# Switchboard: Efficient Resource Management for Conferencing Services

Rahul Bothra, Rohan Gandhi, Ranjita Bhagwan, Venkata N. Padmanabhan,
Rui Liang, Steve Carlson, Vinayaka Kamath, Sreangsu Acharya,
Ken Sueda, Somesh Chaturmohta, Harsha Sharma

**Microsoft**

## ABSTRACT

Resource management is important for conferencing services (such as Microsoft Teams, Zoom) to ensure good user experience while keeping the costs low. Key to this is the efficient provisioning and assignment of media processing (MP) servers, which do the heavy lifting of mixing and redistributing the media streams from and to the call participants.

We introduce Switchboard – a controller for efficient resource management for conferencing services. Switchboard is *peak-aware*, recognizing that cost depends on the peak resource usage and that there is a temporal shift in peak demand across time zones. This allows a server in a region to serve calls at peak time, and double up as backup for other regions during non-peak times. Furthermore, it improves efficiency by performing joint network and compute provisioning and application aware provisioning. We evaluate Switchboard using 1+ year of records from Microsoft Teams. Switchboard achieves upto 51% lower provisioning cost while achieving similar or better latency over state-of-the-art baselines.

## CCS CONCEPTS

• **Networks** → **Application layer protocols**; **Cloud computing**;

## KEYWORDS

Application Layer Networking, Video Conferencing, Network Provisioning, Resource Optimization

**ACM Reference Format:**
Rahul Bothra, Rohan Gandhi, Ranjita Bhagwan, Venkata N. Padmanabhan, Rui Liang, Steve Carlson, Vinayaka Kamath, Sreangsu Acharya, Ken Sueda, Somesh Chaturmohta, Harsha Sharma . 2023. Switchboard: Efficient Resource Management for Conferencing Services. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3603269.3604879

## 1 INTRODUCTION

Conferencing services such as Microsoft Teams [9], Zoom [12] and Google Meet [4] have become critical business services, especially since the COVID-19 pandemic. As the size of these services has grown sharply [10, 14], service providers have grown concerned about their running costs, specifically the need to efficiently provision compute and network resources while ensuring excellent user experience. Such cost efficiency is particularly relevant because of the all-you-can-eat subscription-based nature of such conferencing services (as opposed to pay-as-you-go), which makes it attractive for the service provider to serve its users well while keeping its cost minimal.

One of the most resource-hungry operations in such conferencing services is *media processing and delivery*. All participants on a call send their audio, and/or video and/or screen-share data to a *Media Processing (MP) server* running in a datacenter (DC). The server combines, resizes and processes all streams and sends an aggregated feed back to the participants. Processing the media for each call is not only compute-intensive in view of the multimedia processing that is required, it is also network bandwidth-intensive as large amounts of multimedia content need to be delivered to and from the MP server.

To effectively support media processing and delivery at such large scale, the service has to perform two key functions. First, for each call, the service has to determine which datacenter (DC) should host the call. We call this *MP server allocation*. Second, to enable the allocation of an MP server in a suitable DC for each call that ensures a good latency for each *call leg* (i.e., the segment between the server and each participant), the service has to provision enough serving compute and network capacity at all DCs up front. Such provisioning of dedicated resources is essential given the real-time nature and the large size of the service, which makes on-demand resource rental unviable[1–3]. Additionally, the service has to provision backup compute and network capacity to survive failures, even drastic ones (e.g., an entire DC going down). We call all of this *MP capacity provisioning*.

The need for dedicated resource provisioning means that the service's peak usage at each location (e.g., compute in each DC, bandwidth on each WAN link) determines how much compute and network capacity is provisioned overall, which, in turn, determines the cost incurred. To reduce overall cost, it therefore helps to reduce peak capacity provisioned, which the service can achieve by consolidating resource usage, for example, "packing" more calls onto MP servers with unused capacity, or routing traffic via network links that have free capacity, i.e., are carrying less traffic than their provisioned capacity. As we will later see in §3, the compute and network requirements are intertwined and contingent on the "packing" scheme. The challenge lies in coming up with a cost-efficient scheme without compromising latency for participants.

ACM ISBN 979-8-4007-0236-5/23/09…$15.00
https://doi.org/10.1145/3603269.3604879

The real-time nature of conferencing means that the user workload cannot be reshaped over time. The lever that is available is the choice of MP server placement, which in-turn affects both the compute and network capacity required, as we will see in 3. Keeping this in mind, we make three novel observations:

(1) since work hours vary naturally across time zones, conferencing usage peaks at different times across different regions. We leverage this difference in peak times to lower the peak usage at a particular DC by serving some calls from a different DC with acceptable latency and spare capacity. Furthermore, when the primary serving capacity at a certain location is unused during non-peak hours, we repurpose it as backup capacity for calls served from other primary locations, (2) we observe that compute and network capacity should be provisioned jointly to further decrease the overall cost of provisioning. For instance, if compute is expensive at a certain DC but network capacity to the DC is relatively inexpensive, we may still choose to provision the expensive compute to lower the overall cost, (3) unlike previous work [34], which uses *system-level* resource usage history (compute and network usage logs) to provision resources, we observe that using *application-level* logs, with information such as the geographic distribution of call participants, and the type of media they share, allows us to make more optimal provisioning decisions.

We used these observations to build SWITCHBOARD, a controller for conferencing services that performs cost-efficient capacity provisioning and server allocation while ensuring low latency for participants. To the best of our knowledge, SWITCHBOARD is the first system to combine peak-aware, joint compute-network, and application-specific provisioning together, to make provisioning more efficient. SWITCHBOARD implements MP capacity provisioning using an LP (Linear Program) optimization framework that runs periodically (every few months). SWITCHBOARD partitions server allocation into a compute-intensive LP optimization that runs periodically in the background (once every day), and a real-time lightweight MP server selection implementation that runs in the critical-path of the service (each time a call starts).

We evaluate SWITCHBOARD using data from running Microsoft Teams over Azure compute and network services, and compare it with state-of-the-art baselines that use round robin and locality-first heuristics. We show that SWITCHBOARD reduces the costs by up to 51% by reducing the number of cores and WAN bandwidth required while improving or closely matching the user perceived latency. Additionally, we also show that SWITCHBOARD is scalable; it can handle 1.4× the load currently experienced by Microsoft Teams.

This work does not raise any ethical issues.

## 2 OVERVIEW

### 2.1 Problem

Every conference call gets hosted on a *media processing (MP) server* that runs in one of many *datacenters (DCs)* that host the service. In the case of Microsoft Teams, the service divides the world into *regions* such as Asia-Pacific, Europe, etc., and each region has one or more Datacenter. Therefore, when a call is to be hosted on an MP server in the Asia-Pacific region (say because that is where most or all of the participants are located), the MP server should be located in one of the Asia-Pacific DCs (Hong Kong, India, Japan and Singapore).

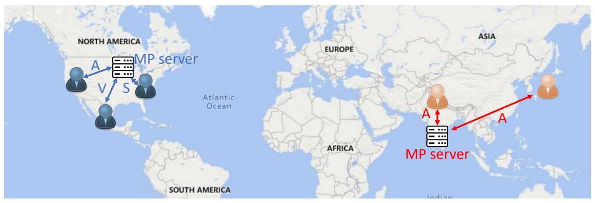


**Figure 1: MP server allocation for two calls marked in blue and red. Participants could share audio (A), video (V), and/or screen-share (S) streams.**

The problem of mapping calls to MP servers across these regions and DCs can be broadly split into two: *MP server allocation* and *MP capacity provisioning*.

**MP server allocation:** When the first participant of a call joins in, the service selects a suitable MP server in one of the DCs within the region from where the call originates. This is called MP server allocation. It then connects all participants who join the same call subsequently to this MP server. Participants send their respective audio, video, and/or screen-share streams to the MP server, which then processes the streams and sends them back to all participants.

*In this paper, we focus on selecting the DC location for the MP server of a call (more details in §2.2).* Fig. 1 shows two examples. When all participants in a call are from the USA, a natural choice is to select an MP server in a North American DC so that latencies are low, and the call uses less wide-area network (WAN) bandwidth. When a call has participants across geographically disparate locations such as the India and Japan, it may help to select an MP server in DC such as India or Japan to ensure a lower average latency experienced by participants. We should keep in mind that the service needs to select MP servers based on network latency, and availability of compute capacity at the DCs and network capacity of the inter-DC WAN links.

