Constellate: Establishing the opportunity for Distributed Unit pooling in real-world 5G Radio Access Networks

Sri Pramodh Rachuri[1], Anshul Gandhi[1], Gueyoung Jung[2*], Shankaranarayanan Puzhavakath Narayanan[2], Alex Zelezniak[2*]
[1]Stony Brook University, [2]AT&T Labs
srachuri@cs.stonybrook.edu, anshul@cs.stonybrook.edu, gueyoung.jung@gmail.com, sn081k@att.com, azelezniak@gmail.com

***Abstract*—As the adoption of Virtualized Radio Access Networks (vRAN) is gaining momentum in 5G networks, Mobility Network Operators are considering a Centralized RAN (CRAN) architecture that moves the baseband functions to a far-edge cloud in order to gain dimensioning flexibility, resiliency and improved RAN performance. However, there have been limited studies on the benefits of centralization in improving RAN compute utilization, especially in the context of pooling the compute-intensive Distributed Unit (DU) resources. In this paper, we present the first study on the benefits of pooling in improving DU server utilization. Using longitudinal traces from a real-world 5G network, we show that significant Capex and Opex gains of 84% and 94%, respectively, can be obtained through fine-grained pooling at a granularity of 1 second. We also present an affinity-based and dynamic pooling algorithm that can reduce the pooling overheads while still achieving significant pooling gains.**

***Index Terms*—5G, Virtualized RAN, DU pooling, RAN utilization**

I. Introduction

Mobility Network Operators are increasingly deploying virtualized Radio Access Networks (vRAN) [1]–[4], where the traditional baseband units (BBUs) are replaced with a disaggregated and virtualized Distributed Unit (vDU) and Centralized Unit (vCU). Unlike BBUs, which are designed as monolithic applications with highly customized software running on purpose-built hardware to provide high performance, the vRAN software is deployed on a containerized cloud-native platform [5]–[8] hosted on Commercial-Off-The-Shelf (COTS) servers [9], [10].

The disaggregated vRAN architecture allows flexible deployment options for the vCU and vDU depending on the performance and resource constraints. While the vCUs are typically centralized further into the network at regional cloud locations, the vDUs can be located at the cell site similar to the BBUs today in a distributed RAN configuration or can be located at a far-edge cloud in a Centralized RAN (CRAN) configuration. Figure 1 shows the schematic of a centralized vRAN deployment with a Radio Unit (RU) located at the cell site, a vDU located at a far-edge cloud (like Central Offices or Mobile Telephone Switching Offices), and a vCU located in a regional cloud data-center. The location of the vDU and vCU is primarily determined by the latency requirements on the fronthaul ($\approx 125\mu s$) and midhaul ($\approx 5ms$).

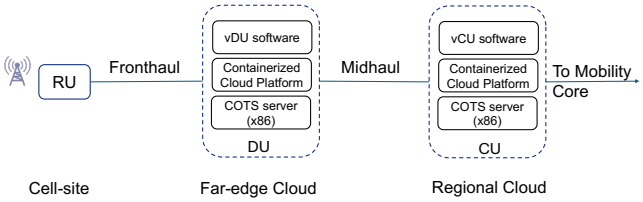


Fig. 1. Architecture of a Centralized vRAN Deployment.

Traditionally, BBUs are designed for a fixed capacity to serve a pre-defined number of RUs, with traffic from any given RU being typically served by the same BBU. These fixed configurations make it challenging to scale BBUs dynamically according to the variability in workloads, resulting in sub-optimal utilization of BBU compute resources. In contrast, vDUs and vCUs are designed to be cloud-native and expected to scale as the traffic changes. This dimensioning flexibility can be leveraged to gain statistical multiplexing benefits of centralization to improve the overall compute utilization by *pooling compute resources (servers) across multiple vDU instances.* While both vCU and vDU compute can be pooled, in this paper, we focus on pooling the vDU which requires significantly higher compute resources. By flexibly (re)mapping the traffic from RUs to a pool of vDUs, the underlying compute resources can scale elastically with the variability in workloads, resulting in improved utilization.

However, the benefits of pooling DU compute resources in 5G networks have not been well studied due to the scarcity of fine-grained RAN workload data. Typically, RAN data is collected along two related but independent dimensions: (i) Cell level aggregates of Key Performance Indicators (KPIs) are collected periodically over longer time periods (e.g., 15 minutes) to reduce the overhead of data collection and storage, and (ii) Session level aggregates of the KPIs collected over the duration of the independent UE (User Equipment) sessions

∗ Authors were employed at AT&T Labs during the duration of this work.

which can vary from a few seconds to few minutes. In this paper, we develop a novel methodology to interpolate the overlapping KPIs obtained from the above two dimensions and generate more fine-grained estimates of the vDU workloads.

We conduct an extensive study on the benefits of pooling DU compute using traces from a large commercial 5G network. We propose three algorithms to improve the pooling efficiency in terms of Capital Expenditure (Capex) and Operational Expenditure (Opex) savings, while minimizing the overhead of reassigning traffic across vDUs. Prior works have explored the benefits of centralization in 4G networks with traditional BBUs [11]. More recent works have developed mechanisms that redirect traffic between vDUs in the context of improving resiliency [12]. To the best of our knowledge, this is the first work that performs an extensive study towards understanding the benefits of pooling in 5G networks.

