NetDEO: Automating Network Design, Evolution, and Optimization

Zhenyu Wu*, Yueping Zhang†, Vishal Singh†, Guofei Jiang†, and Haining Wang*
* Department of Computer Science, College of William and Mary, Williamsburg, VA 23187, USA
E-mail: {adamwu, hnw}@cs.wm.edu
† NEC Laboratories America, Inc., Princeton, NJ 08540, USA
E-mail: {yueping, vishal, gfj}@nec-labs.com

***Abstract*—With the ever-increasing number and complexity of applications deployed in data centers, the underlying network infrastructure can no longer sustain such a trend and exhibits several problems, such as resource fragmentation and low bi-section bandwidth. In pursuit of a real-world applicable data center network (DCN) optimization approach that continuously maintains balanced network performance with high cost effectiveness, we design a topology independent resource allocation and optimization approach, NetDEO. Based on a swarm intelligence optimization model, NetDEO improves the scalability of the DCN by relocating virtual machines (VMs) and matching resource demand and availability. NetDEO is capable of (1) incrementally optimizing an existing VM placement in a data center; (2) deriving optimal deployment plans for newly added VMs; and (3) providing hardware upgrade suggestions and allowing the DCN to evolve as the workload changes over time. We evaluate the performance of NetDEO using realistic workload traces and simulated large-scale DCN under various topologies.**

I. Introduction

Today's data center networks (DCNs) are continuously evolving because of two major factors: *architectural upgrade*, such as network topology expansion and new server deployment, driven by increasing application demand, and *workload dynamics*, such traffic patterns changes and application evolutions, introduced by tenant services running on top. This is especially the case in the Cloud environment with virtualized infrastructures, where users continuously join/leave the system and client instances (i.e., virtual machines) are dynamically created and terminated. Driven by this trend, the quest for a highly scalable and efficient DCN has led to much recent progresses [1–10].

Specifically, one school of research focuses on designing new DCN architectures to achieve high-bandwidth all-to-all connectivity (or 1:1 over-subscription) [1, 3], appealing scalability [4, 5, 7], ideal agility [3, 8], and desirable topology flexibility [2, 6, 9, 10]. However, these schemes require fundamental changes of today's network architectures and/or modifications of hardware equipments, and therefore may encounter nontrivial deployment barrier. In contrast, another class of approaches, such as TVMPP [11], CPA [12], and RAP [13], keep the existing network architecture and routing protocol intact, and instead, aim at reducing network bandwidth demand by optimizing placement of end-nodes (which can be physical or virtual machines[1]). In this paper, we choose to explore the latter research direction due to its low up-front cost and immediate applicability.

978-1-4673-1298-1/12/$31.00 © 2012 IEEE

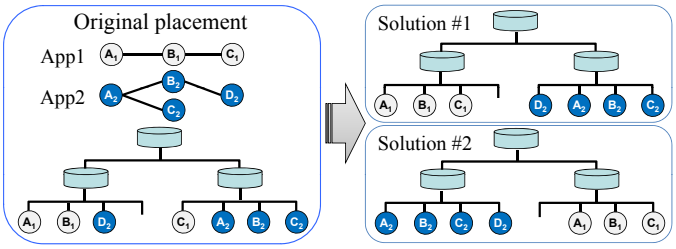


Fig. 1. An example of two equivalent solutions with different transition costs

Aiming to develop a real-world applicable optimization solution that can continuously maintain balanced DCN network performance with high cost effectiveness, we find that existing placement optimization approaches insufficient for fulfilling our goal. In particular, they lack in three key attributes, *feasibility*, *flexibility*, and *expandability*, which we believe are crucial for effective DCN management.

Feasibility: Data center maintenance is not a one-time task, but a series of incremental optimizations performed over time. When optimizing an existing data center configuration, a large number of equally optimal placement solutions exist due to the homogeneous computing and network resources in the DCN. However, the transition costs to these target placements are significantly different, and may even render certain optimal solutions impossible to realize in practice. Targeting only initial deployment optimizations, previous approaches [11, 12, 14] disregard the preexisting configuration, and solve the problem from scratch. And thus, they are very likely to reach a very expensive solution in terms of transition cost. To better understand this, consider an illustration in Figure 1, where two applications `App1` and `App2` are deployed in a two-level tree network. The existing placement of the two applications on the left is sub-optimal, since traffic flows (B_1, C_1) and (B_2, D_2) have to cross subnets. Two optimal solutions are given on the right, both placing components of the same application in the same subnets, thereby eliminating inter-subnet traffic. However, while both solutions are equivalent

[1]Without loss of generality, we consider, in the rest of the paper, virtualized environments where virtual machines are the end-nodes.

in terms of network optimality, the transition cost (i.e., the number of VM migrations) from the original configuration to the final configuration associated with the two solutions are quite different. Specifically, Solution 1 involves only two VM (i.e., D_2 and C_1) migrations, while Solution 2 requires five.

Flexibility: Besides placement optimality, DCN operators are also concerned with practical factors such as transition cost, time, and other administrative constraints. In practice, it is not uncommon for the DCN operators to compromise optimality for these practical considerations. Modeling the network optimization as a rigid mathematical problem, previous approaches could only produce a single "best" solution with no regards for applicability. However, they have left the large space of less-optimal-but-more-applicable solutions unexplored.

Expandability: DCNs evolve over time as the operators have budget to opt for upgrading server and/or network capacity. Moreover, in many cases (as we show in Section V), a complicated optimization solution involving significant server replacement and traffic re-routing may be greatly simplified by upgrading only a few resources (e.g., server and network). This situation leads to many questions that cannot be answered by previous approaches, such as the sets of servers and switches to upgrade and the new capacities to upgrade to.

