traffic *while satisfying all of the server-side constraints*. Our preliminary performance evaluation results show that the proposed method decreases network traffic by up to 81%, when compared to existing approaches.

This paper is organized as follows: in Section II, we describe a formalization of the joint network- and server-aware VM migration problem. In Section III, we propose efficient mechanisms to determine the mapping between virtual machines and physical servers while accounting for the application context of VMs. We evaluate our mechanisms using simulations in Section IV. We finally describe related work and our conclusions in Sections V and VI, respectively.

II. Problem formulation

Let the set of virtual machines in the data center be represented by $V = \{V_1, V_2, V_3, \ldots, V_n\}$ and the set of physical machines by $P = \{P_1, P_2, P_3, \ldots, P_m\}$. The set of overloaded virtual machines is defined as $O = \{V_1, V_2, V_3, \ldots, V_k\}$, such that $O \subset V$.

Assume that the data center administrator can obtain a *dependency graph,* $G = (V, E)$, where V is the set of virtual machines and E is the set of edges defined as $E = (V_i, V_j) : V_i, V_j \in Vc$, such that V_i and V_j are dependent with each other if any communication takes places between them. We further define traffic demand for each edge $W(V_i, V_j)$, which is directly proportional to the traffic transferred between V_i, V_j.

Next, we define $Load(V_i)$ as the vector of CPU, memory and storage load requirements of virtual machine V_i. Further, $Capacity(P_i)$ is defined as the available server-side capacity on physical machine P_i regarding its CPU, memory and storage. We define the cost of migrating virtual machines V_i, V_j to physical machines P_k, P_l as $Cost(V_i, P_k, V_j, P_l) = Distance(P_k, P_l) \times W(V_i, V_j)$. $Distance(P_k, P_l)$ is defined as the latency, delay or number of hops between physical machine P_k, P_l. Finally, we define

$$X_{ik} = \left\{ \begin{array}{ll} 1 & \text{if } V_i \text{ is assigned to } P_k \\ 0 & \text{otherwise} \end{array} \right\} \quad (1)$$

Hence, X_{ik} is 1, iff virtual machine V_i is placed on physical machine P_k. Similarly, we define $X_{ik}^{jl} = X_{ik} * X_{jl}$. This way, X_{ik}^{jl} is 1, iff virtual machines V_i, V_j are located on physical machines P_k, P_l, respectively. Now the optimization framework is defined as:

$$minimize \sum_{A} Cost(V_i, P_k, V_j, P_l) \times X_{ik}^{jl}, \quad (2)$$

where A is defined as $A = \{(i, j, k, l) | i < j, k < l, j < |V|, l < |P|\}$. Equation 2 tries to minimize the overhead of virtual machine migration, which will try to place the dependent virtual machines close to each other in the data center topology. Additionally the following constraints must be satisfied:

$$\sum_{k}^{|P|} X_{ik} = 1, \forall i \ , V_i \in O \quad (3)$$

This equation dictates that each overloaded virtual machine V_i must be placed on at least one physical machine P_k. As a sanity check to force the variable X_{ik}^{jl} to take a value of 1 when both virtual machine V_i, V_j are placed on physical machines P_k, P_l; thus, the following must be satisfied:

$$X_{ik} + X_{jl} \leq 1 + X_{ik}^{jl}, \forall (i, j, k, l) \ , V_i, V_j \in O, P_k, P_l \in P \quad (4)$$

Furthermore, we need to force the indicator variables to take binary values. This is important as virtual machine migration is itself binary in nature, meaning that any given virtual machine can only reside on one physical machine at a time. Thus,

$$X_{ik}^{jl}, Xik \in 0, 1 \quad (5)$$

To ensure that the total load on any physical machine is greater than or equal to its capacity, the following must hold:

$$\sum_{i}^{|V|} Load_i \times X_{ik} \leq Capacity_k, \forall k \ , P_k \in P \quad (6)$$

Finally, the following must also hold:

$$X_{ik} = x_{ik}, \quad (7)$$

where x_{ik} are existing assignments of non-overloaded VMs. This constraint states that the virtual machine to physical machine assignment is already fixed for those virtual machines that are not overloaded. Removing this constraint will reduce this optimization problem into an initial virtual machine placement problem.

III. Application-aware VM Migration

The problem of migrating VMs as formulated in Section II is a variant of the multiple knapsack problem, which is NP-complete [7]. Due to space considerations, we omit the proof in this paper. However, a simpler formulation that does not consider server-side constraints has been shown to be NP-complete in [6].

