NATIONAL TECHNICAL UNIVERSITY OF ATHENS

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF COMMUNICATION, ELECTRONIC AND INFORMATION ENGINEERING

EFFICIENT RESOURCE ALLOCATION FOR DATA CENTERS WITH DYNAMIC OPTICAL NETWORK INFRASTRUCTURE

DOCTORAL DISSERTATION OF

KONSTANTINOS G. KONTODIMAS

Dipl.-Ing. Computer & Informatics Engineer, MSc

SUPERVISOR:

Emmanouel Varvarigos

Professor, NTUA

ATHENS, November 2023

This research is co-financed by Greece and the European Union (European Social Fund – ESF) through the Operational Programme «Human Resources Development, Education and Lifelong Learning» in the context of the project "Strengthening Human Resources Research Potential via Doctorate Research – 2nd Cycle" (MIS-5000432), implemented by the State Scholarships Foundation (ΙΚΥ).

NATIONAL TECHNICAL UNIVERSITY OF ATHENS

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF COMMUNICATION, ELECTRONIC AND INFORMATION ENGINEERING

EFFICIENT RESOURCE ALLOCATION FOR DATA CENTERS WITH DYNAMIC OPTICAL NETWORK INFRASTRUCTURE

DOCTORAL DISSERTATION OF

KONSTANTINOS G. KONTODIMAS

Dipl.-Ing. Computer & Informatics Engineer, MSc

Approved by the seven-member examination committee on November 29th, 2023.

THREE-MEMBER ADVISORY COMMITTEE:

1. E. Varvarigos, Professor, NTUA (Supervisor)
2. H. Avramopoulos, Professor, NTUA
3. S. Papavassiliou, Professor, NTUA

SEVEN-MEMBER EXAMINATION COMMITTEE:

1. E. Varvarigos, Professor, NTUA
2. H. Avramopoulos, Professor, NTUA
3. S. Papavassiliou, Professor, NTUA
4. T. Varvarigou, Professor, NTUA
5. K. Christodoulopoulos, Assistant Professor, University of Athens
6. P. Kokkinos, Assistant Professor, University of Peloponnese
7. K. Yiannopoulos, Associate Professor, University of Peloponnese

ATHENS, November 2023

To all remarkable individuals who have contributed to breaking down barriers and challenging the norms.

They give us a thread to follow.

.....

Konstantinos G. Kontodimas
Computer and Informatics Engineer
Doctor of Engineering of NTUA

Summary

We investigate resource orchestration in a data center interconnection network, which relies on hybrid electro-optical top-of-rack switches to interconnect servers over multi-wavelength optical rings. The bandwidth of the rings is shared, and an efficient utilization of the infrastructure calls for coordination in the time, space, and wavelength domains. To this end, we present offline and incremental dynamic resource assignment algorithms. The algorithms are suitable for implementation in a software defined network control plane, achieving efficient, collision-free, and on demand capacity use. Our simulation results indicate that the proposed algorithms can achieve high utilization and low latency in a variety of traffic scenarios that include hot spots and/or rapidly changing traffic. Furthermore, we evaluate the effect of the control plane delay and traffic estimation policies, using the OMNET++ packet-level simulator with realistic MapReduce traffic.

The introduction of all-optical switching in data center interconnection networks (DCN) is an important step in addressing the shortcomings of current electronic switching solutions. However, limitations in optical switch port count, and reconfiguration speed require new designs for optical switching and DCNs. In this dissertation, we propose a DCN fabric that relies on a "Lean" optical switch design with limited but scalable configurability. This design offers high reconfiguration speeds, real-time scheduling, efficient network control, and a low number of switching elements. To achieve these objectives, we relax the non-blocking network requirement and opt for partially configurable switching modules, limiting the control capability of the scheduler and reducing control overhead. We compare our proposed network with the RotorNet architecture, which operates with fully distributed control, and the Mordia architecture, which operates with centralized control. Each architecture achieves varying levels of functionality and offers distinct advantages. The proposed solution lies in the middle of the other two approaches and combines the benefits of both of them. Additionally, we analyze the crosspoint complexities of the proposed and the aforementioned reference architectures, and evaluate their throughput and latency performance through simulations. Finally, we enhance RotorNet using breakout to control latency, partial configurability with centralized control, and an adaptive scheduling policy that learns and optimizes resource allocation dynamically.

Distributed storage systems spanning across different cloud data centers have substantially improved availability and flexibility for data storage and retrieval operations. However, stringent latency requirements of emerging applications necessitate optimized selection of storage resources that exhibit smaller delay. Introducing edge resources into distributed storage systems enables data placement closer to its source, but simultaneously increases the complexity of decision-making and orchestration processes for optimal data placement. In this work, we develop mechanisms for storing data across an infrastructure that includes both edge and cloud resources. Our approach focuses on optimizing data integrity, longevity, security, and cost, while leveraging erasure coding when performing the resource allocation. We first present a comprehensive mixed integer linear programming formulation of the storage resource orchestration problem. As the search space for the optimal solution can be vast and the execution time prohibitively large for real size problems, we also propose an innovative multi-agent heuristic approach that uses the rollout, a reinforcement based policy, to balance performance and execution time efficiently. Through various simulation experiments, we evaluate the developed mechanisms and trade-offs involved in our approach. By incorporating data from a multi-cloud provider, we further enhance the validity of the simulations and the conclusions drawn.

Keywords – Data center networks, Interconnection networks, Cloud computing, Edge computing, Distributed storage systems, Optical networks, Data security, Optimization methods, Erasure coding

Περίληψη

Μελετούμε την ενορχήστρωσης πόρων σε διασυνδετικό δίκτυο κέντρου δεδομένων (ΔΚΔ) που βασίζεται σε υβριδικούς ηλεκτροοπτικούς top-of-rack μεταγωγείς για τη σύνδεση των εξυπηρετητών μέσω οπτικών δακτυλίων πολλαπλών μηκών κύματος. Το εύρος ζώνης των δακτυλίων είναι κοινόχρηστο, και η αποδοτική χρήση της υποδομής απαιτεί συντονισμό στα πεδία του χρόνου, του χώρου και του μήκους κύματος. Για τον σκοπό αυτό, παρουσιάζονται offline και incremental δυναμικοί αλγόριθμοι ανάθεσης πόρων. Οι αλγόριθμοι αυτοί είναι κατάλληλοι να υλοποιηθούν σε επίπεδο ελέγχου που χρησιμοποιεί software defined network (SDN), επιτυγχάνοντας αποδοτική, χωρίς συγκρούσεις και κατ' απαίτηση χρήση της χωρητικότητας. Τα αποτελέσματα της προσομοίωσής δείχνουν ότι οι προτεινόμενοι αλγόριθμοι μπορούν να επιτύχουν υψηλή χρησιμοποίηση και χαμηλή καθυστέρηση σε διάφορα σενάρια κίνησης που περιλαμβάνουν hot spots και/ή γρήγορα μεταβαλλόμενη κίνηση. Επιπλέον, για την αξιολόγηση της επίδρασης της καθυστέρησης στο επίπεδο ελέγχου και μεθόδων εκτίμησης της κίνησης, γίνεται χρήση ρεαλιστικής κίνησης MapReduce με τον προσομοιώτη επιπέδου πακέτων OMNET++.

Η εισαγωγή της πλήρους οπτικής μεταγωγής στα διασυνδετικά δίκτυα κέντρων δεδομένων (ΔΚΔ) αποτελεί σημαντικό βήμα για την αντιμετώπιση των αδυναμιών των υπάρχοντων λύσεων ηλεκτρονικής μεταγωγής. Ωστόσο, οι περιορισμοί στον αριθμό των θυρών των οπτικών μεταγωγέων και στην ταχύτητα αναδιαμόρφωσης απαιτούν νέες σχεδιαστικές λύσεις για την οπτική μεταγωγή και τα ΔΚΔ. Σε αυτή τη διατριβή, προτείνουμε μια αρχιτεκτονική ΔΚΔ που βασίζεται στον σχεδιασμό ενός οπτικού μεταγωγέα «Lean» με περιορισμένη, αλλά κλιμακούμενη ρυθμισιμότητα (configurability). Αυτός ο σχεδιασμός προσφέρει υψηλές ταχύτητες επαναρρύθμισης, χρονοπρογραμματισμό πραγματικού χρόνου, αποδοτικό έλεγχο του δικτύου και χαμηλό αριθμό στοιχείων μεταγωγής. Για να επιτευχθούν αυτοί οι στόχοι, γίνεται χαλάρωση του non-blocking περιορισμού του δικτύου και επιλέγονται στοιχεία μεταγωγής μερικής ρυθμισιμόττας, περιορίζοντας τη δυνατότητα έλεγχου του χρονοπρογραμματιστή και μειώνοντας την επιβάρυνση της διαδικασίας ελέγχου. Γίνεται σύγκριση του προτεινόμενου δικτύου με την αρχιτεκτονική RotorNet, η οποία λειτουργεί με πλήρως κατανεμημένο έλεγχο, και με την αρχιτεκτονική Mordia, η οποία λειτουργεί με κεντρικοποιημένο έλεγχο. Κάθε αρχιτεκτονική πετυχαίνει

διαφορετικά επίπεδα λειτουργικότητας (functionality) και προσφέρει διαφορετικά πλεονεκτήματα. Η προτεινόμενη λύση βρίσκεται ανάμεσα στις δύο άλλες δύο προσεγγίσεις και συνδυάζει τα οφέλη και των δύο. Επιπλέον, αναλύονται οι crosspoint πολυπλοκότητες της προτεινόμενης αρχιτεκτονικής και των αρχιτεκτονικών αναφοράς και αξιολογούνται ως προς τη ρυθμαπόδοση και την καθυστέρηση μέσω προσομοιώσεων. Τέλος, εφαρμόζονται βελτιώσεις στο ΔΚΔ RotorNet χρησιμοποιώντας τη μέθοδο breakout για τον έλεγχο της καθυστέρησης του επιπέδου ελέγχου, την εφαρμογή μερικής ρυθμισιμότητας με τη βοήθεια κεντρικοποιημένου ελέγχου, και μιας προσαρμοστικής πολιτικής χρονοπρογραμματισμού που βελτιστοποιεί δυναμικά τις αναθέσεις με βάση τα χαρακτηριστικά της κίνησης.

Τα συστήματα κατανεμημένης αποθήκευσης που εκτείνονται σε διαφορετικά κέντρα δεδομένων του cloud έχουν βελτιώσει σημαντικά τη διαθεσιμότητα και την ευελιξία για λειτουργίες αποθήκευσης και ανάκτησης δεδομένων. Ωστόσο, οι αυστηρές απαιτήσεις χρόνου απόκρισης των νέων εφαρμογών απαιτούν τη βέλτιστη επιλογή αποθηκευτικών πόρων που παρουσιάζουν μικρότερη καθυστέρηση. Η εισαγωγή πόρων edge σε συστήματα κατανεμημένης αποθήκευσης επιτρέπει την τοποθέτηση δεδομένων κοντά στην πηγή τους, αλλά ταυτόχρονα αυξάνει την πολυπλοκότητα στις διαδικασίες λήψης αποφάσεων και ενορχήστρωσης για τη βέλτιστη εναπόθεση των δεδομένων. Σε αυτή τη διατριβή, αναπτύσσονται μηχανισμοί για την αποθήκευση δεδομένων σε μια υποδομή που περιλαμβάνει τόσο edge, όσο και cloud πόρους. Η προσέγγισή επικεντρώνεται στον βέλτιστο συνδυασμό της ακεραιότητας των δεδομένων, της μακροβιότητάς τους, της ασφάλειας και του κόστους, ενώ χρησιμοποιείται η τεχνική του erasure coding κατά την ανάθεση των πόρων. Αρχικά παρουσιάζεται μια ολοκληρωμένη διατύπωση μεικτού ακέραιου γραμμικού προγραμματισμού για το πρόβλημα της ενορχήστρωσης των πόρων αποθήκευσης. Δεδομένου ότι ο χώρος αναζήτησης για τη βέλτιστη λύση μπορεί να είναι τεράστιος και ο χρόνος εκτέλεσης απαγορευτικά μεγάλος για προβλήματα πραγματικού μεγέθους, προτείνεται επίσης μια multi-agent rollout ευρετική προσέγγιση, για να ισορροπήσει αποδοτικά μεταξύ απόδοσης και χρόνου εκτέλεσης. Μέσω διαφόρων πειραμάτων προσομοίωσης, αξιολογούνται οι μηχανισμοί και οι συμβιβασμοί αυτής της προσέγγισής. Ενσωματώνοντας πραγματικά δεδομένα που δόθηκαν από πάροχο multi-cloud, ενισχύοντας περαιτέρω την έγκυροτητα των προσομοιώσεων και των συμπερασμάτων που προκύπτουν.

Λέξεις Κλειδιά – Δίκτυα κέντρων δεδομένων, Διασυνδετικά δίκτυα, Υπολογιστικό νέφος, Υπολογιστική παρυφής, Κατανεμημένα συστήματα αποθήκευσης, Οπτικά δίκτυα, Ασφάλεια δεδομένων, Μέθοδοι βελτιστοποίησης, Κωδικοποίηση απαλοιφής

Prologue

In the boundless expanse of human knowledge, the journey of discovery is often characterized by the pursuit of answers to questions that have eluded understanding. This dissertation embarks upon such a journey, a scholarly odyssey that explores the realms of "Efficient Resource Allocation for Data Centers with Dynamic Optical Network Infrastructure" within the Division of Communication, Electronic, and Information Engineering of the School of Electrical and Computer Engineering of the National Technical University of Athens (NTUA), and more specifically, at the High Speed Communication Networks Laboratory (HSCNL). In this prologue, we introduce a deep dive into the subject.

As we peer through the lens of efficient resource allocation for data centers in the context of dynamic optical networks infrastructure, we stand on the precipice of a transformative era. The world is in a constant state of evolution, marked by advancements in technology, shifts in socio-political landscapes, and environmental challenges that demand our collective attention. Amidst these complex and ever-changing dynamics, the study of resource allocation within data centers has emerged as a critical focal point of intellectual inquiry, and this dissertation endeavors to shed light on the intricacies of this multifaceted domain.

Our exploration of efficient resource allocation in this study is marked by a steadfast commitment to knowledge acquisition. It pays tribute to the contributions of predecessors who established the foundational understanding and anticipates the endeavors of future scholars who will further develop upon this groundwork. With the ongoing dedication of scholars, both past and present, this dissertation aims to make a meaningful and innovative addition to the collective knowledge base.

I am deeply grateful to Professor Emmanouel Varvarigos for serving as my supervisor, as his wisdom, mentorship, and support have been instrumental in shaping the direction of this dissertation. Furthermore, I would like to extend my gratitude to the Assistant Professors Konstantinos Christodoulopoulos and Panagiotis Kokkinos, to the Associate Professor Konstantinos Yiannopoulos, as well as to the postdoctoral researchers of the HSCNL, Dr. Polyzois Soumplis and Dr. Aristotelis Kretsis, who have been invaluable collaborators in this academic journey, and whose contributions, insights, and dedication have enriched the research process and enhanced the quality of this work.

To Nikos, Olga, Danae, Nagia, Iro, and all the other dear colleagues and comrades, with whom we managed to make a collective step toward an academic future we dream of.

Last but not least, a deep appreciation to Alexandra Elbakyan. Without her remarkable dedication to providing access to scientific knowledge, I wouldn't have had the chance to read even a fraction of the publications I read during these years.

Throughout the chapters that follow, we will navigate the intricate network of ideas, theories, and empirical evidence that underpin efficient resource allocation in data centers with dynamic optical network infrastructure. We explore data center networks that use combinations of optical and electrical connections for efficient server communication. We've developed algorithms to manage these connections and ensure smooth performance. When it comes to data center switching, we've introduced balanced approaches, offering a mix of speed and control, which falls between other designs. In the realm of data storage, our work focuses on optimizing resource use, whether in cloud or edge environments. We aim to make data management secure, cost-effective, and durable. We've rigorously tested these methods to ensure their practical effectiveness in cloud setups. It is our aspiration that this scholarly endeavor will not only provide new insights and perspectives but also inspire future researchers and enthusiasts to engage in a never-ending quest for understanding.

This dissertation is the result of extensive effort and commitment. As we delve into the topic, it's worth noting that the pursuit of knowledge is a valuable endeavor with the potential to broaden our understanding of the world.

Konstantinos G. Kontodimas
November 2023

Contents

List of Figures

List of Tables

Nomenclature

Functions

δ	Matrix density.
δ_{in}	Intra-POD connection density.
δ_{out}	Inter-POD connection density.
ρ_{sd}	Load intensity between source destination ToR pair (s, d).
ρ	Average network load.
$\mathscr{F}$	Scheduling algorithm.
$\mathscr{G}$	Function for the estimation queue matrix.
$\mathscr{H}$	Critical sum of a matrix.
$\mathscr{I}$	Indicator fuunction where $\mathscr{I}(x) = 1$, when $x > 0$, and 1 otherwise.
$\mathscr{M}$	Number of non-zero entries of a matrix.
$\mathscr{P}$	Returns the load intensity matrix of another matrix.

Sets and Indices

$\mathscr{D}$	Set of all files, indexed by d.
$\mathscr{I}_{km}$	Set of all combinations of $k + m$ nodes, indexed by i.
$\mathscr{K}_d$	Set of fragmentation options of file d, indexed by k.
$\mathscr{M}_d$	Set of encoding schemes of file d, indexed by m.
$\mathscr{N}$	Set of all nodes, indexed by n.

$\mathcal{N}_{\mathrm{r}}^{t}$	Set of selected nodes to retrieve fragments from at retrieval t.
$\mathcal{N}_{\mathrm{s}}$	Set of selected nodes to store fragments to.
$\mathcal{Q}_{d}$	Set of Quality of Service (QoS) requirements of file d.
$\mathcal{T}_{d}$	Set of retrievals of file d, indexed by t.

Fiber Rings Optical Architecture Parameters

δ_{in}	Intra-POD connection density.
δ_{out}	Inter-POD connection density.
ρ	Average network load.
C	Resource allocation cycles (measured in Data periods)
h	Critical sum of a matrix.
I	Number of POD switches per pod/number of ports per ToR switch.
ℓ	Locality parameter: traffic percentage that is destined within the same pod over the total load.
P	Total number of POD switches.
R	Number of fiber rings.
S	Number of servers per rack/number of southbound ports per ToR switch.
T	Number of timeslots/Data period.
W	Number of ToR switches per POD/number of wavelenghts.

Lean/RotorNet Parameters

C	Control cycle delay in Data periods.
m	Number of selector modules per Lean switch.
N	Number of racks/ToR switches
n	Group factor of the Lean switches.
P	Number of servers per rack.

S	Number of spine switches (Lean planes for Lean DCN, Rings with WSSes for Mordia or Rotor switches for RotorNet).
T	Number of timeslots per Data period.
t	Index of Data period.

Distributed Storage Parameters

ϵ	Optimality factor of the erasure code scheme.
ω_d	Index of gateway used for storage of d.
ω'_{dt}	Index of gateway used for retrieval t of d.
ρ	Data size retrieved at each GET request, measured in Data Units.
A_n	Availability probability of node n.
A_{req}	Minimum availability requirement (part of $\mathcal{Q}_d$).
C_n	Storage capacity of node n measured in Data Units.
$D^{\text{enc}}_{\omega_d}$	Encoding/encryption latency per Data Unit in gateway ω_d.
$D^{\text{spl}}_{\omega_d}$	Split latency of a file into two fragments in gateway ω_d.
$D^{\text{dec}}_{\omega'_{dt}}$	Decoding/decryption latency per Data Unit in gateway ω'_{dt}.
$D^{\text{mrg}}_{\omega'_{dt}}$	Merge latency of two fragments in gateway ω'_{dt}.
$D^{\text{prop}}_{n\omega_d}$	Propagation latency between node n and gateway ω_d.
D^{read}_{n}	Reading latency per Data Unit from node n.
D^{wrt}_{n}	Writing latency per Data Unit to node n.
M_d	Size of file d measured in Data Units.
N_c	Number of cloud nodes.
N_e	Number of edge nodes.
$P_{\text{r}n}$	Monetary cost per GET request for retrieving a fragment from node n.
$P_{\text{s}n}$	Monetary cost per Data Unit for storing a fragment to node n.

T_d Hosting duration of file d.

MILP Variables

ψ_d Integer variable denoting the maximum propagation and writing latency of file d.

ψ'_{dt} Integer variable denoting the maximum reading and propagation latency of the retrieve operation t of file d.

ξ_{nd} Binary variable indicating the fragment and the storage node n with the maximum propagation and writing latency of file d.

ξ'_{ndt} Binary variable indicating the fragment and the storage node n with the maximum reading and propagation latency of the retrieve operation t of file d.

$\hat{k}$ Integer variable describing the required fragments to recover a file, equal to $(1+\epsilon)k$.

υ_{nd} Binary variable indicating whether a fragment of file d is placed at a node n.

x'_{nkmdt} Binary variable indicating that a fragment of d is retrieved from node n during the retrieval t, while the fragmentation option is k and the erasure code is $(k+m,\hat{k})$.

x_{nkmd} Binary variable indicating that a fragment of d is stored to node n, while the fragmentation option is k and the erasure code is $(k+m,\hat{k})$.

y_{kmd} Binary variable indicating the fragmentation option k and the erasure code $(k+m,\hat{k})$ of file d.

z_{ikmd} Binary variable indicating the combination $i \in \mathcal{I}_{km}$ of storage nodes, the fragmentation option k and the erasure code $(k+m,\hat{k})$ of file d.

Abbreviations

AONT All-Or-Nothing-Transform

ATM Asynchronous Transfer Mode

AWG Arrayed Waveguide Grating

BvN Birkhoff-von Neumann

CAWG	Cyclic Arrayed Waveguide Grating
C1…5	Constraint 1…5
COTS	Commodity Off-the-Shelf
CU	Cost Unit
DC	Data center
DCN	Data Center Network
DU	Data Unit
EPS	Electronic Packet Switching
FBB	Full Bisection Bandwidth
HOL	Head-of-Line (blocking)
IaaS	Infrastructure-as-a-Service
KPI	Key Performance Indicator
LVD	Lean Valiant Decomposition
MEMS	Microelectromechanical System
MILP	Mixed Integer Linear Programming
MR	Micro-Ring Resonator
NoC	Network-on-Chip
OBS	Optical Burst Switching
OCS	Optical Circuit Switching
OF	OpenFlow
OPS	Optical Packet Switching
PaaS	Platform-as-a-Service
PU	Period Unit
QoS	Quality of Service

RR	Round-Robin
SaaS	Software-as-a-Service
SC1…3	Scheduling Constraint 1…3
SDN	Software Defined Network
TCP	Transmission Control Protocol
TDMA	Time-Division Multiple Access
TDM	Time Division Multiplexing
ToR	Top-of-Rack
TU	Time Unit
UDP	User Datagram Protocol
VLB	Valiant Load Balancing
VOQ	Virtual Output Queue
WDM	Wavelength Division Multiplexing
WSS	Wavelength Selective Switch

Εκτενής Περίληψη

Ανάθεση Πόρων σε ένα Οπτικό Διασυνδετικό Δίκτυο Κέντρου Δεδομένων με Δακτυλίους Ινών και Μεταγωγείς Επιλογής Μήκους Κύματος

Η ευρεία διαθεσιμότητα εφαρμογών στο cloud προς δισεκατομμύρια τελικούς χρήστες και η εμφάνιση μοντέλων Υποδομής ως Υπηρεσία (Infrastructure-as-a-Service – IaaS) και Πλατφόρμας ως Υπηρεσία (Platform-as-a-Service – PaaS) βασίζονται σε συγκεντρωτικές υπολογιστικές υποδομές, τα κέντρα δεδομένων (ΚΔ). Τα ΚΔ συνήθως αποτελούνται από πολλούς διασυνδεδεμένους διακομιστές (servers) που εκτελούν εικονικές μηχανές (virtual machines). Δεδομένου ότι η κίνηση εντός του ΚΔ (east-west) είναι υψηλότερη από την εισερχόμενη/εξερχόμενη (north-south) κίνηση, και αναμένεται να συνεχίσουν να αυξάνονται και οι δύο, τα δίκτυα των ΚΔ (ΔΚΔ) παίζουν κρίσιμο ρόλο. Απαιτούνται ΔΚΔ υψηλής χωρητικότητας, κλιμακούμενα και ενεργειακά/οικονομικά αποδοτικά που να εκμεταλλεύονται πλήρως τη δυναμική των ΚΔ.

Τα προηγμένα ΔΚΔ βασίζονται σε ηλεκτρονική μεταγωγή με τοπολογίες Fat-Tree. Τα Fat-Tree τείνουν να υποεκμεταλλεύονται τους πόρους, απαιτούν πολλά καλώδια και υφίστανται προβλήματα σχετικά με την επεκτασιμότητα και την ενεργειακή αποδοτικότητα. Για τη μείωση του κόστους, τα Fat-Tree συνήθως είναι oversubscribed (π.χ., 1:4) και δεν προσφέρουν πλήρες εύρος ζώνης διατομής (full bisection bandwidth) που μπορεί να χρειαστεί για ορισμένες εφαρμογές. Η δικτύωση που καθορίζεται από τις ανάγκες των εφαρμογών (application-driven networking), μια αναδυόμενη τάση, θα επωφεληθεί από ένα δίκτυο που αναθέτει ευέλικτα τη χωρητικότητα όπου χρειάζεται. Για να αντιμετωπιστούν τα μειονεκτήματα των Fat-Tree, πολλές πρόσφατες έρευνες πρότειναν υβριδικά ηλεκτρικά/οπτικά δίκτυα κέντρων δεδομένων. Μια από αυτές χρησιμοποιεί ένα ΔΚΔ στο οποίο οι βαριές ροές μεγάλης διάρκειας (elephant) δρομολογούνται εκλεκτικά

↬Η διατριβή έχει γίνει αποδεκτή από τη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (ΣΗΜΜΥ) του Εθνικού Μετσόβιου Πολυτεχνείου (ΕΜΠ) τον Νοέμβριο του 2023 για την απόκτηση **Διπλώματος Διδάκτορος Μηχανικού του ΕΜΠ**.

↬Επικοινωνία με τον Κωνσταντίνο Κοντοδήμα μέσω του email: kontodimask@gmail.com.

μέσω ενός δικτύου οπτικής μεταγωγής κυκλώματος (optical circuit switching – OCS), ενώ η υπόλοιπη κίνηση διέρχεται μέσω ενός δικτύου ηλεκτρονικής μεταγωγής πακέτων (electronic packet switching – EPS). Αυτές οι λύσεις βασίζονται στην αναγνώριση των βαριών ροών (elephant), κάτι πού είναι αρκετά δύσκολο, ενώ παρατηρήθηκε ότι τέτοιες μακρόβιες και βαριές ροές δεν είναι τυπικές, κάτι που δυσκολεύει τη διατήρηση υψηλής χρησιμοποίησης (utilization) του OCS. Αντίθετα, απαιτείται υψηλός βαθμός συνδεσιμότητας (connectivity). Για να επιτραπεί μεγαλύτερη συνδεσιμότητα, ερευνητές πρότειναν και προτυποποίησαν ένα πολύ πυκνό υβριδικό ΔΚΔ που υποστήριζε και multi-hop συνδέσεις, μαζί με μια ειδικά κατασκευασμένη στοίβα ελέγχου. Επιπλέον, μέτρησαν τη συνολική καθυστέρηση, συμπεριλαμβανομένης αυτής του επιπέδου ελέγχου και της επαναρρύθμισης του υλικού του OCS (μικροηλεκτρομηχανικά συστήματα – MEMS), να είναι της τάξης των εκατοντάδων χιλιοστών του δευτερολέπτου. Η multi-hop δρομολόγηση αξιοποιήθηκε ως εκ νέου για διαμοιρασμό οπτικών κυκλωμάτων, για το οποίο αναπτύχθηκε ένα επίπεδο ελέγχου βασισμένο στο OpenFlow, δείχνοντας ότι ο διαμοιρασμός των κυκλωμάτων μειώνει την επίδραση της αργής επαναρρύθμισης (reconfiguration) του OCS.

Άλλα προτεινόμενα διασυνδετικά ΔΚΔ δε χρησιμοποιούν καθόλου ηλεκτρονικούς μεταγωγείς. Το Proteus, είναι μια πλήρως οπτική αρχιτεκτονική ΔΚΔ που βασίζεται στον συνδυασμό μεταγωγέων επιλογής μήκους κύματος (wavelength selective switches - WSSs) και MEMS. Ξανά, η multi-hop μετάδοση χρησιμοποιείται για να επιτύχει υψηλή χρησιμοποίηση, αλλά εξακολουθεί να είναι δύσκολο να αντισταθμιστούν οι αργοί χρόνοι επαναρρύθμισης των MEMS μέσω «έξυπνου» ελέγχου. Άλλες προσεγγίσεις πρότειναν υβριδικές αρχιτεκτονικές OCS και οπτικής μεταγωγής πακέτων/ριπών (optical packet/burst switching – OPS/OBS), ελεγχόμενες μέσω SDN. Προτάθηκαν και διάφορες άλλες αρχιτεκτονικές βασισμένες σε OPS/OBS, αλλά οι τεχνολογίες OPS/OBS δεν είναι ακόμα ώριμες, οπότε ο σημερινός στόχος θα μπορούσε να είναι δίκτυα μικρής κλίμακας με περιορισμένη δυνατότητα αναβάθμισης.

Μια υβριδική αρχιτεκτονική ΔΚΔ με το όνομα Mordia χρησιμοποιεί WSS για να παρέχει χρόνους μεταγωγής 11,5 μs. Το Mordia λειτουργεί με δυναμικό τρόπο για να επιτύχει υψηλή συνδεσιμότητα. Ωστόσο, η κλιμάκωση του Mordia είναι περιορισμένη, καθώς χρησιμοποιεί έναν μόνο δακτύλιο πολυπλεξίας διαίρεσης μήκους κύματος (wavelength-division multiplexing – WDM) του οποίου η χωρητικότητα μπορεί να φιλοξενήσει λίγα racks, ενώ οι αλγόριθμοι ανάθεσης πόρων εμφανίζουν υψηλή πολυπλοκότητα και δεν μπορούν να κλιμακωθούν σε μεγάλα ΚΔ.

Στα πλαίσια του ευρωπαϊκού έργο NEPHELE αναπτύχθηκε ένα οπτικό ΔΚΔ που εκμεταλλεύεται την υβριδική ηλεκτρονική/οπτική μεταγωγή με χρήση ελέγχου SDN για

την αντιμετώπιση των τρεχουσών περιορισμών των ΚΔ. Για να επιτρέψει δυναμικό και αποδοτικό διαμοιρασμό των οπτικών πόρων και επικοινωνίας χωρίς συγκρούσεις, το NEPHELE λειτουργεί με σύγχρονο τρόπο, κάνοντας χρήση χρονοθυρίδων (timeslots). Οι χρονοθυρίδες χρησιμοποιούνται για την επικοινωνία από rack-σε-rack και ανατίθενται δυναμικά, κατόπιν ζήτησης, προκειμένου να επιτευχθεί αποδοτική χρησιμοποίηση, οδηγώντας σε εξοικονόμηση ενέργειας και κόστους. Επιπλέον, χρησιμοποιούνται πολλαπλά μήκη κύματος και οπτικά επίπεδα για να επιτευχθεί μια κλιμακούμενη υποδομή ΔΚΔ με υψηλή χωρητικότητα.

Το δίκτυο NEPHELE βασίζεται σε μεταγωγείς επιλογής μήκους κύματος (WSSs), οι οποίοι είναι πιο γρήγοροι από τα μικροηλεκτρομηχανικά συστήματα (MEMS) και πιο ώριμοι από την οπτική μεταγωγή πακέτων και οπτική μεταγωγή ριπών. Οι γρήγοροι χρόνοι μεταγωγής, μαζί με τη δυναμική λειτουργία των χρονοθυρίδων, παρέχουν υψηλή και ευέλικτη συνδεσιμότητα. Σε σύγκριση με άλλες αρχιτεκτονικές που εξαρτώνται επίσης από τους μεταγωγείς WSS (π.χ. Mordia), το NEPHELE είναι πιο επεκτάσιμο: αποτελείται από πολλαπλούς δακτυλίους πολυπλεξίας διαίρεσης μήκους κύματος (WDM), επαναχρησιμοποιεί τα μήκη κύματος και χρησιμοποιεί οικονομικά παθητικά εξαρτήματα δρομολόγησης και επεκτάσιμα σχήματα χρονοπρογραμματισμού. Στο Κεφάλαιο 2 θεωρούμε την αρχιτεκτονική του NEPHELE ως την αρχιτεκτονική αναφοράς.

Όσον αφορά την ανάθεση των πόρων, προβλήματα χρονοπρογραμματισμού παρόμοια με αυτά που αντιμετωπίζονται σε αυτό το έργο μελετήθηκαν στο παρελθόν για δορυφορικούς μεταγωγείς και μεταγωγείς κατάστασης ασύγχρονης μεταφοράς (asynchronous transfer mode – ATM). Πράγματι, μπορεί κανείς να θεωρήσει ολόκληρο το ΔΚΔ πολλαπλών δακτυλίων αναφοράς ως έναν μεγάλο κατανεμημένο μεταγωγέα. Η κύρια διαφορά αυτού του ΚΔ είναι ότι αφορά μεγάλες εγκαταστάσεις δικτύων και δυναμική λειτουργία πολυπλεξίας διαίρεσης χρόνου (time-division multiplexing – TDM)· συνεπώς, ο στόχος δεν είναι η αυστηρή βελτιστότητα, αλλά η χαμηλή πολυπλοκότητα. Επίσης, επεκτείνονται κατάλληλα προηγούμενοι αλγόριθμοι TDM για να αντιμετωπιστούν ορισμένοι εσωτερικοί περιορισμοί συγκρούσεων που απορρέουν από την αρχιτεκτονική αναφοράς.

Στο Κεφάλαιο 2 προτείνουμε και αξιολογούμε ένα σύνολο αλγορίθμων χρονοπρογραμματισμού για το οπτικό ΔΚΔ που χρησιμοποιεί δακτυλίων οπτικών ινών και μεταγωγείς WSS. Περιγράφουμε λεπτομερώς τον κύκλο ελέγχου της αρχιτεκτονικής αναφοράς, καταγράφουμε τις απαιτήσεις του και παρουσιάζουμε έναν αλγόριθμο για τη βέλτιστη ανάθεση των πόρων. Προτείνουμε επίσης τρεις άπληστους ευρετικούς αλγορίθμους χρονοπρογραμματισμού που μειώνουν τους χρόνους ανάθεσης και αξιολογούμε την απόδοσή τους μέσω προσομοιώσεων. Οι τυχαιοποιημένος (randomized) και ο άπληστος (greedy) γραμμικός ευρετικός αλγόριθμος εμφάνισαν κανονικοποιημένη ρυθμαπό-

δοση (throughput) άνω του 85% για όλα τα εξεταζόμενα σενάρια κίνησης. Ο χρόνος εκτέλεσης του άπληστου γραμμικού ευρετικού μετρήθηκε σε εκατοντάδες χιλιοστά του δευτερολέπτου, ενώ ο υπο-γραμμικός άπληστος ευρετικός αλγόριθμος ήταν πιο γρήγορος, θυσιάζοντας κάποια ρυθμαπόδοση. Μελετήσαμε επίσης την επίδραση στην απόδοση του περιορισμού χρονοπρογραμματισμού που σχετίζεται με την αρχιτεκτονική ΚΔ αναφοράς. Για να αντιμετωπίσουμε την προκύπτουσα μείωση της ρυθμαπόδοσης και την αύξηση του χρόνου εκτέλεσης, προτείναμε μια παραλλαγή της αρχιτεκτονικής που χρησιμοποιεί spectrum-shifted οπτικά επίπεδα και επεκτείναμε τον άπληστο γραμμικό ευρετικό αλγόριθμο για να λειτουργεί σε ένα τέτοιο δίκτυο. Οι προσομοιώσεις έδειξαν ότι η ρυθμαπόδοση και ο χρόνος εκτέλεσης πλησιάζουν την απόδοση ενός δικτύου στο οποίο δεν υπάρχει ο περιορισμός της αρχιτεκτονικής ΚΔ. Οι προτεινόμενοι ευρετικοί αλγόριθμοι επιτυγχάνουν υψηλή ρυθμαπόδοση και χαμηλό χρόνο εκτέλεσης, επιβεβαιώνοντας τη δυναμική και αποδοτική λειτουργία του ΚΔ.

Επιπλέον, εξετάσαμε περαιτέρω τις πολιτικές για τη λήψη καλών εκτιμήσεων του πίνακα ουρών που να προσεγγίζουν το μοτίβο της κίνησης μετά την καθυστέρηση του κύκλου ελέγχου. Διενεργήσαμε προσομοιώσεις χρησιμοποιώντας το OMNET++ με ρεαλιστική κίνηση της εφαρμογής MapReduce. Εξετάσαμε τον αντίκτυπο της χρήσης ενός παράλληλου δικτύου για τα TCP ACKs και ενός ευρετικού γεμίσματος της αχρησιμοποίητης χωρητικότητας. Παρατηρήσαμε ότι και οι δύο τεχνικές βελτιώνουν το makespan. Εξετάσαμε την περίπτωση της εφαρμογής μιας στατικής πολιτικής round-robin και δύο πολιτικών που λαμβάνουν υπόψη το μοτίβο της κίνησης. Παρατηρήσαμε ότι όταν η καθυστέρηση του κύκλου ελέγχου είναι υψηλή, φαίνεται να είναι προτιμότερη η στατική πολιτική round-robin. Οι πολιτικές που λαμβάνουν υπόψη την κίνηση προκαλούν σημαντική βελτίωση στο συνολικό makespan που μπορεί να φτάσει το 48% όταν η βραχυπρόθεσμη δυναμικότητα του φόρτου είναι υψηλή.

Γρήγορα Οπτικά Διασυνδετικά Δίκτυα Κέντρων Δεδομένων με Περιορισμένη Ρυθμισιμότητα

Όπως αναφέρθηκε και πιο πάνω, οι τοπολογίες Fat-Tree για τη διασύνδεση ηλεκτρονικών μεταγωγέων με οπτικές ίνες, υποχρησιμοποιούν πόρους, απαιτούν πολλά καλώδια και μεταγωγείς, παρουσιάζουν προβλήματα στην επεκτασιμότητα και τη δυνατότητα αναβάθμισης και καταναλώνουν υψηλά επίπεδα ενέργειας. Η ενσωμάτωση της οπτικής μεταγωγής στα ΔΚΔ αποτελεί έναν κρίσιμο βήμα προς την αντιμετώπιση των περιορισμών των τοπολογιών Fat-Tree. Ενώ οι οπτικοί μεταγωγείς χρησιμοποιούνται κυρίως για τη μεταγωγή κυκλώματος σε μητροπολιτικά δίκτυα και δίκτυα κορμού, πρόσφατες έρευνες

έχουν προτείνει υβριδικά ΔΚΔ που χρησιμοποιούν ηλεκτρονικούς/οπτικούς μεταγωγείς ως λύση. Αυτές οι μελέτες χρησιμοποιούν οπτικούς μεταγωγείς, οι οποίοι ανακατευθύνουν το φως χωρίς μετατροπή από οποιαδήποτε θύρα σε μια άλλη. Ωστόσο, οι χρόνοι επαναρρύθμισης τους (σε χιλιοστά δευτερολέπτου για μεταγωγείς με μεγάλο radix και σε δεκάδες εκατομμυριοστά του δευτερολέπτου για μεταγωγείς με μικρό radix) αποτελούν πρόκληση για τη χρήση τους σε ΔΚΔ, όπου η γρήγορη ρύθμιση των μεταγωγέων είναι απαραίτητη. Παρά τον περιορισμό αυτό, τα δυνητικά οφέλη της ενσωμάτωσης των οπτικών μεταγωγέων στα ΔΚΔ είναι σημαντικά, και η συνεχιζόμενη έρευνα επικεντρώνεται στην ανάπτυξη πιο αποδοτικών και βιώσιμων λύσεων. Το 2022, η Google ανακοίνωσε ότι τα διασυνδετικά δίκτυα του ΚΔ «Jupiter» θα χρησιμοποιούν δυναμική επαναρρύθμιση της τοπολογίας χρησιμοποιώντας οπτική μεταγωγή κυκλώματος, η οποία έχει εξελιχθεί για να επιτυγχάνει υψηλότερη ταχύτητα, μείωση του κόστους, αποδοτικότητα ισχύος και βελτιστοποίηση στα μήκη των διαδρομών.

Το πρώτο εμπόδιο για την υιοθέτηση των τεχνολογιών οπτικής μεταγωγής στα ΔΚΔ πηγάζει από την ταχύτητα επαναρρύθμισης των (πλήρως) οπτικών μεταγωγέων crossbar. Καθώς το μέγεθος των ΔΚΔ αυξάνεται, οι δυνατότητες για χρήση μεταγωγέων ενός σταδίου (single-stage) μειώνονται και/ή η ταχύτητα επαναρρύθμισης γίνεται απαγορευτικά υψηλή. Αντιθέτως, η χρήση crossbars πολλαπλών σταδίων (multi-stage), κατασκευασμένων με μικρότερες λειτουργικές μονάδες, είναι η μόνη λύση. Ωστόσο, αυτή η λύση πάσχει από τον υψηλό συνολικό αριθμό μεταγωγέων και από την πολυπλοκότητα καλωδίωσης, τα οποία μπορούν με τη σειρά τους να επηρεάσουν το κόστος παραγωγής. Απαιτεί επίσης αυστηρό συγχρονισμό και συντονισμένο έλεγχο των πολλαπλών στοιχείων. Στην αρχιτεκτονική NEPHELE γίνεται χρήση μεταγωγέων WSS (σχετικά) χαμηλού radix, των πεδίων του χώρου (πολλαπλοί δακτύλιοι) και του μήκους κύματος (WDM) για να επιτευχθεί χαμηλή ταχύτητα επαναρρύθμισης και υψηλή χωρητικότητα. Ωστόσο, η αρχιτεκτονική NEPHELE δεν είναι ακόμα επεκτάσιμη, όπως περιγράφεται στη συνέχεια (τρίτο εμπόδιο).

Το δεύτερο εμπόδιο για τη χρήση εξ' ολοκλήρου οπτικών ΔΚΔ πηγάζει από τον υπολογισμό των schedules. Η ανάθεση οπτικών πόρων σε χώρο (σύνδεσμοι), χρόνο (χρονοθυρίδες) και/ή μήκος κύματος απαιτεί υψηλή υπολογιστική πολυπλοκότητα, καθιστώντας πρόκληση τη βέλτιστη, ή ακόμα και τη μη-βέλτιστη εκτέλεσή της σε πραγματικά χρόνο. Στην αρχιτεκτονική NEPHELE παρουσιάστηκαν αλγόριθμοι ανάθεσης πόρων με χαμηλή πολυπλοκότητα για αργά μεταβαλλόμενα μοτίβα κίνησης που εκμεταλλεύονται τα προηγούμενα υπολογισμένα schedules. Ωστόσο, ο γρήγορος χρονοπρογραμματισμός αν και επιλύει ένα σημαντικό μέρος του προβλήματος, δεν επιλύει ολόκληρο το πρόβλημα, όπως περιγράφεται στη συνέχεια.

Το τρίτο εμπόδιο αφορά την υπόθεση του πλήρως κεντρικοποιημένου ελέγχου στα υβριδικά ηλεκτρονικά/οπτικά ΔΚΔ, τα οποία συνήθως ακολουθούν το πρότυπο του SDN. Ο κεντρικός ελεγκτής/χρονοπρογραμματιστής συγκεντρώνει όλες τις απαιτήσεις κίνησης και ρυθμίζει αντίστοιχα τους οπτικούς μεταγωγείς, αλλά, όταν το δίκτυο είναι μεγάλο και η λειτουργία κλειστού βρόγχου εφαρμόζεται για όλους τους κόμβους του δικτύου, αυτή η δομή είναι αναποτελεσματική λόγω της υψηλής καθυστέρησης που προκαλείται από τον έλεγχο (monitoring), τον υπολογισμό των schedules, και τη διάδοση των schedules. Ως αποτέλεσμα, μια αποκλειστικά κεντρικοποιημένη προσέγγιση αντιμετωπίζει περιορισμούς ως προς την επεκτασιμότητα και τη λειτουργία σε πραγματικό χρόνο.

Ερευνητές πρότειναν τον σχεδιασμό ενός μεταγωγέα που χρησιμοποιεί ως θεμελιώδες στοιχείο μεταγωγής μια μονολιθική μονάδας πολλαπλής μεταγωγής (monolithic gang-switched module), τη «μονάδα επιλογέα» (selector module). Με αυτόν ως βάση, παρουσιάστηκε ο σχεδιασμός μιας πλήρους αρχιτεκτονικής ΔΚΔ που ονομάζεται «RotorNet», η οποία χρησιμοποιεί μεταγωγείς που αποτελούνται αποκλειστικά από μονάδες επιλογέων, γνωστούς ως «μεταγωγείς Rotor». Ωστόσο, το RotorNet πετυχαίνει χαμηλές επιδόσεις όσον αφορά τη ρυθμαπόδοση, καθώς δαπανά τη μισή χωρητικότητα του δικτύου για λόγους εξισορρόπησης του φόρτου.

Στο Κεφάλαιο 3, εξετάζουμε τρόπους αντιμετώπισης των περιορισμών των υπαρχουσών αρχιτεκτονικών οπτικών ΔΚΔ. Η προσέγγιση που ακολουθούμε είναι η σχεδίαση προσαρμοσμένων οπτικών μεταγωγέων «Lean», οι οποίοι αποτελούνται από δύο στάδια με πολλαπλές μονάδες επιλογέων στο καθένα και συνδυάζονται με ένα σύνολο μεταγωγέων Rotor για να επιτευχθεί πλήρης συνδεσιμότητα στο δίκτυο. Κάθε μονάδα επιλογέα του πρώτου σταδίου συνδέεται με όλες τις μονάδες επιλογέων του δεύτερου σταδίου, ενώ όλες οι μονάδες επιλογέων μπορούν να μεταφέρουν μια ομάδα από πολλαπλά οπτικά σήματα από διαφορετικές εισόδους προς τις αντίστοιχες εξόδους τους σε κάθε κατάσταση μεταγωγής. Αυτό σημαίνει ότι λίγες μονάδες επιλογέων μπορούν να μεταφέρουν ένα πολύ μεγαλύτερο αριθμό οπτικών σημάτων μεταξύ των κόμβων του ΔΚΔ, μειώνοντας τον αριθμό των απαιτούμενων στοιχείων μεταγωγής σε σχέση με ένα οπτικό δίκτυο με πλήρη ρυθμισιμότητα.

Επιπλέον, σε σύγκριση με ένα οπτικό δίκτυο με πλήρη ρυθμισιμότητα, η προσέγγισή μας μειώνει τον βαθμό του κεντρικού ελέγχου, επιτρέποντας την ανάπτυξη μη-βέλτιστων, αλλά σχεδόν-βέλτιστων αλγορίθμων για την ανάθεση πόρων σε πραγματικό χρόνο. Ωστόσο, η ρυθμισιμότητα (configurability) του ΔΚΔ καθορίζεται από τον αριθμό των μεταγωγέων και των καταστάσεων μεταγωγής των εσωτερικών μονάδων επιλογέων του Lean μεταγωγέα και είναι χαμηλότερη σε σύγκριση με ένα δίκτυο με μεταγωγείς crossbar αντίστοιχου μεγέθους. Αυτοί οι παράμετροι σχεδιασμού επηρεάζουν επίσης την ταχύτητα

επαναρρύθμισης των μεταγωγέων, την αλγοριθμική πολυπλοκότητα για τον υπολογισμό των schedules και την πολυπλοκότητα των εντολών ελέγχου. Ο σχεδιασμός του ΔΚΔ που προτείνεται είναι παραμετρικός, όσον αφορά τον αριθμό των ομαδοποιημένων σημάτων που μεταφέρονται από την ίδια μονάδα επιλογέα, επιτρέποντας την αύξηση της ρυθμισιμότητας του δικτύου με την προσθήκη περισσότερων Lean και Rotor μεταγωγέων. Αυτό μπορεί να οδηγήσει σε αύξηση του αριθμού των θυρών, με αποτέλεσμα την αύξηση της διαθέσιμης χωρητικότητας του δικτύου.

Τέλος, εξετάσαμε την επέκταση του ΔΚΔ RotorNet, μέσω της εφαρμογή της τεχνικής breakout για τη μείωση της καθυστέρησης που οφείλεται στην αύξηση του μεγέθους του δικτύου, καθώς και την υιοθέτηση της μερικής ρυθμισιμότητας, με τη βοήθεια ενός κεντρικού επιπέδου ελέγχου. Σε αυτό το πλαίσιο, εισάγαμε μια πολιτική Προσαρμοστικού Weighted Round-Robin (Adaptive Weighted Round-Robin – AWRR), ειδικά σχεδιασμένη για να ξεπερνά τις επιδόσεις της πολιτικής VLB. Η ισχύς του AWRR βρίσκεται στη δυνατότητά του να προσαρμόζεται στα χαρακτηριστικά της κίνησης χωρίς να έχει προηγούμενη γνώση αυτών. Μαθαίνει σταδιακά τα μοτίβα κίνησης και δυναμικά καθορίζει τις χρονοθυρίδες βασιζόμενος σε εξελισσόμενους συντελεστές βάρους. Διενεργήσαμε μια περιεκτική εξέταση διάφορων πολιτικών χρονοδρομολόγησης, συμπεριλαμβανομένης μιας λεπτομερούς αξιολόγησης του AWRR σε διάφορα προφίλ κίνησης.

Τα αποτελέσματα έδειξαν ότι εφαρμόζοντας την τεχνική breakout, και όσο αυξάνεται ο παράγοντας του (breakout factor), εξασφαλίζεται τη σταθερότητα της ρυθμαπόδοσης, ενώ η καθυστέρηση εμφανίζει γραμμική μείωση, αποτρέποντας αποτελεσματικά τα προβλήματα καθυστέρησης που σχετίζονται με την αύξηση του μεγέθους του δικτύου. Σε συγκεκριμένα σενάρια κίνησης, όπως το προφίλ `WTraffic`, ο AWRR επιδεικνύει εξαιρετική απόδοση. Ξεπερνά τον VLB προσφέροντας σημαντική αύξηση της ρυθμαπόδοσης κατά περίπου 30%, χωρίς καμία προηγούμενη γνώση του προφίλ κίνησης. Επιπλέον, ο AWRR επιδεικνύει εξαιρετική προσαρμοστικότητά στο να χειρίζεται αλλαγές στα προφίλ κίνησης. Μπορεί να προσαρμοστεί γρήγορα σε μεταβαλλόμενες συνθήκες, επιτυγχάνοντας σύγκλιση σε μια σταθερή κατάσταση σε μόλις 500 μs.

Στο Κεφάλαιο 3 παρουσιάζονται οι προδιαγραφές σχεδιασμού των μεταγωγέων Rotor και Lean, καθώς και μιας προτεινόμενης αρχιτεκτονική ΔΚΔ μερικής ρυθμισιμότητας που τους χρησιμοποιεί, του ΔΚΔ Lean. Περιγράφεται το επίπεδο ελέγχου και ο κύκλος ελέγχου του, δίνονται ο ορισμός του προβλήματος και οι πολιτικές χρονοπρογραμματισμού που εκμεταλλεύονται την περιορισμένη ρυθμισιμότητα της αρχιτεκτονικής, οι οποίες έχουν χαμηλή αλγοριθμική πολυπλοκότητα.