Our contributions in this paper are as follows: First, we present an affinity-based pooling algorithm that reduces the overheads of reassigning traffic across vDUs. Next, we develop a change-point detection based algorithm to determine when the pooling reassignments should be triggered to further reduce overheads. Then, we develop a novel interpolation based methodology to generate fine-grained estimates of the DU workload (upto 1 second) from periodic coarse-grained KPIs (15 min) and session information. Finally, by analyzing large-scale traces from a real-world network processed through our interpolation method, we show that pooling DU compute resources in a fine-grained manner can achieve significantly higher Capex and Opex savings.

Our evaluation shows that while coarse-grained pooling, such as at the granularity of 12 hours, provides substantial savings that capture the diurnal patterns in the workload, fine-grained pooling is able to provide significantly better overall utilization. For instance, pooling at 1 sec granularity has 84%, 94% capex and opex improvements over pooling at 12 hour granularity. Using our Affinity-based and Dynamic pooling algorithms, we show that it is possible to achieve 40% reduction in pooling overheads, while trading-off 22% in corresponding opex gains. Since our evaluation is focused on establishing the pooling efficiency with different pooling strategies and overheads with remapping traffic within a pool, the insights and strategies presented in this paper can be used in different pooling implementations.

II. Virtualized DU Pooling

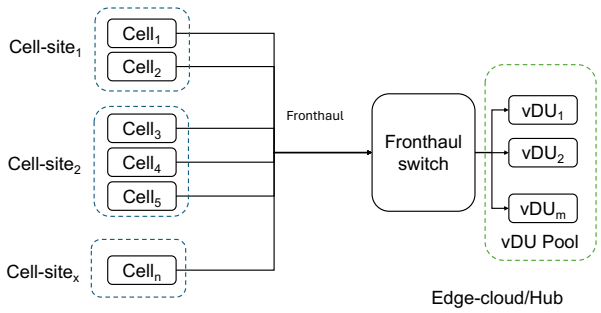


Fig. 2. Architecture of a pooled vDU cluster

Figure 2 shows a high-level architectural depiction of vDU Pooling. Typical cell-sites consist of multiple RUs with multiple NR (New Radio) *Cells*, each with a unique NR Cell Identifier[1] [13]. These cells are mapped to a vDU pool located at a hub location over a fronthaul transport that meets the latency requirements of $\approx 125\mu s$. At the hub, an aggregation switch helps multiplex the fronthaul traffic from the multiple cells and routing it to the appropriate vDU instances.

Pooling Definition: In this paper, we define Pooling as the problem of finding an optimal (re)assignment or (re)mapping of cells to vDU instances in each pooling interval while minimizing the overheads (or cost) of pooling. The precise overhead of remapping depends on the specific implementations including UE context reconfiguration, coordinating state information across vDUs, and the hardware/software architecture of the vDU pool [14]–[16]. Hence, to keep the metric relevant and independent of the implementation choices, we measure the pooling overheads as the number of remappings (or #remaps) in each pooling interval in our evaluation of the pooling strategies.

We formulate pooling as a variant of the classical bin-packing problem (known to be NP-hard) [17] where the Cells and vDU instances correspond to the items and bins, respectively. We conduct our study over a wide range of pooling intervals (or pooling granularity) all the way to a granularity of 1 second.

Pooling algorithms: We implement three pooling algorithms to map cells to vDUs.

(i) A **Greedy (*G*)** algorithm, based on "Best Fit Decreasing" bin packing algorithm [18], that maps cells to vDUs at every pooling interval based on the current overall utilization of the vDU pool. ***G*** aims to minimize the number of vDUs (and therefore servers) required to serve the workload by finding the vDU with ***tightest fit*** (smallest remaining capacity) for each cell. For example, it picks the cell with the highest utilization (largest first) if sufficient capacity is available in the vDU. If not, it maps the cell to a new vDU.

Algorithm 1 shows the pseudo-code for ***G*** , which takes the cell utilization U_c as input and assigns each cell to a vDU while minimizing the number of the vDUs. It starts by sorting the cells by utilization in descending order. For each cell, it finds the vDU that can accommodate the cell with the least remaining capacity. If no such vDU is found, it creates a new vDU and assigns the cell to it. At every pooling interval, it starts with empty mapping between cells and vDUs and runs the entire algorithm without considering the previous mapping.

While the ***G*** is simple, we show that it incurs higher pooling overheads as it does not consider prior mapping between the cells and vDUs, which we address next.

(ii) An **Affinity-Based (*AB*)** Algorithm (pseudo-code shown in Algorithm 2) that reduces the pooling overhead of remapping of cells to vDUs by retaining each cell on the same vDU unless its utilization exceeds (or falls below) pre-defined thresholds. During the initial mapping, ***AB*** maps cells to vDUs

[1]RU may have multiple Cells

Algorithm 1 Greedy Pooling Algorithm (***G***)

Input: Cell Utilization U_c
Output: Cell to vDU assignment $A_{c \to vDU}$
Initialization: $A_{c \to vDU} = \emptyset$
Sort cells by utilization at t_i (decending)
for each cell c **do**
    Sort vDUs by utilization (decending)
    $best_{vDU} = \emptyset$
    for each vDU **do**
        if $U_{vDU} + U_c \leq 1$ **then**
            $best_{vDU} = vDU$
            break
    if $best_{vDU} = \emptyset$ **then**
        Create a new vDU and assign $best_{vDU}$ to it
    $A_c = best_{vDU}$

similar to the ***G*** algorithm. However, if a vDU utilization goes over a high-watermark in a subsequent pooling interval, ***AB*** remaps the busiest cells to other vDUs based on the tightest fit. And, when a vDU load falls below a low-watermark (25% in our work), ***AB*** moves cells from this vDU to other vDUs with a tight fit.

