Patronus: Controlling Thousands of Geo-Distributed Micro Data Centers

Sangeetha Abdu Jyothi†, Ruiyang Chen†, Aditya Akella‡, Brighten Godfrey†
†University of Illinois, Urbana-Champaign ‡University of Wisconsin, Madison

Abstract

Micro Data Centers (MDCs), currently being deployed at the edge of service provider networks, are expected to number in the thousands to support latency-sensitive VNFs, geo-distributed monitoring and analytics, connected cars, and other emerging applications. Managing geographically-separated MDCs is challenging due to the combination of constrained resources in each MDC, latency and bandwidth between MDCs and stringent requirements of applications running within and across MDCs. In this paper, we show the need for a dynamic, scalable resource management scheme in this emerging environment, which we call a Wide Area Network of Data centers (WAND). The geo-distributed nature of the infrastructure and applications necessitate novel API and control schemes. To handle the dual burden of limited resource availability and stringent application needs, we design Patronus, a closed-loop control mechanism that accurately predicts demands, efficiently allocates resources, and adapts to variations, at each individual MDC and collectively across thousands of MDCs. Using central office locations of a cellular provider and realistic traffic data, we show that Patronus can improve the peak resource usage by up to 47% while meeting application requirements.

1 Introduction

Micro data centers (MDCs) at the edge are emerging as prominent components in the Internet infrastructure. Traditionally, MDCs were used for content caching and video performance optimization in Content Delivery Networks [12] and for user load-balancing at entry/exit of content provider networks such as Google [38], Facebook [31], and Microsoft [21]. With the emergence of novel applications with stringent performance requirements, the role of MDCs is expanding. MDCs providing limited compute resources very close to users are increasingly used for supporting a variety of low-latency applications, often augmenting, and sometimes replacing, the traditional hyperscale clouds.

An important use case for increased processing at the edge is the burgeoning Internet of Things (IoT). IoT devices produce data in the form of video, voice, sensor information, etc., which needs to be analyzed and acted upon, often within tight delay constraints. Processing and compressing data from such applications at the edge will also reduce the bandwidth demand in the core. Availability of compute in close proximity is a must-have for offloading Augmented and Virtual Reality (AR/VR) that operate at timescales comparable to the sensitivity of human perception. Self-driving and connected cars will also increasingly rely on these edge DCs [5, 6]. MDCs can also prove beneficial to other latency-sensitive applications with heavy compute requirements such as real-time video analytics and online gaming.

Today, carrier networks are spearheading large-scale deployments of MDCs for supporting Virtualized Network Functions (VNFs). Service providers like AT&T [4] and Verizon [34] are converting traditional Central Offices (COs) with dedicated hardware to MDCs with off-the-shelf servers. A single carrier operates a few thousand COs – AT&T has 4700 COs in the US [8] – and with the advent of 5G, the number of MDC sites is expected to grow further. In addition to the critical cellular services, these MDCs are also designed to support emerging applications like those above [6].

Thus, Wide Area Networks (WANs) are morphing from a network of routers to an interconnection between a multitude of geographically distributed micro data centers. We refer to this emerging model as a WAN As a Network of Data centers (WAND). WAND is indispensable to providing support for the emerging hyperconnected world with billions of devices and geo-distributed applications.

In order to meet the goals of geo-distributed applications while simultaneously utilizing resources efficiently, we need an efficient WAND resource management scheme. However, this is difficult for several reasons. First, the combination of scale and geographic spread has not been addressed by prior large-scale systems (§ 2.1 Table 1). Second, the environment needs to support a motley set of applications with diverse requirements. This includes long-running streaming...
applications (e.g., cellular VNFs, other middlebox service chains), batch analytics (e.g., cellular log analytics, Hadoop jobs) and Lambda-like short-lived jobs (e.g., elastic web servers). To meet the requirements of such geo-distributed high-performance applications, resource allocation on distributed MDCs and the interconnecting WAN will need to be coordinated (§ 2.2). Third, the smaller size of MDCs and the potential for demand bursts mean that resource availability in any particular MDC will be more dynamic and variable than in a hyperscale DC. In other words, MDCs enjoy limited benefits of statistical multiplexing (§ 2.2). Thus, WAND is characterized by its scale, geographic spread, diversity of applications, and limited resources at MDCs. While one or two of these challenges have been addressed in existing large-scale systems [17, 15, 21, 31, 38], the combination of all characteristics calls for novel resource management techniques in WAND.

We design an autonomous WAND control system, Patronus, which enables high utilization of the infrastructure while improving the performance of applications through (a) fast and efficient resource allocation, and (b) intelligent adaptation during variations. The centralized control plane of Patronus has several components. First, Patronus introduces a simple and expressive API which can support representation of diverse requirements of WAND applications using WAND tags. Tags encode bounds on application requirements such as latency, bandwidth, location preference, deadline, etc. Second, Patronus leverages predictability of traffic and resource usage patterns inherent to the WAND environment/applications to obtain a long-term perspective on resource availability and usage.

Third, Patronus provides fast and efficient resource allocation in the complex WAND environment through division of labor across an instantaneous scheduler and a long-term scheduler. The long-term scheduler relies on long-term predictions for packing of jobs across time while complying with deadlines, fairness, etc. The instantaneous scheduler allocates resources to a subset of tasks deemed active in the current instant by the long-term scheduler. Both schedulers rely on hierarchical optimization for handling diverse application objectives and a simple mechanism for converting WAND tags to constraints.

In this paper, we make the following contributions:

- We show that Patronus can schedule resources across thousands of MDCs in seconds while reducing peak resource usage by up to 47%.
- Using realistic data, we evaluate trade-offs in performance and control overhead using closed-loop control in WAND to meet the high-level goals of applications and operators.
- We show that Patronus can schedule resources across thousands of MDCs in seconds while reducing peak resource usage by up to 47%.