Συγκρίνοντας το Lean δίκτυο με ένα δίκτυο RotorNet που χρησιμοποιεί μεταγωγείς Rotor με ελάχιστη ρυθμισιμότητα και ένα οπτικό δίκτυο Mordia που χρησιμοποιεί οπτι-

κούς δακτυλίους WDM και μεταγωγείς WSS με πλήρη ρυθμισιμότητα, το Lean δίκτυο επιτυγχάνει μέτρια crosspoint πολυπλοκότητα, επιτρέποντας μεγάλο αριθμό θυρών στους μεταγωγείς και υψηλή ταχύτητα αναδιαμόρφωσης, ενώ διατηρεί μειωμένη πολυπλοκότητα χρόνου στον χρονοπρογραμματισμό. Η προτεινόμενη πολιτική προγραμματισμού για το δίκτυο Lean, η Lean Valiant Decomposition (LVD) συνδυάζει τις τεχνικές της απαλοιφή (decomposition) Birkhoff-von Neumann και της Εξισορρόπησης Φόρτου Valiant (Valiant Load Balancing), και επιτρέπει είτε άμεσες, είτε έμμεσες μεταδόσεις, ανάλογα με τα μοτίβα της κίνησης, διασφαλίζοντας ότι το δίκτυο επιτυγχάνει ρυθμαπόδοση που κυμαίνεται μεταξύ των περιπτώσεων του δικτύου RotorNet και του δικτύου Mordia.

Οι προσομοιώσεις που παρουσιάζονται περιλαμβάνουν σύγκριση της μέγιστης ρυθμαπόδοσης και των μέσων καθυστερήσεων πακέτου σε τρία σενάρια:

- Στο πρώτο σενάριο, αξιολογούνται τα διάφορα ΔΚΔ και οι πολιτικές χρονοπρογραμματισμού τους και διαπιστώνεται ότι το Lean δίκτυο ξεπερνά το RotorNet κατά 26,8% ως προς τη ρυθμαπόδοση, ενώ πετυχαίνει το 67,3% της ρυθμαπόδοσης του δικτύου Mordia. Παρατηρείται επίσης ότι οι πολιτικές άμεσης μετάδοσης οδηγούν σε χαμηλότερες μέσες καθυστερήσεις πακέτου, αλλά το Lean δίκτυο με την πολιτική LVD πετυχαίνει τουλάχιστον 28% χαμηλότερες καθυστερήσεις από το RotorNet.

- Στο δεύτερο σενάριο, εξετάζονται η επίδοση του Lean δικτύου με διάφορα επίπεδα ομοιομορφίας στο μοτίβο κίνησης χρησιμοποιώντας την πολιτική LVD. Παρατηρείται η σταθερή μείωση της μέσης καθυστέρησης πακέτου κατά 21% όσο αυξάνεται το ποσοστό της κίνησης που ακολουθεί δομημένο μοτίβο.

- Τέλος, στο τρίτο σενάριο, μελετώνται τα δομημένα μοτίβα κίνησης `WTraffic`, `BlkDiag`, και `FFT` στο Lean δίκτυο, χρησιμοποιώντας την πολιτική LVD. Γίνεται η διαπίστωση ότι το Lean δίκτυο πετυχαίνει παρόμοια ρυθμαπόδοση και μέση καθυστέρηση πακέτου με τα μοτίβα `WTraffic` και `BlkDiag`. Ωστόσο, στο μοτίβο `FFT`, το Lean δίκτυο πέτυχε 5% λιγότερη ρυθμαπόδοση από τις άλλες περιπτώσεις και τουλάχιστον 30% αύξηση της μέσης καθυστέρησης πακέτου λόγω της υποχρησιμοποίησης των άμεσων μεταδόσεων με τα δομημένα μοτίβα κίνησης του `FFT`.

Ενορχήστρωση Ασφαλούς Κατανεμημένης Αποθήκευσης σε Cloud-Edge Υποδομές

Ο ψηφιακός μετασχηματισμός έχει επηρεάσει σημαντικά τις απαιτήσεις αποθήκευσης, οι οποίες αναμένεται να αυξηθούν ακόμα περισσότερο στο μέλλον, σύμφωνα με τη Διεθνή

Ένωση Δεδομένων (International Data Corporation – IDC). Καθώς το cloud computing αποτελεί τον θεμέλιο λίθο της ψηφιακής μας κοινωνίας, οι επιχειρήσεις προτιμούν να αποθηκεύουν τα δεδομένα τους στο cloud αντί να εγκαθιστούν τη δική τους υποδομή.Υπάρχουν πολλοί λόγοι για τους οποίους μια επιχείρηση προτιμά να αποθηκεύει τα δεδομένα της στο cloud αντί να τα διατηρεί σε ιδιωτικά μέσα αποθήκευσης. Τα πλεονεκτήματα που προσφέρει περιλαμβάνουν την αποφυγή υψηλών αρχικών κεφαλαιακών δαπανών (CAPEX), την επεκτασιμότητα της παρεχόμενης υπηρεσίας αποθήκευσης και την εύκολη μεταφορά των δεδομένων όταν αυτό απαιτείται. Επιπλέον, μια υπηρεσία αποθήκευσης βασισμένη στο cloud παρέχει υψηλή διαθεσιμότητα, απαλλάσσοντας μια επιχείρηση από την ανάγκη να αναπτύξει περίπλοκους και δαπανηρούς μηχανισμούς για τον πλεονασμό των δεδομένων (redundancy) και την ανοχή σε σφάλματα (fault-tolerance) σε περιπτώσεις διακοπών τροφοδοσίας ή σε σενάρια καταστροφής.

Αντίθετα, τα κατανεμημένα συστήματα αποθήκευσης διατηρούν τα δεδομένα σε πολλές τοποθεσίες και ενοποιούν πόρους από πολλούς παρόχους που, εάν επιλεγούν προσεκτικά, μπορούν να προσφέρουν αυξημένη ευελιξία σε σχέση με μια μεμονωμένη υπηρεσία αποθήκευσης. Με την εμφάνιση του edge computing, η αποθήκευση και η επεξεργασία των δεδομένων κοντά στην πηγή τους (π.χ., κάμερα ή άλλο αισθητήρα) έγινε πραγματικότητα. Η ενσωμάτωση πόρων edge σε κατανεμημένες υπηρεσίες αποθήκευσης βελτιώνει τον τρόπο με τον οποίο εξυπηρετούνται απαιτητικές εφαρμογές: τα δεδομένα αποθηκεύονται και επεξεργάζονται στο πόρους edge για να ελαχιστοποιηθεί η καθυστέρηση και η χρήση του δικτύου, και, εάν απαιτούνται επιπλέον πόροι, χρησιμοποιούνται οι άφθονοι πόροι του cloud. Η συνεχής αύξηση στον αριθμό και την πυκνότητα των πόρων edge αναμένεται να αλλάξει τον τρόπο με τον οποίο λειτουργούν οι τρέχουσες υπηρεσίες αποθήκευσης. Ωστόσο, αυτό αυξάνει και την πολυπλοκότητα, καθώς οι πόροι edge έχουν μεγάλη ποικιλομορφία χαρακτηριστικών, η διαθεσιμότητά τους ποικίλλει με τον χρόνο και είναι πιο αναξιόπιστοι σε σχέση με το cloud.

Η τοποθέτηση θραυσμάτων (fragments) δεδομένων σε απομακρυσμένες τοποθεσίες, όπου μπορεί να συμβούν διαρροές δεδομένων, δημιουργεί ανησυχίες για το απόρρητο και την ασφάλεια αυτών των συστημάτων. Δεδομένου ότι οι πάροχοι αποθήκευσης εν γένει δεν μπορούν να θεωρούνται αξιόπιστοι, υπάρχει η πιθανότητα να ανακτηθούν ευαίσθητα δεδομένα από τα κρυπτογραφημένα θραύσματα. Δεδομένου ότι οι αποτυχίες είναι συνηθισμένες στα κατανεμημένα συστήματα, πρέπει να αποθηκεύονται πλεονάζοντα (redundant) δεδομένα για να αντέχουν στις αποτυχίες χωρίς να υπάρχει απώλεια δεδομένων. Η κωδικοποίηση απαλοιφής (erasure coding) χρησιμοποιεί κώδικες Forward Error Correction (FEC) για να διασφαλίσει την ακεραιότητα και τη μακροβιότητα των δεδομένων. Τα δεδομένα διασπώνται (split), κωδικοποιούνται (encoded) και επιδέχονται κά-

ποια επιβάρυνση (overhead), ανάλογα με τον χρησιμοποιούμενο αλγόριθμο. Ακόμη και όταν δεν είναι δυνατή η ανάκτηση ορισμένων θραυσμάτων, ανάλογα με την τεχνική κωδικοποίησης που χρησιμοποιείται μπορεί να μην απαιτούνται όλα τα θραύσματα για να ανακτηθούν αποτελεσματικά τα αρχικά δεδομένα. Η κωδικοποίηση απαλοιφής παρέχει επιπρόσθετη ασφάλεια σε σχέση με την περίπτωση που γίνεται χρήση μόνο της τεχνικής της κρυπτογράφηση, καθώς απαιτείται συγκεκριμένος αριθμό θραυσμάτων από διαφορετικές τοποθεσίες ούτως ώστε να αποκωδικοποιηθούν από κοινού για να ανακτηθούν τα αρχικά δεδομένα. Η λειτουργία μιας κατανεμημένης υπηρεσίας αποθήκευσης που ενσωματώνει ταυτόχρονα πόρους edge και cloud ενώ χρησιμοποιεί την κωδικοποίηση απαλοιφής για τη διάσπαση των δεδομένων, παρουσιάζει μια σημαντική πρόκληση για τον αντίστοιχο μηχανισμό ενορχήστρωσης πόρων. Εκτός από τον καθορισμό της ποσότητας και της κατανομής των θραυσμάτων των δεδομένων και των θραυσμάτων ισοτιμίας (parity), ο ενορχηστρωτής πρέπει επίσης να ικανοποιεί τις απαιτήσεις των χρηστών και να βελτιστοποιεί τα διάφορα κριτήρια, συμπεριλαμβανομένου του οικονομικού κόστους, της καθυστέρησης και της διαθεσιμότητας.

Στο Κεφάλαιο 4, διατυπώνουμε την ενορχήστρωση πόρων αποθήκευσης ως ένα πρόβλημα Μεικτού Ακέραιου Γραμμικού Προγραμματισμού (Mixed-Integer Linear Programming – MILP) για τον υπολογισμό της βέλτιστης λύσης. Ωστόσο, ο χώρος αναζήτησης μπορεί να είναι τεράστιος, οδηγώντας σε απαγορευτικά μεγάλο χρόνο εκτέλεσης για το MILP, ειδικά όταν χειρίζεται πολλά αιτήματα αποθήκευσης με αυστηρές και ετερογενείς απαιτήσεις. Ο χρόνος εκτέλεσης αναφέρεται στη διάρκεια που απαιτείται από τον μηχανισμό ενορχήστρωσης για να επεξεργαστεί αιτήματα αποθήκευσης και να υπολογίσει ένα schedule ανάθεσης πόρων για το δεδομένο σενάριο. Προτείνουμε μια αποδοτική ευρετική μέθοδο multi-agent rollout που βασίζεται στην ενισχυτική μάθηση (reinforcement learning), η οποία ανταλλάζει απόδοση για χρόνο εκτέλεσης. Αυτό επιτρέπει τη γρήγορη λήψη αποφάσεων σε πραγματικά σενάρια, μειώνοντας τον μέσο χρόνο εκτέλεσης σε σύγκριση με αυτόν του βέλτιστου MILP, διατηρώντας ταυτόχρονα την απόδοση κοντά στη βέλτιστη, όπως αποδεικνύεται στα πειράματα για τα οποία καταφέραμε να υπολογίσουμε τη βέλτιστη λύση. Οι μηχανισμοί που αναπτύχθηκαν χρησιμοποιούν πολλαπλά κριτήρια βελτιστοποίησης, όπως κόστος, χωρητικότητα, αξιοπιστία, απόδοση, διαθεσιμότητα ή συνδυασμό αυτών, ενώ αποφασίζεται ο κατακερματισμό (fragmentation) των δεδομένων, η κρυπτογράφηση και η τοποθέτηση των δεδομένων. Οι μηχανισμοί λαμβάνουν επίσης υπόψη τα διάφορα χαρακτηριστικά των πόρων edge και cloud όσον αφορά την καθυστέρηση, τη διαθεσιμότητα, την ασφάλεια και το οικονομικό κόστος.

Για να αναδειχθεί η αποτελεσματικότητα των προτεινόμενων μηχανισμών, διεξήχθησαν μια σειρά πειραμάτων προσομοίωσης χρησιμοποιώντας ανώνυμα δεδομένα από την

Chocolate Cloud. Η Chocolate Cloud ειδικεύεται στο λογισμικό ασφαλούς και κατανεμημένης αποθήκευσης δεδομένων, και το πιο εμβληματικό της προϊόν, το SkyFlok, είναι ένα Λογισμικό-ως-Υπηρεσία (Software-as-a-Service – SaaS) συστήματος αποθήκευσης και διαμοιρασμού αρχείων. Οι προτεινόμενοι μηχανισμοί ενισχύουν λογικά την ενορχήστρωση της πλατφόρμας SkyFlok, επιτρέποντάς της να ενσωματώσει αποτελεσματικά πόρους edge. Αυτό σημαίνει ότι οι μηχανισμοί μπορούν να ενσωματωθούν στους back-end μηχανισμούς ελέγχου και ενορχήστρωσης της υπηρεσίας, επιτρέποντάς της να συντονίσει την κρυπτογράφηση, την κωδικοποίηση απαλοιφής και την κατανομή των θραυσμάτων των δεδομένων στις επιλεγμένες τοποθεσίες cloud και edge.

Οι προτεινόμενοι μηχανισμοί μας επιτυγχάνουν απόδοση κοντά στη βέλτιστη σε εκτεταμένες προσομοιώσεις χρησιμοποιώντας τόσο συνθετικά όσο και πραγματικά δεδομένα που κυμαίνονται στο εύρος 94,8-97,5%. Στις προσομοιώσεις ανάκτησης δεδομένων, εξετάσαμε τόσο στη λειτουργία αποθήκευσης (store), όσο και στη λειτουργία ανάκτησης (retrieve), την επίδραση διαφόρων κριτηρίων βελτιστοποίησης στα χρηματικά κόστη, τις καθυστερήσεις, τη διαθεσιμότητα και το ποσοστό επιτυχούς ανάκτησης αρχείων. Σε σύγκριση με τις μονοκριτηριακές προσομοιώσεις, οι προτεινόμενοι πολυ-κριτηριακοί μηχανισμοί βελτιστοποίησης ανταποκρίνονται αποτελεσματικά στις διάφορες αντιφατικές απαιτήσεις βελτιστοποιώντας την τοποθέτηση των αρχείων, επιτυγχάνοντας βέλτιστο ή σχεδόν βέλτιστο χρηματικό κόστος, καθυστέρηση και διαθεσιμότητα, όλα με μικρό χρόνο εκτέλεσης μέσω του μηχανισμού multi-agent rollout, ο οποίος βελτιώνει σημαντικά την απόδοση του άπληστου ευρετικού αλγόριθμου (2%-57%). Επιπλέον, η μελέτη μας υπογραμμίζει τη σημασία της ενσωμάτωσης των κόμβων edge στην αντιμετώπιση των αυστηρών απαιτήσεων των εφαρμογών σχετικά με την καθυστέρηση (ο χρόνος καθυστέρησης αποθήκευσης και ανάκτησης βελτιώθηκε κατά 22%). Ο συντοπισμός (colocation) των κόμβων edge οδηγεί στην περαιτέρω μείωση της εμπειρικής καθυστέρησης (experienced latency) (γραμμικά ως προς την αύξηση του συντοπισμού), αναδεικνύοντας την αξία του να λαμβάνεται υπόψη ο συντοπισμός στην ανάπτυξη μεικτών συστημάτων αποθήκευσης edge-cloud.

Τελικά, η παρούσα διδακτορική διατριβή παρέχει σημαντικές γνώσεις για τον σχεδιασμό αποτελεσματικών και αξιόπιστων συστημάτων αποθήκευσης που εκμεταλλεύονται τα πλεονεκτήματα τόσο των πόρων edge όσο και των πόρων cloud, συμβάλλοντας στην ανάπτυξη ανθεκτικών κατανεμημένων υποδομών αποθήκευσης που ανταποκρίνονται αποτελεσματικά στις αυξανόμενες απαιτήσεις της ψηφιακής εποχής.

Chapter 1

Introduction

1.1 Resource Allocation in an Optical Datacenter Interconnect with Fiber Rings and Wavelength Selective Switches

The widespread availability of cloud applications to billions of end users and the emergence of platform- and infrastructure-as-a-service models rely on concentrated computing infrastructures, the data centers (DCs). DCs typically comprise numerous interconnected servers running virtual machines. As traffic within the DC (east-west) is higher than incoming/outgoing traffic, and both are expected to continue to increase (59), DC networks (DCNs) play a crucial role. High throughput, scalable, and energy/cost-efficient DCN networks are required to fully harness DC potential.

State-of-the-art DCNs are based on electronic switching in Fat-Tree topologies (8). Fat-trees tend to underutilize resources, require numerous cables, and suffer from poor scalability and low energy efficiency (14, 65). To reduce cost, Fat-Trees are typically oversubscribed (e.g., 1:4), and do not offer full bisection bandwidth (FBB) that may be needed for certain applications. Application-driven networking (31, 41), an emerging trend, would benefit from a network that flexibly allocates capacity where needed. To cope with the shortcomings of Fat-Trees, many recent works proposed hybrid electrical/optical DCN, a survey of which is presented in (42). The authors of (30, 77) proposed a DCN in which heavy long-lived (elephant) flows are selectively routed over an optical circuit switched (OCS) network, while the rest of traffic goes through the electronic packet switched (EPS) network. These solutions rely on the identification of elephant flows, which is rather difficult, while it was observed that such long-lived heavy flows are not typical (65), making it difficult to sustain high OCS utilization. Instead, a high connectivity degree is needed (65). To enable higher connectivity, (24) proposed and prototyped a very dense hybrid DCN that also supports multi-

hop connections, along with a custom-built control stack. The authors measured the total delay, including control plane and OCS hardware reconfiguration (microelectromechanical system-MEMS-switches), to be of the order of hundreds of milliseconds. Multi-hop routing was exploited anew as shared optical circuits in (13), where an OpenFlow (OF)-based control plane was developed (54), showing that circuit sharing reduces the effect of slow OCS reconfigurations.

Other proposed DC interconnects completely lack electrical switches. Proteus, an all-optical DCN architecture based on a combination of wavelength selective switches (WSSs) and MEMS was presented in (72). Again, multi-hop is used to achieve high utilization. However, it is still hard to compensate the MEMS slow reconfiguration times through sophisticated control. References (62, 67) introduced hybrid OCS and optical packet/burst switching (OPS/OBS) architectures, controlled using SDN. Various other architectures based on OPS/OBS were proposed (20, 42) (and references therein). However, OPS/OBS technologies are not yet mature, so the current target could be small-scale networks with limited upgradability potential.

The authors of (63) presented a hybrid DCN architecture called Mordia, which uses WSS to provide switching times of 11.5 μs. Mordia operates in a dynamic slotted manner to achieve high connectivity. However, the scalability of Mordia is limited as it uses a single wavelength division multiplexing (WDM) ring whose capacity can accommodate only a few racks, while resource allocation algorithms exhibit high complexity and cannot scale to large DCs.

The European project NEPHELE is developing an optical DCN that leverages hybrid electrical/optical switching with SDN control to overcome current datacenter limitations (3). To enable dynamic and efficient sharing of optical resources and collision-free communication, NEPHELE operates in a synchronous slotted manner. Timeslots are used for rack-to-rack communication and are assigned dynamically, on a demand basis, to attain efficient utilization, leading to both energy and cost savings. Moreover, multiple wavelengths and optical planes are utilized to implement a scalable and high capacity DC network infrastructure.

The NEPHELE network relies on WSSs, which are faster than the MEMS used in (13, 24, 30, 77) and more mature than the OPS/OBS used in (20, 62, 67). The fast switching times, along with the dynamic slotted operation, provide high and flexible connectivity. Compared to Mordia (63), which also relies on WSSs, NEPHELE is more scalable: it consists of multiple WDM rings, re-uses wavelengths, and utilizes cheap passive routing components and scalable scheduling schemes.

Regarding resource allocation, scheduling problems similar to those addressed in Chapter 2 were studied in the past for satellite and ATM switches (9, 18, 19, 29, 35, 40, 68, 84). Indeed, one can view the entire multi-ring DCN as a large distributed switch. The key difference of our work is that we consider huge network installations and dynamic time-division multiplexing (TDM) operation; thus strict optimality is not the objective, but we rather target low complexity.

We also encounter certain internal collision constraints that are particular to the reference architecture, and thus we need to extend previous TDM algorithms appropriately. Apart from (63), which considers dynamic TDM operation, a somehow relevant algorithmic work is (21), where the authors present an integrated optical network-on-chip (NoC) based on a ring topology and micro-ring resonators (MRs). The key difference with the NEPHELE network is that MRs target NoC and small networks, where propagation and control plane delays are negligible. Thus, scheduling does not take place in periods, as in NEPHELE, but on a per-slot basis as in electronic switches (53).

In Chapter 2 we begin by introducing the reference optical DCN architecture utilizing fiber rings and WSSs, followed by a description of the dynamic resource allocation problem and the set of algorithms designed to address it effectively. We then delve into an analysis of the resource allocation constraints specific to the reference architecture. Subsequently, we thoroughly evaluate the performance of the proposed algorithms. Furthermore, we explore enhancements in the control plane using traffic estimation and in the architecture with deploying a parallel network, and conduct simulations with realistic traffic scenarios. Finally, our conclusions summarize the key insights and findings from our work.

1.2 Fast Optical Datacenter Interconnects with Partial Configurability

The integration of optical switching in DCNs is a pivotal step towards addressing the limitations of Fat-Tree topologies. While optical switches are primarily used for circuit switching in metro and backbone networks, recent research has proposed hybrid electronic/optical switched DCNs as a solution (12, 20, 24, 30, 42, 62, 63, 67, 72, 76, 77). These studies employ optical switches, which transparently redirect light from any port to another. However, their reconfiguration times (milliseconds for high radix and tens of microseconds for low radix switches) present a challenge to their use in DCNs, where rapid switch configuration is essential. Despite this limitation, the potential benefits of integrating optical switches into DCNs are significant, and ongoing research is focused on developing more efficient

and sustainable solutions. In 2022, Google announced (64) that Jupiter datacenter network fabrics will use dynamic topology reconfiguration using Optical Circuit Switching, which have evolved to achieve higher speed, cost reduction, power efficiency, and optimized path lengths.

The first barrier to the adoption of optical switching technologies in DCNs comes from the reconfiguration speed of (full) crossbar optical switches. As the size of DCNs grows, the options of employing a single-stage optical diminish and/or the reconfiguration speed is prohibitive high. Conversely, using multi-stage crossbars, built with smaller modules, is the only solution. However, this suffers from high overall switch count and wiring complexity, which can in turn affect the production cost. It also requires tight synchronization and coordinated control of the multiple elements. To address these challenges, researchers have proposed two hybrid DCN architectures: Mordia and CBOSS, which utilize WSS which makes use of wavelength domain to reduce the number of required elements and provide low switching times. These proposed solutions operate in a dynamic slotted manner to achieve high connectivity (15, 63). However, the scalability of both Mordia and CBOSS is limited, as they employ a single wavelength division multiplexing (WDM) ring with a capacity that can accommodate only a few racks. In contrast, NEPHELE (12) proposes a distributed crossbar optical network fabric using WSS switches interconnected in several WDM fiber rings. The NEPHELE architecture takes advantage of the use of (relatively) low radix WSS switches, space (multiple rings) and wavelength (WDM) domains to achieve low reconfiguration speed and high throughput. However, the NEPHELE architecture is still not scalable, as discussed below (third barrier).

The second barrier to using all-optical DCNs derives from schedule computation. Allocating optical resources in space (links), time (slots), and/or wavelength (WDM) domains requires high computational complexity, making it challenging to perform optimally or even suboptimally in real-time. The resource allocation algorithms of Mordia (63) and CBOSS (15) exhibit high computational complexity and do not scale well with large DCs, representing a significant challenge for optimizing these networks. NEPHELE (12) introduced resource allocation algorithms with low complexity for slowly changing traffic patterns that take advantage of previously computed schedules. Efforts to address the computational complexity of centralized scheduling calculations have also been explored, such as the parallel scheduler architecture of (61). However, fast scheduling solves a key part but not the whole problem, as discussed next.

The third barrier pertains to the assumption of centralized control in hybrid electronic/optical DCNs, which typically follows the SDN paradigm (12, 24, 67). In this architecture, a central controller/scheduler gathers all traffic demands and configures the optical switches

accordingly. However, when the network is large and the closed-loop operation is applied for all network nodes, it is inefficient due to the high latency induced by the control plane for monitoring, schedule calculation, and schedule dissemination. As a result, this purely centralized approach faces limitations in terms of scalability and real-time operation.

In (57) the authors proposed a switch design that utilizes a monolithic gang-switched module called the "selector module" as its fundamental building block. Building on that, the authors of (56) introduced a full DCN architecture design called "RotorNet", utilizing switches constructed exclusively with selector modules, referred to as "Rotor switches". However, RotorNet achieves poor throughput performance since it spends half its network capacity for load balancing purposes.

In Chapter 3 we explore ways to address the limitations of existing optical data center network (DCN) architectures. The approach we take is to design custom "Lean" optical switches, which have two stages of multiple selector modules and are combined with a set of Rotor switches to achieve full network connectivity. Each selector module of the first stage is connected to all selector modules of the second stage, while all selector modules can carry a group of multiple optical signals from different input ports to their corresponding output ports at each switching state. This means that a few selector modules can carry a much higher number of optical signals between DCN nodes, reducing the number of required switching elements compared to a fully configurable network.

Additionally, compared to a fully configurable network, our solution reduces the level of centralized control, enabling the development of algorithms to allocate resources sub- but near-optimally in real-time. However, the configurability of the DCN is determined by the number of switches and the switching states of the Lean switch internal selector modules, and it is lower compared to a network with crossbar switches of similar size. These design parameters also affect the reconfiguration speed of the switches, the algorithmic complexity for the computation of schedules, and the complexity of the control commands. The proposed DCN design is parametric with respect to the number of the grouped signals carried by the same selector modules, allowing for an increase in the network's configurability by adding more Lean and Rotor switches. This may lead to an increase in the number of ports, which in turn increases the available network capacity. WDM can be utilized with the optical signals to further enhance the network's capacity.

In Chapter 3, we present the design specifications of the Rotor and Lean switches, and our proposed partially configurable DCN architecture that uses them. Furthermore, we discuss the control plane and its control cycle, and we present the problem definition and scheduling policies that take advantage of the limited configurability of the architecture, exhibiting low computational complexity. Additionally, we present some reference architectures, their cor-

responding scheduling policies, and compare them in terms of crosspoint complexity. We evaluate the achieved throughput and average packet latency of the proposed DCN under various scenarios using the packet simulator OMNET++.

Finally, we examine enhancements on RotorNet by applying breakout for mitigating latency due to network size expansion and integrating centralized control for partial configurability. We develop a policy that adapts to traffic characteristics without prior knowledge, designed to surpass VLB. We compare it to various other scheduling policies and evaluates it across diverse traffic profiles, though comprehensive simulations.

1.3 Secure Distributed Storage Orchestration on Cloud-Edge Infrastructures

Digital transformation has had a significant impact on the storage requirements, which are expected to further increase in the foreseeable future, according to the International Data Corporation (IDC) (37). As cloud computing is the cornerstone of our digital society, businesses prefer to store their data in the cloud rather than deploying their own infrastructure, thereby exploiting the offered scalability and increased availability. In this manner, businesses are alleviated from the burden of deploying complex and costly data redundancy and fault-tolerance mechanisms.

There are many reasons for a business to prefer storing its data on the cloud instead of privately held storage devices. The advantages obtained in this way include the avoidance of high initial capital expenditure (CAPEX), the scalability of the storage service provided, and the easy migration of the data when needed. Also, a cloud-based storage service provides high availability, exempting a business from the necessity to deploy complex and costly mechanisms for data redundancy and fault-tolerance to power outages and other disaster scenarios.

Distributed storage systems, on the other hand, store data in multiple locations, consolidating resources from multiple providers that, if selected carefully, can offer increased flexibility compared to a single storage service (33). With the advent of edge computing, the storage and processing of the data close to the generating source (e.g., camera, or other sensor) (48) became a reality. The incorporation of edge resources in distributed storage services improves the way demanding applications are served: data are stored and processed at the edge to minimize latency and network usage, and, if additional resources are required, the abundant cloud resources are utilized. The continuous increase in the number and density of edge resources is expected to change the way current storage services operate. However,

this also increases the complexity as the edge resources exhibit diverse characteristics, their availability varies with time, and they are more unreliable (28).

Laying data fragments to remote storage locations where data leaks can happen raises privacy and security concerns for such systems. As storage providers cannot generally be considered trustworthy, sensitive data can be retrieved from encrypted fragments. Since failures are common in distributed systems, redundant data must be stored to tolerate failures without data loss. Erasure coding (70) uses Forward Error Correction (FEC) codes to ensure data integrity and longevity. Data are split, encoded, and incur an overhead depending on the algorithm. Even when some fragments cannot be retrieved, the original data can be efficiently recovered by fetching a subset of fragments, depending on the encoding technique. Erasure coding provides additional security atop encryption, as data can only be recovered when a specific number of fragments from various locations are jointly decoded.The operation of a distributed storage service that integrates edge and cloud resources while utilizing erasure coding to divide data presents a formidable challenge for the corresponding resource orchestration mechanism. In addition to determining the quantity and distribution of data and parity fragments, the orchestrator must also fulfill user demands and optimize various criteria, including monetary cost, latency, and availability.

We formulate the storage resource orchestration as a Mixed-Integer Linear Programming (MILP) problem to obtain the optimal solution. However, the search space can be vast leading to a prohibitively large execution time for the MILP, especially when handling numerous storage requests with strict and heterogeneous requirements. Execution time refers to the duration required for the orchestration mechanism to process storage demands and generate a resource allocation plan for the given scenario. In this work, we propose an efficient multi-agent rollout heuristic approach that is based on reinforcement learning, which balances performance and execution time. This enables fast decision-making in real-world scenarios, reducing the average execution time over that of the optimal MILP algorithm, while maintaining near-optimal performance, as demonstrated in the experiments for which we were able to track the optimal solution. The developed mechanisms use a multitude of optimization criteria, namely, cost, capacity, reliability, performance, availability, or a combination of them when deciding on the data fragmentation, encryption and data placement. The mechanisms also account for the different characteristics of the edge and cloud resources with respect to latency, data availability, security and monetary cost.

To demonstrate the effectiveness of the proposed mechanisms, a series of simulation experiments were performed using anonymous data from Chocolate Cloud. Chocolate Cloud is specialized in secure and distributed data storage software and its flagship product, SkyFlok (sky), is a Software-as-a-Service file sharing and storage solution. The proposed mecha-

nisms enhance the orchestration logic of the SkyFlok platform, allowing it to incorporate edge resources efficiently. This means that the mechanisms can be integrated into the back-end control and orchestration mechanisms of the distributed service, enabling it to coordinate the encryption, erasure coding and distribution of data fragments across the selected cloud and edge location.

The storage allocation problem has long attracted the interest of many researchers. To address the limitations of single cloud models, multi-cloud resource allocation schemes were initially examined (33, 60) and (49). In (33), Hadji proposed a solution based on commodity flows to minimize the storage monetary cost and latency. Papaioannou et al. (60) proposed Scalia, a cloud storage brokerage solution for data placement that targets to minimize the storage monetary cost. Mansouri et al. (52) proposed an algorithm that minimizes the storage monetary cost, guaranteeing at the same time high data availability and privacy.

Ma et al. (51) proposed a mixed policy that is based on a combination of erasure and replication coding techniques, targeting to minimize latency, as well as storage and network monetary costs. In the same context, Zhang et al. (86) proposed a sub-optimal multi-agent heuristic approach for selecting the storage locations and the appropriate redundancy configuration to minimize the monetary cost with respect to the user's latency and availability requirements. Wu et al. (83) proposed a scheme that trades-off cost for latency, meeting the preset availability requirements. Liu et al. (49) proposed a heuristic (genetic) algorithm to minimize costs while providing Service Level guarantees.

Targeting the experienced latency minimization, Sharov et al. (69) proposed a quorum-based configuration that makes use of replication coding and assigns the fragments to the different locations. Bermbach et al. (16) examined the consistency versus latency trade-off making use of a mechanism from Amazon's Dynamo for replication to multiple cloud providers. Other works, such as (17, 33, 52, 81), rely on replication to multiple providers in order to attain higher availability and avoid vendor lock-ins, while keeping the cost low. However, the use of replication instead of erasure coding does not address the problem of the variations in latency that are experienced by the user.

Other works have proposed mechanisms that improve data availability through redundancy, also minimizing the monetary cost incurred. However, these works rely on replication coding, which requires more storage space, compared to erasure coding. In (6, 58, 60, 86), the authors proposed mechanisms that make use of erasure coding solutions to improve data availability. In this direction, Wang et al. (78, 80) proposed various techniques that minimize the monetary cost while maximizing the availability. Su et al. (73) proposed an erasure coding model for solving the data placement problem. Wang et al. (79) proposed an adaptive

model for data placement that minimizes the monetary cost but also takes into consideration the latency and availability constraints.

In (26), the authors address cloud plan selection by users, introducing a simple language to express user requirements and plan preferences, and propose a model for identifying and ranking the plans that satisfy the requirements. In (27), the same authors extend this work, allocating resources from multiple cloud services. Users can specify allocation requirements to reduce burden and avoid excessive fragmentation. The authors provide an integer programming formulation for finding an allocation satisfying protection and allocation requirements while minimizing costs. (11) adopts All-Or-Nothing-Transform (AONT) and data replication, introducing two strategies for allocating shards to nodes. The analysis of these allocation strategies illustrates tuning to balance availability and security. In (10), the authors address the dynamic version of the problem, relying on fountain codes instead of replication.

Previous works have focused on cloud storage and optimizing individual objectives, such as data availability, latency, and cost. In contrast, (50) and (7) propose caching schemes with erasure code for low latency in distributed storage systems that span across the edge-cloud continuum. The proposed scheme caches popular data chunks at edge servers to achieve low latencies, but costs and availability of storage resources are not optimized. Authors in (71) propose a location aware to optimize the data retrieval latency in ultra-large distributed storage systems, while the authors in (25) propose a rights Management Protocol to enable the sharing of files in distributed storage systems consisting of nodes that are not fully trusted.

In the current work, we extend the storage resource allocation problem considering the intrinsic characteristics of a distributed storage infrastructure that spans over the edge-cloud continuum. Hence, in addition to cloud resources, we consider edge resources, which are located closer to where the data are generated (82, 85) and have limited storage capacity and highly dynamic availability. When edge and cloud resources are allocated, leveraging the erasure code technique, different optimization criteria can be addressed simultaneously.

In Chapter 4 we discuss on the infrastructure and the distributed storage operations. We provide the resource allocation policies, and we present the simulation results.

Chapter 2

Resource Allocation in an Optical Datacenter Interconnect with Fiber Rings

2.1 Introduction and Related Work

The widespread availability of cloud applications to billions of end users and the emergence of Platform- and Infrastructure-as-a-Service (IaaS and PaaS) models rely on concentrated computing infrastructures, the data centers (DCs). DCs typically comprise numerous interconnected servers running virtual machines. As traffic within the DC (east-west) is higher than incoming/outgoing traffic, and both are expected to continue to increase (59), DC networks (DCNs) play a crucial role. High throughput, scalable, and energy/cost-efficient DCN networks are required to fully harness DC potential.

State-of-the-art DCNs are based on electronic switching in Fat-Tree topologies (8). Fat-trees tend to underutilize resources, require numerous cables, and suffer from poor scalability and low energy efficiency (14, 65). To reduce cost, Fat-Trees are typically oversubscribed (e.g., 1:4), and do not offer full bisection bandwidth (FBB) that may be needed for certain applications. Application-driven networking (31, 41), an emerging trend, would benefit from a network that flexibly allocates capacity where needed. To cope with the shortcomings of Fat-Trees, many recent works proposed hybrid electrical/optical DCN, a survey of which is presented in (42). The authors of (30, 77) proposed a DCN in which heavy long-lived (elephant) flows are selectively routed over an optical circuit switched (OCS) network, while the rest of traffic goes through the electronic packet switched (EPS) network. These solutions rely on the identification of elephant flows, which is rather difficult, while it was observed

that such long-lived heavy flows are not typical (65), making it difficult to sustain high OCS utilization. Instead, a high connectivity degree is needed (65). To enable higher connectivity, (24) proposed and prototyped a very dense hybrid DCN that also supports multi-hop connections, along with a custom-built control stack. The authors measured the total delay, including control plane and OCS hardware reconfiguration (microelectromechanical system-MEMS-switches), to be of the order of hundreds of milliseconds. Multi-hop routing was exploited anew as shared optical circuits in (13), where an OpenFlow (OF)-based control plane was developed (54), showing that circuit sharing reduces the effect of slow OCS reconfigurations.

Other proposed DC interconnects completely lack electrical switches. Proteus, an all-optical DCN architecture based on a combination of wavelength selective switches (WSSs) and MEMS was presented in (72). Again, multi-hop is used to achieve high utilization. However, it is still hard to compensate the MEMS slow reconfiguration times through sophisticated control. References (62, 67) introduced hybrid OCS and optical packet/burst switching (OPS/OBS) architectures, controlled using SDN. Various other architectures based on OPS/OBS were proposed (20, 42) (and references therein). However, OPS/OBS technologies are not yet mature, so the current target could be small-scale networks with limited upgradability potential.

The authors of (63) presented a hybrid DCN architecture called Mordia, which uses WSS to provide switching times of 11.5 μs. Mordia operates in a dynamic slotted manner to achieve high connectivity. However, the scalability of Mordia is limited as it uses a single wavelength division multiplexing (WDM) ring whose capacity can accommodate only a few racks, while resource allocation algorithms exhibit high complexity and cannot scale to large DCs.

The European project NEPHELE is developing an optical DCN that leverages hybrid electrical/optical switching with SDN control to overcome current datacenter limitations (3). To enable dynamic and efficient sharing of optical resources and collision-free communication, NEPHELE operates in a synchronous slotted manner. Timeslots are used for rack-to-rack communication and are assigned dynamically, on a demand basis, to attain efficient utilization, leading to both energy and cost savings. Moreover, multiple wavelengths and optical planes are utilized to implement a scalable and high capacity DC network infrastructure.

The NEPHELE network relies on WSSs, which are faster than the MEMS used in (13, 24, 30, 77) and more mature than the OPS/OBS used in (20, 62, 67). The fast switching times, along with the dynamic slotted operation, provide high and flexible connectivity. Compared to Mordia (63), which also relies on WSSs, NEPHELE is more scalable: it consists of mul-

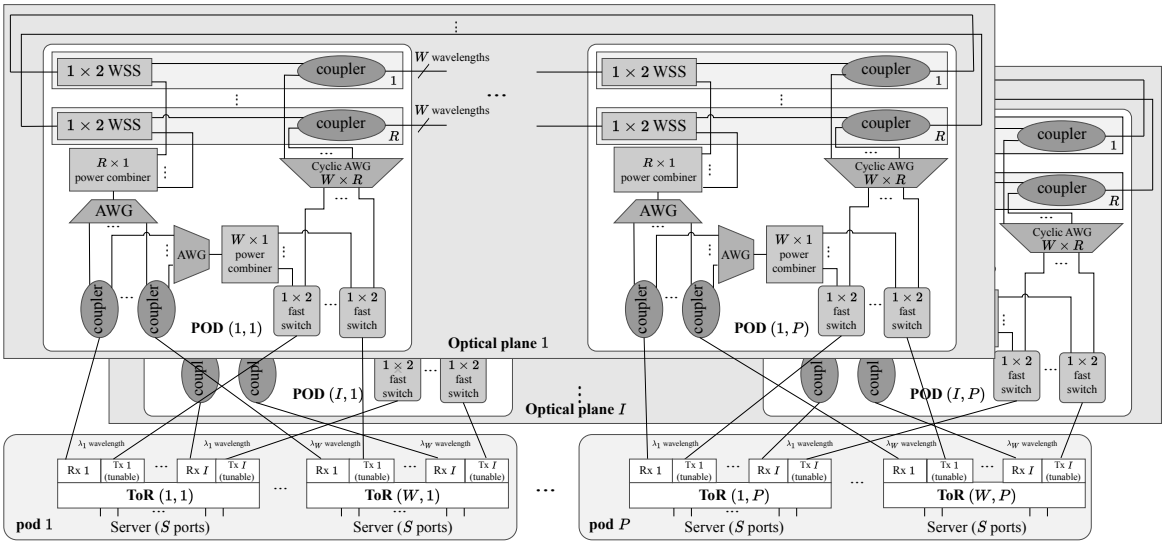
tiple WDM rings, re-uses wavelengths, and utilizes cheap passive routing components and scalable scheduling schemes. The latter is the major contribution of this chapter. We consider NEPHELE as the reference architecture, and we presents fast scheduling algorithms to meet NEPHELE dynamic reconfiguration requirements.

Regarding resource allocation, scheduling problems similar to those addressed in this chapter were studied in the past for satellite and ATM switches (9, 18, 19, 29, 35, 40, 68, 84). Indeed, one can view the entire reference multi-ring DCN as a large distributed switch. The key difference of our work is that we consider huge network installations and dynamic time-division multiplexing (TDM) operation; thus strict optimality is not the objective, but we rather target low complexity.

We also encounter certain internal collision constraints that are particular to the reference architecture (Section 2.6), and thus we need to extend previous TDM algorithms appropriately. Apart from (63), which considers dynamic TDM operation, a somehow relevant algorithmic work is (21), where the authors present an integrated optical network-on-chip (NoC) based on a ring topology and micro-ring resonators (MRs). The key difference with the NEPHELE network is that MRs target NoC and small networks, where propagation and control plane delays are negligible. Thus, scheduling does not take place in periods, as in NEPHELE, but on a per-slot basis as in electronic switches (53).

The research results of this chapter were published in (23), (22) and (44). The remainder of this chapter is organized as follows. In Section 2.2, we introduce the reference optical DCN architecture utilizing fiber rings and WSSs. Section 2.3 is dedicated to the dynamic resource allocation problem, where we describe the challenge and its implications. In Section 2.4, we present a set of algorithms developed to address this problem effectively. Section 2.5 dives into the resource allocation constraints specific to the reference architecture, providing a deeper analysis. The performance of the proposed algorithms is thoroughly evaluated in Section 2.6. Furthermore, in Section 2.7, we explore enhancements in the control plane using traffic estimation and in the architecture with deploying a parallel network, and conduct simulations with realistic traffic scenarios. Finally, our conclusions are summarized in Section 2.8, bringing together the key insights and findings from our work.

Figure 2.1 Optical DCN architecture utilizing fiber rings and WSSs.



2.2 Hybrid Electrical/Optical Interconnect

We investigate a hybrid electrical/optical DCN architecture, built out of POD and top-of-rack (ToR) switches. Figure 2.1 illustrates the NEPHELE DCN, which we use as reference DCN architecture in this chapter. It is divided into P pods[1] of racks, with each pod consisting of I POD switches and W ToR switches, interconnected as follows: *each ToR switch listens to a specific wavelength* (thus, by design, the number of wavelengths equals the number W of racks in a pod) and has I ports. Each port is connected to a different one of the I POD switches. A rack consists of S (computer, storage, or memory) servers. The ToR is a hybrid electrical/optical switch, and each of the S servers of the rack connects to it via a link. Thus, a ToR switch has S ports facing "south" to the servers.

POD switches are interconnected via WDM rings to form "optical planes". An optical plane consists of a single POD switch per pod (for a total of P POD switches in the DCN) connected with R fiber rings. Each fiber ring carries WDM traffic over W wavelengths (W, by design, equals the number of racks in the pod), propagating in the same direction. There are I identical and independent/parallel (in the sense that traffic entering a plane stays in it until the destination) optical planes. In total, there are $I \cdot P$ POD switches, $W \cdot P$ ToR switches, and $I \cdot R$ fiber rings.

We now explain how communication is performed in the reference DCN (Figure 2.1). The key routing concept is that *each ToR switch listens to a specific wavelength* (out of W available), *and wavelengths are re-used* among pods. The ToRs use tunable transmitters that

[1]The term "pod" refers to the cluster of racks, and "POD" to a pod switch.

are tuned according to the desired destination. Each ToR employs Virtual Output Queues (VOQs) per ToR destination ($W \cdot P$ VOQ per ToR) to avoid head-of-line blocking.

Traffic in the form of an optical signal originating from a port (plane) of a ToR switch enters a POD switch and is switched through a fast 1×2 space switch according to locality: if the traffic is destined to a ToR in the pod (intra-pod), it remains within the POD switch; otherwise, it is routed to the rings and to the next POD switch. Local intra-pod traffic enters a $W \times 1$ power combiner, located after the 1×2 space switch, and then an $1 \times W$ arrayed waveguide grating (AWG). The AWG passively routes the traffic, depending on the used wavelength, to the desired destination.

Inter-pod traffic is routed via the fast 1×2 switch toward a $W \times R$ cyclic AWG (CAWG) followed by couplers that combine the CAWG outputs into the R fiber rings. The $W \times R$ CAWG has a passive routing functionality, with the incoming signal being routed to the output port (ring):

$$r = (w_s + w_d - 1) \pmod{R} \tag{2.1}$$

where $1 \leq w_s \leq W$ is the input port (the index of the source rack in the source pod, which equals its listening wavelength), $1 \leq w_d \leq W$ is the wavelength that has to be used to reach the specific destination (thus, also equal to the index of the destination rack in the destination pod), and "mod" denotes the modulo operation. In the simple (not cyclic) $1 \times W$ AWG, the output depends only on the used wavelength. So, the traffic enters the ring according to the CAWG function, propagates in the same ring through intermediate POD switches, and is dropped at the destination pod. These routing decisions are applied by setting appropriately the wavelength selective switches (WSSs) in the related POD switches. The WSSs can select whether traffic passes through or drops on a per-fiber, per-wavelength, and per-slot basis. Thus, each intermediate POD sets the corresponding WSS to the pass state, while at the destination the related WSS is set to the drop state. The drop ports of all the WSSs-corresponding to all the rings-are connected to a power combiner and a $1 \times W$ AWG. So again, the traffic once dropped is passively routed to the desired ToR according to the wavelength used.

Following the above, wavelengths are statically assigned to racks, to simplify optical routing, and are re-used for efficient operation. Conflicts on the WDM rings are avoided in the time and space (plane) domains. Regarding the time domain, the DCN operates in a synchronous slotted manner that closely resembles the operation of a single (huge and distributed) time-division multiple access (TDMA) switch. In particular, it maintains the timeslot component of TDMA, but timeslots are not statically assigned; instead, a central scheduler dynamically assigns them based on traffic needs, enabling efficient utilization of the resources. However, making scheduling decisions on a per-timeslot basis is prohibitive,

Table 2.1 Fully fledged DCN parameters.

Parameter	W	P	S	I
Value	80	20	20	20

due to high communication and processing latency. Instead, it is both more efficient and less computationally demanding to perform resource allocation periodically, so that scheduling decisions are made for periods of T timeslots; this approach facilitates the aggregation and suppression of monitoring and control data and also absorbs traffic peaks.

From the control plane perspective, configurable components are the tunable transmitters (I per ToR switch), the 1×2 optical switches (W per POD switch), and the WSSs (R per POD switch). The timeslot duration is lower bounded by the slowest component, which is the WSS with a switching time of about 10 μs (63). This is reserved as a guardband and the timeslot is taken to be 0.2 ms, so that the network exhibits 95% efficiency. The amount of data transmitted during a timeslot equals the wavelength transmission rate times the timeslot duration (i.e., $0.2\text{ ms} \times 10\text{ Gbps} = 2\text{ Mbits}$) and will be referred to as a data unit (DU), which also is the switching granularity. A reference number for T is 80 timeslots, corresponding to a period of 16 ms.

The existence of I parallel planes provides an additional domain, the space, to resolve conflicts: each timeslot of each plane can be independently allocated. We will refer to a timeslot/plane combination as a generic (time)slot, implying that the space and time domains are interchangeable.

Variations of the above described baseline architecture include cases where each ToR does not listen to a specific wavelength. One such variation will be given in Section 2.5. Still, the routing function remains similar: the transmitter needs to select the appropriate wavelength, which is pre-calculated based on certain parameters (the source, the destination, the plane, etc., as opposed to only the destination in the baseline architecture), while the WSSs are configured according to that wavelength mapping.

Since a CAWG is used to route the W wavelengths on R rings, we must have $W \geq R$ in order for the CAWG to be able to use all R egress ports. This is a system constraint. We can also derive the required conditions for achieving FBB assuming that the DCN is nonblocking (see sections 2.4 and 2.5). We say that a DC interconnect has FBB if, for any bisection of the servers in two equal partitions, each server of one partition is able to communicate at full rate with any server of the other partition. Since a ToR supports S servers, the number of PODs connected to a ToR must be at least $I \geq S$, so that all servers of a ToR can communicate with servers outside their rack. Considering the whole network, there are $P \cdot W \cdot S$ server ports, whereas the overall capacity in the POD-to-POD network is $I \cdot R \cdot W$. Thus, for FBB,

we need to have $I \cdot R \cdot W \geq P \cdot W \cdot S \implies I \cdot R \geq P \cdot S$. Assuming $I = S$, the FBB requirement becomes $R \cdot P$. More flexibility is obtained by increasing the number of planes I. In the presence of traffic locality, the FBB requirement can be relaxed to support larger DCs. Table 2.1 presents target values satisfying the above constraints (including FBB) for a *fully fledged* DCN using commodity off-the-shelf (COTS) equipment and a reference DC size (32K servers).

2.3 Bandwidth Allocation and Control Scheme

The reference DCN architecture exploits the SDN concept that decouples data and control planes through open interfaces, enabling programmability of the networking infrastructure. It utilizes an optical network with I optical planes, R fibers/plane and W wavelengths/fiber to interconnect the ToR switches in P pods. As discussed above, the network operates in a slotted and synchronous manner. A key functionality of the SDN controller is the coordination of the networking resources usage, including the timeslot/plane dimension. Thus, an important building block of the SDN controller is the *scheduling engine*, which allocates resources to communicating ToR pairs in a centralized, periodic, and on-demand manner[1].

Recall that the number of racks per pod is equal to the number of wavelengths, and each rack listens to a specific wavelength. A ToR switch s is thus defined by a unique pair $s = (p_s, w_s)$, where p_s, $1 \leq p_s \leq P$, is the index of the pod it belongs to, and w_s, $1 \leq w_s \leq W$, is the rack index within the pod (w_s is also the wavelength on which ToR s receives data). It will sometimes be convenient to represent the ToR switch by the scalar index $s = p_s \cdot (W-1) + w_s$ instead of the pair representation (p_s, w_s); as the mapping between the two representations is one-to-one, we will use, with a slight abuse of notation, the same symbol s to stand for the ToR itself, the scalar index, and the pair representing it.

We assume that a Data period consists of T timeslots, and we denote by $\mathbf{Q}(n)$ the *queue matrix* for period n. The queue matrix $\mathbf{Q}(n)$ is of size $(W \cdot P) \times (W \cdot P)$, and element $\mathbf{Q}(n)[s, d]$ corresponds to the number of DUs that are queued at the start of the period n at source ToR s and have as destination ToR d, with $s = p_s \cdot (W-1) + w_s$, $d = p_d \cdot (W-1) + w_d$, $1 \leq w_s, w_d \leq W$, and $1 \leq p_s, p_d \leq P$. That is, $\mathbf{Q}(n)[s, d]$ is the number of DUs in VOQ (s, d) at the start of period n. Since the scheduling problems of the different wavelengths are *not* independent, we will avoid breaking this matrix per wavelength.