Motivated by the above observations, we present in this paper a DCN optimization framework called NetDEO. NetDEO facilitates the three key attributes by employing a swarm intelligence [15] optimization algorithm based on modified simulated annealing [16]. First, thanks to the metaheuristic[2] nature of swarm intelligence algorithms, NetDEO is capable of performing *feasible* incremental optimizations—finding efficient optimization solutions for preexisting data center configurations. Second, NetDEO explicitly takes the maintenance flexibility into account by identifying a set of optimization solutions with different benefit-cost characteristics for each problem instance. And finally, NetDEO acknowledges the DCN expandability and explores these new degrees of freedom in a controlled manner, providing the DCN operators with customizable network and server system upgrade suggestions according to their budget.

We evaluate NetDEO using production server traces of multiple transaction systems and simulated large DCN in three different network topologies: non-homogeneous tree, FatTree, and BCube. Our experiments show that, NetDEO significantly improves both solution quality (in terms of transition cost) and running time for incremental optimizations, the key tasks for DCN maintenance. And for initial deployment optimizations, NetDEO achieves comparable or improved solution quality and running time compared with existing approaches. In addition, NetDEO could also provide efficient DCN upgrade suggestions that simplify the optimization process.

The remainder of this paper is structured as follows. In Section II, we give a brief review of related work. In Section III, we present the problem formulation and analysis. In Section IV, we present NetDEO design and algorithm details. In Section V, we evaluate the performance of NetDEO. In Section VI, we conclude this paper and point out directions for future work.

[2]Metaheuristic designates a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality [17].

II. Related Work

Many solutions have been proposed to solve the network optimization problem, falling into two major categories—architectural revolution and placement optimization.

Architectural revolution mainly focuses on designing new network architectures to address various issue identified for today's tree-like networks, such as low bisection bandwidth, low agility, and resource fragmentation. An early theme of this research direction is to provide high-bandwidth connectivity for all pairs of servers [1, 3]. This objective is necessary for certain traffic patterns that involve high-throughput all-to-all communication, which, as recently pointed out by [2], is, in the least, not ubiquitous in today's data centers. Driven by this motivation, several new architectures have been proposed to achieve on-demand connectivity and bandwidth using optical switching [2, 9, 18], wireless networks [6], and VLAN [19]. In addition to bandwidth provisioning, another line of research focuses on providing high scalability [4, 5, 7] and agility [8].

Placement optimization, in contrast to architectural revolution, keeps existing network architecture and routing protocol intact. Instead, it aims at eliminating network bottleneck via optimizing placement of computing service nodes—to organize the computing services so that their computing and communication demands are satisfied by the most suitable hardware resource available. The service placement optimization problem belongs to the class of quadratic assignment problem (QAP), which is one of the hardest problems in the NP-hard class, and is even hard to approximate [20, 21]. As a result, a variety of heuristics based optimization models and problem-solving techniques have been employed. In particular, TVMPP [11] and Starling [22] establish their optimization model based on network communication cost (i.e., traffic volume, link bandwidth, route distance, etc.). CPA [12], RAP [13] and [14] model both network communication and other computing resources, such as processor, storage demand, and availabilities. To solve the optimization problem, CPA and TVMPP transform the original QAP problem into a combination of NP problems (such as Stable Marriage and minimal K-cuts), and solve them using known approximation algorithms. In [14] and Starling, centralized and decentralized heuristic algorithms are employed, respectively. RAP takes the linear programming approach. In addition, recent efforts [23, 24] also take the bandwidth constraints into account and propose heuristics algorithms to solve the application placement problem in a tree or generalized hierarchical network topologies.

III. Problem Definition

We study the placement optimization of a set of service nodes $\{n_1, n_2, ..., n_N\}$ on a collection of networked server systems $\{s_1, s_2, ..., s_M\}$, where N and M are respectively the

total number of service nodes and servers in the system. Each server s_i has a service capacity of C_i, which is a composite metric of its processing, memory, and storage resources. Correspondingly, each service node n_x has a resource requirement U_x, representing its consumption of the aforementioned resources. Thus, a server can host many service nodes as long as the sum of resource requirements of all deployed service nodes does not exceed server capacity C_i. We remark that, in practice, the number of VMs that can co-exist in a given server is also affected by other factors, such as the correlation between individual VMs' workloads and resource sharing characteristics [25]. In the paper, we do not consider these factors for ease of presentation. However, the proposed approach can be easily extended to incorporate more complex resource models.

We denote by p_{ij} the fixed path between servers s_i and s_j and $p_{ij} = \emptyset$ if the two servers are unreachable to each other. Node pair n_x and n_y respectively deployed on servers s_i and s_j communicate via route p_{ij} at traffic rate T_{xy}. Each route p_{ij} consists of a set of link segments, which may partially overlap with those of other routes. We define the length of the route as the number of link segments, i.e., $D_{ij} = |p_{ij}|$. Each link segment l_k has a channel capacity B_k and a *reliability* factor R_k, the latter defined as the complement of the packet loss rate of the link, which is readily available at switches via SNMP. The reliability R_{ij} of a route p_{ij} is then defined as the product of the reliability factors of all links that constitute the route: $R_{ij} = \prod_{k \in p_{ij}} R_k$.

We define the *traffic stress* between two communicating VMs n_x and n_y as the product of their traffic rate, route length, and inverse route reliability:

$$TStress(n_x, n_y) = T_{xy} \times D_{ij} \times R_{ij}^{-1} \ , \tag{1}$$

where n_x and n_y respectively reside on s_i and s_j. The stress value represents the traffic condition between two service nodes – the higher the value, the worse the traffic condition. For example, for two pairs of nodes with identical traffic rate, the node pair that uses the longer or less reliable route has a worse traffic condition, which in turn is reflected in a higher stress value. Notably, when route distance is zero, that is, when two communicating nodes are deployed on the same server, their traffic stress is always zero regardless of their traffic rate. This is consistent with the fact that communication between VMs residing on the same server actually becomes internal memory swapping, and no longer affects the network.