2 Background & Motivation

In this section, we illustrate the differentiating features of WAND that necessitate novel control solutions.

2.1 Related Domains

One may draw parallels between control of thousands of data centers and interconnecting network in WAND and several well-studied control domains such as clusters, traditional ISPs, private WANs, geo-distributed analytics, etc. Next, we argue why control solutions in these domains cannot be borrowed easily for WAND (overview in Table 1).

Traditional ISPs: Traffic engineering is a well-studied problem in traditional ISP networks with a single resource (geographically-distributed network). While the problem has been tackled from different perspectives ranging from oblivious routing [3, 20] to traffic-adaptive schemes [18, 36, 11, 25], these solutions do not apply to the WAND model. Extending network traffic classes and prioritization to an environment with demands and performance constraints on multiple resources (WAN network and DC resources like CPU, memory, etc.) is a non-trivial challenge. Besides, the presence of new applications over which the service provider has complete control (e.g., background analytics jobs) will help the provider to drive the network and DCs to high utilization. This was difficult in traditional ISPs which mostly dealt with user-driven inelastic traffic. As ISPs move to SDN-based control, there are also efforts to scale SDN with distributed controllers. Recursive SDN [23] proposed a hierarchical solution for geo-distributed carrier networks. RSDN and other work [9, 19] in distributing SDN control, however, only considers network control.

Private inter-data center WANs: Inter-DC private WAN solutions such as B4 [17] and SWAN [15] optimize resources across multiple applications similar to WAND. However, private WANs with a sparse network connecting tens of hyperscale data centers operate at a much smaller scale (in terms of number of distinct sites) compared to the carrier environment composed of thousands of micro data centers and interconnecting links. Moreover, the proposed solutions only focus on the bandwidth requirements of the private WAN since the DCs are hyperscale and relatively less constrained (or at least have more reliable statistical multiplexing). In WAND, both DCs and the WAN are frequently constrained. Furthermore, quickly adapting to traffic variations is more critical.
in WAND due to the limited capacity of MDCs that cannot accommodate bursts.

**Cluster schedulers:** Cluster schedulers [32, 14, 27] are responsible for allocation of server resources within hyperscale DCs. They consider demands over multiple types of resources and strive to drive the cluster to high utilization, often with prioritization across applications. They deal with large scale, e.g., allocating at the machine or rack granularity across many racks. However, the resources are highly localized (typically within a single DC) in contrast with a geo-distributed WAND. Moreover, cluster schedulers are typically decoupled from the network controllers which manage symmetric high-bisection bandwidth intra-DC interconnect between servers. This separation of control may not work well in a WAND environment which has an irregular topology and expensive WAN links.

**Traffic-aware VM placement in data centers:** Traffic-aware VM placement is a related domain with joint server and network resource management within a DC. However, this problem is NP-hard, while the WAND resource management can be tackled using a Linear Program. In intra-DC environments, flexibility in VM placement translates to flexibility in the location of source and destination of the flow [24]. This optimization problem can be reduced to the Quadratic Assignment Problem (QAP) which is NP-hard. In the WAND model, the source and the destination of the flow are fixed (typically at the end-user). We have flexibility in choosing way-points, i.e., DCs. Thus, WAND resource management is a resource-augmented Multi-Commodity Flow (MCF) problem with location restrictions. The challenges in WAND are related to scalability and quick adaptation.

**Application-specific solutions:** There exist several solutions tailored to specific geo-distributed applications.  
(a) **NFV solutions:** Elastic Edge [28] is an application-agnostic framework which allocates resources across NFVs. However, this solution is restricted to scheduling within a single DC. PEPC [7] introduced a new architecture for the cellular core to solve state duplication across EPC components and improved throughput at bottlenecked VNFs. KLEIN [30], another cellular-specific solution, put forward a 3GPP-compliant solution for load-balancing cellular traffic across MDCs. However, this does not take into account the presence of multiple applications with diverse performance needs in the DCs, or the constraints on the interconnecting network between DCs.  
(b) **Geo-distributed analytics:** Solutions for big data analytics across the wide area [29, 35] offer optimization for low-latency processing of queries, but do not deal with the difficult problem of cross-application optimization.

**WAND:** Having established that existing resource management techniques cannot be directly extended to generic WAND, we give a brief overview of current domain-specific solutions offered by provider networks.  
(a) **Carriers:** AT&T [4] and Verizon [34] have published white papers on SDN/NFV reference architectures and are building platforms with similar features such as network and DC controllers, data monitoring, a policy module, an orchestration module, etc. In addition to cellular traffic, cellular MDCs also intend to support other micro-services and applications such as connected cars [6]. These platforms, however, are in the early stages of development and do not currently offer solutions for resource management at the scale of thousands of MDCs.  
(b) **Content Providers:** Points of Presence (PoPs) of large content providers like Google [38], Microsoft [21], and Facebook [31] also constitute a WAND. However, solutions in this domain primarily focus on load-balancing traffic across edge locations. Currently, they do not offer compute resources for other applications at the edge.  
(c) **CDNs:** Content Delivery Networks cache objects at the edge to serve user requests with minimal latency. CDNs are a sub-class of WAND where traffic is bandwidth-heavy, and edge resources are mostly used for storage and delivery. Like content providers, they do not typically support other user applications. Application-specific video delivery optimiza-

<table>
<thead>
<tr>
<th>Property</th>
<th>WAND</th>
<th>Traditional ISPs</th>
<th>Private Inter-DC WAN</th>
<th>Cluster Schedulers</th>
<th>Application-specific (e.g., geo-distributed analytics, NFV platforms, etc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geographically fragmented resources</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Large scale</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Cross-application optimization for high utilization</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Demand over multiple resources</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Location constraints</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1: Comparison of features — WAND and other infrastructure
running on W AND, some traffic may be elastic — a provider
Elasticity of traffic: Depending on the types of applications
running on W AND, some traffic may be elastic — a provider
can control the rate and time of resource allocation in DCs
and WAN based on current resource availability. This can be
used for low priority traffic such as backups.