We propose *AppAware*, an approximate solution to this NP-complete problem. AppAware is a greedy algorithm with heuristics for assigning VMs to physical machines one at a time, while trying to minimize the cost that results from the mapping at each step. It takes into account both application dependencies and network topology. We describe AppAware over four stages: (1) the base algorithm, (2) the incorporation of application dependencies, (3) the extension with topological information and server load, (4) two extensions that further refine the selection process.

Base Algorithm. The base procedure for AppAware is shown in Algorithm 1. It accepts as input the dependency graph G , $W(V_i, V_j)$ as defined in Section II. The weights, W, are obtained from measuring the volume of traffic transferred between any two VMs. The algorithm also takes as input the network diameter $Distance_{max}$ of the network topology of physical machines, and an existing mapping $C : V \rightarrow P$, where $C(V_i)$ is the physical machine where the virtual machine V_i currently resides. Also, the *migration set* $M = (V_i, P_k)$ indicates that VM V_i should be migrated to physical machine P_k; it is initialized to $\emptyset$.

For each overloaded virtual machine, the total communication weight $TW(V_i)$ of all its incoming edges is computed as $TW(V_i) = \sum_{\forall V_j \in V} W(V_j, V_i), \forall V_i \in O$, where O denotes the set of overloaded VMs. The overloaded virtual machines are then sorted in descending order of their $TW(V_i)$. Moreover, for each overloaded virtual machine V_i, we define the set of its *dependent VMs*[1] as $D(V_i) = \{V_j : (V_j, V_i) \in E, V_j \notin O\}$, which is derived from the dependency graph G.

The base algorithm (Algorithm 1) finds the most suitable target physical machine and proceeds as follows: for each overloaded virtual machine V_i from the ordered set and for each physical machine P_k, the *migration impact factor* is calculated $Impact(P_k) = compute\_impact(V_i, P_k)$. The migration impact factor represents a measure of the overhead that will be incurred upon moving virtual machine V_i to physical machine P_k. The algorithm selects as candidate destination for migration the physical machine P_k that satisfies all server constraints (CPU, memory, disk capacity limits, software licensing constraints, pinning requirements, security zone limitations, etc.) and for which the migration impact is minimized. The migration decision procedure is repeated until a mapping has been identified for all overloaded virtual machines or no other mappings can be found.

Incorporating application dependencies. The pseudo-code of procedure $ComputeImpact\_Simple(V_i, P_k)$ incorporates information regarding application dependencies by calculating the $Cost(V_i, P_k, V_j, C(V_j))$ for every VM V_j that belongs to the set of application dependencies $D(V_i)$ of the overloaded VM V_i that is being considered for migration to candidate physical machine P_k. This cost depends on the weight of communication between the two VMs V_i and V_j (for example, volume of traffic exchanged between the VMs) and the network distance (e.g., latency, number of hops, etc.) between

[1]We consider the dependent VMs that are *not* overloaded, as these will not be relocated by the execution of the algorithm, thus the migration impact will be independent of future relocations (mappings).

Algorithm 1 Algorithm for virtual machine migration

```
INPUTS: Virtual machines (VMs) $V = \{V_1, V_2 \dots V_n\}$,
Physical machines (PMs) $P = \{P_1, P_2 \dots P_m\}$
    Overloaded VMs $O \subset V$, VM Load $Load(V_i)$, PM
capacity $Capacity(P_k)$
    Mappings $C : V_i \rightarrow P_k$, $V_i \in V$ and $P_k \in P$,
Dependency graph $G = (V, E)$ with weights $W$
    Network distance $Distance(P_k, P_l)$
for each VM $V_i$ in $O$
    $TW(V_i) = \sum_{\forall V_j \in V} W(V_i, V_j)$
end for
sort $V_i \in O$ in decreasing order of $TW(V_i)$
$M \rightarrow \emptyset$    // migration set
for each VM $V_i$ in $O$
    $Impact_{min} = \inf$, $Index_{min} = -1$
    for each PM $P_k$ in $P$
        if $check\_server\_constraints(V_i, P_k) == false$
            continue;
        end if
        $Impact(V_i, P_k) = compute\_impact(V_i, P_k)$
        if $Impact(V_i, P_k) < Impact_{min}$
            $Impact_{min} = Impact(V_i, P_k)$
            $Index_{min} = k$
        end if
    end for
    if $Index_{min} \neq -1$
        $M = M \cup (V_i \rightarrow P_k)$
        Update capacities of source & destination PMs
        $O = O - V_i$
    else
        No PM found for $V_i$; continue
    end if
end for
```