The traffic stress of a service node is defined as the quadratic mean (also called root-mean-square) of the traffic stresses between the node and all its communicating peers:

$$NodeStress(n_x) = \sqrt{\frac{1}{N_x} \sum_{y=1}^{N_x} TStress(n_x, n_y)^2} \ , \tag{2}$$

where N_x is the number of service nodes communicating with node n_x. Note that we use the quadratic mean instead of a simple average in flavor of a more balanced traffic load distribution. Accordingly, the traffic stress of the whole system under a given service node placement scheme π is defined as the quadratic mean of all service nodes' traffic stresses:

$$\begin{aligned} SysStress(\pi) &= \sqrt{\frac{1}{N} \sum_{x=1}^{N} NodeStress(n_x)^2} \\ &= \sqrt{\frac{1}{N} \sum_{x=1}^{N} \sum_{y=1}^{N} TStress(n_x, n_y)^2} \ , \end{aligned} \tag{3}$$

where N is the total number of service nodes.

The optimization objective is to find service node placement schemes that minimize the traffic stress values between all communicating service nodes over the whole system, as expressed below:

$$\arg\min_{\pi \in \Pi} \ SysStress(\pi) \ , \tag{4}$$

where Π represents all possible service node placement schemes, subject to the server capacity and link capacity constraints:

$$\begin{cases} C_i \geq \sum\limits_{x \in \alpha_i} U_x & \text{(for each server } s_i\text{)} \\ \\ B_k \geq \sum\limits_{y \in \beta_k} T_y & \text{(for each link } l_j\text{)} \end{cases} , \tag{5}$$

where α_i is the set of service nodes deployed on server s_i, and β_k is the set of flows that pass through link l_k. In other words, the sum of resource requirement of all nodes deployed on any server must not exceed the capacity of that server; and the sum of traffic rates of all flows on any link segment must not exceed the channel capacity B_k of that link.

Note that as a variation of the QAP, our definition of system traffic stress has two major differences from the typical definition of a QAP cost function:

$$Cost(\pi) = \sum_{x=0}^{n} \sum_{y=0}^{n} W(x, y) D(L_x, L_y) \ , \tag{6}$$

where $W(x, y)$ is the "weight" between two facilities x and y, and $D(L_x, L_y)$ is the "distance" between the locations of the two facilities L_x and L_y. First, quadratic mean is employed for a more balanced system-wide network load. Second, in addition to "weight" and "distance," which respectively correspond to T_{xy} and D_{ij} in (1), we also add the reliability factor R_{ij}, a microscopic variable that reflects the quality of a placement from the physical hardware point-of-view. By introducing this realistic parameter, our definition more accurately reflects the performance situation of a network.

IV. NetDEO Design

In this section, we start with a discussion on the design intuition of the NetDEO algorithm, then we introduce the pseudo-physical optimization model and then present details of the optimization algorithm.

A. Optimization Algorithm Selection

To efficiently derive solutions that are cost effective, flexible, and expandable, our search algorithm should possess two important properties—*incremental* and *exploratory*.

The *incremental* property stems from the requirement of generating cost effective solutions. Due to the massive infrastructure in a data center, the cost of conducting significant global changes, such as deploying all service nodes from scratch, are prohibitively high. As a result, any feasible optimization solution must make only incremental changes. And consequently, our optimization algorithm must accept preexisting configurations as the basis for improved solutions. Meanwhile, the *exploratory* property is necessitated by flexible and expandable DCN maintenance. In order to discover worthwhile candidate solutions, our algorithm must explore into the solution space. Because computing and network resources in DCN are highly homogenous, the solution space is often too large to completely enumerate. And therefore, the most efficient solution is to non-deterministically sample alternative solutions in an unbiased manner.

Guided by these design insights, we determine that swarm-intelligence (SI) optimization algorithms are a good fit to solve the DCN optimization problem. First, SI algorithms are metaheuristic, i.e. optimizations are performed iteratively by gradually improving a candidate solution. And as a consequence, the *incremental* property is implicit. Second, most SI algorithms are stochastic, i.e., the search space is explored in a randomized fashion. This characteristic meets our requirement of efficient alternative solution exploration, and thus satisfies the *exploratory* property.

B. Pseudo-physical SI model

Inspired by the principle of *minimum total potential energy*, a fundamental physics concept, we devise a pseudo-physical SI model for the service node placement optimization problem.

Search space and agents: The networked servers are modeled as the search space, in which each service node represents a search agent. According to the problem definition, a service node can be placed only on a server but not in-between servers, and thus the search space is discrete.

Objective function: The objective function in our model is the system potential energy, the most important component in our model. The system potential energy plays two key roles: to evaluate the solution quality, which corresponds to the system traffic stress in equation (3), and to enforce the optimization constraints given in equation (5). However, instead of expressing the constraints in a rigid binary form, we use a set of barrier functions, which incorporate flexibility into their expressions to better support the *exploratory* property.

First, following the definition of service node traffic stress in equation (2), we define the *traffic potential* of a search agent (ie. service node) n_x in a similar fashion:

$$NodePotential(n_x) = \sqrt{\frac{1}{N_x}\sum_{y=1}^{N_x} TPotential(n_x, n_y)^2} \ ,$$

where $TPotential(n_x, n_y)$ is the traffic potential between a pair of communicating service nodes n_x and n_y:

$$TPotential(n_x, n_y) = T_{xy} \times D_{ij} \times U_{ij}^{-1} \ . \qquad (7)$$

Compared with equation (1), the sole difference of equation (7) is that the route reliability factor is replaced by the route *usability* factor, which is defined as the product of the usability factors of all links that constitute the route, $U_{ij} = \prod_{k \in p_{ij}} U_k$.