Limited statistical multiplexing: Edge DCs do not form a
single large pool of compute. Each DC is small, and there-
fore subject to more load variability due to less statistical
multiplexing. They are distributed geographically, hence
they are non-interchangeable for latency-sensitive applica-
tions. This may also lead to more variable utilization on
individual DCs during regional hotspots. Hence, efficient
dynamic resource management is paramount.

We quantify the impact of limited statistical multiplexing in
Figure 1. We use a real-world dataset from a DNS
provider comprising of DNS queries from across the US
over a single day tagged with origin zip code and timestamp
(more details about the dataset in § 6.1.1 b). For measuring
variability, we use the Coefficient of Variation, CV, defined
as the ratio of standard deviation to the mean. CV is a com-
parable measure for the extent of variability for distributions
with different means.

We divide the data with samples every minute into chunks
different lengths (15 min and 60 min) and compute CV
over each chunk at different geographic scales — per edge
location, per area1 and over the complete dataset (entire
country). In Figure 1, we plot the CDF of CV over all chunks
in one day across all edges/areas. We observe that CV is the
highest when locations are considered independently and the
lowest in the complete dataset. When the timescale is in-
creased from 15 min to 60 min, CV remains nearly the same
at larger scales while it increases more than 100× for DC-
level traffic. This shows that edges are subjected to higher
variability compared to traditional clouds and therefore, ben-
efits accruable from statistical multiplexing are limited.

3 WAND API

Before delving into the design of a WAND API that helps
applications specify their needs to the WAND controller, we
discuss common requirements of WAND applications.

3.1 Application requirements

Geo-distributed applications in WAND can be of two gen-
eral types: (a) streaming with real-time traffic or (b) batch
analytics which processes offline data. Each has a variety of
performance requirements; we describe examples in order to
understand how to build an expressive WAND API.

Latency: When an application prefers an MDC at the edge
over a distant hyperscale cloud with cheaper resources, the
main motivating factor is often latency. For user-facing ap-
plications, the latency constraint is primarily related to end-

1More details about area boundaries in § 6.1.2. Load in an area is the
sum of loads across all edge locations in that area
to-end latency experienced by users. Non-user facing applications may have latency requirements between different concurrent modules.

**Bandwidth:** An application running in an MDC may have bandwidth requirements to users or between different internal components.

**Deadlines:** Deadlines are primarily associated with batch processing jobs. They denote the final time before which the application expects an output. In applications with buffering capability, this can also be the maximum time to fill the buffer. For example, in video feeds to be processed, the edge, the cameras may have limited storage capacity. In such cases, when the analytics is not time-constrained, the video may be pulled by the edge MDC with some time flexibility.

**Intra-application dependencies:** In traditional batch processing jobs, the application is represented using a Directed Acyclic Graph (DAG) where the nodes are tasks and the edges denote dependencies. The same notation may be used in WAND. However, in distributed analytics, we may have additional constraints based on location data and latency/bandwidth constraints. Geo-distributed applications of streaming nature (NFV service chains, traditional stream processing, etc.) can also be represented using a graph. The key difference between streaming and batch jobs is that in streaming all components need to be active at the same time, whereas in batch jobs the tasks are executed sequentially in an order determined by dependencies.

In addition to task-level dependencies, WAND applications may also have state-based dependencies. For example, cellular customers directed to their home MDC does not incur control overhead related with state transfer, whereas those redirected to an MDC farther away will cause mobility-related overheads.

**External dependencies:** An application may have external dependencies on other jobs, type of resources, etc. For example, an analytics job on cellular traffic will prefer to be colocated with the cellular service chain. In addition to such affinity constraints, an application may have anti-affinity constraints (two customers may prefer to not have their VPN provider edges colocated).

**Evictions:** One of the fundamental characteristics a WAND has to deal with is limited statistical multiplexing due to the small size of MDCs. This manifests as high load variability and for medium- or low-priority applications, a risk of being evicted by a burst of load on a high-priority application. Applications may have different preferences regarding this risk, in at least two ways.

First, applications can differ in their tolerance for eviction, relative to the value they place on having resources close to users. An opportunistic application may prefer to grab resources at the edge whenever possible, if evictions are not especially costly (e.g., an opportunistic web cache at the edge can serve requests at lower latency when nearby MDC resources are available). On the other hand, a more conservative application might prefer to run in an MDC only when the probability of eviction is low (e.g., a real-time application serving self-driving cars).

Second, an application may wish to lessen the damage due to eviction by indicating which of its constituent modules are less critical. A video analytics application which processes CCTV feeds may have different levels of resource configuration [39] or a critical minimal set of cameras which provide high coverage. Specifying such intra-application criticality can increase the controller’s scheduling flexibility and thus increase the application’s chances of being scheduled.

**Grouping requirements:** We define a set of users which share the same characteristics, and therefore have a common set of requirements (latency, bandwidth, etc.), as a user group. For example, in cellular environment, the set of users assigned to a specific HSS may be one user group. Such grouping makes it more convenient for applications to specify their needs.

### 3.2 Defining a WAND API

We devise an API that allows geo-distributed WAND applications to represent their diverse requirements. The key operations supported by the API are given in Table 2.

