Cloud RAN Challenges and Solutions

Rajeev Agrawal, Anand Bedekar, Troels Kolding, and Vishnu Ram
Nokia Networks

***Abstract*—In this paper we take an overall look at key technical challenges in the evolution of the Radio Access Network (RAN) architecture towards Cloud RAN, and solutions to overcome them. To address fronthaul limitations, we examine the implications and tradeoffs enabled by functional splits on fronthaul needs, system performance, and centralization scale. We examine the architecture of algorithms for multi-cell coordination and implications in a Cloud RAN environment. To maximize the use of General-Purpose Processors (GPP) and operating systems such as Linux for Cloud RAN, we propose methods of achieving real-time performance suitable for RAN functions. To enable right-sizing the amount of compute used for various RAN functions based on the workload, we propose methods of pooling and elastic scaling for RAN functions that exploit the fact that certain RAN functions perform per-user operations while others perform per-cell operations. Cloud RAN also aims to use cloud management technologies such as virtualized infrastructure management (VIM) and orchestration for automating the instantiation and scaling of RAN functions. We identify special needs for RAN arising from real-time constraints and a mix of GPP and non-GPP hardware.**

Keywords—Cloud RAN, fronthaul, real-time, pooling, elastic scaling, multi-cell coordination, orchestration, NFV, 5G.

I. Introduction

As mobile data traffic demand rapidly grows [1], it is expected that mobile networks will have to add significant amounts of spectrum for the radio access network, as well as significantly increase the density of cells either by cell-splitting at the macro layer or by adding underlay small cells. Due to this, the amount of baseband processing needed will significantly increase, and further, there will be significantly higher levels of interference in the network due to the high density of cells. Cloud Radio Access Network (Cloud RAN)[2][3], in our terminology consisting of both centralization of processing resources for the RAN and use of cloud technologies, is a deployment paradigm that aims to achieve a significantly lower cost-per-bit in such a scenario. In conventional RAN deployments, baseband functions are typically distributed at the cell sites, whereas in Cloud RAN, all or portions of the baseband would be centralized, thereby reducing operating costs of cell sites by simpler site solutions and fewer site visits for maintenance and upgrades. Cloud RAN aims to maximize the use of general purpose processors (GPPs) such as Intel x86 and ARM for RAN functions, thereby simplifying procurement, enabling reuse of hardware across RAN and other network functions, and leveraging economies of scale due to GPP use in the IT industry. Similar to cloud technologies used in datacenters in the IT industry, Cloud RAN aims to maximize scalability and minimize over-provisioning of processing resources by right-sizing the amount of processing used to the workload. Cloud RAN aims to increase automation in the deployment and lifecycle management of RAN functions by leveraging datacenter cloud management technologies like virtualized infrastructure management (VIM) and orchestration. To combat high levels of interference, Cloud RAN aims to enable better performance of multi-cell coordination algorithms than conventional RAN deployments. In this paper, we investigate challenges for these objectives of Cloud RAN, and identify solution approaches.

When some or all portions of the baseband are centralized, the network interconnecting the central portion to the cell sites is termed as fronthaul. The ideal transport medium for today's fronthaul is dark point-to-point fiber, but this is not widely available. More widely available transport such as Metro Ethernet may be constrained in bandwidth or may provide higher latency or jitter. To address fronthaul limitations, in Section II, we examine the implications and tradeoffs enabled by functional splits on fronthaul needs, system performance, and centralization scale. In Section III, we examine the architecture of multi-cell coordination algorithms and implications in a Cloud RAN environment. We make the case that multi-cell coordination algorithms should be designed to have a decentralized algorithm architecture and use a flexible coordination cluster design, so as to maximize the flexibility of placing RAN functions within the RAN cloud as well as the benefit from multi-cell coordination. A key challenge in the use of GPP/Linux environments for RAN functions is that RAN functions have real-time constraints at the millisecond level. In Section IV, we propose methods of achieving such real-time performance. To right-size the amount of compute used for various RAN functions based on the workload, we propose methods of pooling, elasticity, and load-balancing for RAN functions in Section V, exploiting the fact that certain RAN functions perform per-user operations while others perform per-cell operations. For the application of cloud management technologies such as VIM and orchestration for RAN functions, we identify special needs for RAN arising from real-time needs and supporting a mix of GPP and non-GPP hardware in Section VI that should be taken into account as these technologies are adapted for RAN in forums such as ETSI NFV. While our focus is on LTE, we also consider implications of 5G on various aspects of Cloud RAN.

II. Fronthaul and Functional Splits

In traditional Centralized RAN deployments, conventional baseband units are housed in a centralized location (a so-called "BTS hotel") and interconnected to the radio heads at the cell sites with fiber, typically dark point-to-point fiber. The protocol used over the fiber is typically CPRI [6] or OBSAI RP3 [7]. These protocols have very stringent latency and jitter requirements, typically less than 200us, and result in outage if the requirements are not met. Further, the dark fiber needed to

ensure adequate bandwidth and low latency/jitter for CPRI or OBSAI is not widely available. As 5G calls for sub-ms RTT at air interface and use of massive MIMO, the latency and bandwidth scalability challenge becomes even more pronounced. More widely available transport such as Metro Ethernet is typically packet-switched transport, however, and thus typically has higher jitter than what CPRI or OBSAI can tolerate, and further may impose bandwidth constraints as well.

