Traffic-Aware Compute Resource Tuning for Energy Efficient Cloud RANs

Ujjwal Pawar, Bheemarjuna Reddy Tamma, Antony Franklin A
Indian Institute of Technology Hyderabad, India
{cs19mtech01006,tbr,antony.franklin}@iith.ac.in

***Abstract*—Cloud Radio Access Network (C-RAN) disaggregates the functionalities of the base station in a way that some of the radio processing tasks are centralized in a virtualized computer pool of general-purpose processors (GPPs) on a cloud platform. This enables efficient utilization of the computational resources based on the spatio-temporal traffic fluctuations at cell sites. In this paper, we attempt to further reduce the computation resources by C-RAN on the cloud platform. First, we profiled the energy consumed in an OpenAirInterface (OAI) based C-RAN system using the existing Linux CPU frequency scaling governors. Based on the observations, we propose a traffic-aware compute resource tuning (CRT) scheme that reduces the energy consumption of C-RANs. The CRT scheme opportunistically lowers Modulation Coding Scheme (MCS) used while serving users by utilizing all of the available radio resources in every scheduling interval during non-peak hours. This reduction in the MCS helps in reducing energy consumption (due to usage of lower CPU clock frequency in the GPPs of the cloud platform) and fronthaul bandwidth requirements. Another benefit of the CRT scheme is its ability to work with any MAC scheduler. The extensive simulation results show how the CRT outperforms the existing frequency scaling governors in energy consumption while reducing fronthaul bandwidth requirements.**

***Index Terms*—Cloud-RAN, CPU frequency scaling, Energy Efficiency.**

I. Introduction

The rise in the number of mobile devices and their demand for high data rates pose serious challenges to mobile operators. Cisco [1] envisages that the number of mobile devices (including IoT devices) might reach 27.8 billion by 2023. More crucially, these devices come from different use cases requiring high bandwidth, low latency, or ultra-high reliable connectivity, all simultaneously sometimes.

To support the influx of billions of mobile-connected devices, mobile operators need to go for network densification by adding heterogeneous base stations for improving coverage and capacity. But, this puts a huge burden on the operators in terms of CAPEX and OPEX. One way to reduce the OPEX is by improving the energy efficiency of the base stations. Cloud Radio Access Network (C-RAN) architecture was proposed for reducing CAPEX and OPEX of next-generation cellular networks by employing disaggregation of base station functionalities and improving their energy efficiency. The general idea behind C-RAN is to isolate Remote Radio Heads (RRHs) from Base-Band Units (BBUs) of conventional base stations. Hundreds of RRHs can be associated with a single BBU pool realized on a central cloud platform using GPPs through high-speed fronthaul (usually fiber optic links) connections. This provides centralization of computing due to which high energy efficiency can be achieved. C-RAN is realized mainly by softwarization of RAN stack and deployed in the cloud infrastructure using virtualization technologies. As legacy cellular architectures heavily relied upon vendor-locked proprietary equipment, they offered very less programmability and therefore the scope to apply energy-efficient schemes for base-band processing was less. This resulted in most of the research being focused on reducing radio energy of RAN.

Due to the cloud-native architecture of C-RAN, the majority of its components would run on commodity x86 hardware. The migration from vendor-locked proprietary hardware to general-purpose hardware comes with various benefits that the next generation cellular network demands. Mobile operators can leverage the enhancements made in general-purpose processors for better performance and energy efficiency in C-RAN. Some of these enhancements include containerization, virtualization, CPU frequency scaling, real-time container scaling, and core pinning. Although running RAN components on general-purpose hardware on a cloud platform provides better compute sharing at the BBU pool, from our experiments, we found that there is still a lot of scope for improvements towards energy efficiency of C-RAN when running on x86 hardware.

