NPS-CS-16-003

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

AN APPLICATION AWARE APPROACH TO SCALABLE ONLINE PLACEMENT OF DATA CENTER WORKLOADS

by

Alan Bairley
Geoffrey G. Xie

16 December 2016

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
16-12-2016	Technical Report	2016-06-10 to 2016-12-16

4. TITLE AND SUBTITLE

AN APPLICATION AWARE APPROACH TO SCALABLE ONLINE PLACEMENT OF DATA CENTER WORKLOADS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Alan Bairley, Geoffrey G. Xie

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

NPS-CS-16-003

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of the Department of Defense
Chief Information Officer
6000 Defense Pentagon
Washington, D.C. 20301-6000

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release. Distribution is unlimited.

13. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.IRB Protocol Number:N/A.

14. ABSTRACT

Data center operators strive for maximum resource utilization while satisfying tenant service level agreements; however, they face major challenges as application workload types are diverse and tenants add, remove, and update their workloads sporadically to meet changing user demands.Currently, operators allocate workload VMs primarily in an application agnostic fashion, focusing on minimizing total resource usage.In this work, we first show that such allocations can be suboptimal and then present a new application aware approach that explicitly models resource preferences of individual workloads.Further, we propose a new logical application workload (LAW) abstraction to enable precomputation of the required relative positioning of an application's VMs and allocation of these VMs in a single atomic step, leading to online algorithms that are one order of magnitude faster than existing per VM placement solutions.We then develop a statistical extension of LAW to add flexibility in characterizing application requirements and to support prioritization of workloads.Using realistic workload traces and physical topologies, we evaluate our algorithms in a simulated large-scale data center setting, and demonstrate their performance advantages and potential tradeoffs versus existing solutions.

15. SUBJECT TERMS

Software-Defined Networking; Network State; Datacenter Network; Genetic Algorithms

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Alan Bairley
Unclassified	Unclassified	Unclassified	UU	55	19b. TELEPHONE NUMBER *(include area code)* 910-447-9707

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Ronald A. Route
President

Steven R. Lerman
Provost

The report entitled "An Application Aware Approach to Scalable Online Placement of Data Center Workloads" was prepared for the general public and funded by the Department of Defense Information Assurance Scholarship Program (DoD IASP).

Further distribution of all or part of this report is authorized.

This report was prepared by:

Alan Bairley

Geoffrey G. Xie

Reviewed by:

Peter J. Denning, Chairman
Department of Computer Science

Released by:

Jeffrey D. Paduan
Dean of Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Data center operators strive for maximum resource utilization while satisfying tenant service level agreements; however, they face major challenges as application workload types are diverse and tenants add, remove, and update their workloads sporadically to meet changing user demands. Currently, operators allocate workload VMs primarily in an application agnostic fashion, focusing on minimizing total resource usage. In this work, we first show that such allocations can be suboptimal and then present a new application aware approach that explicitly models resource preferences of individual workloads. Further, we propose a new logical application workload (LAW) abstraction to enable precomputation of the required relative positioning of an application's VMs and allocation of these VMs in a single atomic step, leading to online algorithms that are one order of magnitude faster than existing per VM placement solutions. We then develop a statistical extension of LAW to add flexibility in characterizing application requirements and to support prioritization of workloads. Using realistic workload traces and physical topologies, we evaluate our algorithms in a simulated large-scale data center setting, and demonstrate their performance advantages and potential tradeoffs versus existing solutions.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

Acknowledgements

The authors would like to thank the DoD Information Assurance Scholarship Program (IASP) for providing general support towards this work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

The data center operators of large enterprises and public cloud providers face major challenges in allocating virtual machines (VMs) to individual application workloads. First, the workloads are diverse in sizes (measured by the number of VMs required, the amount of link bandwidth required, running time, etc.), in resource consumption characteristics (e.g., computation intensive vs. data intensive), and in performance requirements such as bounds on response times and application availability [1]. Second, the increased flexibility and agility provided by software-defined data center networks allows enterprises and public cloud tenants to rapidly add, remove, update, and prioritize their workloads to meet sporadically changing user demands. There is an urgent need for online allocation algorithms that can deal with such dynamic behaviors without incurring too much latency [2,3]. Last, but not least, the operators must strive for efficient uses of their physical resources (link bandwidth, electrical power, etc.) in order to minimize cost while meeting the quality of service requirements of applications.

Maximizing resource utilization within data centers is a problem that many prior efforts have attempted to solve using various approaches, including integer linear programming [4], greedy heuristics [5–8], and genetic algorithms [9, 10]. However, we observe that this body of work overwhelmingly strives to optimize *cumulative* resource usage among all hosted tenant applications while deferring the *per application* performance concerns to relatively low level mechanisms such as CPU scheduling and traffic engineering, despite recent studies demonstrating that *different types of workloads contend for different types of resources* and consequently how VMs of an application are *relatively positioned* can significantly impact the performance of an application [11, 12]. Specifically, it would be advisable not to co-locate VMs of computation intensive workloads to avoid unnecessary CPU contention while at the same time, position VMs of the same data intensive workload as close as possible to reduce both bandwidth contention and communication latency [12].

In other words, while workload placement should concern about cumulative resource usage, additional opportunities exist to intelligently place the VMs of individual workloads to improve per application performance of all hosted applications. In this paper, we investigate a placement strategy, which we term "application-aware", to leverage such opportunities. More specifically, we explicitly model the resource preference of each workload and develop a unified framework to characterize and minimize resource contentions introduced by a new workload, which can be between VMs of the

same workload as well as with respect to existing workloads. Further, we enhance the application-aware approach by introducing a new logical application workload (LAW) abstraction. The LAW represents the most desirable relative positioning of the workload's VM placement in a physical topology in terms of *both* meeting operator specific resource efficiency goals *and* minimizing resource contentions. We show that a data center controller can precompute LAWs and subsequently assign VMs of each workload in a single atomic step, leading to online algorithms that are one order of magnitude faster than current per VM placement solutions.

Our contribution is multi-fold as follows.

1. First, we show that application agnostic workload placement can introduce unnecessary resource contentions, and demonstrate potential performance gains from precisely modeling resource contentions that can be introduced by a workload. (Chapter 2)
2. Second, we formally define LAW and describe how to construct LAWs that minimize potential resource contentions. We then design a range of bin packing heuristics to place workloads on a per LAW basis and at the same time optimize different cumulative resource usage objectives. (Chapter 3)
3. Third, we observe that LAW based workload placement can be infeasible sooner than per VM placement. Consequently, we propose a statistical extension of the LAW abstraction to add flexibility in characterizing application requirements. We show that the extension permits a data center operator to increase LAW placement feasibility via graceful degradation of application performance. (Chapter 4)
4. Fourth, we show that the statistical LAW extension provides a natural mechanism to support prioritization of workloads, conceptually similar to the Random Early Detection (RED) queueing [13]. (Chapter 5)
5. Fifth, using a simulated scenario with realistic workload traces and a relatively large data center topology, we evaluate the LAW based workload placement heuristics and demonstrate their performance advantages and potential tradeoffs versus existing solutions. (Chapters 4, 5, and 6)

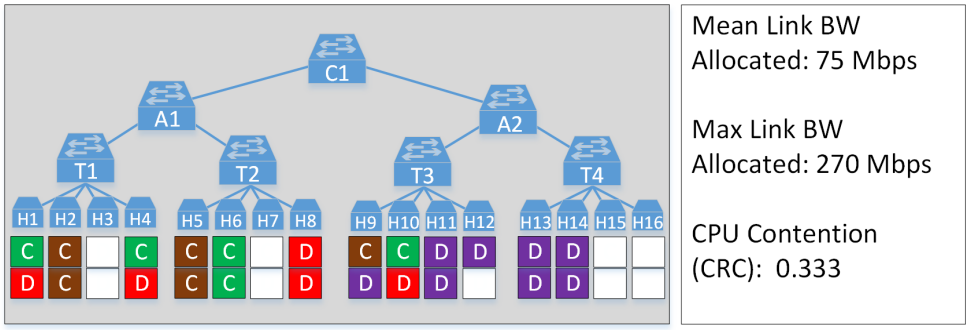
CHAPTER 2:
Application-Aware Placement

We motivate and illustrate the advantages of application-aware placement with a simplistic but telling scenario. We also describe how to model a workload's resource preference and present metrics to characterize resource contentions introduced by a workload.

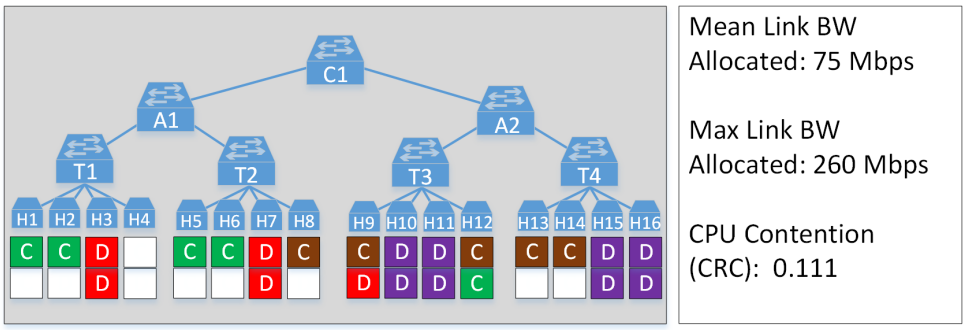
First, from prior work we observe that there are two basic types of application workloads: 1) CPU or memory intensive workloads (e.g., meteorological, geological, and particle physics simulations, or other high performance computing tasks), commonly referred to as compute-intensive (CI) workloads, and 2) network or storage intensive workloads (e.g., client-server or Hadoop and other big data applications), commonly referred to as data-intensive (DI) applications. Generally, state-of-the-art research [11, 12] concludes that CI workloads perform better when the processes and VMs are spread across *separate* CPU cores and hosts, respectively, while DI workloads perform better when the VMs (or processes) are placed on the *same* host (or CPU core). For example, by placing all VMs of a client-server (DI) application on the same physical host, tenfold increases in throughput have been observed [11]. In contrast, over-contention of CPU resources by VMs of CI applications have been shown to increase job completion time by as much as 260% [11].

Therefore, we propose to extend the current workload representations, which primarily specify the number of VMs and intra-VM bandwidth [5, 6, 14], to also include a designation of application type (CI or DI). Such a classification may be specified a priori by tenants, or determined post-allocation by the operator upon monitoring workload resource usage in practice. In addition, we believe it is advantageous for a data center to model the survivability requirement (i.e., resiliency against single top-of-rack (ToR) switch failures) on a per application basis because typically, different applications have different levels of importance to the tenants. Commonly, the survivability requirement is represented by a metric called worst-case survivability (WCS), which is the maximal possible fraction of VMs taken offline due to a single ToR failure [15].