Two operation modes are envisioned: (i) application-aware and (ii) feedback-based networking. The former approach (31, 41) assumes that applications communicate to the SDN controller (or via the DC orchestrator) their topology and traffic requirements. In that case, the queue matrix is constructed from input from the applications. The latter, feedback-based, mode assumes that the central controller collects (monitors) data from the ToR queues (77) to build the queue matrix. We can also have a hybrid application-aware and feedback-based network. In the following, we focus on the feedback-based approach, which is the hardest of the two from the control and scheduling viewpoint. The analysis and the proposed algorithms are applicable with minor changes to application-aware and hybrid operation.

[1] In the following, the terms "bandwidth allocation", "resource allocation", and "scheduling" will be used interchangeably

Recall that the matrix $\mathbf{Q}(n)$ records the queue sizes at the start of the period n. We denote by $\mathbf{A}(n)$ the matrix of arrivals at the queues during period n and by $\mathbf{S}(n)$ the schedule calculated for period n. Element $\mathbf{Q}(n)[s,d]$ denotes the DUs in the (s,d) queue at the start of the period n, element $\mathbf{A}(n)[s,d]$ the DU arrivals during the period n, and from $\mathbf{S}(n)$ we get the DUs scheduled to be transferred from s to d during the period n. We will describe in the next section the way schedule $\mathbf{S}(n)$ is calculated.

Under feedback-based operation, the DCN operates in two parallel cycles:

1. data communication cycles of T timeslots (also referred to as a *Data period*), where the actual communication between ToRs takes place, and

2. resource allocation cycles of duration C (measured in Data periods of T timeslots), where control information is exchanged. If the duration of the resource allocation process is not fixed, C is an upper bound on it.

The resource allocation cycle n corresponds to a Data period n, and computes the schedule $\mathbf{S}(n)$ to be used during that period. Note, however, that the schedule is computed based on information that was available C periods earlier than the Data period to which the resource allocation cycle corresponds (and is applied). Thus, $\mathbf{S}(n)$ is a function of $\mathbf{Q}(n-C)$, i.e.,

$$\mathbf{S}(n) = \mathscr{F}(\mathscr{G}(\mathbf{Q}(n-C))) \tag{2.2}$$

where $\hat{\mathbf{Q}}(n) = \mathscr{G}(\mathbf{Q}(n-C))$ is the function that creates the *estimated queue matrix* $\hat{\mathbf{Q}}(n)$ from $\mathbf{Q}(n-C)$ upon which the schedule is calculated, and $\mathscr{F}$ is the scheduling algorithm. When $C > 1$ period (control delay larger than the Data period), a new resource allocation cycle still starts every Data period. So, there are C resource allocation cycles (or virtual control planes) running in parallel. For determining the schedule $\mathbf{S}(n)$ to be used during Data period n:

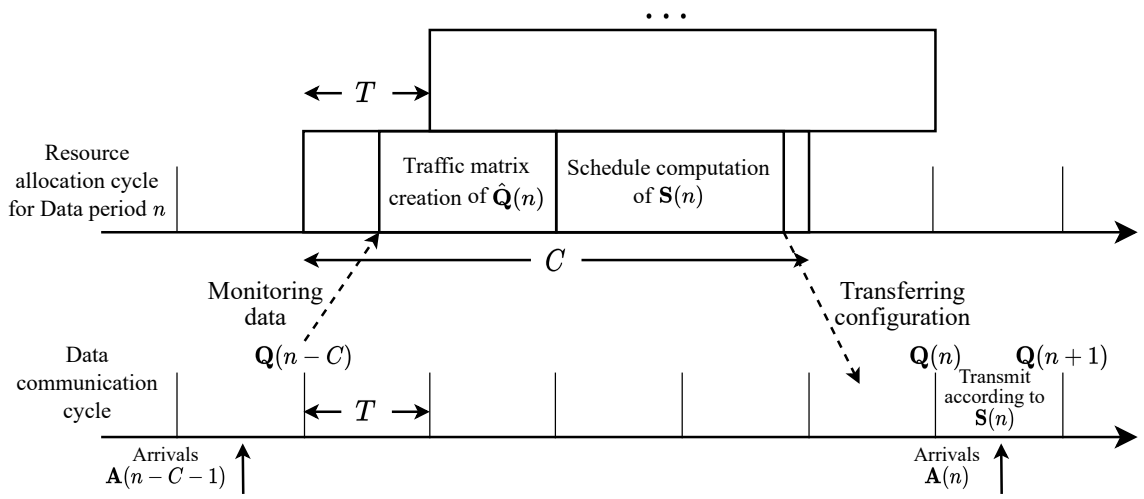
a) the traffic matrix engine of the SDN controller collects the queue sizes from the ToRs to build $\mathbf{Q}(n-C)$ and runs the queue estimation algorithm $\mathscr{G}$ to create the *estimated queue matrix* $\hat{\mathbf{Q}}(n) = \mathscr{G}(\mathbf{Q}(n-C))$,

b) the scheduling engine of the SDN controller runs the algorithm $\mathscr{F}$ to calculate the schedule $\mathbf{S}(n) = \mathscr{F}(\hat{\mathbf{Q}}(n))$, and

c) the SDN controller communicates the scheduling output $\mathbf{S}(n)$ to the data plane devices (POD and ToR switches) to be used during Data period n.

Figure 2.2 shows the resource allocation and data cycles (control and data plane, respectively). As discussed, there is a delay between the two cycles: the schedule $\mathbf{S}(n)$ applied

Figure 2.2 Resource allocation and data cycles.



in Data cycle n is computed based on queue matrix $\mathbf{Q}(n-C)$, since it takes C periods to compute and reach the data plane devices. The queue evolution is described by

$$\mathbf{Q}(n+1) = \mathbf{Q}(n) + \mathbf{A}(n) - \mathbf{S}(n) \tag{2.3}$$

where $\mathbf{S}(n) = \mathscr{F}(\mathscr{G}(\mathbf{Q}(n-C)))$. The value C does not affect the achievable throughput, as long as scheduling decisions are efficient (more on that later), but affects the traffic delay. The control plane delay C depends on many factors, including the execution time of the scheduling algorithm, and the delay of the control protocol carrying information from ToRs to the SDN controller (if monitoring is assumed) and from the SDN controller to the data plane devices. Both delays depend on the network size and the choice of the Data period T.

For scheduling decisions to be efficient, the scheduling matrix $\mathbf{S}(n)$, computed based on an estimated queue matrix $\hat{\mathbf{Q}}(n)$, which in turn is calculated by $\mathbf{Q}(n-C)$, should be a "good" scheduling to be used during the Data interval n. This is true when $\hat{\mathbf{Q}}(n)$ is a good approximation of $\mathbf{Q}(n)$.

For slowly and medium changing traffic, we expect calculations made for previous periods to be valid. In estimating $\hat{\mathbf{Q}}(n)$ from $\mathbf{Q}(n-C)$, it is possible to also use statistical predictions, filters, and other (notably application-aware) methods to improve performance. Moreover, it is possible for the scheduler to fill unallocated resources in $\mathbf{S}(n)$ by opportunistic transmissions, which can have collisions or be collision free (e.g., nodes agree to use slots in lexicographic order, mimicking static TDM, which under heavy load is efficient). Finally, the overall scheme is "self-correcting": if some queues are not served for some periods due

to poor scheduling and their size grows due to new arrivals, this will be communicated with some delay to the controller, and the queues will eventually be served.

In the following, we will focus on the scheduling problem. We start from the estimated queue matrix $\hat{\mathbf{Q}}(n)$ and devise fast algorithms to calculate the schedule $\mathbf{S}(n)$ (function $\mathscr{F}$ in Eq. (2.2)). For reference, we can assume that we calculate the estimated queue matrix (function g in Eq. (2.2)) as

$$\hat{\mathbf{Q}}(n) = \mathbf{A}(n - C - 1) + \hat{\mathbf{Q}}(n - 1) - \mathbf{S}(n - 1),$$

where we acknowledge that due to control plane delay C, the central scheduler has access to (delayed) arrival information $\mathbf{A}(n - C - 1)$ instead of $\mathbf{A}(n)$. This corresponds to the case where the schedule $\mathbf{S}(n)$ calculated on $\hat{\mathbf{Q}}(n)$ serves the arrived traffic $\mathbf{A}(n - C - 1)$, plus a correction equal to traffic not served in the previous period $\hat{\mathbf{Q}}(n - 1) - \mathbf{S}(n - 1)$.

We now describe the form of the schedule $\mathbf{S}(n)$. The scheduling engine provides the ToR pairs that communicate *during each timeslot and for each optical plane* within the upcoming Data period. Note that wavelengths and rings are dependent resources; the selected wavelength is determined by the destination, and the ring depends on the source and destination according to Eq. (2.1). Thus, the allocated resources are the timeslots and the optical planes ($I \cdot T$ in total), or the generic slots, as stated previously.

The scheduling algorithm takes the estimated queue matrix $\hat{\mathbf{Q}}(n)$ and decomposes it (fully or, if not possible, partially) into a sum of $I \cdot T$ *permutation matrices* $\mathbf{P}^{(g)}(n)$, $g = 1, 2, \ldots, I \cdot T$, each corresponding to a generic slot. A permutation matrix is binary of size $(W \cdot P) \times (W \cdot P)$; an entry $\mathbf{P}^{(g)}(n)[s, d]$ equals "1" if a DU is to be transferred from ToR s to ToR d during the g^{th} generic slot of period n, and "0" otherwise. In other words, $\mathbf{P}^{(g)}(n)[s, d]$ identifies if one DU at the d-VOQ of ToR s will be transmitted during the g^{th} generic slot of period n:

$$\mathbf{P}^{(g)}(n)[s, d] = \begin{cases} 1, \text{if } \mathbf{S}(n)[g, s] = d \\ 0, \text{otherwise.} \end{cases}$$

A permutation matrix determines a configuration of the network for a specific generic slot. For the communication to be contention free, the *scheduling constraints* SC1, SC2, and SC3 that are summarized in Table 2.1 should be satisfied. In particular, the first two constraints, SC1 and SC2, ensure that each ToR transmits to and receives from at most one ToR per generic slot. Constraints SC1 and SC2 are relevant to all TDMA-like architectures and are readily enforced by the decomposition process.

The third constraint, SC3, is related to the (not nonblocking character of the) architecture, and particularly, it is a result of the usage of static routed CAWGs as opposed to dynami-

Table 2.2 Scheduling constraints (SC).

Constraint ID	Description[1]
SC1	$\sum_s \mathbf{P}^{(g)}(n)[s,d] \leq 1$
SC2	$\sum_d \mathbf{P}^{(g)}(n)[s,d] \leq 1$
SC3	$\mathbf{P}^{(g)}(n)[s,d] + \mathbf{P}^{(g)}(n)[s,' d'] \leq 1$, for $p_s < p_{s'} < p_d$ or $p_s < p_{d'} < p_d$ and $(w_{s'} - w_s) \pmod R = 0$

[1] $\mathbf{P}^{(g)}(n)[s,d] = 1, s = p_s(W-1) + w_s$, and $d = p_d \cdot (W-1) + w_d$ indicate that one DU is scheduled for transfer from source ToR (w_s, p_s) to destination ToR (w_d, p_d) in the g^{th} generic slot of period n.

cally configured components. To better illustrate SC3, assume that a source ToR (w_s, p_s) communicates with a destination ToR (w_d, p_d). This communication takes place over the optical ring that is calculated from Eq. (2.1), and it occupies a wavelength w_d on the ring segment between p_s and p_d. If another source ToR $(w_s, p_{s'})$ within the aforementioned ring segment (i.e., $p_s < p_{s'} < p_d$) concurrently communicates with destination ToR $(w_d, p_{d'})$, a collision will occur irrespective of the destination pod $(p_{d'})$, since it occupies the same ring and wavelength. A similar contention will occur if the destination pod lies in the initial ring segment (i.e., $p_s < p_{d'} < p_d$), irrespective of the source pod. Note that SC3 is alleviated for $R \geq W$, which, however, leads to underutilization of rings. Moreover, the effect of the lack of the nonblocking property for the architecture (when seen as a huge switch), or equivalently the existence of SC3, is small, and will be discussed in sections 2.5 and 2.6.

The set $\mathbf{P}^{(g)}(n), g = 1, 2, \ldots, I \cdot T$, of permutation matrices comprise the schedule $\mathbf{S}(n)$, which records information for all generic slots of period n. The permutation matrices $\mathbf{P}^{(g)}(n)$ are stored as sparse matrices, each with $W \cdot P$ entries. Similarly, $\mathbf{S}(n)$ is sparse with $I \cdot T \cdot W \cdot P$ entries.

2.4 Scheduling Algorithms

Having described the DCN operation, we now proceed to present a set of scheduling algorithms. We assume that we start with the estimated queue matrix $\hat{\mathbf{Q}}(n)$ and calculate the schedule $\mathbf{S}(n)$ (function $\mathscr{F}$ in Eq. (2.2)). To target both static and dynamic resource allocation scenarios, we developed two classes of scheduling algorithms: (i) *offline* and (ii) *incremental*. Offline algorithms, given in Section 2.4.1, take the estimated queue matrix $\hat{\mathbf{Q}}(n)$ and compute schedule $\mathbf{S}(n)$ "from scratch". Incremental algorithms, given in Section 2.4.3, use the previous schedule $\mathbf{S}(n-1)$ and update it based on traffic changes to obtain $\mathbf{S}(n)$. Offline algorithms are better suited for semi-static traffic, take longer to execute, and achieve better utilization; incremental algorithms are faster and can handle dynamic scenarios.

2.4.1 Offline Scheduling

As discussed above, offline scheduling decomposes the matrix $\mathbf{Q}(n)$ into a sequence of permutation matrices $\mathbf{P}^{(1)}(n), \mathbf{P}^{(2)}(n), \ldots, \mathbf{P}^{(I \cdot T)}(n)$, without taking into account the previous decomposition. We start by presenting the optimal offline scheduling algorithm.

The decomposition of $\hat{\mathbf{Q}}(n)$ can be performed optimally following the well-known Hall's theorem (an integer version of the Birkhoff-von Neumann theorem (18)). We define the critical sum $\mathscr{H}(\hat{\mathbf{Q}}(n)) = h$ of a matrix $\hat{\mathbf{Q}}(n)$ as the maximum of its row sums and column sums. Then the following theorem holds:

Theorem 2.4.1 (Hall's Theorem)**.** *An integer matrix of critical sum h can be written as the sum of h permutation matrices.*

The following algorithm calculates the optimal decomposition of matrix $\hat{\mathbf{Q}}(n)$:

1. Find a matrix of non-negative integers $\mathbf{E}(n)$ so that the matrix $\mathbf{M}(n) = \hat{\mathbf{Q}}(n) + \mathbf{E}(n)$ is a perfect matrix with critical sum $\mathscr{H}(\mathbf{M}(n)) = \mathscr{H}(\hat{\mathbf{Q}}(n)) = h$. A perfect matrix has the sum of each row and each column equal to the critical sum. An algorithm to obtain $\mathbf{E}(n)$ is found in (19).

2. Treat $\mathbf{M}(n)$ as a (bipartite) graph adjacency matrix and obtain a maximum matching $j \rightarrow p(j), j = 1, 2, \ldots, P \cdot W$. This matching can then be represented as a permutation matrix $\mathbf{P}^{(i)}(n)$, whose $(j, p(j))$ entries are equal to 1, and all other entries are 0.

3. Find the weight c_i as the smallest element of $\mathbf{M}(n)$ that corresponds to a nonzero entry in $\mathbf{P}^{(i)}(n)$.

4. Repeat $\mathbf{P}^{(i)}(n)$ for c_i times in the schedule and update $\mathbf{M}(n) = \mathbf{M}(n) - c_i \cdot \mathbf{P}^{(i)}(n)$.

5. If $\mathbf{M}(n)$ is not equal to zero, repeat steps 1-4. Otherwise, an optimal decomposition for $\mathbf{M}(n)$ has been found.

6. Set the entries of the dummy matrix $\mathbf{E}(n)$ to zero.

Steps 2-4 are repeated h times at most, and we have that $\sum_i c_i = h$. Note that the decomposition of an integer matrix as a sum of h permutation matrices is not unique and that the permutation matrices in the decomposition of $\mathbf{M}(n)$ are full rank (corresponding to full utilization of the $I \cdot T$ generic slots), while those in the decomposition of $\hat{\mathbf{Q}}(n) = \mathbf{M}(n) - \mathbf{E}(n)$ may not be full rank (leaving some generic slots unused, namely, the entries of $\mathbf{E}(n)$, and available for opportunistic transmissions). In general, a decomposition that uses a limited number of permutations, each carrying a considerable amount of traffic c_i, is preferable as it results to fewer reconfigurations of the switches.

The preceding algorithm assumes that the critical (row or column) sum is constrained, but this will not always be the case. The arrival matrix $\mathbf{A}(n)$ corresponds to traffic created by the servers and aggregated at the related ToR switches in period n. Since one link connects a server to the ToR, the server sends to its ToR switch at most 1 DU during a timeslot. Therefore, the row sums of $\mathbf{A}(n)$ are at most $S \cdot T$. Some of $\mathbf{A}(n)$'s column sums, however, may be larger than that, e.g., in the presence of hotspot destinations. Note that the capacity connecting a ToR to the destination servers can transfer $S \cdot T$ DUs, and this is the same for all DCNs. So hotspot problems, where traffic toward some ToRs (columns of $\mathbf{A}$) exceeds the available capacity, are present in all DCNs.

We could, in principle, devise flow control mechanisms to guarantee that the critical sum of $\mathbf{A}(n)$ satisfies $\mathscr{H}(\mathbf{A}(n)) \leq S \cdot T$. Using an entry flow control mechanism between servers and source ToRs, like the "stop and go" queuing proposed in (32), which limits (smoothens) the entry of DUs toward the destinations, we can enforce the column sum to be less than $S \cdot T$. In particular, each source ToR can check the destination ToR d of the packets forwarded to it by the source servers and, through a back-pressure mechanism, guarantee that packets equivalent to at most $(S \cdot T)/(W \cdot P)$ DUs are destined for each destination during a period of duration T. Such a source flow control mechanism, however, may be too restrictive, unnecessarily, and introduce large entry delays, as packets are queued at the servers, outside the interconnection network. To relax somewhat the constraint, a credit-based flow control mechanism can be used at the pod level, where each source POD is given $W_d = (S \cdot T)/P$ credits for each destination ToR d per period, which it can distribute to the ToRs below it that can, in turn, distribute them to the servers. This would relax considerably the input flow control constraints and the corresponding delays at the servers, but requires a clever mechanism for distributing credits.

Even if a flow control mechanism is not present, the column sums will be on average less or equal to $S \cdot T$, assuming the destinations of packets are uniformly distributed on average. Actually, the critical sum will be less or equal to $S \cdot T$ not only on average, but also with high probability, if the network operates at less than full load. Finally, note that TCP flow control smoothens the traffic to a given destination. Since the downstream links from a ToR to the servers can support up to $S \cdot T$ DU per Data period, the previous condition will tend to hold with high probability in a DC network that employs TCP.

Based on the previous discussion, we conclude that in the "typical case" the column sums of the arrival matrix $\mathbf{A}(n)$ will be less or equal to $S \cdot T$ and so will also be its critical sum (since the row sums are always less or equal to $S \cdot T$). In that case, the schedule $\mathbf{S}(n)$, that is calculated based on $\hat{\mathbf{Q}}(n) = \mathbf{A}(n - C)$, assuming $S \leq I$, can be chosen to completely serve all the arrivals in $\mathbf{A}(n - C)$ in the available $I \leq T$ generic slots.

Note that in the reference FBB network scenario we assume $S = I$ and so we will interchangeably use S and I in the following. Thus, in this case, all packets generated in a Data period will be served C periods later, emptying the queue from such packets. So the delay in the DCN is upper bounded by C periods when appropriate input flow control is used, or with high probability when the load is far enough from full load. Thus, in the typical case, the reference DC provides both *full throughput and delay guarantees*.

In the more general case where the critical sum of $\hat{\mathbf{Q}}(n)$ is not bounded by $I \cdot T$, we stop when we find the first $I \cdot T$ permutations, while the traffic $\mathbf{Q}(n) - \mathbf{S}(n)$ that is not served is fed to produce the estimated matrix for the next period $\hat{\mathbf{Q}}(n+1)$. Fairness and priority issues can also be handled with small extensions to the above process without a requirement for additional flow control.

2.4.2 Complexity of Offline Scheduling and Stability

For general traffic, we define the load intensity between source destination ToR pair (s, d) as

$$\rho_{sd}(\mathbf{A}) = \mathbb{E}[\mathbf{A}[s, d]]/(I \cdot T), \tag{2.4}$$

where $\mathbb{E}$ stands for expected value and $0 \leq \rho_{sd}(\mathbf{A}) \leq 1$ for an FBB DCN ($S = I$). The load intensity matrix $\mathscr{P}(\mathbf{A})$ is defined as the matrix with $\rho_{sd}(\mathbf{A})$ entries. The row sums of $\mathscr{P}(\mathbf{A})$ are always less than or equal to 1, while for a stable network (finite queues), the column sums should also be less than or equal to 1.

Necessary condition for stability: For the DCN to be stable, the load intensity matrix $\mathscr{P}(\mathbf{A})$ should be at most a double stochastic matrix.

When the previous condition does not hold, it is impossible to find a schedule to serve the queues in a stable manner. It is thus up to the DC orchestrator to allocate tasks to servers so that their communication requirements meet this constraint. Our target is to provide schedules that can serve any (long-term) stable matrix $\mathscr{P}(\mathbf{A})$.

We define the *average* network load $\rho(\mathbf{A})$ (also represented by ρ) for arrival matrices $\mathbf{A}$ as the scalar

$$\rho(\mathbf{A}) = \rho = \sum_{s,d} \rho_{sd}(\mathbf{A})/(P \cdot W) = \sum_{s,d} \mathbb{E}[\mathbf{A}[s,d]]/(I \cdot T \cdot P \cdot W), \tag{2.5}$$

and $\rho(\mathbf{A}) \in [0,1]$. The quantity $\rho \cdot P \cdot W \cdot I \cdot T$ equals the average of the entries of the arrival matrix $\mathbf{A}$ during a period (or, equivalently, $\rho \cdot P \cdot W \cdot I$ is the average number of arrivals per timeslot and ToR-to-ToR pair).

Besides the load, another parameter that is important in characterizing the arrival process and the algorithmic complexity is the *arrival matrix density* $\delta(\mathbf{A})$, which is complementary to the sparsity of $\mathbf{A}$. In particular, if we define the indicator function $\mathscr{I}(\cdot)$, as

$$\mathscr{I}(x) = \begin{cases} 1, \text{if } x > 0 \\ 0, \text{otherwise,} \end{cases}$$

then the density $\delta(\mathbf{A})$ of the matrix $\mathbf{A}$ is defined as

$$\delta(\mathbf{A}) = \mathbb{E}\left[\sum_{s,d} \mathscr{I}(\mathbf{A}[s,d])\right] /(W \cdot P)^2, \tag{2.6}$$

where $\mathbb{E}\left[\sum_{s,d} \mathscr{I}(\mathbf{A}[s,d])\right]$ is the average number of nonzero entries of $\mathbf{A}$ and, clearly, $0 \leq \delta(\mathbf{A}) \leq 1$. In other words, $\delta(\mathbf{A})$ is the fraction of nonzero entries of $\mathbf{A}$. Then, the number of nonzero entries $\mathscr{M}(\mathbf{A})$ is given by $\mathscr{M}(\mathbf{A}) = \delta(\mathbf{A}) \cdot (W \cdot P)^2$.

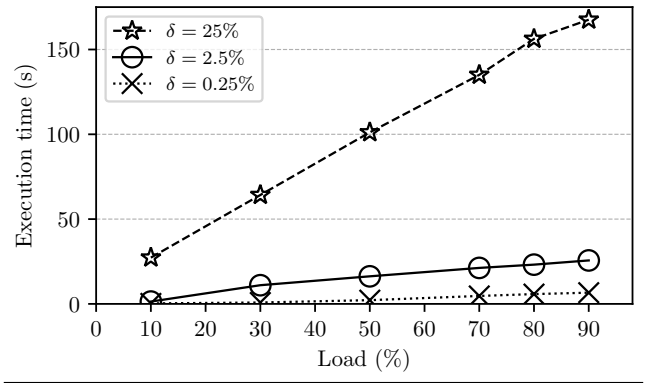
In the worst case, the optimal algorithm described earlier executes a maximum matching algorithm $I \cdot T$ times (uniform traffic). Finding a maximum matching can be time-consuming, and even the well-known Hopcroft-Karp bipartite graph matching algorithm (35) exhibits a complexity of $O(\mathscr{M}(\mathbf{A}) \cdot \sqrt{W \cdot P})$, where $\mathscr{M}(\mathbf{A})$ is the number of nonzero elements in $\mathbf{A}$. The number of different matches is $\rho \cdot I \cdot T$, and thus the complexity of the optimal offline algorithm is $O(\rho \cdot \delta \cdot I \cdot T \cdot (W \cdot P)^{\frac{5}{2}})$.

An indicative example of the execution time required for optimal decomposition with the Birkhoff-von Neumann and Hopcroft-Karp algorithms is shown in Figure 2.3, for a fully fledged DCN (parameters listed in Table 2.1). The algorithm was developed in MATLAB

(38) and the simulations were performed on an Intel® Core™ i5 laptop. Figure 2.3 plots the average execution time of the optimal decomposition algorithm against the load ρ and density δ, which are shown to range from tens of seconds to minutes.

In these simulations, the traffic was created as follows: at each period, each source ToR communicated with $\delta \cdot W \cdot P$ uniformly chosen ToR destinations by transmitting a total number of $\rho \cdot I \cdot T$ DUs.

Figure 2.3 Average execution time of optimal decomposition algorithm as a function of load ρ and arrival matrix density δ.



Based on the above result, and given the size of a fully fledged FBB DCN (Table 2.1), we deduce that, even with an optimized software and hardware environment, the optimal algorithm would only be viable under a static resource allocation scenario, where traffic remains unchanged for prolonged periods. The requirement for dynamic resource allocation can be pursued via non-optimal algorithms that employ maximal rather than maximum matchings, at the expense of blocking at high loads. To this end, we also developed faster offline heuristics of reduced complexity and performance quite close to the optimal. In particular, we developed a greedy offline algorithm of complexity $O(\rho(W \cdot P)^2 \cdot I \cdot T)$, which is linear in the size of the problem (note that the number of DUs to be scheduled is $O(\rho \cdot (W \cdot P)^2)$ and the number of resources is $O(I \cdot T)$). For brevity, we do not discuss this algorithm, as it still cannot meet dynamic resource allocation requirements, but describe a variation of it in the next subsection. To further reduce scheduling complexity, we have to exploit the variations (temporal and spatial) of traffic, as is done in the incremental scheduling algorithms of the next subsection.

2.4.3 Incremental Scheduling Algorithms for Locality Persistent Traffic

It is evident from the previous results that offline scheduling is not suitable for bursty traffic. Measurements in commercial DCs indicate that application traffic can be relatively bursty, with flows activating/deactivating within milliseconds (65). Although traffic can be bursty, it tends to be highly locally persistent: a server tends to communicate with a set of destina-

tions that are located in the same rack or the same cluster/pod (65). This is due to the way applications are placed in DCs, each occupying only a small fraction of the DC.

ToR switches aggregate the flows of the servers in a rack, smoothening out the burstiness of individual flows, especially considering locality-persistent traffic. To formally define locality persistency, we define the *arrival matrix difference* as $\mathbf{D}_{\mathrm{A}}(n) = \mathbf{A}(n) - \mathbf{A}(n-1)$, the load $\rho(|\mathbf{D}_{\mathrm{A}}(n)|)$, and the density $\delta(|\mathbf{D}_{\mathrm{A}}(n)|)$ of the difference by replacing $\mathbf{A}$ with $\mathbf{D}_{\mathrm{A}}$ in Eq. (2.5) and (2.6), where $|\cdot|$ stands for the entrywise absolute value.

Locality Persistency Property: holds if

$$\delta(|\mathbf{D}_{\mathrm{A}}(n)|) \ll 1. \tag{2.7}$$

We also define the *estimated queue matrix difference* as

$$\mathbf{D}_{\hat{\mathrm{Q}}}(n) = \hat{\mathbf{Q}}(n) - \hat{\mathbf{Q}}(n-1). \tag{2.8}$$

Note that when arrivals have the locality persistency property (i.e., Eq. (2.6) holds), then, in view of the Section 2.3 discussion, we also expect $\delta(|\mathbf{D}_{\hat{\mathrm{Q}}}(n)|) \ll 1$. For example, in the typical case where

$$\hat{\mathbf{Q}}(n) = \mathbf{A}(n-C-1) + \mathbf{Q}(n-1) - \mathbf{S}(n-1), \tag{2.9}$$

the persistency property of $\mathbf{A}$ also holds for the estimated matrix $\hat{\mathbf{Q}}$.

Motivated from this observation, we propose and investigate *incremental scheduling*, i.e., rely on the previous schedule to calculate the new one. The expected benefit is that we need to update only specific elements of the permutation matrices of the decomposition of $\hat{\mathbf{Q}}(n+1)$, corresponding to traffic that has changed, but there is no need to modify the rest of the elements.

To be more specific, let $\hat{\mathbf{Q}}(n-1)$ be the estimated queue matrix and $\mathbf{S}(n-1)$ be the schedule produced at period $n-1$. To compute schedule $\mathbf{S}(n)$ for the next period n with estimated queue matrix $\hat{\mathbf{Q}}(n)$, we perform the following:

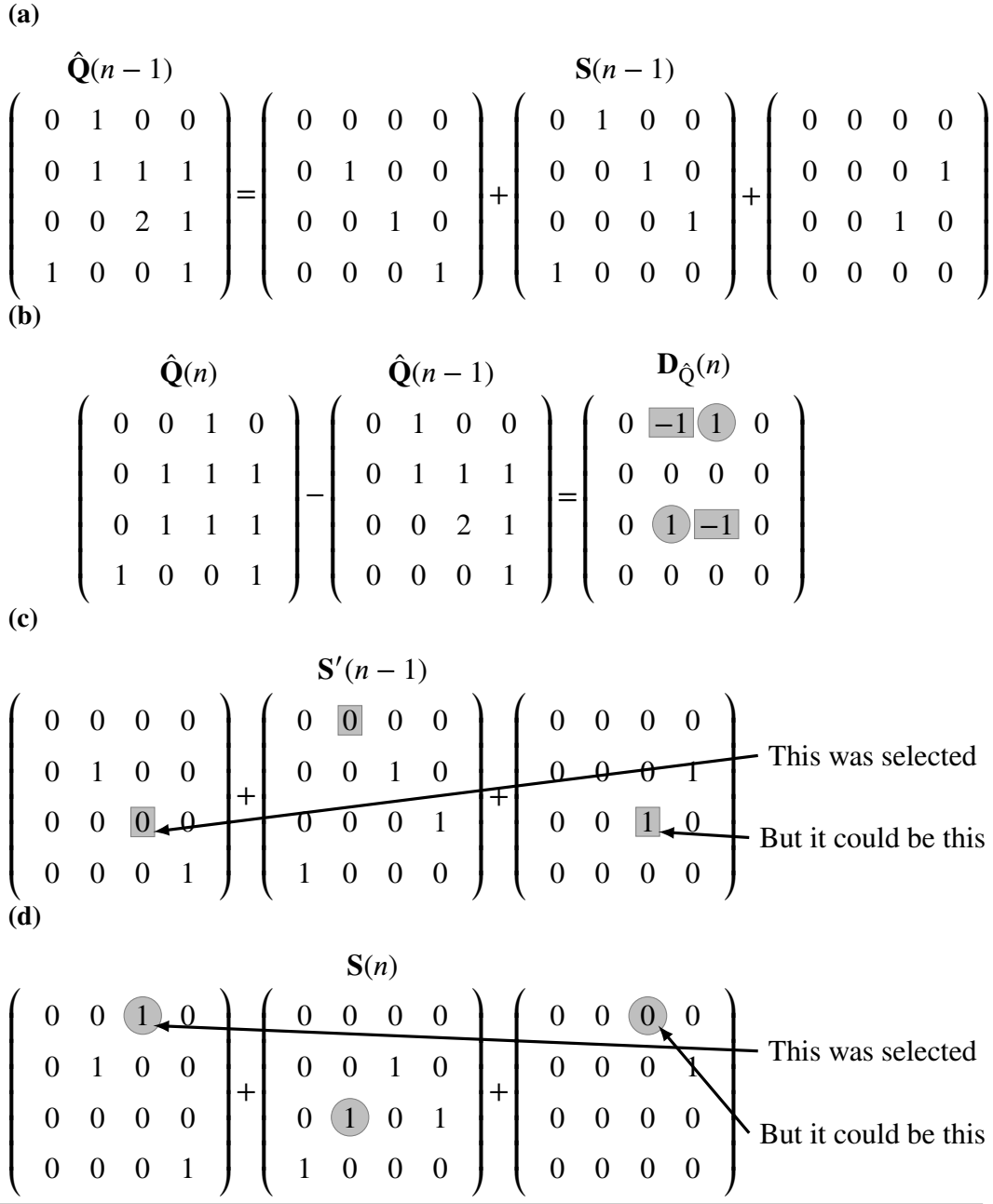
1. Compute $\mathbf{D}_{\hat{\mathrm{Q}}}(n) = \hat{\mathbf{Q}}(n) - \hat{\mathbf{Q}}(n-1)$ (Figure 2.4b).

2. Split $\mathbf{D}_{\hat{\mathrm{Q}}}(n)$ into $\mathbf{D}^{+}(n)$ and $\mathbf{D}^{-}(n)$, where $\mathbf{D}^{+}(n)$ denotes the matrix consisting only of the positive entries of difference matrix $\mathbf{D}_{\hat{\mathrm{Q}}}(n)$, and $\mathbf{D}^{-}(n)$ denotes the matrix consisting only of the negative entries of difference matrix $\mathbf{D}_{\hat{\mathrm{Q}}}(n)$.

3. Use a *freeing algorithm* to free entries of $\mathbf{S}(n-1)$ according to the matrix $\mathbf{D}^{-}(n)$ and obtain the half-filled schedule, denoted as $\mathbf{S}'(n-1)$ (Figure 2.4c).

4. Use a *scheduling algorithm* to add entries in $\mathbf{S}'(n-1)$ (half-filled schedule) according to $\mathbf{D}^{+}(n)$ to obtain the current period's schedule $\mathbf{S}(n)$ (Figure 2.4d).

Figure 2.4 Concept of incremental scheduling.



The complexity of incremental scheduling is $O(\delta(|\mathbf{D}_{\hat{\mathbf{Q}}}|) \cdot \rho \cdot I \cdot T \cdot (W \cdot P)^2)$, where, $\delta(|\mathbf{D}_{\hat{\mathbf{Q}}}|) \ll 1$ in view of the persistency property of Eq. (2.6) and the related discussion.

The above describes the core of the incremental algorithms. In the first two algorithms that we will present, we used a greedy freeing algorithm in step 3 to free entries that works

as follows: by iterating each element of $\mathbf{D}^{-}(n)$, we find the last permutation matrix of $\mathbf{S}(n-1)$ that serves that element, and we free that entry (set it to zero). This algorithm frees sequentially the scheduled resources for the demands whose traffic was reduced, leaving the entries that satisfy the current traffic. Regarding the scheduling algorithm of step 4, we present three heuristic schemes: a) a *linear-time greedy incremental heuristic*, b) a *sublinear greedy incremental heuristic*, and c) a *randomized heuristic*.

Linear-Time Greedy Incremental Heuristic

The greedy heuristic is a non-optimal algorithm running in linear time to the size of the problem and the number of generic slots per period. The greedy heuristic can be used as an offline or as an incremental algorithm. In the following, we focus on the incremental case. The algorithm takes as input the difference traffic matrix $\mathbf{D}_{\hat{\mathbf{Q}}}(n)$ (or $\hat{\mathbf{Q}}(n)$ in offline). It follows steps 1-3 described above, so that it finds the half-filled schedule, denoted as, $\mathbf{S}'(n-1)$ and the positive difference matrix $\mathbf{D}^{+}(n)$. By iterating on each non-zero element of $\mathbf{D}^{+}(n)$, it greedily finds the available generic slots to use. This is done by taking into account constraints SC1 and SC2, of Table 2.1, which ensure that at each generic slot a ToR can send to or receive from only one other ToR. Data structures $\mathbf{TC}(n)$ and $\mathbf{RC}(n)$ are used to keep track of two constraints. In particular, element $\mathbf{TC}(n)[s,g]$ (or $\mathbf{RC}(n)[d,g]$) records whether the transmitter (or receiver) at source s (or destination d, respectively) and generic slot g is active or not. The pseudocode of the incremental greedy algorithm is given in Algorithm 1.

Sublinear Greedy Incremental Heuristic

The sublinear greedy algorithm is a variation of the linear greedy heuristic, but it schedules blocks of DUs instead of DUs. In particular, an integer $k = O(I)$ is chosen and used to calculate the *block estimated queue matrix* $\hat{\mathbf{Q}}^{k}(n) = \frac{\hat{\mathbf{Q}}(n)}{k}$ (in our implementation we chose $k = 5$, and I was a multiple of 5). The purpose of this procedure is to reduce the amount of load to be scheduled, within a span of $T \cdot \frac{I}{k}$ generic slots, speeding up the scheduling process roughly by a factor of k. The block estimated queue matrix is treated as the estimated queue matrix, while applying the previous greedy algorithm. The schedule produced by the greedy algorithm is reproduced k times, in order to cover the initial traffic. As expected, the speedup obtained comes at a cost: dummy DUs are introduced when the ceiling function is applied, which are allocated some generic slots, reducing the resource usage. In particular, the load

Algorithm 1 Linear Greedy Algorithm

Input: $\mathbf{D}^+(n), \mathbf{S}'(n-1), \mathbf{TC}(n-1), \mathbf{RC}(n-1), P, W, T, I$
Output: $\mathbf{S}(n), \mathbf{TC}(n), \mathbf{RC}(n)$

```
1:  S(n) ← Copy of S'(n − 1)
2:  TC(n) ← Copy of TC(n − 1)
3:  RC(n) ← Copy of RC(n − 1)
4:  for s ← 1, 2, ..., P · W do
5:      for d ← 1, 2, ..., P · W do
6:          slots ← D+(n)[s, d]
7:          g ← 1
8:          while g ≤ T · I and slots > 0 do
9:              if TC(n)[s, g] = 0 and RC(n)[d, g] = 0 then
10:                 S(n)[s, g] ← d
11:                 TC(n)[s, g] ← 1
12:                 RC(n)[d, g] ← 1
13:                 slots ← slots − 1
14:             end if
15:         end while
16:         g ← g + 1
17:     end for
18: end for
19: return S(n), TC(n), RC(n)
```

overhead introduced is

$$\text{Number of dummy DUs} = \sum_{s,d} \left(k \cdot \left(\frac{\hat{\mathbf{Q}}(n)[s,d]}{k} \right) - \hat{\mathbf{Q}}(n)[s,d] \right). \tag{2.10}$$

In order for the algorithm to run in sublinear time (a speedup of roughly k is expected), some filtering has to be applied to $\hat{\mathbf{Q}}(n)$ in such a way that its critical sum is at most $(T \cdot I)/k$ after the division, rather than $T \cdot I$. This process takes place in the estimated queue matrix creation module and requires at least linear time to complete. These two operations, however, namely, the estimated queue matrix creation and the scheduling, are performed by different modules. The queue matrix creation module can start executing while receiving monitoring information; once the block estimated queue matrix is created, the scheduling algorithm is executed in sublinear time. We consider this to be technically feasible for the reference DC's architecture.

Randomized Heuristic

A randomized variation of the greedy heuristic was also implemented for an incremental resource assignment. Randomized operation avoids the greedy first find approach, aiming to increase (on average) the traffic that is served (74). The algorithm follows an approach similar to the four steps presented at the start of this subsection: it receives as input the previous period's schedule $\mathbf{S}(n-1)$, the estimated queue matrix $\hat{\mathbf{Q}}(n)$, and calculates the schedule $\mathbf{S}(n)$. In the first phase, it examines the previous period's permutation matrices $\mathbf{P}^{(g)}(n-1)$ against the traffic they can carry in the new period and discards any $\mathbf{P}^{(g)}(n-1)$ that carries less traffic than $\sum_{s,d} \hat{\mathbf{Q}}(n)[s,d]/(I \cdot T)$, expecting that a new randomized allocation could provide a better solution for the corresponding generic slot. The $\mathbf{P}^{(g)}(n-1)$ matrices that carry their fair share of the traffic load are then subtracted from $\hat{\mathbf{Q}}(n)$:

1. If the subtraction of a $\mathbf{P}^{(g)}(n-1)$ leaves no negative entries, then the $\mathbf{P}^{(g)}(n-1)$ is kept unaltered in $\mathbf{S}'(n-1)$.

2. Whenever negative entries occur, the corresponding entries on both $\mathbf{P}^{(g)}(n-1)$ and $\hat{\mathbf{Q}}(n)$ are set to zero, and the updated $\mathbf{P}^{(g)}(n-1)$ is used in $\mathbf{S}'(n-1)$.

The previous steps calculate (i) the updated set of permutations $\mathbf{S}'(n-1)$, by skipping the calculation of $\mathbf{D}^{-}(n)$, and (ii) the positive change matrix $\mathbf{D}^{+}(n)$, which is the $\hat{\mathbf{Q}}(n)$ matrix after the subtractions. In this case, $\mathbf{D}^{+}(n)$ includes the new connections, the old connections with increased traffic, and the old connections that belonged to discarded permutations. Then the entries of $\mathbf{D}^{+}(n)$ are distributed randomly on $\mathbf{S}'(n-1)$ following the algorithm below:

Algorithm 2 Randomized Heuristic Algorithm

1. Select a random destination ToR (column) d of $\mathbf{D}^{+}(n)$.
2. Find the m active source ToRs for destination d, corresponding to rows $\{s_1, s_2, \ldots, s_m\}$ of the non-zero entries in the column d, and re-arrange them randomly.
3. For each row s_k in the randomized arrangement:
   a) Find the existing $\mathbf{P}^{(g)}(n)$ that are available for the (s_k, d) communication (by checking the related scheduling constraints, using the data structures $\mathbf{TC}(n-1)$ and $\mathbf{RC}(n-1)$, as discussed in Section 2.4.3).
   b) If the number of available $\mathbf{P}^{(g)}(n)$ is greater than the $\mathbf{D}^{+}(n)[s_k, d]$ entry (i.e., more resources are available than those required), randomly select the required number; otherwise select all of them.
4. Repeat steps 1-3 for all columns of $\mathbf{D}^{+}(n)$.

Finally, if any traffic remained in $\mathbf{D}^{+}(n)$ and not all the $I \cdot T$ permutations are utilized, then the algorithm performs a final round where it repeats steps 1-4, with the only difference being that new permutations are considered to be initially available to all connections.

2.5 Architecture-Related Constraint

The resource allocation problem at hand is quite similar to scheduling problems for TDM satellite or ATM crossbar switches (19, 29, 40, 68, 84). Scheduling constraints SC1 and SC2 are common, but constraint SC3 (Table 2.1) is new and is a result of specific architecture choices, and particularly of using static routed (C)AWGs instead of reconfigurable components. This design choice, which was decided to keep the cost and routing complexity low, results in a reference (NEPHELE) DCN (when seen as a huge switch connecting ToRs) losing its nonblocking character even for $I = S$. In the previous section, we described algorithms that operate without taking into account SC3, whose effect is studied here. To evaluate the performance under the additional constraint SC3, we extended the incremental greedy heuristic Section 2.4.3) to account for SC3. The algorithm to be described is referred to as the *ring-segment greedy heuristic*.

To be more specific, consider a transmission from source ToR $s = (w_s, p_s)$ to destination ToR $d = (w_d, p_d)$ at generic slot g (timeslot t over optical plane i), where $p_s < p_d$ without loss of generality. Such a communication is represented in the schedule by $\mathbf{P}^{(g)}(n)[s, d] = 1$. Under the baseline architecture of Section 2.2 that uses $W \times R$ CAWGs at the input of the rings, this communication uses wavelength w_d and ring $r_{sd} = [(w_s + w_d - 1) \pmod R]$, according to Eq. (2.1). So, the communication from s to d captures the ring-wavelength resource, indexed

$$\ell_{sd} = \left[(w_s + w_d - 1) \pmod R\right] \cdot W + w_d \tag{2.11}$$

Resource ℓ_{sd} is actually captured only for the segment of the ring that is between pods p_s and p_d and can be used by other connections if they use non-overlapping segments of the ring. SC3 constrains that $s \to d$ communication cannot take place simultaneously with communication from $s' \to d' \implies (w_{s'}, p_{s'}) \to (w_{d'}, p_{d'})$, with $p_s < p_{s'} < p_d$ or $p_s < p_{d'} < p_d$, $w_{d'} = w_d$ and $(w_{s'} - w_s) \pmod R = 0$ (see Table 2.1).

The ring-segment greedy heuristic algorithm keeps track of the utilization of the ring-wavelength resources and the specific ring segments utilized. To accommodate the communication from s to d at generic slot g, we need to check whether ring-wavelength resource ℓ_{sd} is used be tween pods p_s and p_d. If it is not used, we reserve it to block any future conflicting communication. The data structure records for each generic slot $g = 1, 2, \ldots, I \cdot T$, the ring-wavelength resource $\ell = 1, 2, \ldots, R \cdot W$, and the specific ring segment it uses (P ring segments in the worst case), resulting in size $O(P \cdot R \cdot W \cdot I \cdot T)$. This data structure can be similar to $\mathbf{TC}(n)$ and $\mathbf{RC}(n)$ used to keep track of SC1 and SC2 (Section 2.4.3), which, however, are of size $O(P \cdot W \cdot I \cdot T)$. Specifically, line 9 of the pseudocode of Algorithm 1, should also search for maximum P ring segments, which increases the complexity.

The worst case traffic pattern is obtained when we have the maximum number of conflicting communication pairs defined by SC3, and all of them carry maximum traffic. Regarding the constraint on the overlapping of ring segments, there are P such conflicting (s, d) pairs for unidirectional traffic (p_1 to p_P, p_2 to $p_1, \ldots, p_{P-1}$ to p_P, p_P to p_{P-1}), and since they are in different pods they can have maximum traffic equal to $\hat{\mathbf{Q}}(n)[s, d] = S \cdot T$. In this case, we require $I = P \cdot S$ planes to fully serve the worst case traffic. Such worst case traffic is, of course, highly improbable to occur. Still, our simulations show that the throughput is affected even in the average case when considering SC3, while the execution time increases, since we need to account for the ring segment utilization.

We developed two solutions to address this problem: the first extends the incremental greedy algorithm of Section 2.4.3, considering in a more coarse way the utilization of the ring-wavelength resources, while the second relies on a variation of the architecture that uses spectrum-shifted optical planes.

2.5.1 Full-Ring Greedy Heuristic

In the first solution, called the full-ring greedy heuristic algorithm, communication from s to d is taken to occupy the entire ring-wavelength resource ℓ_{sd}, i.e., the whole ring and not only the segment between pods p_s and p_d. This reduces the size of the data structure needed to $O(R \cdot W \cdot I \cdot T)$ and improves the execution time over the ring-segment greedy heuristic discussed above, sacrificing somewhat the (already lower) throughput performance.

2.5.2 Spectrum-Shifted Optical Planes

A problem of the baseline architecture is that, if two communicating source-destination pairs, (s, d) and (s', d'), conflict over an optical plane, by using the same ring-wavelength resource $\ell_{sd} = \ell_{s'd'} = \ell$, they will use the same resource ℓ and conflict over all planes. This problem affects all planes, so we have available only the time domain (T) to resolve such conflicts, as opposed to having both the plane and time dimensions (all $I \cdot T$ generic slots), resulting in lower performance. To address this, we developed an architecture variation where the optical planes are *spectrum shifted*. To be more specific, in the architecture of Figure 2.1, traffic for destination ToR $d = (w_d, p_d)$ always uses wavelength w_d. The main idea of spectrum-shifted optical planes is to make the ring-wavelength in Eq. (2.11) depend on plane i and on other source/destination location parameters. This proposed variation uses the desired passive components, i.e., (C)AWGs, instead of replacing them by reconfigurable ones that would significantly increase the cost, due to their high radix.

The goal is to design *all-pair conflict-free optical planes*, so that ToR pairs conflicting on some optical plane do not conflict on another one. There are various ways to achieve that, such as permuting the rings between pods, or varying the CAWG routing function by changing the way CAWGs are coupled/added to the rings. One such efficient solution is to replace the $1 \times W$ AWG connected to the drop ports of the WSSs with an $P \times W$ CAWG connected as follows: We connect the drop ports of all the WSSs of plane i and pod p through the $R \times 1$ power combiner to the input port $z(i,p), 1 \leq z(i,p) \leq P$ of the $P \times W$ CAWG. The W output ports of the $P \times W$ CAWG are connected to the ToRs as before. We make the wavelength $w_{sd}(i)$, used for communication between source $s = (w_s, p_s)$ and destination $d = (w_d, p_d)$ over the plane i depend on s, d, and i, as opposed to the baseline architecture where this was fixed and equal to w_d. Considering the routing function of the CAWG, $w_{sd}(i)$ should satisfy the following condition in order to reach the desired destination:

$$(w_{sd}(i) + z(i, p_d) - 1) \pmod{W} = w_d \tag{2.12}$$

where w_d in this equation indicates only the location of the destination ToR in the related pod (and not, as previously, the receiving wavelength), and $z(i, p_d)$ is the input port of the CAWG. The routing function of the CAWG that adds the traffic to the rings at the source gives the ring used:

$$r_{sd}(i) = (w_s + w_{sd}(i) - 1) \pmod{R}. \tag{2.13}$$

Then, the ring-wavelenth resource of plane i that is used is

$$\ell_{sd}(i) = \left[(w_s + w_{sd}(i) - 1) \pmod{R}\right] \cdot W + w_{sd}(i). \tag{2.14}$$

Consider now another ToR pair communication $s' \to d' \implies (w_{s'}, p_{s'}) \to (w_{d'}, p_{d'})$ on the same plane i. To create conflict, this communication has to use the same wavelength and the same ring with the $s \to d$ communication, i.e.,

$$\begin{aligned} w_{sd}(i) &= w_{s'd'}(i) \wedge (w_s + w_{sd}(i)) \pmod{R} \\ &= (w_{s'} + w_{s'd'}(i)) \pmod{R} \end{aligned} \tag{2.15}$$

or, equivalently,

$$z(i, p_d) - z(i, p_{d'}) = (w_d - w_{d'}) \pmod{W} \wedge (w_{s'} = w_s) \pmod{R} \tag{2.16}$$

Our goal is to avoid pairs $s \to d$ and $s' \to d'$ to conflict in any other plane. This can be satisfied if $|z(i, p_d) - z(i, p_{d'})| \neq |z(i', p_d) - z(i', p_{d'})|$, for all $1 \leq i' \leq I, i' \neq i$. Generally,

we want that to hold for any conflicting pair of any plane, i.e., we need the following to hold for all $i, (i' \neq i)$, all p_d, $p_{d'}$:

$$|z(i, p_d) - z(i, p_{d'})| \neq |z(i', p_d) - z(i', p_{d'})|. \tag{2.17}$$

Remember that $1 \leq z(i, p) \leq P$, since $z(i, p)$ corresponds to the input port of the $P \times W$ CAWG that the WSSs of pod p at plane i are connected. For a prime number of pods P, one choice (along with others) that satisfies Eq. (2.16) is

$$z(i, p) = [1 + (p - 1) \cdot (i - 1)] \pmod{P}. \tag{2.18}$$

For prime P, with the above function, we construct P all-pair conflict-free planes. The number of planes I required to serve any pattern is then $I \geq P$. To see this, assume that we have several conflicting pairs on a plane (P is the maximum number of pairs, as discussed previously), and each requires the full capacity (all the timeslots) of the plane. This plane can serve any of those, but the remaining pairs conflicting on that plane are not conflicting on the other $I-1$ planes. Thus, if $I \geq P$ (which also holds for the reference NEPHELE architecture – Table 2.1), conflicts can be solved using the plane dimension in addition to the timeslot dimension. In that case, the entire DCN is actually a *nonblocking time-wavelength-space switch.*

If P is not prime (in the reference $P = 20$), the above function constructs all-pair conflict-free planes equal to the smallest divisor (=2 for the reference architecture). However, even in this case, the conflicts are reduced substantially. The average performance improves when the number of conflicting pairs among the planes is small, and the proposed solution reduces this number. All-pair conflict-free planes mean that this number is zero, which results in the best worst case and average performance. We rely on simulations to evaluate the performance of our solution for average traffic.