To make efficient use of more widely available transport options, one approach is to not centralize all the baseband functions, but centralize a subset leaving the rest at the cell sites. A range of such *functional splits* are possible, each of which presents different needs on bandwidth and latency and tolerance to jitter on the fronthaul. Figure 1 illustrates a family of such functional splits. Bandwidth needs of various functional splits have been analyzed in [8]. Our focus here is to present some key insights and tradeoffs in functional splits.

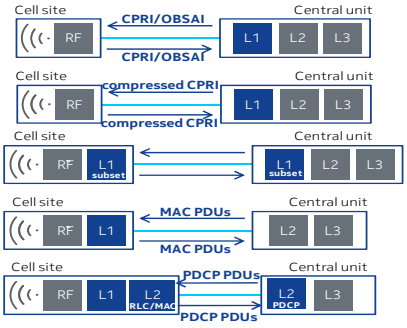

Figure 1. Functional Splits.

We observe that in LTE, the HARQ loop and limit on the number of HARQ processes impose constraints on the fronthaul and processing latency for functional splits where the HARQ loop is terminated in the central site. This is illustrated in Figure 2 for FDD-LTE for the functional split where all baseband functions are centralized. A similar constraint also applies to TDD-LTE. For an ack/nack transmission sent by the user equipment (UE) in TTI n, the RAN's radio Scheduler must process the ack/nack and grant a fresh transmission or retransmission to be transmitted over the air in TTI n+4 to ensure peak throughput. If the RAN cannot meet this, e.g. because of latency or jitter in the fronthaul or in processing, TTIs have to be skipped, leading to a loss of air interface throughput as quantified in [9].

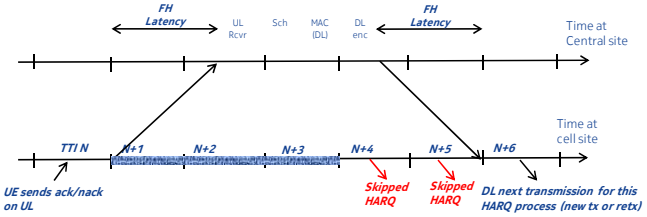

Figure 2. LTE-FDD HARQ constraint on latency.

This has several ramifications. First, the total latency plus jitter of the fronthaul must be such as to allow finishing the HARQ-related baseband processing within 3ms (i.e. from TTI n+1 to TTI n+3) to avoid a loss in air interface throughput. In this sense, jitter on the fronthaul is like additional latency that must be accounted for in the time budget. Second, we observe that this bounds the distance between the cell site and central location, limiting the scale of centralization. Centralization scale is typically tied to the ability to extract pooling gains and other cloud benefits. For example, if the baseband processing consumes 2.5ms, that leaves 0.5ms for the round-trip fronthaul latency plus jitter, which constrains the distance for light propagation in optic fiber to a maximum of around 50km. In a real network fiber is not typically laid along geodesic lines from the cell sites to the central site, so the limitation on geographic distance between the cell sites and central site in a real network may be even worse. Large-scale centralization is thus not realistic for such functional splits – a Cloud RAN deployment for such splits may thus consist of multiple Cloud RAN hosting locations distributed fairly close to the cell sites.

In contrast, for functional splits where the HARQ loop is terminated at the cell site (e.g. where only non-real-time parts of Layer-2 (PDC) and Layer-3 (RRC) are centralized), it is possible to achieve larger-scale centralization; however since a significant portion of baseband functionality is left at the cell sites in these splits, the potential benefits of Cloud RAN such as lower OpEx and higher pooling gains would be reduced as well. Thus in choosing the functional split for their network deployment, operators are faced with a multi-dimensional tradeoff between conflicting objectives of reducing the bandwidth needs by leaving more functionality at the cell site, relaxing the tolerable latency and jitter, achieving greater operational benefits by greater scale of centralization, maximizing air interface performance, and cost/complexity.

An example illustration of such a tradeoff is shown in Figure 3 with some qualitative scoring of current Distributed RAN and CPRI-based Centralized RAN architectures. Hybrid functional-split-based architectures seek to achieve a better overall tradeoff weighing pros and cons within specific operator's conditions. A methodology to evaluate this type of tradeoff has been proposed in [15].

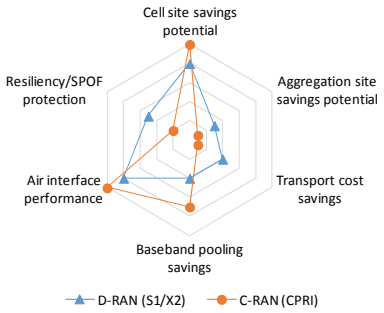

Figure 3. Multi-dimensional tradeoff for functional splits.

In dense metro areas where the density of cells is high, there may be hundreds of cells within a few kilometers of cloud RAN centralization sites such as central offices, so significant scale of centralization may be obtained even using central sites that are near the cell sites. Further, the availability of low-latency fiber is typically also easier in dense metros. Thus the HARQ loop can be more easily centralized, and such areas may be sweet spots for maximal Cloud RAN benefits.

In 5G, it is anticipated that TTI duration will be even shorter than the LTE TTI of 1ms, while the carrier bandwidths are expected to be significantly larger than the 20MHz in LTE. This means that the processing execution time may increase, while in principle the total time budget available for fronthaul and processing will shrink compared to LTE, which may constrain the scale of centralization even further in 5G unless the 5G air interface designs a more flexible HARQ mechanism.

We also observe that to maximize the latency+jitter budget available for the fronthaul, the latency+jitter consumed by the baseband processing execution should be minimized. We will show in Section IV that executing some of the baseband real-time functions on General Purpose Processor and Linux environments can potentially introduce significant jitter in the execution time, and provide ways to minimize that.