Both streaming and batch applications are represented using a DAG. In streaming the dependencies denote the connections between tasks running concurrently. In batch applications, the dependencies denote the sequence of execution. A tag may be associated with a node (task) or an edge (dependency) in the application graph. An application can use these tags to denote its preferences such as bandwidth and latency requirements. A tag may be of the type `min`, `max` or `sum`. When a tag specifies a `min` value, it is the minimum requirement on that attribute required by the application (e.g., minimum bandwidth). When a tag is specified as `sum`, the limit applies to sum of the attribute across all resources with the same tag. For example, the end-to-end latency requirement of a service chain can be specified using this type, with no constraints on individual dependencies.

The `app_id` also acts as the tag for the entire application. Hence, to represent external dependencies, an application may use the `app_id` of the external application in the SetConstraints() function. A specific task of an external application can be represented using the format `app_id:tag`. In order to represent location constraints, the infrastructure provider can provide location tags, either at per-MDC level or at a regional-level.

Our WAND API also supports an extendable list of application preferences. Current supported preferences are `eviction_tolerance`, which specifies an approximate acceptable probability of eviction of a resource allocated to the application; and `optional_modules`, which specifies constituent...
tasks within the application that it prefers to be evicted first, rather than evicting the entire application.

Currently, Patronus supports linear constraints only since the schedulers rely on specialized Linear Programs. However, this is sufficient to handle the requirements of most WAND applications we considered.

## 4 WAND Control Plane

We design an automated control system, Patronus, to achieve high efficiency in the WAND environment. Patronus is a centralized controller managing resources across distributed MDCs and the interconnecting WAN. Patronus is designed for a private WAND infrastructure, such as that operated by cellular providers, with a variety of streaming and batch applications over which the infrastructure provider has some control and visibility. We do not currently consider external workloads, and hence adversarial traffic. Automated control in Patronus is realized with a sense-control-actuate loop composed of three main components: the prediction module, the resource allocation module, and the monitoring module.

### 4.1 Prediction Module

The prediction module is responsible for accurately estimating the requirements of applications. Depending on the information made available by applications, the prediction may involve several phases. For all applications, the prediction module estimates the resource needs for future instances based on history. The prediction module is also responsible for estimating user traffic patterns, such as diurnal behavior, across time to facilitate long-term resource planning. For blackbox applications where the dependencies and requirements are not explicitly known, the prediction module also estimates these dependencies between modules.

### 4.2 Resource Allocation Module

The resource allocation module is responsible for efficiently allocating resources across WAND applications, both streaming and batch jobs with a wide range of requirements. The scheduling involves two phases: long-term scheduling and instantaneous scheduling. The long-term scheduler maintains a long-term plan of the entire WAND system. It is responsible for handling complex requirements such as deadlines, location constraints, fairness, etc. At the beginning of each scheduling interval, the long-term scheduler shares the subset of tasks and the predicted loads to the instantaneous scheduler for immediate scheduling. The unallocated tasks are returned to the long-term scheduler for future scheduling or offloading to remote data centers. This separation allows Patronus to have fast and simple instantaneous scheduling, while conforming to complex application requirements across longer timescales.

The pre-processing phase in resource allocation involves the conversion of application details obtained from the API to a concise set of actionable requirements easily parsable by the schedulers. The $\text{min}$, $\text{max}$ and $\text{sum}$ values associated with tags provide bounds on latency and bandwidth for tasks/dependencies. Latency bounds also determine a subset of DCs where the task may be scheduled.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create ()</td>
<td>Returns app_id</td>
</tr>
<tr>
<td>setType (app_id, app_type)</td>
<td>$\text{app_type} \in { \text{streaming, batch} }$</td>
</tr>
<tr>
<td>setUserGroups (app_id, list[ (user_group_id, list[ &lt;tag_i&gt; ] ) ] )</td>
<td>$\text{user_group_id}$ may be IP prefix, tags are optional</td>
</tr>
<tr>
<td>setTasks (app_id, list[ (task_id, list[ &lt;tag_j&gt; ] ) ] )</td>
<td>each task may have one or more optional tags</td>
</tr>
<tr>
<td>setDependencies (app_id, list[ (task_id, task_id, list[ &lt;tag_mn&gt; ] ) ] )</td>
<td>each dependency may have one or more optional tags</td>
</tr>
<tr>
<td>setConstraints (app_id, list[ (tag_i, constraint_name, constraint_type, constraint_value) ] )</td>
<td>$\text{constraint_name} \in { \text{containers, latency, bandwidth, affinity, deadline} }$</td>
</tr>
<tr>
<td></td>
<td>$\text{constraint_type} \in { \text{min, max, sum} }$</td>
</tr>
<tr>
<td></td>
<td>For latency, value is in milliseconds</td>
</tr>
<tr>
<td></td>
<td>For bandwidth, value is bandwidth in Mbps</td>
</tr>
<tr>
<td></td>
<td>For affinity, value is 0 for anti-affinity, 1 for affinity</td>
</tr>
<tr>
<td></td>
<td>For deadline, value is UNIX epoch time</td>
</tr>
<tr>
<td>UpdateConstraints (app_id, list[ (tag_i, constraint_name, constraint_type, constraint_change) ] )</td>
<td>$\text{constraint_type} \in { \text{latency, bandwidth} }$ and</td>
</tr>
<tr>
<td></td>
<td>Signed integer indicating change in latency/bandwidth</td>
</tr>
<tr>
<td>setRedirectionOptions (app_id, redirect)</td>
<td>$\text{redirect} \in { \text{drop, edge, remote} }$</td>
</tr>
<tr>
<td>setApplicationPreferences (app_id, preference, value)</td>
<td>Extendable list of preference attributes. Current set:</td>
</tr>
<tr>
<td></td>
<td>$\text{preference} \in { \text{eviction_tolerance, optional_modules} }$</td>
</tr>
<tr>
<td></td>
<td>$\text{For eviction_tolerance, value is } (0, 1)$</td>
</tr>
<tr>
<td></td>
<td>$\text{For optional_modules: value is list[ (tag_j, weight_j) ]}$</td>
</tr>
<tr>
<td>setMonitorInterval (app_id, interval)</td>
<td>monitoring interval in milliseconds</td>
</tr>
</tbody>
</table>

Table 2: WAND API fields
Before we delve into the schedulers, we discuss hierarchical optimization and the conversion of application requirements to constraints which form the core of both long-term and instantaneous schedulers.