Now consider the following simplistic scenario. Suppose four independent tenant application workloads R1, R2, R3, R4, arrive in sequential order and have requirements of <CI, 5, 10 Mbps, 0.5>, <DI, 5, 50 Mbps, 0.5>, <CI, 5, 10 Mbps, 0.5>, and <DI, 8, 50 Mbps, 0.5> respectively. The first value in the tuple represents the application type (appType), the second value represents the number of VMs (numVMs) required,



(a) Application Agnostic Allocation



(b) Application Aware Allocation

Figure 2.1: An example contrasting application-aware vs. application agnostic VM allocations. Green slots are for VMs of R1, red R2, brown R3, and purple R4. "C" and "D" denote a CI and DI workload, respectively. The allocation on the right considers application type (CI or DI) of each workload, and thus attempts to spread VMs of R1 and R3 (CI) to minimize CPU contention, while consolidating VMs of R2 and R4 (DI) to minimize communication overhead.

the third value represents the inter-VM bandwidth (BW) reservation requirement[1], and the fourth value represents the minimum required WCS[2]. Suppose the underlying physical infrastructure has a hierarchical tree topology, consisting of one core switch as root, two aggregation switches, four ToR switches, four host servers per ToR switch, and two VM slots per host server, where a "VM slot" is defined as a standard physical resource unit (e.g., CPU, Memory) provisioned to a VM. Therefore, the VMs are to be allocated against $4 \times 4 \times 2 = 32$ possible slots.

Current solutions, such as [5, 7, 8, 10], overwhelmingly attempt to maximize resource

[1]Inter-VM bandwidth is reserved according to the "hose" model, as done in [14].

[2]Prior work in datacenter network orchestration [5, 6] asserts that a WCS of 0.5 should be the bare minimum acceptable to the network operator.

utilization by finding a "best fitted" allocation of tenant application workloads that minimizes some weighted global cost function associated with data center resource usage, and more specifically, bandwidth usage. In theory, such approaches seem effective in maximizing global resource utilization. However, because global cost functions are not capable of representing fine-grained *contention points* of independent tenant application workloads (e.g., CI or DI), the "globally optimal" network allocations produced by current solutions are *unlikely to be optimal for each tenant application comprising it.* This situation is illustrated in Figure 2.1. The left part (2.1a) depicts a best-fitted allocation representative of current solutions, which minimizes network-wide bandwidth usage while satisfying application requirements R1, R2, and R3. However, note that CI applications R1 and R3 are in contention with themselves for CPU and memory on hosts H2, H5, and H6, and DI applications R2 and R4 do not achieve optimal network/storage sharing benefit via host colocation: R2 misses an opportunity for colocation under rack T1 and R4 misses a colocation opportunity under rack T3. Figure 2.1b, in contrast, depicts an *application-aware* best-fitted allocation, which not only minimizes network-wide bandwidth usage, but also *provides optimal conditions for each tenant application.* Although CI applications R1 and R3 compete with each other on H12, they do not compete with themselves on any host, and DI applications R2 and R4 achieve maximum intra-host colocation.

The example in Figure 2.1 also shows that application-aware allocation does not always degrade each global utility metric. Here, the application-aware allocation method achieves less link congestion (i.e., smaller max BW) by using more physical servers to host the VMs, a tradeoff that typically results in more power usage.

The insight that different types of tenant workloads have different host assignment preferences motivates the need for a metric to capture such preferences. To this end we propose a new Resource Contention Index (RCI) to evaluate the placement of individual tenant applications (Def. 1).

Def. 1. (**RCI** for placement of workload w)

$$RCI(w) = \begin{cases} 0 & \text{workload requires 1 VM} \\ Host\_Affinity(w) & \text{workload is CI} \\ 1 - Host\_Affinity(w) & \text{workload is DI} \end{cases}$$

where:

$$Host\_Affinity(w) = 1 - \frac{(\#\text{of hosts hosting VMs of } w) - 1}{(\text{total VMs allocated for } w) - 1},$$

and $Host\_Affinity()$ is defined only when the total number of VMs allocated for w is greater than 1.

The host affinity for some tenant application represents the degree by which its VMs are intra-host colocated. A value of '0' indicates that no application VMs share the same host, and thus minimizes intra-application computational contention for a CI workload, whereas a value of '1' indicates that all application VMs share the same host, hence maximizing intra-application data throughput for a DI workload. RCI captures this fundamental difference in application preference between CI and DI workloads in a single application-specific metric. Furthermore, we can take the average (mean) RCI across all allocated workloads to provide a measure of global application efficiency. Note that achieving an RCI of zero for all workloads is not realistic because tenants may have competing objectives or constraints. Survivability for instance, is directly at odds with RCI for DI workloads, as a host affinity of '1' implies WCS = 0. But having zero survivability is not acceptable for most tenants. As another example, consider tenant budget constraints for a CI workload. It may be cost prohibitive for a tenant to place each workload VM on a separate physical host or CPU core. In such cases, a tenant may choose to settle a higher intra-VM contention ratio (i.e., RCI > 0) in the interest of lower SLA costs.

A useful adaptation of RCI, termed *computational resource contention* (CRC), is presented in Def. 2.

Def. 2. (Computational Resource Contention (**CRC**))

$$CRC = 1 - \frac{(\#\text{ of hosts hosting CI VMs}) - 1}{(\text{total CI VMs allocated}) - 1}, \tag{2.1}$$

and CRC is defined only when the total CI VMs allocated is greater than 1.

This metric, by treating all CI applications as one group, represents the degree of contention for computational (CPU/Memory) resources throughout the network. We argue that CRC should be added to the collection of global cost functions used for workload placement, as this metric is particularly relevant as an indicator of throughput bottlenecking for CI applications, similar in nature and importance to the commonly used link congestion metrics of mean and maximum bandwidth usage for DI applications. For the example depicted in Fig. 2.1, the application-aware allocation on the right achieves a much lower CRC than its counterpart.

CHAPTER 3:
LAWs for Application Efficiency

To overcome the shortcomings of current, non-application-aware workload placement approaches, we propose a new logical application workload (LAW) abstraction, which serves as the fundamental unit of allocation for an application workload, as opposed to individual VMs, in order to speed up the allocation and *explicitly preserve* the relative positioning of intra-application VMs with respect to the best allocation the operator would desire to obtain for the application on a simulated empty physical infrastructure.

3.1 Natures of LAW

Essentially, a LAW represents the best possible allocation for an application workload from the perspective of the operator with respect to a given physical infrastructure. It is *specific to* the allocation method used by the operator to meet the application's resource and performance requirements, and as such it unambiguously captures the design intent of the operator.

Formally, we define a LAW as follows.

Def. 3. (**Logical Application Workload (LAW)**) *Given an requirement specification R for workload w, a data center infrastructure P to host VMs of w, and a VM allocation method M used by the operator, a LAW $L(w, P, M, R)$ is a hypothetical allocation for w by applying M on an empty P subject to requirements R.*

When the physical infrastructure has a tree topology, a LAW can be simply modeled as the subtree that contains the hypothetically allocated VMs, with additional annotations of required resources. For the application requirement tuple considered in this paper, i.e., the four-tuple of (appType, numVMs, BW, WCS), each switch of the subtree should be annotated with two parameters: *r_slots* for the total number of VMs supported underneath, and *r_bw* for the total amount of bandwidth reservation required on its upstream link. The two parameters are driven by the number of VMs and number of computational resource units (i.e., slots) required by each VM, and the BW requirements, while the relative positions of the VMs should minimize RCI while meeting the WCS bound.

When the physical resource capacity is much larger than what the workload requires, many subtrees can accommodate the LAW, i.e., support the relative positioning of

the VMs. In such a case, we model the LAW using the leftmost subtree. Similarly, before the physically infrastructure is heavily utilized, allocation of a LAW should be straightforward, involving a small number of checks of feasibility against the r_slots and r_bw parameters. In other words, heuristics that allocate a workload on the basis of its LAW should run faster than their per-VM counterparts in most scenarios and therefore should be *more suitable as an online solution*.

Intuitively, compared to a finer-grain per-VM allocation, a LAW based allocation method may trade off some network-wide performance such as power efficiency and bandwidth utilization in order to maximize the quality of service of individual applications. However, the former has its own problem as it allocates VMs sequentially and as such VMs allocated earlier in the sequence may prevent the overall allocation from maximizing global utility. An investigation of this interesting tradeoff will be presented in Chapter 4. In the rest of this section, we focus on how to leverage the concept of LAW to create an online VM allocation solution that can respond to dynamic workload input.

3.2 Constructing Efficient LAWs

While the existing application agnostic solutions [5,5–10] can be directly used to create LAWs, we propose to extend these solutions to consider application characteristics (e.g., CI vs. DI) by applying Algorithm 3.1 post-allocation in order to obtain LAWs with the *smallest RCI possible while meeting other application requirements*. This is accomplished by both a) spreading the CI VMs of each rack over the maximum number of available hosts, and b) concentrating the DI VMs of each rack onto the smallest possible number of physical hosts. Specifically, as presented in Algorithm 3.2, the procedure CONSTRUCT-LAW(P_0, N) first runs an existing per VM allocation scheme on an empty physical topology (simulated), and then calls Algorithm 3.1 on the same input, which, depending on the application type, rearranges some of the VMs to minimize RCI while satisfying other requirements. The selected allocation is then converted into a LAW by (i) removing all elements of the physical infrastructure tree not used by the application, and (ii) tallying up the number of descendant VMs (r_slots) and the reserved upstream bandwidth (r_bw) parameters for each of the remaining nodes.

It is important to note that a data center controller can pre-construct LAWs for expected workloads[3] against available infrastructures and store the results (LAW subtrees) in a hash table. This way, the online allocation algorithm (presented next) will need minimum time to obtain the LAW for a new workload request.

[3]For the scope of this paper, store all combinations of elements of each of the discrete ranges of numVMs, intra-VM BW, and WCS for both CI and DI app types.

Algorithm 3.1 "App-Aware Allocation Adjustment"

```
1: procedure App-Aware-Adjust(w, R, P)
2:                                          ▷ input w: workload id
3:                                          ▷ input R: requirement spec. for w
4:                                          ▷ input P: physical topology; allocated
5:     if R.type = CI then                                  ▷ Minimize host affinity
6:         for each ToR t ∈ P do
7:             Spread VMs of w across hosts in t
8:               while meeting BW and WCS requirements
9:         end for
10:    else if R.type = DI then                             ▷ Maximize host affinity
11:        for each ToR t ∈ P do
12:            Concentrate VMs of w onto hosts in t
13:              while meeting BW and WCS requirements
14:        end for
15:    end if
16: end procedure
```

Algorithm 3.2 "LAW Construction"

```
1: procedure Construct-LAW(w, R, P_0)
2:                                          ▷ input w: workload id
3:                                          ▷ input R: requirement spec. for w
4:                                          ▷ input P_0: physical topology; empty
5:     Per-VM-Allocation(w, R, P_0)
6:     App-Aware-Adjust(w, R, P_0)
7: end procedure
```

3.3 Allocating LAWs

Allocating a LAW to a physical infrastructure (which may or may not be empty) needs to meet a set of global objectives defined by the network operator. These global objectives have traditionally included minimizing power usage [16] and link congestion [10], but we argue that the CRC objective (Def. 2) defined in Section 2 should also be considered.