**MP capacity provisioning:** The MP server allocation problem is preceded by MP capacity provisioning. MP capacity provisioning determines how much compute to provision at each DC and how much WAN capacity to provision between DCs, to support the anticipated call workload, even in the face of failures that might render some of the compute and/or network capacity unavailable. The capacity provisioned, both compute and network, is the sum of both *serving capacity*, i.e., capacity required to serve calls in the absence of failures, and the *backup capacity*, which is the capacity required to fail-over calls in case there are compute or network failures. MP capacity provisioning is an *offline* process as it runs periodically in the background, say every few months. Thus, based on the output of MP capacity provisioning using a forecast of the service usage, the cloud provider may need to change the amount of compute and network provisioned at each DC and network path from time to time.

Both capacity provisioning and server allocation need to be cognizant of the following requirements:

(1) **Good user experience:** Conferencing experience is very sensitive to network latency, jitter, and packet loss. Of these, latency is fundamental, and is our focus here, since it is a significant function of distance and cannot be mitigated by provisioning sufficient capacity to avoid congestion. To reduce latency, the service should select MP servers that are "close" to all call participants. In

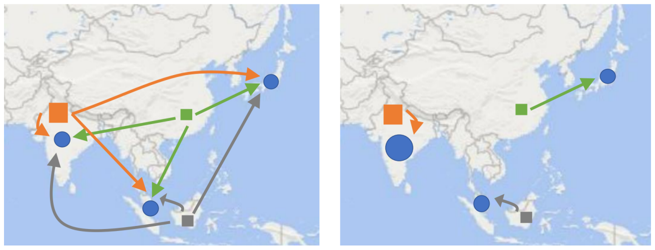


**Figure 2: Depiction of Round-Robin (left) and Locality-First (right) baseline strategies. Blue circles are DCs and squares are source countries of call participants. Bigger circles/squares denote DCs with more capacity and countries with more call participants respectively.**

our work, we seek to keep the *Average Call Latency* (ACL), defined as the average one-way latency of all call legs in a call, under 120ms and where this is infeasible, say because the participants are widely dispersed, we seek to minimize the ACL of the call.

(2) **Failure resilience:** The service infrastructure should be resilient to failures that result in the loss of compute and/or network capacity. Therefore, redundant compute and network capacity should be provisioned so that, for instance, the service can fail over to alternate DCs in the event that an entire DC becomes unavailable. Of course, the failure modes need to be circumscribed, to keep the provisioning requirement bounded. For our experiments, we assume that an entire DC or any one WAN link can fail at a time. However, we note that our framework can easily incorporate more sophisticated failure scenarios.

Our goal is to meet the above latency and failure resilience requirements while minimizing the overall cost incurred on the compute and network resources. As discussed in §1, given the growing and large-scale nature of conferencing services, it calls for *dedicated resources* to be provisioned. So, the cost is a function of the *peak* resource usage of all DCs and WAN links.

## 2.2 Scope

In this paper, we focus on a first-party conferencing service, i.e. Microsoft Teams. First-party implies that the service has visibility into the compute and network resources of the cloud provider where the service is hosted, Azure in our case. We focus on provisioning and allocation of compute resources per DC, and network bandwidth for inter-DC links. The problem of intra-DC MP selection, i.e. selecting a specific server within a chosen DC, is well-studied [20, 33] and beyond the scope of this work.

We also do not consider the cost of Internet connectivity between the edge of Azure's network and the call participants, as it is beyond the control, as well as inescapable, for the conferencing service.

## 3 BASELINE STRATEGIES

We present two well-established [39] and state-of-the-art heuristic baselines for server allocation and capacity provisioning. For simplicity of exposition, we assume that all call participants belong to the same region (such as Asia-Pacific).

## 3.1 Round-Robin

**Server allocation:** Amongst the DCs available within the same region as participants, we then use round-robin (RR) across calls to select the datacenter within the region. Fig.2(a) depicts this policy. While, in general, a weighted RR scheme could be used, we focus here on RR with equal weights since it helps equalize load across the sites, thereby minimizing the need for backup compute capacity, as discussed next.

**Capacity provisioning:** With RR, the total compute provisioned across all DCs is the compute required to serve the *total peak*, or the maximum number of calls arising at any time across the region, which is the minimum total compute required to serve all calls.

Additionally, RR's load equalization minimizes backup compute capacity needed in the event of a single DC failure. For instance, if there are four DCs in a region over which RR spreads all calls, each DC would have 25% of the total peak serving capacity of compute. To handle a single DC failure, 8.33% (25/3) of total peak serving capacity needs to be provisioned at each DC as backup to provide enough capacity for single DC failure.

While RR minimizes the amount of compute capacity required, both serving and backup, it is inefficient in its use of WAN bandwidth, because it sprays all calls around the available DCs. This results in some calls getting hosted in far-off DCs – inflating their WAN capacity requirement as well as the call latency.

## 3.2 Locality-First

**Server allocation:** At the other end of the spectrum, we consider a policy to minimize the call latency [21, 23, 24, 39]. Locality-first (LF) assigns the MP server to the DC location which will have the lowest *Average Call Latency* while serving the participants. It also ends up reducing the WAN capacity requirement as traffic needs to traverse shorter paths. Fig.2(b) captures the LF policy.

**Capacity provisioning:** LF helps reduce the amount of network capacity provisioned. However, both compute requirements – serving capacity and backup capacity – increase significantly. Each DC's compute serving capacity needs to accommodate the *local peak* demand from the sub-region for which the DC in question is the nearest one. Hence, the total compute serving capacity needed is the sum of peaks of demands coming to individual DCs, with the peaks likely being shifted relative to each other because of the spread of time zones. Therefore, the total serving capacity needed would, in general, be greater than RR's, as *the sum of time-shifted local peaks is greater than the global peak.*

The backup compute capacity needed also increases when compared to RR because of the greater skew in the serving capacity across the sites. Consider the example where four DCs are used to host all calls in the Asia-Pacific region, but where 75% of all calls involve participants only from India. Then the India DC contains 75% of all serving compute capacity. To handle failure of this DC, the other three DCs need to provision a cumulative backup capacity that is 75% of total serving capacity, as opposed to only 33% in the case of RR.

We formulate the backup compute capacity calculation as follows, which minimizes the total backup capacity needed while handling one DC failure at a time,

$$Minimize \sum_{x}^{DC} Backup_x \tag{1}$$

$$Serving_x <= \sum_{y,y!=x}^{DC} Backup_y, \quad \forall x \in MP \tag{2}$$

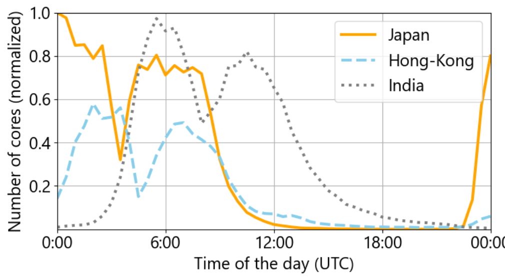


**Figure 3: Demand (number of cores) required from 3 different countries normalized to max. peak observed. It shows the time-varying peaks – peaks in Japan, Hong-Kong and India form at 0:00, 2:00 and 5:30 hours.**

where $Serving_x$ is the serving capacity of DC x, and $Backup_x$ is the backup capacity required in DC x, and the objective is to minimize the total backup capacities across all DCs.

## 4 KEY IDEAS

### 4.1 Peak-aware provisioning

We observe that the number of participants from different time zones peak at different times, and therefore the compute and network capacity required to support these participants peak at different times. Fig.3 demonstrates this by plotting the amount of compute required to serve callers from Hong Kong, India and Japan in a one-day period. SWITCHBOARD's optimization formulation leverages this in the following way. First, the compute usage due to call participants from Japan peaks at around 00:00 hours UTC, when usage from India and Hong Kong is very low. Rather than serve all calls during this peak from the Japan DC, which would drive up the compute capacity to be provisioned in Japan, SWITCHBOARD allocates servers for some of the calls in the India and Hong Kong DCs, but only if the latency between Japan and these two DCs are below an acceptable threshold. On the other hand, at non-peak hours for Japan such as at 09:00 hours UTC, SWITCHBOARD accommodates all calls from Japan in the Japan DC itself, thereby minimizing latency at that time.