Algorithm 2 Heuristics for estimating migration impact

```
Require: ComputeImpact_simple($V_i, P_k$):
    $Impact = \frac{\sum_{\forall V_j \in D(V_i)} Cost(V_i, P_k, V_j, C(V_j))}{Distance_{max} \times W_{max}}$
    return $Impact$

Require: ComputeImpact_topo_load($V_i, P_k$):
    $Impact_{current} = ComputeImpact\_simple(V_i, P_k)$
    $Impact_{topology} = \frac{\sum_{\forall P_l \in P} Distance(P_k, P_l) \times Load(P_l)}{Distance_{max} \times |P|}$
    $Impact = Impact_{current} + Impact_{topology}$
    return $Impact$
```

the physical machine P_k that is a candidate destination for V_i and the current physical host $C(V_j)$ of the dependent VM V_j.

Adding topological information and server load. So far we do not consider the topology of the physical machines that are candidates for migration, or their load (CPU, memory, etc.). Such information can be useful for making migration decisions, as a physical server that is "close" (in the topological sense) to other lightly loaded servers would be preferred as a destination for a VM due to its potential for accommodating its dependent VMs to servers in close proximity. This intuition is captured in the method $ComputeImpact\_topo\_load(V_i, P_k)$ of Algorithm 2, which incorporates into the estimation of migration impact the weighted (by server load) node centrality of the candidate destination physical machines.

Variable	Distribution	Mean	Var.
Demand(VM)	Normal	0.4, 0.3, 0.2	0.3, 0.2, 0.1
Capacity(PM)	Normal	0.8, 0.6	0.4, 0.3, 0.2
Edges(VM comm. dependencies)	Normal, Exp, Uniform	0.8, 0.5, 0.2	0.4 0.2
# of VMs	5-12, 20-240		
fraction of overloaded VMs	0.1, 0.4		
# of PMs	7-10, 100		
Architecture(PM)	Tree, VL2		
Algorithms	AppAware Sandpiper		

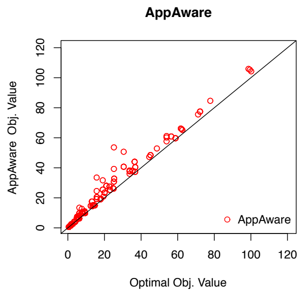
TABLE I
PARAMETERS FOR SIMULATIONS

Further extensions: iterative refinements. We have improved the above application- and network-aware algorithm by implementing two additional extensions that produce mappings that further minimize the traffic that is transferred by the data center network: the first extension involves the calculation of multiple values of the migration impact over several iterations of the algorithm. Each iteration tries to further minimize the migration impact upon the mapping that was calculated from the previous one and this extended algorithm stops either when the marginal reduction in the total migration impact is smaller than a predefined threshold, or a maximum number of iterations has been reached. The second extension improves upon the previous one by considering the expected migration impact of *future* mappings of other VMs, for a given candidate destination PM at each iteration.

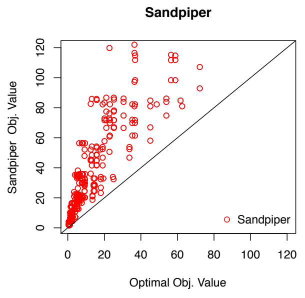
IV. EVALUATION

We evaluate the efficacy and scalability of application- and network-aware migration schemes on simulated data center environments of various sizes. We focus on the following questions: (1) How well do our proposed mechanisms select physical servers for migration? How close do they get to the optimal solution? (2) What is the performance of our mechanisms under varying demand and capacity patterns? (3) What is the impact of data center topology on different mechanisms?

We use numerical simulations to answer the above questions. Our intention is to compare the quality of the virtual to physical machine (VM-to-PM) mappings generated by AppAware against the one calculated with the optimal solution. Since the ultimate goal is to minimize traffic in the data center network, we use this metric as representative of the quality of VM-to-PM mappings, which, at the same time, have to satisfy all server-side constraints (i.e. total load of VMs assigned to a

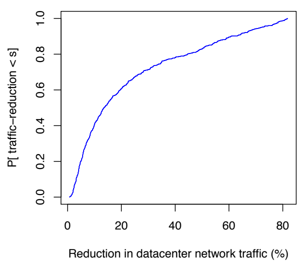


(a) AppAware

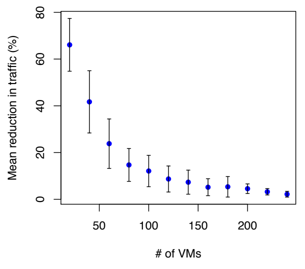


(b) Sandpiper

Fig. 1. Scatter plots for the final objective value achieved by (a) AppAware and (b) Sandpiper as compared with the optimal.