In this paper, our objective is to improve the energy efficiency of C-RAN by adaptively tuning compute resources needed. The major contributions of this paper are as follows:

- The energy consumption of OAI based LTE base station running on x86 hardware is profiled. This profiling is done by varying the traffic load, CPU clock frequency, and MCS used. Results from this study provide an insight into C-RAN in terms of energy consumption under different channel conditions and traffic load. One of the key observations from energy profiling is that higher MCS requires more computation resources at the base station.
- Using the experimental results on the OAI testbed, we find that there is an optimal CPU clock frequency for each MCS at which energy consumption is minimal without degradation in the throughput of the connected users.
- We propose a novel traffic-aware computational resource tuning (CRT) scheme that opportunistically reduces the base-band processing energy by reducing the MCS and setting the CPU clock frequency to its optimal value for any MAC scheduler.

II. Related Work

The subject of energy efficiency in cellular networks is always a hot topic for both researchers and mobile operators. Methods to make 5G networks more energy-efficient by resource allocation are required as the energy efficiency emerged as one of the Key Performance Indicators (KPIs) of the 5G. Earlier, cellular networks were optimized to achieve maximum throughput, and resource allocation techniques were designed to achieve the same, but a paradigm shift began; new metrics like Joules per bit transfer is also seen as an important metric for resource allocation decisions.

Most of the previous research works focused on the radio transmit power's energy efficiency, where they mainly exploited the downlink power control in multi-cell coordination and multi-carrier multiple access schemes, such as Orthogonal Frequency Division Multiplexing (OFDM). In [2], a low-complexity energy-efficient resource allocation approach was formulated for the downlink OFDM/OFDMA system. In [3], user scheduling and power allocation were optimized across a cluster of coordinated base stations with a constraint on the maximum transmit power.

The C-RAN with its centralized architecture heavily relies on GPPs and benefits due to the savings in CAPEX/OPEX. In [4], the authors showed that the centralized architecture could potentially result in savings of at least 22% in computing resources by utilizing the variations in the processing load across cell sites. In this work, the authors presented that the LTE base station computation requirement is a function of the number of Physical Resource Blocks (PRBs) and MCS. In [5], the authors presented a mathematical analysis for subframe processing time given the PRBs and MCS. In [6], the authors showed the energy consumption and minimum CPU clock frequency required to run an LTE base station for different cell bandwidths. To further these efforts, in this work, we propose an energy-efficient traffic-aware compute resource tuning scheme which reduces the cost of base-band processing tasks and thereby reduces the fronthaul bandwidth requirements using the recent enhancements and features available in GPPs.

III. Motivation

This section describes results and analysis from our experimental study. The main aim for this experiment is to study the effect of MCS on CPU clock frequency, energy consumption, and user throughput. We report the threshold CPU clock frequencies required to run C-RAN for different MCS values.

A. Testbed description

To perform this experimental study, an OAI based LTE testbed is set up as shown in Fig. 1. OAI [7] is an open-source software implementation of 3GPP cellular network architectures with support for C-RAN. The RRH part of C-RAN is realized using an USRP B210 SDR board. Ubuntu 18.04 is utilized to run the BBU which has complete L1 and L2 components. The system running this BBU has Intel Xeon W series skylake based processor with 6 physical cores (12 logical cores), 140 Watt TDP, and maximum clock frequency of $3.6\ GHz$. For User Equipment (UE), LG Nexus 5, a COTS UE with 4G LTE support is used with a programmable SIM card. To vary PRBs and MCS used by the UE in a controlled manner, OAI FlexRAN controller is used as it provides an API to control the radio resources in real-time. To change the CPU clock frequency, *cpufreq* tool is used. To measure the energy consumed, *perf* tool combined with *turbostat* is used which reports processor topology, CPU clock frequency, idle power-state statistics, temperature, and power consumed on x86 processors. Table I shows various parameters used in the testbed setup.

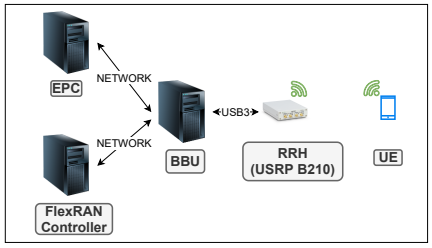

Fig. 1: OAI-based LTE Testbed Setup.