The extensions needed in the scheduling algorithm to account for spectrum-shifted planes are straightforward, and require the calculation of the wavelength based on the source, destination, and plane. This can be done with pre-calculated tables and does not affect the complexity. We also need to use either the ring-segment or the full-ring heuristic algorithm to keep track of ring-wavelength resource utilization. We decided to use the faster full-ring greedy heuristic in the performance evaluation section.

2.6 Performance Evaluation

2.6.1 Evaluation Without Architecture Constraint SC3

The proposed incremental scheduling algorithms were evaluated via simulations for various traffic scenarios. We assumed a DCN of the reference architecture, with $W = 80$ racks/pod, $P = 20$ pods, $S = 20$ server ports/rack, and $I = S = 20$ optical planes (see Table 2.1), and set $T = 80$ timeslots. We used a custom traffic matrix generator where we could control the following parameters (4):

1. the average network load $\rho(\mathbf{A})$, defined from Eq. (2.4) as the ratio of the total traffic over the total capacity. The individual ToR loads $\rho_{sd}(\mathbf{A})$ were generated as independent Gaussian random variables, assuming that a ToR aggregates numerous TCP/UDP flows. The distribution mean was set equal to the desired load, while its variance was correlated to the load dynamicity $\rho(|\mathbf{D}_\mathrm{A}|)$;

2. the load dynamicity $\rho(|\mathbf{D}_\mathrm{A}|)$, defined as the average change in traffic between successive periods;

3. the connection density $\delta(\mathbf{A})$, defined from Eq. (2.6). Low connection density corresponds to a few destinations per source, thus an increased number of traffic hotspots. To accommodate the description of traffic patterns of previous works (65), where ToRs systematically prefer to communicate with peers in specific pods, or even the same pod, we further distinguished between intra-POD density $\delta_{\mathrm{in}}(\mathbf{A})$ and inter-POD density $\delta_{\mathrm{out}}(\mathbf{A})$. A *locality* parameter is then defined as the traffic percentage that is destined within the same pod over the total load:

$$\ell = \frac{\delta_{\mathrm{in}} \cdot W}{\delta_{\mathrm{in}} \cdot W + \delta_{\mathrm{out}} \cdot W \cdot (P-1)},$$

   given that the local POD comprises W ToRs out of $W \cdot P$ that are available in total; and

4. the locality dynamicity $\delta(|\mathbf{D}_\mathrm{A}|)$, defined as the average number of connections that change from active to inactive and vice versa at each period. Traffic exhibits locality persistency (Eq. (2.7)) when $\delta(|\mathbf{D}_\mathrm{A}|)$ is low.

To evaluate the proposed algorithms, we developed a simulator in MATLAB (38). For each simulation instance, we chose to vary one parameter, while the rest of the parameters were set to their default values (Table 2.3). To focus on the performance of the scheduling

Table 2.3 Networking parameters.

Parameter	Symbol	Value	Default
Network load	$\rho(\mathbf{A})$	0.1-0.9	–
Intra-POD connection density	$\delta_{\text{in}}(\mathbf{A})$	100%,25%,2.5%	25%
Inter-POD connection density	$\delta_{\text{out}}(\mathbf{A})$	25%,2.5%,0.5%	2.5%
Load dynamicity	$\rho(\lvert\mathbf{D}_{\text{A}}\rvert)$	10%,1%,0.1%	1%
Locality dynamicity	$\delta(\lvert\mathbf{D}_{\text{A}}\rvert)$	10%,1%,0.1%	1%

algorithms, we assumed a resource cycle with $C = 1$, which corresponds to the schedule being calculated within a Data period. We also assumed the reference case where the estimated queue matrix on which the schedule is calculated based on the arrivals: $\hat{\mathbf{Q}}(n+1) = \mathbf{A}(n-C) + \hat{\mathbf{Q}}(n) - \mathbf{S}(n)$. As discussed in Section 2.4.3, this ensures that the persistency property of $\mathbf{A}$ is also true for the estimated queue matrix $\hat{\mathbf{Q}}$.

For each parameter set, we measured a) the additional average queuing latency, i.e., the average number of periods a packet remains buffered in addition to the $C = 1$ period that it takes for the schedule to be calculated, to focus on the efficiency of the algorithm and not of the whole control cycle, and b) the scheduling algorithm's execution time (s) against the network load. We also measured the *maximum network throughput*, defined as the maximum offered load at which the queues and the latency are finite. Thus, the maximum throughput indicates the load that can be transferred by the network under stable operation. Note that maximum throughput is identified in the latency/load graphs as the load at which the latency becomes (asymptotically) infinite.

Queuing Latency

Initially, we present the results on the latency. In the first set of simulations, the examined parameter is intra-POD density, which is set to 100% for the results of Figure 2.5a and to 2.5% for Figure 2.5b; the other parameters were set to their default values (Table 2.3). Figure 2.5a shows that the sublinear greedy heuristic clearly underperforms, as expected, the other two algorithms, resulting in average latency that increases at load 0.7 and becomes (asymptotically) infinite at load 0.8 (= maximum network throughput). The linear greedy heuristic comes next, followed by the randomized heuristic with slightly better performance. In the results of the second set of simulations, shown in Figure 2.5b, the density of intra-POD connections is set very low to 2.5%. The queuing latency of all three algorithms start to increase at load around 0.7. The increase is steeper for the sublinear greedy heuristic, followed by the linear greedy, and finally by the randomized heuristic. The latter two algorithms have very similar performance regarding latency and stability.

Figure 2.5 Average queuing latency resulting from the examined scheduling algorithms, measured in Data periods additional to the control cycle, for intra-pod density (a) $\delta_{\text{in}} = 100\%$ (locality $\ell = 68\%$) and (b) $\delta_{\text{in}} = 2.5\%$ (locality $\ell = 5\%$).

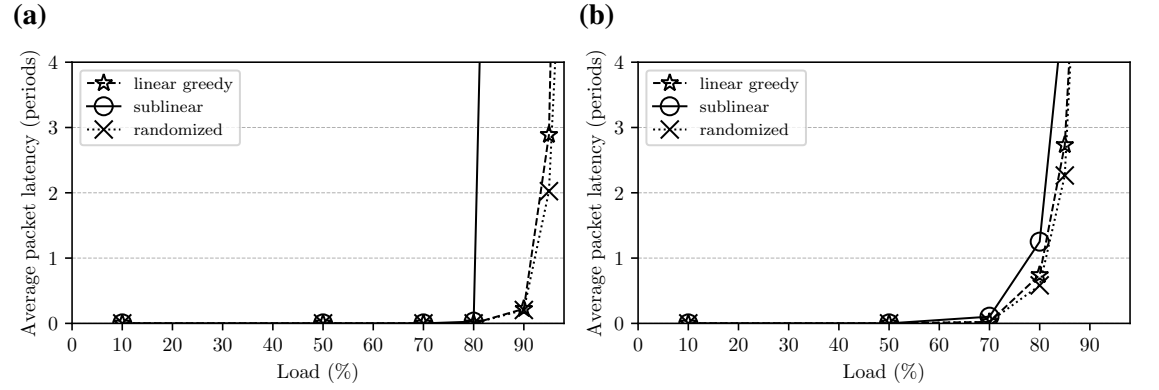
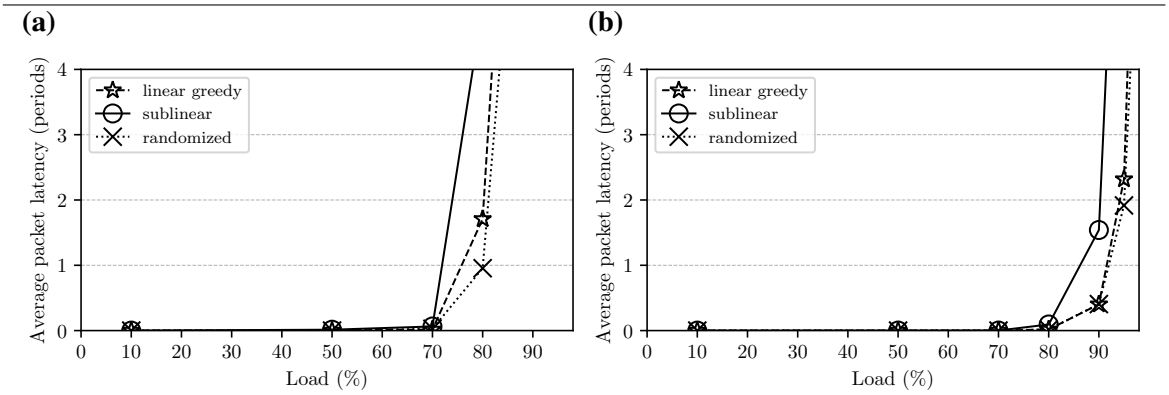


Figure 2.6 Average queuing latency resulting from the examined scheduling algorithms, measured in Data periods additional to the control cycle, for locality dynamicity (a) $\delta(|\mathbf{D}_{\text{A}}|) = 0.1\%$ and (b) $\delta(|\mathbf{D}_{\text{A}}|) = 10\%$.



Locality considerably impacts the performance. The greedy and the random algorithms are more efficient when the matrix is concentrated in small blocks (δ_{in} high, heavy intra-POD traffic – Figure 2.5a) than when it is spread out (Figure 2.5b). In contrast, the sublinear algorithm underperforms when locality is high; introducing several dummy DUs in a small block increases the column sum more than when traffic and the locations of the dummy DUs are spread out.

We next examine the effect of the locality dynamicity parameter $\delta(|\mathbf{D}_{\text{A}}|)$. When $\delta(|\mathbf{D}_{\text{A}}|) = 0.1\%$ (Figure 2.6a), all three heuristic algorithms start to induce high latency at network load of about 0.7. As in the previous cases, the queuing latency increase with network load is steeper for the case of the sublinear heuristic, followed by the linear greedy, and then by the randomized heuristic. This is more clear at load 0.8, where the sublinear greedy heuristic is

already in the unstable region, while the linear greedy and the randomized heuristic remain stable until load 0.85.

When the locality dynamicity parameter $\delta(|\mathbf{D}_A|) = 10\%$ (Figure 2.6b), all three algorithms improve their results by increasing their maximum throughput (latency asymptote moves to the right). Higher dynamicity reduces the persistency of bad scheduling matrices, improving the performance, but as expected, has negative effects on execution times, as will be discussed in the following.

Scheduling Algorithms Execution Times

Next, we present results on the execution times of the considered algorithms. We provide four plots for the same parameters examined in Section 2.6.1.

As shown in Figure 2.7a, the algorithms' performance in order of increasing execution times is randomized, linear greedy, and sublinear greedy heuristic. As expected, the average execution times increase with the load. At load 0.8, the randomized heuristic needs an average of 1.5 s to complete. Next comes the linear greedy heuristic with an execution time (at 0.8 load) of about 0.7 s, and last comes the sublinear greedy heuristic with about 0.5 s. These results were expected from the theoretical complexity analysis given in Section 2.4. The relative order of the algorithms with respect to their execution times remains the same when intra-POD connection density is set to 2.5% (Figure 2.7b). The decrease in the execution times for low intra-POD density is due to the fewer connections, each of higher load, which reduces the complexity of all three algorithms. The execution times for different values of locality dynamicity parameter $\delta(|\mathbf{D}_A|)$ are depicted in Figure 2.8. As expected, by complexity analysis, execution time increases as load and locality dynamicity $\delta(|\mathbf{D}_A|)$ increases.

Maximum Network Throughput

We now focus on the maximum network throughput achieved by the scheduling algorithms, defined as the load at which the queues and the latency become (asymptotically) infinite and the system becomes unstable. The throughput is examined with respect to two parameters that were not discussed above: (i) the inter-POD connection density δ_{out} and (ii) the load dynamicity $\delta(\mathbf{D}_A)$. The results are shown in Table 2.4. We see that the impact of inter-POD connection density δ_{out} is quite significant, since for dense traffic ($\delta_{\text{out}} = 50\%$), the throughput reaches about 0.97, while for sparse traffic, it drops to 0.85 at most. The reason is similar to the one discussed for the role of intra-POD density. It should be noted that, for dense inter-POD connections ($\delta_{\text{out}} = 0.5\%$), the sublinear greedy heuristic is unstable

Figure 2.7 Execution times of the algorithms for intra-pod density (a) $\delta_{\text{in}} = 100\%$ (locality $\ell = 68\%$) and (b) $\delta_{\text{in}} = 2.5\%$ (locality $\ell = 5\%$).

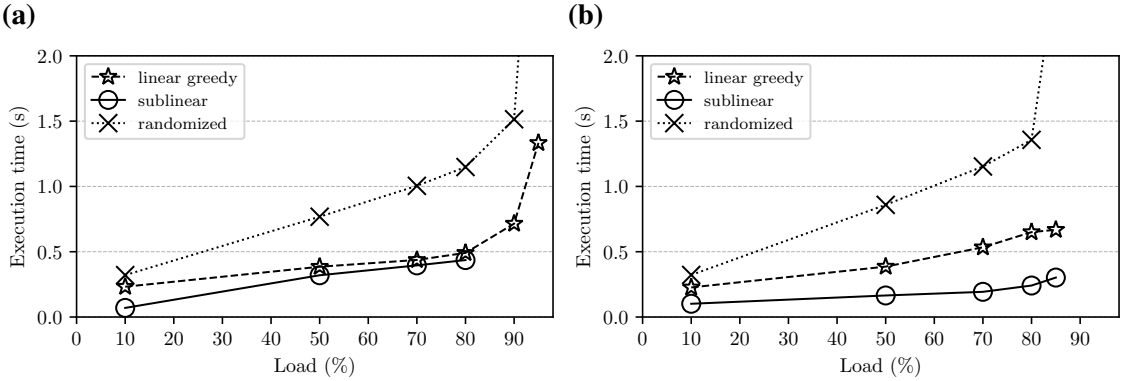
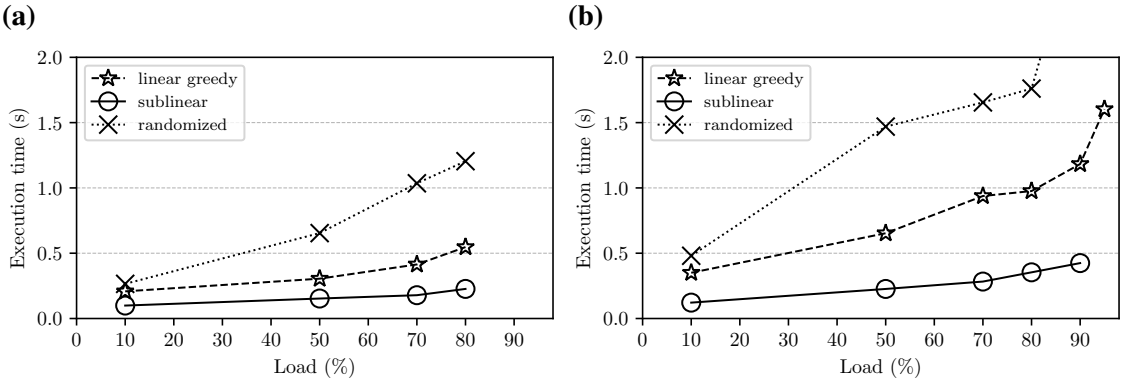


Figure 2.8 Execution times of the algorithms considered for locality dynamicity (a) $\delta(|\mathbf{D}_{\text{A}}|) = 0.1\%$ and (b) $\delta(|\mathbf{D}_{\text{A}}|) = 10\%$.



even at low traffic loads, since it wastes too much capacity. This should be expected, as small and spread demands result in many entries that create many dummy DUs, thus wasting network capacity. Regarding load dynamicity, we consider the cases $\rho(|\mathbf{D}_{\text{A}}|) = 0.1\%$ and $\rho(|\mathbf{D}_{\text{A}}|) = 10\%$. We observe that this parameter does not affect substantially the throughput, nor the execution time. The throughput performance of all the algorithms was similar, with the sublinear greedy heuristic being slightly worse and faster (lower than 0.4 s in almost all cases).

2.6.2 Evaluating the Effect of the SC3 Constraint

We evaluated the performance of the reference DCN under the architecture constraint SC3 and also for the architecture variation that uses the spectrum-shifted planes. In particular, we assessed the performance for

a) reference architecture/greedy (no SC3),

Table 2.4 Maximum throughput of algorithms considered as a function of inter-POD connection density δ_{out} and load dynamicity $\rho(|\mathbf{D}_{\text{A}}|)$.

Parameter	Symbol	Value	Linear	Randomized	Sublinear
Intra-POD connection density	δ_{out}	50%	0.97	0.97	0.4
		0.5%	0.85	0.85	0.82
Load dynamicity	$\rho(\|\mathbf{D}_{\text{A}}\|)$	0.1%	0.92	0.93	0.9
		10%	0.88	0.88	0.87

b) reference architecture/segment-ring greedy,

c) reference architecture/full-ring greedy, and

d) spectrum-shifted planes/segment-ring greedy.

In all examined cases, the number of planes was the same ($I = 20$). Case (a) was examined in the previous subsections and is used here as a reference. The network of case (a) can achieve maximum throughput; that is, it can accommodate any traffic if an optimal algorithm is used. The network of cases (b) and (c) has worst-case traffic that requires more (20 times) planes, while case (d) also requires more planes than the I available, but lower than those of cases (b) and (c). The probability of generating the worst-case traffic is extremely low, but cases (b) and (c) have several traffic instances that require more than I planes, while for case (d) this probability is low. Note, however, that we use a heuristic (incremental greedy) and thus blocking is expected even for case (a).

Figure 2.9a shows the latency for density between pods $\delta_{\text{out}} = 50\%$, corresponding to $\ell = 2.5\%$ locality (default $\delta_{\text{in}} = 25\%$). Such a low locality results in heavy utilization of the inter-pod WDM rings and creates SC3 conflicts. We observe that the asymptotic throughput of the reference architecture/segment-ring greedy reduces to 0.8 compared to 0.9 of the reference architecture/greedy, where SC3 is neglected. The reference architecture/full-ring greedy has even lower throughput, measured to be 0.7, but exhibits lower execution times (see the following). The spectrum-shifted planes architecture resolves conflicts in one plane by serving in another plane, and thus improves the throughput. The achieved throughput was 0.85, which is close to the case where SC3 is neglected, as shown by the reference architecture/greedy (no SC3).

As locality increases, inter-pod traffic decreases, and eventually, at high locality, the performance of all algorithms converges. For example, in Figure 2.9b, where the density between pods is $\delta_{\text{out}} = 0.5\%$ (or $\ell = 70\%$ locality), we observe that the reference architecture/greedy (no SC3) achieves throughput close to 0.95, very close to the rest of the cases examined. Note that, according to (65), locality is very high in a Facebook DC, higher than

Figure 2.9 Latency (in periods) as a function of load for density between pods (a) $\delta_{\text{out}} = 50\%$ (locality $\ell = 2.5\%$) and (b) $\delta_{\text{out}} = 0.5\%$ (locality $\ell = 70\%$).

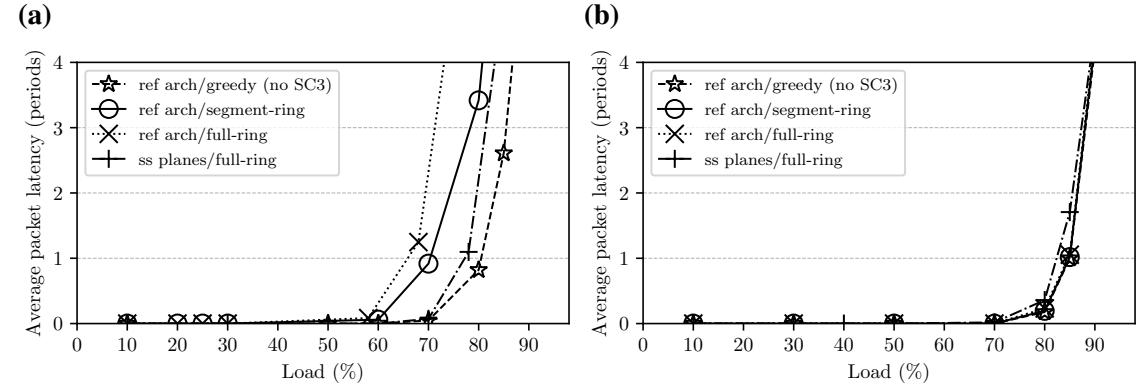
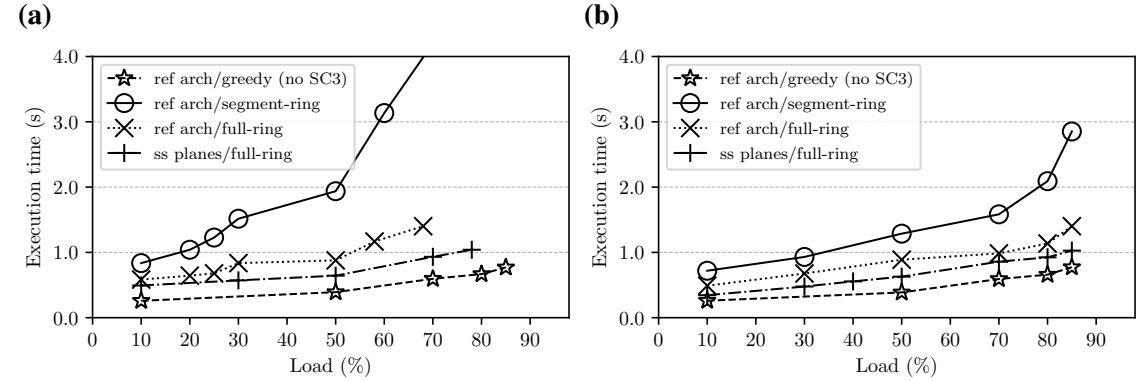


Figure 2.10 Execution time as a function of load for density between pods (a) $\delta_{\text{out}} = 50\%$ (locality $\ell = 2.5\%$) and (b) $\delta_{\text{out}} = 0.5\%$ (locality $\ell = 70\%$).



50% for typical DC applications, such as web and map-reduce. Figure 2.10 shows the related execution times. We observe that the reference architecture/segment-ring greedy has the highest running time, well above 1 s. Keeping track of ring segments yields higher complexity. Execution time is reduced in the reference architecture/full-ring greedy (but it wastes resources – has lower throughput, as seen in Figure 2.9). The spectrum-shifted planes/full-ring greedy case has quite low execution time, similar to the reference architecture/full-ring greedy. Thus, it combines the execution time benefits of the full-ring algorithm while achieving throughput close to the case without SC3 (by reducing the conflicting sets). As locality increases, the execution times of the reference architecture/full-ring and spectrum-shifted planes/full-ring converge to that of the reference architecture/greedy (no SC3).

2.7 Realistic Evaluation of Control Plane and Architecture Enhancements

2.7.1 Realistic Traffic Simulations Setup

To evaluate the performance of the reference DC's control cycle, we developed a packet level network simulator. The simulator is an extension of OMNET++ 4.3.1 with INET 2.4.0, a framework that contains implementations for various real-life network components and protocols. We evaluated the network performance using an application that simulates MapReduce, which was implemented by Mellanox.

In our simulation model, we consider that the control plane delay, which includes the time to gather monitoring information (if we operate the network in feedback based, would be zero in application-aware mode), to calculate the schedule (which as previously discussed is fast, within 1 Data period (61)) and to distribute the schedule to the data plane devices, is described through the parameter C. This in turn defines the number of multiple identical (virtual) schedulers that work in parallel. We also assume that each parallel scheduler knows the C previous schedules (feasible, as the schedule is computed in 1 Data period).

In the simulated network, we run a number of MapReduce jobs simultaneously. Each MapReduce job requires a number of worker nodes: *mappers*, *reducers* and *storage servers* and runs for a number of iterations. The communication pattern for each particular MapReduce job, regarding the server where each worker node resides, the size of the MapReduce data produced in each phase, the number of MapReduce iterations and the computational delay for *map* and *reduce* operations, are described using appropriate semantics in an input file. In the simulations, the assignment of the worker nodes to the servers was random. This means that a server could host simultaneously multiple types of worker nodes for the same or different jobs.

The communication between the worker nodes is achieved via Ethernet packets over TCP/IP. We assumed full-duplex 10G Ethernet from a server to the corresponding ToR switch. For the ToR to ToR communication, we rely on the TDMA operation. The Ethernet packets are stored in Virtual Output Queues (VOQs) and served in slots according to the computed schedules.

We study the impact of various parameters, such as the Control cycle delay C, the number of MapReduce jobs, or the cluster size ($P \cdot W$), on the throughput, in terms of total makespan. The makespan is defined as the time it takes for all MapReduce jobs to finish. Table 2.5 summarizes the DCN parameters, as well as the TCP-related parameters. Note that a target for the DCN would be to have 1600 racks with 20 servers each, while each timeslot (of

Table 2.5 Simulation parameters.

Parameter	Value
Number of servers in each rack (S)	2
Number of planes (I)	2
Link capacity per plane (each direction)	10 Gbps
Timeslot duration	200 μs
Maximum segment size (MSS)	625 bytes
TCP window size	65000 bytes
Storage server | Mapper | Reducer output	5 | 10 | 5 Mbytes
Mapper processing time	25 μs
Reducer processing time	20 μs
Number of MapReduce iterations	3

duration 200 μs) aggregates the traffic of all servers residing in a rack. Since it is not possible to simulate a fully-fledged DCN, but only smaller clusters with fewer servers per rack, the parameters are also scaled down accordingly. We assumed $I = 2$ optical planes, and the scheduling period T took values so that the generalized slots/resources equals to the number of racks ($T \cdot I = P \cdot W$).

The key parameters that we examine are the Control cycle delay C, the number of MapReduce jobs that run simultaneously in the cluster and the number of cluster's racks; their default values are 4, 5 and 8, respectively. In all scenarios, the ratios of the MapReduce worker nodes types remained the same: the number of mappers equals to half, while the number of reducers and storage servers equals to a quarter of the available servers. A parallel (dual) network (utilizing 1 Gbps capacity) is also used to route the TCP ACKs. We examine three queue matrix estimation policies:

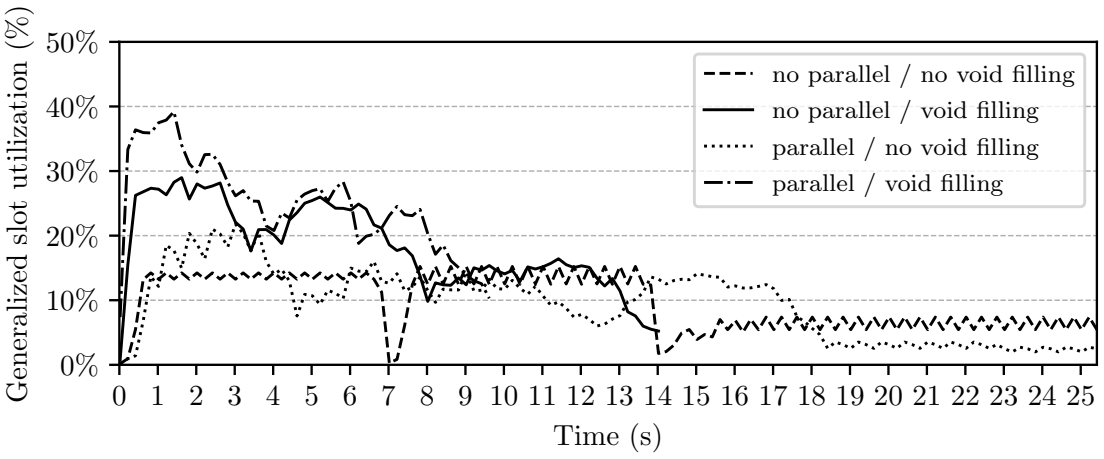
Round-robin policy: An policy assuming static uniform traffic under which no traffic identification mode (monitoring or application awareness) is assumed, and the resource allocation is evenly distributed among the ToR pairs (round-robin scheduling)

Follow the arrivals policy: A policy assuming that $\hat{\mathbf{Q}}(n)$ (described in sections 2.3 and 2.4) is computed based on the most recent known arrivals $\mathbf{A}(n - C - 1)$.

Prediction policy: A simplistic prediction mechanism that assumes that the arrivals for the next C Data periods will be equal to the latest $\mathbf{A}(n)$.

It then virtually applies the latest C known schedules and computes an estimation for the remainder in the queues when the schedule will be applied (after C Data periods). The above queue estimation policies are combined with the incremental scheduling algorithm, which

Figure 2.11 Effect of the parallel network and randomized void filling heuristic on slot utilization.



is extended with a greedy randomized void filling heuristic. Void filling is used to fill the unallocated slots left empty by the scheduling algorithm. In particular, a randomized greedy heuristic greedily computes a set of matchings in order to fill the free slots in a uniform way, taking into account the previously allocated slots and the transmission constraints that they yield.

2.7.2 Simulation Experiments

We initially examine the effect of utilizing i) a parallel packet switched network over which we sent TCP ACK packets and ii) a randomized void filling heuristic to fill the empty slots/permutations of the schedules on slot (network capacity) utilization over time. As it can be observed in Figure 2.11, both the effect of the parallel network and the randomized void filling heuristic is quite significant. Since, TCP features congestion control, the TCP window limits the traffic load the servers transmit. This has a major impact to the overall slot utilization and thus to the throughput and the makespan of the network.

These two techniques improve the TCP window pipelining, resulting in improved slot utilization and reduced makespan. In particular, we observed a reduction of the makespan for the 4 MapReduce jobs from 27.4 s in the case of *no parallel/no void filling* to 27.2 s in the case of *parallel/no void filling* and to 14 s in the case of *no parallel/void filling*. The combination of *parallel/void filling* achieves a substantially lower makespan of 10.3 s. In the following, we will assume that the DCN uses both *parallel/void filling*.

We now examine the effect of the control delay C which was varied from 0, 5, 10, 20, 50 to 200 Data periods. As it is shown in Figure 2.12, the makespan for the case of the static round-robin policy remains constant at about 0.36 s, regardless of the Control cycle delay.

Meanwhile, the other two policies seem to perform better for at most 19%, given that they take into account the traffic (monitoring or application awareness) and carry out scheduling based on $\hat{\mathbf{Q}}(n)$ estimates. This performance improvement decreases as the Control cycle delay increases, and eventually in the sample of Control cycles equal to 200 Data periods, it gets worse than the static round-robin for at most 13%. This is expected, since the longer control delay results in an increased chance of the actual traffic at the queues to substantially differ from the calculated schedule. It can also be observed that in small numbers of Control cycles, utilizing prediction also improves the performance. However, this improvement fades out from 20 Control cycles and on.

In the next scenario, we consider the cases where we have 1, 4, 7 and 10 MapReduce jobs simultaneously running on the cluster. It is expected that as the number of jobs increases, the network load increases, but also the traffic dynamicity decreases, given that the assignment of the worker nodes with the servers is done randomly and uniformly. As shown in Figure 2.13, the makespan increases with the job number in all queue matrix estimation policies, since the network load increases. However, especially in the case of 1 job, where only certain parts of the network are utilized in each MapReduce phase, we can see that the static round-robin policy performs much worse than the other two policies for about 32%. This difference is reduced for larger numbers of jobs to at least 16%.

In the last considered scenario, we have different cluster sizes, namely of 4, 8, 16 and 32 racks (8, 16, 32 and 64 servers, respectively). Figure 2.14 shows the performance of the three queue matrix estimation policies. In particular, we can observe that the policies that take into account the traffic have a much better performance than the static round-robin that ranges between 12-48% and increases with the increase of the cluster size.

Figure 2.12 Effect of the Control cycle (in Data periods) on makespan.

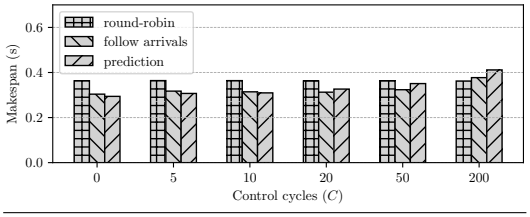


Figure 2.13 Effect of the number of MapReduce jobs on makespan.

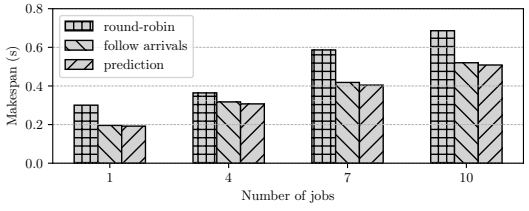
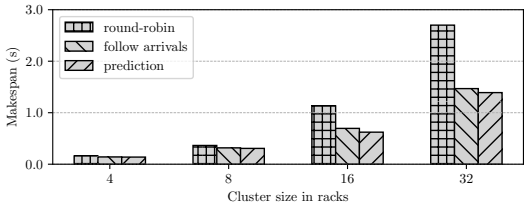


Figure 2.14 Effect of the cluster size on makespan.



2.8 Conclusion

We proposed and evaluated a set of scheduling algorithms specifically designed for an optical DCN utilizing fiber rings and wavelength-selective switches, which allows dynamic allocation of resources according to traffic requirements. To avoid contention, a centralized allocation process enforces three scheduling constraints. We described in detail the DC control cycle, outlined its requirements, and presented an algorithm to optimally allocate resources. We also proposed three incremental heuristic scheduling algorithms that reduce the execution times of allocation, and evaluated their performance through simulations. The randomized and greedy heuristics exhibited normalized throughput higher than 0.85 for all examined traffic scenarios. The execution time of the greedy heuristic was measured in hundreds of milliseconds, while the sublinear greedy heuristic was faster, sacrificing some throughput. The parallel implementations of the proposed algorithms on specialized hardware (field-programmable gate array) to further reduce execution time is ongoing. We also studied the effect on performance of the third scheduling constraint (SC3), which is specific to the reference DC's architecture. To cope with the resulting reduction of throughput and increase of execution time, we proposed an architecture variation that employs spectrum-shifted optical planes and extended the greedy heuristic to function in such a network. Simulations showed that the throughput and execution time performance approaches that of a network without SC3. The proposed incremental heuristic algorithms achieve high throughput and low execution time, asserting the dynamic and efficient operation of DC.

We further examined the control cycle, including the importance of the policy used to obtain good queue matrix estimates that approximate the traffic pattern after the control cycle delay. We conducted simulations using OMNET++ under MapReduce realistic traffic. We examined the effect of utilizing a parallel network for TCP ACKs, and of a void filling heuristic. We observed that both these techniques, improve the makespan. We considered the case of applying a static round-robin policy and two policies that take into account the traffic. We observed that when the control cycle delay is high, a static round-robin policy seems preferable. The policies that take into account the traffic induce a significant improvement to the total makespan that can reach 48% when the short-term load dynamicity is high.

Chapter 3

Fast Optical Datacenter Interconnects with Partial Configurability

3.1 Introduction and Related Work

The widespread availability of cloud applications has allowed billions of users to access software-, platform-, and infrastructure-as-a-service models. These services rely heavily on Data Centers (DCs), which comprise large numbers of interconnected servers. It has been observed that incoming/outgoing (north-south) traffic in DCs is low, while the traffic within a DC (east-west) is high (39), making the DC interconnection networks (DCNs) critical to overall performance. Currently, state-of-the-art DCNs use Fat-Tree topologies to interconnect electronic switches with optical fibers, using electro-opto-electrical transformation at each electronic switch hop (8). However, this approach underutilizes resources, requires numerous cables and switches, suffers from poor scalability and upgradability (lack of transparency), and consumes high levels of energy (65).

The integration of optical switching in DCNs is a pivotal step towards addressing the limitations of Fat-Tree topologies. While optical switches are primarily used for circuit switching in metro and backbone networks, recent research has proposed hybrid electronic/optical switched DCNs as a solution (12, 20, 24, 30, 42, 62, 63, 67, 72, 76, 77). These studies employ optical switches, which transparently redirect light from any port to another. However, their reconfiguration times (milliseconds for high radix and tens of microseconds for low radix switches) present a challenge to their use in DCNs, where rapid switch configuration is essential. Despite this limitation, the potential benefits of integrating optical switches into DCNs are significant, and ongoing research is focused on developing more efficient and sustainable solutions. In 2022, Google announced (64) that Jupiter datacenter network

fabrics will use dynamic topology reconfiguration using Optical Circuit Switching, which have evolved to achieve higher speed, cost reduction, power efficiency, and optimized path lengths.

The first barrier to the adoption of optical switching technologies in DCNs comes from the reconfiguration speed of (full) crossbar optical switches. As the size of DCNs grows, the options of employing a single-stage optical switch diminish and/or the reconfiguration speed is prohibitive high. Conversely, using multi-stage crossbars, built with smaller modules, is the only solution. However, this suffers from high overall switch count and wiring complexity, which can in turn affect the production cost. It also requires tight synchronization and coordinated control of the multiple elements. To address these challenges, researchers have proposed two hybrid DCN architectures: Mordia and CBOSS, which utilize WSS making use of the wavelength domain to reduce the number of required elements and provide low switching times. These proposed solutions operate in a dynamic slotted manner to achieve high connectivity (15, 63). However, the scalability of both Mordia and CBOSS is limited, as they employ a single wavelength division multiplexing (WDM) ring with a capacity that can accommodate only a few racks. In contrast, NEPHELE (12) proposes a distributed crossbar optical network fabric using WSS switches interconnected in several parallel WDM fiber rings. The NEPHELE architecture takes advantage of the use of (relatively) low radix WSS switches, space (multiple rings) and wavelength (WDM) domains to achieve low reconfiguration speed and high throughput. However, the NEPHELE architecture is still not scalable, as discussed in the following section (third barrier).

The second barrier to using all-optical DCNs derives from schedule computation. Allocating optical resources in space (links), time (slots), and/or wavelength (WDM) domains requires high computational complexity, making it challenging to perform optimally or even sub-optimally in real-time. The resource allocation algorithms of Mordia (63) and CBOSS (15) exhibit high computational complexity and do not scale well with large DCs, representing a significant challenge for optimizing these networks. NEPHELE (12) introduced resource allocation algorithms with low complexity for slowly changing traffic patterns that take advantage of previously computed schedules. Efforts to address the computational complexity of centralized scheduling calculations have also been explored, such as the parallel scheduler architecture of (61). However, fast scheduling solves a key part but not the whole problem, as discussed next.

The third barrier pertains to the assumption of centralized control in hybrid electronic/optical DCNs, which typically follows the SDN paradigm (12, 24, 67). In this architecture, a central controller/scheduler gathers all traffic demands and configures the optical switches accordingly. However, when the network is large and the closed-loop DCN control operation

is applied to all network nodes, it is inefficient due to the high latency induced by the control plane for monitoring, schedule calculation, and schedule dissemination. As a result, this purely centralized approach faces limitations in terms of scalability and real-time operation.

In (57) the authors proposed a switch design that utilizes a monolithic gang-switched module called the "selector module" as its fundamental building block. Building on that, the authors of (56) introduced a full DCN architecture design called "RotorNet", utilizing switches constructed exclusively with selector modules, referred to as "Rotor switches". However, RotorNet achieves poor throughput performance, as it spends half of its network capacity for load balancing purposes.

This chapter explores ways to address the limitations of existing optical data center network (DCN) architectures. The approach we take is to design custom "Lean" optical switches, which have two stages of multiple selector modules and are combined with a set of Rotor switches to achieve full network connectivity. Each selector module of the first stage is connected to all selector modules of the second stage, while all selector modules can carry a group of multiple optical signals from different input ports to their corresponding output ports at each switching state. This means that a few selector modules can carry a much higher number of optical signals between DCN nodes, reducing the number of required switching elements compared to a fully configurable network.

Additionally, compared to a fully configurable network, our solution reduces the level of centralized control, enabling the development of algorithms to allocate resources sub- but near-optimally in real-time. However, the configurability of the DCN is determined by the number of switches and the switching states of the Lean switch internal selector modules, and it is lower compared to a fully configurable network. These design parameters also affect the reconfiguration speed of the switches, the algorithmic complexity for the computation of schedules, and the complexity of the control commands. The proposed DCN design is parametric with respect to the number of the grouped signals carried by the same selector modules, allowing for an increase in the network's configurability by adding more Lean and Rotor switches. This may lead to an increase in the number of ports, which in turn increases the available network capacity. WDM can be utilized with the optical signals to further enhance the network's capacity.

The research results of this chapter were partly published in (43). Also, a manuscript was recently submitted to *Optical Switching and Networking* (OSN) and is currently under review.

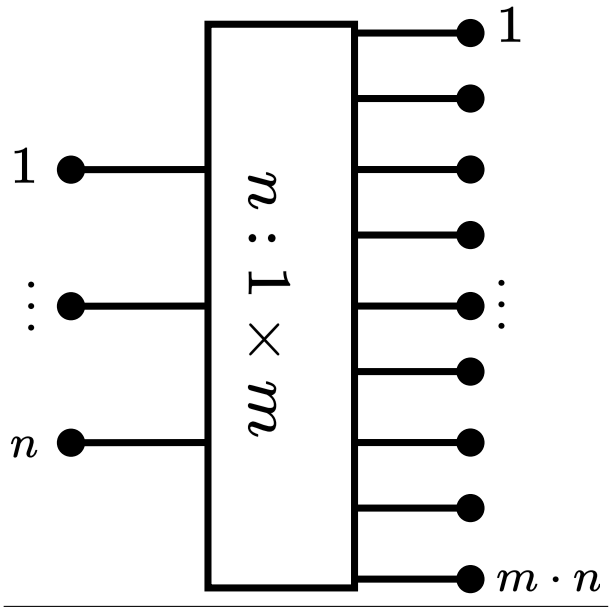
The chapter is structured as follows. In Section 3.2, we present the design specifications of the Rotor and Lean switches, and our proposed partially configurable DCN architecture that uses them. In Section 3.3, we discuss the control plane and its control cycle. Sec-

tion 3.4 presents the problem definition and scheduling policies that take advantage of the limited configurability of the architecture, exhibiting low computational complexity. Additionally, in Section 3.5, we present reference architectures, their corresponding scheduling policies, and compare them in terms of crosspoint complexity and reconfiguration delay. In Section 3.6, we evaluate the achieved throughput and average packet latency of the proposed DCN under various scenarios using the packet simulator OMNET++. Finally, in sections 3.7 and 3.8, we examine enhancements on RotorNet by applying breakout for mitigating latency due to network size expansion and integrating centralized control for partial configurability. We develop a policy that adapts to traffic characteristics without prior knowledge, designed to surpass VLB. We compare it to various other scheduling policies and evaluates it across diverse traffic profiles, though comprehensive simulations.

3.2 A DCN Architecture with Lean Switching Components

In large optical networks connecting hundreds of endpoints (i.e. racks), full-optical operation using crossbar switches is rather unrealistic. Building a large network with a single crossbar switch is constrained by the feasible number of tilting positions of the MEMS, putting a limitation on the radix but also increasing substantially the reconfiguration speed. On the other hand, interconnecting several lower-radix switches, increases the number of elements, fibers and complicates their control. Additionally, in a full-crossbar network (irrespectively of how the crossbar is built) the computational complexity required for scheduling can be challenging to handle in real-time.

Figure 3.1 A $n : 1 \times m$ gang-switched selector module.



The proposed architecture avoids using switches with individually-switched elements, i.e. elements that switch each input signal independently of others, and instead employs a design built with the selector module, a monolithic gang-switched element (57), as its building block. This module uses MEMS beam-steering micromirrors and employs a fixed and small set of switching states and hard-wired interconnection mappings. The selector module comes in two flavors:

$n : 1 \times m$**:** a switching element that simultaneously routes all n inputs to one of its m groups of outputs (each group having n ports, for a total of $m \cdot n$ outputs), and

$n : m \times 1$**:** a switching element that routes one of its m groups of n inputs to its n outputs,

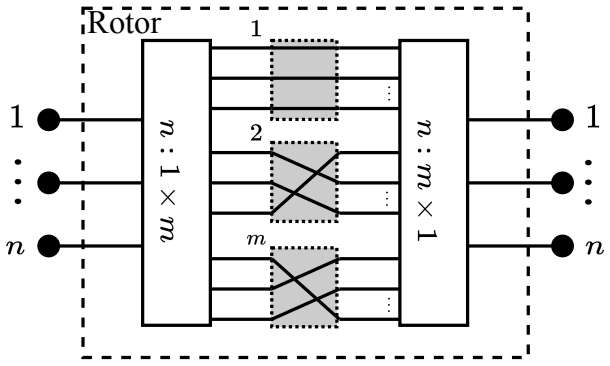
both implementing m states and port mappings.

The proposed architecture uses two types of switches built with selector modules, *Rotor* and *Lean*. Both types have two stages containing one or more selector modules (as shown in Figure 3.1). The selector modules of the first stage are $n : 1 \times m$, while the ones of the second stage are $n : m \times 1$. The number of simultaneously routed signals is described by the design parameter n, which we refer to as the *group factor*.

3.2.1 The Rotor Switches

The Rotor switch (56, 57) is a partially configurable switch that utilizes a single selector module in both the first and second stages. This switch can configure $m = O(n)$ instead of the $n!$ ($m \ll n!$) states, thereby implementing m port mappings of n ports (as shown in Figure 3.2). The switch's various port mappings are implemented through fixed m (hard-wired) shift shuffles, each for a set of n optical fibers, implemented between the two stages.

Figure 3.2 A Rotor switch implementing m port mappings of n ports.



3.2.2 The Lean Switches

The proposed architecture includes a partially configurable switch called the Lean switch (Figure 3.3), which has $m \cdot n$ inputs and outputs and utilizes m selector modules of type $n : 1 \times m$ at the input stage and m selector modules of type $n : m \times 1$ at the output stage. By using these selector modules, the switch is capable of routing a group of n input signals to a group of output ports and can configure $m!$ switching states. Although the Lean switch operates similarly to an $m \times m$ optical crossbar switch, the use of gang-switched (selector) elements allows (57) for simultaneous routing of $m \cdot n$ optical signals, rather than m.

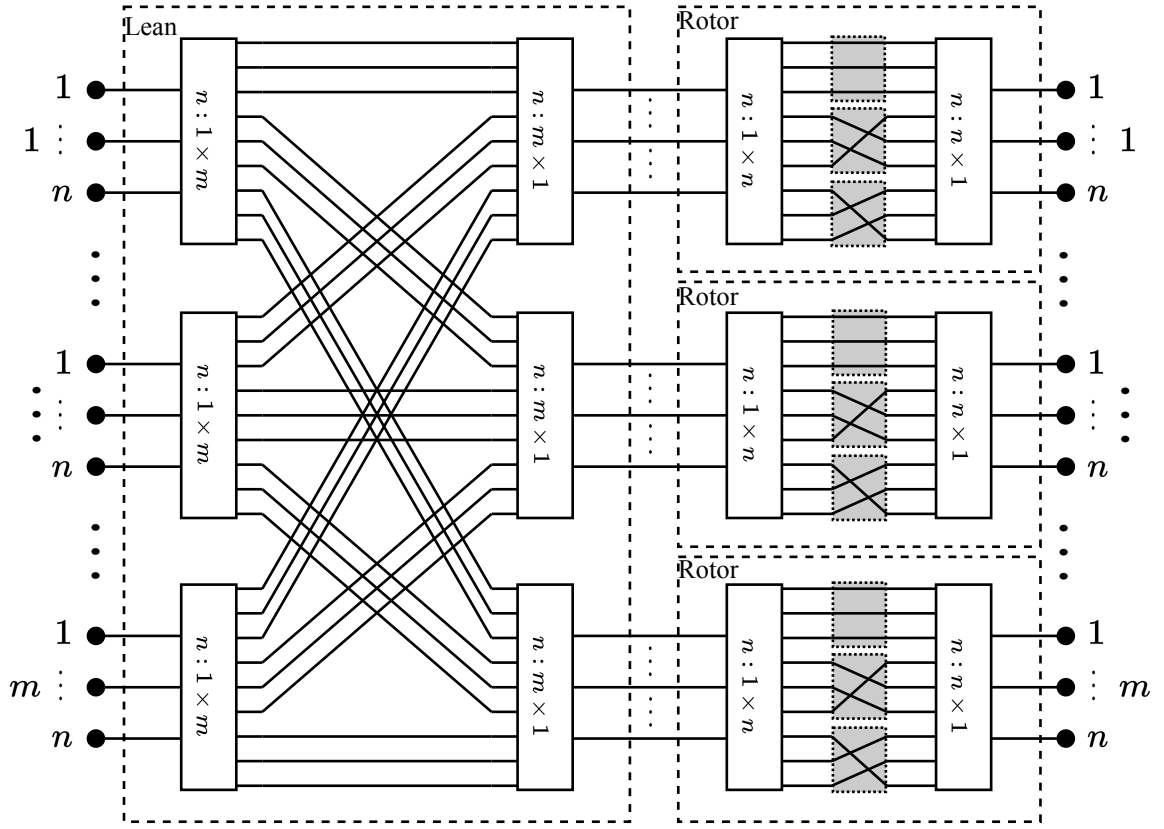
To ensure satisfactory switch functionality, fast reconfiguration speed, and simple scheduling policies, the number of selector modules in a Lean switch and their corresponding supported mappings should be on the order of $O(n)$.

3.2.3 Combining Lean and Rotor Switches for Full Connectivity

While the Lean switch is highly efficient in routing a group of input signals to a group of output ports, it does not provide full connectivity. To achieve configuration with all port mappings, a secondary switching layer is required. This layer uses m Rotor switches, each capable of supporting n port mappings of n ports, as shown in Figure 3.3. We refer to a Lean switch combined with Rotor switches as a *Lean plane*.

With this combination, the architecture offers several advantages over using large Rotor switches alone, as in RotorNet (56). Not only does it increase the network's functionality, but

Figure 3.3 The proposed network of a Lean plane combining a Lean switch with $m \cdot n$ inputs/outputs and a layer of m Rotor switches implementing n port mappings of n ports each.



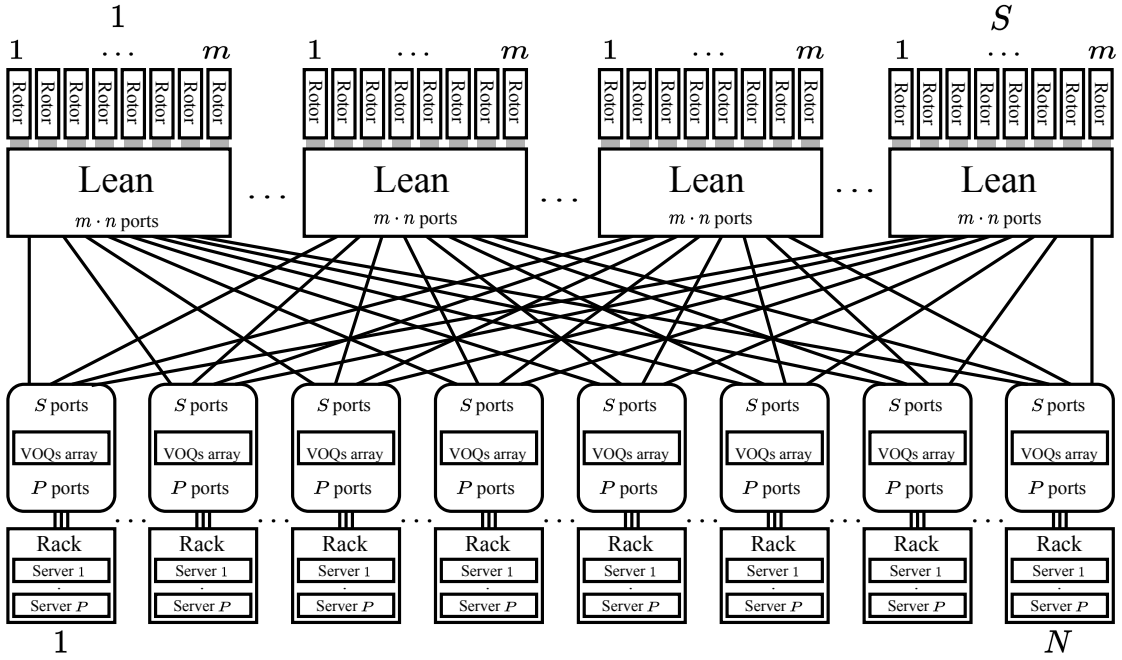
it also allows for adaptive switching configurations through scheduling, resulting in improved throughput.