(iii) Finally, we also propose ***Dynamic pooling*** (***D***) algorithm that tries to minimize the pooling overheads by reducing the frequency of pooling (or the number of pooling intervals) based on the workload changes. ***D*** detects change points to identify changes in cell utilization and triggers a cell remapping only if the utilization changes by more than a pre-defined threshold (hyper-parameter in our study) in any given time interval. ***D*** has a cooldown period (another hyper-parameter) to prevent frequent remappings.

Algorithm 3 shows the pseudo-code for the ***D*** algorithm. It takes the cell utilization as a function of time $U_c(t)$, Change Point Threshold, and Cooldown-Period as input. It calculates the total utilization of all cells at every second and finds the maximum utilization every minute. If the difference between the maximum utilization of two consecutive minutes is greater than the threshold, it checks if the cooldown period has passed since the last pooling interval. If it has, ***D*** adds the timestamp to the list of pooling timestamps. It then returns the list of pooling timestamps.

Finally, we note that fine-grained pooling allows a wide range of implementation options depending on how subsets of traffic from the RUs are mapped to vDU instances. While mapping workload per UE session is a feasible design point, we find that this requires maintaining uplink affinity between the cells and the vDU instances where uplink control messages and grants were received. Also, long lasting UE sessions would require explicit migration across vDU instances using handover mechanisms, which can incur further overheads. Therefore, we believe that mapping cells to vDUs provides a good trade-off between the capex/opex gains and the complexity of maintaining a coherent session-state across multiple vDU servers. Integrating our algorithms in a vDU implementation is an ongoing work but out of scope for this paper.

Algorithm 2 Affinity-Based Pooling Algorithm (***AB***)

Input: Cell Utilization U_c
Output: Cell to vDU assignment $A_{c \to vDU}$
Initialization: $A_{c \to vDU} = A_{c \to vDU,\text{last}}$
if $A_{c \to vDU} = \emptyset$ **then** $A_{c \to vDU} = \text{Greedy}(U_c, C_{vDU})$ at t_0
for each vDU **do**
    if $U_{vDU} > 1$ **then**
        Sort cells assigned to vDU by utilization
        for each cell c **do**
            Sort vDUs by utilization (decending)
            $best_{vDU} = \emptyset$
            for each vDU **do**
                if $U_{vDU} + U_c \leq 1$ **then**
                $best_{vDU} = vDU$
                break
            if $best_{vDU} = \emptyset$ **then**
                Create a new vDU and assign $best_{vDU}$ to it
            $A_c = best_{vDU}$
            if $U_{vDU} < 1$ **then** break
    if $U_{vDU} <$ Low-Watermark **then**
        Sort cells assigned to vDU by utilization (decending)
        for each cell c **do**
            Sort vDUs by utilization (decending)
            $best_{vDU} = \emptyset$
            for each vDU **do**
                if $U_{vDU} + U_c \leq 1$ **then**
                $best_{vDU} = vDU$
                break
            if $best_{vDU} = \emptyset$ **then** break
            $A_c = best_{vDU}$

Algorithm 3 Dynamic Pooling Algorithm (***D***)

Input: Cell Utilization as a function of time $U_c(t)$, Change Point Threshold, Cooldown-Period
Output: List of Pooling timestamps $P =< p_0, p_1, \ldots >$
$U_{\text{total}}(t) = \sum_c U_c(t)$
$U_{1min}(t) = \max_{t' \in [t, t+60]} U_{\text{total}}(t')$
$P = \emptyset$
for each minute t **do**
    if $|U_{1min}(t) - U_{1min}(t-60) >$ Threshold$|$ **then**
        if $t - p_{\text{last}} >$ Cooldown-Period **then** $P = P \cup t$

III. Trace Data Collection

We collected two sets of longitudinal traces from a large commercial 5G network for a period of 7 days. The first set of traces consists of 15 minute aggregates of Physical Resource Blocks (PRB) utilization and data volume (bytes) for each cell mapped to a given gNodeB (e.g., 3GPP TS 28.552 [19]). The second set of traces consists session start times, durations and aggregated session-level data volumes (bytes) along with the cell identifiers associated with each session. All cell

TABLE I
DISTRIBUTION OF CELLS ACROSS CLUSTERS IN OUR DATA

	Low Band			Mid Band			mmWave		
Cluster ID	1	2	3	1	2	3	1	2	3
# Cells	1328	158	63	1262	59	3	37	-	-

and session identifiers in both traces were anonymized with cryptographic hash functions while preserving the associations needed to correlate the above metrics used in our study.

We collect both traces for a total of 1549 cells of Low Band, 1324 cells of Mid Band and 37 mmWave cells from a large geographical region. We clustered these cells into three pools using K-means with a pooling radius of 10 kilometers each—which is an approximation of the acceptable fronthaul latency of 125μs. The centers of these clusters are present in Urban (ID 1), Suburban (ID 2), and Rural areas (ID 3). Table I shows the distribution of cells in the three clusters. The traces are obtained from deployments comprising traditional BBUs; in our study, we extend these traces to vRAN configurations.