(a) CDF over all experiments



(b) Mean reduction in traffic

Fig. 2. Reduction in data center network traffic volume achieved using *AppAware* over Sandpiper: (a) CDF over large topologies (100 physical machines and up to 240 virtual machines) and (b) as a function of new VMs to be migrated.

PM should be less or equal to the capacity of that PM). That said, finding the optimal solution for large physical and virtual topologies (i.e., topologies with a large number of physical and virtual machines) is infeasible as the solver (CPLEX) that was used for computing the optimal solution runs out of memory. In the following, we report the results obtained by solving the optimal solution only for small networks, while for large topologies, we compare AppAware with a previously proposed migration scheme called Sandpiper [3].

Simulation Setup. We generated a range of scenarios for our simulations that includes varying the number of physical machines for the data center network topologies, varying the number of virtual machines and the interdependencies amongst them that represents various application topologies, as well as varying the load characteristics of the PMs and VMs. The simulation parameters are noted in Table I. The numbers of physical and virtual machines are limited to 10 and 12 respectively for small topologies, for which we manage to calculate the optimal mappings. For larger topologies (for which we only compare AppAware and Sandpiper), we fix the number of physical machines to 100, while we vary the number of VMs from 20 to 240, in steps of 20. In our simulation scenarios, the parameter "fraction of overloaded VMs" denotes the percentage of the VMs that would require migration from their initial mapping to some other physical machines. Two cases where investigated, 10% and 40%. The distance between physical machines is dependent on the architecture and is computed as shown in [6]. We use *Tree* architecture [8] to compute the distances between physical machines, unless stated otherwise. Overall, we ran simulated experiments for 866 scenarios for small topologies and 3132 scenarios for the large ones, in various combinations of the values of the parameters of Table I.

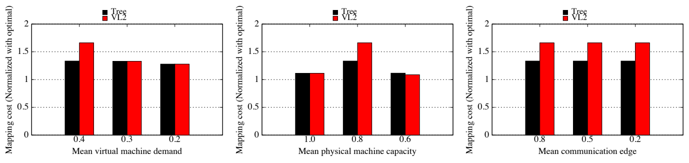
Comparison Metrics. As mentioned above, the metric used for the evaluation of AppAware is the total traffic volume that is transported by the data center network once all overloaded VMs have been assigned to physical machines using one of the methods. Both the absolute form of the metric (objective value), as well as the relative reduction in traffic that is achieved when comparing AppAware with Sandpiper are used.

Results. Figure 1 shows the performance of AppAware (includes all extensions) and Sandpiper as compared to the optimal solution, for a large number of experiments with various configurations. As shown in the Figure, the AppAware algorithm manages to find mappings that are very close to those computed optimally (the "dots" fall very close to the optimal $y = x$ line in the Figure). On the other hand, Sandpiper, which does not take into account application dependencies, performs much worse, as evident by how far from the optimal its mappings fall.

Figure 2(a) shows results with experiments with much larger topologies and number of VMs, for which computing the optimal solution was not feasible. It plots the cumulative distribution function of relative reduction in traffic achieved by AppAware over the Sandpiper algorithm, for 3132 simulated experiments. As it can be seen from the Figure, AppAware consistently outperformed Sandpiper in these larger configurations, producing mappings that decreased traffic volume transported by the network by up to 81%.

In Figure 2(b), the mean and standard deviation of the reduction in network traffic achieved by using AppAware over Sandpiper is plotted as a function of the number of *new* virtual machines that are assigned to the physical servers in the data center. This number ranges from 20 to 240, while the number of physical machines is always set to 100 for all 3132 simulations that were ran. Note that the simulation assumes an existing load on physical machines following the Capacity(PM) distribution and each new VM requires resources following the distribution Demand(VM)—both described in Table I. As it can be seen from the Figure, the largest reduction in traffic is achieved when the number of VMs to be mapped is small relative to the PMs in the data center. This is due to the fact that there are more possibilities for VM-to-PM mappings that minimize network traffic, a fact that is exploited by the AppAware algorithm, and in contrast to Sandpiper, which does not take this into account.