3.2.4 The Architecture Specifications

The proposed network architecture interconnects N racks of P servers each. Each rack includes an electro-optical *ToR* (top-of-rack) switch that facilitates inter-rack (through the proposed optical network) and intra-rack communication for its P servers. The optical part of the network consists of *Lean* and *Rotor* switches, with the ToR switches as its endpoints. Figure 3.4 illustrates the complete layout of a datacenter with N racks, S Lean switches, each connected to m Rotor switches (S Lean planes). Each Lean switch has $N = m \cdot n$ bi-directional input and output ports, each connected to a different ToR switch.

The ToR switches are equipped with S bi-directional optical ports which connect with the Lean switches via optical fibers (northbound ports), and P Ethernet bi-directional ports which connect to the underlying racks' servers (southbound ports). They also incorporate an

Figure 3.4 The DCN architecture utilizing N racks/ToRs with P servers each, and S Lean switches/planes. ToR switches use $S + P$ ports, S communicate with Lean switches and P with the racks' servers. Lean switches use $N = m \cdot n$ ports.



electronic switching fabric, a $1 \times (N - 1)$ dispatcher, a set of $N - 1$ Virtual Output Queues (VOQs) for all possible destination racks, an internal scheduler, and a set of transceivers (performing electro-optical and opto-electronic transformation). The incoming Ethernet frames from the southbound ports are dispatched though the electronic switching fabric to their corresponding VOQs, to mitigate head-of-line blocking (HOL) (55). The internal scheduler routes the Ethernet frames from the VOQs to one of the S transmitters at the northbound ports, where electro-optical transformation is performed. The incoming optical signals from the northbound ports reach the S receivers, where opto-electronic transformation is performed. Then the Ethernet frames are routed to the electronic switching fabric and then to their corresponding southbound ports.

To achieve full bisection bandwidth, it is necessary to maintain the total capacity among the network levels. Given that all links have the same capacity, this balance is achieved when the number of links ($P \cdot N$) between the racks and ToR switches is equal to the number of links between the ToR and Lean switches. So the number of northbound and southbound ports of each ToR should be equal, thus, $S = P$. Also, we employ S Lean switches each with $N = m \cdot n$ ports, so again we need $S = P$. Oversubscription is achieved with $S < P$.

In our approach, network operation occurs in discrete time intervals called *timeslots* of a fixed duration, during which all optical elements are configured to a specific state. The deci-

sions for the configurations of switches are taken dynamically by a centralized scheduler, according to traffic characteristics. However, scheduling on a per-slot basis seems prohibitive due to communication and processing latency limitations. Therefore, such configurations (also called schedules) are generated in batches of T timeslots, what we call *periods*, as discussed in Section 3.3. This enables significant savings through the aggregation and suppression of monitoring and control information. It also helps absorb traffic peaks, smoothing out the resource allocation process.

At each timeslot, there is a finite reconfiguration time t_{setup}, during which no data can be sent. The remaining time, t_{stable}, is dedicated to data transmissions. We denote the *duty cycle* of the network as:

$$D = \frac{t_{\text{stable}}}{t_{\text{setup}} + t_{\text{stable}}} \cdot 100\%, \tag{3.1}$$

where $t_{\text{setup}} + t_{\text{stable}}$ is the total duration of each timeslot.

3.2.5 Crosspoint Complexity and Reconfiguration Delay

The Lean switches have m selector modules at each of their two stages, each of which can be configured in m ways ($m!$ configurations). Each Lean switch is connected to m Rotor switches, each of which can be configured in n ways. Since we assume $N = m \cdot n$, the crosspoint complexity of a Lean switch is $m^2 = (N/n)^2$, and that of m Rotor switches is $mn = N$.

Since the entire network comprises S Lean switches, its total crosspoint complexity is derived as $((N/n)^2 + N)S$.

The authors of (57) developed a prototype design for the selector module based on commercial off-the-shelf MEMS mirrors, which accomplished 151 µs (re)configuration delay with 61 ports. Subsequently, they proposed a detailed custom module design which can accomplish (re)configuration in 20 µs with up to 2048 ports, using micro-optic port mappings and a micromirror array. Since, the elemental component of the Rotor and Lean switches is based on their selector module design, we consider the Lean DCN to achieve a (re)configuration delay of 20 µs.

3.3 The Control Plane

3.3.1 Preliminaries

The proposed DCN's control is managed through an SDN-enabled control plane, which is divided into three phases: *monitoring*, *scheduling*, and *reconfiguration*. During the monitoring phase, the control plane reads the reported traffic demands from the buffers of the ToR switches and estimates the current traffic demands, while also considering the monitoring delay, as will be discussed later. Next, the control plane executes a batch scheduling task to determine which connections (represented as source-destination rack pairs) will take place during each slot of the period, while taking into account the estimated traffic demands. In the final phase, the control plane distributes the computed schedule to the corresponding switches for reconfiguration.

However, using switches based on the selector module (i.e. Rotor or Lean) determines the switching state for multiple of their ports, constraining the source-destination ToR pairs communicating in each timeslot. The control plane takes these constraints into account and calculates schedules for serving the source-destination pairs that are interdependent at each slot. These schedules that route packets from a source to a destination ToR result in *direct* transmissions.

The timeslots of a period that are not assigned for direct transmissions are used for two-phase routing, where a transient ToR is used as an intermediate node from source to destination ToR. In the first phase, the end nodes (ToRs) decide in a distributed manner to transmit packets to random intermediate hops, and in the second phase, the packets are then transmitted to their corresponding destinations. Notably, the network can use predetermined schedules during these timeslots. This technique is Valiant Load Balancing (VLB) (87, 88), which is also used in RotorNet. We call these transmissions *indirect* transmissions.

This technique makes the control model "semi-centralized" since the control plane does not necessarily decide on the assignments of all timeslots of a period. The degree of centralized control is determined by the degree of the existence of structures in the traffic pattern that can be directly served based on the supported states of the switches, while the remaining traffic is served in an indirect and decentralized way.

3.3.2 Control Cycle

As mentioned earlier, time in the DCN is divided into periods of T timeslots. The DCN operates in two parallel cycles: a) data communication cycles of T timeslots (referred to as Data periods), where communication between nodes occurs, and b) Control plane cycles that

take C Data periods to complete (including monitoring, scheduling and configuration). A Control plane cycle, computes the schedule $\mathbf{S}(t)$ to be applied during the Data period t. It is important to note, however, that the schedule is calculated based on information that was available C periods prior to the Data period to which the Control plane cycle is applied, due to the control communication delay. So, the network controller calculates the schedule for period t, receives information about the queues of the ToR switches during period $t - C$, denoted by queue matrix $\mathbf{Q}(t - C)$ of size $N \times N$. Based on that, the controller creates an estimation matrix $\mathbf{D}(t)$, which it uses to calculate the schedule $\mathbf{S}(t)$ for the period t. Note that during the part of the period that is not scheduled, in $\mathbf{S}(t)$, indirect transmissions are applied through VLB policy.

We provide the following definition for the *traffic matrix*:

Definition 3.3.1 (Traffic Matrix)**.** *A traffic matrix, denoted by* $\mathbf{D}(t) = [d_{ij}(t)]$ *for* $i, j = 1, 2 \ldots, N$*, is a square matrix of size* $N \times N$*. The entries of this matrix, represented by* $d_{ij}(t)$*, are non-negative integers that specify the number of demanded bandwidth slots from rack* i *to rack* j *during period* t*.*

The Control Plane cycle can be summarized in the following steps:

1. Obtain the monitored queue matrix $\mathbf{Q}(t - C)$ and estimate the traffic matrix $\mathbf{D}(t)$.

2. Compute the schedule $\mathbf{S}(t)$ for the direct transmissions based on the estimated traffic matrix $\mathbf{D}(t)$.

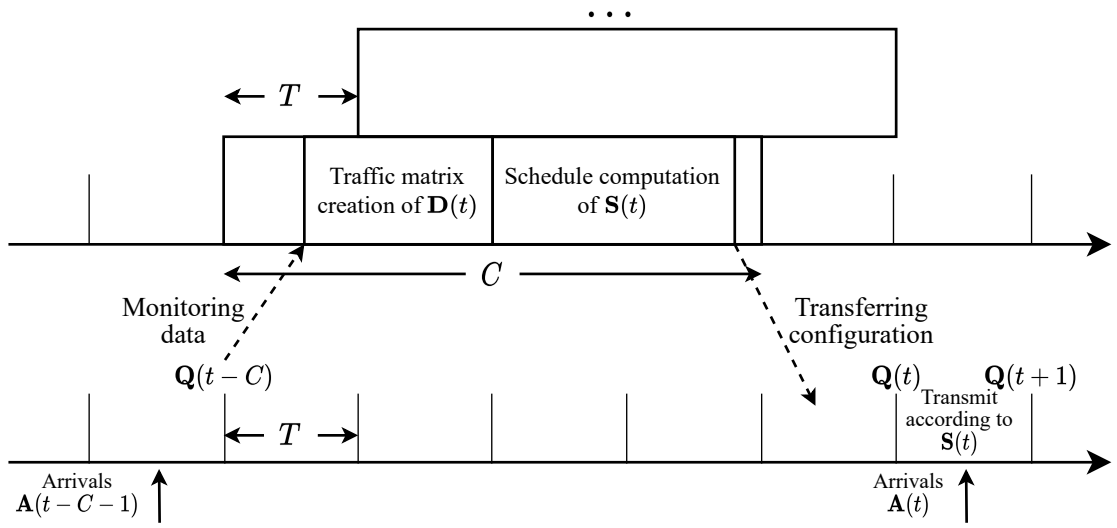
3. Transfer the reconfiguration $\mathbf{S}(t)$ to the corresponding switches.

If the control delay C is greater than one Data period (i.e., the delay required for obtaining the monitored queue matrix, making scheduling decisions and configuring the network, is longer than the data period), a new Control Plane cycle will still begin every Data period, in a "pipelined" way as shown in Fig. 3.5. Therefore, C Control Plane cycles will run in parallel.

The control plane delay C depends on several factors, such as the execution time of the scheduling algorithm and the delay of the monitoring and control protocol used to transmit information between the ToRs and the SDN controller, and between the SDN controller and the data plane devices, respectively. Both delays are influenced by the network size and the Data period, T.

To ensure efficient scheduling, the schedule $\mathbf{S}(t)$ should be sufficiently accurate to be applied during the Data period t if $\mathbf{D}(t)$ is a good approximation of $\mathbf{Q}(t)$ (the ToR queues at the time that it is applied). To estimate $\mathbf{D}(t)$ from $\mathbf{Q}(t - C)$, various methods can be

Figure 3.5 Data and Control Plane cycles.



used such as statistical predictions, filters, and cases where the application communication pattern is known or communicated in advance to the scheduler. The overall scheme is also designed to be self-correcting. If some queues are not served for some periods due to poor scheduling and their size increases due to new arrivals, this information will eventually be communicated to the controller with some delay, and the unserved queues will eventually be serviced.

3.4 Problem Definition and Scheduling Policies

3.4.1 Problem Definition

In a Lean datacenter network, the configurability of the Lean and Rotor switches is limited compared to crossbars of the same size. This constrains the combinations of source-destination pairs that can communicate simultaneously, and limits the traffic that is transmitted directly. The indirect routing solution based on VLB spends half of the throughput for transmissions to intermediate nodes, and thus results in lower throughput.

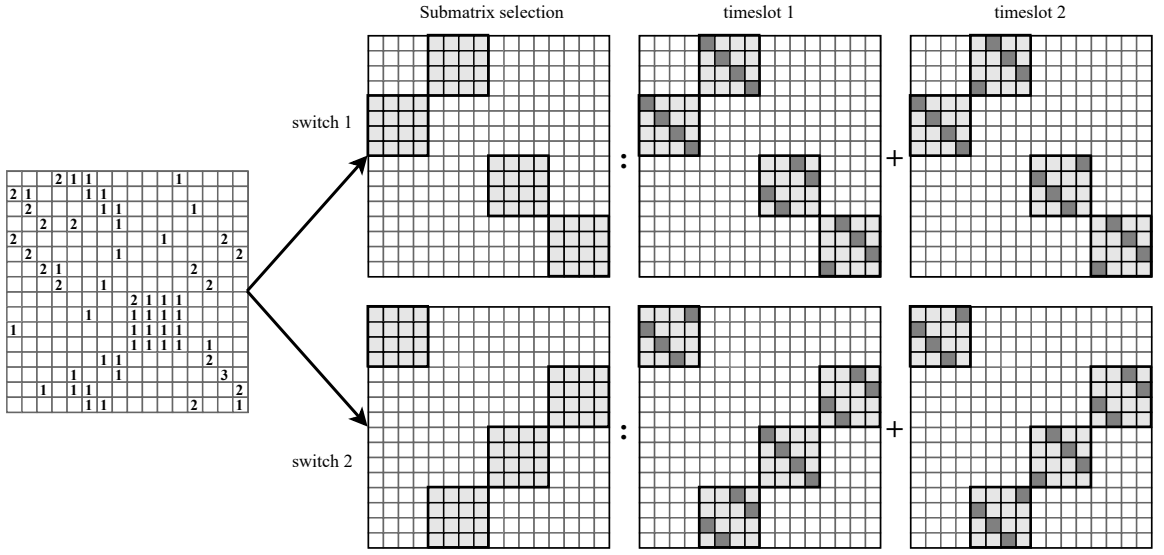
The objective is to maximize the throughput in a Lean datacenter network, comprising N racks with P servers. The network consists of N top-of-rack (ToR) switches with $S + P$ bi-directional ports each, and S Lean planes/switches, where each Lean switch has $m \cdot n$ bi-directional ports and is connected to m Rotor switches, where n is the group factor, and $N = m \cdot n$. During each timeslot, the Lean network is configured in a specific way, enabling the communication between specific ToRs. The SDN-enabled scheduler makes decisions for a period of T timeslots. It examines the traffic and computes schedules for the portion that can be directly transmitted, guaranteeing maximum utilization[1] of the slot capacity during these timeslots. The remaining timeslots of the period are used for indirect transmissions, where the packets are transmitted to random intermediate hops instead of their true destinations, following the VLB policy. A maximization of direct transmissions results in a decrease of the slots/capacity that is reserved for transmissions to intermediate destinations by the VLB policy, leading to an overall increase of the throughput.

3.4.2 Scheduling Constraints

We use matrix notation from linear algebra to describe switching patterns, represented by *block matrices*. A block matrix is a square, non-negative, and integer matrix of size $N \times N$, where $N = m \cdot n$. It has m *row partitions* and m *column partitions*, each of size n. Each intersection of row and column partitions forms an $n \times n$ *submatrix*, and there are m^2 submatrices. We refer to each submatrix's main diagonal and all the cyclic shifts above the main diagonal as *diagonals*. The configuration of the Lean switches is a permutation of the submatrices, while the Rotor switches' configuration corresponds to a selection of a diagonal. Figure 3.6 illustrates how a block matrix is decomposed into submatrices and diagonals to form a sequence of valid switching patterns.

[1] In this work, we exclusively focus on schedules that achieve full utilization. However, it's worth noting that the use of partially-filled schedules could be further investigated.

Figure 3.6 Decomposition of a traffic matrix into direct transmission permutation matrices, in a network with $S = 2$ Lean switches and $N = 16$ racks ($n = 4, m = 4$).



A valid switching pattern of the proposed DCN fabric can be represented by a block matrix $\mathbf{S}(t)$ that satisfies the following constraints:

C1 Each row can have at most one entry set to '1', while the rest of entries are set to '0'.

C2 Each column can have at most one entry set to '1', while the rest of entries are set to '0'.

C3 Each row partition can have at most one non-zero submatrix.

C4 Each column partition can have at most one non-zero submatrix.

C5 Each submatrix can have at most one diagonal with non-zero entries.

Constraints C1 and C2 are straightforward and dictate that a given source ToR can only connect to a single destination ToR, and vice versa. These two are the typical constraints in crossbar switches. Constraints C4 and C5 reflect the fact that a given first-stage selector module of a Lean switch can only connect to one second-stage selector module, and vice versa. Constraint C5 reflect the use of cyclic port mappings in the Rotor switches.

3.4.3 Scheduling Policies

We introduce the *Lean Valiant Decomposition (LVD)* as a combined scheduling policy based on Birkhoff-von Neumann decomposition (BvN) (40, 66) and Valiant Load Balancing (VLB) (87, 88) for our proposed DCN architecture.

Definition 3.4.1 (Perfect matrix)**.** *Perfect is a non-negative matrix, all rows and columns of which sum up to the same number.*

Definition 3.4.2 (Critical sum)**.** *We define the critical sum h of an integer and non-negative matrix* $\mathbf{D} \equiv [d_{ij}]_{i,j=1}^{N}$ *as*

$$h = \max\left(\max_{j:j\in\{1,2,\ldots,N\}}\left(\sum_{i=1}^{N} d_{ij}\right), \max_{i:i\in\{1,2,\ldots,N\}}\left(\sum_{j=1}^{N} d_{ij}\right)\right).$$

The BvN decomposition method builds on the observation that to decompose a traffic matrix $\mathbf{D}$ with a critical sum of h into a sum of permutation matrices, a scheduler would need to execute a maximum cardinality matching algorithm for at most h iterations, as described in (66) (p. 57). This decomposition results in a sequence of h permutations, each with maximum cardinality. The permutations are then translated into configurations for the switches by applying the Lee-Hwang-Capinelli algorithm (47). Note that if the critical sum h of a traffic matrix exceeds T, the decomposition has to be carried out to yield only T switching matrices, as each period has T slots.

On the other hand, VLB is a method that utilizes randomized two-phase routing to support all possible traffic matrices. This method allows each ToR to be agnostic to the global traffic and make randomized local decisions, as randomness translates any traffic pattern to a uniform one, which can be more easily served. However, despite efficiently serving non-uniform traffic without the need for centralized control, VLB may waste half of the available network capacity due to the two-phase routing.

The proposed LVD policy aims to achieve higher throughput compared to the VLB method by taking into account the existence of structured traffic patterns. The policy takes place in two steps, the *decomposition step* and the *load balancing step*:

1. In the decomposition step, the algorithm decomposes the traffic matrix into a sequence of permutation matrices, with respect to the scheduling constraints outlined in Section 3.4.2. These permutations are then used to configure the Lean and the Rotor switches to directly transmit the traffic demands to their corresponding destinations.

2. In the load balancing step, the timeslots of the period that weren't used in the decomposition step are used for the remaining demands to be transmitted indirectly through intermediate destination hops (ToR switches). The transmissions to the intermediate destination hops take place according to predetermined round-robin schedules of the Rotor and the Lean switches. Hence, the intermediate hops are uniformized deterministically.

We refer to the matrix that describes the capacity allocations for directly transmitted demands as the *direct capacity matrix*, and to the matrix that describes the capacity allocations for both directly and indirectly transmitted demands as the *capacity matrix*. In the following pseudocode of the scheduling policy (Algorithm 3) we denote the number of timeslots used for the decomposition step (direct transmissions) as θ, while the remaining $T - \theta$ timeslots are used for the load balancing step (indirect transmissions).

By recalling the Definition 3.4.1 of perfect matrices and the Definition 3.4.2 of the critical sum, we derive the following theorem:

Theorem 3.4.1. *Suppose* $\mathbf{D}$ *is a perfect integer traffic matrix with a critical sum of* h_d*. Let* $\mathbf{A}$ *be a direct capacity matrix with a critical sum of* h_a*, and let* $\mathbf{E}$ *be an indirect capacity matrix obtained by transmitting* $\mathbf{D} - \mathbf{A}$ *through an indirect routing scheme. Then, the capacity matrix obtained by combining* $\mathbf{A}$ *and* $\mathbf{E}$ *is also a perfect integer matrix, with a critical sum of* $2h_d - h_a$*.*

Proof. Let $\mathbf{D}$ be a perfect traffic matrix with critical sum h_d. Since $\mathbf{D}$ is perfect, it can be decomposed into its permutations, which are all perfect matrices. Let $\mathbf{A}$ be the resulting direct capacity matrix, which is also perfect with critical sum h_a. Each permutation corresponds to a set of paths in the network, and the capacity of each link in $\mathbf{A}$ is set to the minimum of the capacities along these paths.

Since $\mathbf{D}$ is perfect, we know that $\mathbf{D} - \mathbf{A}$ is also perfect. We can then apply the Valiant Load Balancing policy to the matrix $\mathbf{D} - \mathbf{A}$ to transmit in two phases, described by matrices which sum up to a perfect indirect capacity matrix $\mathbf{E}$.

The resulting matrix $\mathbf{A} + \mathbf{E}$ is a perfect capacity matrix, since it is obtained by adding two perfect matrices. Moreover, its critical sum is $h_a + 2(h_d - h_a) = 2h_d - h_a$, as claimed, since the critical sum of $\mathbf{E}$ is $2(h_d - h_a)$.

Therefore, the capacity matrix is also perfect with critical sum $2h_d - h_a$, completing the proof. □

According to the theorem, if the LVD policy is applied to a fully structured matrix where all traffic demands can be served directly ($h_a = h_d$), the policy can achieve maximum throughput ($2h_d - h_d = h_d$). However, if there are no structured patterns present ($h_a = 0$), the policy will produce a capacity matrix with a critical sum of $2h_d$, resulting in a loss of half of the throughput.

The time complexity analysis of Algorithm 3 can be summarized as follows:

1. Line 7 is executed $m^2 n$ times, where n is the number of elements per diagonal. Therefore, the complexity of this line is $\Theta(N^2)$, where $N = m \cdot n$.

Algorithm 3 Lean Decomposition

Input: $\mathbf{D} = [\mathbf{D}_{ij} \mid \mathbf{D}_{ij} \in \mathbb{R}^{n\times n}], \forall i, j \in \{1, 2, \dots, m\}, N, T, S, P$
Output: $\mathbf{S} \in \mathbb{Z}^{T\times N}; \mathbf{R} \in \mathbb{Z}^{T\times m}$

$\mathbf{S} \leftarrow [0]^{T\times N}; \bar{\mathbf{S}} \leftarrow [0]^{T\times m}; \mathbf{R} \leftarrow [0]^{T\times m}$
$\mathscr{M} \leftarrow m \times m$ array of lists; $\bar{\mathscr{M}} \leftarrow m \times m$ array of lists
$\mathbf{B} \leftarrow [0]^{m\times m}; \mathbf{P} \leftarrow [0]^{m\times m}$
for $i, j \in \{1, 2, \dots m\}$ **do**
    $\mathscr{M}_{ij} \leftarrow empty\_list(); \bar{\mathscr{M}}_{ij} \leftarrow empty\_list()$
    for $\delta \in \{1, 2, \dots, n\}$ **do**
        $\mu \leftarrow$ Get minimum entry of diagonal δ of $\mathbf{D}_{ij}$
        $\mathscr{M}_{ij} \leftarrow \mathscr{M}_{ij} \cup (\mu)$
        $\bar{\mathscr{M}}_{ij} \leftarrow \bar{\mathscr{M}}_{ij} \cup (\delta)$
    end for
    $\bar{\mathscr{M}}_{ij} \leftarrow$ Sort (descending) the elements of $\bar{\mathscr{M}}_{ij}$ according to $\mathscr{M}_{ij}$
end for
for $i \in \{1, 2, \dots, m\}$ **do** ▷ Initialize maximum values
    for $j \in \{1, 2, \dots, m\}$ **do**
        $\mathbf{B}[i, j] \leftarrow \mathscr{M}_{ij}(\bar{\mathscr{M}}_{ij}(1))$ ▷ Get maximum value
    end for
end for
$\theta \leftarrow 0$
$\mathbf{P} \leftarrow$ Find Maximum Cardinality Matching with input $\mathbf{B}$
while $\mathbf{P}$ is a perfect matrix AND $\theta < T$ **do**
    $\beta \leftarrow \min_{i,j:i,j\in\{1,2,\dots,m\}}\{\mathbf{B}[i, j] \mid \mathbf{P}[i, j] = 1\}$
    $\bar{\mu} \leftarrow 1$
    while $\bar{\mu} \leq \beta$ AND $\theta < T$ **do**
        $\theta \leftarrow \theta + 1$
        $\mathbf{S}[\theta, i] \leftarrow \bar{\mathscr{M}}_{ij}(1), \forall i \in \{1, 2, \dots, m\}$ where $\mathbf{P}[i, j] = 1$
        $\bar{\mathbf{S}}[\theta, i] \leftarrow j, \forall i \in \{1, 2, \dots, m\}$ where $\mathbf{P}[i, j] = 1$
        $\bar{\mu} \leftarrow \bar{\mu} + 1$
    end while
    for $i, j \in \{1, 2, \dots, m\}$ where $\mathbf{P}[i, j] = 1$ **do**
        $\mathscr{M}_{ij}(\bar{\mathscr{M}}_{ij}(1)) \leftarrow \mathbf{B}[i, j] - \beta$
        $\bar{\mathscr{M}}_{ij} \leftarrow$ Update sorting order with given $\mathscr{M}_{ij}$ and $\bar{\mathscr{M}}_{ij}$
        $\mathbf{B}[i, j] \leftarrow \mathscr{M}_{ij}(\bar{\mathscr{M}}_{ij}(1))$
    end for
    $\mathbf{P} \leftarrow$ Find Maximum Cardinality Matching with input $\mathbf{B}$
end while
for $\tau \in \{1, \theta\}$ **do**
    $\mathbf{R}[\tau, :] \leftarrow$ Find configurations for the switches with $\mathbf{S}[\tau, :]$
end for
return $\mathbf{S}, \mathbf{R}$ ▷ For $\tau \in [1, \theta]$, transmit directly according to $\mathbf{S}$.
▷ For $\tau \in [\theta + 1, T]$, transmit indirectly using VLB.

2. In line 11, the heapsort algorithm is executed m^2 times, each time sorting n elements, which requires $O(n \log n)$ time complexity. Thus, the total worst-case complexity induced by this line is $O(m^2 n \log n) < O(N^2)$.

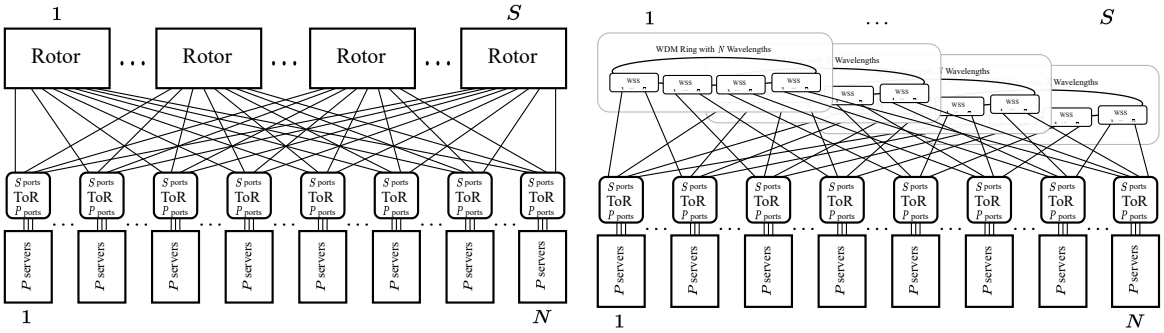
3. Line 25 is executed at most T times, assigning the destinations of N racks. Hence, the complexity is $O(TN)$.

4. Line 34 involves the Hopcroft-Karp algorithm for bipartite matchings of size $m \times m$. This algorithm has a time complexity of $O(m^{\frac{5}{2}}) = O((N/n)^{\frac{5}{2}})$. Since this line is executed at most T times, the total time complexity it induces is $O(T(N/n)^{\frac{5}{2}})$.

5. Finally, line 37 involves Lee-Hwang-Capinelli algorithm. This algorithm has complexity $O(m^2)$. Since it's called at most T times, the total time complexity it induces is $O(Tm^2) = O(T(N/n)^2) < O(T(N/n)^{\frac{5}{2}})$.

Therefore, the worst-case time complexity of Algorithm 3 is $O(N^2 + T(N/n)^{\frac{5}{2}} + TN)$.

Figure 3.7 Alternative 2-level DCN architectures.



(a) DCN with Rotor switches (RotorNet).

(b) DCN with WSSes/WDM rings (Mordia).

3.5 Alternative DCN Architectures

In this section, we describe two alternative 2-level architecture designs: a) the RotorNet architecture (56) (Figure 3.7a), and b) the Mordia architecture (63) with optical crossbar switches (Figure 3.7b). These are compared in this section with the proposed Lean architecture in terms of crosspoint complexities and reconfiguration delay, and in the next section through simulations.

We assume that all considered DCNs have N racks with P servers each, and N ToR switches with $S + P$ bi-directional optical ports, where P of these ports communicate with the underlying servers, and S communicate with the higher level switches. In the following, we will refer to the switches at the second level as *spine switches* to distinguish them from the ToR switches.

Table 3.1 provides a comparison of the network component specifications, and their corresponding crosspoint complexities.

3.5.1 RotorNet: DCN with Rotor Switches

Assuming the network architecture illustrated in Figure 3.7a, a RotorNet network comprises S Rotor switches. Each Rotor switch has N ports, and supports N port mappings of N.

The Rotor switches have a crosspoint complexity of N. Thus, the total crosspoint complexity of the entire network is NS.

As for the reconfiguration delay of the network, the Rotor switches can be designed to be reconfigured in 20 µs (56).

A DCN equipped only with Rotor switches does not require centralized control, as the Rotor switches can operate in a decentralized deterministic manner. For direct transmissions, a round-robin (RR) policy cyclically rotates through all source-destination pairs, which is

sufficient for serving uniformly distributed traffic patterns. However, non-uniform traffic patterns require the use of indirect transmissions through Valiant Load-Balancing (VLB) (87, 88) with the cost of half of the available capacity due to two-phase routing.

Both RR and VLB are fully decentralized and hence their computational complexity is $\Theta(1)$ (87).

3.5.2 Mordia: DCN with WDM Rings and Wavelength-Selective Switches

We assume the architecture illustrated in Figure 3.7b, which corresponds to Mordia DCN. The network comprises S unidirectional rings constructed by optical fibers, each carrying N individual multiplexed wavelengths (WDM). At certain interconnection points, wavelengths are dropped from the rings to the ToRs and added from the ToRs to the rings using *wavelength-selective switches* (WSS) (**?**). Each ToR is designated to transmit with a distinct wavelength, exclusive to any other ToR, ensuring that ToRs connect through unique wavelength channels. The WSS selects at most $\bar{n}$ wavelengths ($1 \times \bar{n}$ WSSes) to route to its $\bar{n}$ bi-directional ports, and onto the $\bar{n}$ underlying ToRs. The other wavelengths are multiplexed with the wavelengths that originate from the underlying ToRs and forwarded to the next interconnection point. The WSSes connect to their WDM ring through 2 unidirectional ports, additional to their $\bar{n}$ bi-directional ports connected with the ToRs.

Since the WDM signal contains N wavelengths, the WSSes' crosspoint complexity is $N(\bar{n}+1)$. A non-blocking network consisting of a ring carrying N wavelengths and $1 \times \bar{n}$ WSSes requires $N/\bar{n}$ WSSes/interconnection points. Therefore, the crosspoint complexity is $N^2+N/\bar{n}$. Furthermore, a full network consists of S rings, resulting in a total of $(N^2+N/\bar{n})S$ crosspoints.

As for the reconfiguration delay of the network, the WSSes can be designed to be reconfigured in 11.5 μs (63).

Similar to the Lean architecture, we assume an SDN-enabled scheduler which makes decisions every T timeslots for direct transmission. To describe the scheduling, we use a similar notation. The scheduler uses as input a traffic matrix $\mathbf{D}(t) \in \mathbb{Z}^{N \times N}$ which describes the (estimated) accumulated traffic between ToR pairs for Data period t. As in Lean architecture, this is assumed to be created through monitoring and an estimation algorithm and/or application traffic awareness. The critical sum of $\mathbf{D}(t)$ is assumed to be h. According to (66), Birkhoff-von Neumann (BvN) decomposition can be used for optimal scheduling, which decomposes a traffic matrix $\mathbf{D}(t)$ with critical sum h into a sequence of permutation matrices

with maximum cardinality:

$$\mathbf{D}(t) = \mathbf{P}^{(1)} + \mathbf{P}^{(2)} + \mathbf{P}^{(3)} + \cdots + \mathbf{P}^{(h-1)} + \mathbf{P}^{(h)}$$

This is done by executing a maximum cardinality matching algorithm for at most h iterations. We refer to the exact BvN decomposition method as *EXACT* (75):

Algorithm 4 EXACT Decomposition

1. Set slot counter to $\theta \leftarrow 1$.
2. Find maximum cardinality matching $\mathbf{P}$ with input $\mathbf{D}(t)$.
3. Allocate bandwidth according to $\mathbf{P}$; $\theta \leftarrow \theta + 1$.
4. Find switch configurations for $\mathbf{P}$.
5. Subtract $\mathbf{P}$ from $\mathbf{D}(t)$. If $\theta \leq T$ and $\mathbf{D}(t)[i,j] > 0, \forall i,j \in \{(i,j) \,|\, \mathbf{P}[i,j] = 1 \wedge i,j = 1,2,\ldots,N\}$, go to step 3.
6. If $\theta \leq T$ go to step 2.

A traffic matrix is considered "admissible" if it can be served within the available capacity provided by the available slots in a serving Data period. For the DCN architecture shown in Figure 3.7b, a traffic matrix is admissible if its critical sum satisfies $h \leq T$.

For an admissible traffic matrix, the number of decomposition steps is limited to T. We can use the Hopcroft-Karp algorithm (35) for finding maximum matchings, which has a worst-case complexity of $O(N^{\frac{5}{2}})$ for a dense matrix. For finding the switch configurations we can use the Lee-Hwang-Capinelli algorithm (47) which has complexity $O(N^2)$. Both algorithms are called for at most T times. Therefore, the complexity of the former dominates the latter's, the total worst-case complexity of the algorithm is $O(TN^{\frac{5}{2}})$.

3.5.3 Comparison of the Lean and the Alternative DCN Architectures

The Lean, the RotorNet and the Mordia DCN architectures are compared in Table 3.1. All three architectures have the same number of racks/ToR switches N with the same number of ports (P southbound and S northbound) and VOQs ($N-1$), the same number of servers per rack P, and the same number of spine switches S (S Lean planes in the case of the Lean DCN).

The RotorNet exhibits the lowest crosspoint complexity among all three (NS) and the Mordia the highest ($(N^2 + N/\bar{n})S$). However, the Lean DCN's crosspoint complexity highly depends on the group factor n. As $n \to N$, the crosspoint complexity tends to be $O(NS)$. On the contrary, as $n \to 1$, the crosspoint complexity tends to be $O(N^2S)$. In an in-between scenario, where $n = \sqrt{N}$, the crosspoint complexity tends to be $O(NS)$. According to (36), crosspoint complexity often bears a direct relationship with minimizing power consumption and other cost criteria. As a result, the Lean DCN architecture may lower the total monetary cost of the network.

As for the worst-case time complexities of the algorithms, both RR and VLB ("RotorNet" in Table 3.1) exhibit the lowest one among all three, namely $\Theta(1)$, and the EXACT ("Mordia" in Table 3.1) the highest, namely $O(TN^{\frac{5}{2}})$. The worst-case time complexity of the LVD algorithm ("Lean DCN" in Table 3.1) again depends on the group factor n. As $n \to N$, the time complexity tends to be $O(N^2 + TN)$. On the contrary, as $n \to 1$, the worst-case time complexity tends to be $O(TN^{\frac{5}{2}})$, equal to EXACT's. For $n = \sqrt{N}$, the worst-case time complexity of the LVD algorithm tends to be $O(N^2 + TN^{\frac{5}{4}})$.

Table 3.1 Comparison between fully connected RotorNet, Mordia and Lean DCNs, where $S + P$ is the ToR switch radix (n: group factor in Lean, $\bar{n}$: WSSes' bidirectional ports in Mordia).

	RotorNet	**Mordia**	**Lean DCN**
Number of racks	N	N	N
Number of servers per rack	P	P	P
# of spine switches[1]	S	S	S
# of ToR switches	N	N	N
# of bi-directional ports of spine switches	N	N	N
# of bi-directional northbound ports of ToR switches	S	S	S
# of bi-directional southbound ports of ToR switches	P	P	P
# of bi-directional ports of WSSes	–	$\bar{n}$	–
# of wavelengths	1	N	1
# of VOQs per ToR switch	$N-1$	$N-1$	$N-1$
# of crosspoints per spine switch	N	$N^2 + N/\bar{n}$	$(N/n)^2 + N$
Reconfiguration delay	20 µs	11.5 µs	20 µs
Duty cycle	D	D	D
Timeslot duration	$20/(1-D)$ µs	$11.5/(1-D)$ µs	$20/(1-D)$ µs
Total number of VOQs	$N(N-1)$	$N(N-1)$	$N(N-1)$
Total number of crosspoints	NS	$(N^2 + N/\bar{n})S$	$((N/n)^2 + N)S$
Scheduling worst-case time complexity	$\Theta(1)$	$O(TN^{\frac{5}{2}})$	$O(N^2+T(N/n)^{\frac{5}{2}}+TN)$

[1] With "spine switch" we refer to a) a Rotor switch in RotorNet, b) an optical ring with WSSes in Mordia, and c) a Lean plane in Lean DCN.

3.6 Simulation Experiments

3.6.1 Simulation Setup

In this section, we present a set of simulation results using OMNET++ packet-level simulator. We consider three network setups comprising 128 racks each, namely a RotorNet, a Mordia and a Lean network. Each rack has a Top-of-Rack (ToR) switch with $P = 16$ Ethernet ports facing the servers of the underlying rack (southbound ports) and $S = 16$ optical ports facing the spine switches of the network (northbound ports), with each link being 100 Gbps. The Data cycle was $T = 64$ timeslots and each timeslot lasts for 200 μwith RotorNet and Lean, and 115 μs with Mordia. In the scenarios of the Mordia and the Lean networks, a new batch schedule is generated for each Data cycle, and we assumed that the control cycle takes $C = 1$ period (61) to generate the schedule.

RotorNet Network Setup

The RotorNet network comprises 128 optical ToR switches and 16 128×128 Rotor (spine) switches, which implement 128 port mappings. We examine the RotorNet architecture by applying both Round-Robin (RR) and Valiant Load Balancing (VLB) policies. The RR policy transmits only directly to the destinations when the network configuration allows a transmission. On the other hand, with the VLB policy, the packets are initially transmitted indirectly from the source to a random intermediate rack and then from the intermediate to the destination rack. In both policies, the decisions are made by hosts without centralized orchestration.

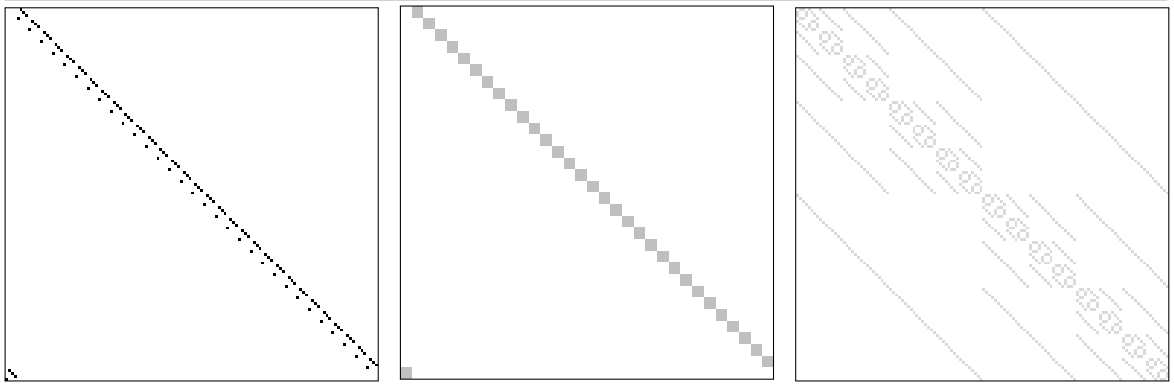
Mordia Network Setup

The Mordia network comprises 128 optical ToR switches and 16 optical WDM rings with 4 1×32 WSSes each. We refer to an optical ring and its corresponding WSSes as a spine switch with 128 ports. We examine the Mordia architecture by applying the EXACT scheduling policy (Section 3.5.2) with the assistance of a centralized orchestrator, which computes the batch schedules for each Data cycle.

Lean Network Setup

The Lean network comprises 128 optical ToR switches and 16 128×128 Lean planes. At each Lean plane, there is a Lean switch and 32 Rotor switches with 4 ports (512 Rotor switches in total). We examine the Lean architecture by applying the LVD policy with the assistance of a centralized orchestrator, which computes the batch schedules for each Data cycle.

Figure 3.8 Structured traffic patterns.



(a) `WTraffic` **(b)** `BlkDiag` **(c)** `FFT`

The traffic is divided into *structured* and *unstructured*. A percentile of the load is transmitted following the corresponding structured pattern. We apply three different traffic structured patterns: W traffic, Block diagonal traffic and Fast Fourier Transform traffic (34), which we abbreviate as `WTraffic`, `BlkDiag` and `FFT`, respectively. The unstructured traffic is distributed uniformly among the ToRs which are randomly selected during each time slot. The percentile of the structured traffic is denoted by a parameter w within the range of $(0, 1)$.

In the `WTraffic` structured traffic, a source rack transmits to a specified destination rack that is located at a predefined distance from it. A depiction of `WTraffic` structured pattern is given in Figure 3.8a. In the `BlkDiag` case, the racks are partitioned in clusters and traffic is transmitted inside each cluster. We examine a scenario of 32 clusters of 4 racks each. A depiction of `BlkDiag` structured pattern is given in Figure 3.8b. Finally, in the case of `FFT`, the racks shape the structured traffic part by applying a 7-stage radix-2 butterfly FFT algorithm of size 128 (34), with all the racks participating in all stages. A depiction of the corresponding pattern is given in Figure 3.8c.

The setup parameters are provided in Table 3.2.

3.6.2 Performance Comparison Between Different DCNs and Policies

For the first evaluation, we compared the performance of network architectures and their corresponding algorithms. We utilized RR and VLB for the RotorNet, EXACT for the Mordia, and LVD for the Lean network, using `WTraffic` with $w = 0.5$.

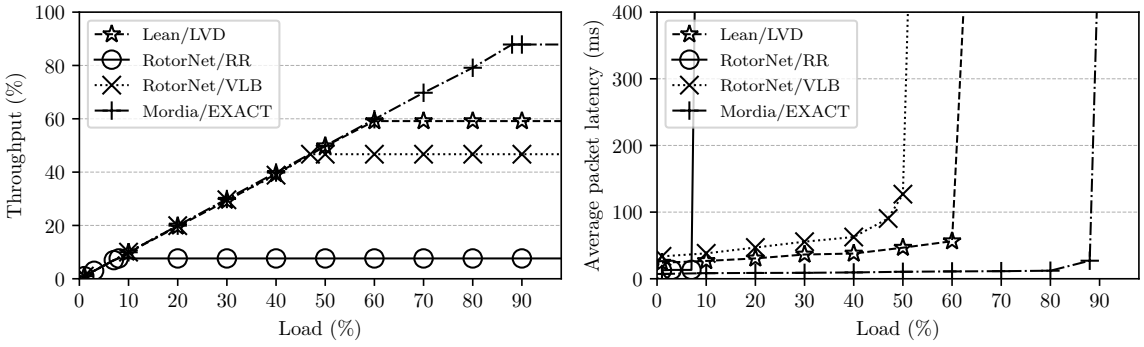
In Figure 3.9a, we present the corresponding throughput results. The RotorNet achieves a maximum throughput of 7.6% using the RR policy and 46.7% using the VLB policy. The Mordia achieves the highest throughput, reaching 87.9% when using the EXACT policy. Finally, for the Lean network, using the LVD policy, the achieved throughput is 59.1%, which

Table 3.2 Network simulation setup.

	RotorNet	**Mordia**	**Lean**
Number of racks (N)	128	128	128
Number of servers per rack (P)	16	16	16
Total number of servers	2048	2048	2048
Group factor of spine switches (n)	128	–	4
Data communic. cycle timeslots (T)	64	64	64
Control plane cycle periods (C)	1	1	1
Link capacity (Gbps)	100	100	100
Timeslot duration (µs)	10	10	10
# of spine switches[1] (S)	16	16	16
# of ToR switches	128	128	128
# of WSSes per spine	–	4	–
# of ports of spine switches[1] (N)	128	128	128
# of north. ports of ToR switch. (S)	16	16	16
# of south. ports of ToR switch. (P)	16	16	16
# of bi-directional ports of WSSes ($\bar{n}$)	–	32	–
# of VOQs per ToR switch	127	127	127
# of crosspoints per spine switch	128	16388	1152
Reconfiguration delay	20 µs	11.5 µs	20 µs
Duty cycle (D)	90%	90%	90%
Timeslot duration	200 µs	115 µs	200 µs
Total number of VOQs	16256	16256	16256
Total number of crosspoints	2048	262208	18432

[1] With "spine switch" we refer to a) a Rotor switch in RotorNet, b) an optical ring with WSSes in Mordia, and c) a Lean plane in Lean DCN.

Figure 3.9 Network and policy comparison.



(a) Examination of throughput. **(b)** Examination of average packet latency.

places it between the RotorNet/VLB and the Mordia/EXACT cases. This result is expected since the structured part of the traffic allows the LVD policy to directly transmit a signifi-

cant portion of the total traffic to their corresponding destinations while using the remaining slots to transmit indirectly with VLB. In contrast, the RotorNet/VLB is traffic-agnostic, uniformizing the entire traffic load by transmitting to intermediate destination racks. While theoretically capable of achieving a throughput of at least 50%, in our simulations it achieved 46.7%. This discrepancy can be attributed to the fact that, although a fraction of the traffic reached its destination in the first hop, providing a marginal increase in throughput, this increase was offset by the reconfiguration delay during which no transmissions occurred. We conclude that the Lean/LVD case improved the throughput by 26.8% compared to the RotorNet/VLB case, while achieving 67.3% of the total throughput of the Mordia/EXACT case.

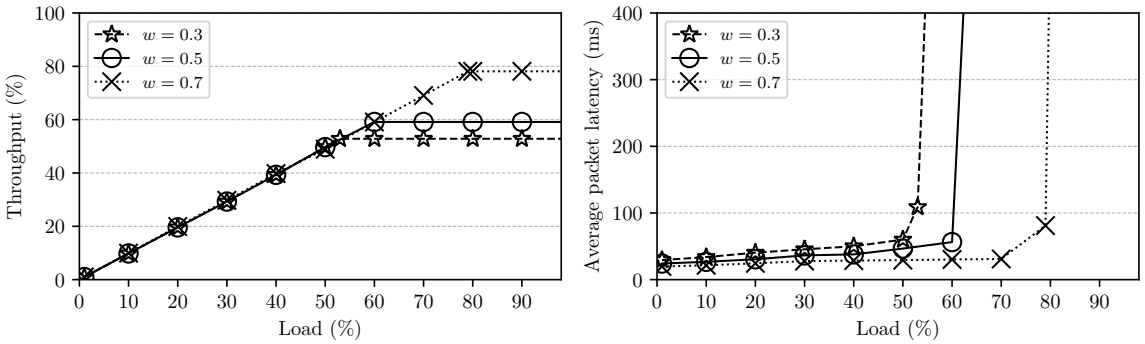
In Figure 3.9b, we present the average packet latencies for the scenarios discussed earlier. In the RotorNet network, the latency scales linearly from 12.74 to 13.31 ms (7% load) when using the RR policy, and from 33.62 to 62.63 ms (40% load) when using the VLB policy. For the Mordia network, the latency scales linearly from 7.23 to 12.07 ms (80% load) when using the EXACT policy. Finally, for the Lean network, using the LVD policy, the average packet latency scales between 24.14 and 56.34 ms (60% load). It is evident that policies that transmit only directly (RR and EXACT) have lower average packet latencies than the indirect policies (VLB and LVD). However, compared to VLB, the LVD policy has lower average packet latencies by 28–39% (up to 40% load).

3.6.3 Performance Comparison with Different Levels of Traffic Uniformity

In Figure 3.10a, we utilize `WTraffic` to investigate the effect of different values of the parameter w ($w = 0.3$, $w = 0.5$, and $w = 0.7$) on the performance of the Lean network with the LVD policy. In the first evaluation, where $w = 0.3$, the network achieves a maximum throughput of 52.8%. With $w = 0.5$, the achieved throughput improves to 59.1%, and by further increasing the percentile of the structured part of the traffic with $w = 0.7$, the achieved maximum throughput reaches 78.1%. It is apparent that as the percentile of the structured traffic increases, the maximum throughput increases as well. This is because an increased amount of traffic reaches its destinations directly, saving more network capacity from transmissions to intermediate destination racks.

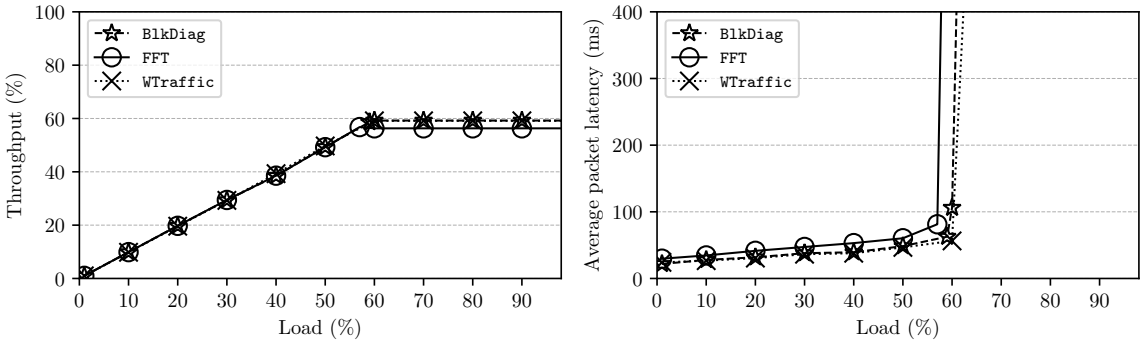
In Figure 3.10b, we also evaluate the average packet latencies of the mentioned cases. Initially, for $w = 0.3$, the average packet latency scales linearly from 29.35 to 60.02 ms (50% load). Then, for $w = 0.5$, the latency scales from 24.14 to 56.34 ms (60% load), and finally, for $w = 0.7$, the average packet latency scales linearly from 19.66 to 31.05 ms (70% load).

Figure 3.10 `WTraffic` uniformity comparison.



(a) Examination of throughput. **(b)** Examination of average packet latency.

Figure 3.11 Structured traffic pattern comparison.



(a) Examination of throughput. **(b)** Examination of average packet latency.

We can observe a small decrease of around 21% on average between $w = 0.3$ and $w = 0.5$ and between $w = 0.5$ and $w = 0.7$, in low traffic loads (up to 40%).