The usability factor U_k is a metric we introduced to both evaluate the optimization objective, and reflect the traffic load constraint on a link segment. Similar to reliability, usability is inversely correlated to the traffic load. But unlike reliability, usability decreases faster when the load approaches a per-determined "maximum" value and becomes zero when the load exceeds the "maximum" threshold, signifying that the link is carrying infeasible traffic loads. This characteristic allows us to flexibly yet precisely control the exploration of the problem search space. We use the following split function to calculate the inverse usability value of link l_k:

$$InvU(l_k) = \begin{cases} 1 & (\theta_k \le \theta_T) \\ 1 + \tan(\frac{\pi}{2}\frac{\theta_k - \theta_T}{\theta_{Max} - \theta_T}) & (\theta_T < \theta_k < \theta_{Max}) \\ \infty & (\theta_k \ge \theta_{Max}) \end{cases} .$$

where θ_k is the load factor (ie. load over capacity) of link l_k, θ_T is a threshold ($\theta_T \le 1$), and θ_{Max} is the maximum load factor. See Figure 2 for an illustration of this step function.

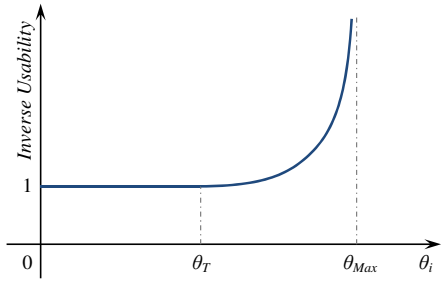


Fig. 2. The Inverse Usability Function

Analogous to the load constraint on each link, each server also has constraint on the number of deployed nodes. To express this constraint, we define *repulsive potential* between a server and all nodes deployed on it. The strength of the repulsive potential depends on the load of the server: under low load, the repulsion is zero or very small; as the load approaches the constraint, the repulsion increases rapidly; and when the constraint is violated, the repulsion becomes infinitely large. The computation of the repulsive potential is similarly defined as the inverse usability value:

$$RPotential(s_i) = \begin{cases} 0 & (\gamma_i \le \gamma_T) \\ \tan(\frac{\pi}{2}\frac{\gamma_i - \gamma_T}{\gamma_{Max} - \gamma_T}) & (\gamma_T < \gamma_i < \gamma_{Max}) \\ \infty & (\gamma_i \ge \gamma_{Max}) \end{cases} ,$$

where γ_i is the load factor of the server, i.e. sum of all resource requirements over the service capacity, γ_T is a load factor threshold ($\gamma_T \le 1$), and γ_{Max} is the maximum load factor.

Algorithm 1 The Optimization Algorithm

```
/* Input: π – current node placement scheme
          E_T0 – initial system thermal energy
          Budget – number of iterations to run */
E_P:= SystemE_P(π); E_T:= E_T0; E_Free:= 0; Cnt:= 0;
repeat
    /* Step 1: Select the node to move */
    MoveNode:= GetMoveNode();
    /* Step 2: Evaluate trial moves */
    CurServer:= CurrentServer(MoveNode);
    for each Server_i ≠ CurServer do
        /* Step 2.1: Try 1-displacement neighborhood */
        π':= π + Place MoveNode onto Server_i;
        ΔE_P:= E_P - SystemE_P(π');
        if AcceptMove(ΔE_P, E_T) then break
        /* Step 2.2: Try 2-displacement neighborhoods */
        for each Node_j on Server_i do
            π'':= π' + Place Node_j onto CurServer;
            ΔE_P:= E_P - SystemE_P(π'');
            if AcceptMove(ΔE_P, E_T) then break(2)
        end for
    end for
    /* Step 3: Handle accepted move */
    if Accepted move then
        π:= π'';
        E_Free:= E_Free + ΔE_P; // Energy pooling
        E_T:= E_T - ActFun_1(E_T); // and dissipation
    end if
    /* Step 4: Thermal Energy Conversion */
    ΔE_T:= ActFun_2(E_Free)−1;
    E_Free:= E_Free − ΔE_T; E_T:= E_T + ΔE_T;
    Increment Cnt;
until Cnt ≥ Budget;
```

Algorithm 2 Selecting a Service Node to Move

```
function GETMOVENODE
    P_Max:= 0;
    for each Node_i do
        ProbScore:= NodeEp(Node_i);
        Increment P_Max by ProbScore;
        MoveScore_i:= P_Max;
    end for
    P_Rand:= Random(0, P_Max);
    for each Node_i do
        if MoveScore_i ≥ P_Rand then return Node_i
    end for
end function
```

Algorithm 3 Decide Whether to Accept a Move

```
function ACCEPTMOVE(ΔE_P, E_T)
    /* Input: ΔE_P – Reduction of potential energy
              E_T – System thermal energy */
    /* Obtain the greediness control value */
    G:= ActFun_3(E_T);
    /* Convert ΔE_P to acceptance probability */
    AccptProb:= ActFun_4(ΔE_P, G);
    return AcceptProb ≥ Random(0, 1);
end function
```

The *potential energy* of a service node n_x on server s_i is thus defined as the summation of the service node's traffic potential and the server's repulsive potential,

$$NodeE_P(n_x) = TPotential(n_x) + RPotential(s_i)\ . \quad (8)$$

The service node potential energy indicates the quality of the node's placement – the higher the potential energy, the worse the placement. In particular, on servers with resource utilization ratios below θ_T, the goodness of nodes' placement is solely determined by the optimization objective function. On the other hand, when a node placement causes the server resource utilization to approach or exceed load factor θ_{Max}, the repulsive potential becomes the dominant factor. A potential energy of infinity signifies an infeasible placement.