### 4.2.1 Hierarchical optimization

Patronus employs incremental *multi-objective optimization* for scheduling diverse applications with a wide range of requirements. Unlike traditional Linear Programming (LP) with a single objective, this technique supports multiple objectives within the same optimization. This allows the WAND controller to allocate resources of different types across diverse applications in a single optimization with priorities on applications/resources/locations determined by the operator.

**Objectives:** In the multi-objective optimization, each objective has four tunable parameters: priority \( P \), weight \( W \), absolute tolerance \( \alpha \) and relative tolerance \( \rho \). The LP solver repeats the optimization \( n \) times in the order of priorities, where \( n \) is the number of priority classes. When multiple objectives belong to the same priority class, the weight denotes the relative weight of each objective within the class. In this case, the aggregate objective of the priority class with multiple sub-objectives is given by the weighted sum. The absolute tolerance of an objective represents the absolute value of degradation in the optimal value tolerated while optimizing lower priority classes. The relative tolerance, \( \rho \in [0, \infty] \), denotes the degradation tolerable as a multiplicative factor of the optimal value.

A simple example with two objectives on DC utilization for a single application with a single task — nearest MDC assignment and load balancing at DCs is given below. The highest priority objective (higher P value) minimizes the fraction of traffic allocated to each DC weighted by the distance from each user-group to the MDC. This is the nearest-MDC objective. Allowing 50% tolerance on this optimal value, the next round of optimization minimizes \( K \), the difference in utilization between all pairs of MDCs. In short, this optimization supports nearest-MDC assignment with load-balancing across MDCs up to a certain limit of deviation from nearest MDC distance. For this example, the LP is:

**Obj0:** \( P=2, W=1, \alpha=0, \rho=0.5 \)

\[
\text{Minimize } \sum_{m \in M, g \in G^a} D_m^{a,g} * f_m^{a,g}
\]  

**Obj1:** \( P=1, W=1, \alpha=0, \rho=0 \)

\[
\text{Minimize } K
\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M )</td>
<td>The set of MDCs</td>
</tr>
<tr>
<td>( E )</td>
<td>The set of WAN links</td>
</tr>
<tr>
<td>( C_m )</td>
<td>Capacity of MDC ( m ) (in containers)</td>
</tr>
<tr>
<td>( B_l )</td>
<td>Capacity of link ( l ) (in Gbps)</td>
</tr>
<tr>
<td>( N_k^a )</td>
<td>Number of containers consumed by task ( k ) of app ( a ) per unit traffic</td>
</tr>
<tr>
<td>( G^a )</td>
<td>The set of user-groups, ( g ), of app ( a )</td>
</tr>
<tr>
<td>( D_m^{a,g} )</td>
<td>Distance from user-group ( g ) of app ( a ) to MDC ( m )</td>
</tr>
<tr>
<td>( u^a_g(t) )</td>
<td>Traffic at ( t ) for user-group ( g ) of app ( a )</td>
</tr>
<tr>
<td>( f_m^{a,g}(t) )</td>
<td>Traffic from user-group ( g ) of app ( a ) assigned to MDC ( m ) at ( t )</td>
</tr>
</tbody>
</table>

Table 3: ILP Notation

Subject to:

\[
\forall m \in M : \quad N^a_k * f_m^{a,g} \leq C_m \quad (3)
\]

\[
\forall m_1, m_2 \in M : \quad N^a_k * f_m^{a,g} - N^a_{m_1} * f_{m_2}^{a,g} \leq K \quad (4)
\]

\[
\forall g \in G^a : \quad \sum_{m \in M} f_m^{a,g} = u^a_g \quad (5)
\]

Thus, hierarchical optimization allows combining multiple objectives within and across applications in Patronus with varying levels of tolerance determined by the operator. This technique forms the core of both long-term and instantaneous schedulers.

### 4.2.2 Long-Term Scheduler

Long-term scheduling in WAND offers two benefits: (i) Analysis of historical traffic patterns provides the long-term scheduler with an approximate estimate of resource usage far ahead in time. Most user-facing applications in WAND have diurnal traffic patterns with predictable patterns across days. (ii) For deadline-bound jobs, it allows planning resource allocation ahead of time to meet the deadline. The WAND environment may host a large number of opportunistic and flexible jobs which are not time-critical. Analyzing the complete set of such jobs and their extent of flexibility during instantaneous resource optimization will increase the complexity of the scheduler considerably. In order to speed up instantaneous allocation and ensure fairness/cost-sharing across time, scheduling of flexible jobs are handled through long-term resource planning.

In this phase, we also mitigate the impact of limited statistical multiplexing at individual MDCs. While each individual MDC in WAND suffers from high variability due to its small size, a pool of MDCs in a region enjoys less variability when aggregated resources are considered (as discussed in § 2.2 Figure 1). In addition to identifying the most suitable MDC for a task, the long-term planner also identifies areas for backup scheduling. This loose allocation of tasks to a group of MDCs during the long-term planning phase also
helps avoid the need for a global search for an alternative allocation during instantaneous scheduling when the primary choice MDC is unavailable. Thus, long-term planning reduces the number of active low-priority tasks and their possible locations to be considered during a given instant while ensuring that the deadlines and other constraints are met in longer timescales.