Conceivably, an algorithm to explore the efficient frontier of LAW allocations with respect to the global objective space could be developed, for instance an algorithm similar to EASO [9] that takes a set of LAWs as input and produces a set of efficient proposed allocations as output, where each satisfies all LAWs. However, we defer this

challenge to future work, and instead focus efforts on the more fundamental, low-level problem of allocating each individual LAW in a manner that is both fast and resource efficient, with the goal of minimizing some global cost function. Specifically, we consider the single-objective cost functions of 1) power usage, 2) link congestion (i.e., mean and max BW usage), and 3) CRC.

In the remainder of this section, we describe a general algorithm for allocating a LAW to the physical infrastructure, and then discuss the use of different heuristics for minimizing the global cost functions described previously. Because we are allocating LAWs, i.e., sets of VMs that are placed with strict relative positioning requirements, we cannot directly apply heuristics for individual VM placement, as done in [17, 18].

Thus, we propose Algorithm 3.3, which takes the precomputed LAW L for an application workload (assumed to satisfy all requirements R of workload w), a physical network state P (i.e., physical infrastructure with current allocation), and a cost-minimizing heuristic h as input, and attempts to find a feasible allocation for the workload. Specially, we consider three heuristics in this paper: (1) "Min Power": referred to as *best-fit* or *tightest-fit* in the classic bin packing problem literature [17], it seeks to map VMs of L onto hosts with the smallest number of free slots, effectively minimizing the number of active hosts; (2) "Min BW": similar to the *max-rest* bin packing heuristic [17], it seeks to map VMs of L onto hosts with the largest number of free slots, effectively balancing bandwidth allocation and minimizing BW congestion (i.e., maximum link BW); (3) "Min CRC": designed to spread CI workloads as evenly among hosts as possible while attempting to place DI workloads within the same subtree as previously allocated CI workloads (to allow maximum CI workload distribution). While the underlying approach of Algorithm 3.3 is generally applicable to tree-based data center topologies including Fat-Tree and VL2, for ease of exposition, we assume that P has a simple tree structure.

At the heart of LAW based allocation is to map each LAW node (ln) to a unique physical node (denoted by $ln.pn$) at the same tree level that has sufficient resources to support the requirements of $ln.r\_slots$ and $ln.r\_bw$. After considering several different tree search algorithms to explore the feasible $ln \longrightarrow ln.pn$ search space, including depth-first search, breadth-first search, best-first (A*) search, and backtracking approaches, in the interests of scalability, we ultimately decided to model the problem of finding the "best" set of feasible $ln \longrightarrow ln.pn$ mappings as a minimization variant of the classical assignment problem, also known as the minimum weighted bipartite matching problem [19]. To this end, Algorithm 3.3 uses the Hungarian or Kuhn-Munkres algorithm, originally proposed by Kuhn [20] and later refined by Munkres [21], as a subroutine in the Assign-Min-Cost sub-procedure. The details are given below.

Algorithm 3.3 "Online LAW Allocation"

```
1: procedure ALLOCATE-LAW(L, P, h)
2:                                   ▷ input L: LAW, P: physical topology
3:                                   ▷ input h: cost-minimizing heuristic method
4:     for each level l from "ToR" to "core" do
5:         for each LAW node ln ∈ L at level l do
6:             for each physical ToR pn ∈ P at level l do
7:                 if ASSIGN-MIN-COST(ln, pn) < ∞ then
8:                     Save best matching of ln.children,
                         pn.children for (ln, pn)
9:                 end if
10:            end for
11:            if no saved matchings for ln then
                   return false
12:            end if
13:        end for
14:    end for
15:    Assign mapping L.root.pn = P.root                    ▷ core nodes
16:    ME = edge set for saved best matching of
         L.root.children, P.root.children
17:    while ME ≠ ∅ do
18:        Extract next edge (ln, pn) from ME
19:        Assign mapping ln.pn = pn
20:        Add edge set for saved best matching of
             ln.children, pn.children to ME
21:    end while
22:    Allocate VMs according to ln ⟶ ln.pn mappings
         and update f_slots and f_bw of each pn accordingly.
23:    Call Alg. 3.1 for application-aware adjustment.
24:    return true
25: end procedure
```

3.3.1 Checking Feasibility

Algorithm 3.3 determines LAW allocation feasibility by first checking at ToR level as follows. For each LAW ToR T_l and each physical ToR T_p, it constructs a bipartite graph consisting of hosts of T_l on one end and hosts of T_p on the other. An edge is added between a pair of nodes on the opposite ends *if and only if* the physical host has sufficient VM slots to support the LAW host. The edge weight assigned is equal to the number of slots required by the LAW host (i.e., the r_slots parameter) multiplied by a heuristic value associated with the the physical host's current state: The heuristic value equals the number of free VM slots (f_slots) for "Min Power"

Algorithm 3.3 "Online LAW Allocation" (Cont.)

19: **procedure** ASSIGN-MIN-COST(ln, pn)
20: Initialize a bipartite graph $G =< V_l, V_p, E = \emptyset >$
21: where $V_l = ln.children$ and $V_p = pn.children$
22: **for each** pair of nodes $(u \in V_l, v \in V_p)$ **do**
23: add new edge $< u, v >$ to E
24: **if** $v.f\_slots \geq u.r\_slots \wedge pn.f\_bw \geq u.r\_bw$ **then**
25: $h\_val =$ GET-HEURISTIC(pn, h, L)
26: set edge weight $= u.r\_slots \times h\_val$
27: **else** set edge weight $= \infty$
28: **end if**
29: **end for**
30: Add dummy nodes to V_l and associated ∞ edges
31: until $\|V_l\| = \|V_p\|$
32: ME = KUHN-MUNKRES(G)
33: ▷ ME stores the minimum weighted edge set
34: Remove edges with dummy nodes from ME
35: **return** sum of edge weights of ME
36: **end procedure**

37: **procedure** GET-HEURISTIC(pn, h, L)
38: **if** $h =$ "Min Power" **then return** $pn.f\_slots$
39: **end if**
40: **if** $h =$ "Min BW" **then return** $pn.r\_slots$
41: **end if**
42: **if** $h =$ "Min CRC" **then**
43: **if** $L.type =$ "CI'' **then return** $pn.ci\_slots$
44: **else return** $pn.slot\_capacity - pn.ci\_slots$
45: **end if**
46: **end if**
47: **end procedure**

and the number of reserved slots (r_slots) for "Min BW". There are two cases for "Min CRC". If the LAW represents a CI workload, the heuristic value equals the number of reserved CI slots (ci_slots); otherwise, it equals the slot capacity less the number of reserved CI slots. The scaling of r_slots ensures that when a set of LAW hosts can be supported by multiple physical hosts, the LAW host with the largest VM requirement will be matched with the physical host with the smallest heuristic value, and so on.

Because typically T_l has a smaller number of hosts than T_p but the Hungarian algo-

rithm requires a complete bipartite graph of equal size partitions as input, we modify the bipartite graph as done in [22], by 1) adding special edges of infinite weight (representing "infeasibility") for all pairs of LAW and physical hosts that are not yet connected, and 2) adding dummy nodes to T_l with infeasible (infinite weight) edges from each dummy node to all nodes in T_p. As such, the Hungarian algorithm always returns a complete minimum weight matching, but this matching may include some infinite weight edges, representing "no feasible assignment" [22]. After running the Hungarian algorithm on the modified bipartite graph, we remove the dummy nodes (and associated edges) from the returned matching. If the remaining mapping still contains edge(s) with infinite weight, then it is safe to to say that T_p cannot support T_l.

Once the feasibility between each pair of LAW and physical ToR's is determined, the same process repeats for nodes upward the LAW hierarchy (e.g., first the two aggregation switches, and then the core switch for the example scenario in Figure 2.1), until (i) the LAW root (core) is reached, at which point the LAW is determined to be feasible, or (ii) if, for some intermediate node of L, no feasible mappings exist between it and a physical counterpart, then ALLOCATE-LAW returns false and LAW L is determined to be infeasible for P.

ALLOCATE-LAW(L, P, h) returns **true** if and only if the Kuhn-Munkres algorithm returns a finite sum of edge weights. At each level of feasibility checking, a finite edge weight is assigned to a prospective $ln \longrightarrow ln.pn$ mapping if and only if the mapping is feasible, i.e., the physical node $ln.pn$ has sufficient VM slots and available uplink bandwidth to meet the requirements of the LAW node ln. And because the Kuhn-Munkres algorithm returns the set of edges comprising a minimum-weight matching, the sum of the edge weights returned by the Kuhn-Munkres algorithm is finite if and only if there exists a feasible $ln \longrightarrow ln.pn$ mapping for each LAW node. Thus, ALLOCATE-LAW(L, P, h) returns **true** if and only if there exists a feasible mapping for each LAW node.

3.3.2 General LAW Allocation

Once a LAW L is determined to be feasible as described previously (lines 4 to 11), Algorithm 3.3 proceeds to determine the best $ln \longrightarrow ln.pn$ mappings using a top-down approach, from the LAW root (core) down to the LAW leaf nodes (hosts), selecting the best LAW-to-physical mappings at each level of the hierarchy based upon the values of the saved matchings that minimize the chosen heuristic (lines 15 to 20). Next, after the ToR mappings have been determined, the LAW VMs are allocated according to their respective $ln \longrightarrow ln.pn$ mappings (line 22). Finally, VM placements are adjusted to ideal placement for application objectives by calling Algorithm 3.1.

3.3.3 Complexity Analysis

Space Complexity $= O(|L| \cdot C^2)$, where C represents the maximum number of children of any node $pn \in P$. Space complexity is dominated by one of two factors: 1) the number of edges in the largest bipartite graph G, which may contain up to C^2 edges, and 2) the total number of edges maintained in the saved matchings. Because each of the $O(|L|)$ internal LAW nodes may contain up to C matchings of size C, the total number of edges maintained in the saved matchings is $O(|L| \cdot C^2)$. Thus, the resultant space complexity is $O(C^2 + |L| \cdot C^2) = O(|L| \cdot C^2)$.

Time Complexity $= O(|L| \cdot C^4)$. Time complexity is dominated by the execution time of the Kuhn-Munkres algorithm, which runs in $O(C^3)$ and executes up to C times for each of the $O(|L|)$ internal LAW nodes, thus yielding a resultant time complexity of $O(|L| \cdot C^4)$.