PRB utilization as a unit of workload: The precise estimate of vDU compute resources required in any given time period for processing the workload from the cells mapped to it depends on many complex factors including the number of UEs, antennas, location of the UEs, and channel conditions [20]. Further, the amount of compute required also depends on vendor and operator specific features and carrier configurations. Hence Operators and RAN vendors estimate the needed capacity on the servers by aligning to a pre-established traffic model that captures the key inputs affecting the compute requirements. In the RAN processing pipeline, it is well-known that the majority of the processing time is spent on the L1 (Physical Layer) and L2 (Data Link Layer) processing, which is largely proportional to the number of PRBs processed in a given time period [21], [22]. Hence, we use PRB utilization per unit time as a proxy for the cell's workload, and the compute resources required to serve that cell. Mirroring the real-world deployments, we let each vDU serve multiple cells but any given cell can be served by only one vDU at a given time.

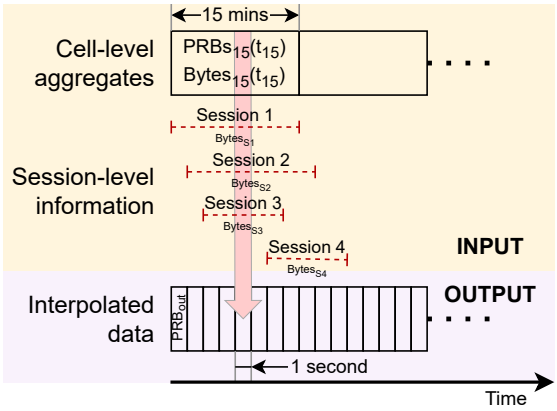
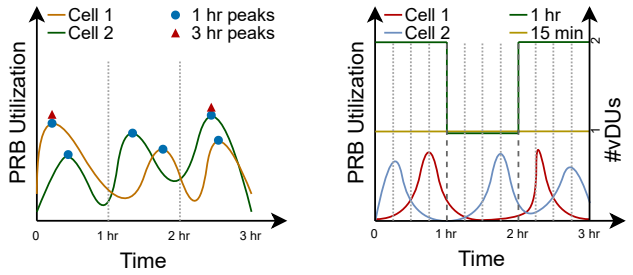


Fig. 3. Generating fine-grained workload data from coarse-grained traces and session-level information.

Generating fine-grained workload estimates from coarse-grained data sources: As described in Section I, a key challenge in evaluating the benefits of pooling is the lack



(a) PRB utilization for two cells over time highlighting the peak utilization time along with the #vDUs required. (b) PRB utilization for two cells over time at different times.

Fig. 4. Illustrative examples of pooling 2 cells.

of fine-grained workload data. In this paper, we present an interpolation-based methodology (illustrated in Figure 3) to generate fine-grained workload. We estimate fine-grained PRB utilization by distributing coarse-grained PRB-to-byte ratios across sessions using their precise timing and data volumes. First, we obtain the total PRB utilization ($PRBs_{15}(t_{15})$) and the total volume of bytes processed ($Bytes_{15}(t_{15})$) for a cell at a given time t_{15} with 15 minute granularity from the coarse-grained PRB utilization traces. Next, we use the session traces to get the aggregated data volumes ($Bytes_s$) per session along with the precise (milliseconds) start time ($T_{s,start}$ and end time $T_{s,end}$) of each session (s). We align all the active sessions associated with any given cell in each 15 minute window, and calculate the PRB utilization at fine-grained time intervals using the formulation below.

The set of sessions that are active at a given time t is:

$$S(t) = \{s | T_{s,start} \leq t \leq T_{s,end}\}$$

Let us say t_{15} is the last 15-minute interval before t. We can calculate the fine-grained PRB utilization for a given cell s at time t with 1-second granularity as:

$$PRB_{out}(t) = \frac{PRBs(t_{15})}{Bytes(t_{15})} \times \sum_{s \in S(t)} \frac{Bytes_s}{T_{s,end} - T_{s,start}} \quad (1)$$

This interpolation methodology makes a simplifying assumption of uniform distribution of PRB utilization within each session. While PRB utilization varies over time within each session, we note that this approximation works well in practice at scale when aggregating the large number of sessions associated with each cell. Further, a high percentile of these sessions are short-lived (in the order of seconds), which captures the fine-grained workload variability associated with each cell, and this estimate gets more accurate as the workload observed at each vDU gets aggregated across all the cells mapped to the vDU server. Finally, we verified that the fine-grained PRB utilization obtained using our interpolation-based methodology matches the 15-minute PRB utilization in the coarse-grained traces.

Determining required compute capacity: Determining the required compute capacity of a pooled vDU cluster is an essential first step in provisioning. In a pooled system, accommodating the variability and burstiness in workloads requires

provisioning the compute to handle the sum of peak workload observed across all the cells in a given pool during each pooling interval. Figure 4(a) shows the sum of peak utilization for the 3 hour pooling interval is higher than that of the sum of peak utilization in each 1 hour interval. In general, fine-grained pooling results in better overall compute utilization. However, fine-grained pooling *may not always* incur higher pooling overheads than coarse-grained pooling as shown in Figure 4(b). Here, assuming processing the peak workload at each cell requires a separate vDU, the num of remappings (two) is higher at 1 hr granularity compared to 15 min granularity (zero). This is also observed in our trace-driven evaluation (§IV-A).

vDU utilization and server configuration: We observe that vDU processing capacity depends on the server configuration, including the number of cores and networking. To simplify our study, we use (anonymized) configurations that match the BBU dimensioning observed in the network traces: Config 1 representing a low-band only vDU, Config 2 representing a mid-band only vDU, and Config 3 representing mmwave only vDU. During our evaluation, we choose the configuration that matches the cell's band.