The long-term scheduler can operate in two modes: (i) fast mode, (ii) replan mode. When a new job arrives at the long-term scheduler, first, the fast mode is initiated. The aggregate load at all DCs for all previously allocated jobs is used to determine the available capacity and an attempt is made to place the job. If its requirements (latency/bandwidth/deadline etc.) are met with allocation at preferred MDCs, the allocation is finalized. If the fast mode fails, a replan is invoked over applications at or below the priority class of the new job. In this phase, the resources are shared fairly across multiple jobs belonging to the same class. If the new job is not completely accommodated at the MDCs, parts of it may be scheduled on a regional or remote clouds in this phase. The long-term scheduler may also invoke replan of the schedule if long-term resource predictions change significantly relative to prior estimates.

The long-term scheduler is also responsible for providing inputs to instantaneous scheduling at each scheduling interval. It provides a limited set of feasible locations for each task based on traffic estimates. It also includes a set of optional tasks to ensure work-conservation during underestimation of traffic.

4.2.3 Instantaneous Scheduler

The instantaneous resource allocation module is responsible for per-instance scheduling of application components determined to be scheduled by the long-term scheduler. This includes (a) a subset of batch tasks, (b) loads on active streaming applications, and (c) optional tasks which are not critical in the current instance but may be scheduled if resources are available. These application components may belong to different priority classes and may have different objectives. For highest priority applications guaranteed to be accommodated by an MDC (such as cellular traffic which is guaranteed to have a utilization below a low threshold, say 50%), optimization is not invoked unless there is a failure in the system.

The instantaneous scheduler also relies on hierarchical optimization, but typically it has fewer objectives than the long-term planning mode. This is possible because deadlines, fairness, and the complete set of preferences are not handled by the instantaneous scheduler (long-term scheduler shares a critical subset). Even within the same class, the long-term planner will include relative priorities that determine which task is to be eliminated if sufficient capacity is not available. Such tasks are determined by long-term fairness estimates.

4.2.4 Other Optimization Features

We discuss two features in the context of WAND optimization: eviction tolerance of WAND applications and geodistributed fairness objectives employed by WAND provider.

Eviction Tolerance: Lower priority applications in WAND will be evicted when load of applications in higher priority classes spikes high enough that the MDC becomes overloaded. Therefore, being allocated resources in MDCs near users comes with some risk. We allow applications to control this tradeoff through the eviction tolerance parameter in the WAND API. During resource allocation, this high-level application requirement is incorporated in the optimization based on statistical analysis.

The goal of the eviction tolerance, $\theta$, is to represent the maximum eviction probability per unit-time tolerable by the application ($\theta \in [0, 1]$). However, evictions are not perfectly predictable. When considering scheduling an application in an MDC, Patronus attempts to predict eviction probability based on three factors: (i) mean load ($M$) of applications with priority $\geq c$, (ii) variability of their load ($V$), and (iii) resource requirements ($R$) of app $a$. Mean and variability are computed over a period of length equal to the length of the task being scheduled. i.e., if a task is 5 min long, the mean and variability of historical data from past 5 min is estimated. Variability is measured as Coefficient of Variation ($\gamma 2.2$).

The application with demand $R$ is evicted when the higher priority load exceeds $C - R$ where $C$ is the capacity of the MDC. Assuming that the variability of this load follows a normal distribution (with mean $M$ and coefficient of variation $V$), the probability of eviction at an instant is $P(L > C - R) = 1 - P(L < C - R) = 1 - \Phi\left(\frac{C - R - M}{MV}\right)$, where $\Phi$ is the cumulative probability of a standard normal distribution, $X \sim N(0, 1), \Phi(x) = P(X < x)$.

Note that all the terms in the probability expression are constants that can be determined apriori based on historical data. An MDC is considered a candidate for placement only if this estimated probability of eviction for the app is smaller than the application’s eviction tolerance ($\theta$). This is an approximate heuristic since the distribution of load may not be perfectly normal. However, the application can still adjust its eviction probability by increasing or decreasing $\theta$.

Fairness: The second optimization feature is the fairness objective for geo-distributed applications. The distributed nature of the applications and the underlying infrastructure calls for new notion of fairness. Defining fairness in WAND can be challenging for two main reasons: (i) WAND is a multi-resource environment with requirements on multiple resources, and (ii) in with most applications composed of geo-distributed components, it is difficult to define a domain of fairness. Does each application get a fair-share at each MDC? Can underallocation for an app in one MDC be compensated by overallocation at another location?

Patronus employs a simple notion of fairness. Since
DCs are the bottleneck in current cellular WANDs, we use fair-share allocation across DC resources for applications. However, instead of fair-sharing at each individual DC, the fairness domain is areas (collective group of neighboring MDCs). The choice of area as the domain for determining fairness is justified by the behavior of applications in WAND and the structure of the underlying topology. Streaming applications such as cellular VNFs are capable of load-balancing across adjacent DCs. When a single edge is overloaded, the application can obtain its fair-share through a neighbor.

4.3 Monitoring Module

Monitoring of applications serves two purposes: (i) enable quick reaction to changes in application behavior and (ii) provide input for prediction and long-term planning. The monitor interval can be explicitly set by an application through the WAND API or determined by the provider during runtime based on the extent of variability. We develop a simple tunable monitoring module in Patronus which can run alongside applications in WAND. We assume that this monitored information is made available centrally to the resource allocation module. The prediction module that relies on the output of monitoring may be co-located in the same MDC or centralized.

5 Implementation

We implement Patronus as a stand-alone control system with pluggable modules for each component. The current implementation of various modules is described below.

Resource allocation module: Both the instantaneous and long-term schedulers are Java applications with an integrated Gurobi [13] environment for solving the Linear Programs associated with resource allocation. Both schedulers take as input application requirements as JSON files containing task-level/user-group level requirements and dependencies.