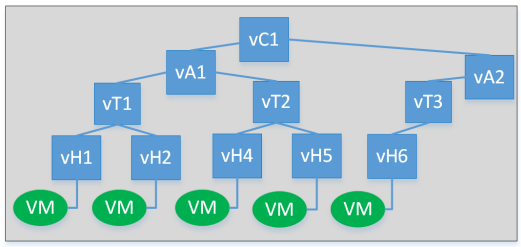
3.3.4 LAW Allocation Example

We conclude this section by illustrating the execution of Algorithm 3.3 for each heuristic method to allocate workloads in the example scenario presented in Section 2. First, ideal LAWs are constructed for each workload, depicted in Figure 3.1, using Algorithm 3.2. Next, Algorithm 3.3 allocates the workloads in the order they arrive (R1, R2, R3, R4), resulting in the allocations shown in Figure 3.2, depending on the chosen allocation heuristic.

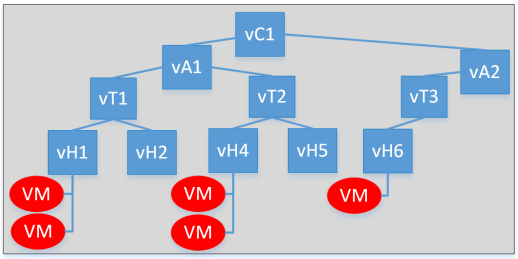
Note that LAW R4 is not feasible using the "Min CRC" heuristic method, since the resultant network state after allocating LAWs R1-R3 does not permit any feasible allocation of LAW R4. Therefore, mapping an entire LAW subtree to the physical infrastructure in a single atomic step appears to incur a tradeoff between performance gain and an increased likelihood of infeasibility.

3.3.5 Observation on Infeasibility

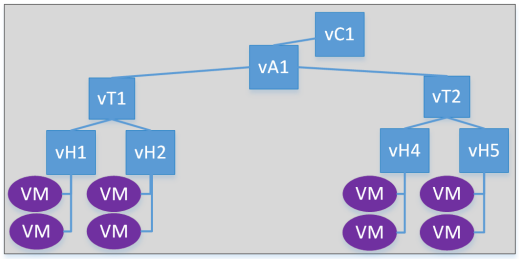
Although not necessarily intuitive, the reason why "Min CRC" encounters infeasibility sooner than the other heuristic methods is relatively straightforward. Because "Min CRC" explicitly seeks the maximum spread of CI workload VMs across ToRs and hosts within ToRs, as more CI LAWs are allocated, it becomes increasingly difficult to allocate DI LAWs, which require the use of a large number VM slots on a single host in order to achieve minimum RCI. In the case of the example scenario, LAW R4 could not be feasibly allocated given the allocations of LAWs R1-R3 using "Min CRC" (Figure 3.2c). Thus, in order to allocate workload R4, either the use of a per VM allocation method, such as GASO, or some method for deallocating and reallocating



(a) LAW R1, R3



(b) LAW R2



(c) LAW R4

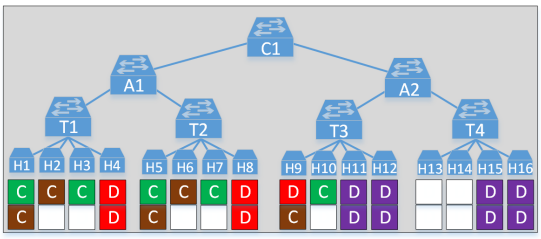
Figure 3.1: LAWs for workloads R1-R4 used in the example scenario of Fig. 2.1. The *r_slots* and *r_bw* annotations are omitted for simplicity.

previous LAWs to make room for the next one, e.g., a backtracking approach, is required. In the next section, we will investigate this performance vs. infeasibility tradeoff further using a large-scale data center scenario, and subsequently propose our Statistical LAW solution to this infeasibility challenge in Section 5.

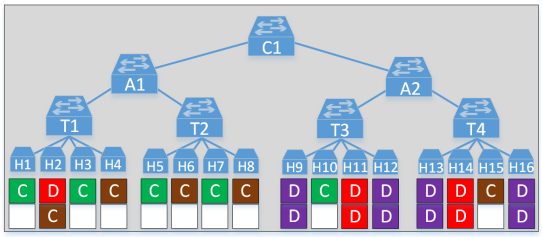
3.4 Applying LAWs to Different Topologies

In this section we describe how to apply LAWs to different types of physical network topologies like fat-tree [23], VL2 [24], and BCube [25]. Because LAWs are topology-dependent, by definition, it is important to understand how LAW construction and allocation may be performed for different types of physical topologies. Fat-tree and VL2 are both multi-rooted hierarchical trees with redundant links connecting devices between levels of the tree hierarchy. BCube is a non-hierarchical topology that uses a number of vertically arranged switches in addition to the ToR switches, which provide end-to-end server communication via server-to-switch network links exclusively, i.e. there are no physical connections between switches.

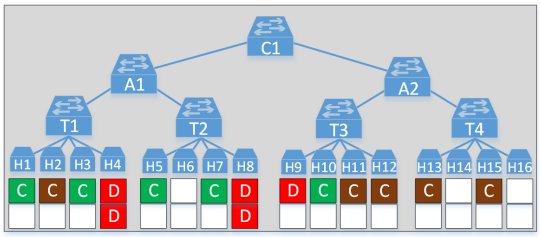
Although the algorithms presented in this chapter assume a single-rooted, simple hierarchical tree topology about which to construct and allocate LAWs, applying the LAW Construction (Alg. 3.2)) and LAW Allocation (Alg. 3.3) algorithms to handle different types of topologies is relatively straightforward. First, Algorithm 3.2 is topology agnostic, and hence applicable "as-is" for constructing LAWs for any arbitrary data center topology, since it takes the physical topology as an input parameter,



(a) "Min Power"



(b) "Min BW"



(c) "Min CRC"

Figure 3.2: LAW allocations using "Min Power," "Min BW," and "Min CRC" heuristics for example scenario.

and subsequently uses an operator-specified per VM allocation method assumed to place the VMs onto the topology in a desirable fashion.

Second, although Algorithm 3.3 is topology dependent, it can be extended to handle multi-rooted, redundant link topologies with the addition of a single "dummy" root node of which each of the core switches comprising the multi-rooted topology are made children of the dummy root using dummy network links of capacity = 0. Redundant links are accounted for by way of setting multiple "uplink BW" variables for devices with multiple available upstream connections. When placing a workload across devices with several available uplinks of sufficient capacity, a method of selecting one must be chosen. Link selection heuristics such as "Min Congestion" or "Min Available BW" may be appropriate. By way of these implementation adjustments, we can soundly perform LAW allocation for multi-rooted and redundant hierarchical network topologies using Algorithm 3.3.

For non-hierarchical data center topologies such as BCube, Algorithm 3.3 can be extended by first using the dummy root strategy described above, and then by making all non-hierarchical component switches children of the root using dummy network links of capacity = 0. However, although this approach remains sound, it may not be very well suited for large non-hierarchical data center networks, as the value of C (maximum number of children of any node), will likely be large due to the high number of dummy root children created when attempting to adapt this approach to handle a non-hierarchical network.

CHAPTER 4:
Evaluation

In this section, we evaluate the performance of proposed LAW based placement heuristics against existing solutions in a simulated (relatively) large-scale data center environment comparable to ones used in related work [8]. The topology consists of 40 aggregation switches, 160 ToR switches (4 x ToRs per aggregate), 2560 hosts (16 x hosts per ToR), and 40960 VM slots (16 x VM slots per host).

4.1 Setup

4.1.1 Representation of Existing Solutions

We choose to adapt the GASO allocation algorithm presented in [9] to represent existing solutions. GASO is chosen because it is extensible to an arbitrary number of disparate objectives and customizable by the operator via weightings for each objective, similar to [5, 6]. For our purpose, the global cost function is a weighted sum of power consumption and mean link bandwidth usage. By default, GASO uses a greedy search for solutions that will minimize the global cost, and as such may converge prematurely to local minima [9]. Therefore, we enhance GASO by implementing a genetic algorithm to diversify candidate solutions as done in related work [9, 10]. We set the candidate solution population size to 10, and perform up to 25 evolutions for each workload request; these parameters are shown to be required for the size of our topology [9, 10].

4.1.2 Per VM Application-Aware Solutions

To make our evaluation more complete, we seek to understand how the performance of LAW based heuristics compare to that of solution that is application-aware but places workloads one VM at a time. We call the latter a per VM application-aware solution. We have developed such a solution by further enhancing GASO. Therefore, we evaluate two distinct implementations of GASO: 1) The "default" version, which weighs power and BW usage highest, and 2) an "application-aware" version, which weighs RCI and CRC highest. Both versions of GASO we use the power and BW usage reduction heuristics from [9]. The application-aware version additionally uses an RCI and CRC reduction heuristic that executes Algorithm 3.1 to move (if necessary) application VMs between hosts in each rack to reduce RCI and CRC. Since the purpose of the default version of GASO is to model non-application-aware solutions, it does not use this application-aware heuristic.

In addition, we refine the LAW based heuristics so that when it is infeasible to allocate a LAW in one atomic step, they resort to per VM allocation for that workload.

4.1.3 Workload Traces

The workloads are randomly generated and their size distributions are comparable to what used in related work [8]. Specifically, four types of workloads are used to model heterogeneous [8] compute-intensive and network-intensive resource requirements:

Type 1: <CI, $numVMs$ x 4 slots, 5 Mbps, 0.5 WCS>

Type 2: <CI, $numVMs$ x 2 slots, 10 Mbps, 0.7 WCS>

Type 3: <DI, $numVMs$ x 1 slot, 60 Mbps, 0.5 WCS>

Type 4: <DI, $numVMs$ x 2 slots, 30 Mbps, 0.7 WCS>

where $numVMs$ ranges between 40 and 200, and also includes the number of slots per VM to represent heterogeneous workload requirements, as done in [8].

In each run, the workload trace is produced by selecting one of the workload types (1-4) and a value for $numVMs$ (40-200) uniformly at random, and deducting the number of slots required of the workload from the total number of slots available in the physical infrastructure. Each randomly generated workload is added to the trace until the next generated workload exceeds the infrastructure slot capacity. After the workload trace is generated, each heuristic under evaluation attempts to place workloads of the trace in the order that they were generated, i.e., online, to the simulated infrastructure.

4.1.4 Performance Metrics

Our evaluation focuses on the following metrics: (1) power usage, 2) BW usage (mean link BW), 3) link congestion (maximum link BW), 4) mean RCI, 5) CRC, and 6) execution time. We also seek to understand the extent of LAW allocation infeasibility tradeoff. Specifically, for each run we identify the fraction of VM capacity allocated when the first infeasible LAW occurs.