### 4.2 Joint serving + backup provisioning

SWITCHBOARD takes advantage of the difference in the peak usage across time zones to multiplex serving capacity and backup capacity. Consider the trends shown in Fig.3 again. The compute requirement for calls from India peaks at around 05:30 hours UTC, and hence for good latency, SWITCHBOARD provisions a significant amount of capacity at the India DC. However, at 00:00 hours UTC, calls from India need very little of this already-provisioned capacity. SWITCHBOARD earmarks some of this capacity as *backup* capacity for Japan, so that it is available to pick up the slack if the Japan DC were to fail at 00:00 hours. In other words, SWITCHBOARD *repurposes* some of the capacity used to serve India calls at 05:30 hours UTC as backup capacity for Japan calls at 00:00 hours UTC.

Fig.4's example contrasts this approach with a baseline that uses locality-first (LF) to first provision serving capacity and then provisions additional backup capacity over and above this. Say the compute cores required for Japan, Hong Kong, and India calls are as shown in Fig.4(a), where the peak serving capacity required is 100 in Japan, 110 in Hong Kong, and 110 in India. Then the LF policy provisions 100 cores in Japan, 110 in Hong Kong, and 110 in India DCs, since these are the maximum requirements across time.

Fig.4(b) shows the total serving and backup capacity needed in each DC per time-slot for the LF baseline calculated using the LP in §3.2. For instance, to support a failure of the Japan DC, since its serving capacity is 100 cores, the India DC and the Hong Kong DC each need to provision backup capacity of an additional 50 cores.

Fig.4(c) shows how SWITCHBOARD reduces the total compute capacity across all DCs. Consider a Japan DC failure at time T1. Since India uses only 20 of its cores for serving calls at time T1, SWITCHBOARD repurposes 60 of its remaining 90 serving cores as backup capacity for Japan at time T1. It repurposes 40 of Hong Kong's 50 unused serving cores as backup capacity at time T1. Thus, a total of 100 serving cores across India and Hong Kong DCs get repurposed as backup capacity to cater to the possibility of Japan's 100 serving cores failing at time T1. By doing so, SWITCHBOARD reduces the compute capacity required in Japan from 160 to 100, in Hong Kong from 160 to 110, and in India from 160 to 110.

The same idea applies to network provisioning as well. Consider the scenario described in Fig. 5. To accommodate the Japan DC failing at time T1, SWITCHBOARD provisions both backup compute capacity at the India DC and backup network capacity from participants A and B to the India DC. Consequently, under a WAN link failure (say the link connecting participant A to the Japan DC fails), we do not need to plan backup network capacity since we can instead host such calls in India, in view of the backup compute available in India, and the backup network capacity provisioned for the link connecting participant A to the India DC. This results in a reduced overall WAN backup capacity required compared to network redundancy for links on both paths, i.e., those connecting participant A to India and to Japan.

### 4.3 Joint compute + network provisioning

SWITCHBOARD reasons about both compute and network jointly in its optimization framework. To illustrate the benefits with a toy example, SWITCHBOARD can provision compute and network for calls from Indonesia either at the Japan DC or the Singapore DC since either would satisfy the latency constraint. Despite compute costs in Singapore being higher than that in Japan, SWITCHBOARD provisions capacity for Indonesia in Singapore since the network links in the path between Indonesia and Singapore are significantly less expensive than the links between Indonesia and Japan.

### 4.4 Application-specific provisioning

SWITCHBOARD's capacity provisioning uses forecasts based on historical data from Microsoft Teams's call logs, their placement, and the latency recorded thereof. The primary input to MP capacity provisioning is the number of calls of different "types" or what we call *call configurations* (described in §5.1) that we forecast will need to be supported in the future. This is unlike state-of-the-art provisioning techniques [34] which forecast compute and network requirements based on system logs that record compute and network usage. For instance, let's say calls with all their users in India are increasing. On one hand, if SWITCHBOARD were making provisioning decisions simply based on compute and network-specific resource usage, it would end up adding more capacity in India, and

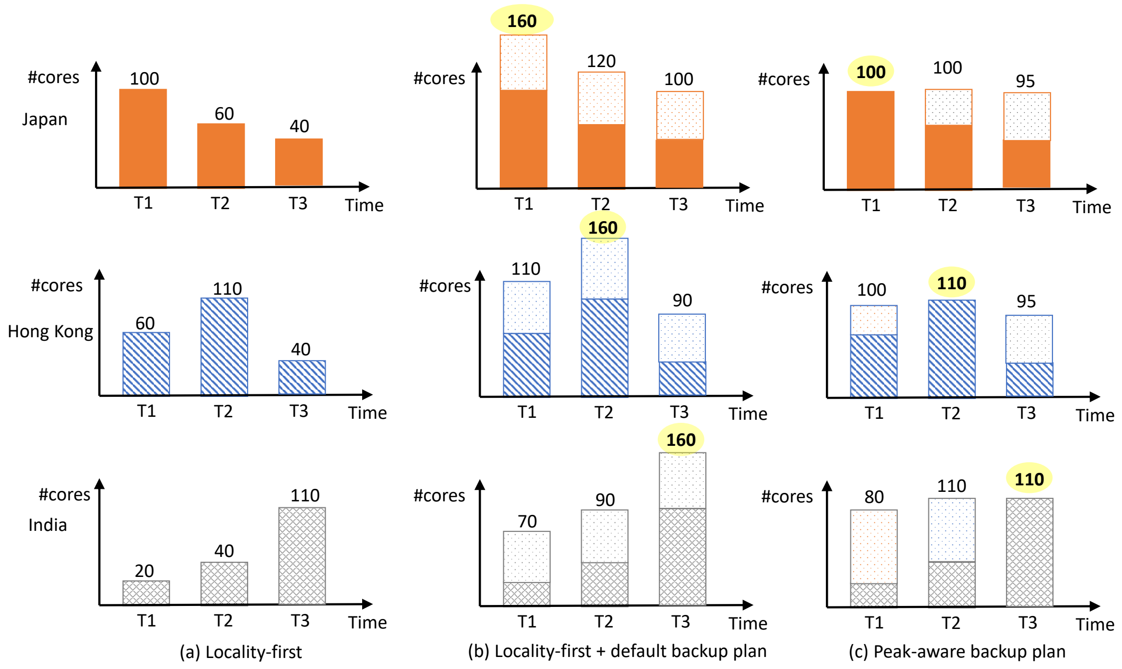


**Figure 4: Peak-aware capacity planning. (a) shows the demand from 3 countries in terms of number of cores shown in different colors. Using locality-first policy, demand from each country is handled by DC in the same country. Note that the DCs are to be provisioned for the peak, (b) shows the backup capacity (dotted boxes) needed using default backup plan using an LP described in §3.2. Note that, the backup capacity calculated by LP is added across all timeslots, (c) shows the backup capacity needed using peak-aware backup plan by re-purposing the serving cores for backup when demand for such cores is not at peak. Note that the change in *peak* capacity is highlighted.**

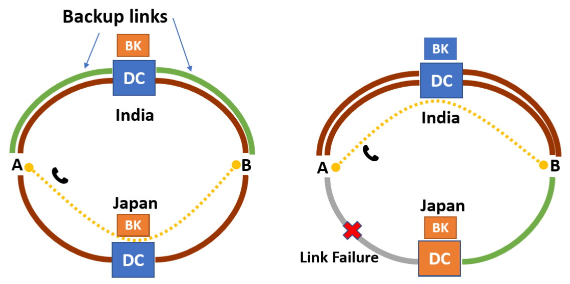
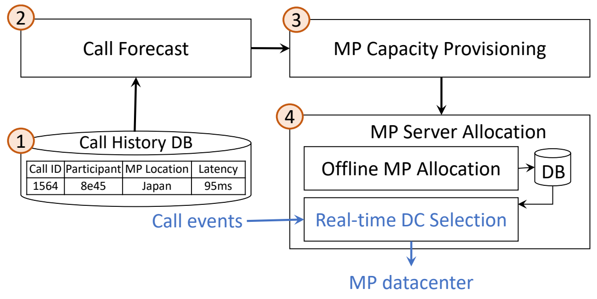


**Figure 5: Joint compute and network provisioning can result in less network redundancy requirement.**



| Call ID | Participant | MP Location | Latency |
|---|---|---|---|
| 1564 | 8e45 | Japan | 95ms |

**Figure 6: Building blocks in SWITCHBOARD. The blue arrows show the real-time operations.**

potentially increasing the peak. However, application-specific provisioning, we could absorb this surge in demand by shifting calls to another DC, and thereby not increase the peak (and therefore, cost).