Finally, the potential energy of the entire system under placement π is defined as the quadratic mean of all service nodes' potential energy:

$$SystemE_P(\pi) = \sqrt{\frac{1}{N}\sum_{x=1}^{N} NodeE_P(n_x)^2}\ . \quad (9)$$

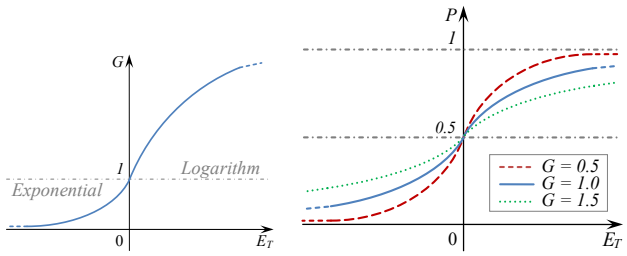
C. Optimization Algorithm

We choose Simulated Annealing (SA) as the design basis of our optimization algorithm. SA is an SI optimization algorithm that models the annealing process of metallurgy. It is designed to perform stochastic search on discrete search space and therefore is particularly suitable for solving QAP. We customize the SA by introducing an adaptive greediness control scheme with our novel cooling scheduling algorithm, which accelerates the annealing process and achieves much faster convergence and better optimization than those of the classic SA algorithms.

The iterative optimization algorithm of NetDEO is shown as pseudo-code in Algorithm 1. Similar to the classic SA algorithms, for each iteration NetDEO performs randomized exploration of $(1, 2)$-displacement neighborhood configurations (i.e., configurations that require only one or two node migrations to reach), and probabilistically accepts the new configuration. The randomized exploration is implemented in the `GetMoveNode` function, shown as pseudo-code in Algorithm 2, and the probabilistic configuration acceptance is implemented in the `AcceptMove` function shown as pseudo-code in Algorithm 3. However, the similarity between the classic SA algorithms and NetDEO stops as we dive deeper into the detailed design.

First, instead of exploring the neighborhood configurations completely randomly, the `GetMoveNode` function performs *controlled stochastic selection*. The potential energy of each node corresponds to a probability score, which represents the fair share of probability a node is chosen to move over all other nodes. The rational behind this design is to encourage migration of nodes in relatively worse configurations, and thus yielding faster convergence.

Second, unlike classic SA algorithms, which greedily accept all improved configurations while probabilistically accept degraded ones, the `AcceptMove` function performs *full-range*



(a) E_T vs. G (b) Effect of G on ΔE_T vs. P

Fig. 3. The Greediness Control Mechanism

probabilistic acceptance. In other words, the probabilistic behavior covers both improved and degraded configurations—improved configurations are always preferred to degraded ones, and the more improvement, the higher the preference. This design helps the algorithm to better differentiate the quality of sampled solutions, while still preserving the degree of randomness in the stochastic search.

Notably, the implementation of the `AcceptMove` function deserves some explanation. The probability of accepting a configuration is derived from two factors: the improvement of objective function (ΔE_P) and the system thermal energy (E_T). First, E_T is "compressed" by an activation function `ActFun_3()` into a positively correlated greediness control value G with range $(0, \infty)$. And then, ΔE_P and G are passed as parameters to another activation function `ActFun_4()`, which transforms ΔE_P into the acceptance probability. The greediness control value G is used as a "slope" value to control the behavior of `ActFun_4()`. To help comprehension, we have visualized the two activation functions and the effect of G on `ActFun_4()` in Figure 3—a high thermal energy yields to a large G value, which in turn results in a flatter curve of `ActFun_4()`, causing the the acceptance probability to be less sensitive (greedy) to ΔE_P; On the other hand, a low thermal energy yields to a small G value, a steeper curve of `ActFun_4()`, and thus a more sensitive (greedy) acceptance probability to ΔE_P.

The last, but not the least difference between NetDEO and the classic SA algorithms lies in the cooling schedule. Conceptually similar to *temperature* in SA algorithms, the system thermal energy is inversely correlated with the greediness of the algorithm, and it gradually decreases as the iteration progresses, allowing the system to converge to a low energy state. However, our algorithm do not control system thermal energy using deterministic cooling schedules as classic SA does for temperature. Inspired by the laws of physics, we associate the system thermal energy with the system potential energy (defined in equation (9)), and introduce a novel "conversion-and-dissipation" mechanism for an *adaptive system cooling*.

Shown in step 3 and 4 of Algorithm 1, we set up an energy conversion rule, storing the potential energy released during iterative optimization in an energy pool, which then gradually converts its energy into thermal energy. In addition, we consider our pseudo-physical system a black body, which continuously releases a portion of thermal energy proportional to its current thermal energy. The energy conversion and dissipation provide a proportional feedback mechanism from the optimization procedure (i.e. the reduction of potential energy) to the system greediness control (i.e. the system thermal energy), and allow the system to self-regulate the cooling process and to promote faster convergence.

TABLE I
LIST OF ACTIVATION FUNCTIONS

Name	Description	Slope
ActFun_1 ActFun_2 ActFun_3	$\begin{cases}\exp(x \cdot Slope + 1) + 1 & (x \leq 0)\\ \log(x \cdot Slope) & (x > 0)\end{cases}$	0.1 0.01 0.2
ActFun_4	$(1 + \exp(-x \cdot Slope))^{-1}$	G

The four activation functions used in Algorithms 1 and 3 and their parameters are listed in Table I. The slope parameters of activation functions 1–3 are responsible of controlling the convergence behavior of the algorithm. Their values are obtained from a quick human-supervise training, and we believe they are good for solving general problems. However, they are by no means "the optimal" values. In fact, we believe that there may not be a single set of "good-for-all" parameters – different problem setup and optimization objective may have their unique optimal parameter set. We provide a general guideline of determining the range and relationship of these parameters as the following:

- The slope of `ActFun_1()` determines the thermal dissipation rate – the higher the value, the faster the thermal energy is dissipated to zero. This value functions similar to the cooling schedule of the classic SA algorithm;
- The slope of `ActFun_2()` determines the "free energy" conversion rate – the higher the value, the faster the free energy is converted to thermal energy, in other words, the stronger the feedback of current optimization progress to the cooling schedule. This value should be smaller than the thermal dissipation rate, otherwise the thermal energy would increase too fast during optimization and thus slow down the convergence;
- The slope of `ActFun_3()` determines the sensitivity of greediness to the thermal energy – the higher the value, the more sensitive. This value should be set within the range of one order of magnitude of the thermal dissipation rate.