III. Multi-Cell Coordination

To deal with the growth in traffic demand, mobile wireless networks are expected to densify by increased cell splitting as well as addition of small cell underlays. As the cell sites get closer together, interference in the network will significantly increase. Further, the additional sites are likely to be more poorly planned due to the lack of optimal site placement options, as well as the effort required to optimize site parameters such as antenna orientation and downtilt. Such sub-optimal planning will add to the level of interference. To combat interference, various forms of multi-cell coordination are expected to be increasingly used. These may operate at a slow time-scale, e.g. Enhanced Inter-Cell Interference Coordination (EICIC) for heterogeneous networks, or at a fast time-scale, e.g. Coordinated Scheduling and Dynamic Point Selection for downlink, and Joint Reception for uplink [10]. LTE standards have been adding support for fast-time-scale Coordinated Multi-Point (CoMP) transmission and reception, such as richer channel state information feedback from user equipment. Performance analysis of multi-cell coordination schemes has shown that the latency of the information exchange between cells and the bandwidth available for coordination are key aspects which impact the multi-cell coordination performance [11][12].

In Cloud RAN, depending on the functional split, the layer at which coordination happens (i.e. Layer 1 (Physical Layer) or Layer 2 (MAC Scheduler)) may be in the central site or at cell sites, depending on the functional split, and correspondingly the multi-cell coordination may operate within the centralized site or in a distributed manner among the cell sites.

We observe that even when the relevant parts of the baseband involved in multi-cell coordination (e.g. Physical Layer or MAC/Scheduler) are centralized, coordination between cells can be accomplished in a decentralized manner within the central complex. That is, instead of bringing all the relevant information (e.g. users' channel conditions relative to the different cells) together into a common central processor, it is possible to achieve coordination in a decentralized manner by exchanging appropriately chosen metrics among coordinating cells. The baseband of the coordinating cells may be in a centralized location, but even within the centralized complex, the *algorithm architecture* can still be decentralized. We propose the following framework for decentralized algorithm architecture. The framework is illustrated in Figure 4 for the case where coordination happens at layer-2 Scheduler, e.g. for Coordinated Scheduling or Dynamic Point Selection.

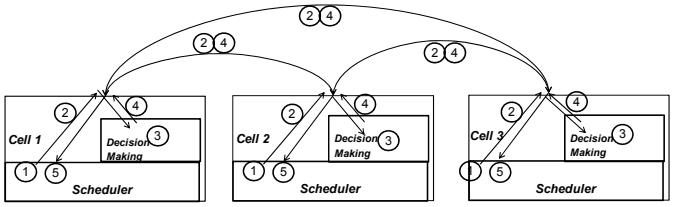

Figure 4. Decentralized algorithm architecture for multi-cell coordination.

In this example of the proposed framework for Scheduler-based coordination, each cell has its own Scheduler, which is instantiated at the cell site in the case of a distributed RAN and as a separate entity (e.g. a virtual machine) in the case of Cloud RAN. In Step 1, the Scheduler derives certain metrics which provide a concise representation of the operation of the Scheduler within that cell, with possibly separate metrics derived by the cell for each coordinating neighbor. For example, for Dynamic Point Selection with a Proportionally Fair (PF) Scheduler [13], the metric may be the maximum or average of the per-user PF metrics of the users within the cell, or in the case of Coordinated Scheduling, it may be a Benefit Metric quantifying the benefit to the cell if a given neighbor were to mute [14]. In Step 2, the Scheduler of each cell exchanges this metric with an appropriate set of neighbor cells. Each cell has its own local Decision-making Function, which in Step 3 uses its local information about its own users together with the metrics received from neighbors in Step 2 to make a decision on its coordination action (e.g. whether to mute on certain resources for Coordinated Scheduling, or whether to transfer a user to another cell for Dynamic Point Selection, etc). In Step 4 each cell communicates its decision to the relevant neighbors. In Step 5, each cell then performs its local scheduling to allocate its resources among the users that are to be transmitted from that cell. This decentralized algorithm architecture enables a very scalable framework that gracefully adapts to increasing density of cells, and works the same way regardless of whether the coordination is within the central site or distributed among the cell sites.

One key to obtaining efficient performance from multi-cell coordination is that each cell should be able to coordinate with the *right set of neighbors*. From a given cell's perspective, it should be able to coordinate with all cells from which it receives (or towards which it causes) significant interference. In contrast, a design that includes only a subset of a cell's significant interferers within the coordination cluster will be suboptimal. Liquid Clusters are an efficient way of structuring coordination cluster relationships to capture this notion. As illustrated in Figure 5, each cell determines its appropriate set of neighbors with whom coordination would be beneficial – neighboring cells may determine their own (overlapping, but non-identical) Liquid Clusters. Liquid Clusters are thus well suited to the decentralized coordination algorithm architecture.

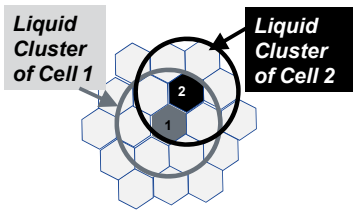

Figure 5. Liquid Clusters for multi-cell coordination.