The long-term scheduler holds a plan which contains estimated allocations for all applications across time. We store a plan for 1 day in 1 minute time-slots. The instantaneous prediction runs every minute. These parameters are tunable. When a new application arrives in the system, it is initially placed in this plan by the long-term scheduler. At each allocation interval, the long-term scheduler sends the list of currently active tasks to the instantaneous scheduler for actual placement. The schedulers convert requirements to linear objectives and constraints in a hierarchical linear program and solve the multi-objective LP using Gurobi. The output from the instantaneous scheduler is the set of allocated tasks, their location and the amount of resources allocated. Currently, the schedulers consider resources at the scale of VMs/containers within data centers.

Prediction module: The prediction module takes as input timeseries data on traffic volume/resource usage characteristics and predicts the resources required for the next instance. Currently, the prediction module can operate in two ways: (i) learning using a Neural Network (NN) or (ii) statistical estimates as mean, variance, weighted average etc. Depending on the characteristics of the application, one of these techniques is used. A 4-layer NN with 16, 8, 4, and 1 nodes at each of the layers respectively with full connection between the layers is used. Other cloud-based solutions [33, 2] can be easily integrated with this module.

Monitoring module: We implement the monitoring module as a lightweight Linux application which will run on the application VM to record its resource and traffic patterns. The monitoring interval is a tunable parameter. Note that the Patronus evaluation is based on simulation that does not use real-time monitoring.

6 Experiments

In this section, we evaluate scalability and efficiency of the Patronus system.
6.1 Methodology

The first step of evaluation is the development of a realistic test environment. We use real-world traffic datasets and publicly available infrastructure information to generate a realistic WAND topology.

6.1.1 Workloads

We integrate several real-world workloads to generate the topology and the traffic in WAND.

(a) Cellular dataset: The cellular dataset is obtained from a large cellular provider in China and consists of utilization information at 50 virtualized DCs over a period of 5 days. At each DC, it provides information on (a) the mean and the maximum CPU utilization per server, and (b) total data sent and received from the DC. The utilization values are monitored every 10s by the cellular provider. The dataset contains the mean and the maximum at 15 min intervals over this data. The utilization at two locations are plotted in Figure 2 (a). In Figure 3 (a), we plot the CDF of mean and maximum CPU utilization across all 50 locations on all days. We observe that the utilization is approximately between 10% and 30%. The distribution of normalized DC sizes (# servers) is plotted in Figure 4 (a). The number of servers range from < 10 to hundreds. The largest DC has more than 1000 servers.

(b) DNS dataset: This dataset contains DNS queries received by a single DNS provider at locations across the globe over a period of 24 hours (508GB of compressed data). Entries include the timestamp (at microsecond granularity) and the origin zipcode of each DNS request. From this global dataset, we filter requests originating in the continental US, which include 17,440 origin zipcodes with more than 20 billion queries in the 24-hour period. The number of queries are aggregated at minute-scale and load at two zipcodes are plotted in Figure 2 (b). The CDF distributions of minimum and mean with respect to the maximum load are shown in Figure 3 (b). We observe that the DNS queries have higher variability compared with the cellular dataset. The maximum load at a DC can be more than 10× the mean in certain cases.

(c) Video camera dataset: Video processing at the edge is a most prominent application that will benefit from WAND deployments. The number of security cameras in the US in 2016 is estimated to be 62 million [22]. We use the locations of red-light and speeding cameras in the US available in a public GPS forum. This dataset includes 4932 cameras across the continental US. While these cameras take snapshot pictures during traffic violations, we use this as a realistic proxy for video camera locations.

(d) Big data analytics: We use publicly available TPC-DS microbenchmarks [26] with synthetic location constraints to simulate batch analytics workloads in WAND. We use traces of 16 TPC-DS queries originally run on 20 machines.

6.1.2 Generating the Test Environment

We run the Patronus controller on a Dell PowerEdge R440 machine with 32 cores and 128GB RAM.

(a) Topology: We generate a realistic WAND topology based on publicly available information on cellular infrastructure. The locations of MDCs are determined using Central Office (CO) information of a large cellular provider in the US [1]. We obtain 1864 Central Office locations associated with the 17440 zip codes in the DNS dataset.

Each MDC is assigned a capacity which is proportional to the population associated with that zip code (2016 population [37]). The number of servers in the US and the real size distribution in China (Figure 4 (a)) follow similar distributions. The number of servers in the US simulation environment ranges from 2 to 2200. In line with regional offices of cellular providers, we also add 15 regional DCs (publicly available locations of colocation centers of the same cellular provider in the US) with capacity proportional to the sum of MDCs that are closest to it.

The core topology of the same cellular provider is obtained from the Intertubes dataset [10] and has 116 nodes and 151 links. Each DC is connected to its closest core node. Conversations with cellular operator reveal that currently cellular MDCs follow a hub-and-spoke-like model with MDCs connecting to the core. With 5G, MDCs are expected to have more interconnections among them. Currently, we do not consider such interconnections in our test environment. The bandwidth of the outgoing link of an MDC is set proportional to its capacity.

(b) Workload: The cellular data for the test environment is obtained by applying the percentage of utilization from a randomly chosen DC in the Chinese dataset to the DC. These mapping are done a priori to obtain the timeseries for a day. VPN/other high-priority applications of cellular provider is generated synthetically. We generate 100 such applications, each with a random number of sites between 3 and 15 with a randomly chosen load at each location. These are streaming applications where all sites are concurrently active.

The scaling factors for datasets are determined proportional to their loads in such a way that the load (as number of containers) is neither too low nor too high at most DCs. For the DNS dataset, the load in containers is proportional to the number of requests. For the camera dataset, the number of containers is proportional to the cameras assigned to a DC.