4.2 Results

Here, we present the results of LAW and GASO allocation methods over an average of ten workload traces by depicting their performance in power consumption (Fig. 4.1),

BW usage (Fig. 4.2), link congestion (Fig. 4.3), CRC (Fig. 4.4), RCI (Fig. 4.5), and execution time[4] (Fig. 4.6). Plotted points in each figure represent the average objective metric values of a feasible state for a given fraction of VM capacity allocated (i.e., physical infrastructure utilization) over a series of ten runs using different allocation methods. The vertical lines are color coordinated to match the allocation methods and each represents the average capacity allocated when the first infeasibility occurred for the corresponding allocation method.

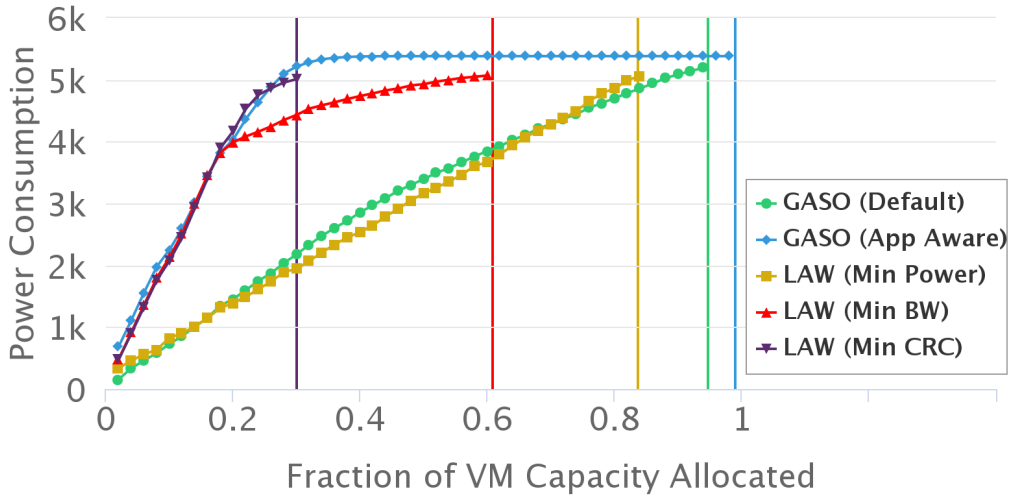


Figure 4.1: Power usage vs. capacity allocated.

Based on these results, we make several observations. First, observe that the "Min Power" LAW allocation heuristic dominates the default GASO per VM allocation method. The "Min Power" LAW allocation heuristic offers similar power conservation, BW usage, and CRC as default GASO, but additionally provides much lower RCI, with an execution time that is an order of magnitude faster. Thus, based on this comparison it is clear that application-aware allocation for this scenario may be strictly superior to non-application-aware allocation, i.e., tenant application objectives may be achieved nearly "for free," with little to no degradation of operator objectives. Next, observe the straightforward tradeoff between power usage and CRC.

Default GASO and "Min Power" maintain relatively high CRC in order to minimize power usage, whereas App-Aware GASO, "Min BW," and "Min CRC" tradeoff higher power consumption for reduced CRC. Furthermore, although App-Aware GASO is designed to jointly minimize RCI and CRC, because it is bounded by the parameters

[4]The simulation consists of approximately 3000 lines of Java code, and was run (serially) using 64-bit JVM. The host PC was running 64-bit Windows on an Intel 2.4 GHz quad-core processor with 24 GB of RAM.

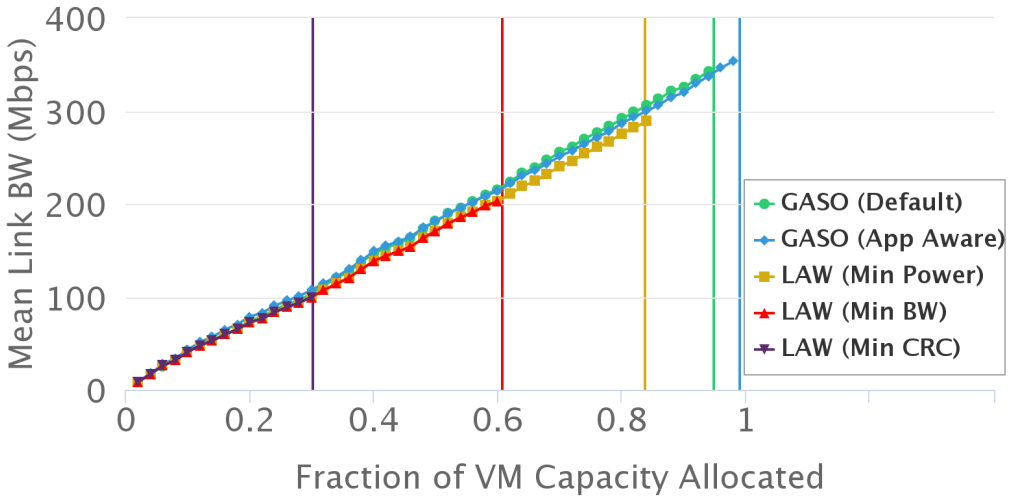


Figure 4.2: Mean BW usage vs. capacity allocated.

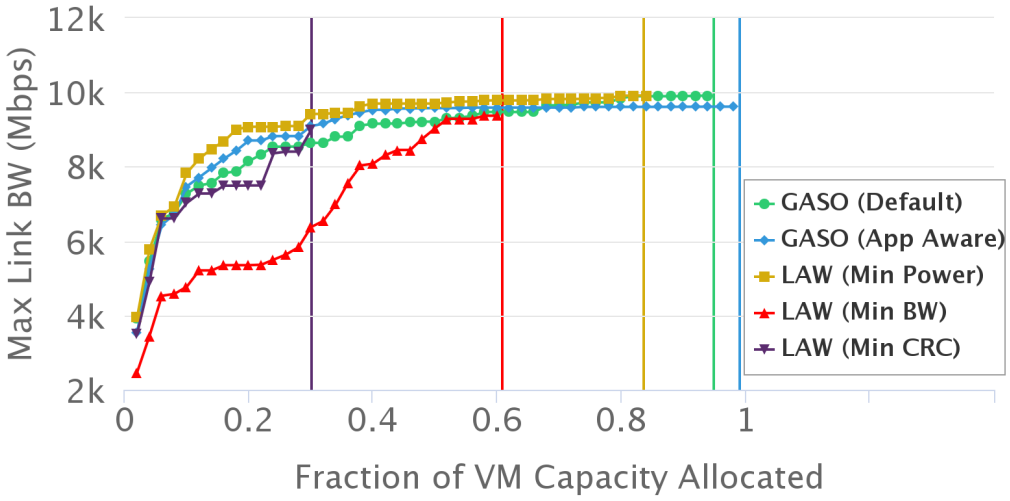


Figure 4.3: Max. link BW usage vs. capacity allocated.

and limitations of a genetic algorithm to perform its search (e.g., population size, number of evolutions, etc), it does not achieve ideal Mean RCI as the physical infrastructure becomes utilized when compared to the LAW allocation heuristics, which each explicitly preserve the RCI for each constructed LAW when making allocations. This RCI gap between GASO App-Aware and LAW becomes more pronounced as the infrastructure becomes more utilized. At 30% utilization, RCI for GASO App-Aware is only slighty higher than LAW, but at 60% utilization, RCI for GASO App-Aware is more than double that of LAW.

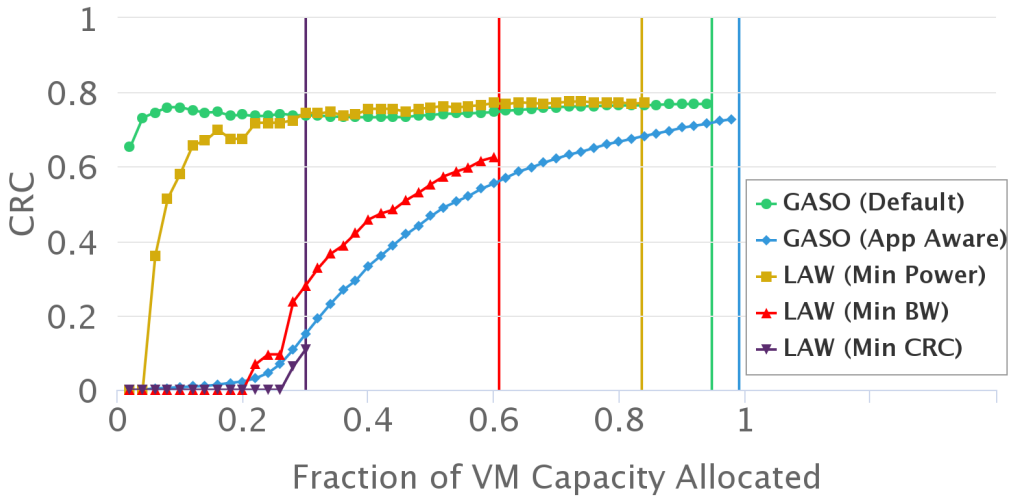


Figure 4.4: CRC vs. capacity allocated.

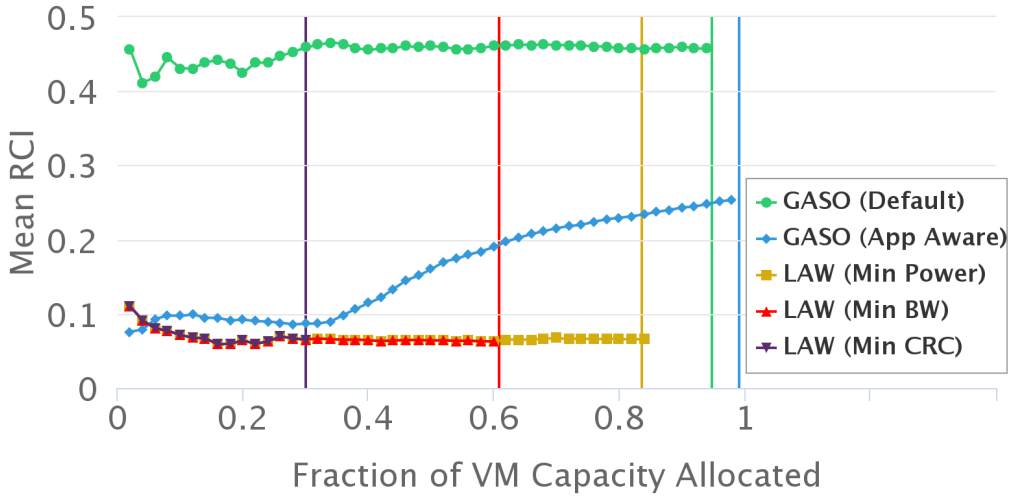


Figure 4.5: Mean RCI vs. capacity allocated.