3.6.4 Performance Comparison with Different Traffic Patterns

In this section, we investigate the performance of the Lean network using the LVD policy with different traffic patterns: `WTraffic`, `BlkDiag`, and `FFT` with $w = 0.5$. To visualize the structured parts of these traffic patterns, we present them as stochastic bitmaps in Figure 3.8.

Figure 3.11a displays the maximum throughput achieved by each traffic pattern. The Lean network achieves a maximum throughput of 59.1% with `WTraffic`, 59.2% with `BlkDiag`, and 56.3% with `FFT`. Overall, the network and policy perform similarly in terms of throughput for all three traffic patterns. However, the `FFT` traffic pattern achieves around 5% less throughput compared to the other two due to a part of its structured traffic not matching patterns that can be fully served directly by the Lean network. This leads to some structured

traffic being served indirectly, resulting in a reduction of the maximum achievable throughput.

In Figure 3.11b, we also evaluate the average packet latencies of the three structured traffic patterns. For `WTraffic`, the average packet latency scales linearly from 24.14 to 56.34 ms at 60% load. For `BlkDiag`, the latency scales from 21.81 to 62.66 ms at 59% load, and for `FFT`, it scales linearly from 29.67 to 60.33 ms at 50% load. In low traffic loads (up to 40%) we observe a small increase in latency when we apply `FFT`, of around 31% on average compared to the `WTraffic`, and around 30% on average compared to the `BlkDiag`. As in the throughput case, this is due to the fact that a bigger part of the `FFT` traffic is transmitted indirectly, compared to the other, and hence more packets stay queued.

3.7 Partial Configurability Applied to RotorNet

In this section, we examine scheduling policies aimed at optimizing the performance of the RotorNet architecture as described in (56). To set the stage, we revisit the architectural depiction provided in Figure 3.7a, which was initially introduced in Section 3.5.1. In pursuit of this goal, we harness partial configurability, empowered by a centralized control plane, as previously detailed in Section 3.3.

Our primary aim in this investigation is to devise a scheduling and routing scheme that outperforms Valiant Load Balancing (VLB). The policies we scrutinize include:

Round-Robin (RR): The most straightforward scheduling algorithm applicable to such a network is the Round-Robin (RR) method. In this approach, optical switches establish circuits among the ToR uplinks in a cyclic manner, and packets are forwarded once a circuit connecting to their destination ToR is established. RR is anticipated to excel in scenarios with uniform traffic patterns but may falter when dealing with structured or non-uniform traffic.

Valiant Load Balancing (VLB): The authors of (56) employed Valiant Load Balancing (VLB). VLB's objective is to harmonize structured traffic patterns, effectively rendering them akin to a uniform traffic pattern. It accomplishes this by introducing a random step in packet forwarding, directing them to an intermediate destination before reaching their final destination. VLB exhibits robust performance across various traffic patterns, but this comes at the expense of a reduction in maximum throughput by half and increased latency.

Weighted Round-Robin (WRR): This policy involves connections among the ToR uplinks, where the connection weights, which may or may not be uniform, remain static during network operation. This approach is particularly applicable when there is prior knowledge of job placement and traffic requirements for host applications, making it a suitable choice for application-aware networking. For evaluation purposes, weights can be preset with knowledge of the anticipated traffic pattern to maximize performance tailored to that specific scenario. In this setup, packets are forwarded directly to their intended destinations

Adaptive Weighted Round-Robin (AWRR): AWRR was designed with the goal of surpassing VLB while requiring minimal knowledge about the status of the network buffers. The fundamental concept behind AWRR involves learning the traffic pattern based on gradual feedback from network devices. To begin, let N represent the number of available permutations $\mathbf{P}^{(\theta)}$ for $\theta = 1, 2, \ldots, T$ that can be used during a

period. Similar to the Weighted Round-Robin (WRR) approach, a weight vector is maintained to determine the allocation of slots for each $\mathbf{P}^{(\theta)}$. However, in the case of AWRR, these weights are not known in advance and evolve over time, adapting to the traffic pattern by considering the success level of each $\mathbf{P}^{(\theta)}$.

We define $(\bar{w}_1(t), \bar{w}_2(t), \dots, \bar{w}_N(t))$ as the weight vector at round t, where $\sum_{i=1}^{N} \bar{w}_i(t) = 1$. Also, let $\mathbf{D}(t) = [d_{ij}(t)]$ for $i, j = 1, 2, \dots, N$ represent the traffic matrix at period t. For each diagonal i, where $i = 1, 2, \dots, N$, of $\mathbf{D}(t)$, the maximum number of DUs $\bar{q}_i(t)$ is maintained, defined as $\bar{q}_i(t) = \max_{i:i\in\{1,2,\dots,N\}}(d_{j,\ell(i,j)}(t))$, where

$$\ell(i, j) = (i + j - 2) \pmod{N + 1}. \tag{3.2}$$

Initially, the weights are set to $\bar{w}_i(t_0) = 1/N$, for $i \in \{1, 2, \dots, N\}$.

The success $\sigma_i(t)$ is the number of DUs that are served from $\mathbf{P}^{(\theta)}$ during period t; the relative success is calculated as

$$\bar{\sigma}_i(t) = \frac{\sigma_i(t)}{\sum_{j=1}^{N} \sigma_j(t)} \tag{3.3}$$

for $i = 1, 2, \dots, N$. During period t, the mean relative success is given by:

$$\mathbb{E}[\bar{\sigma}](t) = \frac{1}{N} \sum_{i=1}^{N} \bar{\sigma}_i(t). \tag{3.4}$$

The weights evolve according to a training scheme:

$$\bar{w}_i(t + 1) = \bar{w}_i(t) + \gamma \cdot \left(\bar{\sigma}_i(t) - \mathbb{E}[\bar{\sigma}](t)\right),$$

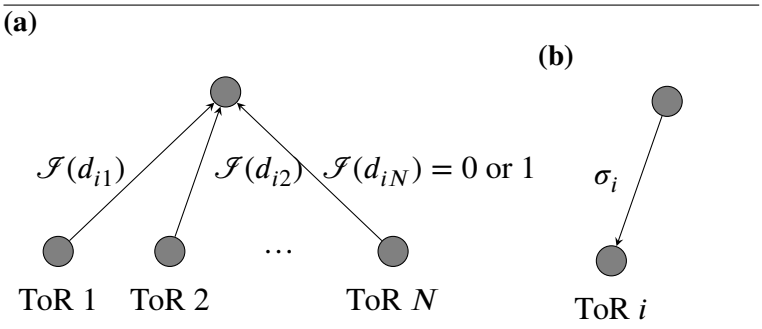
where $\gamma \in (0, 1)$ determines the training memory and rate of convergence. Upper and lower bounds are applied to all weights. The lower threshold ensures that some weights do not converge to zero, allowing permutations that are rarely used to be occasionally tried again. This adaptation helps the system respond to pattern changes. The value of the lower threshold balances the rate of learning new traffic patterns with the maximum throughput achievable for each pattern.

Additionally, a credit policy scheme is employed to control the rounding process by managing rounding errors and noise, ensuring that they average to zero in the long run. Credits are tracked using a vector $(c_1, c_2, \dots, c_N)$. Algorithm 5 is called at each period t to implement this scheme.

Algorithm 5 Adaptive Weighted Round-Robin

for $i \in \{1, 2, \dots N\}$ **do**
    $\bar{q}_i(t) \leftarrow \max_{i:i\in\{1,2,\dots,N\}} \left(d_{j,\nu(i,j)}(t)\right)$, with $\nu(i,j) = (i + j - 2) \pmod{N+1}$
end for
Update $\bar{w}_i(t), \forall i \in \{1, 2, \dots, N\}$, according to the training scheme.
$c \leftarrow \min_{i:i\in\{1,2,\dots,N\}}(\bar{w}_i(t))$
$\theta \leftarrow 1$
while $\theta \leq T$ and $\sum_{i=1}^{N} \bar{q}_i(t) > 0$ **do**
    for $i \in \{1, 2, \dots, N\}$ **do**
        $c_i \leftarrow c_i + c$
        while $\bar{q}_i(t)$ not empty and $c_i \geq \bar{w}_i(t)$ and $\theta \leq T$ **do**
            $p \leftarrow \min(\bar{q}_i(t)$, slot capacity in packets$)$
            for $s \in \{1, 2, \dots, N\}$ **do**
                $d \leftarrow \nu(s, i)$
                Schedule p packets $s \rightarrow d$ at slot θ, during period t.
            end for
            $\bar{q}_i(t) \leftarrow \bar{q}_i(t) - p$
            $c_i \leftarrow c_i - 1$
            $\theta \leftarrow \theta + 1$
        end while
    end for
end while

Figure 3.12 Communication of successes for AWRR.



A centralized scheduler requires extensive information, as it needs to be aware of the complete state of the queues at the ToRs. This means it must be informed about the buffer occupancies, denoted as $d_{ij}(t)$, for all sources i, destinations j, and at all times t. Providing this level of detail is demanding in terms of state information, requiring constant transmission to the central scheduler.

In contrast, the AWRR algorithm has a more efficient approach. It solely needs knowledge of the relative success $\bar{\sigma}_i, i \in \{1, 2, \dots, N\}$ to make scheduling decisions (Figure 3.12).

In practice, σ_i can be computed as $\sigma_i = \sum_{j=1}^{N} \mathcal{J}(d_{ij})$, where

$$\mathcal{J}(x) = \begin{cases} 1, \text{if } x > 0 \\ 0, \text{otherwise,} \end{cases}$$

is an indicator function, signaling success or "hit" for the (i, j) entry of $\mathbf{P}^{(\theta)}$. The value of σ_i represents the total number of successes for permutation $\mathbf{P}^{(\theta)}$.

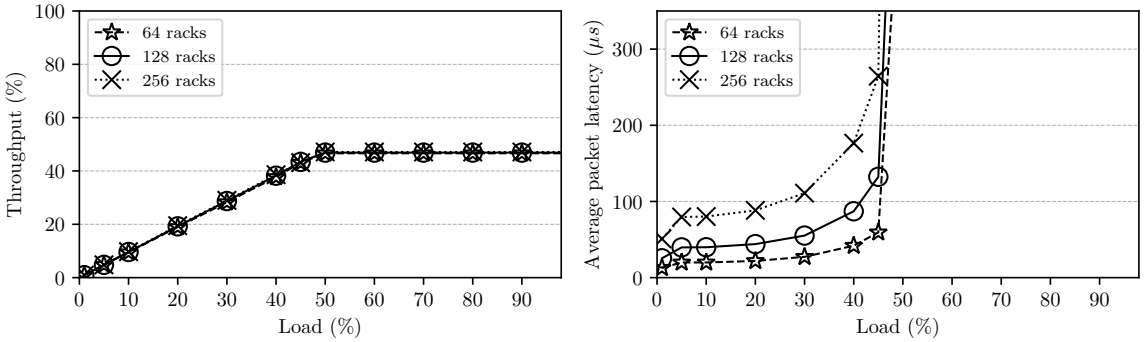
It's important to note that the computation of these summations can be performed efficiently. Given that summation is an associative operation and many parallel computation models consider it to be of $O(1)$ complexity, it can be executed rapidly using a parallel prefix operation over the control tree's wires. This approach minimizes the communication overhead and computational load on the system.

3.8 Additional Simulation Experiments on RotorNet

3.8.1 Traffic Profiles Setup

The traffic is divided into *structured* and *unstructured*. A percentile of the load is transmitted following the corresponding structured pattern. We apply three different traffic structured patterns: W traffic, Block diagonal traffic and Fast Fourier Transform traffic (34), which we abbreviate as `WTraffic`, `BlkDiag` and `FFT`, respectively (recall Figure 3.8). The unstruc-

Figure 3.13 Dimensioning study comparison.



(a) Examination of throughput.

(b) Examination of average packet latency.

tured traffic is distributed uniformly among the ToRs which are randomly selected during each time slot.

The percentile of the structured traffic is denoted by a parameter w which falls within the range of $(0, 1)$. For `WTraffic`, we also use the parameter $wCount$ to denote the number of diagonals where the traffic hotspot is concentrated on (a scenario involving multi-diagonal traffic).

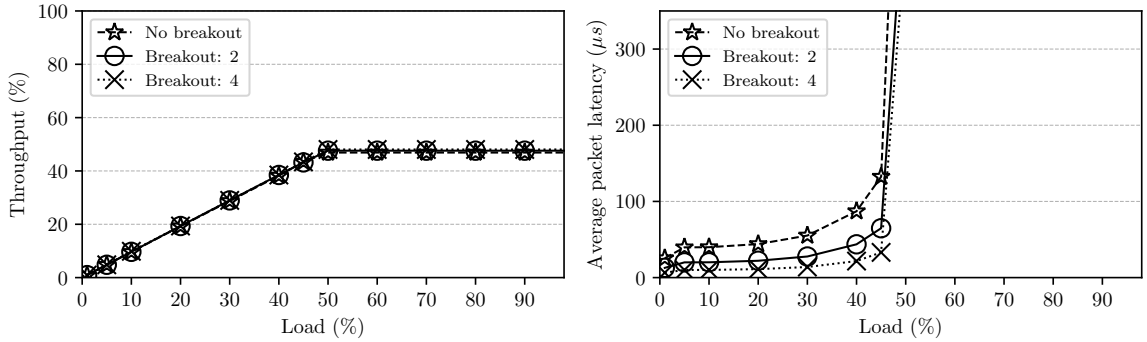
3.8.2 Dimensioning and Breakout Performance Study

The network's latency is anticipated to rise as the network's size expands. When additional Top-of-Rack (ToR) switches are introduced while maintaining a constant number of uplinks, it implies that the circuits connecting any pair of ToRs will be established less frequently. This effect is first demonstrated using our developed simulator. To address this issue, we propose breaking out the ToR uplinks. In this approach, each port of modern switches, such as a 100 Gbps port, is divided into multiple lanes, for example, 4×25 Gbps lanes, which can be connected to different optical switches.

Figure 3.13 illustrates the throughput and latency for networks of different sizes: 64, 128, and 256 racks. Packet forwarding is executed through random intermediate nodes, following Valiant Load Balancing (VLB). The throughput remains consistent as the network scales, remaining close to 50%. This is due to the use of VLB. However, the latency shows a linear increase with the network's size.

As explained earlier, this increase in latency was expected when the number of ToRs increases while the number of uplinks for each ToR remains constant. To provide more specific details, the increase in latency under light loads is twice as pronounced when the number of ToRs is doubled and quadruples when the number of ToRs is multiplied by four, in line with our expectations.

Figure 3.14 Breakout factor comparison.



(a) Examination of throughput.

(b) Examination of average packet latency.

Figures 3.14a and 3.14b illustrate the throughput and latency for different breakout factors. When the breakout factor is set to 2, the lanes are divided into two groups, with each group consisting of two lanes at 25 Gbps. When the breakout factor is 4, the lanes are divided into four groups, each with one lane at 25 Gbps.

As shown in the figures, when breakout is employed, throughput remains consistent and latency decreases linearly, as the breakout factor increases. This indicates that by using ToR uplink breakouts, the latency increase can be effectively mitigated as the network scales.

3.8.3 Performance Comparison with Different Policies

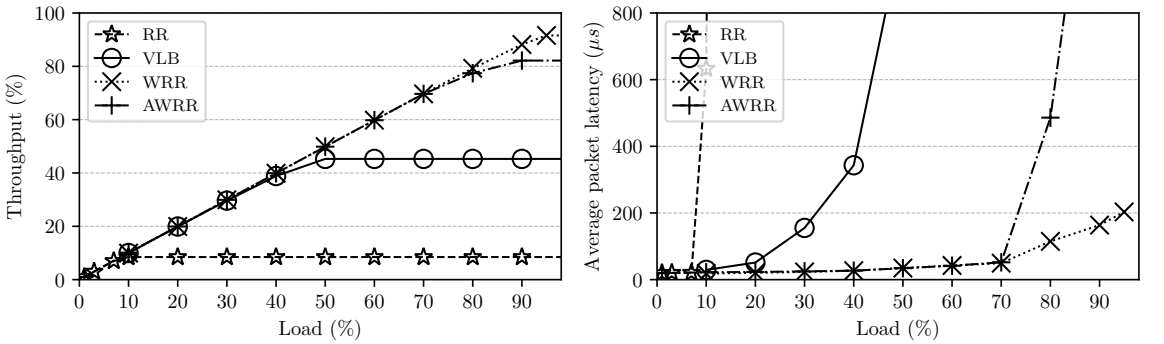
In this section, we assess the performance of the scheduling algorithms and routing schemes described previously, with a primary focus on two critical network metrics: latency and throughput.

Figure 3.15 provides a comprehensive view of the scheduling algorithms' performance under varying network loads. Notably, RR exhibits significant underperformance, particularly when dealing with a traffic pattern where a substantial portion (30%) of the total load is concentrated in a few connections (as indicated by the $\omega = 0.3$ pattern). RR's inefficiency leads to the creation of numerous underutilized connections.

VLB, on the other hand, alleviates the underutilization issue by rerouting concentrated traffic through random destinations. However, this redirection comes at a cost, limiting the network's throughput to a maximum of 50%. Furthermore, it has a negative impact on latency.

WRR serves as a reference for comparison with AWRR, as it has prior knowledge of the primary hotspots and achieves nearly 100% utilization. AWRR was specifically designed to outperform VLB, and Figure 3.15 clearly illustrates its success in achieving this goal.

Figure 3.15 Scheduling algorithms comparison.



(a) Examination of throughput.

(b) Examination of average packet latency.

AWRR performs similarly to WRR until the network load reaches around 80%, signifying a remarkable improvement of over 30% compared to VLB.

3.8.4 Performance Comparison of AWRR with Different Traffic Profiles

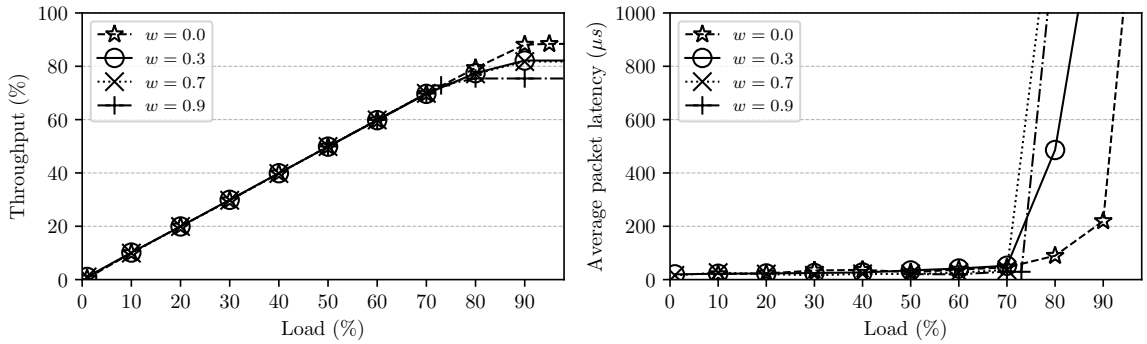
In this section, we delve into the performance of AWRR under different traffic patterns and explore how it adapts when the traffic pattern undergoes changes.

Our initial investigation, depicted in Figure 3.16, focuses on AWRR's performance across a range of w values. Regardless of the specific w value, AWRR consistently achieves nearly 100% utilization until the network load approaches 80%. For higher load values, AWRR demonstrates superior performance when dealing with smaller w values. Smaller w values are indicative of traffic patterns that closely resemble a uniform distribution, a scenario where weighted round-robin algorithms are known to reach peak performance.

Figure 3.17 explores the impact of varying values of $wCount$. Notably, we observe that there are no substantial performance differences. However, a slight improvement can be discerned when considering larger values of $wCount$. Greater $wCount$ signifies a higher number of "useful" matchings, making it easier for AWRR to accommodate the traffic and enhance traffic uniformity.

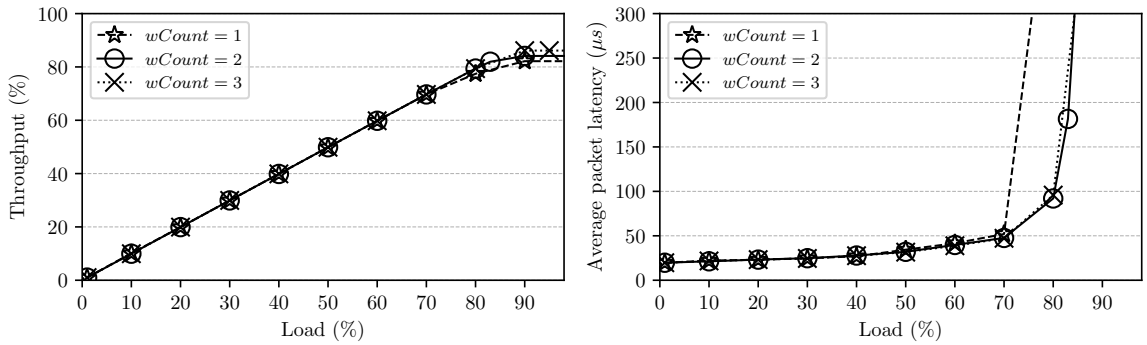
In Figure 3.18, we delve into the convergence behavior of AWRR when confronted with changing traffic patterns. The graph illustrates the performance concerning three distinct traffic patterns evolving over time (X-axis): `WTraffic`, `BlkDiag`, and `FFT`. The Y-axis represents the achieved throughput. Notably, it takes approximately 500 µs for AWRR to reach a steady state when there is a shift in traffic patterns. The convergence time is influenced by

Figure 3.16 AWRR with w comparison.



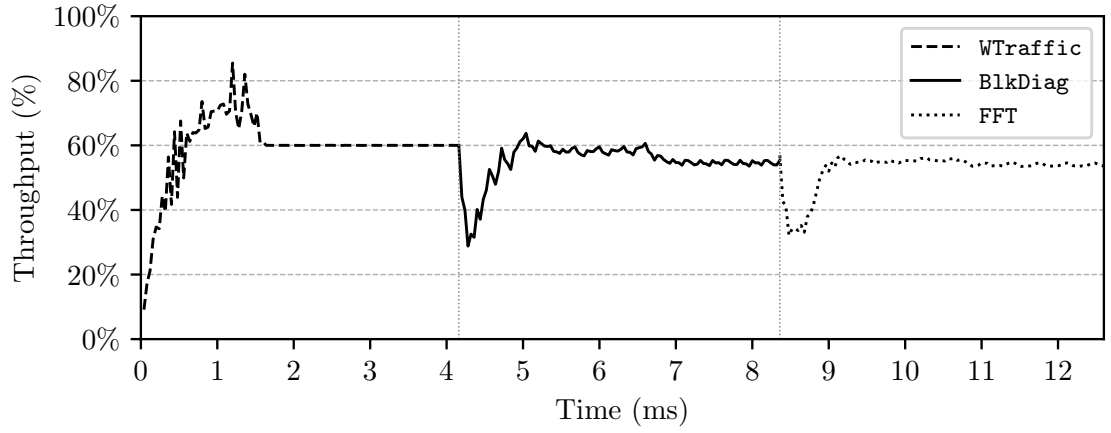
(a) Examination of throughput. **(b)** Examination of average packet latency.

Figure 3.17 AWRR with *wCount* comparison.



(a) Examination of throughput. **(b)** Examination of average packet latency.

Figure 3.18 Convergence behavior for changing traffic patterns.



the parameters described in the AWRR section, primarily the lower threshold for the weights. For the experiment presented in Figure 3.18, the network load is set at 60%.

3.9 Conclusion

Despite the potential benefits of integrating optical switches into Data Center Networks (DCNs), there are several limitations that prevent an efficient and sustainable fully-optical operation with full configurability. These are the limited optical switch port count, the high switch reconfiguration speed, and the need for real-time control plane scheduling. To overcome these issues, we proposed a partially configurable DCN architecture that utilizes Lean switches. These switches use gang-switched optical elements to achieve all-optical partial configurability.

Comparing with a RotorNet network that utilizes Rotor switches with minimal configurability and a Mordia network that employs optical WDM rings and WSSes with full configurability, the Lean network achieves a mediocre crosspoint complexity, allowing for a large port count in the switches and high-speed reconfiguration, while maintaining reduced scheduling time complexity.

The proposed scheduling policy Lean Valiant Decomposition combines the Birkhoff-von Neumann decomposition and the Valiant Load Balancing techniques, allows either direct or indirect transmissions according to the traffic patterns, ensuring that the network achieves throughput that ranges between the cases of the RotorNet and the Mordia networks.

Our simulations involved a comparison of maximum throughput and average packet latencies across three scenarios. In the first scenario, we evaluated different DCNs and their scheduling policies, and found that the Lean network outperformed RotorNet by 26.8% in terms of throughput, while achieving 67.3% of the throughput of the Mordia network. We also observed that direct transmission policies resulted in lower average packet latencies, but the Lean network with the LVD policy achieved at least 28% lower latencies than RotorNet. In the second scenario, we examined different levels of traffic uniformity with the Lean network using the LVD policy, and observed a steady decrease in average packet latency of 21% as the percentile of structured traffic increased. Finally, in the third scenario, we studied different structured traffic patterns with the Lean network using the LVD policy. We found that the Lean network performed similarly in terms of throughput and average packet latency with `WTraffic` and `BlkDiag` patterns. However, with `FFT` pattern, the Lean network achieved 5% less throughput than the other cases and at least 30% increase in average packet latency due to underutilization of direct-only transmissions with the structured traffic patterns of the `FFT`.

Ultimately, we investigated enhancements on RotorNet, including the application of the breakout technique to mitigate latency escalation stemming from network size increase, and also the integration of partial configurability facilitated by a centralized control plane. In this context, we introduced an Adaptive Weighted Round-Robin (AWRR) policy, specifically de-

signed to surpass VLB. AWRR's strength lies in its ability to adapt to traffic characteristics without prior knowledge. It gradually learns traffic patterns and dynamically allocates slots based on evolving weight factors. We conducted a comprehensive examination of various scheduling policies, including a thorough evaluation of AWRR under different traffic profiles.

The results showed that the introduction of breakouts with higher factors ensures that throughput remains stable while latency exhibits a linear decrease, effectively mitigating the latency concerns associated with network scaling. In specific traffic scenarios, like the `WTraffic` profile, AWRR demonstrates exceptional performance. It outperforms VLB by delivering a substantial 30% increase in throughput, all of this achieved without any prior knowledge of the traffic profile. Moreover, AWRR showcases its remarkable adaptability when encountering diverse traffic profiles. It can swiftly adapt to changing conditions, achieving convergence to a stable state in as little as 500 µs.

Chapter 4

Secure Distributed Storage Orchestration on Cloud-Edge Infrastructures

4.1 Introduction and Related Work

Digital transformation has had a significant impact on the storage requirements, which are expected to further increase in the foreseeable future, according to the International Data Corporation (IDC) (37). As cloud computing is the cornerstone of our digital society, businesses prefer to store their data in the cloud rather than deploying their own infrastructure. There are many reasons for a business to prefer storing its data on the cloud instead of privately held storage devices. The advantages obtained in this way include the avoidance of high initial capital expenditure (CAPEX), the scalability of the storage service provided, and the easy migration of the data when needed. Also, a cloud-based storage service provides high availability, exempting a business from the necessity to deploy complex and costly mechanisms for data redundancy and fault-tolerance to power outages and other disaster scenarios.

Distributed storage systems, on the other hand, store data in multiple locations, consolidating resources from multiple providers that, if selected carefully, can offer increased flexibility compared to a single storage service (33). With the advent of edge computing, the storage and processing of the data close to the generating source (e.g., camera, or other sensor) (48) became a reality. The incorporation of edge resources in distributed storage services improves the way demanding applications are served: data are stored and processed at the edge to minimize latency and network usage, and, if additional resources are required, the abundant cloud resources are utilized. The continuous increase in the number and density

of edge resources is expected to change the way current storage services operate. However, this also increases the complexity as the edge resources exhibit diverse characteristics, their availability varies with time, and they are more unreliable (28).

Laying data fragments to remote storage locations where data leaks can happen raises privacy and security concerns for such systems. As storage providers cannot generally be considered trustworthy, sensitive data can be retrieved from encrypted fragments. Since failures are common in distributed systems, redundant data must be stored to tolerate failures without data loss. Erasure coding (70) uses Forward Error Correction (FEC) codes to ensure data integrity and longevity. Data are split, encoded, and incur an overhead depending on the algorithm. Even when some fragments cannot be retrieved, the original data can be efficiently recovered by fetching a subset of fragments, depending on the encoding technique. Erasure coding provides additional security atop encryption, as data can only be recovered when a specific number of fragments from various locations are jointly decoded.The operation of a distributed storage service that integrates edge and cloud resources while utilizing erasure coding to divide data presents a formidable challenge for the corresponding resource orchestration mechanism. In addition to determining the quantity and distribution of data and parity fragments, the orchestrator must also fulfill user demands and optimize various criteria, including monetary cost, latency, and availability.

We formulate the storage resource orchestration as a Mixed-Integer Linear Programming (MILP) problem to obtain the optimal solution. However, the search space can be vast leading to a prohibitively large execution time for the MILP, especially when handling numerous storage requests with strict and heterogeneous requirements. Execution time refers to the duration required for the orchestration mechanism to process storage demands and generate a resource allocation plan for the given scenario. In this work, we propose an efficient multi-agent rollout heuristic approach that is based on reinforcement learning, which balances performance and execution time. This enables fast decision-making in real-world scenarios, reducing the average execution time over that of the optimal MILP algorithm, while maintaining near-optimal performance, as demonstrated in the experiments for which we were able to track the optimal solution. The developed mechanisms use a multitude of optimization criteria, namely, cost, capacity, reliability, performance, availability, or a combination of them when deciding on the data fragmentation, encryption and data placement. The mechanisms also account for the different characteristics of the edge and cloud resources with respect to latency, data availability, security and monetary cost.

To demonstrate the effectiveness of the proposed mechanisms, a series of simulation experiments were performed using anonymous data from Chocolate Cloud. Chocolate Cloud is specialized in secure and distributed data storage software and its flagship product, SkyFlok

(sky), is a Software-as-a-Service (SaaS) file sharing and storage solution. The proposed mechanisms enhance the orchestration logic of the SkyFlok platform, allowing it to incorporate edge resources efficiently. This means that the mechanisms can be integrated into the back-end control and orchestration mechanisms of the distributed service, enabling it to coordinate the encryption, erasure coding and distribution of data fragments across the selected cloud and edge location.

The storage allocation problem has long attracted the interest of many researchers. To address the limitations of single cloud models, multi-cloud resource allocation schemes were initially examined (33, 60) and (49). In (33), Hadji proposed a solution based on commodity flows to minimize the storage monetary cost and latency. Papaioannou et al. (60) proposed Scalia, a cloud storage brokerage solution for data placement that targets to minimize the storage monetary cost. Mansouri et al. (52) proposed an algorithm that minimizes the storage monetary cost, guaranteeing at the same time high data availability and privacy.

Ma et al. (51) proposed a mixed policy that is based on a combination of erasure and replication coding techniques, targeting to minimize latency, as well as storage and network monetary costs. In the same context, Zhang et al. (86) proposed a sub-optimal multi-agent heuristic approach for selecting the storage locations and the appropriate redundancy configuration to minimize the monetary cost with respect to the user's latency and availability requirements. Wu et al. (83) proposed a scheme that trades-off cost for latency, meeting the preset availability requirements. Liu et al. (49) proposed a heuristic (genetic) algorithm to minimize costs while providing Service Level guarantees.

Targeting the experienced latency minimization, Sharov et al. (69) proposed a quorum-based configuration that makes use of replication coding and assigns the fragments to the different locations. Bermbach et al. (16) examined the consistency versus latency trade-off making use of a mechanism from Amazon's Dynamo for replication to multiple cloud providers. Other works, such as (17, 33, 52, 81), rely on replication to multiple providers in order to attain higher availability and avoid vendor lock-ins, while keeping the cost low. However, the use of replication instead of erasure coding does not address the problem of the variations in latency that are experienced by the user.

Other works have proposed mechanisms that improve data availability through redundancy, also minimizing the monetary cost incurred. However, these works rely on replication coding, which requires more storage space, compared to erasure coding. In (6, 58, 60, 86), the authors proposed mechanisms that make use of erasure coding solutions to improve data availability. In this direction, Wang et al. (78, 80) proposed various techniques that minimize the monetary cost while maximizing the availability. Su et al. (73) proposed an erasure coding model for solving the data placement problem. Wang et al. (79) proposed an adaptive

model for data placement that minimizes the monetary cost but also takes into consideration the latency and availability constraints.

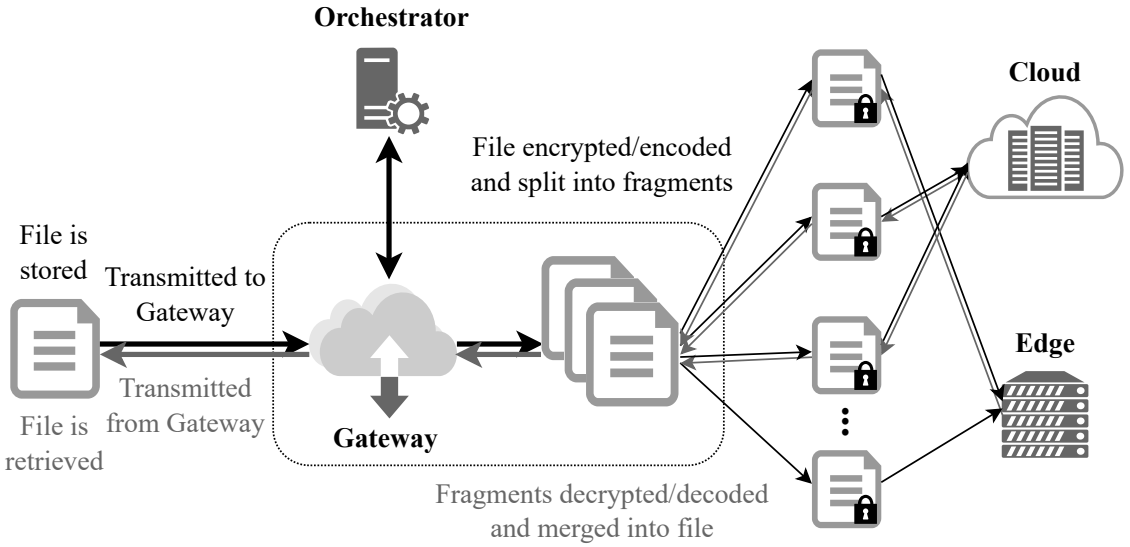
In (26), the authors address cloud plan selection by users, introducing a simple language to express user requirements and plan preferences, and propose a model for identifying and ranking the plans that satisfy the requirements. In (27), the same authors extend this work, allocating resources from multiple cloud services. Users can specify allocation requirements to reduce burden and avoid excessive fragmentation. The authors provide an integer programming formulation for finding an allocation satisfying protection and allocation requirements while minimizing costs. (11) adopts All-Or-Nothing-Transform (AONT) and data replication, introducing two strategies for allocating shards to nodes. The analysis of these allocation strategies illustrates tuning to balance availability and security. In (10), the authors address the dynamic version of the problem, relying on fountain codes instead of replication.

Previous works have focused on cloud storage and optimizing individual objectives, such as data availability, latency, and cost. In contrast, (50) and (7) propose caching schemes with erasure code for low latency in distributed storage systems that span across the edge-cloud continuum. The proposed scheme caches popular data chunks at edge servers to achieve low latencies, but costs and availability of storage resources are not optimized. Authors in (71) propose a location aware to optimize the data retrieval latency in ultra-large distributed storage systems, while the authors in (25) propose a rights Management Protocol to enable the sharing of files in distributed storage systems consisting of nodes that are not fully trusted.

In the current work, we extend the storage resource allocation problem considering the intrinsic characteristics of a distributed storage infrastructure that spans over the edge-cloud continuum. Hence, in addition to cloud resources, we consider edge resources, which are located closer to where the data are generated (82, 85) and have limited storage capacity and highly dynamic availability. When edge and cloud resources are allocated, leveraging the erasure code technique, different optimization criteria can be addressed simultaneously.

The research results of this chapter were published in (46) winning a Best Paper Award, and (45). The remainder of our work is organized as follows. In Section 4.2, we discuss on the infrastructure and the distributed storage operations. In Section 4.3, we provide the resource allocation policies, while in Section 4.4 we present the simulation results.

Figure 4.1 Components and data flow.



4.2 Distributed Storage Infrastructure and Operations

A distributed storage service utilizes storage resources that can be classified, based on their location, into two broad categorizes: (i) cloud and (ii) edge resources. Cloud resources comprise vast data centers boasting substantial storage capacity and excellent uptime. However, their centralized nature and distance from data generation sites often results in higher perceived latency for store and retrieve operations. Conversely, edge nodes possess limited storage capacity (85), exhibiting dynamic participation in the distributed storage system and are often considered less reliable (28). However, their widespread presence across various locations allows them to be situated closer to data sources, substantially enhancing the quality of service offered. It is important to note that file hosting charges differ based on the characteristics of the cloud or edge node, with cloud node charges commonly being lower than edge node charges (edg).

To handle crucial operations like file encryption and decryption, as well as splitting and merging data fragments (Figure 4.1), a typical distributed storage infrastructure relies not only on storage nodes, but also on processing power. These essential devices are commonly referred to as *gateways*, serving as the crucial link between users and the distributed storage service. Gateways can be implemented through software, such as pieces of code that run within users' client software (e.g., browsers), or through hardware, such as privately owned dedicated devices that are usually placed on-premise.

The operations executed within a distributed storage infrastructure can generally be classified as: (i) Data processing operations, including file encoding/decoding and merging/splitting,

(ii) Store operations necessary to store the encoded fragments in storage nodes, and (iii) Retrieve operations necessary to reconstruct the user's files. When a file d is stored, a monetary cost (measured in Cost Units - CU) and a store/retrieve operation latency are incurred. These factors depend on the file's (i) size M_d (measured in Data Units - DU), (ii) hosting duration T_d (measured in Period Units - PU), and (iii) set of expected retrievals $\mathcal{T}_d$ within the hosting duration.

When a file d is stored on a distributed storage infrastructure, it is split into k data fragments at the gateway. These data fragments are expanded into $(k + m)$ encoded fragments using erasure coding, which is a data protection method commonly used in storage systems to ensure data integrity and availability in the event of failures. This allows the initial file to be successfully retrieved from a subset $\hat{k}$ of the encoded fragments. The value of ϵ indicates the number of tolerated failures, and varies depending on the erasure coding scheme used. Optimal schemes, such as Reed Solomon, Parity, and Regenerating codes, have $\epsilon = 0$, while near-optimal schemes, such as fountain codes, have $\epsilon > 0$. However, reducing ϵ increases the encoding/decoding latency of files for a given gateway.

4.2.1 Store/Retrieve Data Processing Operation

When a file d is split at a gateway ω_d, the split latency is denoted by $D^{\text{spl}}_{\omega_d}$ (in TUs/split), and the latency of encoding/encryption is $D^{\text{enc}}_{\omega_d}$ (in TUs per DU). The overall latency for the $k : k + m$ *store* operation is calculated as:

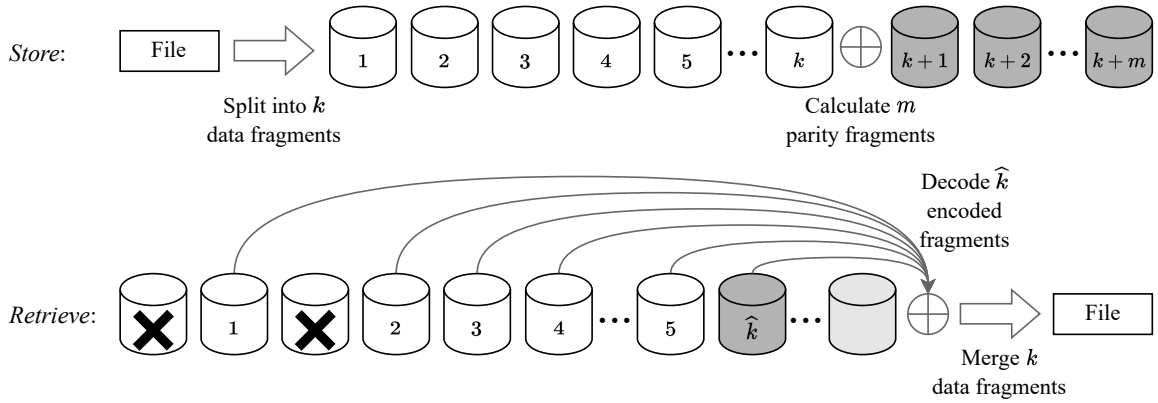
$$G_{kmd} = \frac{k+m}{k} M_d D^{\text{enc}}_{\omega_d} + k D^{\text{spl}}_{\omega_d}. \tag{4.1}$$

For the retrieval t of a file d, decryption and decoding of the fragments are performed at a gateway ω'_{dt}. A subset of $\hat{k}$ fragments is then linearly combined, and the decoded fragments are merged into the file d. The latency of decoding/decryption per DU is denoted by $D^{\text{dec}}_{\omega_d}$ (in TUs per DU), and the overall latency is proportional to the file size M_d. The latency of merging per DU, denoted by $D^{\text{mrg}}_{\omega_d}$ (in TUs per DU merged), and the total merging latency is proportional to $\hat{k}$. Therefore, the overall latency of the *retrieve* operation is derived as:

$$G'_{kmdt} = \frac{\hat{k}}{k} M_d D^{\text{dec}}_{\omega'_{dt}} + k D^{\text{mrg}}_{\omega'_{dt}}. \tag{4.2}$$

This process is illustrated in Figure 4.2. The number of fragments and erasure codes used can be adjusted, creating a trade-off between the number of fragments and the associated overhead. The storage allocation mechanism can exploit this trade-off to further enhance infrastructure performance.

Figure 4.2 Store/retrieve data processing.



4.2.2 Store Operation

Store Operation Monetary Cost

The encoded fragments of a file are stored at various locations in the available edge/cloud infrastructure (as shown in Figure 4.3). Let $\mathcal{N}_s$ denote the selected nodes hosting these fragments, with $|\mathcal{N}_s| = k + m$. The monetary cost of storing the fragments of file d, assuming that each storage node $n \in \mathcal{N}_s$ charges P_{sn} CUs for each DU, can be calculated as:

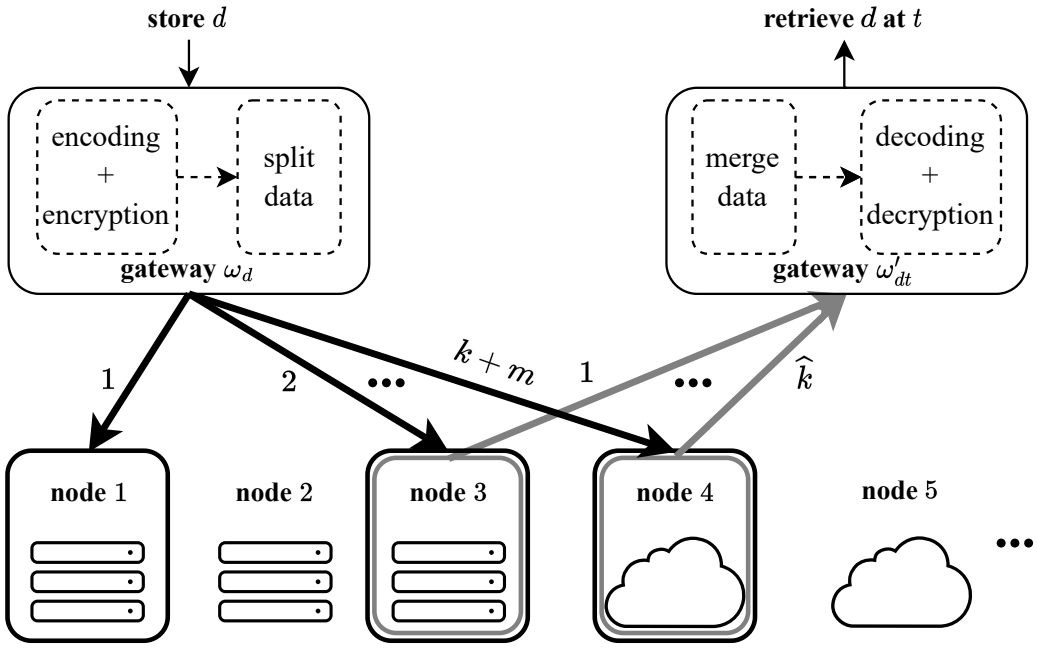
$$\phi_1(k, m, d) = \sum_{n \in \mathcal{N}_s} \frac{M_d}{k} T_d P_{sn}. \tag{4.3}$$

The latency of the store operation comprises three components: (i) *processing latency*, (ii) *propagation latency*, and (iii) *writing latency*, which are required for placing the fragments into the storage nodes.

As discussed in Section 4.2.1, the processing of a data fragment at a gateway requires time equal to G_{kmd}. Next, transmitting an encoded fragment from a gateway ω_d to a storage node n requires a propagation latency of $D^{\text{prop}}_{n\omega_d}$ TUs, which depends on the relative distance between the two components. Finally, placing a fragment into a storage node requires a writing latency D^{wrt}_n in TUs per DU. Given the set of nodes $\mathcal{N}_s$ where the encoded fragments are stored, the total latency for storing a file d can be calculated as:

$$\phi_2(k, m, d) = G_{kmd} + \max_{n \in \mathcal{N}_s} \left\{ \frac{M_d}{k} D^{\text{wrt}}_n + D^{\text{prop}}_{n\omega_d} \right\} \tag{4.4}$$

Figure 4.3 Store and retrieve operation.



4.2.3 Retrieve Operation

Retrieve Operation Monetary Cost

The fragments are retrieved from a subset of storage nodes $\mathcal{N}_{\mathrm{r}}^{t} \subseteq \mathcal{N}_{\mathrm{s}}$ at retrieval t, where $|\mathcal{N}_{\mathrm{r}}^{t}| = \hat{k}$ (Figure 4.3). The selected set of nodes determines the cost of a retrieve operation, which may involve multiple GET requests. Each storage node $n \in \mathcal{N}_{\mathrm{r}}^{t}$ charges the user based on the number of GET requests required to retrieve the entire fragment. Assuming that each GET request retrieves ρ DUs and costs $P_{\mathrm{r}n}$ CUs, we can calculate the monetary cost for the retrieve operation as:

$$\phi_3(k, m, d, t) = \sum_{n \in \mathcal{N}_{\mathrm{r}}^{t}} \frac{1}{\rho} \frac{M_d}{k} P_{\mathrm{r}n}. \tag{4.5}$$

Retrieve Operation Latency

Similarly to the store operation latency, a retrieve operation t involves latency that includes: (i) *processing latency*, (ii) *propagation latency*, and (iii) *reading latency* for recovering fragments from the storage nodes.

In particular, the latency for recovering an encoded fragment from a storage node n requires a reading time of D_n^{read} TUs per DU. Next, the data propagation between a storage node n and a gateway ω'_{dt} introduces a propagation latency of $D_{n\omega'_{dt}}^{\mathrm{prop}}$, which depends on the

distance between the two components. Finally, a processing latency of G'_{kmdt} TUs is introduced at the gateway ω'_{dt} for decrypting and decoding the $\hat{k}$ fragments and merging them into the initial file d (as discussed in Section 4.2.1). Assuming that the storage allocation mechanism selects a set of nodes $\mathcal{N}_{\mathrm{r}}^{t}$ for retrieving the fragments, the total latency for the retrieve operation can be calculated as:

$$\phi_4(k, m, d, t) = G'_{kmdt} + \max_{n \in \mathcal{N}_{\mathrm{r}}^{t}} \left\{ \frac{M_d}{k} D_n^{\mathrm{read}} + D_{n\omega'_{dt}}^{\mathrm{prop}} \right\}. \tag{4.6}$$

4.3 Distributed Storage Resource Allocation

We consider a distributed storage infrastructure comprising a set $\mathcal{N}$ of storage nodes, which can range from cloud data centers to edge mini-data centers and workstations. Each node $n \in \mathcal{N}$ contributes a writing latency $\mathcal{D}_n^{\text{wrt}}$ and a reading latency $\mathcal{D}_n^{\text{read}}$ measured in TUs for each DU that is stored or retrieved, respectively. Additionally, each node periodically charges P_{sn} CU per DU for hosting a file, for a total duration of PU periods. A set of gateways is also present in the storage infrastructure, responsible for splitting, encrypting, decrypting, and merging files during storage and retrieval. Propagation latency is introduced when transferring files to and from the storage nodes, which depends on the relative distance between the components.

The storage resource allocation problem involves deciding how a set of files $\mathcal{D}$ is to be served, while minimizing the optimization criteria. Each file $d \in \mathcal{D}$ is described by the tuple $(M_d, T_d, \mathcal{T}_d, \omega_d, \omega'_{dt}, \mathcal{Q}_d, \mathcal{K}_d, \mathcal{M}_d, \epsilon)$, for $t \in \mathcal{T}_d$, where T_d is the hosting duration (in PUs), $\mathcal{T}_d$ is the set of future retrievals (an offline problem), ω_d is the gateway where d is processed during the store operation, ω'_{dt} is the gateway used for retrieve operation $t \in \mathcal{T}_d$, $\mathcal{Q}_d$ is the set of Quality of Service (QoS) requirements, $\mathcal{K}_d$ is the set of fragmentation options, $\mathcal{M}_d$ is the set of encoding schemes, and ϵ is the optimality factor of the erasure code scheme.

To optimize the quality of service, the resource allocation mechanism examines various fragmentation and encoding options for each file at the gateway. The optimal combination is selected based on preset criteria, which include:

i The number of data fragments each file should be split into (k), selected from the set of fragmentation options ($\mathcal{K}_d$).

ii The type of erasure code to be used, which adds parity fragments for redundancy (m), selected from the set of encoding schemes ($\mathcal{M}_d$).

iii The set of storage nodes where the encoded fragments will be stored ($\mathcal{N}_{\text{s}}$).

iv The set of storage nodes from which the fragments will be retrieved at retrieval t ($\mathcal{N}_{\text{r}}^t$), to minimize the weighted cost resulting from different optimization criteria.

After the optimal options have been determined, each file is split into the chosen number of data fragments, and parity fragments are added using the selected erasure code. The total number of encoded fragments is then $k + m$.

4.3.1 Pre-processing Phase - Availability

To optimize the availability of a file d after encoding, a pre-processing phase is performed in which all possible combinations of $k+m$ storage nodes to host the encoded fragments are computed, for all $k \in \mathscr{K}_d$ and $m \in \mathscr{M}_d$. We assume that each storage node can only host one fragment of each file, and that each node $n \in \mathscr{N}$ has an availability probability A_n. There are $\Phi_{km} = \binom{|\mathscr{N}|}{k+m}$ combinations of $k+m$ nodes, which we denote by $\mathscr{J}_{1km}, \mathscr{J}_{2km}, \ldots, \mathscr{J}_{\Phi_{km}}$. Since up to $k+m-\hat{k}$ node failures can be tolerated, the availability of the encoded file is calculated by summing all the probabilities that $\kappa \in [\hat{k}, k+m]$ nodes are functioning and accessible. We denote the number of collections of κ available nodes as $\Theta = \binom{|\mathscr{J}_{ikm}|}{\kappa}$ and the j^{th} collection as S_j^{Θ}. Assuming that nodes fail independently, the availability of a file hosted by the storage nodes of set $\mathscr{J}_{ikm}$ is given by:

$$a_{ikm} = \sum_{\kappa=\hat{k}}^{k+m} \sum_{j=1}^{\Theta} \Big[\prod_{n \in S_j^{\Theta}} A_n \prod_{n \in \mathscr{J}_{ikm} \setminus S_j^{\Theta}} (1 - A_n) \Big], \tag{4.7}$$

where $\mathscr{J}_{ikm} \setminus S_j^{\Theta}$ denotes the unavailable nodes. We then discard combinations that do not meet the minimum availability requirement $A_{\text{req}} \in \mathscr{Q}_d$, and denote the indices of the remaining combinations as $\mathscr{I}_{km} = \{ i : a_{ikm} \geq A_{\text{req}} \in \mathscr{Q}_d \wedge i = 1, 2, \ldots, \Phi_{km} \}$, for all $k \in \mathscr{K}_d$ and $m \in \mathscr{M}_d$.