V. EVALUATION

In this section, we present a comprehensive evaluation of the NetDEO using a realistic setup and four experiments covering different usage scenarios.

A. Setup and Evaluation Methodology

Data Center Network: We simulate data centers of over 1000 servers with heterogeneous resource capacities, and in three different topologies. We first randomly generate 1080 servers with three levels of resource capacities—50% "main-stream" servers have a "standard" capacity, capable of hosting 3 to 6 VMs, 30% "upgraded" servers have a capacity doubles the

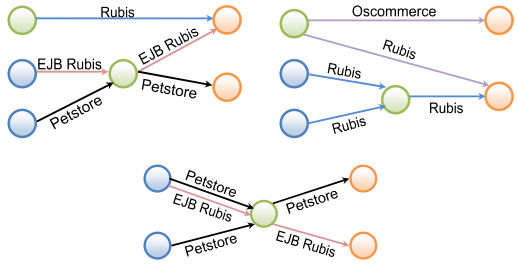


Fig. 4. Enterprise Applications Templates

"main-stream" servers, and the final 20% "advanced" server have a capacity doubles the "advanced" servers. Then we arrange the servers in three topologies—heterogeneous tree (Tree), FatTree [1, 3, 8], and BCube [4]. The Tree topology mimics the layout of today's data center with 3-tier network, while the FatTree topology represents a variety of bandwidth-enhancing tree-like layouts, and the BCube topology represents an alternative (hypercube variant) network layout.

Service Applications: We synthesize 143 service applications with 1067 service nodes, based on real traffic traces captured from our local testbed hosting multiple multi-tier applications. The original traffic traces consist of three applications and 21 traffic nodes, and their composition are show in Figure 4. Using them as template, we generate synthetic applications by randomly select template applications and scaling the traffic and resource requirement up or down, as well as scaling out the components (i.e. double the number of nodes in each component, and then split the corresponding traffic and resource requirements of each node).

Evaluation Methodology: We conduct four experiments to thoroughly evaluate NetDEO's characteristics. Experiment 1 and 2 compare NetDEO and TVMPP on solving the initial deployment and incremental optimization problems, respectively. Experiment 3 examines NetDEO's unique ability to help data center operators to efficiently upgrade the network and servers. Experiment 4 evaluates NetDEO's scalability and time complexity, as well as its performance improvement over classic SA algorithm.

The results of NetDEO and classic SA algorithms are obtained by taking the average of 100 runs with different random seeds; the results of TVMPP are obtained with a single execution, because it is a deterministic algorithm.

B. Experiment 1: Initial Deployment

In this experiment, we compare the optimization for initial deployment of NetDEO and TVMPP. Since there is no preexisting assignments, service nodes can be freely deployed to any suitable servers. The two major performance metrics in this experiment are the degree of optimization and the running time, and for both metrics the smaller value the better.

Figure 5(a) shows the best optimization stress of both algorithms running for about the same amount of time. For Tree and FatTree topologies TVMPP is slightly better than NetDEO, by 3.25% and 2.53% respectively. However, for BCube topology, NetDEO outperform TVMPP by 21.4%.

NetDEO's slightly worse optimization results for Tree and FatTree can be attributed to the unusual server setup that each server can only host a single service[3]. This setup heavily restricts the search space so that only two-displacement neighborhood exploration is possible, which limits the effectiveness of stochastic search algorithms such as NetDEO. However, interestingly, although NetDEO suffer from the similar effects for BCube, its performs significant better than TVMPP. We believe this is due to TVMPP's weakness on this topology—BCube, as well as other hypercube variant topologies, organize servers in high dimensional space, which cause the K-means clustering algorithm, a critical component of TVMPP, to perform poorly.

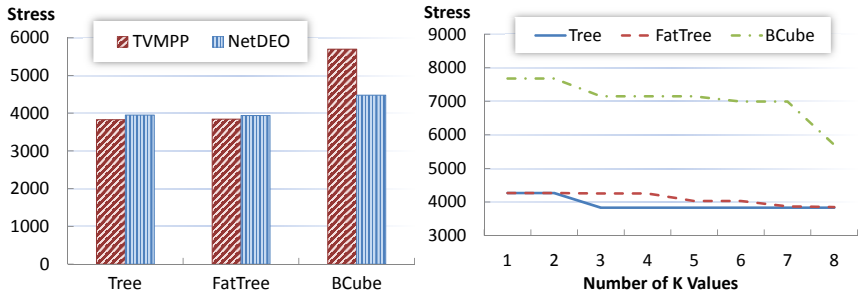
Note that, instead of comparing the running time between NetDEO and TVMPP, we let both algorithms run for about the same amount of time. This is because like NetDEO, TVMPP also has a running time–optimization trade off, which makes comparing both metric at the same time meaningless. TVMPP requires the number of clusters (i.e., K in the K-means clustering algorithm) as one of its parameter. However, determining the optimal value of K is an open hard problem. To work around the problem, TVMPP runs the cluster-and-cut algorithm multiple times using a series of K values, and pick the best optimization. Figure 5(b) shows that the improvement of TVMPP optimizations as we run it for more K values.