6.2 Results

We evaluate the ability of Patronus controller to utilize the resources efficiently and meet the application requirements.

Efficiency of Hierarchical optimization

VPN and other applications which belong to the highest priority class are slightly more flexible than the cellular traffic.
We assume that the cellular traffic need to be processed at the nearest MDC (when resources are available) while the VPN allocation may be at any MDC within 1ms from the origin of the traffic. Hence, to determine long-term placement for non-cellular high priority applications, we run a two-level hierarchical optimization at the peak period of cellular load.

The optimization has 2 objectives. The highest priority objective is shortest path allocation of cellular traffic. In Figure 5, we observe that shortest path allocation on cellular traffic load results in a maximum edge DC utilization of 50%. For the VPN traffic, we evaluate several techniques. Minimizing the maximum load of DCs with latency-based placement constraints in Patronus can accommodate all the non-cellular applications while maintaining the maximum utilization at 53.6%. On the other hand, placing the VPN load at the closest MDC to the origin or at a randomly chosen MDC within the prescribed radius (1ms) leads to high utilization in some DCs. Note that jointly optimizing the cellular and VPN loads may result in some cellular traffic being redirected to non-shortest DCs. Hence, multi-level optimization may be essential even within the same priority class.

**Eviction Tolerance/Latency Trade-off**

The trade-off between eviction tolerance (4.2.4) and latency at a single MDC in third priority class is given in Figure 6 (first priority is cellular and VPN applications, second priority is DNS workload). We consider the video analytics application load in third priority class at a single MDC. Eviction tolerance is computed based on the mean and standard deviation of total traffic in the higher priority classes for an interval of 15min. The figure shows mean latency over a period of 19 hours at each data point. When the eviction tolerance is the highest at 1.0, the latency is 0.15ms. As the eviction tolerance decreases, the latency of the application increases. At eviction tolerance value of 0.7, the latency increases 10× to 1.5ms. When the application is very conservative (θ < 0.1), the latency increases to approximately 3ms.

**Latency of applications**

We measure the latency perceived by the video analytics dataset in priority class 3 (Figure 7). We compare Patronus optimization with nearest-MDC placement. In both scenarios, when an MDC is not available, the task is placed on the nearest regional DC. We observe that the tail latency is more than 2× lower in Patronus. Note that the latency is computed based on geodesic distance here. With additional infrastructure overhead, the differences will be more pronounced since Patronus places tasks at the edge at a higher rate.

**Prediction Efficiency**

We evaluate the predictability of the two real-world datasets. In the sparse cellular dataset, we use statistical measures. We consider two scenarios: (a) past hour on the same day, or (b) Time of Day (ToD) in past n days. When the prediction is defined as the maximum observed load, ToD of 4 days is a better predictor. Over a day, prediction based on recent history can lead to under-prediction up to 6.5% and over- allocation up to 8% while ToD prediction limits the errors to 3.1% and 6% respectively. Hence, we use ToD demand of 4 days with a 5% over-allocation to meet the demand.
we observe that the coefficient of variation of penalty across cloud when edge resources are not available. In Figure 10, define the penalty as the amount of resources allocated on able, they can run in the cloud, but this is expensive. We the same class determined by the LP solver). We assume that forced (arbitrary sharing of resources across applications in scheme compared with a scheme where no fairness is en-

We compare the fairness achieved with Patronus fairness
scheme compared with a scheme where no fairness is en-
forced (arbitrary sharing of resources across applications in
the same class determined by the LP solver). We assume that
for batch applications, when the edge resources are un-
available, they can run in the cloud, but this is expensive. We
define the penalty as the amount of resources allocated on
cloud when edge resources are not available. In Figure 10,
we observe that the coefficient of variation of penalty across
applications (a heuristic for unfairness) is reduced with fair-
sharing in Patronus.

Performance monitoring
We measure the overhead associated with monitoring using a
lightweight Linux module which measures applications’ us-
age of CPU, memory, network, and disk. The CPU and disk
utilization of the monitoring module at various monitoring
intervals are given in Figure 11(a) and 11(b) respectively. The
overhead is negligible (< 0.5% in CPU and 9Kbps in IO) when monitoring interval is 1s.

7 Discussion
Patronus is a first step towards scalable WAND resource
management. Several challenges remain open.
Offering WAND services: Patronus is designed for a private
WAND environment. If a WAND provider chooses to offer resources to external applications, similar to the cloud, we
need a novel service characterization. MDCs have different
compute capabilities and latency profiles between them. The cost and performance using the same amount of resource vary widely depending on the size and location of the DCs and within the same DC across time due to limited statistical multiplexing.

More stringent application constraints: Patronus can respond to application changes at the scale of seconds. To handle emerging applications that require millisecond-level application response times and for improving resource utilization when fast VM/container resource scaling is feasible, Patronus may be extended with edge controllers that supplement the centralized controller.

Handling the data deluge: Monitoring and profiling of
thousands of geo-distributed applications generate signifi-
cant amount of data. Patronus assumed that this data is avail-
able to the centralized controller. In practice, we need effi-
cient techniques to collect and monitor such data. The tem-
poral and spatial dimension of generated data in WAND calls
for novel database techniques for efficient storage and query
response-times.

8 Conclusion
In this paper, we identify control challenges in an emerging
dynamic environment of interconnected Micro Data Centers
which we refer to as WAND (WAN As A Network of Data
Centers). We build Patronus, an automated control system,
for efficient resource management in WAND. We address the
scalability challenge by partitioning the scheduling problem
into two temporal levels: instantaneous scheduling handling
immediate allocation and long-term scheduling for meeting
critical application requirements across time. We show that
Patronus is scalable, resource-efficient with balanced load
across MDCs and capable of meeting stringent application
requirements.