Tradeoffs involving bandwidth are less straightforward. LAW allocation preserves the provisioned BW for each constructed LAW, and both GASO variants include mean link BW as an optimization criterion, so differences in BW usage (Fig. 4.2) are relatively insignificant. The maximum link BW plot (Fig. 4.3) provides an indicator of link congestion. Observe that the "Min BW" allocation heuristic provides significantly lower link congestion than the other allocation methods until the infrastructure becomes about 50% utilized.

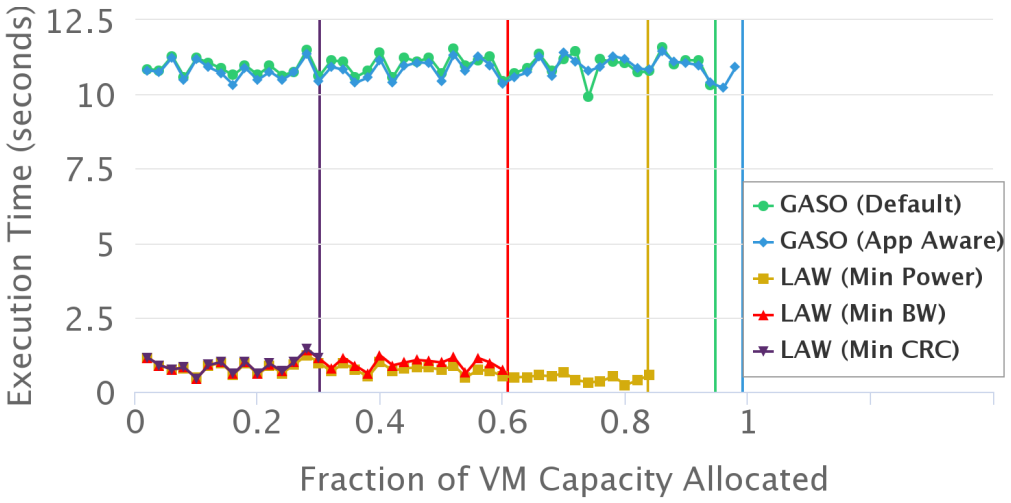


Figure 4.6: Execution time vs. capacity allocated.

Regarding feasibility, observe that each LAW allocation heuristic encounters infeasibility at widely different levels of physical infrastructure utilization. The reason for this is complex, as several factors, including workload size, workload trace composition (CI : DI ratio), physical topology tree structure, physical topology utilization, and heuristic allocation method, determine the likelihood of encountering LAW infeasibility. Moreover, in this scenario there is clear trend indicating that allocation methods that more aggressively attempt to spread CI workloads, i.e., reduce CRC, are more likely to encounter infeasibility sooner. "Min Power" prefers maximum concentration of all workloads and has the highest feasibility rating (83.6%). "Min BW" prefers maximum distribution of all workloads and has moderate feasibility (60.7%). "Min CRC" explicitly spreads CI workloads and has the least feasibility (30.1%).

Finally, observe that "Min CRC," while having relatively low feasibility, appears to dominate GASO App-Aware in this scenario, by achieving lower BW usage, link congestion, CRC, and RCI while using generally the same amount of power. Furthermore, regarding algorithm execution time (Fig. 4.6), LAW allocation (hundreds of milliseconds) is an order of magnitude faster than the GASO variants (tens of seconds).

4.3 LAW for Architectures of Different Dimensions

In this section, we look at how varying the physical dimensions of hierarchical network architectures affect allocation feasibility of different LAW placement heuristics.

Here, physical dimensions are defined in terms the four-tuple (A, T, H, S), where A represents the number of aggregate switches under each core switch, T represents the number of ToR switches under each aggregate switch, H represents the number of host servers under each ToR switch, and S represents the number of VM slots per host server. Regarding different values for these dimensions, we are particularly concerned with how LAW feasibility is affected as the network architecture is "scaled-out," or made more horizontal by increasing the number of physical devices (i.e., increasing the values of A, T, and H), as well as "scaled-up," or made more vertical by increasing the capacity of each physical host server (i.e., increasing the value of S).

In order to provide a fair LAW feasibility comparison across architectures of varying physical proportions, the total VM slot capacity for each architecture is held constant at 10240 slots. Thus, although each architecture evaluated in this section has different values for (A, T, H, S), they all have the same slot capacity, i.e., for each architecture, $A \cdot T \cdot H \cdot S = 10240$. The workload traces used are the same as before (Section 4.1).

Table 4.1 illustrates the feasibility results for each of the "Min Power," "Min BW," and "Min CRC" LAW placement heuristics across a range of hierarchical network architectures with different dimensions and a 10240 VM slot capacity. The range of dimensions in this table represents a spectrum of hierarchical network architectures that are scaled-up and/or scaled-out to varying degrees. The rows closer to the top of the table represent network architectures that are proportionally scaled-up, while those closer to the bottom represent more scaled-out architectures.

From these results, we observe two clear trends, using the (10,4,16,16) dimension architecture as a reference point. First, as an architecture is scaled "out" and "down" from (10,4,16,16) to (20,8,16,4) i.e., approaches dimensions closer to the bottom of Table 4.1, LAW feasibility more than doubles for "Min CRC" (120% increase), and increases significantly for "Min Power" (17% increase), but only increases marginally for "Min BW" (4% increase). Intuitively, these results should be expected: by increasing the number of switches and servers, there is more room to spread CI workloads throughout the infrastructure, resulting in much higher feasibility for "Min CRC," but less so for the other heuristics, as their allocation feasibilities are not as dependent as "Min CRC" on the ability of the infrastructure to handle distributed CI workloads.

Second, as an architecture is scaled "up" and "in" from (10,4,16,16) to (5,4,16,32), i.e., approaches dimensions closer to the top of Table 4.1, LAW feasibility drastically decreases for all heuristics due to the inability to spread CI workloads throughout the infrastructure. Although allocation feasibilities for "Min Power" and "Min BW" are generally not as infrastructure-sensitive as "Min CRC," as the network architecture approaches more vertical physical topologies like (5,4,16,32), the resultant inability

Allocation Heuristic / Physical Dimensions	**Min Power**	**Min BW**	**Min CRC**
(5, 4, 16, 32)	0.44	0.18	0.27
(5, 4, 32, 16)	0.58	0.38	0.28
(5, 8, 16, 16)	0.72	0.33	0.33
(10, 4, 16, 16)*	0.70	0.48	0.30
(10, 8, 8, 16)	0.72	0.48	0.38
(10, 8, 16, 8)	0.77	0.50	0.56
(10, 16, 16, 4)	0.81	0.50	0.63
(20, 8, 16, 4)	**0.82**	**0.50**	**0.66**

Table 4.1: LAW feasibility results for "Min Power," "Min BW," and "Min CRC" for different 10240 VM slot infrastructures of varying dimensions. The values denote the average physical infrastructure utilization (fraction of VM capacity allocated) when the corresponding allocation heuristic and LAW type first encounter infeasibility. Poor results (less than 50%) are shaded red, moderate (at least 50% but less than 80%) yellow, and good (at least 80%) green. * reference dimensions

to spread CI workloads begins to hinder allocation feasibility regardless of heuristic type (less than 50% feasibility across all heuristic types in this example, as illustrated by the first row of Table 4.1).

Therefore, based on these results, we conclude that scaled-out architectures are generally better for achieving higher LAW feasibility versus scaled-up architectures. However, this problem of finding the physical network architecture that yields the highest LAW feasibility for a given workload forecast (e.g., trace) and LAW allocation heuristic seems like a fertile ground for future work.

CHAPTER 5:
Statistical LAWs

As seen from the the execution time plot (Figure 4.6), the GASO per VM workload placement algorithm runs an order of magnitude slower than LAW allocation, and as such, is less suitable for online workload placement tasks, particularly for large data center networks. The GASO execution time is comparable to other workload placement solutions that use genetic algorithms[5], such as [9, 10]. Therefore, although per VM allocation may be used to address the problem of allocating a workload when LAW infeasibility is encountered, we argue that an online solution that gracefully relaxes the ideal application allocation, as represented by the LAW structure, is preferable in the interests of both reduced RCI and reduced execution time. We also considered an alternative LAW backtracking approach to address the issue of LAW infeasibility, but such an approach disrupts current allocations and initial evaluations significantly degraded execution time, so we defer the exploration of such alternative options to future work.

Therefore, we propose a Statistical LAW allocation strategy to explore the LAW feasibility vs. performance tradeoff. A Statistical LAW is natural LAW relaxation which offers a compromise between complete LAW and per VM placement approaches, by representing a percentage of the workload VMs as a "relaxed" LAW, and considering the remainder of the workload VMs as individual units of allocation for some per VM allocation method.

5.1 Example

The 70% Statistical LAW model of workload R4 from the Chapter 2 example scenario is illustrated in Figure 5.1, and represents the preservation of at least 70% of the original LAW VMs (6 VMs) while using per VM allocation for the remainder (2 VMs). Figure 5.2 depicts the resultant network state after using the "Min CRC" LAW heuristic to allocate the 70% LAW for R4 (Figure 5.1) to the physical network state depicted in Figure 3.2c.

By using a statistical measure to relax LAW VM positioning requirements, a compromise between ideal workload allocation and feasibility is achieved, while still maintaining the fast LAW allocation times compared to traditional per VM approaches.

[5]Greedy heuristics, like those used in [7, 8] run faster (order of seconds) for placing large workloads, but they are prone to suboptimal convergence to local minima when multiple objectives are considered.

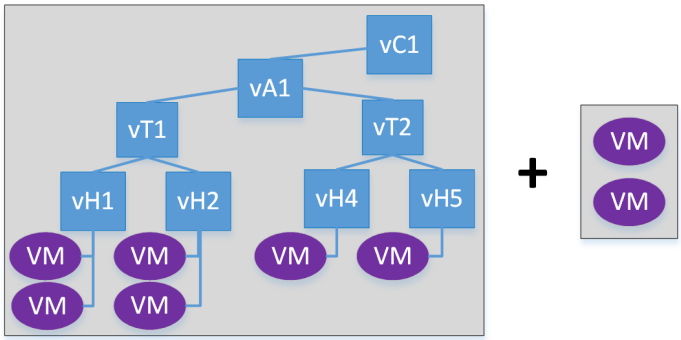


Figure 5.1: Statistical LAW (70%) for workload R4 in example scenario.