C. Experiment 2: Incremental Optimization

Using the optimized deployment schemes generated in the previous experiment as the existing service node placements, we simulate realistic workload / traffic pattern changes in the data center by manually scaling up and down the traffic and resource requirements of 12 applications (102 service nodes, or about 10% of all nodes and traffic flows). The goal of this experiment is to recover the performance degradation by finding new placements with stress no greater than that of the original optimized placements. We compare the performance of NetDEO and TVMPP in terms of the running time and the number of displaced service nodes in their solutions (the smaller value the better for both metrics).

As shown in Figure 6, NetDEO outperforms TVMPP with dominating factors. With respect to Tree, FatTree and BCube topologies, NetDEO is 67.25%, 95.54% and 98.40% faster, and moves 26.55%, 78.41% and 98.66% less number of service nodes than TVMPP. The dominating success is well expected for NetDEO, because it is designed with incremental optimization in mind, while TVMPP is not. TVMPP treats the slightly altered setup as a totally new problem and solves it from scratch. The changes in the traffic matrix leads to a different partitioning sequence which in turn yields to different service–server mappings. And as a consequence of this avalanche effect, with even a small change in traffic, the new solution of TVMPP is likely to be totally different from its previous solution.

[3]We have modified (simplified) our DCN setup for experiment 1 and 2 in order to compare with TVMPP, which is not designed to handle heterogeneous server resource capacities and service node resource requirements.



(a) (b)

Fig. 5. **Experiment 1:**(a) NetDEO vs. TVMPP on Initial Deployment;(b) TVMPP Run-time vs. Optimization.

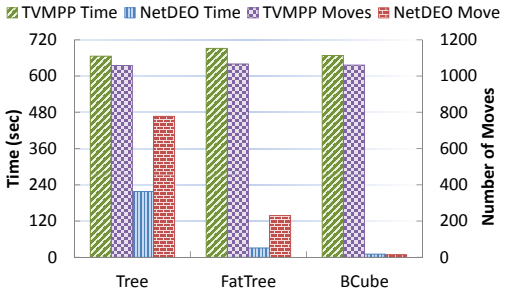
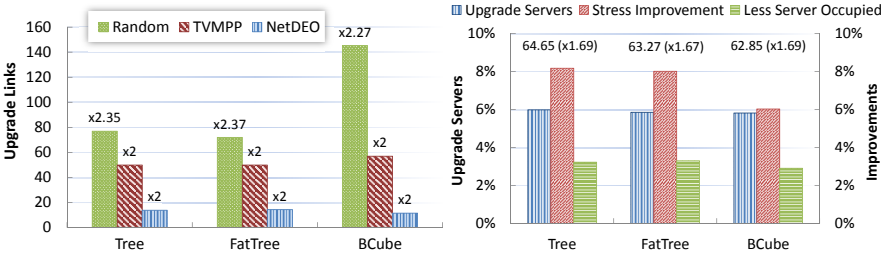


Fig. 6. **Experiment 2:***NetDEO* vs. *TVMPP* on Incremental Optimization



(a) Upgraded Links (b) Server Upgrade Improvements

Fig. 7. **Experiment 3:**(a) Network Upgrade Solutions;(b) Server Upgrade Solutions.

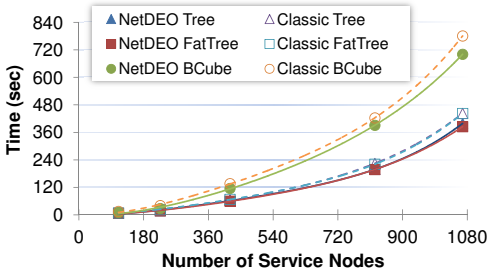


Fig. 8. **Experiment 4:**Scalability Test Results

D. Experiment 3: Upgrade Suggestions

This experiment consists of two sub-experiments which examines NetDEO's capability to provide data center network and server system upgrade suggestions. We use setups similar to that of Experiment 1, but with some modifications according to our use cases.

1) Network Upgrade: In this experiment, we simulate an overloaded data center network by scaling up all traffic rates of our service applications by 100 times. As a result, it is impossible to deploy the applications on any of the three topologies without violating network capacity constraints. And thus, the challenge is to deploy all applications without network capacity violation, with the minimum number of network links requiring an upgrade.

Shown in Figure 7(a), for Tree, FatTree and BCube topologies, NetDEO identifies viable deployment solutions that (on average) upgrade 13.90, 14.41, and 11.57 network links with double capacity, respectively. In contract TVMPP's solutions require over 3 times more link upgrades than NetDEO for all topologies. The results of a randomized deployment are also shown in the same figure as references.

2) Server Upgrade: In this experiment, we solve the problem of service consolidation and server upgrading. Suppose the DCN operator has a limited budget to upgrade some servers. The goal of this experiment is to find answers to the questions of which servers to upgrade, and how to deploy services after the upgrade to benefit the most from the upgrade.

We configure NetDEO to search solutions with up to 4× server resource capacity upgrade. Shown in Figure 7(b), for Tree, FatTree and BCube topologies, NetDEO provides deployment solutions that (on average) upgrade 64.65 (5.99%), 63.27 (5.86%), and 62.85 (5.82%) servers with average capacity increase of 1.69×, 1.67×, and 1.69×, respectively. Correspondingly, with the new deployment solutions, the stress is improved by 8.18%, 8.01%, and 6.03%, and the server occupation is reduced by 3.23%, 3.30%, and 2.91%, for Tree, FatTree and BCube topologies respectively. These new solutions with server upgrades enjoy not only improved performance comparable to the cost of upgrade, but also reduced server occupation, which can lead to lowered energy consumption as well as reduced maintenance cost.

E. Experiment 4: Scalability and Time Complexity

In this experiment, we evaluate the scalability of NetDEO with respect to the number of service nodes. Because the NetDEO algorithm belongs to the SA optimization, presenting the asymptotic upper bound of the runtime is meaningless in practice[4]. Instead, we present its empirical running time of reaching an acceptable level of optimization, given different sizes of input, and approximate the runtime using curve-fitting.