In the context of Cloud RAN, this decentralized algorithm architecture enables flexible placement of the coordinating cells' baseband functions (e.g. Schedulers) at any suitable location within the cloud where processing resources are available. For example, it does not require that the Schedulers of coordinating cells be instantiated on the same physical server. However, as noted earlier, the latency or bandwidth of coordination impacts the achievable gain. In constructing its Liquid Cluster, a cell may choose to exclude neighbor cells which may be significant from an air interface perspective, but due to poor latency or bandwidth available for coordination, would not provide enough coordination benefit. So we propose that the Cloud RAN should have an optimal placement algorithm that decides the mapping or placement of baseband functions of the various cells into the processors in the Cloud infrastructure so as to maximize the coordination gain, taking into account the topology and characteristics (e.g. latency, bandwidth) of the network interconnecting the processors and the potential implications on Liquid Clusters of the cells.

For functional splits where the coordination happens within the central site, the performance will be impacted by the latency of the fronthaul, whereas for other functional splits where the same type of coordination is done in a distributed manner among the cell sites, the performance will be impacted by latency of the backhaul network interconnecting the sites. We note here a key insight from [15] that the performance of a multi-cell coordination technique in the centralized case, for a given latency of fronthaul, will be no better than its performance in the distributed case, if the backhaul latency is the same as the fronthaul latency in the centralized case.

IV. Real-Time RAN Functions on GPPs/Linux

Conventionally, baseband processing has been implemented using special purpose processors such as Digital Signal Processors (DSPs) and Systems-on-Chip (SOCs), using real-time operating systems, due to the tight real-time requirements of many baseband functions of 1ms or less. In contrast, one of the key drivers for Cloud RAN is to maximize the use of General-Purpose Processors (such as those based on the Intel x86 Architecture or ARM), using general-purpose operating systems like Linux. These GPP platforms have traditionally been used for non-real-time functions, and are more naturally applicable for baseband functionality without tight real-time constraints e.g. PDCP or RRC. For some functional splits the real-time portions of the baseband are also desired to be centralized, and the question of the suitability of GPP/Linux to these functions arises. For Layer-1, functions such as turbo coding/decoding are quite heavy and can benefit significantly from special instructions or accelerators in terms of cost, power consumption, and form factor. Further, in the march to 5G, radio standards will likely continue to evolve at the edge of capability of dedicated hardware leaving a gap between GPP capability and the most critical requirements for the physical layer. Due to this, specialized hardware may continue to remain suitable for Layer-1. However, for other real-time parts of the baseband which are compute heavy, such as the RLC/MAC and the Scheduler, GPP/Linux is a realistic possibility, and we explore this in the following.

The Scheduler and MAC have to execute a certain set of actions in every TTI (1ms in LTE, and expected to be even smaller in 5G). The operating environment has to be able to initiate the tasks every TTI with very low latency, and ensure completion of the execution with very low variability (execution jitter). The Linux kernel typically does not ensure either of these. Special low-latency versions and real-time patches of the kernel are possible, but they typically compromise throughput performance to ensure real-time response, and also violate the desire to have generic environments for RAN and other functions.

Instead of using linux kernel interrupts to wake up tasks every TTI, an alternative paradigm is to use a "run-to-completion" execution model. Tasks (e.g. Scheduler or MAC operation for a given TTI) wait in queues, and a "dispatcher" is prioritizes the next task to be executed and provides it to one of multiple software instances. For low-latency dispatching, software instances poll the dispatcher for tasks, rather than using linux kernel interrupts. The software instance executes the provided task, and on finishing ("run-to-completion") goes back to check for the next task. A benefit of such a model is the automatic load-balancing of tasks across multiple processor cores. Open Event Machine [15] is an open-source framework for a user-space run-to-completion model, and is a suitable basis for real-time RAN baseband functions.

The linux kernel, however, does not necessarily guarantee uninterrupted access of any particular software instance to the underlying CPU core. This can cause significant variability in the execution time. To understand the performance challenges that a real-time application such as Scheduler or MAC faces, we consider a simple CPU-bound application that iteratively repeats some set of operations (e.g. arithmetic operations), and examine its *execution jitter* i.e. variability in the execution time of an iteration of the operation. Ideally the execution time of an iteration should be constant, but in real-world scenarios, significant variability is observed, as shown in Figure 6.

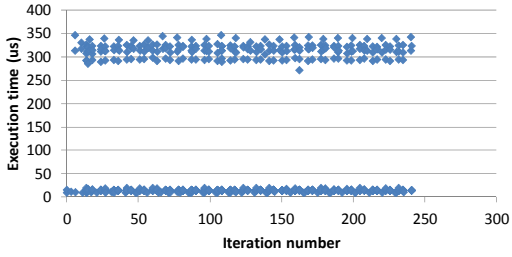

Figure 6. Variability in execution time on Linux.

Figure 6 shows observations on an Intel Xeon E5-2695v2 running Linux with kernel version 3.13.04-20-generic. Typical

execution time of an iteration is around 18us. However, for a significant fraction of iterations, the maximum execution time spikes to greater than 300 microseconds. Considering the LTE TTI of 1 ms, and especially in 5G where the TTI is expected to be much smaller, such variability in the execution time of Scheduler or MAC would result in unacceptably high missed TTI deadlines. We identify below potential causes of and solutions for this variability.