IV. Evaluation

This section presents our results and key observations from our trace-driven evaluation of our pooling strategies. Since our focus is on understanding the opportunities and overheads of pooling, we do not include implementation specific metrics like QoS (Quality of Service) or QoE (Quality of Experience) metrics in our evaluation.

- **Capital Expenditure (Capex)** is the maximum number of vDUs required to serve the workload in the network and depends on the peak PRB utilization. Capex serves as a proxy for the building cost of the network.
- **Operational Expenditure (Opex)** is the mean number of vDUs required to serve the workload in the network and depends on the average PRB utilization. Opex is a proxy for the operational cost of the network.
- **Number of remaps (#Remaps)** is the sum (across time) of the number of cells that are moved from one vDU to another between pooling intervals. This is a proxy for the overhead of re-pooling.

Unless mentioned otherwise, we present values normalized to the corresponding 24-hour pooling granularity value. We select different vDU compute capacities for each band based on the server configurations. Table II lists all the parameters we use in our evaluation and their values; we use the underlined values in this table as the default values in our experiments.

TABLE II
Experiment parameters and values; default values are underlined.

Parameter	Values
Pooling Intervals	1 sec, 10 sec, 1 min, 5 min, 30 min, 1 hr, 3 hr, 12hr, 24 hr
Pooling Algorithm	Affinity-Based, Greedy
Server capacities	1x, 0.5x, 2x, 4x
Bands	Low-band, Mid-band, mmWave
Clusters	1 (Urban), 2 (Suburban), 3 (Rural)
Data Duration	1 Week, 1 Day

A. Pooling Opportunities

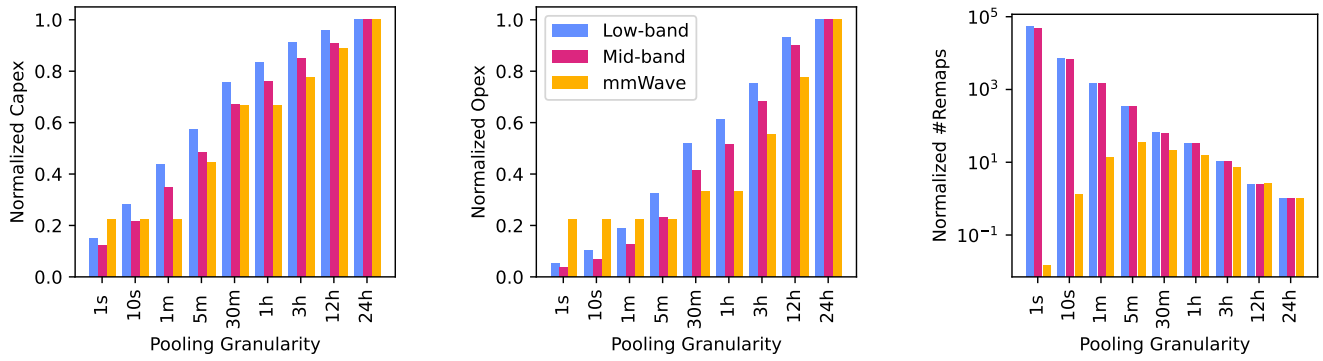
We start our evaluation by investigating the fundamental benefits (Opex, Capex) and overheads (#Remaps) of pooling. Figure 5 shows our results for cells running different bands in cluster 1 under Greedy pooling for the full length of traces. The values for each band are normalized by the corresponding 24hr pooling result for *that* band; as such, the depicted values should not be compared across bands.

Starting with Capex, we see that, as we pool at a finer granularity (going from right to left, from 24hr to 1s pooling interval), the Capex decreases. We see a similar general trend for Opex as well. Interestingly, the Capex and Opex savings *do not* plateau out as we go to very fine pooling granularities. For example, for Low-band, compared to 24hr pooling, we achieve a 56% reduction in Capex when we pool at 1min granularity. However, if we could go down to 1s granularity, we can additionally save more than 29% in Capex. As such, ***there are significant cost savings to be achieved if we can pool at very fine granularities***.

Across the bands, we find that the Mid-band exhibits a higher reduction in Opex, and also (to a lesser extent) Capex, with the pooling granularity. The Mid-band is comparatively more erratic than the Low-band due to the higher possible data rates [23] leading to higher savings when pooling more frequently. For mmWave, we observe that the Opex savings do start to converge at very fine granularities, unlike the other bands. This is due to quantization to the closest number of vDUs in mmWave.

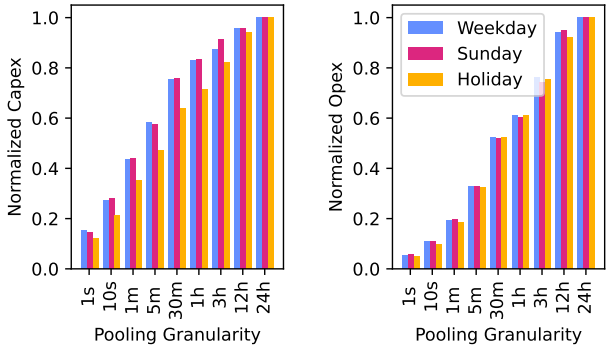
In terms of #Remaps, which we use as a proxy for pooling overhead, we see that all bands except mmWave have a steady decrease in #Remaps with an increase in pooling granularity; see Figure 5(c). At first glance, this may seem intuitive as the #Remaps should decrease as we pool less frequently (i.e., with a coarser pooling granularity). However, *this is not always the case*. For mmWave, as we move from left to right, the #Remaps initially *increases* as we pool at coarser granularities, and then decreases. Compared to other bands, we find that mmWave uses fewer vDUs. As such, at very fine granularities, the pooling is quite efficient, and so requires only a handful of vDUs for mmWave, resulting in few remaps. This scenario is similar to the example we saw in §II and Figure 4(b). As the pooling granularity increases, the pooling becomes less efficient, leading to more vDUs and remappings. Beyond a certain granularity (1min, in this case), the #Remaps decreases due to a decrease in the number of re-pooling occurrences. This result shows that ***#Remaps can exhibit non-monotonic behavior with pooling granularity***. This insight is significant as a finer pooling granularity may be incorrectly associated with a high overhead (proportional to #Remaps), which we find is not always the case.