Shown in Figure 8, we plot the run time vs. service node count of NetDEO with 110, 226, 420, 823 and 1067 service nodes for each topology. We found that the series of data points for each topology can be curve-fitted by a 4th degree polynomial with very high ($> 99.9\%$) confidence. The results indicate that the empirical time complexity of NetDEO algorithm is $O(n^4)$, in agreement with previous SA research

[4]SA optimization algorithms guarantee to find the optimal solution when they completely converge, and their time complexity for solving NP-hard problem is known to be exponential.

[26]. Also shown in the same figure are the running time and fitting curves of a classic SA algorithm solving the same set of problems. And we observe that NetDEO has effectively speed up the convergence rate (by about 8–10%), thanks to our algorithm enhancements.

VI. Conclusion

In this paper, we presented NetDEO, a DCN performance optimization framework designed continuous and cost effective data center maintenance. Different from previous approaches, NetDEO takes into account the applicability of solutions, the evolutionary nature of data center networks, and the real-world constraints encountered by network operators. NetDEO employs a pseudo-physical optimization model and an enhanced simulated annealing optimization algorithm. Our comprehensive evaluation shows that for incremental optimization problems, NetDEO significant outperforms existing solutions, in terms of solutions quality and running time. For initial deployment problem, NetDEO's optimization and running time are comparable to or better than existing solutions. In addition, NetDEO can help the operators to efficiently upgrade data center hardware.

In terms of limitations, there are two application constraints. First, in order for NetDEO to perform efficient optimizations, the DCN operators need to supply the correct network configurations, as well as an accurate estimation of service workload. And sometimes the latter is difficult to obtain. Second, the theoretical time complexity of NetDEO algorithm is exponential. However, empirically NetDEO can reach good optimization within a reasonable time frame.

We are currently in the process of porting NetDEO onto operational data centers and evaluating its performance using traffic traces generated by real-world applications. Our future work also involves designing techniques that reduce NetDEO's monitoring overhead, exploring mechanisms that minimize operators' intervention, and incorporating into NetDEO additional constraints such as security rules and administrative policies.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity, data center network architecture," in *ACM SIGCOMM*, Aug. 2008.

[2] N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *ACM SIGCOMM*, Aug. 2010.

[3] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM*, Aug. 2009, pp. 51–62.

[4] 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 *ACM SIGCOMM*, Aug. 2009, pp. 63–74.

[5] 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 *ACM SIGCOMM*, Aug. 2008, pp. 75–86.

[6] S. Kandula, J. Padhye, and P. Bahl, "Flyways to de-congest data center networks," in *ACM HotNets*, Oct. 2009.

[7] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "Ficonn: Using backup port for server interconnection in data centers," in *IEEE INFOCOM*, Aug. 2009, pp. 2276–2285.

[8] 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 *ACM SIGCOMM*, Aug. 2009, pp. 39–50.

[9] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang, "Proteus: A topology malleable data center network," in *ACM HotNets*, Oct. 2010.

[10] G. Wang, D. G. Andersen, M. Kaminsky, M. Kozuch, T. S. E. Ng, K. Papagiannaki, M. Glick, and L. Mummert, "Your data center is a router: The case for reconfigurable optical circuit switched paths," in *ACM HotNets*, Oct. 2009.

[11] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *IEEE INFOCOM*, San Diego, CA, USA, March 2010.

[12] M. R. Korupolu, A. Singh, and B. Bamba, "Coupled placement in modern data centers," in *IEEE IPDPS*, Rome, Italy, May 2009.

[13] X. Zhu, C. Santos, C. Santos, J. Ward, J. Ward, D. Beyer, D. Beyer, S. Singhal, and S. Singhal, "Resource assignment for large-scale computing utilities using mathematical programming," Mathematical Programming, HP Lab, Tech. Rep., 2003.

[14] D. Oppenheimer, B. Chun, D. Patterson, A. C. Snoeren, and A. Vahdat, "Service placement in a shared wide-area platform," in *the annual conference on USENIX '06 Annual Technical Conference*, Boston, MA, USA, June 2006.

[15] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. New York, NY, USA: Oxford University Press, Inc., 1999.

[16] P. J. M. Laarhoven and E. H. L. Aarts, Eds., *Simulated annealing: theory and applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1987.

[17] (2012, Jan.) Metaheuristic. [Online]. Available: http://en.wikipedia.org/wiki/Metaheuristic

[18] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," in *ACM SIGCOMM*, Aug. 2010.

[19] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, "Spain: Cots data-center ethernet for multipathing over arbitrary topologies," in *the 7th USENIX NSDI*, San Jose, CA, USA, April 2010.

[20] S. Sahni and T. Gonzalez, "P-complete approximation problems," *J. ACM*, vol. 23, pp. 555–565, July 1976.

[21] R. Hassin, A. Levin, and M. Sviridenko, "Approximating the minimum quadratic assignment problems," *ACM Trans. Algorithms*, vol. 6, pp. 18:1–18:10, December 2009.

[22] J. Sonnek, J. Greensky, R. Reutiman, and A. Chandra, "Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration," Computer Science and Engineering, University of Minnesota, Tech. Rep., December 2009.

[23] H. Ballani, P. Costa, T. Karagiannis, and A. I. T. Rowstron, "Towards predictable datacenter networks." in *SIGCOMM'11*, 2011, pp. 242–253.

[24] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees." in *CoNEXT'10*, 2010, pp. 15–15.

[25] H. Chen, H. Kang, G. Jiang, K. Yoshihira, and A. Saxena, "Vcae: a virtualization and consolidation analysis engine for enterprise data centers," in *IEEE SASO*, Sep. 2010.

[26] G. H. Sasaki and B. Hajek, "The time complexity of maximum matching by simulated annealing," *Journal of the ACM*, vol. 35, pp. 387–403, 1988.