Further, we evaluate the performance of our application and topology aware heuristic with varying data center architec-



(a) Demand (b) Capacity (c) Communication

Fig. 3. Impact of (a) VM demand, (b) PM capacity and (c) communication frequency on the performance of AppAware. algorithm. For all scenarios, the number of physical machines and number of virtual machines are set to 10. (a) Capacity of physical machines is set to 0.8 and the mean communication edge is 0.5 (b) Demand of virtual machine is set to 0.4 and the mean communication edge weight is fixed at 0.5 (c) Demand and physical capacity of virtual and physical machines is set at 0.4 and 0.8 respectively.

tures. Figure 3 shows the performance of our heuristic with the optimal solution on VL2 [9] and Tree [8] architectures. As shown in the figure, performance gap between optimal and AppAware is larger for VL2 than for Tree architecture, which can be attributed to the design of VL2, in which a suboptimal physical machine selection is more heavily penalized (many more hops in heuristic than in optimal). Also, the performance gap is larger for less congested demand patterns, as there are more choices for placement.

V. Related Work

Virtual machine migration and its application to data center server farms has been a major focus of research recently [3], [4], [10], [11]. Wood et al. propose *Sandpiper* that uses black- and gray-box techniques like monitoring memory, CPU and network utilization to detect hotspots and mitigate them by migrating the VMs to a suitable physical server [3]. In [4], the authors outline a scheme for distributing resources among VMs on the same physical server. They consider the relative priorities of different VMs and their inter-dependencies while allocating resources to them. The work in [5] considers the load on the communication paths that connect servers to shared network storage. None of the these works considers the combined effects of application dependencies and network topology for minimizing traffic in the data center network. The work closest to our efforts is [6], where the authors formulate a LP for computing virtual-to-physical machine mappings in the presence of application dependencies. However, their work does not incorporate server-side constraints. We propose practical heuristics for VM placement that simultaneously satisfy server-side limitations.

Research in the area of data center network architecture and routing seeks to maximize flexibility and efficiency of underlying resources. Representative works include Monsoon [12], Fat-tree [8], VL2 [9], Portland [13], DCell [14] and BCube [15]. Our scheme is complementary to such network designs; as shown in our evaluation, it can provide gains for such different architectures.

VI. Conclusion

Multi-tier enterprise applications that run across multiple VMs in data centers may exhibit strong communication dependencies. Such dependencies need to be taken into account during the migration process to ensure efficient network resource utilization and minimize redundant cross-traffic between switches in the data center. We propose an efficient mechanism for application-aware VM migration and demonstrate through numerical simulations that it minimizes network traffic inside data centers. We are in the process of implementing and evaluating our scheme on a large-scale data center testbed.

References

[1] S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. P. Pazel, J. Pershing, and B. Rochwerger, "Oceano - sla based management of a computing utility," in *In Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, 2001, pp. 855–868.

[2] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM, 2007, pp. 331–340.

[3] T. Wood, P. Shenoy, and Arun, "Black-box and gray-box strategies for virtual machine migration," in *NSDI '07*, pp. 229–242. [Online]. Available: http://www.usenix.org/events/nsdi07/tech/wood.html

[4] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," in *EuroSys*, 2007, pp. 289–302.

[5] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–12.

[6] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *In Proc. INFOCOM*, 2010.

[7] M. J. Magazine and M.-S. Chern, "A note on approximation schemes for multidimensional knapsack problems," *Mathematics of Operations Research*, vol. 9, no. 2, pp. 244–247, 1984.

[8] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[9] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *In Proceedings of ACM SIGCOMM*. ACM, 2009, pp. 75–86.

[10] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*. USENIX Association, 2005.

[11] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *ATEC'05: Proceedings of the USENIX Annual Technical Conference*. USENIX Association, 2005. [Online]. Available: http://dx.doi.org/10.1145/323627.323629

[12] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Towards a next generation data center architecture: scalability and commoditization," in *PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*. New York, NY, USA: ACM, 2008, pp. 57–62.

[13] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *In Proceedings of ACM SIGCOMM*. ACM, 2008, pp. 75–86.

[14] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *In Proceedings of ACM SIGCOMM*. New York, NY, USA: ACM, 2008, pp. 75–86.

[15] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," in *In SIGCOMM*, 2009.