Linux kernel scheduler can migrate a process across cores. To mitigate this, the RAN function process can be "pinned" to a specific CPU core ("affinity"). The Linux kernel may schedule other processes or kernel tasks on the core on which the RAN function is running, due to which the RAN function may not get continuous access to the core. To mitigate this, the CPU core on which the application is running can be "isolated" from other processes or kernel tasks. This can be accomplished by either the isolcpus boot parameter, or cgroups. The Linux kernel may service soft IRQs on the same core as the desired application. Interrupt redirection can mitigate this, e.g. by configuring the irqbalance daemon which would otherwise service interrupts across all CPU cores. In extreme cases, the fact that the linux kernel scheduler sends a "tick" to each CPU core itself can cause some variability in the application execution. If the RAN function is run inside a virtual machine (VM), the actual physical CPUs (pCPUs) of the host on which the VM's virtual CPUs (vCPUs) are executed may change, adding further variability. Pinning specific vCPUs to pCPUs can mitigate this. System Management Interrupts (SMIs) initiated from the BIOS can take control of the processor away from the OS. These can result in unpredictable outage for RAN real-time functions. Their impact may be mitigated by disabling them. SMI behavior, frequency and impact is dependent on the server OEM, so it is essential that server OEMs allow proper control of the SMIs to the OS or hypervisor.

Taken together, these mechanisms have the effect of creating an essentially dedicated environment for RAN functions, free from interruptions, on an otherwise non-real-time processor-shared system like Linux. Using all these mechanisms, Figure 7 shows the performance of the example application above running on the same CPU and OS as in Figure 6, on the host Linux (black line) and in a virtual machine with the same guest kernel (red line).

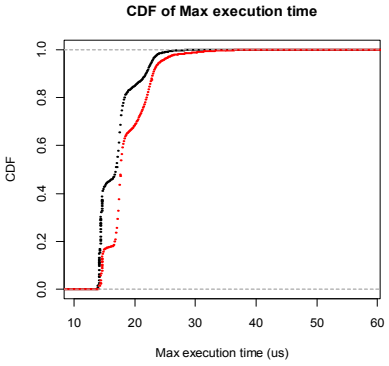

Figure 7. Cumulative Distribution Function (CDF) of execution jitter on Linux.

It is seen that large spikes are eliminated, execution time never exceeds 60ms, and is largely below 30us. We conclude that for LTE TTI of 1ms, the potential variability introduced by a generic linux environment can be brought to a tolerable range of 30 to 60us. However, we note that for even shorter 5G TTIs, such variability may still pose problems.

V. Pooling and Elastic Scaling

A key objective for Cloud RAN is to enable dynamic right-sizing of the processing resources based on the workload. In this section, we take a look at design considerations for mechanisms pooling across cells, load-balancing, and elastic scaling, which are important for achieving this objective.

A conventional view is that Cloud RAN will consist of a set of virtualized entities (e.g. Virtual Machines or Containers), each serving one cell (a "virtualized cell"), as illustrated in Figure 8, each using its own CPU cores. In the figure, "L2" represents the RAN Layer-2 functions PDCP/RLC/MAC, "Sch" stands for Scheduler, "Trs" represents the Transport module that terminates GTP tunnels towards the gateway, and "L3" represents the Radio Resource Control (RRC) signaling. As noted earlier, Layer-1 (PHY) functions may possibly be not virtualized, and hence are not shown below, but the following concepts apply as well to Layer-1 if virtualized.

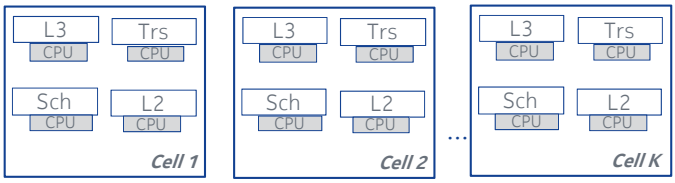

Figure 8. Conventional view of Cloud RAN virtualized cells

In principle processor pooling can be achieved among these per-cell VMs by sharing access to physical CPU cores via a hypervisor such as KVM/Linux. However, because some of the RAN functions of each cell have real-time constraints, a generic processor-sharing hypervisor will not be able to ensure that each cell's functions will meet their deadlines in each TTI. Further, more careful configuration is needed to ensure low execution jitter as discussed in the previous section. Moreover, each additional virtual machine can bring overheads e.g. memory consumption and guest kernel overhead. So having separate VMs for each cell may not be the best, especially for low-traffic cells. In the following we propose a different structure for the RAN that involves decomposing and reassembling the RAN functions across cells to yield an efficient overall structure that aims to maximize pooling (statistical multiplexing) gains in terms of processor allocations, while assuring deadlines are met.

We observe that some RAN functions operate on one user at a time (per-user operations), while others operate on a cell context at a time (per-cell operations). For example, the PDCP, RLC, and MAC operations are per-user operations. The PDCP layer performs ciphering and header compression, and when processing any given packet, it only needs to access the context of the user that the packet belongs to (e.g. bearer state, ciphering key, sequence numbers, etc), regardless of the cell to which the user is connected. In contrast, other functions such

as the Scheduler are per-cell functions, because a cell's Scheduler has to consider the state of all the users in the cell (e.g. channel state and scheduling metrics of all users, and the resources available to the cell) in order to make optimal resource allocations. Figure 9 shows an illustration of per-user and per-cell functions of the RAN.

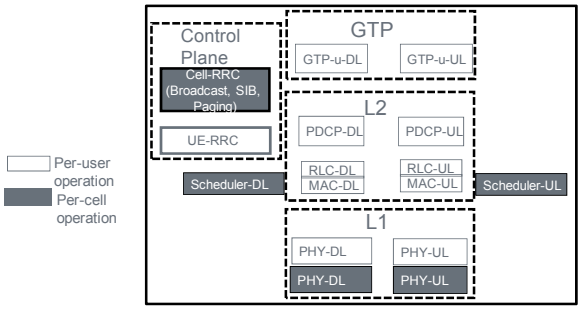

Figure 9. Per-cell and Per-user operations in RAN.