## 5 DESIGN

SWITCHBOARD's design, as shown in Fig.6, has four modules,

(1) **Call Records Database**: Recall that the connection between each call participant in a call and its MP server is called a *call leg*. For each call leg, Microsoft Teams records and stores some metadata, which includes the MP server's location, start time, and the latency experienced by the user. SWITCHBOARD uses these call records to forecast demands as well as to predict latencies of call legs. The data is anonymized to adhere to global privacy standards.

(2) **Call Forecasting**: Since provisioning capacity would, in general, require a significant lead time (months or more), there is a need for workload forecasting. Ideally, SWITCHBOARD should forecast the resource requirements for each scheduled call and provision accordingly. However this approach would not scale well, given the sheer number of calls the service handles. Instead, SWITCHBOARD groups calls by *call configuration, i.e.,* calls that have similar resource requirements, and generates forecasts per call configuration. §5.1 explains call configurations further. For each call configuration, SWITCHBOARD uses Holt-Winters exponential smoothing[5] and forecasts the number of calls months into the future. In our work, we perform forecasting 3 months into the future

and at the granularity of 30 minute time buckets. §5.2 provides details of the forecast module.

(3) **MP capacity provisioning:** Using the forecasts, this module computes the amount of compute and network capacity to be provisioned at each DC and WAN link. It minimizes the total cost of provisioning while constraining the average call latency (ACL) within a threshold. In case a call has widely spread participants, and no DCs can host the call while satisfying the latency threshold, then we host such a call in the DC whereby the ACL would be minimum. We run this module once every few months. §5.3 describes this further.

(4) **MP server allocation module:** Given the compute and network capacities from (3) above, this module assigns the DC locations to each incoming call, with the objective of minimizing the mean ACL. Switchboard splits this into a compute-intensive offline optimizer that runs at the start of the day, and a lightweight, real-time server selector. §5.3 and §5.4 provide details.

## 5.1 Call configuration

The total number of calls managed by the service could be very large. As such, working with individual calls as the unit of optimization would place a heavy strain on Switchboard's LP framework. To avoid this problem, we define the notion of a call configuration (call config, for short), which captures the size, spread, and media type of a call. The call config effectively captures the resource needs of the call, with the significant computational advantage that there are order of magnitude fewer call configs than there are calls.

A call config comprises: (1) the location (at the granularity of countries) of the participants, (2) participant count from each location, (3) media-type, which could be audio, video, or screen-share. By default, all calls have an audio. Both video and screen-share are more resource-intensive than audio. If no participant shares their screen but at least one participant turns their video on, then the media-type is video. If at least one person has shared their screen in the call, we say that the media-type of the call is "screen-share". This classification is based on the relative resource requirements of different media types. An example call config is *((India-2,Japan-1), audio)* which groups together all audio-only calls that have 2 participants from India and 1 from Japan.

All calls with the same call config are fungible in the sense that they have largely the same resource requirement. Therefore, Switchboard provisions and allocates resources at the granularity of the call config, since call configs are significantly fewer in number than calls, and hence more scalable. In our analyses, we found that the number of call configs are almost 30× fewer than the number of calls in a given time window.

## 5.2 Forecasting call count

For each call config, Switchboard forecasts the number of calls to expect for the call config at the granularity of 30 minute time-windows. From the call records database, Switchboard groups calls happening every 30-minute by their call config, which provides a timeseries as shown in Fig.7(a). It then applies Holt-Winters exponential smoothing[5] for timeseries forecasting.

We forecast the number of calls for individual call configs separately because the trend (i.e., growth rate in terms of number of calls) across call configs could be quite different. Fig.7(b) shows the growth rate for 15 call configs over a 4-month duration. Since the actual growth rate is business-sensitive information, we normalize the growth rate for each call config using the maximum growth rate observed across the chosen 15 call configs. Instead of using overall growth across total number of calls (aggregated across all call configs), such fine grained forecasting per call config provides greater accuracy for the purposes of provisioning (§5.3).

We found 10+ million unique call configs in Microsoft Teams's call records. However, we also observed that a very small fraction of the most populous call configs can account for a large fraction of the calls. For example, the top 0.1% and 1% call configs account for 86% and 93%, respectively, of all calls, as shown in Fig.7(c). Based on this observation, we forecast call counts for the top 1% call configs, which allows us to be scalable, while covering over 90% of the calls.

To account for the call configs not considered while forecasting, and the newer call configs which might come up in the future, we inflate the projected amount of resources to be provisioned with an adequate cushion, which is estimated by comparing our forecast-based projections with the ground truth in a validation dataset.

## 5.3 Capacity provisioning

Given the forecast for every time-slot for a certain time duration, Switchboard uses a linear program formulation to determine compute and network capacities needed at various DCs and WAN links to support the calls within that duration. The objective is to minimize cost while keeping average call latency under a threshold despite compute and network failures. We first define cost and the average call latency and then describe the failure model used in the formulation.

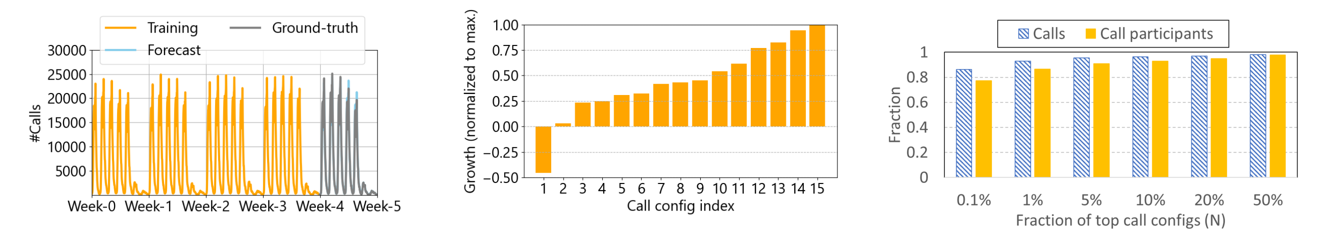
**Cost:** The total cost incurred by Microsoft Teams is the sum of the cost of provisioning, both serving and backup, compute cores across Azure DCs and network capacity on the various links in Azure WAN topology. In the context of this work, we limit ourselves to Azure WAN links and do not include costs of egressing to the Internet till the last-mile.

**Average Call Latency (ACL):** The average call latency for a call config is the average of the latencies of all call legs that a call of this call config would have. We constrain this one-way latency to 120 ms in our experiments, which is a reasonable bound based on our experience of running the service, and is consistent with the industry norm of a 250 ms target for an acceptable round-trip latency[6].

**Failure model:** Switchboard provisions capacity assuming that at any time, at most one entire DC or one WAN link can fail or get disconnected. Note that the failure of an entire DC would also render all network capacity connecting that DC unusable. Switchboard provisions sufficient backup compute and network capacity to enable other DCs to host calls that would normally have been hosted at the failed DC, or have redundant WAN capacity to be able to support the desired network traffic even when a WAN link fails.

We now explain the LP formulation. Table 2 states the definitions of all variables used for the reader's reference. The following equation captures the objective.

$$\text{Minimize} \sum_{l}^{L} WAN\_Cost(l) * \mathbf{NP_l} + \sum_{x}^{DC} DC\_Cost(x) * \mathbf{CP_x} \quad (3)$$



(a) Forecasting number of calls (ground-truth and forecast lines overlap for most points)

(b) Growth in terms of number of calls for 15 randomly selected call configs.

(c) Fraction of calls and call participants covered by the top-N call configs

**Figure 7: Forecasting number of calls for individual call configs.**

| | CL | NL | (NL / CL) |
|---|---|---|---|
| **Audio** | 1x | 1x | 1x |
| **Screen-share** | 1-2x | 10-20x | 10-15x |
| **Video** | 2-4x | 30-40x | 15-20x |

**Table 1: Relative compute load (CL) and network loads (NL), and the ratio of network to compute load for different media types[1].**

where $WAN\_Cost(l)$ is the per-unit cost of WAN link $l$, $DC\_Cost(x)$ is the per-unit cost of an MP server in DC $x$, $\mathbf{NP_l}$ is the peak usage of link $l$, and $\mathbf{CP_x}$ is the peak compute usage in DC $x$.

The following equations capture the constraints.

**Latency constraint:** We can allocate MP servers to call configs only in DCs where the average call latency is lower than the threshold.

$$ACL(x,c) > LAT_{th} \implies \mathbf{S_{tcx}} == 0, \forall t \in T \tag{4}$$

where $ACL(x,c)$ is the average call latency for hosting a call config $c$ in DC $x$, $LAT_{th}$ is the maximum average call latency allowed, and $S_{tcx}$ is the fraction of calls with call config $c$ in time slot $t$ and hosted in DC x.

**Serving capacity constraints:** For each resource, the serving capacity should be greater than or equal to the demand for that resource at all times.

$$\mathbf{CP_x} >= \sum_{c}^{C} \mathbf{S_{tcx}} * CL_{MT(c)} * |P(c)| \quad, \forall t \in T \tag{5}$$

$$\mathbf{NP_l} >= \sum_{c}^{C} \sum_{p}^{P(c)} \sum_{x}^{DC} \mathbf{S_{tcx}} * NL_{MT(c)} * InPath(l,x,p) \quad, \forall t \in T \tag{6}$$

where $CL_{MT(c)}$ is the compute load generated by a single participant in a call with a call config c (media type MT(c)), $|P(c)|$ is the number of participants in a call with call config c, $NL_{MT(c)}$ is the network load generated by a single participant in a call with call config c (media type MT(c)), $InPath(l,x,p)$ is 1 if link l is in the WAN path from DC x to location p.

**Backup capacity constraints:** Failure scenarios are expressed as $F_0$, $F_{DC1}$, ... $F_{DCn}$, $F_{L1}$, ... $F_{Lm}$ where $F_0$ has no failures and $F_{DCx}$ captures the failure of DC $x$ for $x = 1,\ldots,n$, and $F_{Ly}$ captures the failure of WAN link $y$ for $y = 1,\ldots,m$ We run the LP with all the above constraints for every failure scenario. When running for scenario $F_{DCx}$, the LP sets DC $x$'s peak compute usage to 0, i.e., $\mathbf{CP_{xf}}$ == 0 and computes the required backup capacity (both compute and network) for the scenario. Similarly, when running for scenario $F_{Ly}$, the LP sets link $y$'s peak usage to 0, *i.e.,* $\mathbf{NP_{lf}}$ == 0. Finally, it computes the compute capacity for every DC and network capacity for every link as the maximum required capacity across all failure scenarios as shown below:

$$\mathbf{NP_l} >= \mathbf{NP_{lf}}, \forall f \in F \tag{7}$$

$$\mathbf{CP_x} >= \mathbf{CP_{xf}}, \forall f \in F \tag{8}$$

**Completeness constraint:** All calls across all call configs should be allocated to MPs in different DCs in some share.

$$D_{tc} == \sum_{x}^{DC} \mathbf{S_{tcx}} \quad \forall t \in T, c \in C \tag{9}$$

where $D_{tc}$ is the number of meetings expected to occur in time t, and call config c.

**Note:** If there is no MP where a call config will not exceed the latency threshold, then we place all calls of the call config on the MP with the lowest average call latency.

**Allocation plan:** Switchboard divides the MP allocation into an offline, compute-heavy stage that uses an LP formulation and a real-time fast server selector. Switchboard runs the offline stage of MP allocation once every day. This stage determines, for every time-slot in the subsequent day, and for every call config, what fraction of calls in the call config should be placed on each DC so as to minimize mean ACL across all calls. To do this, Switchboard adds the following secondary objective to the LP above.

$$\text{Minimize} \sum_{t}^{T} \sum_{c}^{C} \sum_{x}^{DC} \mathbf{S_{tcx}} * ACL(c,x) \tag{10}$$

This objective generates an *allocation plan* for all call configs for all time-slots that minimizes the latency. For example, if there are going to be 100 calls for a call config ((JP-4, ID-2), video) in a given time-slot, the allocation plan may place 80 of them in an MP in Japan, 10 of them in Singapore, and 10 in India. Thus, it attempts to allocate more MP servers in the DC which would offer the lowest ACL. This is especially true in the case of non-peak hours when the resources are not constrained.

[1]Exact values are not provided because of Microsoft Teams's business sensitivity.

| Notation | Definition | Notation | Definition |
|---|---|---|---|
| $T$ | Set of time slots | $DC$ | Set of data centers |
| $C$ | Set of call configs | $M$ | Set of media types |
| $NL_m$ | Per-user network load by media type m | $CL_m$ | Per-user compute load by media type m |
| $D_{tc}$ | Total calls in time slot t, and call config c | $\mathbf{S_{tcx}}$ | Share of $D_{tc}$ hosted in DC x |
| $P(c)$ | List of locations of participants in call config c | $LAT_{th}$ | Latency threshold |
| $Lat(x,u)$ | Latency b/w DC x and location u | $ACL(x,c)$ | $\sum_p^{P_c} Lat(x,p)/\lvert P_c \rvert$ |
| $WAN\_Cost(l)$ | Per-unit bandwidth cost of link l | $DC\_Cost(x)$ | Per-unit compute cost for DC x |
| $Path(x,u)$ | List of links b/w DC x and location u | $InPath(l,x,u)$ | 1, if link l ∈ $Path(x,u)$, else 0 |
| $\mathbf{CP_x}$ | Peak compute usage of DC x | $\mathbf{NP_l}$ | Peak bandwidth usage of link l |
| $MT(c)$ | Media type of call config c | | |

**Table 2: Notations used in the LP (variables are marked in bold).**

## 5.4 Real-time MP assignment

The key goal of the real-time MP selector is to assign the MP datacenter to a new call when the first participant of the call joins. As mentioned in §5.3, it uses the pre-computed allocation to assign a DC to the call. Recall that this precomputed allocation is at the granularity of the call configs, *i.e.,* for each call config, how many calls should be hosted in each DC. The challenge, however, is that when the first participant joins a call, *the service does not know the entire call config, i.e.,* the number of participants, their locations, and call media type is not known. This makes it challenging to assign the DC based on the pre-computed allocation plan, which is also the basis on which capacity was provisioned.

There are multiple decisions to make. (a) First, we need to determine in which DC to host a new call even when we do not know its call config. Second, we may need to make adjustments once we do know the call config. Such adjustment could entail either (b) merely accounting for the resource usage of the call correctly (easy) or (c) migrating the call to a new DC because the initial location chosen was incorrect, *i.e.,* not in accordance with the precomputed plan (more difficult).

**(a) Assigning new call to a DC:** It turns out that the location (country) of the first joiner in a call is a pretty good predictor of where the majority of participants will join from. For instance, in our analysis, 95.2% calls have majority participants are from the same country as the first joiner. Furthermore, the precomputed allocation plan typically calls for placing a call where the majority of participants are, since it is beneficial from the viewpoint of minimizing latency.

Therefore, we use a simple heuristic: we first assign a call to the DC *closest* to the first joiner on the call. This often enables us to assign the new call to an MP in the correct DC (i.e., where the MP would have been placed if we had prescience and knew the call config), obviating the need for the call to be migrated (case (c) above).

**(b) Tallying up resource usage once call config is known:** As noted in previous section, the precomputed allocation plan specifies where to host calls of each call config with a view to minimizing latency while remaining confined to the resources already provisioned across the DCs. For instance, when the "slots" set aside for calls of a certain configuration have been exhausted at a DC, we would know to host further calls with the same configuration elsewhere. Therefore, as new calls arrive and old calls end at runtime, it is important that the resource usage be tallied up accurately against the plan that was precomputed, so that we can proceed in keeping with the plan and derive the associated benefits such as a latency-optimized allocation.

Between when a new call begins and its configuration is known ($A$ minutes into the call, as discussed in §6.4), we leave the call unassigned with respect to the precomputed plan. Of course, in the unlikely event of any resource exhaustion during this transient period, we look to migrate the call, as discussed in (c) below. Otherwise, once the call config is known, we simply debit 1 from the number of slots set aside for the configuration in question at the chosen DC.

Lastly, when a call has an unanticipated call config for which we have not allocated any slots, we map such call config to its *closest* DC. Such occurrences are rare, and we did not find any capacity crunch due to such calls.

**(c) Migrating to an MP server in a new DC:** Once the call config is known $A$ minutes into the call, we might realize that the initial choice of MP location was not correct, i.e., not in keeping with the precomputed allocation plan. For instance, consider an example where the first participant to join call might be from Japan (and so the call gets assigned to an MP server in Japan), but the configuration eventually turns out to be ((JP-3,ID-5), video), *i.e.,* with a majority of participants being in Indonesia. Per the precomputed plan, if all such calls were to be hosted say, in the Singapore DC, then this call will have to be migrated.

## 6 EVALUATION

In this section, we evaluate Switchboard (SB) against the round-robin (RR) and locality-first (LF) baselines. Our experiments show: (a) Switchboard can substantially reduce provisioning costs (51% compared to RR and 23% compared to LF). (b) Switchboard achieves average call latencies comparable to LF which is a significant improvement over RR (2.2×). (c) Switchboard needs to migrate only a small fraction of calls (1.53%), which is the same as LF. (d) The controller is performant and scalable - a single instance of a controller can handle 1.4× the current demand as observed by Microsoft Teams.

### 6.1 Evaluation metrics

We evaluate Switchboard using the following metrics.

(1) **Mean average call latency (Mean ACL:)** For each call, we compute the average latency across all its call legs (ACL). Then, we compute the mean ACL across all calls and report this value for each experiment.

(2) **Total WAN capacity:** We concern ourselves only with the capacity on the inter-country WAN links, since these tend to have a disproportionately high cost relative to intra-country links. We consider the peak rate (Gbps) of Microsoft Teams traffic on each link, and not the traffic volume (GB), since *the peak rate is what determines the provisioning cost*. We report the sum of the peaks across all WAN links, even if the individual peaks occur at different times. We note that WAN bandwidth costs are based on overall traffic peak, including the non-"Microsoft Teams" traffic that may be flowing on the same links. While we report only the capacity required to support Microsoft Teams's peak traffic, our formulation can be extended to include the non-Microsoft Teams traffic to minimize the overall peak.

(3) **Compute cores:** We consider the number of compute cores needed for MP servers across all DCs. As with WAN capacity, we calculate the sum of the peak compute core usage at each DC.

(4) **Cost:** While we look at the peak usage of WAN bandwidth and compute cores since they determine the network and compute cost respectively, their combined cost is what determines the total cost of provisioning resources for Microsoft Teams, which is what Switchboard intends to minimize.

## 6.2 Data sources

We run our experiments using 15 months of Microsoft Teams's call records from September 2021 to November 2022.

**Latency:** Microsoft Teams's call records database stores the latency for each participant in a call, *i.e.,* for each call leg. Each record contains the DC where the MP server was located as well as the participant location (country). Note that the logs only contain the latency corresponding to the actual MP server location for a call. To estimate the counterfactual, i.e., the latency with a different choice of MP server location, we pool together latency information across all calls and then estimate the latency between each MP location (DC) and participant location (country) as the median of all the recorded latencies for this pair.

**Costs:** To determine cost of each provisioning approach, we use the internal per-DC compute costs provided by Azure (per core, per unit-time). This amount varies significantly based on region and DC location. To characterize network cost, we obtain the cost per unit bandwidth usage cost in a given region, and divide it by the average number of links used by traffic in that region.

**Forecasting call count:** We used the call records of Microsoft Teams from September 2021 to May 2022 (9 months) to forecast call count for each call config for the months of September 2022 to November 2022. The 3-month look-ahead in forecasting reflects the lead time required to provision resources, especially WAN capacity. Since Switchboard's core contribution is to optimize resource management, and not forecasting, we first evaluate Switchboard (SB) versus RR and LF with the *ground truth* call counts (Table 3), and then evaluate the difference between the ground truth and forecasted call counts (Table 4). We also present a separate evaluation of the accuracy of forecasting in §6.5.

## 6.3 MP capacity provisioning

In this section, we evaluate and compare the three provisioning techniques: RR, LF, and Switchboard (SB). Our experiments run using data from the month of September 2022 to November 2022 consisting of 180+ million calls. To provide greater insights, we report the results separately for cases with and without the provisioning of backup capacity (Equations 7 and 8 for SB).

Table 3 reports the total compute cores and WAN capacity provisioned, the corresponding cost, and the mean ACL for RR, LF, and SB. We normalize all values with respect to the values for the RR baseline for reasons of confidentiality. As described in §3, RR achieves the lowest compute cores provisioned, but at the cost of provisioning large amounts of WAN capacity. LF, on the other hand, achieves the best mean ACL (45% of RR) and provisions much less WAN capacity in comparison to RR (55% of RR), but with the tradeoff of provisioning 10% more compute cores than RR.

SB achieves the best of both worlds by minimizing cost subject to an ACL constraint of 120ms. Table 3 shows that with backup capacity, SB uses the same number of compute cores as RR, but uses 22% less WAN capacity than LF (0.43× vs 0.55×) and 57% less WAN capacity than RR, while achieving the same mean ACL as LF. By doing so, SB saves 51% cost compared to RR and 23% cost compared to LF without compromising latency.

SB's lower WAN capacity requirements than RR arises in large part because unlike RR which hosts calls to remote DCs, SB tends to host a large fraction of the calls on DCs which would result in the least impact on the WAN peak, typically the *closest* DC.

SB provisions less compute and WAN capacity than LF primarily because LF optimizes for call latency, whereas SB optimizes for cost while keeping latency constrained. To reduce cost while satisfying the constraint, SB is able to reuse, "for free", some of the already existing WAN capacity which had to be provisioned for serving calls at a different (peak) time. SB takes advantage of the shift in peak load across locations by having some calls hosted in a DC that is remote with respect to the location of the majority of participants. This results in an even lower WAN capacity requirement than LF.

Note that SB trades off some latency compared to LF (0.51× vs 0.45× in the case without backup capacity) while still meeting the specified latency constraint in return for the WAN bandwidth and cost gains.

Furthermore, when the need to offload calls to a remote DC arises, SB preferentially offloads audio calls, followed by screen-share calls, before lastly offloading video calls. The reason is that the network load imposed by audio calls is much smaller as compared to the compute load shed via offloading, e.g., as discussed in Table 1, video calls would incur about 30-40× higher network load while offloading 2-4× compute as compared to audio calls.

As expected, SB results in a sharply reduced latency compared to RR, owing to two reasons, a) SB tries to minimize WAN peaks while provisioning by travelling shorter paths – which is positively correlated with lower latency, and b) SB attempts to minimize latency during allocation with the provisioned resources. It turns out in our experiments that after provisioning for failure scenarios, i.e. with backup capacity constraints, SB does not need to rely on offloading calls to remote DCs to smoothen out peaks because of the additional backup capacity provisioned provides sufficient local

| | **Without backup** | | | | **With backup** | | | |
|---|---|---|---|---|---|---|---|---|
| Scheme | Cores | WAN | Cost | Mean ACL | Cores | WAN | Cost | Mean ACL |
| **RR** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| **LF** | 1.08 | 0.18 | 0.35 | 0.45 | 1.10 | 0.55 | 0.64 | 0.45 |
| **SB** | 1.00 | 0.14 | 0.29 | 0.51 | 1.00 | 0.43 | 0.49 | 0.45 |

**Table 3: Resources provisioned (compute cores and total WAN capacity in Gbps), cost incurred ($), and mean ACL (ms) for RR, LF and SWITCHBOARD (SB) with and without backup capacity, *i.e.,* with and without considering failure scenarios. The original units have been removed since all numbers are normalized to the values of RR.**

| | **Without backup** | | **With backup** | |
|---|---|---|---|---|
| Scheme | Cores | WAN | Cores | WAN |
| **RR** | -5% | -13% | -5% | -13% |
| **LF** | -6% | -8% | -7% | -11% |
| **SB** | -5% | +10% | -5% | -11% |

**Table 4: Difference between resources provisioned using ground-truth records and using forecasts for call counts.**

compute for the no-failure scenario, which is typically the case. So, in effect, the call placement with SB ends up being identical to LF in the absence of failures by using the backup capacity. Hence, the ACL values are identical for LF and SB.