Impact of specific days: Rather than aggregating the pooling results across the full 7 days of the trace, we can also



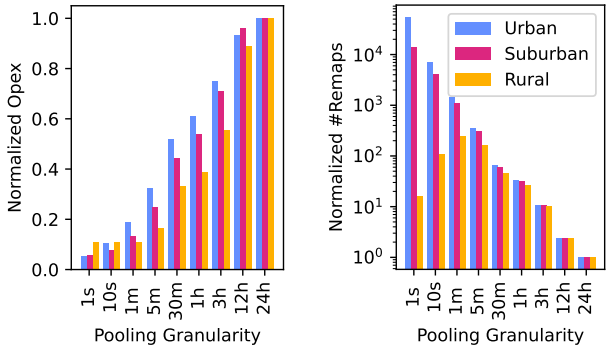
(a) Capex (b) Opex (c) Remaps

Fig. 5. Capex, Opex, and #Remaps for cells across different bands in cluster 1 with Greedy pooling. Values are normalized to 24hr pooling for each band.



(a) Capex (b) Opex

Fig. 6. Capex and Opex for Low-band cells in cluster 1 with Greedy pooling for specific days. Values normalized to 24hr pooling for that group of days.



(a) Opex (b) Remaps

Fig. 7. Opex and #Remaps for Low-band cells in different clusters with Greedy pooling. Values are normalized to 24hr pooling for that cluster.

evaluate the pooling for specific days. Figure 6 shows the results of pooling on a weekday, a Sunday, and a national holiday (anonymized). Interestingly, ***fine-grained pooling is more beneficial, with respect to Capex, on Sunday and the national holiday***. This is because the traffic on these days has more variance, leading to greater benefits with more frequent pooling. However, this observation does *not* extend to Opex. On further inspection, we found that the Capex was high for coarse-granularity pooling because (under the highly variable non-weekday traffic) the peak load for different cells occurred at different times of the day, and so coarse-grained pooling had to provision enough capacity to handle this "sum of peaks". This behavior is similar to the concept we discussed in §II and Figure 4(b). We observed similar results in traces from a different week, which are not included in the paper for brevity.

Impact of specific locations: We also study the effects of geographical location on pooling. Figure 7 shows the results of pooling cells in different clusters. We see that the Suburban and Rural clusters benefit more, in terms of Opex savings, from coarse-grained pooling (up to 1min) than the Urban cluster. On further inspection, we find that the non-Urban clusters have more pronounced diurnal patterns in traffic, leading to more cost savings with even coarse-grained pooling. At very fine granularities, the Opex savings converge for Rural due to quantization to the closest number of vDUs, similar to mmWave in Figure 5(b). We also note the non-monotonic trend in #Remaps as a function of pooling granularity for the Rural cluster. This is because the Rural cluster has fewer cells and less traffic, leading to fewer vDUs and remaps, behaving similar to mmWave in Figure 5.

Impact of low-load conditions on pooling overheads with Greedy: In this section, we present our observations on the pooling overheads with Greedy algorithm during low-load conditions. Figure 8 shows the normalized number of remaps in different clusters on different days of week for Urban, Suburban and Rural locations. From the figure, we see that the #remaps don't vary significantly across different days except in the rural cluster on Sundays. This is due to the low-load conditions in the rural clusters on Sundays, where the required number of servers is small enough that the workload variability causes noticeable churn in the remapping of cells to the servers with the Greedy algorithm which tries to aggressively remap even for incremental improvements in OPEX. We note that the number of remaps reduces by 78% for this scenario with Affinity-based algorithm, while incurring a moderate trade-off on the Opex as we show in the next section.

B. Benefits of Affinity-Based Pooling

Thus far, we have employed the Greedy algorithm when packing cells to a vDU. However, this can result in high over-

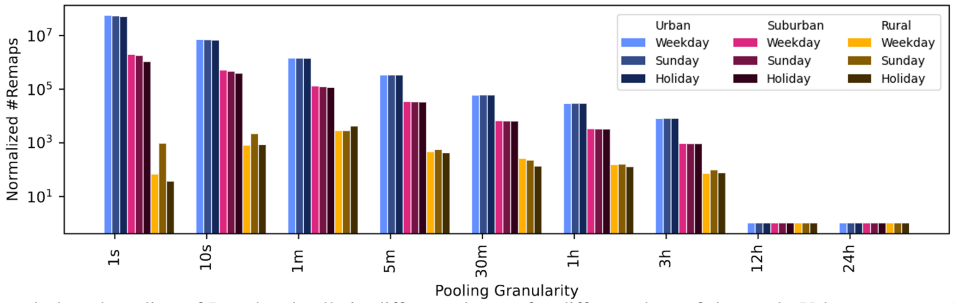
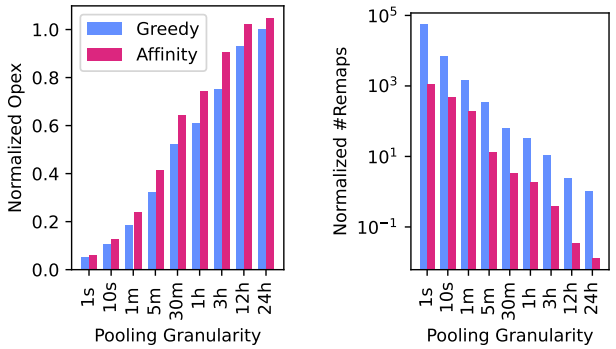
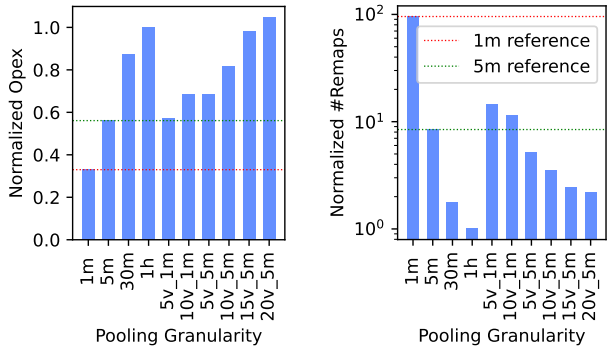


Fig. 8. #Remaps for Greedy based pooling of Low-band cells in different clusters for different days of the week. Values are normalized to 24hr pooling for the respective clusters.



(a) Opex (b) #Remaps

Fig. 9. Opex and #Remaps for Greedy and Affinity-Based pooling for Low-band cells in cluster 1. Values are normalized to 24hr with Greedy Pooling.



(a) Opex (b) Remaps

Fig. 10. Opex and #Remaps for static and dynamic pooling (using Affinity-Based) for Low-band cells in cluster 1. Normalized relative to 1hr pooling.

head, represented by the #Remaps. To reduce this overhead, we consider the Affinity-Based algorithm, presented in §II, which tries to retain the cell-to-vDU affinity across pooling intervals to minimize remappings.

Figure 9 shows the normalized results under Affinity-Based pooling and Greedy pooling for Low-band and cluster 1; results are qualitatively similar for other settings as well. We see that Greedy achieves about 4%–22% lower Opex compared to Affinity-Based pooling. This is due to Greedy's more aggressive pooling approach where it does not consider cells to vDU mappings in prior pooling intervals, leading to more flexibility in cell mapping to vDUs. However, this lower cost comes at the expense of a significantly higher overhead—***Greedy has 7×–76× higher #Remaps compared to Affinity-Based pooling***. We see a similar trend for other clusters as well. In Suburban cluster, Greedy has 5×–31× higher #Remaps compared to Affinity-Based pooling. In Rural cluster, Greedy has 2×–31× higher #Remaps compared to Affinity-Based pooling.

C. Benefits of Dynamic Pooling

We now evaluate the benefits of Dynamic pooling (discussed in §II) over the static/periodic pooling that we have considered thus far. Figure 10 shows the normalized results for fine-grained Affinity-Based static pooling and Dynamic pooling under various hyper-parameter configurations for Low-band and cluster 1; results are qualitatively similar for other settings as well. We denote the change point size and cooldown period values for Dynamic pooling policies on the x-axis tick values; for example, "20v_5m" refers to a load change threshold equivalent to the capacity of 20 vDUs and a cooldown period of 5 minutes. For reference, we show the performance of specific static pooling policies with horizontal dotted lines.

In general, ***Dynamic pooling is not always superior to static pooling***. For example, the Dynamic 20v_5m policy has about 86% higher Opex compared to the static 5min pooling granularity policy but 74% lower #Remaps. Similarly, the Dynamic 5v_5m policy has about 22% higher Opex compared to the static 5min pooling granularity policy but 40% lower #Remaps. We see similar tradeoffs for other Dynamic policies as well. In real-world scenarios, an operator can choose hyperparameters for Dynamic pooling based on their tolerance for overhead (proportional to #Remaps), which allows them to balance their cost savings versus overheads.

V. Related Work

Recent works have proposed techniques to improve vRAN performance and resiliency in shared environments. Concordia [24] develops a userspace deadline scheduling framework to improve RAN performance by reserving a minimum number of cores required for vRAN tasks. Nuberu [25] guarantees minimum resources for signals that preserve synchronization between the DU and its users. Agora [20] identifies parallelism in baseband processing and leverages multiple CPU cores to eliminate the need for specialized hardware like FPGAs for massive MIMO processing. Unlike these mechanisms that improve RAN efficiency within the compute, our work

explores improving overall compute utilization by pooling DU resources and exploiting the variability in traffic.

Atlas [12] develops a framework to redirect traffic between vDUs using existing mechanisms like handovers and cell reselection to improve vDU resiliency. Our work tackles the orthogonal problem of allocating workload across multiple vDU instances and can possibly leverage mechanisms proposed in Atlas to remap workload across pooled vDUs.

CloudIQ [26] proposed mechanisms for pooling base stations for shared processing in 5Mhz LTE networks, while providing a statistical guarantee for meeting real-time processing constraints. CloudIQ uses real-world data from 175 WCDMA cells at a granularity of 15 min to evaluate the potential savings by processing signals from multiple base stations. In contrast, our work focuses on DU pooling in a 5G network. Our study uses real-world data from 2910 cells across multiple carriers with significantly higher carrier bandwidths and data volumes. Unlike CloudIQ, we focus on the benefits of fine-grained pooling (upto 1s), while considering the tradeoff with the pooling overheads.