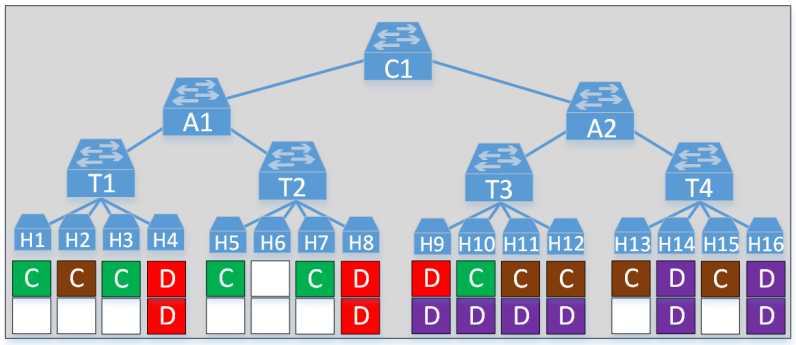


Figure 5.2: "Min CRC" LAW allocation using a 70% Statistical LAW to allocate previously infeasible workload R4 for the example scenario.

5.2 Construction and Allocation

An x% Statistical LAW for some original LAW L is constructed by removing $(100-x)$% of L's VMs (floor) from L by removing the VMs sequentially, where each VM is removed from the LAW host containing the most VMs (or from a host under the most populated ToR in the case of a tie). The removed VMs are maintained with the LAW, and are allocated in a per VM fashion after the Statistical LAW is allocated. For Statistical LAW, the goal of the per VM allocation heuristic for the "remainder" VMs is to minimize RCI subject to WCS constraints. Hence, for CI workloads, the per VM allocations strive to minimize host affinity by allocating the VMs to hosts with the fewest number of CI slots allocated. For DI workloads, these allocations seek to maximize host affinity by concentrating the VMs as closely as possible to the rest of the LAW VMs without violating the WCS bound.

5.3 Statistical LAW Results

Here, we present the results of Statistical LAW allocation using the same workload traces and simulated physical infrastructure as the Chapter 4 evaluation. Statistical LAW allocation methods use a progressive backoff approach. For each workload in the trace, first complete LAW allocation is attempted, then progressively 90%, 70%, 50%, and finally 30% Statistical LAW allocations are attempted if the previous Statistical LAW allocation attempt failed to feasibly allocate the workload. For Statistical LAW, the workload is considered infeasible only if all allocation attempts (complete, 90%, 70%, 50%, 30%) fail. Of course, an even lower Statistical LAW, such as 10% LAW may be attempted if 30% LAW allocation fails, but at that point we observe the resultant LAW structure would be too degraded to provide much benefit.

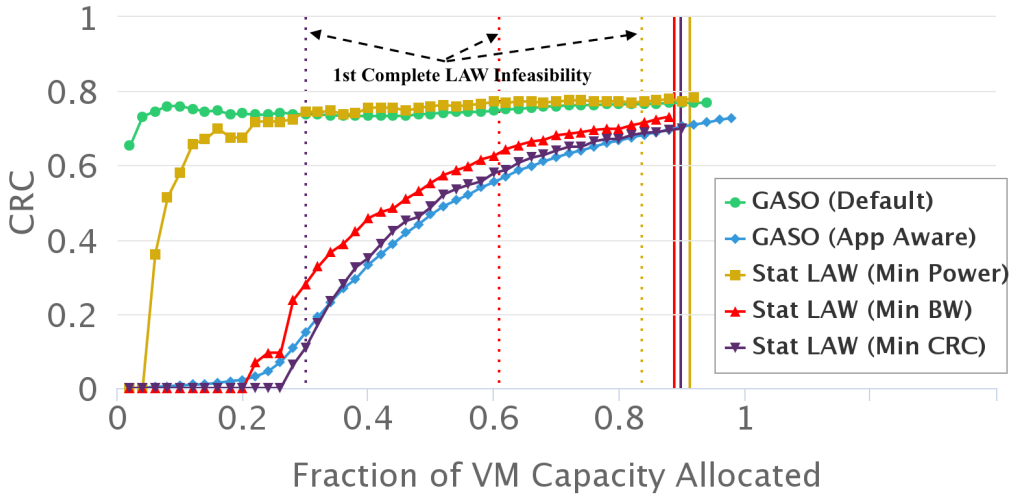


Figure 5.3: Effect of Statistical LAW on CRC.

The Statistical LAW results as shown in Table 5.1 clearly demonstrate the advantage of using a Statistical LAW approach vs. per VM allocation when complete LAW allocation is not feasible. Notably, in this scenario, the Statistical LAW allocation performs similarly to complete LAW allocation, with both RCI and execution time increased by only a small constant factor as workloads become overwhelmingly infeasible for Statistical LAW, which begins to occur around 85% infrastructure utilization, as seen in Figures 5.4 and 5.5, respectively. In other words, for each resource usage metric, there is very little performance degradation for using Statistical LAW compared to complete LAW allocation, as can be seen by observation of plotted points for Statistical LAW allocation methods beyond the amount of physical infrastructure utilization at which complete LAW allocation fails (values in the first row of Table 5.1, denoted by dotted vertical lines in each figure).

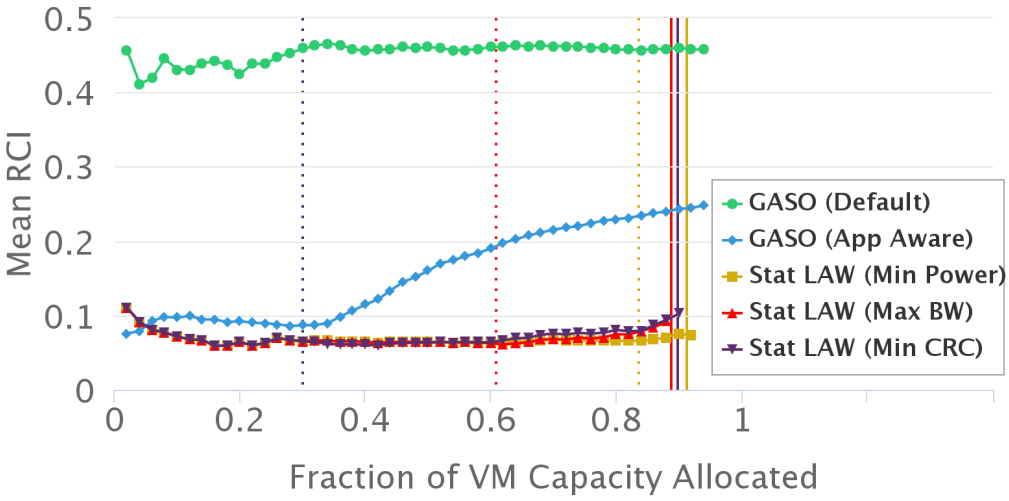


Figure 5.4: Effect of Statistical LAW on Mean RCI.

Allocation Heuristic / LAW Type	**Min Power**	**Min BW**	**Min CRC**
Complete LAW	0.84	0.61	0.30
90% Stat. LAW	0.84	0.63	0.35
70% Stat. LAW	0.86	0.67	0.62
50% Stat. LAW	0.87	0.80	0.86
30% Stat. LAW	0.91	0.89	0.90

Table 5.1: Statistical LAW feasibility results for "Min Power," "Min BW," and "Min CRC" for the large-scale evaluation. The values denote the average physical infrastructure utilization (fraction of VM capacity allocated) when the corresponding allocation heuristic and LAW type first encounter infeasibility.

For instance, in Figure 5.3, although the "Min CRC" LAW placement heuristic begins to incur slightly higher CRC than App-Aware GASO at approximately 40% infrastructure utilization, remains very similar (within a few percentage points) throughout the entire trace.

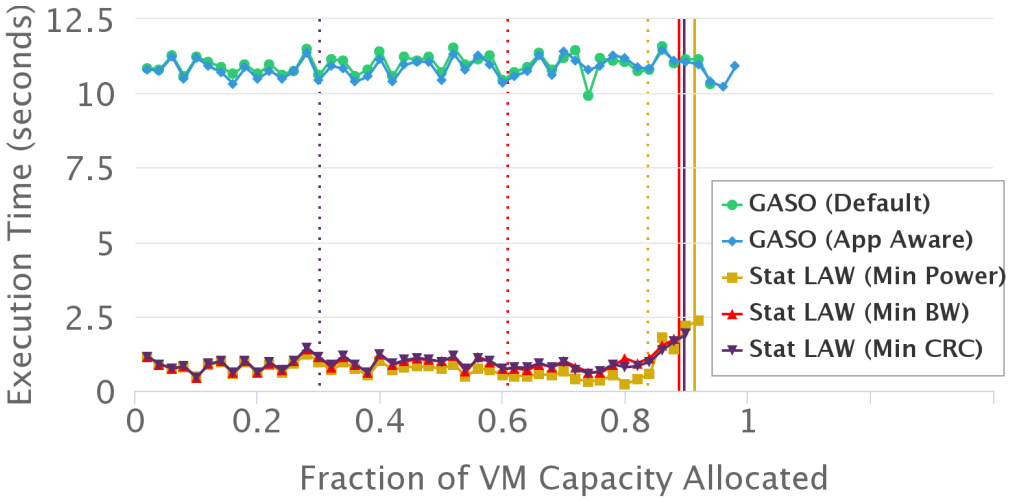


Figure 5.5: Effect of Statistical LAW on execution time.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
LAWs for Workload Prioritization

As demonstrated in the previous section, using Statistical LAW allocation may greatly improve LAW allocation feasibility versus complete LAW allocation, while still producing resource and application efficient allocations with fast workload placement times. However, because Statistical LAW may result in increased application RCI for statistically allocated workloads, the operator may prefer to take a proactive approach in applying Statistical LAW allocations to low priority workloads in order to increase the LAW feasibility of high priority workloads to ensure minimum RCI for them. In this section, we propose the Early Statistical LAW allocation strategy, a proactive approach that trades off suboptimal placement of low priority workloads to increase LAW feasibility for high priority workloads.

6.1 Early Statistical LAW Allocation

In this work, we explore a simple binary priority scheme and leave more elaborate prioritization to future work. We seek to increase the feasibility of high priority workloads by allocating low priority workloads using Early Statistical LAW, which differs from default Statistical LAW allocation only in the per VM allocation heuristic used to allocate the non-LAW "remainder" VMs. While per VM allocations using default Statistical LAW strive to minimize RCI, Early Statistical LAW uses these per VM allocations as a "compromise" to increase the feasibility for future high priority LAWs.

We achieve these compromise allocations by placing the remainder CI VMs in the infrastructure subtree with the fewest available slots using the classical "tightest fit" bin packing heuristic, with a preference for placement on hosts with more DI VMs. Intuitively, using such a heuristic should provide more "contiguous" VM slot space in other infrastructure subtrees, thus providing a higher likelihood of LAW feasibility for future high priority workloads. Because this additional contiguous slot space comes at the cost of bandwidth efficiency, we do not compromise the placement of DI VMs, but instead allocate them in the same fashion as default Statistical LAW: with maximum intra-application host affinity while satisfying WCS constraints.