Thus, the average combined availability of a set of nodes $\mathscr{J}_{ikm}$, with given k and m, is defined as:

$$\phi_5(i, k, m) = a_{ikm}. \tag{4.8}$$

In this study, we consider the offline version of the storage resource allocation problem. Therefore, based on the availability probability A_n of each storage node n, we define a sequence of binary random variables $H[n, d, t] \sim Bernoulli(A_n)$, which determines whether the node n is available during a retrieve operation t of file d, for all $n \in \mathscr{N}$, $d \in \mathscr{D}$, and $t \in \mathscr{T}_d$. The related retrieve operation costs of t are then calculated based on these node availabilities.

4.3.2 Mixed-Integer Linear Programming Formulation

In this section, we present the MILP formulation of the distributed storage resource allocation mechanism.

The objectives under examination are: (i) the monetary cost of store and retrieve operations, (ii) the latency of store and retrieve operations, and (iii) the successful retrievals of

the files (availability). To calculate these objectives, we use two parameters, U and U', to represent the maximum propagation and writing delay and the maximum propagation and reading delay, respectively, in a given infrastructure. These parameters are large positive numbers that are computed as follows (Eq. (4.9) and Eq. (4.10)).

$$U = \max_{\substack{d,n:d\in\mathscr{D},\\ n\in\mathscr{N}}} \left\{ \max_{k:k\in\mathscr{K}_d} \left\{ \frac{M_d}{k} D_n^{\text{wrt}} \right\} + D_{n\omega_d}^{\text{prop}} \right\}, \tag{4.9}$$

and

$$U' = \max_{\substack{d,n:d\in\mathscr{D},\\ n\in\mathscr{N}}} \left\{ \max_{k:k\in\mathscr{K}_d} \left\{ \frac{M_d}{k} D_n^{\text{read}} \right\} + \max_{t:t\in\mathscr{T}_d} \left\{ D_{n\omega'_{dt}}^{\text{prop}} \right\} \right\}. \tag{4.10}$$

To indicate whether a node n belongs to a particular combination of nodes (as described in Section 4.3.1), we define the following function:

$$\theta(i,k,m,n) = \begin{cases} 1, \text{if } n \in \mathscr{J}_{ikm} \\ 0, \text{otherwise.} \end{cases} \tag{4.11}$$

The MILP variables are given in Table 4.1.

min $\left[w_1\bar{\phi}_1, w_2\bar{\phi}_2, w_3\bar{\phi}_3, w_4\bar{\phi}_4, w_5\bar{\phi}_5, \sum_{d\in\mathscr{D}} \psi_d, \sum_{d\in\mathscr{D}} \sum_{t\in\mathscr{T}_d} \psi'_{dt} \right]$

where $\bar{\phi}_1 := \sum_{n\in\mathscr{N}} \sum_{d\in\mathscr{D}} \sum_{k\in\mathscr{K}_d} \sum_{m\in\mathscr{M}_d} \frac{M_d}{k} T_d P_{sn} x_{nkmd}$

$\bar{\phi}_2 := \sum_{d\in\mathscr{D}} \sum_{k\in\mathscr{K}_d} \sum_{m\in\mathscr{M}_d} G_{kmd} y_{kmd} + \sum_{d\in\mathscr{D}} \psi_d$

$\bar{\phi}_3 := \sum_{n\in\mathscr{N}} \sum_{d\in\mathscr{D}} \sum_{k\in\mathscr{K}_d} \sum_{m\in\mathscr{M}_d} \sum_{t\in\mathscr{T}_d} \frac{1}{\rho} \frac{M_d}{k} P_{rn} x'_{nkmdt}$

$\bar{\phi}_4 := \sum_{d\in\mathscr{D}} \sum_{k\in\mathscr{K}_d} \sum_{m\in\mathscr{M}_d} \sum_{t\in\mathscr{T}_d} G'_{kmdt} y_{kmd} + \sum_{d\in\mathscr{D}} \sum_{t\in\mathscr{T}_d} \psi'_{dt}$

$\bar{\phi}_5 := -\sum_{d\in\mathscr{D}} \sum_{k\in\mathscr{K}_d} \sum_{m\in\mathscr{M}_d} \sum_{i\in\mathscr{I}_{km}} a_{ikm} z_{ikmd}$

subject to:

C1 $\sum_{k\in\mathscr{K}_d} \sum_{m\in\mathscr{M}_d} \frac{M_d}{k} D_n^{\text{wrt}} x_{nkmd} +$

$\sum_{k\in\mathscr{K}_d} \sum_{m\in\mathscr{M}_d} D_{n\omega_d}^{\text{prop}} x_{nkmd} - \psi_d \leq 0, n \in \mathscr{N}, d \in \mathscr{D}$

Table 4.1 Definition of MILP variables.

Var.	Definition
$\hat{k}$	Integer variable describing the required fragments to recover a file, equal to $(1+\epsilon)k$.
v_{nd}	Binary variable indicating whether a fragment of file d is placed at a node n.
y_{kmd}	Binary variable indicating the fragmentation option k and the erasure code $(k+m,\hat{k})$ of file d.
x_{nkmd}	Binary variable indicating that a fragment of d is stored to node n, while the fragmentation option is k and the erasure code is $(k+m,\hat{k})$.
x'_{nkmdt}	Binary variable indicating that a fragment of d is retrieved from node n during the retrieval t, while the fragmentation option is k and the erasure code is $(k+m,\hat{k})$.
ξ_{nd}	Binary variable indicating the fragment and the storage node n with the maximum propagation and writing latency of file d.
ξ'_{ndt}	Binary variable indicating the fragment and the storage node n with the maximum reading and propagation latency of the retrieve operation t of file d.
ψ_d	Integer variable denoting the maximum propagation and writing latency of file d.
ψ'_{dt}	Integer variable denoting the maximum reading and propagation latency of the retrieve operation t of file d.
z_{ikmd}	Binary variable indicating the combination $i \in \mathcal{I}_{km}$ of storage nodes, the fragmentation option k and the erasure code $(k+m,\hat{k})$ of file d.

$$\text{C2} \quad -\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}\frac{M_d}{k}D_n^{\text{wrt}}x_{nkmd}-\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}D_{n\omega_d}^{\text{prop}}x_{nkmd}+ U\xi_{nd}+\psi_d\le U, \qquad n\in\mathcal{N}, d\in\mathcal{D}$$

$$\text{C3} \quad \sum_{n\in\mathcal{N}}\xi_{nd}=1, \qquad d\in\mathcal{D}$$

$$\text{C4} \quad \sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}\frac{M_d}{k}D_n^{\text{read}}x'_{nkmdt}+\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}D_{n\omega'_{dt}}^{\text{prop}}x'_{nkmdt}- \psi'_{dt}\le 0, \qquad n\in\mathcal{N}, d\in\mathcal{D}, t\in\mathcal{T}_d$$

$$\text{C5} \quad -\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}\frac{M_d}{k}D_n^{\text{read}}x'_{nkmdt}-\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}D_{n\omega'_{dt}}^{\text{prop}}x'_{nkmdt}+ U'\xi'_{ndt}+\psi'_{dt}\le U', \qquad n\in\mathcal{N}, d\in\mathcal{D}, t\in\mathcal{T}_d$$

$$\text{C6} \quad \sum_{n\in\mathcal{N}}\xi'_{ndt}=1, \qquad d\in\mathcal{D}, t\in\mathcal{T}_d$$

$$\text{C7} \quad \sum_{d\in\mathcal{D}}\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}\frac{M_d}{k}x_{nkmd}\le C_n, \qquad n\in\mathcal{N}$$

C8 $$\sum_{n\in\mathcal{N}} v_{nd} = \sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}(k+m)y_{kmd}, \qquad d\in\mathcal{D}$$

C9 $$\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d} y_{kmd} = 1, \qquad d\in\mathcal{D}$$

C10 $$\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d} x'_{nkmdt} = \sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}\hat{k}y_{kmd}, \quad d\in\mathcal{D}, t\in\mathcal{T}_d$$

C11 $$x'_{nkmdt} \le x_{nkmd}, \quad n\in\mathcal{N}, d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d, t\in\mathcal{T}_d$$

C12 $$-x_{nkmd} + v_{nd} + y_{kmd} \le 1, n\in\mathcal{N}, d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d$$

C13 $$x_{nkmd} - v_{nd} \le 0, \qquad n\in\mathcal{N}, d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d$$

C14 $$x_{nkmd} - y_{kmd} \le 0, \qquad n\in\mathcal{N}, d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d$$

C15 $$-\sum_{n\in\mathcal{N}}\theta(i,k,m,n)x_{nkmd} + \sum_{n\in\mathcal{N}}\theta(i,k,m,n)z_{ikmd} \le 0,$$
$$d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d, i\in\mathcal{I}_{km}$$

C16 $$\sum_{n\in\mathcal{N}}\theta(i,k,m,n)x_{nkmd} - \sum_{n\in\mathcal{N}}\theta(i,k,m,n)z_{ikmd} \le$$
$$(1-z_{ikmd})|\mathcal{N}|, d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d, i\in\mathcal{I}_{km}$$

C17 $$\sum_{i\in\mathcal{I}_{km}} z_{ikmd} = y_{kmd}, \qquad d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d$$

C18 $$\sum_{i=1}^{5} w_i = 1.0$$

To calculate the latency when the fragments are stored at different storage nodes, we use constraints C1-C3. Similarly, to calculate the latency of the retrieve operation, we use constraints C4-C6. Constraint C7 ensures that the fragments are placed based on the available storage capacity, while constraint C8 ensures that all fragments are hosted by the infrastructure. To ensure that each file undergoes a selection of the number of split data fragments and the erasure code, we use constraint C9. Constraints C10-C14 express that the minimum number of encoded fragments is retrieved along with their respective storage nodes. To determine the nodes where the total number of encoded fragments for a specific file are placed, we use constraints C15-C17. Finally, constraint C18 is a weighting coefficient-related constraint.

We examine the number of variables and constraints required by the MILP formulation. The number of retrievals of each file $d \in \mathcal{D}$ is bounded by $|\mathcal{T}| = \max_{d:d\in\mathcal{D}} |\mathcal{T}_d|$. Each

file can be split into a maximum of $|\mathscr{K}| = \max_{d:d\in\mathscr{D}} |\mathscr{K}_d|$ options for the number of data fragments and a maximum of $|\mathscr{M}| = \max_{d:d\in\mathscr{D}} |\mathscr{M}_d|$ options for the number of additional parity ones. All the encoded fragments are stored over a storage infrastructure consisting of $|\mathscr{N}|$ nodes. For each file, there are a total of $|\mathscr{I}|$ combinations of storing the fragments.

The MILP formulation requires $|\mathscr{D}| \cdot (|\mathscr{N}|(|\mathscr{T}|+2) + (2|\mathscr{N}| + |\mathscr{I}| + 1) \cdot |\mathscr{K}| \cdot |\mathscr{M}|)$ variables. It also requires the following inequality constraints: $|\mathscr{N}| \cdot |\mathscr{D}|$ for C1-C2, $|\mathscr{D}|$ for C3, $|\mathscr{N}| \cdot |\mathscr{D}| \cdot |\mathscr{T}|$ for C4-C5, $|\mathscr{D}| \cdot |\mathscr{T}|$ for C6, $|\mathscr{N}|$ for C7, $|\mathscr{D}|$ for C8-C9, $|\mathscr{D}| \cdot |\mathscr{T}|$ for C10, $|\mathscr{N}| \cdot |\mathscr{D}| \cdot |\mathscr{K}| \cdot |\mathscr{M}| \cdot |\mathscr{T}|$ for C11, $|\mathscr{N}| \cdot |\mathscr{D}| \cdot |\mathscr{K}| \cdot |\mathscr{M}|$ for C12-C14, $|\mathscr{D}| \cdot |\mathscr{K}| \cdot |\mathscr{M}| \cdot |\mathscr{I}|$ for C15-C16, $|\mathscr{D}| \cdot |\mathscr{K}| \cdot |\mathscr{M}|$ for C17, and 1 for C18. Hence, the total number of constraints in the MILP formulation is derived as $(2|\mathscr{D}| \cdot (|\mathscr{T}|+1)+1) \cdot (|\mathscr{N}|+1) + |\mathscr{D}| + |\mathscr{D}| \cdot |\mathscr{K}| \cdot |\mathscr{M}| \cdot ((|\mathscr{T}|+3) \cdot |\mathscr{N}| + 2|\mathscr{I}| + 1)$.

4.3.3 Multi-Agent Rollout Heuristic Algorithm

For large instances, the optimal MILP mechanism can exhibit a high execution time, making it less practical when fast placement and retrieval of fragments decisions are required for numerous files. To address this limitation, we developed a multi-agent rollout algorithm that solves the respective resource allocation problem sequentially, with one-at-a-time agent controlling the selection made. The complete solution is provided in stages and is built by extending the partial solution from the previous stages. At each stage, one of the available options is selected for storing the examined file, while the rest of the unallocated resources are handled using the base-heuristic Algorithm 8. At the end of each round, the resource allocation with the minimum cost for each file is selected, based on the exhibited cost.

The pseudocode for the proposed mechanism is presented in Algorithms 6-9. The main body of the multi-agent rollout heuristic is described in Algorithm 6, while Algorithm 7 is used to calculate the costs of a single storage node in case it is selected to host an encoded fragment.

The rollout algorithm relies on a base-heuristic (Algorithm 8) to handle the current resource utilization and serve the demands sequentially by placing encoded fragments in the first available combination of storage locations that can accommodate them. To calculate the total cost of serving a single demand provided by the base heuristic, the algorithm uses Algorithm 9, which considers decisions on storage nodes, fragmentation, and erasure code options. By serving demands one-by-one, the base heuristic yields a complete assignment of files to storage resources, extending the provided partial solution with the decisions made by the first fit approach.

To examine the computational complexity of the multi-agent rollout heuristic, we introduce some notation. Let $\mathscr{K}$ and $\mathscr{M}$ be the sets with the maximum sizes among $\mathscr{K}_d$ and $\mathscr{M}_d$,

Algorithm 6 Multi-Agent Rollout Heuristic

Input: $\mathbf{w}, \mathcal{N}, \mathcal{D}, \rho, \epsilon, \mathbf{P}^{\mathrm{s}}, \mathbf{P}^{\mathrm{r}}, \mathbf{D}^{\mathrm{read}}, \mathbf{D}^{\mathrm{wrt}}, \mathbf{A}, \mathscr{K}, \mathscr{M}, \mathscr{Q}, \mathbf{D}^{\mathrm{prop}}, \boldsymbol{\omega}, \boldsymbol{\omega}', \mathscr{T}, \mathbf{D}^{\mathrm{spl}}, \mathbf{D}^{\mathrm{mrg}}, \mathbf{D}^{\mathrm{enc}}, \mathbf{D}^{\mathrm{dec}}$
Output: $totalSolution_{|\mathcal{D}|}$ and $totalCost$

$nodePar \leftarrow (\mathbf{P}^{\mathrm{s}}, \mathbf{P}^{\mathrm{r}}, \mathbf{D}^{\mathrm{read}}, \mathbf{D}^{\mathrm{wrt}}, \mathbf{A}, \mathbf{D}^{\mathrm{prop}})$; $filePar \leftarrow (\mathbf{D}^{\mathrm{spl}}, \mathbf{D}^{\mathrm{mrg}}, \mathbf{D}^{\mathrm{enc}}, \mathbf{D}^{\mathrm{dec}})$
$cost \leftarrow$ array of size $|\mathcal{N}| \times |\mathcal{D}|$
$solution \leftarrow$ array of size $|\mathcal{D}|$; $totalCost \leftarrow 0$; $\mathcal{D}^* \leftarrow \{\}$
for $d \in \mathcal{D}$ **do**
  $cdtAlloc[d] \leftarrow \{\}$
  for $k \in \mathscr{K}_d, m \in \mathscr{M}_d$ **do**
    $CS \leftarrow \{\}$; $\hat{k} \leftarrow \min(\lceil (1+\epsilon)k \rceil, k+m)$
    for $n \in \mathcal{N}$ **do**
      if $C_n \geq M_d$ **then**
        $cost[n,d] \leftarrow$ CalcNodeCost$(n, d, k, \rho, M_d, T_d, \mathscr{T}_d, \boldsymbol{\omega}, \boldsymbol{\omega}', nodePar)$
        $CS \leftarrow CS \cup \{n\}$
      end if
    end for
    $\overline{CS} \leftarrow$ Sort CS according to $cost$
    $\overline{CS} \leftarrow$ Select from $\overline{CS}$ the first $2(k+m)$ elements
    $i \leftarrow 0$
    $\mathcal{N}^* \leftarrow \{\overline{CS}[1+i], \overline{CS}[2+i], \ldots, \overline{CS}[k+m+i]\}$
    $avail \leftarrow$ given $\mathbf{A}$, calculate combined availability of $\mathcal{N}^*$
    while $avail < A_{\mathrm{req}}$ and $i < k+m-1$ **do**
      $i \leftarrow i+1$
      $\mathcal{N}^* \leftarrow \{\overline{CS}[1+i], \overline{CS}[2+i], \ldots, \overline{CS}[k+m+i]\}$
      $avail \leftarrow$ given $\mathbf{A}$, calculate combined availability of $\mathcal{N}^*$
    end while
    if $avail \geq A_{\mathrm{req}}$ **then**
      $\mathscr{T}^* \leftarrow \{\}$
      for $t \in \mathscr{T}_d$ **do**
        $CR \leftarrow$ sort $\mathcal{N}^*$ according to $cost$
        $\mathscr{T}^* \leftarrow \mathscr{T}^* \cup \{(CR[1], t), (CR[2], t), \ldots, (CR[\hat{k}], t)\}$
      end for
      break
    else
      $avail \leftarrow \infty$
    end if
    $cdtAlloc[d] \leftarrow cdtAlloc[d] \cup \{(k, m, \mathcal{N}^*, \mathscr{T}^*)\}$
  end for
end for
for $d \in \mathcal{D}$ **do**
  $\mathcal{D}^* \leftarrow \mathcal{D}^* \cup \{d\}$
  $tCost^* \leftarrow \infty$
  for $(k, m, \mathcal{N}^*, \mathscr{T}^*) \in cdtAlloc[d]$ **do**
    $c^* \leftarrow$ CalcFileCost$(\mathbf{w}, d, k, \hat{k}, m, M, T, \mathscr{T}_d, \boldsymbol{\omega}, \boldsymbol{\omega}', \mathcal{N}^*, \mathscr{T}^*, cost, avail, nodePar, filePar)$
    $tCost \leftarrow$ BaseHeuristic$(\mathbf{w}, \mathcal{D}, \mathcal{D}^*, \mathcal{N}, c^*, \mathbf{M}, \mathbf{T}, \mathscr{T}, \boldsymbol{\omega}, \boldsymbol{\omega}', cdtAlloc, C, totalCost, \ldots$
      $cost, avail, nodePar, filePar)$
    if $tCost < tCost^*$ **then**
      $tCost^* \leftarrow tCost$
      $tSolution^* \leftarrow (k, m, \mathcal{N}^*, \mathscr{T}^*)$
    end if
  end for
  $totalSolution \leftarrow totalSolution \cup \{tSolution^*\}$
  Update C according to $tSolution^*$
  Calculate $totalCost$ according to $tCost^*$
end for

Algorithm 7

function CALCNODECOST($n, d, k, \rho, M, T, \mathcal{T}, \boldsymbol{\omega}, \boldsymbol{\omega}', nodePar$)
  $(P_{sn}, P_{rn}, D_n^{\text{wrt}}, D_n^{\text{read}}, A_n) \leftarrow nodePar[n]$
  $hostStorCost \leftarrow \frac{M}{k} T P_{sn}$
  $netwStorLtnc \leftarrow \frac{M}{k} D_n^{\text{wrt}} + D_{n\omega_d}^{\text{prop}}$
  $hostRetrCost \leftarrow \frac{1}{\rho} \frac{M}{k} P_{rn}$
  for $t \in \mathcal{T}$ **do**
    $netwRetrLtnc[t] \leftarrow \frac{M}{k} D_n^{\text{read}} + D_{n\omega'_{dt}}^{\text{prop}}$
  end for
  return $(hostStorCost, netwStorLtnc, hostRetrCost, netwRetrLtnc)$
end function

Algorithm 8

function BASEHEURISTIC($\mathbf{w}, \mathcal{D}, \mathcal{D}^*, \mathcal{N}, c^*, \mathbf{M}, \mathbf{T}, \boldsymbol{\mathcal{T}}, \boldsymbol{\omega}, \boldsymbol{\omega}', cdtAlloc, C, totalCost, cost, avail, \ldots$
$nodePar, filePar$)
  $tempCost \leftarrow totalCost$
  for $n \in \mathcal{N}$ **do**
    $tempC[n] \leftarrow C[n]$
  end for
  Update $tempC[n], \forall n \in \mathcal{N}^*$
  Calculate $tempCost$ according to c^*
  for $d \in \mathcal{D} \setminus \mathcal{D}^*$ **do**
    for $(k, m, \mathcal{N}^*, \mathcal{T}^*) \in cdtAlloc[d]$ **do**
      $c \leftarrow$ CALCFILECOST($\mathbf{w}, d, k, \hat{k}, m, M_d, T_d, \mathcal{T}_d, \boldsymbol{\omega}, \boldsymbol{\omega}', \mathcal{N}^*, \mathcal{T}^*, cost, avail, nodePar, filePar$)
      if $(k, m, \mathcal{N}^*, \mathcal{T}^*)$ fits in $tempC$ **then**
        break
      end if
    end for
    Update $tempC[n], \forall n \in \mathcal{N}^*$
    Calculate $tempCost$ according to c
  end for
  return $tempCost$
end function

Algorithm 9

1: **function** CALCFILECOST($\mathbf{w}, d, k, \hat{k}, m, M, T, \mathscr{T}, \boldsymbol{\omega}, \boldsymbol{\omega}', \mathscr{N}^*, \mathscr{T}^*, cost, avail, nodePar, filePar$)
2: $(\mathbf{P}^{\mathrm{s}}, \mathbf{P}^{\mathrm{r}}, \mathbf{D}^{\mathrm{read}}, \mathbf{D}^{\mathrm{wrt}}, \mathbf{A}, \mathbf{D}^{\mathrm{prop}}) \leftarrow nodePar$
3: $\phi_1 \leftarrow 0; \phi_2 \leftarrow 0; \phi_3 \leftarrow 0; \phi_4 \leftarrow 0; \phi_5 \leftarrow 0$
4: **for** $n \in \mathscr{N}^*$ **do**
5: $\phi_1 \leftarrow \phi_1 + cost[n,d].hostStorCost$
6: **end for**
7: $\phi_2 \leftarrow \phi_2 + \frac{k+m}{k} M D^{\mathrm{enc}}_{\omega_d} + k D^{\mathrm{spl}}_{\omega_d}$
8: $maxNetwStorLtnc \leftarrow \max_{n:n\in\mathscr{N}^*}\{cost[n,d].netwStorLtnc\}$
9: $\phi_2 \leftarrow \phi_2 + maxNetwStorLtnc$
10: **for** $(n,t) \in \mathscr{T}^*$ **do**
11: $\phi_3 \leftarrow \phi_3 + cost[n,d].hostRetrCost$
12: **end for**
13: $\phi_4 \leftarrow \phi_4 + \frac{\hat{k}}{k} M D^{\mathrm{dec}}_{\omega'_{dt}} + k D^{\mathrm{mrg}}_{\omega'_{dt}}$
14: $maxNetwRetrLtnc \leftarrow \max_{n,t:(n,t)\in\mathscr{T}^*}\{cost[n,d].netwRetrLtnc[t]\}$
15: $\phi_4 \leftarrow \phi_4 + |\mathscr{T}| \cdot maxNetwRetrLtnc$
16: $\phi_5 \leftarrow avail$
17: **return** $\sum_{i=1}^{5} w_i \phi_i$
18: **end function**

and let $\mathscr{T}$ be the set with the maximum number of retrievals among $\mathscr{T}_d$, for all $d \in \mathscr{D}$. Also, let $k^* = \max_{d\in\mathscr{D}} \max\{k : k \in \mathscr{K}_d\}$ and $m^* = \max_{d\in\mathscr{D}} \max\{m : m \in \mathscr{M}_d\}$, and let's assume that the required number of encoded fragments at each retrieval is $k^* + m^*$.

The computational complexity of Algorithm 7 is $O(|\mathscr{T}|)$, which is determined by lines 6-8. The computational complexity of Algorithm 9 is $O(|\mathscr{T}|(k^* + m^*))$, which is determined by lines 4-8 (complexity $O(k^* + m^*)$) and lines 10-14 (complexity $O(|\mathscr{T}|(k^* + m^*))$). The computational complexity of Algorithm 8 is $O(|\mathscr{D}| \cdot |\mathscr{K}| \cdot |\mathscr{M}| \cdot |\mathscr{T}|(k^* + m^*))$, since in the worst case it calls Algorithm 9 at most $|\mathscr{D}| \cdot |\mathscr{K}| \cdot |\mathscr{M}|$ times.

Based on the above individual algorithms' complexities, we derive the overall computational complexity of the multi-agent rollout heuristic as $O((|\mathscr{D}| \cdot |\mathscr{K}| \cdot |\mathscr{M}|)^2 \cdot |\mathscr{T}| \cdot |\mathscr{N}| \log(|\mathscr{N}|))$. This is because the sorting algorithms in lines 14 and 27 (Algorithm 6) have worst-case complexity $O(|\mathscr{T}| \cdot |\mathscr{N}| \log(|\mathscr{N}|))$ (called $|\mathscr{D}| \cdot |\mathscr{K}| \cdot |\mathscr{M}|$ times at most), Algorithm 8 is called at most $|\mathscr{D}| \cdot |\mathscr{K}| \cdot |\mathscr{M}|$ times, and $k^* + m^* \leq |\mathscr{N}|$.

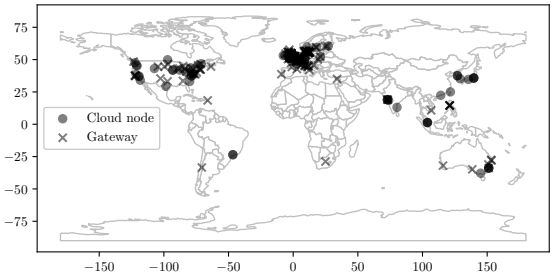
4.4 Simulation Experiments

4.4.1 Simulation Setup

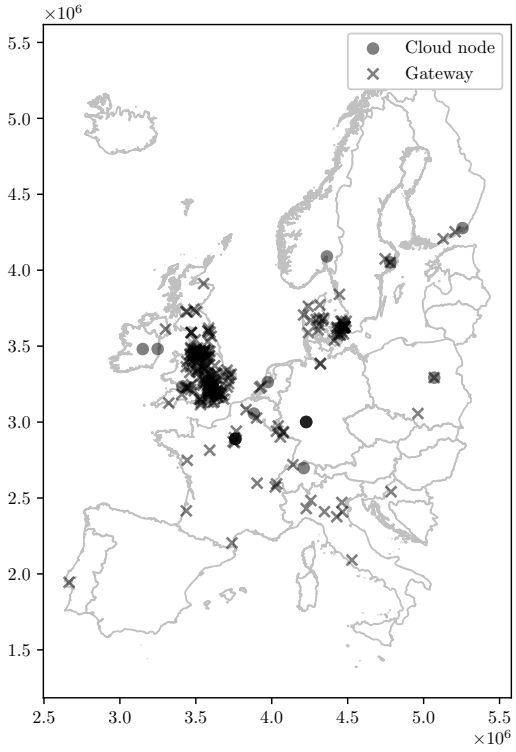
To assess the effectiveness of the proposed mechanisms, we utilized a Python simulation framework and the Gurobi Optimizer (gur) to solve the MILP model. The simulation experiments were conducted on a computer with an Intel® Core™ i7-9700K processor operating at 3.6 GHz with 32 GB of RAM. We examined two network topologies: (i) a basic topology with randomly generated parameters, and (ii) an extended topology where we employed data logs provided by Chocolate Cloud's Skyflok storage service (sky).

To evaluate the performance of the developed schemes, we considered the following cases: (i) maximizing the availability of stored files, (ii) minimizing retrieval latency, (iii) minimizing retrieval cost, (iv) minimizing storage latency, (v) minimizing storage cost, and (vi) a weighted function combining all the above criteria. We focused on the following Key Performance Indicators (KPIs): (i) monetary costs of store and retrieve operations, (ii) latencies of store and retrieve operations, (iii) availability, and (iv) the percentage of successful file retrievals.

Figure 4.4 Geo-distribution of cloud storage nodes and gateways.



(a) World map.



(b) Western and Central Europe map.

The basic topology includes a single gateway and 12 storage nodes, with 6 of these nodes inheriting characteristics from the cloud nodes and the remaining 6 from the edge nodes. The details of the various components of the monetary costs, latencies, average availability (A_n), and storage capacity (C_n) are presented in Table 4.2. The extended network topology comprises 576 edge nodes and

64 cloud nodes. The edge nodes are randomly placed in 36 countries around the world, with 16 edge nodes per country, as per the data logs. The locations of the cloud nodes are determined using the provided data logs, and are illustrated in Figure 4.4. Each city listed in the data logs incorporates a single gateway, that can transact with all cloud nodes and edge nodes present in the same country. The various components of monetary costs, latencies, availability probabilities (A_n), and storage capacities (C_n) are further detailed in Table 4.3.

We conducted several simulation experiments with different requests for the two network topologies. For the basic network infrastructure, we considered a set of 20, 40, 60, 80, or 100 file store requests, depending on the number of files in each case. Each file $d \in \mathscr{D}$ had a size of $M_d = 5$ DUs, was hosted for a period of $T_d = 10$ PUs, and was retrieved $|\mathscr{T}_d| = 100$ times from the storage service. In addition, all files were split into $k = 4$ data fragments ($\mathscr{K}_d = 4, \forall d \in \mathscr{D}$) by default, and encoded with an erasure code that used $m = 2$ parity fragments ($\mathscr{M}_d = 2, \forall d \in \mathscr{D}$). The minimum required availability for each file was set to 98%.

For the extended network topology, we used the data provided by Chocolate Cloud's Skyflok (sky). The Skyflok data logs contained the transaction entries of 12749 file requests, along with the corresponding file sizes M_d, hosting durations T_d, and the numbers of retrievals $|\mathscr{T}_d|$. All files were split into $k = 5$ data fragments ($\mathscr{K}_d = 5, \forall d \in \mathscr{D}$) by default, with the addition of $m = 4$ parity fragments ($\mathscr{M}_d = 4, \forall d \in \mathscr{D}$). The default minimum required availability was set to 98%.

In the following simulation scenarios, unless otherwise stated, we assumed the use of an optimal erasure code scheme ($\epsilon = 0$), i.e., the number of required encoded fragments to be retrieved was equal to the number of data fragments ($\hat{k} = k$). The parameters used in the simulations are detailed in Table 4.3.

4.4.2 Optimality Performance Evaluation of Heuristic and Multi-Agent Rollout Mechanisms

Initially, we conducted simulation experiments to evaluate the performance of the MILP and the multi-agent rollout mechanism. The experiment considered various numbers of files ($|\mathscr{D}| \in [20, 100]$) for the previously described optimization cases using the basic topology. The critical parameters of the simulation experiments are presented in Table 4.2.

As expected, the total cost increases linearly with the number of hosted files (Figure 4.5a). Using the heuristic mechanism, in the weighted multi-objective optimization case, the monetary cost ranges from 679 CUs (20 hosted files) to 3341 CUs (100 hosted files), for an average cost per file of around 34 CUs. When the objective is either the minimization of

Table 4.2 Default parameters used with the basic topology setup.

Par.	Value		Unit
M_d	5,	$\forall d \in \mathscr{D}$	DU
$\mathscr{K}_d$	$\{4\}$,	$\forall d \in \mathscr{D}$	–
$\mathscr{M}_d$	$\{2\}$,	$\forall d \in \mathscr{D}$	–
T_d	10,	$\forall d \in \mathscr{D}$	–
$\lvert\mathscr{T}_d\rvert$	100,	$\forall d \in \mathscr{D}$	–
$\mathscr{Q}_d$	$\{A_{\text{req}} = 98.0\%\}$,	$\forall d \in \mathscr{D}$	%
ϵ	0		–
N_c	6		–
N_e	6		–
ρ	$10e^6$		DU/GET
$D^{\text{spl}}_{\omega_d}$	$\mathscr{U}(0.15, 0.195)$		TU/split
$D^{\text{mrg}}_{\omega_d}$	$\mathscr{U}(0.12, 0.156)$		TU/merge
$D^{\text{enc}}_{\omega_d}$	$\mathscr{U}(300, 390)$		TU/DU
$D^{\text{dec}}_{\omega_d}$	$\mathscr{U}(150, 195)$		TU/DU
	Cloud nodes	**Edge nodes**	
D^{wrt}_{n}	$\mathscr{U}(300, 390)$	$\mathscr{U}(100, 130)$	TU/DU
D^{read}_{n}	$\mathscr{U}(150, 195)$	$\mathscr{U}(50, 65)$	TU/DU
$D^{\text{prop}}_{n\omega_d}$	$\mathscr{U}(0.5, 0.65)$	$\mathscr{U}(0.05, 0.065)$	TU
$P_{\text{s}n}$	$\mathscr{U}(0.25, 0.325)$	$\mathscr{U}(0.4, 0.52)$	CU/(DU·PU)
$P_{\text{r}n}$	$\mathscr{U}(200e^{-8}, 260e^{-8})$	$\mathscr{U}(320e^{-8}, 416e^{-8})$	CU/GET
A_n	$\mathscr{U}(99.9\%, 99.99\%)$	$\mathscr{U}(70.0\%, 75.0\%)$	%
C_n	∞	$\mathscr{U}(2500, 3250)$	DU

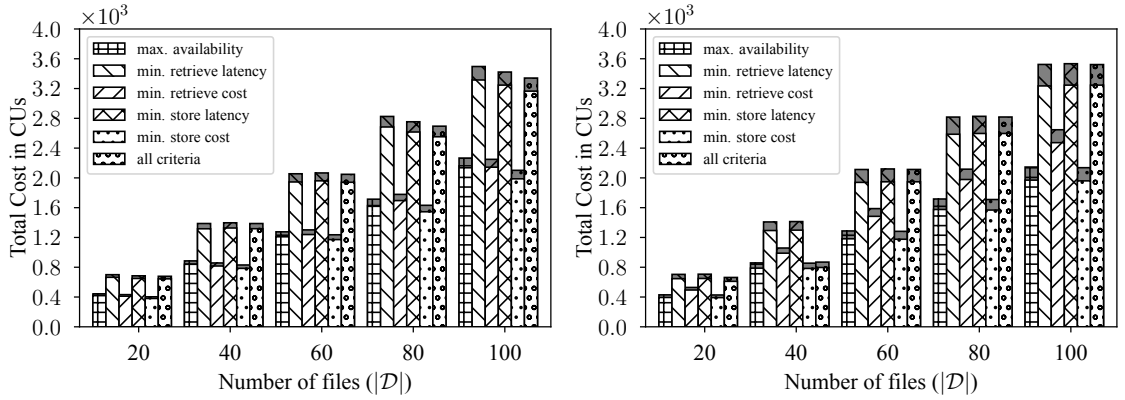
store or retrieve operation latencies, the cost is higher than the multi-objective scenario by 2.4% and 4.7%, respectively. When the objective is either the minimization of store or retrieve operation monetary costs or the maximization of availability, the cost is lower than the multi-objective case by 37.1%, 32.7%, and 32.2%, respectively.

With the MILP mechanism (Figure 4.5b) and considering all objectives, the monetary cost ranges from 663 CUs (20 hosted files) to 3522 CUs (100 hosted files), for an average cost per file of 32.1 CUs, which is about 5.6% lower than that obtained with the heuristic mechanism. When the objective is either only the minimization of store or retrieve operation latency, the monetary cost is nearly identical to the multi-objective scenario. When the objective is either the minimization of store or retrieve operation monetary costs or the maximization of availability, the cost is lower than the multi-objective scenario by 65.6%, 31.3%, and 64.8%, respectively.

Table 4.3 Default parameters used with the extended topology setup.

Par.	Value		Unit
$\mathcal{K}_d$	$\{5\}$,	$\forall d \in \mathcal{D}$	–
$\mathcal{M}_d$	$\{4\}$,	$\forall d \in \mathcal{D}$	–
$\mathcal{Q}_d$	$\{A_{\text{req}} = 98.0\%\}$,	$\forall d \in \mathcal{D}$	%
ϵ	0		–
N_c	64 in total		–
N_e	16 per country / 576 in total		–
ρ	$10e^6$		DU/GET
$D^{\text{spl}}_{\omega_d}$	$\mathcal{U}(0.15, 0.195)$		TU/split
$D^{\text{mrg}}_{\omega_d}$	$\mathcal{U}(0.12, 0.156)$		TU/merge
$D^{\text{enc}}_{\omega_d}$	$\mathcal{U}(300, 390)$		TU/DU
$D^{\text{dec}}_{\omega_d}$	$\mathcal{U}(150, 195)$		TU/DU
	Edge nodes		
D^{wrt}_{n}	$\mathcal{U}(100, 120)$		TU/DU
D^{read}_{n}	$\mathcal{U}(50, 60)$		TU/DU
$D^{\text{prop}}_{n\omega_d}$	$\mathcal{U}(0.05, 0.06)$		TU
P_{sn}	$\mathcal{U}(0.32, 0.40)$		CU/(DU·PU)
P_{rn}	$\mathcal{U}(130e^{-8}, 156e^{-8})$		CU/GET
	Cloud nodes	**Edge nodes**	
A_n	$\mathcal{U}(99.9\%, 99.99\%)$	$\mathcal{U}(70.0\%, 72.0\%)$	%
C_n	∞	$\mathcal{U}(2500, 3250)$	DU

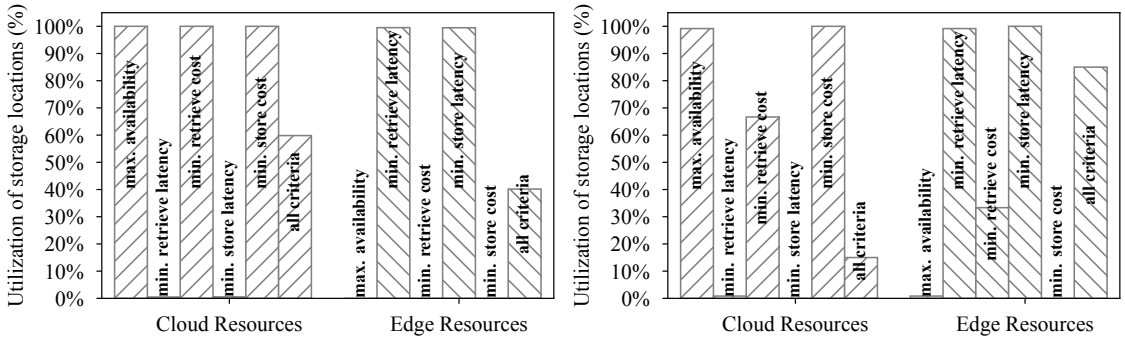
Figure 4.5 Monetary costs as a function of the number of files for the basic topology (white: store costs, gray: retrieve costs).



(a) Heuristic mechanism. **(b)** MILP mechanism.

We then analyzed the utilization of resources in the cloud and edge nodes (Figures 4.6a and 4.6b). In Figure 4.6a, we can see that when the objective is to minimize the store or

Figure 4.6 Effect of the optimization objectives on the percentage of utilized cloud and edge nodes with the basic topology.



(a) Heuristic mechanism. **(b)** MILP mechanism.

retrieve operation latency, the heuristic mechanism tends to prefer edge nodes over the cloud. This preference is also observed in the MILP mechanism results, which are presented in Figure 4.6b. The reason for this is that edge nodes are typically closer to gateways compared to cloud data centers (85).

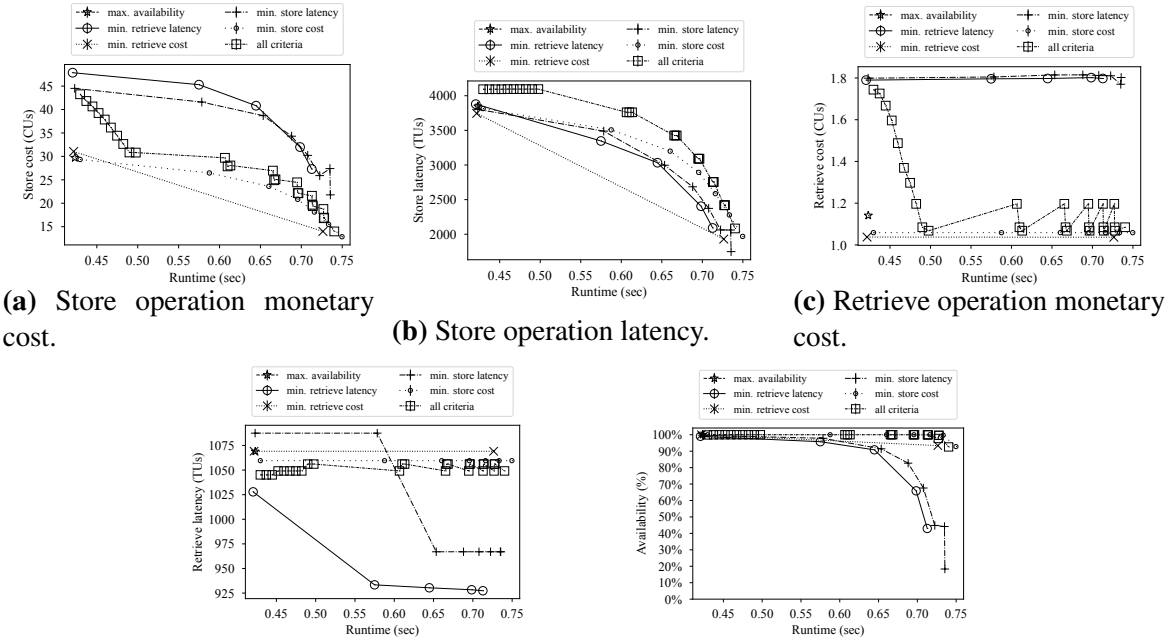
However, when the objective is set to minimize the store or retrieve operation (monetary) cost or maximize availability, both the heuristic and MILP mechanisms prefer to host the fragments on the cloud, as it is cheaper (edg) and has higher availability (28). Specifically, the heuristic mechanism utilizes only cloud nodes in all these cases (Figure 4.6a), while the MILP mechanism's optimal solution uses only cloud nodes when optimizing the monetary cost of store operation and availability. However, the percentage of cloud resources used drops to 67% when minimizing the retrieve operation cost with the MILP mechanism.

When all the objectives are taken into consideration, the heuristic scheme prefers edge to cloud nodes in 60.5% of the utilized storage nodes (Figure 4.6a). Correspondingly, the MILP mechanism places the fragments in edge nodes over cloud nodes in 83.3% of the utilized storage nodes (Figure 4.6b). We observe that the heuristic mechanism achieves a solution that is close to optimal, with a 72.6% match.

Table 4.4 Comparison of execution times between mechanisms (sec).

$\|D\|$	MILP	Base Heuristic	Multi-agent Rollout
20	363.38	0.003	0.143
40	726.53	0.005	0.560
60	1089.69	0.008	1.247
80	1452.84	0.010	2.208
100	1816.02	0.013	3.442

Figure 4.7 Progress of the optimization over time.



(a) Store operation monetary cost.

(b) Store operation latency.

(c) Retrieve operation monetary cost.

(d) Retrieve operation latency.

(e) Average availability.

Table 4.4 presents the execution times of the different mechanisms developed. The MILP mechanism exhibits the highest execution time, which could render it impractical for large-size distributed storage systems or when fast decisions need to be made. In contrast, the greedy heuristic algorithm demonstrates the lowest execution time, taking advantage of the sequential execution and the limited number of choices that are examined. The multi-agent rollout mechanism, which exhibits an execution time below 3.5 seconds, manages also to maintain a resource allocation close to the optimal solution, taking advantage of the use of the low-complexity heuristic in a reinforcement learning approach. This clearly shows its ability to effectively balance performance and execution time.

We also examined the multi-agent rollout objective cost evolution as the number of iterations increases. Figures 4.7a-4.7e show the improvement brought about by the multi-agent rollout algorithm compared to the performance of the base heuristic. The improvement is presented over the simulation time. In each figure, we examine the temporary achieved costs in terms of the following optimization criteria: store operation monetary cost (Figure 4.7a), store operation latency (Figure 4.7b), retrieve operation monetary cost (Figure 4.7c), retrieve operation latency (Figure 4.7d), and availability (Figure 4.7e).

All the figures depict various scenarios of optimization criteria. There are $|\mathscr{D}| = 10$ files that are split into $k = 5$ data fragments, and we provide the following options for the number of parity fragments to the heuristic: $m \in \{0, 1, 2, 3, 4, 5, 6\}$. Under the store

monetary cost minimization criterion, the rollout heuristic improves the initial solution by up to 57% in 0.33 seconds, while also decreasing store operation latency by 47% from 3814 TUs to 1969 TUs. The algorithm achieves this improvement by gradually selecting solutions with less redundancy, reducing the number of parity fragments and therefore the data size. Furthermore, a lower number of fragments also reduces the store operation latency, as this is determined by the slowest fragment placement. Retrieve operation monetary cost and latency remain unchanged throughout the simulations, as the redundancy does not affect the number of fragments or the size of the retrieved data. Meanwhile, availability remains above 93% for all scenarios, as the minimization of the store operation monetary cost favors the utilization of cloud nodes.

The results when minimizing the store operation latency are quite similar to the previous scenario. The heuristic achieves an improvement of up to 27% in 0.32 seconds. As in the previous scenario, the algorithm selects options with less redundancy, reducing the number of parity fragments and, therefore, the store operation latency. However, since the number of fragments retrieved each time is not directly affected by the redundancy, the monetary cost remains unchanged.

When the optimization criterion is minimizing the retrieve operation monetary cost, the initial solution is derived with a cost of 1.03 CUs per operation and is slightly improved to 1.04 CUs in 0.73 seconds. The data size is not directly affected by the reduced redundancy, but since the transmitted data size is reduced, the store operation cost is affected and reduced from 31 CUs to 14 CUs. The store operation latency is also reduced from 3745 CUs to 1930 CUs. On the other hand, the retrieve operation latency remains almost unchanged, as it is not directly affected by the redundancy.

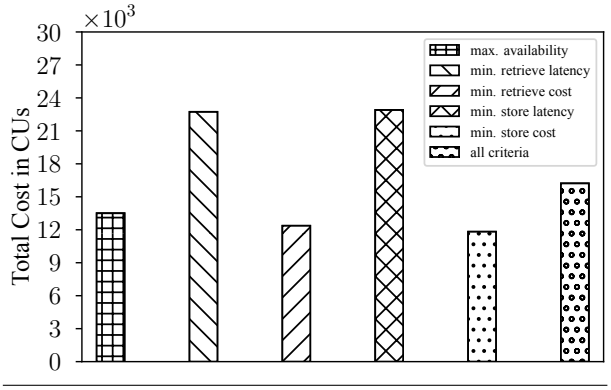
Retrieve operation latency is minimized by gradually improving the initial solution of latency 1028 TUs to 927 TUs in 0.71 seconds, an overall reduction of 10%. The algorithm initially selects a cloud node to host the fragments, and then the redundancy is further reduced, limiting the maximum distance between a fragment and a gateway.

In conclusion, the multi-agent rollout algorithm significantly outperforms the base heuristic in terms of cost and latency. By gradually selecting solutions with less redundancy, the algorithm reduces the number of parity fragments and consequently the data size, also leading to decreased store operation latency. However, the retrieve operation costs and latency remain unchanged during the simulations, as redundancy does not influence the number of fragments or the size of the retrieved data. In all scenarios, the availability remains above 93% because minimizing store operation costs favors the use of cloud nodes.

4.4.3 Evaluating Performance Based on Distributed Storage KPIs

We proceeded with simulation experiments using real-world data, as outlined in Section 4.4.1, evaluating the impact of various optimization criteria on users' monetary cost charges. As depicted in Figure 4.8, the average total cost per user was 22907 and 22735 CUs when minimizing store and retrieve operation latency, respectively. Conversely, when minimizing store or retrieve operation monetary costs, the average total cost was 11827 and 12370 CUs, respectively. Minimizing files' availability resulted in an average cost of 13518 CUs per user. In the multi-objective optimization scenario, the average total cost was 16228 CUs. When minimizing store or retrieve operation latencies, the cost was 41.2% and 40% higher, compared to the multi-objective scenario. However, optimizing store or retrieve operation monetary costs, or availability, the cost was 27.1%, 23.8%, and 16.7% lower than the multi-objective scenario, respectively.

Figure 4.8 Evaluation of the monetary costs.



In the next step of our simulation experiments, we analyzed the utilization of both cloud and edge nodes, as shown in Figure 4.9. When the objective is set to minimize store or retrieve operation latency, the heuristic mechanism utilizes only edge nodes. However, when store or retrieve operation (monetary) cost or maximize availability are minimized, the heuristic mechanism prefers to host fragments on the cloud. This is because the cloud is cheaper (edg) for hosting and provides significantly higher availability (28). In the multi-objective scenario, the heuristic mechanism prefers the cloud instead of the edge nodes for 59.8% of the utilized storage nodes, highlighting the potential cost-saving benefits of cloud usage.

Figure 4.9 Effect of the optimization objectives on the percentage of utilized cloud and edge nodes.

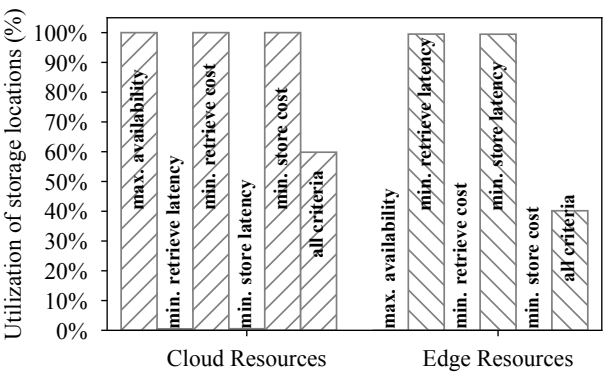
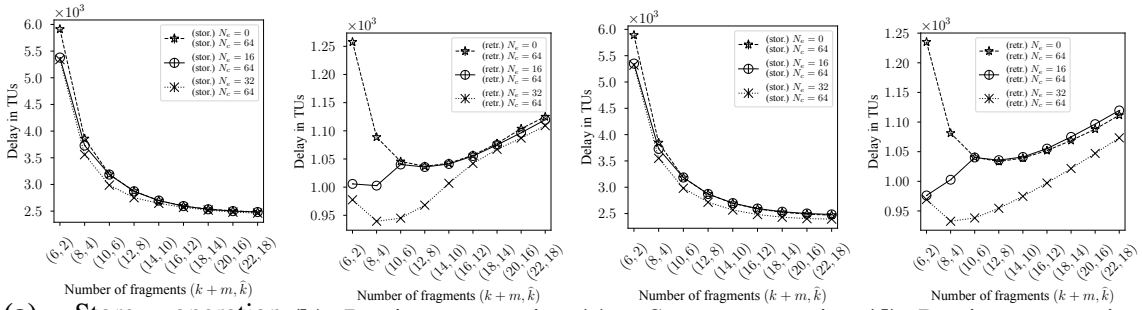


Figure 4.10 Effect of the number of data fragments the files are split into, on the store and retrieve operation latencies.