TABLE I: Parameters used in the Testbed Experimentation

Parameter	Value
System bandwidth	10 MHz (50 PRBs)
LTE Band	7
MCS Used	6, 13, 16, 21, 24 28
Duplexing Mode	FDD
Percentage of PRBs Used	25%, 50%, 75%, 100%
Traffic type	iperf3 UDP
Experiment Duration	120 seconds
Linux CPU Frequency Scaling Governor	Performance, Conservative

B. Results

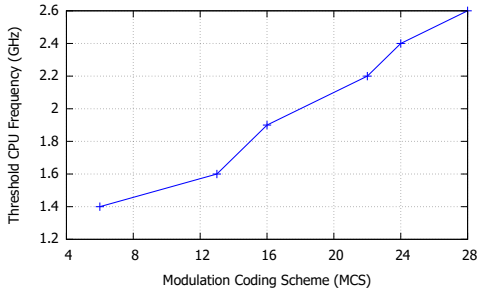

Fig. 2: Threshold CPU Clock Frequency vs MCS.

1) Threshold CPU Clock Frequencies: We define *threshold CPU clock frequency* as the frequency below which the radio packet processing task is not completed in the available time window by the BBU. This implies that the UE gets disconnected or that the errors crop up in the BBU code execution, which decreases the UE's throughput. Therefore, in our experiments, the CPU clock frequency is continuously reduced till the threshold point is reached for each MCS. This experiment is performed with both peak and reduced traffic

load from the UE. When running below the threshold values, errors are very prominent at the peak load and the respective UE gets disconnected immediately after a few Transmission Time Intervals (TTIs). For the reduced load, throughput is reduced in a few cases, and the errors are occurred in some TTIs. In this study, for a given MCS, we consider the same value as the *threshold CPU clock frequency* for both peak and reduced loads.

Fig. 2 shows *threshold CPU clock frequencies* for different MCS in case of 10 MHz (50 $PRBs$) channel bandwidth. It is clear from the plot that higher the MCS value, higher the CPU clock frequency needed for executing the BBU code.

2) Threshold CPU Clock Frequency vs. Linux CPU Frequency Scaling Governors: To understand the energy benefits of C-RAN operating at *threshold CPU clock frequency*, it is compared with C-RAN operating on default Linux scaling governors [8]. A C-RAN running at the threshold CPU clock frequency for MCS 28 is compared against the one using Linux frequency scaling governors. The Linux kernel supports CPU performance scaling through the *CPUFreq* subsystem, which provides various governors for scaling CPU clock frequency. Out of these governors, OAI suggests using *Performance* CPU scaling governor that, by definition, pushes the CPU to use the highest possible clock frequency. Another frequency scaling governor is the *Ondemand* governor. This dynamic governor allows the CPU to operate at the maximum clock frequency when the system load is high and at the minimum clock frequency when the system is idle. While this enables the system to adjust power consumption according to the system's load, it does so at the cost of latency between frequency switching. In LTE, the MAC scheduling takes place every 1 ms, therefore, the system might switch between idle and heavy workloads too frequently due to traffic fluctuations. When *Ondemand* governor is used, it also frequently switches between maximum and minimum CPU clock frequency. This frequent switching sometimes causes high latency due to which the UE gets disconnected, or data rate gets reduced. Hence, seeing this drawback, we decided not to include the *Ondemand* governor in our experimental studies. However, similar to the *Ondemand* governor, the *Conservative* governor adjusts the clock frequency according to the CPU load. While the *Ondemand* governor does so in a more aggressive manner, the *Conservative* governor will adjust to a clock frequency that it considers fit for the load, rather than solely choosing between maximum and minimum. In our experiments, the *Conservative* governor works perfectly without any errors.