6.2 Early Statistical LAW Results

We evaluate Early Statistical LAW using the same scenario as before except this time we randomly assign each workload a binary priority value ("high" or "low"). We compare the performance of Early Statistical LAW to default Statistical LAW, specifically with regard to the feasibility of high priority workloads. Each low priority workload is allocated as a 50% Statistical LAW, using the compromise placement heuristic described in the preceding paragraph, while high priority workloads are allocated identically to Statistical LAW, using the same progressive backoff schedule.

Allocation Heuristic / High Priority LAW Type	**Min Power**	**Min BW**	**Min CRC**
Complete LAW	0.95 **(14.5%)**	0.63 (5.0%)	0.43 **(43.3%)**
90% Stat. LAW	0.96 **(14.3%)**	0.67 (8.1%)	0.51 **(45.7%)**
70% Stat. LAW	0.97 **(12.8%)**	0.85 **(28.8%)**	0.80 **(29.0%)**
50% Stat. LAW	0.98 **(12.6%)**	0.89 **(11.3%)**	0.97 **(12.8%)**
30% Stat. LAW	0.99 (8.8%)	0.96 (7.9%)	0.99 **(11.2%)**

Table 6.1: Early Statistical LAW feasibility results for high priority workloads in the large-scale evaluation. The raw values denote the average physical infrastructure utilization when the corresponding allocation heuristic and LAW type first encounter high priority workload infeasibility. The values in parentheses denote the percentage of feasibility increase over default Statistical LAW allocation.

Table 6.1 illustrates the merits of using Early Statistical LAW to improve feasibility for high priority workloads. By proactively allocating low priority workloads using Early Statistical LAW, high priority feasibility is increased for each LAW type versus default Statistical LAW allocation. Consider "Min CRC," the LAW allocation heuristic that is generally the least feasible. By using the Early Statistical LAW allocation strategy, complete LAW feasibility is increased by 43.3% for high priority workloads. However, while Early Statistical LAW allocation leads to better placements for high priority workloads (i.e., closer to desired, more likely to meet SLA), this increased feasibility comes at the cost of increased RCI, depicted in Figure 6.1.

Regarding this feasibility vs. RCI tradeoff, it is clear that the increase in RCI with Early Statistical LAW is a result of suboptimal allocations for low priority workloads, since complete LAW allocations preserve RCI and high priority LAW feasibility is increased using this strategy. This feasibility vs. RCI tradeoff should be considered

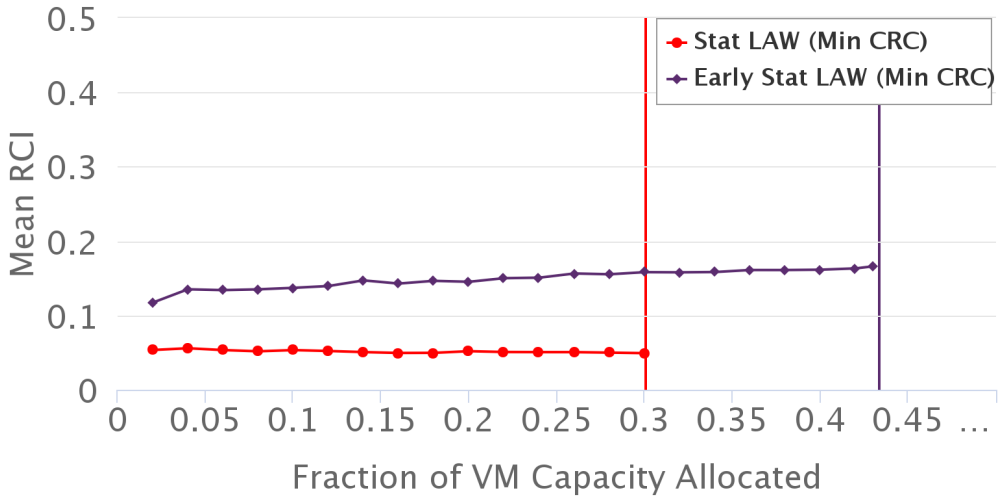


Figure 6.1: RCI vs. feasibility tradeoff introduced by Early Statistical LAW (using Min CRC heuristic).

by the operator when making a determination to use Early Statistical LAW. For example, suppose an operator decides to use the "Min CRC" LAW placement heuristic based on forecasting evidence that more CI workloads than DI workloads are likely to arrive. Since LAW placement using "Min CRC" has been shown to have relatively low LAW feasibility, he/she may want to invoke Early Statistical LAW allocation for low priority workloads, especially if there are high priority tasks or tenants active, and accept higher RCI for low priority workloads as a tradeoff. On the other hand, if there are relatively few high priority tasks or tenants active, or if the operator is unsure of the types of workloads likely to arrive, then he/she may decide to use the "Min Power" LAW placement heuristic with no Early Statistical LAW allocations, since "Min Power" has been shown to have relatively high complete LAW feasibility as is, and the relatively benign workload forecast may allow electrical cost savings while still providing good RCI for active tasks and tenants.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 7:
Related Work

Prior work on application-aware workload placement mostly focuses on maximizing the performance of certain applications (e.g., data intensive [26], high-performance computing applications [27], etc.) or optimizing specific aspects of application performance (e.g., network throughput [4, 28], fairness [2]). Other related solutions such as CloudMirror [7] and Ostro [8] provide bandwidth guarantees and ensure high application availability. To the best of our knowledge, this work is the first to propose precomputing desired workload placements for individual applications (i.e., LAWs) and subsequently using them to speed up and prioritize workload placement while meeting per application performance requirements.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 8:
Conclusion

We have demonstrated that an application-aware approach, by optimizing the relative positioning of VMs of individual workloads, can meet both per application requirements and cumulative resource usage goals. Furthermore, we have proposed a new abstraction (i.e., LAW) to enable online workload placement that is one order of magnitude faster than existing solutions, and scalable to large data center scenarios.

In the big picture, we view this work as a first step towards understanding the tradeoff between maximizing application performance and optimizing network-wide resource usage. We believe there is a wide design space for new formulations and heuristics to meet specific combinations of application-level requirements and operational goals.

THIS PAGE INTENTIONALLY LEFT BLANK

References

[1] "Practical guide to cloud service agreements, version 2.0," Cloud Standards Customer Council, Tech. Rep., Apr. 2015.

[2] J. Li, D. Li, Y. Ye, and X. Lu, "Efficient multi-tenant virtual machine allocation in cloud data centers," *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 81–89, Feb 2015.

[3] F. Hao, M. Kodialam, T. V. Lakshman, and S. Mukherjee, "Online allocation of virtual machines in a distributed cloud," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, April 2014, pp. 10–18.

[4] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–9.

[5] J. C. Mogul, A. AuYoung, S. Banerjee, L. Popa, J. Lee, J. Mudigonda, P. Sharma, and Y. Turner, "Corybantic: Towards the modular composition of sdn control programs," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, ser. HotNets-XII. New York, NY, USA: ACM, 2013, pp. 1:1–1:7. [Online]. Available: http://doi.acm.org/10.1145/2535771.2535795

[6] A. AuYoung, Y. Ma, S. Banerjee, J. Lee, P. Sharma, Y. Turner, C. Liang, and J. C. Mogul, "Democratic resolution of resource conflicts between sdn control programs," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: ACM, 2014, pp. 391–402. [Online]. Available: http://doi.acm.org/10.1145/2674005.2674992

[7] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 467–478, Aug. 2014. [Online]. Available: http://doi.acm.org/10.1145/2740070.2626326

[8] G. Jung, M. A. Hiltunen, K. R. Joshi, R. K. Panta, and R. D. Schlichting, "Ostro: Scalable placement optimization of complex application topologies in large-scale data centers." in *ICDCS*. IEEE, 2015, pp. 143–152. [Online]. Available: http://dblp.uni-trier.de/db/conf/icdcs/icdcs2015.html#JungHJPS15

[9] A. Bairley and G. G. Xie, "Orchestrating network control functions via comprehensive trade-off exploration," in *IEEE NFV-SDN '16*. Palo Alto, CA: IEEE, Nov 2016.

[10] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in *IFIP/IEEE International Symposium on Integrated Network Management, IM 2015, Ottawa, ON, Canada, 11-15 May, 2015*, 2015, pp. 89–97. [Online]. Available: http://dx.doi.org/10.1109/INM.2015.7140280

[11] A. Gulati, N. Kodirov, and G. Kulkarni, "Quantifying the noisy neighbor problem in openstack," in *2016 OpenStack Summit*, Austin, TX, Apr. 2016. [Online]. Available: https://www.openstack.org/assets/presentation-media/ZeroStack-Austin-Presentation.pdf

[12] S. Srikanthan, S. Dwarkadas, and K. Shen, "Data sharing or resource contention: Toward performance transparency on multicore systems," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. Santa Clara, CA: USENIX Association, Jul. 2015, pp. 529–540. [Online]. Available: https://www.usenix.org/conference/atc15/technical-session/presentation/srikanthan

[13] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM TRANSACTIONS ON NETWORKING*, vol. 1, 1993.

[14] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 242–253. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018465

[15] P. Bodik, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 431–442. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342439

[16] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 17–17. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855728

[17] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, p. 1, 2013.

[18] C. C. Lin, P. Liu, and J. J. Wu, "Energy-efficient virtual machine provision algorithms for cloud systems," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, Dec 2011, pp. 81–88.

[19] R. E. Burkard and E. Çela, "Linear assignment problems and extensions," Boston, MA, pp. 75–149, 1999. [Online]. Available: http://dx.doi.org/10.1007/978-1-4757-3023-4_2

[20] H. W. Kuhn and B. Yaw, "The hungarian method for the assignment problem," *Naval Res. Logist. Quart*, pp. 83–97, 1955.

[21] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society of Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, March 1957.

[22] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[23] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402967

[24] 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 *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 51–62. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592576

[25] 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 *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592577

[26] J. T. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," in *Proceedings of the 2010 Ninth International Conference on Grid and Cloud Computing*, ser. GCC '10.

Washington, DC, USA: IEEE Computer Society, 2010, pp. 87–92. [Online]. Available: http://dx.doi.org/10.1109/GCC.2010.29

[27] A. Gupta, L. V. Kale, D. Milojicic, P. Faraboschi, and S. M. Balle, "Hpc-aware vm placement in infrastructure clouds," in *Proceedings of the 2013 IEEE International Conference on Cloud Engineering*, ser. IC2E '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 11–20. [Online]. Available: http://dx.doi.org/10.1109/IC2E.2013.38

[28] D. S. Dias and L. H. M. Costa, "Online traffic-aware virtual machine placement in data center networks," in *2012 Global Information Infrastructure and Networking Symposium (GIIS)*. IEEE, 2012, pp. 1–8.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California