(a) Store operation with multi-objective optimization.

(b) Retrieve operation with multi-objective optimization.

(c) Store operation while optimizing the latencies.

(d) Retrieve operation while optimizing the latencies.

Effect of the Erasure Code on the Experienced Latency and Monetary Cost

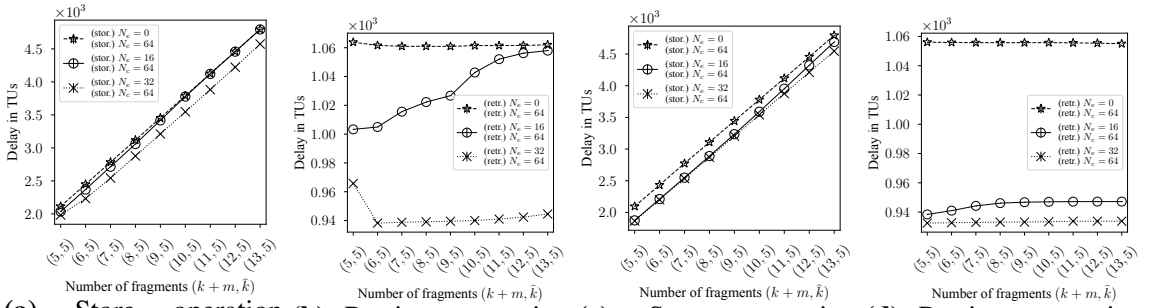
We also investigated the impact of the number of data fragments on store and retrieve operation latencies. We consider three scenarios for the number of randomly-computed edge nodes in each real-world country, $N_e \in \{0, 16, 32\}$, while keeping the number of cloud nodes set to $N_c = 64$, as seen in the data logs. The number of data fragments is varied from $k = 2$ to $k = 18$, while the number of parity fragments is set to $m = 4$.

Figure 4.10a and 4.10b show the results for the weighted multi-objective scenario, while Figure 4.10c and 4.10d display the results for the scenarios where only store and retrieve operation latencies are optimized. The number of data fragments significantly impacts the store and retrieve operation latencies, with both latencies decreasing as the number of fragments increases. In the scenario where only cloud nodes are available (Figure 4.10a), the store operation latency steadily decreases from 5911 TUs with an erasure code of $(6, 2)$ ($k = 2$) to around 2484 TUs with an erasure code of $(22, 18)$ ($k = 18$). A similar trend is observed in Figure 4.10b, where the store operation latency decreases from around 5890 TUs with an erasure code of $(6, 2)$ ($k = 2$) to 2469 TUs with an erasure code of $(22, 18)$ ($k = 18$). In contrast, the retrieve operation latency in Figure 4.10b decreases from around 1258 TUs with an erasure code of $(6, 2)$ ($k = 2$) to 1034 TUs with an erasure code of $(22, 18)$ ($k = 18$), and then gradually increases as k increases.

In the case where both cloud and edge nodes are available (Figure 4.10c), the retrieve operation latency decreases from 1235 TUs with an erasure code of $(6, 2)$ ($k = 2$) to 1033 TUs with an erasure code of $(12, 8)$ ($k = 8$) and then begins to increase. These scenarios exhibit maximum delays, since only cloud nodes are available.

In the scenario of $(N_e, N_c) = (16, 64)$, the store and retrieve operation latencies can be lower than in the previous scenario where only cloud nodes were available, thanks to the

Figure 4.11 Effect of the number of parity fragments used for redundancy to the store and retrieve operation latencies.



(a) Store operation with multi-objective optimization.

(b) Retrieve operation with multi-objective optimization.

(c) Store operation while optimizing the latencies.

(d) Retrieve operation while optimizing the latencies.

selection of edge nodes. Specifically, although the store operation latency is initially lower, when more than $k + m = 10$ encoded fragments are selected, the delay is nearly equal to the previous case, as the heuristic selects some cloud storage resources. This is also shown in Figure 4.10c, where the objective is to minimize latencies. In this scenario, when the erasure code is $(18, 14)$ ($\hat{k} = 14$), the delay reaches the levels of the previous scenario since the edge nodes are not sufficient to host all the fragments ($k + m = 18$ transmitted). The retrieve operation latency (Figure 4.10b) is initially low due to the selection of edge nodes. However, from the erasure code of $(10, 6)$ and onwards, the heuristic performs similarly to the previous scenario due to the inclusion of cloud nodes in the solution. Again, this is more evident in Figure 4.10d, where a sharp delay increase occurs when using the erasure code of $(22, 18)$. At this point, the 16 edge nodes are not sufficient for all required $\hat{k} = 18$ fragments. Lastly, in the scenario of $(N_e, N_c) = (32, 64)$, the edge nodes are sufficient to store all the fragments, resulting in lower latencies than all previous scenarios.

We also evaluated the store operation latency (Figures 4.11a and 4.11c) and the retrieve operation latency (Figures 4.11b and 4.11d), with the number of data fragments set to $k = 5$ while changing the number of parity fragments m. Figures 4.11a and 4.11b show the results of the multi-objective optimization scenario, while Figures 4.11c and 4.11d show the optimization of both the store and retrieve operation latencies, excluding monetary costs and availability.

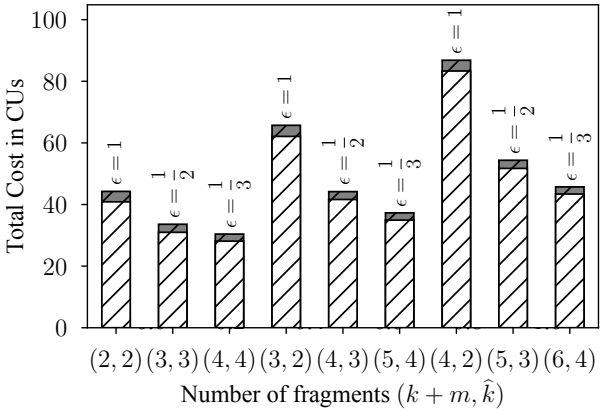
As seen in Figure 4.11, increasing the redundancy results in a linear increase in the store operation latency (Figures 4.11a and 4.11c). This is expected, since the total size of the data increases linearly with the addition of more parity fragments. Moreover, introducing more edge nodes into the infrastructure reduces the store operation latency, and to some extent, the retrieve operation latency.

In the scenario of $(N_e, N_c) = (0, 64)$ (Figure 4.11b), the heuristic selects only cloud nodes, resulting in the highest average latency per retrieve operation. In contrast, the scenario of $(N_e, N_c) = (32, 64)$ (Figure 4.11d) results in the lowest average latency per retrieve operation, as the heuristic only selects edge nodes. The latency is not affected by the addition of parity fragments, since the retrievals always require $\hat{k} = 5$ fragments out of the $k + m$ hosted ones.

In the mid-case scenario of $(N_e, N_c) = (16, 64)$, the latency is observed to increase in the results shown in Figureσ 4.11b and 4.11d, but at a higher pace in Figure 4.11b. In Figure 4.11b, which depicts the multi-objective scenario, the heuristic selects to retrieve some fragments from cloud nodes even though the 16 edge nodes per country are sufficient to host all the required $\hat{k} = 5$ ones. In contrast, Figure 4.11d only shows edge nodes being selected to host and recover the fragments. However, the latency is higher compared to the scenario with $N_e = 32$, since a lower number of available edge nodes increases the average maximum distance between the gateway and the selected nodes. Overall, increasing the number of parity fragments greatly increases the store operation latency and to some extent the retrieve operation latency. However, the retrieve operation latency can be reduced by increasing the number and density of deployed edge nodes, providing more options for node selections.

We also examined the effect of erasure code selection on monetary cost for different options of data and parity fragments (Figure 4.12). We considered $k \in \{1, 2, 3\}$ and $m \in \{1, 2, 3\}$ and examined three cases of near-optimal erasure codes with $\epsilon = 1$, $\epsilon = \frac{1}{2}$, and $\epsilon = \frac{1}{3}$. The white part of the bars depicts the average monetary cost of file storage, while the gray part shows the average monetary cost of file retrieval. We first considered three samples, i.e., $(k + m, \hat{k}) = (2, 2)$, $(k + m, \hat{k}) = (3, 3)$, and $(k + m, \hat{k}) = (4, 4)$ (split into $k = 1, 2, 3$), where we used one parity fragment. In this scenario, as the number of fragments increased, the total cost decreased from 44.3 to 28.1 CUs. The parity fragments and the data fragments had the same size, which was M_d/k. Therefore, the total data size transferred to the storage nodes was $M_d + M_d/k$, i.e., $2M_d$, $\frac{3}{2}M_d$, and $\frac{4}{3}M_d$. The total data size transferred during each retrieval

Figure 4.12 Effect of the erasure code policy on the total monetary costs (white: store costs, gray: retrieve costs).



from the storage nodes to the gateways was $\hat{k}M_d/k = (1+\epsilon)M_d$, i.e., $2M_d$, $\frac{3}{2}M_d$, and $\frac{4}{3}M_d$. The store and retrieve data size was proportional to the corresponding costs, which explained the cost decrease.

Next, we considered three samples, i.e., $(k+m, \hat{k}) = (3,2)$, $(k+m, \hat{k}) = (4,3)$, and $(k+m, \hat{k}) = (5,4)$ (split into $k = 1,2,3$), where we used two additional parity fragments. As in the previous scenario, the data and the parity fragments had size M_d/k DUs, and the total data size transferred to the storage nodes was $M_d + 2M_d/k$, i.e., $3M_d$, $2M_d$, and $\frac{5}{3}M_d$. The total data size transferred from the storage nodes to the gateways during each retrieval was $(1+\epsilon)M_d$, i.e., $2M_d$, $\frac{3}{2}M_d$, and $\frac{4}{3}M_d$. Again, the total size was proportional to the store operation cost, which explained the cost decrease.

Lastly, we considered three samples, i.e., $(k+m, \hat{k}) = (4,2)$, $(k+m, \hat{k}) = (5,3)$, and $(k+m, \hat{k}) = (6,4)$ (split into $k = 1,2,3$), where there were three parity fragments with size M_d/k each. Hence, the total data size transferred was $M_d + 3M_d/k$ (i.e., $4M_d$, $\frac{5}{2}M_d$, $2M_d$) during each store operation and $(1+\epsilon)M_d$ (i.e., $2M_d$, $\frac{3}{2}M_d$, and $\frac{4}{3}M_d$) during each retrieve operation. As in the previous cases, the total size is proportional to the depicted costs.

The results presented demonstrate that the overall monetary cost is directly proportional to both $(k+m)M_d/k$ and $\hat{k}M_d/k$. As such, the monetary cost achieved is dependent on the effectiveness of the chosen erasure coding scheme. When the objective is to minimize the monetary cost, optimal erasure code schemes are utilized. However, it is important to note that the optimal schemes may impact the latency due to their need for more intensive processing.

Effect of the Minimum Availability Requirement on the Redundancy and Node Selection

In Figure 4.13, we investigated the impact of the minimum availability requirement on redundancy. We divided all files into $k = 5$ data fragments, and considered the number of candidate parity fragments to be $m \in \{0,1,2,3,4,5,6\}$.

When the goal is to optimize file availability, the heuristic chooses the maximum redundancy, which is 6 parity fragments in our setup. In all other cases, the number of parity fragments increases as the availability requirement grows. When the objective is to minimize the monetary cost of the store operation, the heuristic selects a lower number of parity fragments compared to all other scenarios. Meanwhile, the retrieve operation cost is independent of redundancy, since the data of size $(\hat{k}/k)M_d$ is always needed. Therefore, when the store operation cost is minimized and the minimum availability requirement is satisfied, the redundancy is also minimized. Specifically, we choose $m = 0$ parity fragments for $A_{\text{req}} = 75\%, 79\%, 83\%, 87\%, 91\%$, and $m = 1$ for $A_{\text{req}} = 95\%, 99\%$. When the retrieve

operation cost is minimized, the number of parity fragments is not a consideration, and any number of parity fragments can be chosen as long as the availability requirement is met.

In the case of minimizing the store operation latency, edge are preferred to cloud nodes. Furthermore, since edge nodes have lower average availability than cloud nodes, more nodes are chosen to meet the availability constraint. Also, the number of parity fragments is minimized, since all the $k + m$ encoded fragments are transmitted to their corresponding nodes during the store operation. The latency is determined by the slowest fragment placement. Therefore, increasing the redundancy leads to the inclusion of more distant nodes, which increases the latency. Thus, the heuristic selects $m = 3$ parity fragments for $A_{\text{req}} = 75\%, 79\%, 83\%, 87\%, 91\%$, and $m = 4$ for $A_{\text{req}} = 95\%, 99\%$. On the other hand, during the retrieve operation, only $\hat{k}$ out of $k + m$ encoded fragments are retrieved. Thus, increasing the redundancy provides a larger pool of node selections for each retrieval and does not burden the latency.

Figure 4.13 Effect of the minimum availability requirement on the selection of the level of redundancy.

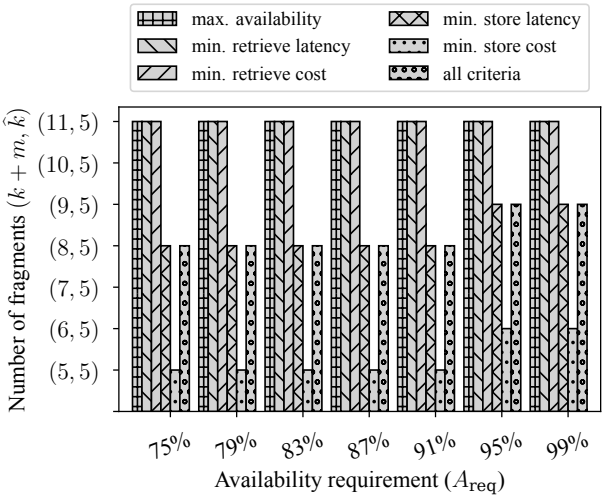
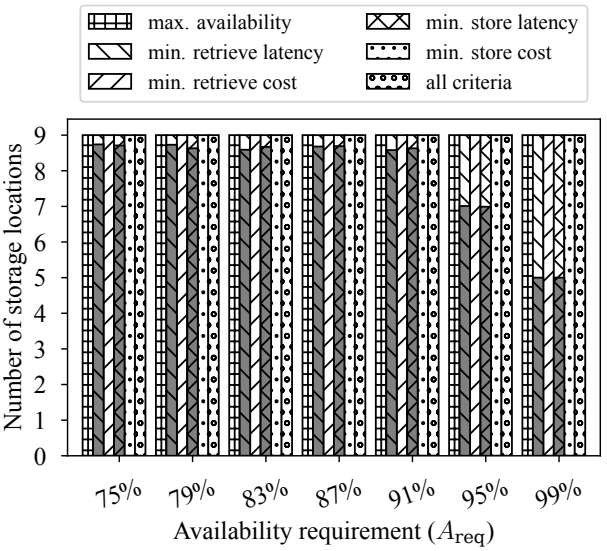


Figure 4.14 Effect of the minimum availability requirement on the types of the selected storage nodes.



Finally, the results of the multi-objective scenario are similar to those of minimizing the store operation latency. This is because edge nodes, which generally have lower availability than cloud nodes (28), are chosen. We expect this scenario to be somewhere between the other two scenarios, depending on the weight coefficients of the objectives and the infrastructure's characteristics. In Figure 4.14, we analyze the impact of the minimum availability requirement on the choice of

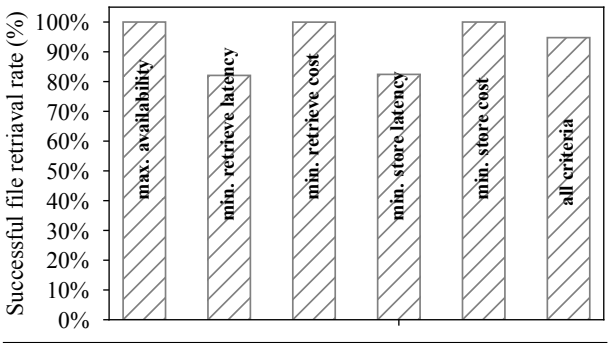
storage resources (cloud or edge nodes) to host file fragments. The experiment considers files split into $k = 5$ data fragments and $m = 4$ additional parity fragments using the erasure code $(k + m, \hat{k}) = (9, 5)$. The white-colored bars in the figure represent the average number of utilized cloud nodes, and the gray-colored ones the average number of edge nodes.

We observe that the heuristic uses only cloud nodes when optimizing for availability, store and retrieve monetary costs, or in the multi-objective scenario, since cloud nodes offer higher availability and lower monetary costs. In contrast, when optimizing for store or retrieve operation latency and the minimum required availability is at least 91%, the heuristic predominantly employs edge nodes, as they offer lower latencies (82, 85). For higher availability requirements, such as 95%, the number of cloud nodes increases, but edge nodes still dominate. Edge nodes alone may not suffice for high-availability scenarios; therefore, the heuristic adds cloud nodes to achieve the required availability level.

Effect of the Optimization Objective on the File Availability

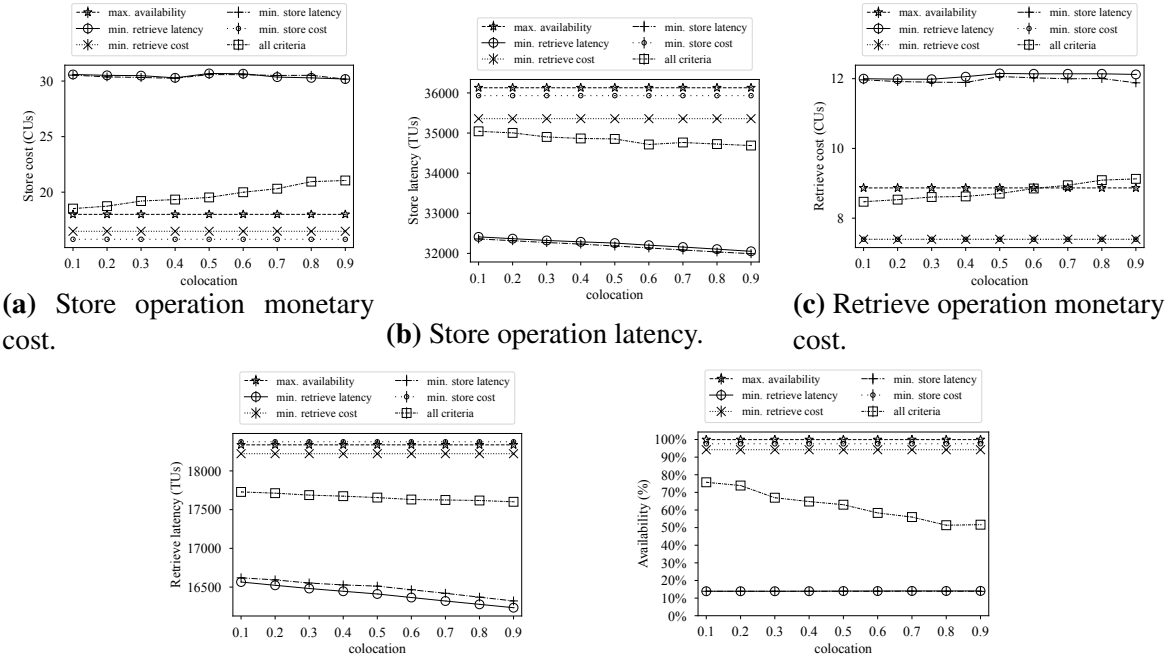
Figure 4.15 shows the successful retrieval rate of stored files, which is a parameter that directly affects both monetary cost and latency. A file is considered successfully retrieved only when all $\hat{k}$ storage nodes selected by the algorithm for that particular retrieval are operational and accessible when requested for fragment retrieval.

Figure 4.15 Percentage of successful retrievals for each optimization objective.



The results indicate that when the optimization criterion is availability, or store and retrieve monetary costs, files are always retrieved successfully because the algorithm chooses only cloud nodes. However, when the optimization criterion is either store or retrieve operation latency, the successful file retrieval rate drops to around 82%. This is due to the selection of edge nodes, which generally exhibit lower availability than cloud nodes (28). In the multi-objective scenario, the successful retrieval rate is 94.8% due to the selection of a mixed combination of edge and cloud nodes.

Figure 4.16 Effect of the edge nodes' colocation.



(a) Store operation monetary cost.

(b) Store operation latency.

(c) Retrieve operation monetary cost.

(d) Retrieve operation latency.

(e) Average availability.

Effect of the Colocation of the Edge Nodes on the Examined KPIs for the Different Optimization Criteria

We examined the impact of edge node colocation on the Key Performance Indicators (KPIs). When the optimization criterion is the store and retrieve monetary costs and availability, the colocation degree of edge nodes does not affect the results, since the algorithm only selects cloud nodes to host fragments. Cloud nodes have lower monetary costs than edge nodes and provide higher availability, as reported in (edg) and (28).

When the store operation monetary cost is minimized, the achieved cost is 15.8 CUs (Figure 4.16a), which is the lowest cost among all the scenarios. The retrieve operation monetary cost is also minimized (Figure 4.16c) with a cost of 7.4 CUs. However, the achieved store operation latency (Figure 4.16b) is 35932 TUs, the second-highest delay among all scenarios, while the retrieve operation latency is maximal at 18376 TUs (Figure 4.16d). The availability reaches the second-highest value of 97.6% (Figure 4.16e), with cloud nodes being preferred for fragment hosting.

Next, we evaluated minimizing retrieve operation monetary cost on KPIs for different colocation levels. Retrieve monetary cost (Figure 4.16c) was constant at 7.4 CUs, nearly equal to the previous scenario's store operation cost. Store operation cost (Figure 4.16a) reached 16.5 CUs, slightly higher than the store optimization scenario, as cloud nodes were

preferred over edge nodes. The achieved value is nearly equal to the cost of the previous scenario, with a retrieve monetary cost of 7.4 CUs per retrieval. When evaluating edge node colocation on KPIs, results are unaffected by colocation degree for store operation monetary cost, retrieve operation monetary cost, and availability, as only cloud nodes host fragments. Store operation monetary cost is minimized at 15.8 CUs (Figure 4.16a), retrieve operation cost at 7.4 CUs (Figure 4.16c), and availability reaches 97.6% (Figure 4.16e). Store operation latency is the second-highest delay at 35932 TUs (Figure 4.16b), and retrieve operation latency is maximal at 18376 TUs (Figure 4.16d). The heuristic selects only cloud nodes for fragment hosting.

When examining the effect of minimizing the retrieve operation monetary cost on the examined KPIs, the achieved retrieve monetary cost is almost 7.4 CUs for different levels of colocation (Figure 4.16c). The store operation cost is slightly higher than the previous scenario, at 16.5 CUs (Figure 4.16a), as cloud nodes were preferred over edge nodes. When the optimization objective minimizes the store operation latency, the colocation degree affects the achieved latency. With a colocation value of 0.1, the store latency is 32411 TUs (Figure 4.16b), and it decreases linearly as the colocation increases until it reaches around 32052 TUs at a colocation of 0.9. Other factors that can affect the latency include encoding and encryption delay and delay of the fragment placement. The retrieve operation latency exhibits a similar behavior, starting at 16566 TUs at a colocation of 0.1 and reducing linearly to 16234 TUs at a colocation of 0.9 (Figure 4.16d). The store and retrieve operation monetary costs are nearly constant at 30.6 CUs and 12 CUs, respectively (Figures 4.16a and 4.16c), while the availability is nearly constant at around 14%, the lowest of all scenarios (Figure 4.16e). As edge nodes become more proximate to gateways due to increased colocation, latency decreases (Figure 4.16b). For instance, with a colocation value of 0.1, the store latency is 32411 TUs, which linearly decreases until the colocation becomes 0.9, reaching around 32052 TUs. Apart from colocation, other factors can affect latency, such as encoding and encryption delay at the gateways, which depend on the total data size, and the delay of fragment placement. Edge nodes' proximity to gateways reduces delay when multiple locations are present. This behavior is similar when optimizing retrieve operation latency. For a small degree of colocation (0.1), the latency starts at 16566 TUs, which linearly reduces to 16234 TUs until 0.9 (Figure 4.16d). Store and retrieve operation monetary costs (Figures 4.16a and 4.16c) are nearly constant at 30.6 CUs and 12 CUs, respectively, and unaffected by colocation. Fluctuations observed in the range of 0.4-0.6 are derived from the monetary charging differences between the edge nodes.

Maximizing availability does not directly affect examined KPIs in our simulations, using only cloud nodes with higher availability than edge nodes (28). 99.9% availability (Fig-

ure 4.16e) is achieved, while store operation cost is 18 CUs (Figure 4.16a), higher than in cost-minimized scenarios. Retrieve operation cost per operation is low at 8.9 CUs (Figure 4.16c). In multi-objective optimization, store operation cost increases linearly from 18.5 to 21 CUs, and retrieve operation cost from 8.5 to 9.2 CUs. This behavior is due to the decrease in cloud nodes in favor of edge nodes, taking advantage of proximity to gateways. Availability decreases linearly from 75.7% to 52%. The increase in proximity benefits latency (Figures 4.16b and 4.16d), allowing for relatively more expensive edge nodes. Finally, store and retrieve operation latencies are affected by increased colocation. Store operation latency decreases from 35044 to 34689 TUs, and retrieve operation latency from 17729 to 17601 TUs (colocation degrees of 0.1 and 0.9). This improvement is achieved by reducing cloud nodes in favor of edge nodes, which are closer to the gateways.

4.5 Conclusion

As the amount of digital data continues to increase dramatically, there is an increasing need for efficient and reliable distributed storage infrastructures to accommodate the rising storage capacity requirements. Edge computing plays a crucial role in reducing data transaction latency, thereby enhancing overall performance, while erasure coding techniques can significantly improve data security and availability. We proposed storage resource allocation mechanisms over the edge-cloud continuum. We developed an optimal mixed-integer linear programming formulation (MILP) and a sub-optimal multi-agent rollout heuristic for storage resource selection aimed at hosting the file fragments as provided by the application of the erasure coding. These mechanisms jointly optimize monetary costs, latency, and average availability, all while satisfying user requirements. By trading off performance for execution time, our proposed mechanisms achieve near-optimal performance in extensive simulations using both synthetic and real data that varies in the range 94.8-97.5%. In read data simulations, we examined the impact of different optimization criteria on store and retrieve operation monetary costs, latencies, availability, and percentage of successful file retrievals. Compared to the single-objective scenarios, our proposed multi-objective optimization mechanisms effectively addressed the various conflicting requirements by optimizing the placement of files, achieving optimal or close to optimal minimum cost, latency and availability, all within a small execution time with the multi-agent rollout mechanism that significantly advanced the greedy heuristic's performance (2%-57%). Furthermore, our study highlights the importance of incorporating edge nodes in addressing the stringent application requirements concerning latency (storage and retrieval latency time improved by 22%). The colocation of edge node results in decreasing further the experienced latency (linearly to the colocation increase), demonstrating the value of considering colocation when developing mixed edge-cloud storage systems. In conclusion, our work provides valuable insights into the design of efficient and reliable storage systems that leverage the advantages of both edge and cloud nodes, contributing to the development of robust distributed storage infrastructures that efficiently address the growing demands of the digital era.

Bibliography

[edg] The economics of edge computing. https://edgecomputing-news.com/2020/10/29/analysis-economics-of-edge-computing.

[gur] Gurobi optimizer. https://www.gurobi.com/.

[3] NEPHELE project. http://www.nepheleproject.eu/.

[4] NEPHELE traffic generator. https://github.com/kchristodou/Datacenter-network-traffic-generator.

[sky] Skyflok. https://www.skyflok.com/.

[6] Abu-Libdeh, H., Princehouse, L., and Weatherspoon, H. (2010). RACS: a case for cloud storage diversity. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 229–240.

[7] Al-Abbasi, A. O. and Aggarwal, V. (2020). TTLCache: Taming latency in erasure-coded storage through TTL caching. *IEEE Transactions on Network and Service Management*, 17(3):1582–1596.

[8] Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74.

[9] Anderson, T. E., Owicki, S. S., Saxe, J. B., and Thacker, C. P. (1993). High-speed switch scheduling for local-area networks. *ACM Transactions on Computer Systems (TOCS)*, 11(4):319–352.

[10] Bacis, E., di Vimercati, S. D. C., Foresti, S., Paraboschi, S., Rosa, M., and Samarati, P. (2019a). Dynamic allocation for resource protection in decentralized cloud storage. In *2019 IEEE global communications conference (GLOBECOM)*, pages 1–6. IEEE.

[11] Bacis, E., di Vimercati, S. D. C., Foresti, S., Paraboschi, S., Rosa, M., and Samarati, P. (2019b). Securing resources in decentralized cloud storage. *IEEE Transactions on Information Forensics and Security*, 15:286–298.

[12] Bakopoulos, P., Christodoulopoulos, K., Landi, G., Aziz, M., Zahavi, E., Gallico, D., Pitwon, R., Tokas, K., Patronas, I., Capitani, M., et al. (2018). NEPHELE: An end-to-end scalable and dynamically reconfigurable optical architecture for application-aware sdn cloud data centers. *IEEE Communications Magazine*, 56(2):178–188.

[13] Ben-Itzhak, Y., Caba, C., Schour, L., and Vargaftik, S. (2016). C-share: Optical circuits sharing for software-defined data-centers. *arXiv preprint arXiv:1609.04521*.

[14] Benson, T., Akella, A., and Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280.

[15] Benzaoui, N., Estarán, J., Dutisseuil, E., Mardoyan, H., De Valicourt, G., Dupas, A., Van, Q. P., Verchere, D., Uščumlić, B., Gonzalez, M. S., et al. (2018). CBOSS: bringing traffic engineering inside data center networks. *Journal of Optical Communications and Networking*, 10(7):B117–B125.

[16] Bermbach, D., Klems, M., Tai, S., and Menzel, M. (2011). Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 452–459. IEEE.

[17] Bessani, A., Correia, M., Quaresma, B., André, F., and Sousa, P. (2013). DepSky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):1–33.

[18] Birkhoff, G. (1946). Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucuman, Ser. A*, 5:147–154.

[19] Bongiovanni, G., Coppersmith, D., and Wong, C. (1981). An optimum time slot assignment algorithm for an ss/tdma system with variable number of transponders. *IEEE Transactions on Communications*, 29(5):721–726.

[20] Calabretta, N. and Miao, W. (2018). Optical switching in data centers: Architectures based on optical packet/burst switching. *Optical Switching in Next Generation Data Centers*, pages 45–69.

[21] Cerutti, I., Andriolli, N., Pintus, P., Faralli, S., Gambini, F., Liboiron-Ladouceur, O., and Castoldi, P. (2015). Fast scheduling based on iterative parallel wavelength matching for a multi-wavelength ring network-on-chip. In *2015 International Conference on Optical Network Design and Modeling (ONDM)*, pages 180–185. IEEE.

[22] Christodoulopoulos, K., Kontodimas, K., Siokis, A., Yiannopoulos, K., and Varvarigos, E. (2017). Efficient bandwidth allocation in the nephele optical/electrical datacenter interconnect. *Journal of Optical Communications and Networking*, 9(12):1145–1160.

[23] Christodoulopoulos, K., Kontodimas, K., Yiannopoulos, K., and Varvarigos, E. (2016). Bandwidth allocation in the nephele hybrid optical interconnect. In *2016 18th International Conference on Transparent Optical Networks (ICTON)*, pages 1–4. IEEE.

[24] Christodoulopoulos, K., Lugones, D., Katrinis, K., Ruffini, M., and O'Mahony, D. (2015). Performance evaluation of a hybrid optical/electrical interconnect. *Journal of Optical Communications and Networking*, 7(3):193–204.

[25] Confais, B., Rostirolla, G., Parrein, B., Lacan, J., and Marques, F. (2022). *Mutida: A Rights Management Protocol for Distributed Storage Systems Without Fully Trusted Nodes*, pages 1–34. Springer Berlin Heidelberg, Berlin, Heidelberg.

[26] Di Vimercati, S. D. C., Foresti, S., Livraga, G., Piuri, V., and Samarati, P. (2017). Supporting user requirements and preferences in cloud plan selection. *IEEE Transactions on Services Computing*, 14(1):274–285.

[27] di Vimercati, S. D. C., Foresti, S., Livraga, G., Piuri, V., and Samarati, P. (2019). Security-aware data allocation in multicloud scenarios. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2456–2468.

[28] Duc, T. L., Leiva, R. G., Casari, P., and Östberg, P.-O. (2019). Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. *ACM Computing Surveys (CSUR)*, 52(5):1–39.

[29] Eng, K. and Acampora, A. (1987). Fundamental conditions governing tdm switching assignments in terrestrial and satellite networks. *IEEE transactions on communications*, 35(7):755–761.

[30] Farrington, N., Porter, G., Radhakrishnan, S., Bazzaz, H. H., Subramanya, V., Fainman, Y., Papen, G., and Vahdat, A. (2010). Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 339–350.

[31] Follows, J. and Straeten, D. (1999). *Application driven networking: Concepts and architecture for policy-based systems*. IBM Corporation.

[32] Golestani, S. J. (1991). A framing strategy for congestion management. *IEEE Journal on Selected Areas in Communications*, 9(7):1064–1077.

[33] Hadji, M. (2015). Scalable and cost-efficient algorithms for reliable and distributed cloud storage. In *International Conference on Cloud Computing and Services Science*, pages 15–37. Springer.

[34] He, S. and Torkelson, M. (1996). A new approach to pipeline fft processor. In *Proceedings of International Conference on Parallel Processing*, pages 766–770. IEEE.

[35] Hopcroft, J. E. and Karp, R. M. (1973). An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231.

[36] Hui, J. Y. (2012). *Switching and traffic theory for integrated broadband networks*, volume 91. Springer Science & Business Media.

[37] IDC and Seagate (2017). Data age 2025: The evolution of data to life-critical.

[38] Inc., T. M. (2017). Matlab.

[39] Index, C. V. N. (2016). Forecast and methodology, 2015–2020. *White paper*, pages 1–41.

[40] Inukai, T. (1979). An efficient SS/TDMA time slot assignment algorithm. *IEEE Transactions on Communications*, 27(10):1449–1455.

[41] Jarschel, M., Wamser, F., Hohn, T., Zinner, T., and Tran-Gia, P. (2013). Sdn-based application-aware networking on the example of youtube video streaming. In *2013 Second European Workshop on Software Defined Networks*, pages 87–92. IEEE.

[42] Kachris, C. and Tomkos, I. (2012). A survey on optical interconnects for data centers. *IEEE Communications Surveys & Tutorials*, 14(4):1021–1036.

[43] Kontodimas, K., Christodoulopoulos, K., and Varvarigos, E. (2020). Simplifying optical dcn fabrics with blocking space switching and wavelength-constrained wdm. In *Optical Network Design and Modeling: 23rd IFIP WG 6.10 International Conference, ONDM 2019, Athens, Greece, May 13–16, 2019, Proceedings 23*, pages 286–298. Springer.

[44] Kontodimas, K., Christodoulopoulos, K., Zahavi, E., and Varvarigos, E. (2018). Resource allocation in slotted optical data center networks. In *2018 International Conference on Optical Network Design and Modeling (ONDM)*, pages 248–253. IEEE.

[45] Kontodimas, K., Soumplis, P., Kretsis, A., Kokkinos, P., Fehér, M., Lucani, D. E., and Varvarigos, E. (2023). Secure distributed storage orchestration on heterogeneous cloud-edge infrastructures. *IEEE Transactions on Cloud Computing*.

[46] Kontodimas, K., Soumplis, P., Kretsis, A., Kokkinos, P., and Varvarigos, E. (2021). Secure distributed storage on cloud-edge infrastructures. In *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, pages 127–132. IEEE.

[47] Lee, H. Y., Hwang, F. K., and Carpinelli, J. D. (1996). A new decomposition algorithm for rearrangeable clos interconnection networks. *IEEE Transactions on Communications*, 44(11):1572–1578.

[48] Li, J. and Li, B. (2013). Erasure coding for cloud storage systems: A survey. *Tsinghua Science and Technology*, 18(3):259–272.

[49] Liu, G., Shen, H., and Wang, H. (2017). An economical and slo-guaranteed cloud storage service across multiple cloud service providers. *IEEE Transactions on Parallel and Distributed Systems*, 28(9):2440–2453.

[50] Liu, K., Peng, J., Wang, J., Huang, Z., and Pan, J. (2022). Adaptive and scalable caching with erasure codes in distributed cloud-edge storage systems. *IEEE Transactions on Cloud Computing*, pages 1–1.

[51] Ma, Y., Nandagopal, T., Puttaswamy, K. P., and Banerjee, S. (2013). An ensemble of replication and erasure codes for cloud file systems. In *2013 Proceedings IEEE INFOCOM*, pages 1276–1284. IEEE.

[52] Mansouri, Y., Toosi, A. N., and Buyya, R. (2013). Brokering algorithms for optimizing the availability and cost of cloud storage services. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 1, pages 581–589. IEEE.

[53] McKeown, N. (1999). The islip scheduling algorithm for input-queued switches. *IEEE/ACM transactions on networking*, 7(2):188–201.

[54] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74.

[55] McKeown, N., Mekkittikul, A., Anantharam, V., and Walrand, J. (1999). Achieving 100% throughput in an input-queued switch. *IEEE Transactions on Communications*, 47(8):1260–1267.

[56] Mellette, W. M., McGuinness, R., Roy, A., Forencich, A., Papen, G., Snoeren, A. C., and Porter, G. (2017). Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 267–280.

[57] Mellette, W. M., Schuster, G. M., Porter, G., Papen, G., and Ford, J. E. (2016). A scalable, partially configurable optical switch for data center networks. *Journal of Lightwave Technology*, 35(2):136–144.

[58] Mu, S., Chen, K., Gao, P., Ye, F., Wu, Y., and Zheng, W. (2012). μlibcloud: Providing high available and uniform accessing to multiple cloud storages. In *2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 201–208. IEEE.

[59] Networking, C. V. (2013). Cisco global cloud index: Forecast and methodology, 2014–2019. *white paper*.

[60] Papaioannou, T. G., Bonvin, N., and Aberer, K. (2012). Scalia: An adaptive scheme for efficient multi-cloud storage. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–10. IEEE.

[61] Patronas, I., Gkatzios, N., Kitsakis, V., Reisis, D., Christodoulopoulos, K., and Varvarigos, E. (2018). Scheduler accelerator for TDMA data centers. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 162–169. IEEE.

[62] Peng, S., Guo, B., Jackson, C., Nejabati, R., Agraz, F., Spadaro, S., Bernini, G., Ciulli, N., and Simeonidou, D. (2015). Multi-tenant software-defined hybrid optical switched data centre. *Journal of Lightwave Technology*, 33(15):3224–3233.

[63] Porter, G., Strong, R., Farrington, N., Forencich, A., Chen-Sun, P., Rosing, T., Fainman, Y., Papen, G., and Vahdat, A. (2013). Integrating microsecond circuit switching into the data center. *ACM SIGCOMM Computer Communication Review*, 43(4):447–458.

[64] Poutievski, L., Mashayekhi, O., Ong, J., Singh, A., Tariq, M., Wang, R., Zhang, J., Beauregard, V., Conner, P., Gribble, S., Kapoor, R., Kratzer, S., Li, N., Liu, H., Nagaraj, K., Ornstein, J., Sawhney, S., Urata, R., Vicisano, L., Yasumura, K., Zhang, S., Zhou, J., and Vahdat, A. (2022). Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of ACM SIGCOMM 2022*.

[65] Roy, A., Zeng, H., Bagga, J., Porter, G., and Snoeren, A. C. (2015). Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137.

[66] Ryser, H. J. (1963). *Combinatorial mathematics*, volume 14. American Mathematical Soc.

[67] Saridis, G. M., Peng, S., Yan, Y., Aguado, A., Guo, B., Arslan, M., Jackson, C., Miao, W., Calabretta, N., Agraz, F., et al. (2016). Lightness: A function-virtualizable software defined data center network with all-optical circuit/packet switching. *Journal of Lightwave Technology*, 34(7):1618–1627.

[68] Serpanos, D. N. and Antoniadis, P. (2000). Firm: A class of distributed scheduling algorithms for high-speed atm switches with multiple input queues. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, volume 2, pages 548–555. IEEE.

[69] Sharov, A., Shraer, A., Merchant, A., and Stokely, M. (2015). Take me to your leader! online optimization of distributed storage configurations.

[70] Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646.

[71] Singh, H. J. and Bawa, S. (2022). Lameta: An efficient locality aware metadata management technique for an ultra-large distributed storage system. *Journal of King Saud University - Computer and Information Sciences*, 34(10, Part A):8323–8335.

[72] Singla, A., Singh, A., Ramachandran, K., Xu, L., and Zhang, Y. (2010). Proteus: a topology malleable data center network. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6.

[73] Su, M., Zhang, L., Wu, Y., Chen, K., and Li, K. (2015). Systematic data placement optimization in multi-cloud storage for complex requirements. *IEEE Transactions on Computers*, 65(6):1964–1977.

[74] Tassiulas, L. (1998). Linear complexity algorithms for maximum throughput in radio networks and input queued switches. In *Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98*, volume 2, pages 533–539. IEEE.

[75] Towles, B. and Dally, W. J. (2003). Guaranteed scheduling for switches with configuration overhead. *IEEE/ACM Transactions on Networking*, 11(5):835–847.

[76] Vargaftik, S., Caba, C., Schour, L., and Ben-Itzhak, Y. (2020). C-share: Optical circuits sharing for software-defined data-centers. *ACM SIGCOMM Computer Communication Review*, 50(1):2–9.

[77] Wang, G., Andersen, D. G., Kaminsky, M., Papagiannaki, K., Ng, T. E., Kozuch, M., and Ryan, M. (2010). c-Through: Part-time optics in data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 327–338.

[78] Wang, P., Zhao, C., Liu, W., Chen, Z., and Zhang, Z. (2020a). Optimizing data placement for cost effective and high available multi-cloud storage. *Computing and Informatics*, 39(1-2):51–82.

[79] Wang, P., Zhao, C., Wei, Y., Wang, D., and Zhang, Z. (2020b). An adaptive data placement architecture in multicloud environments. *Scientific Programming*, 2020.

[80] Wang, P., Zhao, C., and Zhang, Z. (2018). An ant colony algorithm-based approach for cost-effective data hosting with high availability in multi-cloud environments. In *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*, pages 1–6. IEEE.

[81] Wei, Q., Veeravalli, B., Gong, B., Zeng, L., and Feng, D. (2010). CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster. In *2010 IEEE international conference on cluster computing*, pages 188–196. IEEE.

[82] Wu, H., Deng, S., Li, W., Yin, J., Li, X., Feng, Z., and Zomaya, A. Y. (2019). Mobility-aware service selection in mobile edge computing systems. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 201–208. IEEE.

[83] Wu, Z., Butkiewicz, M., Perkins, D., Katz-Bassett, E., and Madhyastha, H. V. (2013). Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 292–308.

[84] Yeung, K. L. (2001). Efficient time slot assignment algorithms for tdm hierarchical and nonhierarchical switching systems. *IEEE Transactions on Communications*, 49(2):351–359.

[85] Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J., and Yang, X. (2017). A survey on the edge computing for the internet of things. *IEEE access*, 6:6900–6919.

[86] Zhang, Q., Li, S., Li, Z., Xing, Y., Yang, Z., and Dai, Y. (2015). CHARM: A cost-efficient multi-cloud data hosting scheme with high availability. *IEEE Transactions on Cloud computing*, 3(3):372–386.

[87] Zhang-Shen, R. and McKeown, N. (2004). Designing a predictable internet backbone network. HotNets.

[88] Zhang-Shen, R. and McKeown, N. (2005). Designing a predictable internet backbone with valiant load-balancing. In *International Workshop on Quality of Service*, pages 178–192. Springer.

Curriculum Vitae

Personal Information

FIRST NAME: **Konstantinos**
LAST NAME: **Kontodimas**
FATHER'S NAME: Georgios
Contact information:
ADDRESS: Ekalis 4, 11636, Athens
TEL.: +306970449953
EMAIL: kontodimask@gmail.com

Education

2016-2023 **Doctoral student, School of Electrical & Computer Engineering, National Technical University of Athens**
Field: "Efficient resource allocation in data centers with dynamic optical networking infrastructures"
Supervisor: Prof. Emmanuel VARVARIGOS

2016 **MSc degree in Computer Science & Engineering, Dept. of Computer Engineering and Informatics, University of Patras**
Thesis: *"Analysis and evaluation of scheduling policies in consolidated I/O operations"*

2014 **Dipl.-Ing. degree in Computer Engineering & Informatics, Dept. of Computer Engineering and Informatics, University of Patras**
Thesis: *"Implementation of a math-heuristic algorithm for routing and spectrum allocation in elastic fiber optic networks"*

List of Publications

Articles in Scientific Journals

1. Kontodimas, K., Christodoulopoulos, K., & Varvarigos, E. (2023). **A Lean and Fast Optical Datacenter Interconnection Fabric with Partial Configurability.** *(submitted to Optical Switching and Networking (OSN) – under review)*

2. Kontodimas, K., Soumplis, P., Kretsis, A., Kokkinos, P., Fehér, M., Lucani, D. E., & Varvarigos, E. (2023). **Secure Distributed Storage Orchestration on Heterogeneous Cloud-Edge Infrastructures.** *IEEE Transactions on Cloud Computing.* DOI: 10.1109/TCC.2023.3287653

3. Bakopoulos, P., Christodoulopoulos, K., Landi, G., Aziz, M., Zahavi, E., Gallico, D., Pitwon, R., Tokas, K., Patronas, I., Capitani, M., Spatharakis, C., Yiannopoulos, K., Wang, K., Kontodimas, K., Lazarou, I., Wieder, P., Reisis, D., Varvarigos, E., Biancani, M., & Avramopoulos, H. (2018). **NEPHELE: An end-to-end scalable and dynamically reconfigurable optical architecture for application-aware SDN cloud data centers.** *IEEE Communications Magazine,* 56(2), 178-188. DOI: 10.1109/MCOM.2018.1600804.

4. Landi, G., Capitani, M., Kretsis, A., Kontodimas, K., Kokkinos, P., Gallico, D., Biancani, M., Christodoulopoulos, K., & Varvarigos, E. (2018). **Inter-domain optimization and orchestration for optical datacenter networks.** *Journal of Optical Communications and Networking,* 10(7), B140-B151. DOI: 10.1364/JOCN.10.00B140

5. Christodoulopoulos, K., Kontodimas, K., Siokis, A., Yiannopoulos, K., & Varvarigos, E. (2017). **Efficient bandwidth allocation in the NEPHELE optical/electrical datacenter interconnect.** *Journal of Optical Communications and Networking,* 9(12), 1145-1160. DOI: 10.1364/JOCN.9.001145

Publications in Conference Proceedings

1. Kontodimas, K., Soumplis, P., Kretsis, A., Kokkinos, P., & Varvarigos, E. (2021, November). **Secure distributed storage on cloud-edge infrastructures.** In *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)* (pp. 127-132). IEEE. DOI: 10.1109/CloudNet53349.2021.9657156

2. Kontodimas, K., Christodoulopoulos, K., & Varvarigos, E. (2020). **Simplifying Optical DCN Fabrics with Blocking Space Switching and Wavelength-Constrained**

WDM. In *Optical Network Design and Modeling: 23rd IFIP WG 6.10 International Conference, ONDM 2019, Athens, Greece, May 13–16, 2019, Proceedings 23* (pp. 286-298). Springer International Publishing. DOI: 10.1007/978-3-030-38085-4_25

3. Christodoulopoulos, K., Kontodimas, K., Dembeck, L., & Varvarigos, E. (2019, March). **Slotted optical datacenter networks with sub-wavelength resource allocation.** In *Optical Fiber Communication Conference* (pp. W1J-1). Optica Publishing Group. DOI: 10.1364/OFC.2019.W1J.1

4. Kontodimas, K., Christodoulopoulos, K., Zahavi, E., & Varvarigos, E. (2018, May). **Resource allocation in slotted optical data center networks.** In *2018 International Conference on Optical Network Design and Modeling (ONDM)* (pp. 248-253). IEEE. DOI: 10.23919/ONDM.2018.8396140

5. Landi, G., Patronas, I., Kontodimas, K., Aziz, M., Christodoulopoulos, K., Kyriakos, A., Capitani, M., Hamedani, A. F., Reisis, D., Varvarigos, E., Bakopoulos, P., & Avramopoulos, H. (2017, March). **SDN control framework with dynamic resource assignment for slotted optical datacenter networks.** In *Optical Fiber Communication Conference* (pp. Tu3L-1). Optica Publishing Group. DOI: 10.1364/OFC.2017.Tu3L.1

6. Yiannopoulos, K., Kontodimas, K., Christodoulopoulos, K., & Varvarigos, E. (2017, July). **Resource partitioning in the NEPHELE datacentre interconnect.** In *2017 19th International Conference on Transparent Optical Networks (ICTON)* (pp. 1-4). IEEE. DOI: 10.1109/ICTON.2017.8024738

7. Christodoulopoulos, K., Kontodimas, K., Siokis, A., Yiannopoulos, K., & Varvarigos, E. (2016, December). **Collisions free scheduling in the NEPHELE hybrid electrical/optical datacenter interconnect.** In *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)* (pp. 368-371). IEEE. DOI: 10.1109/ICECS.2016.7841209

8. Christodoulopoulos, K., Kontodimas, K., Yiannopoulos, K., & Varvarigos, E. (2016, July). **Bandwidth allocation in the NEPHELE hybrid optical interconnect.** In *2016 18th International Conference on Transparent Optical Networks (ICTON)* (pp. 1-4). IEEE. DOI: 10.1109/ICTON.2016.7550704

List of older Articles in Scientific Journals

1. Kontodimas, K., Kokkinos, P., Kuperman, Y., Houbavlis, A., & Varvarigos, E. (2017). **Analysis and evaluation of scheduling policies for consolidated I/O operations.** *Journal of Grid Computing,* 15(1), 107-125. DOI: 10.1007/s10723-017-9392-4

2. Bouras, C., Diles, G., Kokkinos, V., Kontodimas, K., & Papazois, A. (2014). **A simulation framework for evaluating interference mitigation techniques in heterogeneous cellular environments.** *Wireless Personal Communications,* 77, 1213-1237. DOI: 10.1007/s11277-013-1562-5

List of older Publications in Conference Proceedings

1. Kontodimas, K., Kokkinos, P., Kuperman, Y., & Varvarigos, E. (2015, December). **Analysis and evaluation of I/O hypervisor scheduling.** In *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)* (pp. 45-54). IEEE. DOI: 10.1109/UCC.2015.19

2. Akribopoulos, O., Amaxilatis, D., Georgitzikis, V., Logaras, M., Keramidas, V., Kontodimas, K., Lagoudianakis, …, & Chatzigiannakis, I. (2013). **Making p-space smart: Integrating iot technologies in a multi-office environment.** In *Mobile Wireless Middleware, Operating Systems, and Applications: 5th International Conference, Mobilware 2012,* Berlin, Germany, November 13-14, 2012, Revised Selected Papers 5 (pp. 31-44). Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-36660-4_3

3. Bouras, C., Kokkinos, V., Kontodimas, K., & Papazois, A. (2012, October). **A simulation framework for LTE-A systems with femtocell overlays.** In *Proceedings of the 7th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks* (pp. 85-90). DOI: 10.1145/2387191.2387204

4. Alexiou, A., Bouras, C., Kokkinos, V., Kontodimas, K., & Papazois, A. (2011, October). **Interference behavior of integrated femto and macrocell environments.** In *2011 IFIP Wireless Days (WD)* (pp. 1-5). IEEE. DOI: 10.1109/WD.2011.6098161

List of Awards and Scholarships

1. **Best Paper Award** in *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)* for **Secure distributed storage on cloud-edge infrastructures.** (Kontodimas, K., Soumplis, P., Kretsis, A., Kokkinos, P., & Varvarigos, E.)

2. **Scholarship** for Doctorate Research, funded by the State Scholarship Foundation (IKY).