Bin packing algorithms have been widely used for virtual machine placement in cloud computing [27]. In the context of cellular networks, bin packing has been previously used for frequency allocation in cellular networks [28]. Unlike these works, our work focuses on bin packing cells onto vDUs while exploiting the temporal variations in the cell traffic to improve vRAN pooling efficiency.

VI. Conclusion

To the best of our knowledge, this is the first paper studying the gains in compute utilization with DU pooling in 5G networks. Using traces from a real-world 5G network and leveraging our interpolation based methodology, we show that significant Capex and Opex gains (84% and 94%, respectively) can be achieved through fine-grained pooling. Through our Affinity-based and Dynamic pooling approaches, we show that it is possible to achieve a 40% reduction in pooling overheads while only incurring 22% more the pooling benefits. In this paper, we focus on establishing the opportunity with DU pooling, studying the key dimensions that impact the benefits and overheads. Integrating our pooling approaches with a DU and addressing the system's challenges in implementation is ongoing and future work.

Acknowledgment

This work was supported by NSF grants CCF-2324859, CNS-2214980, CNS-2106434.

References

[1] "At&t switches on ericsson cloud ran on 5g commercial network," about.att.com/story/2024/ericsson-cloud-ran-technology.html, 2024.

[2] K. Schulz, "Verizon deploys more than 8,000 vran cell sites, rapidly marches towards goal of 20,000," www.verizon.com/about/news/verizon-deploys-more-8000-vran-cell-sites, 2022.

[3] "Dish wireless expands cloud-native open ran network with mavenir open vran software solutions," about.dish.com/2023-02-23-DISH-Wireless-Expands-Cloud-Native-Open-RAN-Network-With-Mavenir-Open-vRAN-Software-Solutions, 2023.

[4] D. Jones, "vran is catching on," www.fierce-network.com/wireless/vran-moves-contention, 2024.

[5] "Red hat openshift container platform," www.redhat.com/en/technologies/cloud-computing/openshift/container-platform, 2024.

[6] "Azure operator nexus," azure.microsoft.com/en-us/products/operator-nexus, 2024.

[7] "Ericsson cloud native infrastructure," www.ericsson.com/en/portfolio/networks/ericsson-cloud-ran/cloud-ran-infrastructure/cloud-infrastructure/cloud-native-infrastructure, 2024.

[8] "Telco cloud platform ran," telco.vmware.com/products/telco-cloud-platform-ran.html, 2024.

[9] "Dell poweredge xr5610," www.delltechnologies.com/asset/en-us/products/servers/technical-support/poweredge-xr5610-spec-sheet.pdf, 2023.

[10] "Hpe proliant dl110 gen10 plus telco server," www.hpe.com/psnow/doc/a50002566enw, 2024.

[11] R. T. Rodoshi *et al.*, "Resource management in cloud radio access network: Conventional and new approaches," *Sensors*, vol. 20, no. 9, 2020.

[12] J. Xing *et al.*, "Enabling resilience in virtualized rans with atlas," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–15.

[13] "3gpp ts 38.300 - 5g; nr and ng-ran overall description; stage 2," www.3gpp.org/ftp/Specs/archive/38_series/38.300/, 2022.

[14] A. Zelezniak *et al.*, "Baseband unit pooling using shared scheduler," US Patent US12 177 871B2, Dec., 2024.

[15] ——, "Sharing of baseband units in fifth generation networks and beyond," US Patent US20 230 276 354A1, Aug., 2023.

[16] G. Jung *et al.*, "Pooling of baseband units in fifth generation networks and beyond," US Patent US12 034 603B2, Jul., 2024.

[17] C. Mandal *et al.*, "Complexity of fragmentable object bin packing and an application," *Computers & Mathematics with Applications*, vol. 35, no. 11, pp. 91–97, 1998.

[18] E. G. Coffman *et al.*, "Approximation algorithms for bin-packing — an updated survey," 1984. [Online]. Available: api.semanticscholar.org/CorpusID:117984956

[19] "3gpp ts 28.552 - 5g; next generation radio access network (ng-ran); architecture description," www.3gpp.org/ftp/Specs/archive/28_series/28.552/, 2022.

[20] J. Ding *et al.*, "Agora: Real-time massive mimo baseband processing in software," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 232–244.

[21] H. Khedher *et al.*, "Processing time evaluation and prediction in cloud-ran," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[22] S. Khatibi *et al.*, "Modelling of computational resources for 5g ran," in *2018 European Conference on Networks and Communications (EuCNC)*, 2018, pp. 1–5.

[23] Celona, "5g bands explained: How they work & why they matter — celona.io," www.celona.io/5g-lan/5g-bands, [Accessed 15-05-2024].

[24] X. Foukas *et al.*, "Concordia: teaching the 5g vran to share compute," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 580–596.

[25] G. Garcia-Aviles *et al.*, "Nuberu: reliable ran virtualization in shared platforms," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 749–761.

[26] S. Bhaumik *et al.*, "Cloudiq: a framework for processing base stations in a data center," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, ser. Mobicom '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 125–136.

[27] K. S *et al.*, "Bin packing algorithms for virtual machine placement in cloud computing: A review," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, p. 512, 02 2019.

[28] N. Bansal *et al.*, "Bin-packing with fragile objects and frequency allocation in cellular networks," *Wireless Networks*, vol. 15, pp. 821–830, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:1937426