Based on this, we propose a structure called *Horizontal Aggregation* for per-user functions, where instead of having separate VMs for each cell, we have a set of VMs each of which is capable of serving any user from any cell. Thus, for example, we can have a set of VMs each of which is capable of performing PDCP/RLC/MAC functions for any user from any cell. Users can then be assigned to the PDCP/RLC/MAC VMs in a load-balancing fashion – e.g. new users are assigned to a VM on arrival based on the user or traffic distribution across the VMs. Within a given VM, the processing of different users' packets can be efficiently load-balanced across multiple vCPUs using the Event Machine model [15]. On any given physical server, it is enough to have only one PDCP/RLC/MAC VM to minimize VM overhead, though one may choose to have more than one to limit impacts of VM failure. Further, it is also possible to ensure that the real-time functions (e.g. MAC) meet their deadlines by treating the RLC/MAC tasks within the PDCP/RLC/MAC as higher priority compared to the non-real-time PDCP, e.g. with queues of different priorities in Open Event Machine. Horizontal Aggregation thus efficiently achieves pooling gains by allowing mixing of users across cells, and efficiently sharing between real-time and non-real-time functions.

Even for per-cell operations such as Scheduler, it is not necessary to have separate VMs per cell. One can conceive of a structure wherein a VM with a certain number of vCPUs can serve the Schedulers of a group of cells. When a new cell is added to the network, the cell's Scheduler is assigned to one of the existing Scheduler VMs in a load-balancing fashion. To ensure that all cells' Schedulers assigned to a given VM meet their deadlines, the number of vCPUs (and corresponding mapping to pCPUs) must be dimensioned to be sufficient to ensure adequate processing resources. We also note here that the decentralized algorithm architecture for multi-cell coordination discussed in Section III allows a flexible mapping of cells' Schedulers to VMs. The Schedulers of different cells can coordinate with other cells across the virtual interconnect (e.g. virtual or physical switches), and a cell has flexibility to include other cells in its liquid cluster regardless of whether those cells are mapped to the same VM or not.

Thus, by decomposing the RAN functions of the various cells and reassembling them into VMs for per-cell and per-user operations, we reach a flexible structure that allows the right type of pooling and scalability for each RAN function, as illustrated in Figure 10.

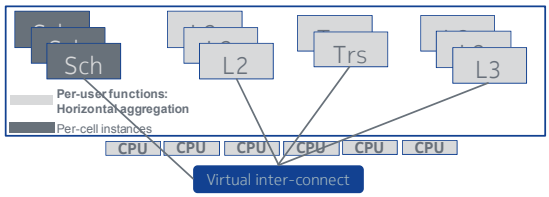

Figure 10. Flexible structure for Cloud RAN based on decomposing and reassembling the RAN functions.

We note that the total processing resources consumed by each type of function should be further dynamically modified based on the workload, a concept known as *elastic scaling* which was introduced in [15]. It was noted there that RAN functions may have non-real-time bottlenecks as well as real-time bottlenecks. For example, for the Layer-2 (PDCP/RLC/MAC), non-real-time bottlenecks may be quantified by the number of queued packets awaiting PDCP processing, while real-time bottlenecks may be quantified by the frequency of missed deadlines by the RLC/MAC. An algorithm for elastic scaling was proposed in [15], which takes into account both the real-time and non-real-time bottlenecks. We note one implication of elastic scaling based on the real-time bottleneck in the context of the constraint introduced by the LTE HARQ on the processing latency and fronthaul latency+jitter noted in Section II. Even if the workload does not change, if the fronthaul jitter or latency increases, that will result in a real-time bottleneck for the processing resulting in increased frequency of missed deadlines. Elastic scaling would then trigger to increase the amount of processing available, so as to finish the same workload in less time with more processors, thus automatically counteracting the increase in the fronthaul latency+jitter.

We propose a framework for Elastic Scaling to enable hyper-scale for RAN, by adapting both the number of cores used by of each VM, as well as adapting the number of VMs to exploit the availability of additional physical servers, while ensuring that both real-time and non-real-time bottleneck metrics meet targets. The framework is illustrated in Figure 11.

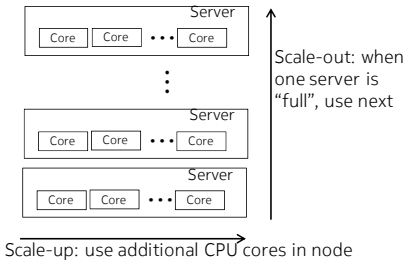

Figure 11. Combining scale-up and scale-out for RAN.

Elastic Scaling can add vCPUs to an existing VM, called *scale-up*. This can happen either by "vCPU hot-swap", or bringing already-initialized but idle vCPUs into service. As

noted in Section IV, to ensure low execution jitter for real-time functions, a VM's vCPUs may have to be pinned to suitably isolated physical CPUs, and this action may have to be dynamically undertaken when the elastic scaling logic decides to add a vCPU to a VM. Elastic Scaling can also add a new VM to perform a given function, called *scale-out*, e.g. when no more pCPUs are available on existing VM hosts to allow adding vCPUs to those VMs, elastic scaling can start a new VM on another server with available pCPUs, load-balancing across VMs by assigning users (for per-user functions) or cells (for per-cell functions) across the VMs. When all physical server CPU resources are used up, operators can simply provision one more server. Our framework for Elastic Scaling thus effectively utilizes both scale-up and scale-out to achieve hyper-scale for RAN, as illustrated in Figure 11.