Fig. 3 shows the average power consumption of C-RAN (i.e., BBU part) running on Linux frequency scaling governors and the *threshold CPU clock frequency* for MCS 28. *Performance* governor consumes the highest power at an average of 63 $Watts$ because it uses the highest available clock frequency of 3.6 GHz, whereas, *Conservative* governor performs better with an average power consumption of 43 $Watts$. Using the *threshold frequency* for MCS 28 i.e., 2.6 GHz, we get the least amount of power consumption at 41 $Watts$.

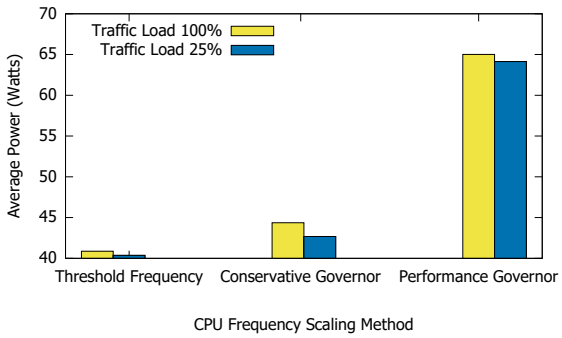

Fig. 3: Comparison of Average Power consumption by Static CPU Frequency and Linux Frequency Scaling Governors.

3) Power Requirement at Threshold CPU Clock Frequency: Experiments are conducted to measure the power consumption of BBU for different MCS by setting CPU at their respective *threshold frequencies*. To examine the effect of the number of PRBs used, the percentage of PRBs used is varied from 25% to 100%. To vary the percentage of PRBs in this experiment the FlexRAN API is used. As shown in Fig. 4, MCS 28 consumes the highest power at an average of 41 $Watts$, whereas MCS 6 consumes the least average power at 29 $Watts$. For all the MCSs, there is a slight increase in power consumed when increasing the percentage of PRBs used. This is because the CPU always runs at the fixed *threshold clock frequency* and sips almost the same power.

4) Effect of MCS Reduction on Energy and Throughput: An experiment is conducted to evaluate the energy gains by reducing the MCS when the system is not loaded i.e., non-saturated. In this experiment, one UE which has perfect channel conditions with a downlink traffic requirement of 10 $Mbps$ for 120 $seconds$ is considered. Fig. 5 shows the energy consumption for the *threshold clock frequency* and two Linux frequency scaling governors. By reducing the value of MCS to 13, all 50 $PRBs$ are utilized for serving the UE at 10 $Mbps$. The CPU frequency is tuned down to the *threshold clock frequency* for MCS 13 i.e., 1.7 GHz using the *Userspace* Governor. This results in the total energy

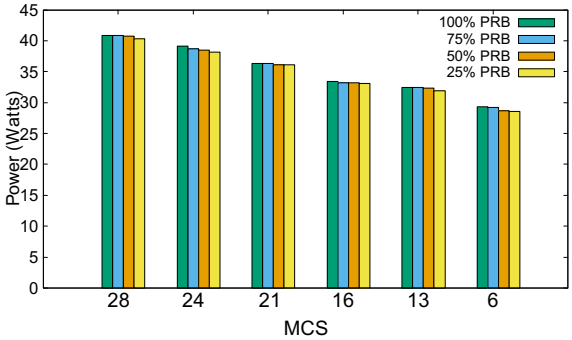

Fig. 4: Average Power Consumption for Different MCS with respective threshold clock frequencies.

consumption of 3776 $Joules$ which is 46% and 24% lesser compared to *Performance* and *Conservative* scaling governors, respectively. The UE does not experience any difference in terms of achieved throughput in all the three cases.

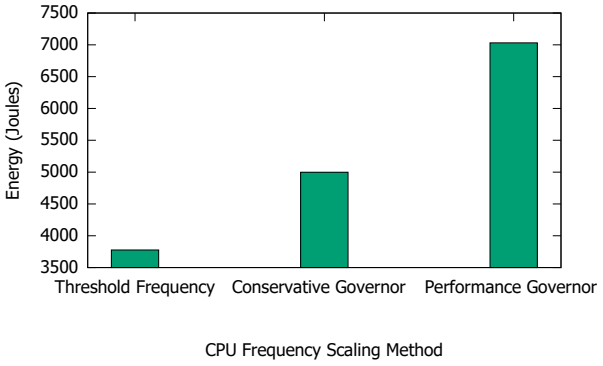

Fig. 5: Energy Consumption vs CPU Frequency Scaling.

C. Key observations from Experimental Results

From Figures. 2, 3, and 4, we can conclude that MCS is the major factor in selecting *threshold CPU clock frequency* of BBU in C-RAN. Even though the *Conservative* governor is a dynamic governor, it does not scale down the CPU clock frequency under reduced traffic load. Using the measurements from Fig. 4, the power consumption can be fitted as a function of CPU frequency and percentage of PRBs used.

$$POWER = \alpha * MCS + \beta * PRBs + \gamma \tag{1}$$

Where α, β, and γ are constants that can be calculated from offline profiling. γ is the static power consumed by C-RAN. Eqn. 1 can be generalised for users $U \ \epsilon \ \{1, 2, 3 \dots n\}$.

$$POWER = \sum_{i=1}^{n} (\alpha * MCS_i + \beta * PRBs_i) + c \tag{2}$$

Where MCS_i is the MCS value and $PRBs_i$ is the percentage of PRBs used by user i who is scheduled in a TTI by the MAC scheduler. From Fig. 2, it can be concluded that CPU frequency setting at the BBU is a function of MCS.

$$CPUfreq \propto MCS \tag{3}$$

This means if multiple users exist in a scheduling interval or TTI, the clock frequency that needs to be set is decided based upon the highest MCS present. From the above equations, to minimize energy in every scheduling interval, the value of highest MCS used needs to be minimized.

$$\begin{aligned} \tau &: min \ POWER \\ &: min \ \sum_{i=1}^{n} (\alpha * MCS_i + \beta * PRBs_i) + c \\ &: min \ \{MAX_{i \epsilon U}(MCS_i)\} \end{aligned} \tag{4}$$

$$s.t. \ \sum_{i=1}^{n} PRBs_i \leq PRBs_{max} \tag{5}$$

Here, $PRBs_{max}$ is the maximum number of PRBs available for a given bandwidth. Although reducing the MCS consumes more PRBs, the CPU clock frequency is reduced and set to the *threshold CPU clock frequency* to ensure energy reduction. Similar energy reduction is possible even when the system is running different functional split architectures of C-RAN.

A significant side benefit of reducing MCS is that it can help in reducing the fronthaul bandwidth requirements if used with modulation fronthaul compression. Supported by O-RAN alliance, the modulation compression technique [9] is a type of lossless compression technique that can be applied to I/Q data symbols prior to encoding them into several bits of length W called bit-width. Given some modulation point to code, this strategy seeks to minimise the bit-width in order to reduce the fronthaul bandwidth requirements. If no compression is used, then bit-width is generally fixed at $W = 32$ bits for both I and Q samples, regardless of the modulation order used. However, by using modulation compression, the bit-width can be dynamically selected according to highest modulation order used in the scheduling interval, thus reducing the bandwidth requirements over the fronthaul link between BBU and RRH.

IV. Proposed Compute Resource Tuning Algorithm

In this section, based on the key observations made from the motivational results shown in the previous section, we propose a novel greedy algorithm (Algorithm 1) called Compute Resource Tuning (CRT). It is intended to work with any MAC scheduling scheme and does not hinder any scheduling decision regarding user selection or system throughput. Besides, it does not modify any power-saving measures such as discontinuous transmission (DRX). From the experimental study and related work, it is well known that transmitting for users with a higher MCS requires higher CPU clock frequency to finish the radio packet processing in time. Higher MCS does

Algorithm 1: CRT: Compute Resource Tuning Algorithm

```
Get scheduling decision from MAC scheduler for the
  next TTI
U_list = list of users scheduled in the next TTI
while all PRBs are not utilized and U_list is not empty
  do
    u = select user with highest MCS from U_list
    u.MCS = u.MCS − 1
    if remaining PRBs are enough to schedule u with
      the new u.MCS in the next TTI then
      update the remaining PRBs
    else
      u.MCS= u.MCS + 1 and
      exit the while loop
    end
end
Set Threshold CPU clock frequency for the highest
  available MCS in the next TTI using the
  Userspace Governor.
```

offer more bits per symbol, thereby increasing user throughput. However, from various studies on traffic variations in cellular networks, it is well established that each cell does not face peak traffic all the time. The CRT algorithm considers this fact and tries to minimize the highest MCS used by consuming all the available PRBs in each scheduling interval or TTI. If all the PRBs are not utilized at the end of scheduling decision by the MAC scheduler, CRT algorithm kicks in and it opportunistically selects the user with the highest MCS from the list of scheduled users and reduces its MCS value until all the PRBs are allocated. As a result of this procedure, the CPU frequency of BBU in C-RAN could be scaled down to reduce the energy consumed by the C-RAN system, without any impact on throughput and delay of the scheduled users.

A. Time Complexity Analysis

The algorithm iterates over a list of scheduled users. Each user can be selected maximum 28 times for MCS reduction. Selecting a user with highest MCS will take $O(log\ u)$ time if heap is used to store the scheduled users. Thus, the time complexity of this algorithm is $O(28{*}log\ u{*}u)$ or $O(u\ log\ u)$, where u is the number of scheduled users in a TTI. Moreover, the CRT algorithm considers the users selected by the scheduler for the next scheduling interval and not the total number of users currently served by the cell. This keeps the value u low due to which the overall run time of the MAC scheduler does not change much.

V. Simulation Environment and Parameters

The proposed CRT algorithm is implemented on the LTE module available in NS-3 simulator [10] which provides various MAC schedulers. NS-3 implements the Femto Forum MAC Scheduler Interface [FFAPI] as a set of C++ abstract classes; in particular, each primitive is translated to a C++ method of a given class. CRT algorithm is implemented on top of existing NS-3 MAC scheduler class. Furthermore, the existing NS-3 LTE scheduler is modified to report MCS and PRBs used in each TTI. These values are then used to calculate the energy consumed in offline using the results shown in Section III. In this paper, we considered NS-3's Proportional Fair (PF) as the MAC scheduler. The simulation parameters are given in Table II. As the algorithm focuses on downlink scheduling, no uplink transmissions were performed. To reduce the complexity of the simulation, HARQ is disabled. To change the load on the eNodeB, the number of UEs is varied from 1 to 30. Each UE receives a packet of random size between 1000 and 1400 bytes with a packet inter-arrival interval of 10 ms. This generates on average 1 $Mbps$ traffic per UE from the eNodeB.

TABLE II: Simulation Parameters

Parameter	Value
Channel Bandwidth	10 MHz
Transmission Time Interval	1 ms
Number of eNBs	1
eNB's Transmit Power	30 dBm
Number of UEs	1 to 30 (variable)
UE's Transmit Power	10 dBm
Packet Size	Random between 1000 and 1400 bytes
Interarrival Rate	10 ms
Traffic Type	UDP
User Mobility	Random2dWalk at 20km/h
Simulation Time	20 seconds

VI. Simulation Results

A. Distribution of MCS Used in eNodeB

The CRT algorithm attempts to reduce the MCS used in every TTI during non peak-hours. Fig. 6 shows how the algorithm performs for 5 $users$. The users are served using MCS 28 for 60% of the TTIs while running PF as the MAC scheduler. But, users are served with MCS 14 for 60% of the TTIs when CRT is applied on PF's schedule. This is because enough traffic is not available to consume all the available PRBs in each TTI with only 5 $users$ and therefore CRT algorithm is able to reduce the MCS.

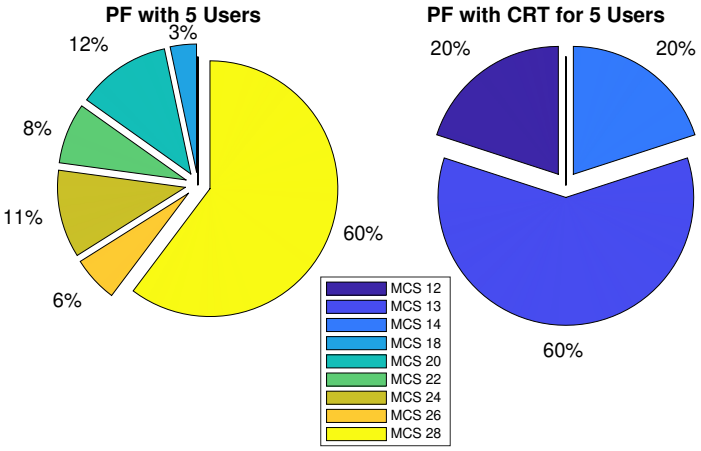

Fig. 6: Distribution of MCS selected with and without CRT

B. Reduction in Energy Consumption

To assess the CRT algorithm's performance on the energy front, it is compared with NS-3's PF scheduler with Linux scaling governors as shown in Fig. 7. It is also compared when the PF runs with threshold frequencies. The results are shown with a confidence interval of 95%. To calculate the energy consumption for each scenario, first, scheduling information for each TTI is taken from NS-3 simulations which is then used to calculate energy consumption using Eqn. 2 and its constants are calculated using profiling results from Section III. It is shown that the PF with *Performance* governor consumes the highest energy because it always runs the system at the peak available frequency i.e., 3.6 GHz. *Conservative* governor performs better than *Performance* governor as it reduces the CPU frequency depending on the load. PF running on threshold frequencies consumes less energy than when PF is used with the two scaling governors. In all the above scenarios, the user scheduling was identical. The difference was seen only in the energy consumed due to the

variation in CPU clock frequency of the system in different cases. PF with CRT algorithm consumes the least amount of energy amongst all. This is because it reduces MCS used and consumes all the PRBs when the traffic load is low. Only at the peak traffic does its energy meet the energy consumption of NS-3's PF running at threshold frequencies. At peak traffic (i.e., 30 $users$) CRT algorithm is able to reduce the energy consumption by 20% and at low traffic (i.e., 1 $user$) by 40% when compared with the *Conservative* governor. On average 25% reduction in energy is observed.

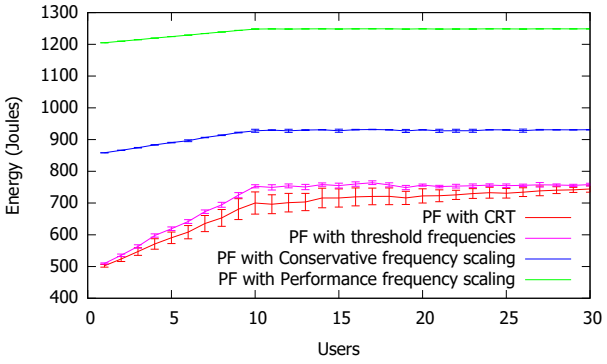

Fig. 7: Energy Consumption by Different Variants of PF scheduler.

C. Fronthaul Capacity Reduction with Fronthaul Modulation Compression

The CRT algorithm complements the modulation compression technique because it reduces the MCS value thereby reducing the modulation order. In essence, if the CRT algorithm is used along with modulation compression, the required fronthaul capacity will be reduced. Fig. 8 gives the required fronthaul capacity for PF with and without CRT. O-RAN supports split option 7.2 for fronthaul and its capacity can be calculated using Eqn. 6.