Table 4 reports the difference between the amount of resources provisioned using forecasts of call counts and ground truth records. A -ve value implies that the forecast over-provisioned resources as compared to what the ground truth required. Note that provisioning based on forecasts is reasonably accurate - within +/-13% of provisioning done with ground truth, across all schemes. We noticed that our forecast module under-estimated the inter-country calls, and over-estimated the intra-country calls, owing to the difference in the nature of the trend and number of calls in them. The total calls (both inter- and intra-country) were still over estimated, which explains the -ve values in the number of cores required across all schemes. As discussed in §3.1, RR sprays calls all around which results in it's WAN bandwidth requirement being proportional to the number of total calls – explaining the -ve value for RR's WAN capacity. LF's WAN capacity is a function of both inter-country calls, as well as the intra-country calls whose closest DC is far off. Given the over-estimation of intra-country calls, the WAN requirement for the forecast call counts is still more than the ground truth, albeit the difference is lesser than RR. SWITCHBOARD on the other hand, is able to effectively reuse the WAN links to pack the intra-country calls in a way that the WAN requirement for the forecast depends largely on the inter-country calls, which is under-estimated, resulting in a +ve value. However, after considering failure scenarios, where both LF and SB resort to hosting calls to far-off DCs, the SB prediction for WAN capacity is again impacted by the intra-country calls and hence becomes negative.

## 6.4 Frequency of call migration

As discussed in §5.4, when a new call arrives, the choice of initial MP location might turn out to be incorrect and we might have to migrate the call once the call config is known, say a few minutes into the call. From Fig.8, we see that 300 seconds (5 minutes) into the call, a significant majority (about 80%) of participants have joined. Therefore, we freeze the call config at $A$ = 300 seconds into the call.

Call migrations are undesirable since these could lead to a user-perceived glitch. However, migration across DCs is unavoidable since the initial choice of hosting location could be wrong and continuing to host a call in the wrong DC (e.g., one that is far from where the majority of call participants are located) would have a negative bearing on not just latency but also WAN bandwidth usage.

Therefore, in our evaluation here, we focus on the frequency of the inter-DC migrations. Our experiments show that both SB needs to migrate only 1.53% of calls need to stick to the pre-computed allocation plan. This is the same as the fraction of call migrations required by LF, which requires to know the exact spread of all participants to minimize the ACL. To further cut down such call migration, we could use history to predict the call configuration, or atleast the location of the majority of participants. We touch on this briefly in §8.

## 6.5 Provisioning forecasting

As mentioned in §5.2, we forecast the expected number of calls for call configs in future using Holt-Winters timeseries forecasting. We built the timeseries for individual call configs using 9 months of data and use it to predict the call config 3 months into the future. For each call config we calculate RMSE (Root Mean Square Error) and MAE (Mean Absulate Error) based on the forecasted count and ground truth. We normalize it using the peak number of calls in ground truth. This way, elephant and mice call configs are treated in the same way. Fig.9 shows the CDF for RMSE and MAE for top 1000 call configs. We observed reasonably small median RMSE and MAE values – 13% and 8%, respectively.

## 6.6 Controller benchmark

We evaluate the controller throughput to check whether we can support the peak rate of arrival of new calls and call participants. We run our controller on a 4-core D-series VM on Azure with 16GB RAM. We replay the trace for 24 hours on a typical weekday. The trace contains millions of calls and events (participants joining and media changes). We use Azure's Redis[11] offering in the same DC as the controller.

We evaluate the throughput supported by the controller as we increase the number of threads. Recall that these threads write back to Redis, the changes to the call config as additional participants join a new call. We found the latency for each *write* operation is in a reasonable range of 0.3ms to 4.2ms. Fig.10 shows the throughput as we vary number of threads (normalized to the peak traffic seen in the trace). We found that we can support 1.4× peak load using 10 threads. Importantly, the throughput supported scales with number of threads. These results demonstrate that our the controller is performant enough.

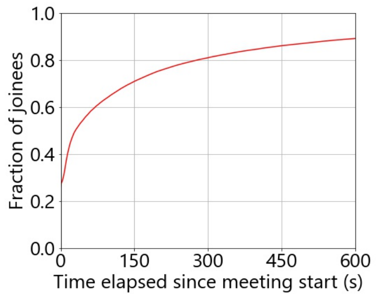


**Figure 8: Ave. fraction of participants joining since meeting starts.**

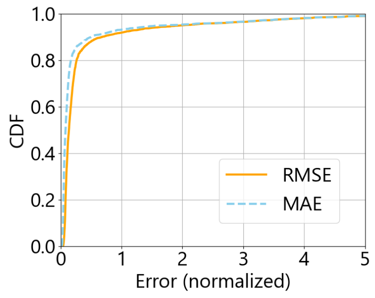


**Figure 9: CDF for normalized RMSE and MAE for call configs.**

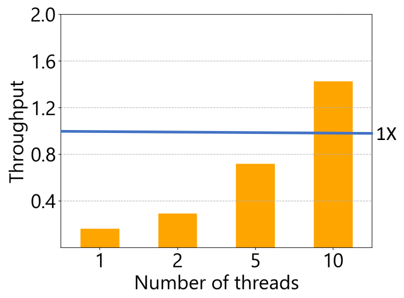


**Figure 10: Throughput using different Redis writing threads.**

## 7 RELATED WORK

**Media-stack optimizations:** Conferencing applications such as Microsoft Teams[9], Zoom[12], Skype[8] and Google Meet[4] have received widespread attention[17, 25, 29, 32]. Recent work has focused on adapting video quality based on the network conditions[29], codec and transport collaboration[16, 22, 42], or low latency video transport network[28]. Our work complements these studies by focusing on resource provisioning for conferencing systems.

**Server selection:** There has been extensive prior work to study server selection[27, 31, 37–39, 41], and DC/replica selection[19, 21]. Taiji[19] assigns weights at edge sites for splitting traffic across individual DCs. [35, 36] focuses on replica selection to improve the tail latency. However, these systems do not take advantage of jointly provisioning compute and network, while also leveraging backup capacity to optimize for performance.

**Provisioning:** Recent works[7, 15, 18, 34] have been seminal systems focusing on provisioning *network capacity* for online services. However, they do not consider cross-resource provisioning even though they are inter-dependent and the cost depends on both. Similarly, other systems[26, 30] have focused on proactive provisioning for network faults. Switchboard focuses on proactive provisioning for compute (MP servers) as well as network.

**Leveraging time varying demands:** Like Switchboard, prior work has also leveraged time varying demands but in different contexts. TIVC[40] improves cluster utilization based on time varying demands. Approv[34] leverages it to schedule network traffic.

**Capacity limits in cloud:** It has been reported that customers sometime cannot get resources because the cloud runs out of resources in certain regions/availability-zones[1–3]. Additionally, it is also reported that Zoom runs its own hardware for enterprise customers[13]. These point to the importance of planned provisioning of adequate resources, especially for large scale services.