VI. Virtualized Infrastructure Management and Orchestration

A key objective of Cloud RAN is increase automation in the deployment and lifecycle management of RAN functions. The ETSI Network Functions Virtualization (NFV) industry specifications group has developed a reference architectural framework [18] to formalize the adaptation of datacenter cloud management technologies such as virtualized infrastructure management (VIM) and orchestration for Virtualized Network Functions (VNFs), including RAN. In this section, we examine how the needs of real-time RAN functions and the framework for pooling and elastic scaling discussed in earlier sections compares with the NFV framework. The NFV reference architectural framework is shown in Figure 12.

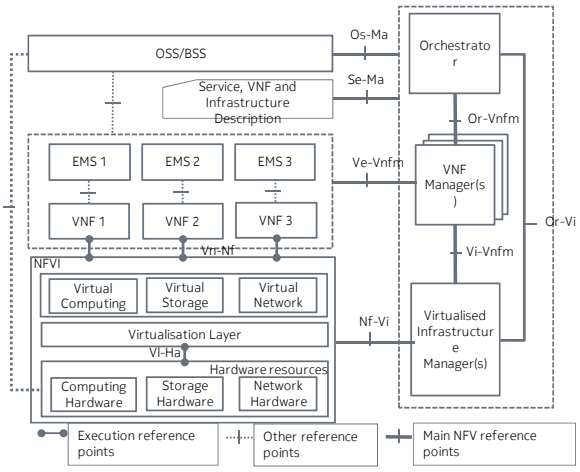

Figure 12. NFV reference architectural framework.

The Virtualized Infrastructure Manager (VIM) interacts with the hypervisor of a compute node to manage, initiate, and terminate VMs. A VNF can consist of one or more VMs, also known as VNF Components or VNFCs. The VNF Manager interacts with the VM instances in a possibly application-specific manner, and has an interface to the VIM to take actions on VMs. The Orchestrator is expected to take a system-wide view to determine the right actions to take across multiple VNFs, and also has an interface to the VIM to take actions on VMs. The VIM, VNF Manager, and Orchestrator are together referred to as MANO (Management and Orchestration) components. A common industry software which can serve as a VIM is OpenStack which is often used in data centers in the Information Technology (IT) industry to deploy and manage VMs in a cloud infrastructure.

We propose the following methodology for using NFV architectural framework to facilitate Cloud RAN operations such as the addition of a new cell, as illustrated in Figure 13.

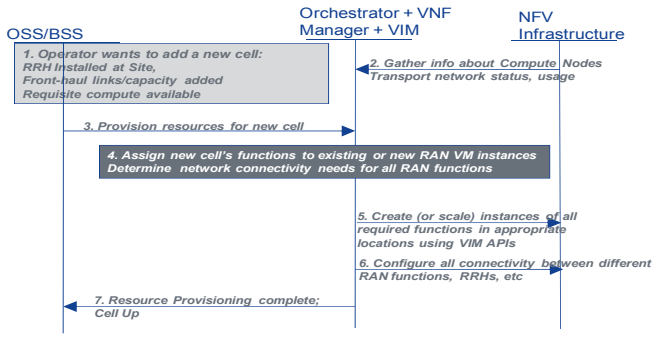

Figure 13. Framework for using the NFV architecture framework for addition of a cell.

Before a cell can be added, certain pre-requisites may have to be first put in place, e.g. installing transport connectivity such as fiber between the Cloud RAN and radio heads etc. Depending on the functional split, certain baseband functionality of the cell may have to reside at the cell site as well. So we propose a broader definition of the NFV infrastructure to include not just the centrally hosted processing resources but the processing resources at the cell sites as well, and potentially also the transport network interconnecting the cell site to the centrally hosted processing. In our framework, the NFV MANO components gather information (on a periodic or event-driven basis) regarding the usage and status of the NFV infrastructure and transport network. When the operator management console (OSS/BSS) requests the addition of a new cell, the NFV MANO components have to translate this directive into a set of actions in the NFV infrastructure. In the context of the horizontal aggregation structure we proposed in Section V, the MANO components have to decide whether existing VM instances are already suitable in terms of resources and placement to host the functions for the new cell, or whether elastic scaling needs to be invoked to scale-up (add vCPUs/pCPUs to existing VMs) or scale-out (create new VMs). Once this placement and scaling decision is made, the VIM APIs are used to direct the NFV infrastructure compute nodes to take the appropriate actions on the VMs, as well as reconfigure the transport connectivity.

We observed in Section IV that in order to ensure low execution jitter for real-time RAN functions on GPP/Linux compute nodes, a series of actions may be needed to create the appropriate environment. These include pinning of guest VM vCPUs to underlying physical CPUs, suitably isolating the pCPUs to prevent other processes from taking time away from the real-time RAN functions, etc. Further, certain RAN functions such as the Layer-1 may need to be instantiated on non-GPP hardware. These requirements are key to RAN virtualization, and do not exist in conventional IT datacenter VIM tools such as OpenStack. Thus we propose that in Step 5

in Figure 13, the VIM APIs that are used to instantiate the RAN functions in the NFV Infrastructure need to have suitable facilities to execute these actions on demand, taking the type of hardware (GPP or non-GPP) into account.

We proposed in Section III that the decision on placement of certain functions for the new cell, such as the Scheduler, should take into account the relationships between cells for multi-cell coordination or carrier aggregation and potential implications of the network topology interconnecting the processors on the Liquid Clusters of various cells. We further note that there are also real-time timing and bandwidth requirements between the different functions of a cell, such as Scheduler, RLC/MAC, and PHY, which may force constraints on the relative placement of these functions. Moreover, the possible need to place the PHY on certain dedicated, non-virtualized hardware may further constrain the placement of functions such as the MAC and Scheduler which interact with the PHY. We propose that the VNF Manager or Orchestrator jointly optimize the placement incorporating these constraints.