$$FH = N_{SC} * N_{symb} * W * N_L * 1000 + MACinfo \quad (6)$$

Where N_{sc} is the number of subcarrier/PRBs, N_{symb} is the number of symbols in 1 ms, W is bit-width (number of IQ bits), and N_L is the number of layers. For our simulations, N_{sc} is 12, N_{symb} is $14 \times 50 = 700$ symbols (for 50 $PRBs$) and N_L as 1. The value of W depends on the modulation order i.e., on the MCS used. MCS values used in every TTI are taken from the NS-3 simulation. Fig. 8 shows the average fronthaul bandwidth used. CRT algorithm under low traffic can reduce the fronthaul requirement as it reduces the maximum modulation order used. On an average the fronthaul requirement is reduced by 80% when modulation fronthaul compression is used compared to no compression. This is another saving that can be gained by using the proposed CRT algorithm with any MAC scheduler in C-RAN.

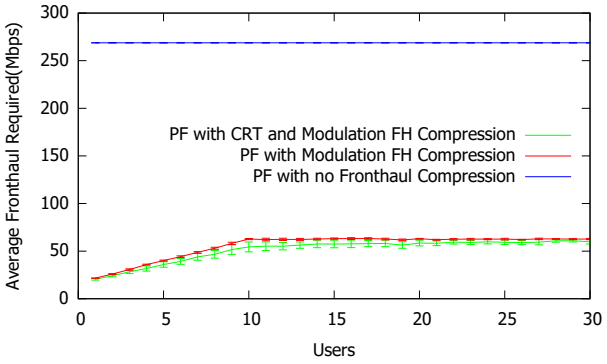

Fig. 8: Average Fronthaul Requirement by Different Techniques

VII. Conclusions

In this paper, we profiled the energy consumption in C-RANs using a small scale testbed and determined the limitations of Linux frequency governors. We identified that there is a threshold CPU clock frequency for each MCS using which energy consumption of baseband processing in the C-RAN system can be reduced. Taking the advantage of frequency scaling, we proposed a novel greedy-based CRT algorithm which can work with any MAC scheduler. Additionally, the CRT helps in reducing the fronthaul bandwidth requirements if used with modulation fronthaul compression technique. The insights drawn in this work could greatly help mobile operators in reducing their OPEX by adaptively tuning various system resources available in the BBU pool.

Acknowledgment

This work has been supported by the project "CCRAN: Energy Efficiency in Converged Cloud Radio Next Generation Access Network" funded by Intel India.

References

[1] Cisco. Annual internet report (2018–2023) white paper.
[2] F. Héliot et al. Energy-efficient resource allocation for orthogonal multi-antenna multi-carrier channel. In *IEEE Online Conference on Green Communications*, 2013.
[3] L. Venturino et al. Energy-efficient scheduling and power allocation in downlink ofdma networks with base station coordination. *IEEE Transactions on Wireless Communications*, 2015.
[4] Sourjya Bhaumik, Shoban Preeth Chandrabose, Manjunath Kashyap Jataprolu, Gautam Kumar, Anand Muralidhar, Paul Polakos, Vikram Srinivasan, and Thomas Woo. Cloudiq: A framework for processing base stations in a data center. ACM Mobicom, 2012.
[5] T. X. Tran et al. Understanding the computational requirements of virtualized baseband units using a programmable cloud radio access network testbed. In *IEEE International Conference on Autonomic Computing*, 2017.
[6] U. Pawar, A. K. Singh, K. Malde, B. R. Tamma, and A. Antony Franklin. Understanding energy consumption of cloud radio access networks: an experimental study. In *IEEE 3rd 5G World Forum Workshop*, 2020.
[7] EURECOM. Openairinterface url:www.openairinterface.org/.
[8] Dominik Brodowski et. al. Cpu frequency and voltage scaling code in the linux(tm) kernel url: www.kernel.org/doc/documentation/cpu-freq/governors.txt.
[9] S. Lagén, L. Giupponi, A. Hansson, and X. Gelabert. Modulation compression in next generation ran: Air interface and fronthaul tradeoffs. *IEEE Communications Magazine*, 2021.
[10] Network simulator url:https://www.nsnam.org.