## 8 DISCUSSION

**Predicting call configuration for incoming calls:** If Switchboard could accurately predict the config for each new incoming call, it could potentially eliminate inter-DC migrations. Predicting call configs is particularly feasible for recurring calls. We train a two-part model on a small set of 24,000 call records which consisted of calls with at least 3 past occurrences as part of the same meeting series. We use a variable length multi-order Markov chains (MOMC) setup to capture temporal predispositions in terms of attendance that a participant exhibits over the past few instances. We feed the output of the MOMC apparatus into a logistic regression that predicts the desired binary — the attendance of that particular participant in the upcoming instance, which gets aggregated across participants to get the call-config. We compute Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) with the predicted and ground truth participant count of each country in the call config. While predicting results for 3,600 unseen meeting instances, we observed an average RMSE value of 0.97 and an average MAE of 0.90 against a baseline that predicted the call config simply based on the previous call instance which had RMSE and MAE of 24.90 and 23.60 respectively. Note that for calls with a large number of participants (dozens or even hundreds), the baseline prediction is particularly inaccurate, and our history-based MOMC approach does much better. Making this prediction model more scalable can significantly reduce inter-DC migrations.

**Using Switchboard for other applications:** Switchboard can be adapted to other online applications with identical workload characteristics, for instance, multiplayer cloud gaming. Other user-facing applications can take still take advantage of the key ideas behind resources provisioning and allocation, though the formulation would be different.

## 9 ACKNOWLEDGEMENTS

We thank our shepherd, Maria Gorlatova, and the anonymous SIGCOMM reviewers, for their feedback. We thank Bhaskar Kataria for their feedback on the paper. Rahul Bothra thanks his parents – Anju and Ratan Bothra for their sacrifices and support in helping him pursue a career in research.

## REFERENCES

[1] https://www.reddit.com/r/googlecloud/comments/ggyipd/asiasouth1a_does_not_have_enough_resources/.
[2] https://stackoverflow.com/questions/52684656/the-zone-does-not-have-enough-resources-available-to-fulfill-the-request-the-re.
[3] https://stackoverflow.com/questions/52113153/ec2-error-starting-instances-insufficient-capacity.
[4] Google Meet. https://apps.google.com/meet.
[5] Holt-Winters exponential smoothing. https://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.ExponentialSmoothing.html.
[6] Latency Test for VoIP: How it Impacts Call Quality and Ways to Fix It. https://getvoip.com/blog/2021/06/21/voip-latency-test.
[7] Long-Term Capacity Planning in Facebook Network. https://www.researchgate.net/publication/323564414_Long-Term_Capacity_Planning_in_Facebook_Network.
[8] Microsoft Skype. https://www.skype.com/en/get-skype/.

[9] Microsoft Teams. https://www.microsoft.com/en-us/microsoft-teams/group-chat-software.

[10] Microsoft Teams user growth. https://www.businessofapps.com/data/microsoft-teams-statistics/.

[11] Redis in-memory data store. https://redis.io.

[12] Zoom. https://zoom.us/.

[13] Zoom on AWS. https://www.datacenterdynamics.com/en/news/most-zoom-runs-aws-not-oracle-says-aws/.

[14] Zoom user growth. https://backlinko.com/zoom-users.

[15] S. S. Ahuja, V. Gupta, V. Dangui, S. Bali, A. Gopalan, H. Zhong, P. Lapukhov, Y. Xia, and Y. Zhang. Capacity-efficient and uncertainty-resilient backbone network planning with hose. In *ACM SIGCOMM 2021*, pages 547–559, 2021.

[16] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo. Analysis and design of the google congestion control for web real-time communication (WebRTC). In *ACM MMSys*, 2016.

[17] H. Chang, M. Varvello, F. Hao, and S. Mukherjee. Can you see me now? A measurement study of Zoom, Webex, and Meet. In *ACM IMC*, 2021.

[18] Y. Chang, S. Rao, and M. Tawarmalani. Robust validation of network designs under uncertain demands and failures. In *USENIX NSDI*, 2017.

[19] D. Chou, T. Xu, K. Veeraraghavan, A. Newell, S. Margulis, L. Xiao, P. M. Ruiz, J. Meza, K. Ha, S. Padmanabha, et al. Taiji: managing global user traffic for large-scale internet services at the edge. In *ACM SOSP*, 2019.

[20] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein. Maglev: A fast and reliable software network load balancer. In *USENIX NSDI*, 2016.

[21] A. Flavel, P. Mani, D. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev. FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs. In *USENIX NSDI*, 2015.

[22] S. Fouladi, J. Emmons, E. Orbay, C. Wu, R. S. Wahby, and K. Winstein. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *USENIX NSDI*, 2018.

[23] R. Gandhi, Y. C. Hu, C.-k. Koh, H. H. Liu, and M. Zhang. Rubik: Unlocking the power of locality and end-point flexibility in cloud scale load balancing. In *USENIX ATC*, 2015.

[24] M. Guo, M. H. Ammar, and E. F. Zegura. Selecting among replicated batching video-on-demand servers. In *ACM NOSSDAV*, 2002.

[25] J. Jiang, R. Das, G. Ananthanarayanan, P. A. Chou, V. Padmanabhan, V. Sekar, E. Dominique, M. Goliszewski, D. Kukoleca, R. Vafin, and H. Zhang. Via: Improving Internet Telephony Call Quality Using Predictive Relay Selection. In *ACM SIGCOMM*, 2016.

[26] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. Kleinberg, P. Lapukhov, C. L. Lim, and R. Soulé. Semi-oblivious traffic engineering: The road not taken. In *USENIX NSDI*, 2018.

[27] M. Kwon, Z. Dou, W. Heinzelman, T. Soyata, H. Ba, and J. Shi. Use of Network Latency Profiling and Redundancy for Cloud Server Selection. In *IEEE International Conference on Cloud Computing*, 2014.

[28] J. Li, Z. Li, R. Lu, K. Xiao, S. Li, J. Chen, J. Yang, C. Zong, A. Chen, Q. Wu, C. Sun, G. Tyson, and H. H. Liu. LiveNet: A Low-Latency Video Transport Network for Large-Scale Live Streaming. In *ACM SIGCOMM*, 2022.

[29] X. Lin, Y. Ma, J. Zhang, Y. Cui, J. Li, S. Bai, Z. Zhang, D. Cai, H. H. Liu, and M. Zhang. GSO-simulcast: global stream orchestration in simulcast video conferencing systems. In *ACM SIGCOMM*, 2022.

[30] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter. Traffic engineering with forward fault correction. In *ACM SIGCOMM*, 2014.

[31] Z. Liu, M. Lin, A. Wierman, S. Low, and L. L. H. Andrew. Greening Geographical Load Balancing. *IEEE/ACM Transactions on Networking*, 2015.

[32] K. MacMillan, T. Mangla, J. Saxon, and N. Feamster. Measuring the performance and network utilization of popular video conferencing applications. In *ACM IMC*, 2021.

[33] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, et al. Ananta: Cloud scale load balancing. In *ACM SIGCOMM*, 2013.

[34] H. Sharma, P. Thakkar, S. Bharadwaj, R. Bhagwan, V. N. Padmanabhan, Y. Bansal, V. Kumar, and K. Voelbel. Optimizing network provisioning through cooperation. In *USENIX NSDI*, 2022.

[35] S. M. Shithil and M. A. Adnan. A prediction based replica selection strategy for reducing tail latency in distributed systems. In *2020 IEEE CLOUD*, 2020.

[36] L. Suresh, M. Canini, S. Schmid, and A. Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *USENIX NSDI*, 2015.

[37] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao. Dissecting Video Server Selection Strategies in the YouTube CDN. In *IEEE ICDCS*, 2011.

[38] W. Wang and G. Casale. Evaluating weighted round robin load balancing for cloud web services. In *IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2014.

[39] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. DONAR: Decentralized Server Selection for Cloud Services. In *ACM SIGCOMM*, 2010.

[40] D. Xie, N. Ding, Y. C. Hu, and R. Kompella. The only constant is change: Incorporating time-varying network reservations in data centers. In *ACM SIGCOMM*, 2012.

[41] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein. Dynamic Service Placement in Geographically Distributed Clouds. *IEEE Journal on Selected Areas in Communications*, 2013.

[42] A. Zhou, H. Zhang, G. Su, L. Wu, R. Ma, Z. Meng, X. Zhang, X. Xie, H. Ma, and X. Chen. Learning to coordinate video codec with transport protocol for mobile video telephony. In *ACM International Conference on Mobile Computing and Networking*, 2019.