We also propose a joint optimization for the elastic scaling and placement decisions. As noted in Section IV, in order to ensure low execution jitter for real-time RAN functions, it may be necessary to pin a guest VM's vCPUs to suitably isolated pCPUs on the host. If either scale-up or scale-out is needed, the placement decision needs to take into account the availability of suitably isolated and unused pCPUs, and this results in the scaling and placement decisions becoming a joint optimization.

VII. Conclusion

In this paper, we investigated key challenges in achieving the objectives of Cloud RAN, and proposed solutions. In Section II, we examined some implications of functional splits, wherein all or a portion of the baseband functionality is centralized. We pointed out that the LTE HARQ loop limits the achievable scale of centralization for functional splits where the HARQ loop is terminated in the central site, and identified a complex tradeoff that underlies the choice of functional split. In Section III, we examined the architecture of multi-cell coordination algorithms, proposing that they should be designed to have a decentralized algorithm which will maximize the flexibility of placement of RAN functions within the RAN cloud as well as the benefit from multi-cell coordination. In Section IV, we took a deeper look from a software perspective at achieving real-time performance suitable for RAN functions with GPP/Linux and virtualization, proposing various techniques to improve the execution jitter. In Section V, we proposed a flexible structure, based on decomposing the RAN functions of the various cells and reassembling them into per-cell and per-user components, that allows the right type of pooling and scalability for each RAN function. We showed how elastic scaling can help achieve hyper-scale for RAN, exploiting both scaling-up of existing VMs by adding cores as well as scaling out to additional servers. In Section VI, for the application of cloud management technologies such as VIM and orchestration for RAN functions, we proposed a framework to translate radio-network management actions such as addition of a cell into interactions between the virtualized management and orchestration functions and the compute infrastructure. We proposed that special needs for RAN due to real-time constraints and a mix of GPP and non-GPP hardware require extensions to the APIs. We also proposed a joint optimization of the placement and elastic scaling of RAN functions, taking into account the interconnect between processors, multi-cell coordination relationships between cells, timing constraints between RAN functions, and special operations needed to assure low execution jitter for real-time RAN functions.

Acknowledgment

The authors would like to acknowledge detailed technical discussions with many colleagues in Nokia Networks, as well as with wireless network operators in several countries.

References

[1] Cisco, "Cisco VNI Global Mobile Data Traffic Forecast 2014-2019," Online at http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html

[2] A. Checko *et al*, "Cloud RAN for Mobile Networks - A Technology Overview," *IEEE Comm. Surveys and Tutorials*, Vol. 17, No. 1, First Quarter 2015.

[3] P. Rost *et al,* "Cloud Technologies for Flexible 5G Radio Access Networks," *IEEE Comm. Mag.,* Vol. 52, No. 5, pp. 68-76, May 2014.

[4] 3GPP, "E-UTRA and E-UTRAN: Overall Description: Stage 2," TS 36.300, Release 12, Dec. 2014.

[5] D. Lee *et al,* "Coordinated Multipoint Transmission and Reception in LTE-Advanced: Deployment Scenarios and Operational Challenges," IEEE Comm. Mag., pp. 148-155, Feb. 2012.

[6] CPRI, "Common Public Radio Interface (CPRI): Interface Specification," version 6.1, July 2014. Online at http://www.cpri.info

[7] OBSAI, "Reference Point 3 Specification," version 4.2. Online at http://www.obsai.com

[8] NGMN Alliance, "Further Studies on Critical C-RAN Technologies," Version 1.0, March 2015. Online at https://www.ngmn.org/uploads/media/NGMN_RANEV_D2_Further_Study_on_Critical_C-RAN_Technologes_v1.0.pdf

[9] S. Gulati *et al*, "Performance Analysis of Centralized RAN Deployment with Non-ideal fronthaul in LTE-Advanced Networks," submitted to IEEE VTC-Spring 2016.

[10] 3GPP, "Coordinated Multi-Point Operation for LTE Physical Layer Aspects (Release 11)," TR36.819, v11.2.0, Sept. 2013.

[11] S. Gulati *et al*, "Performance Analysis of Distributed Multi-cell Coordinated Scheduler," in *Proc. IEEE VTC-Fall 2015*.

[12] D. Pengoria *et al*, "Performance of Co-Operative Uplink Reception with Non-Ideal Backhaul," in *Proc. IEEE VTC Spring 2015.*

[13] R. Agrawal *et al*, "Dynamic Point Selection for LTE-Advanced: Algorithms and Performance," in *Proc. IEEE WCNC 2014*.

[14] 3GPP, "X2 Application Protocol (X2-AP) Release 12," TS 36.423, v12.7.0, Sept. 2015.

[15] R. Agrawal *et al,* "Architecture Principles for Cloud RAN," submitted to IEEE VTC-Spring 2016.

[16] Open Event Machine Development Team, "Open Event Machine: An event driven processing runtime for multicore," Online at http://sourceforge.net/projects/eventmachine

[17] ETSI, "Network Functions Virtualization: An Introduction, Benefits, Enablers, Challenges & Call for Action," Oct. 2012. Online at https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[18] ETSI, "Network Functions Virtualization (NFV): Architectural Framework," ETSI GS NFV 002 v1.1.1, Oct. 2013. Online at http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf