https://helda.helsinki.fi

"-"

NoSQL stores for coreless mobile networks

Ojala, Frans

2017-10-30

Ojala, F, Rao, A, Flinck, H & Tarkoma, S 2017, NoSQL stores for coreless mobile networks. in 2017 IEEE Conference on Standards for Communications and Networking (CSCN). IEEE, New York, pp. 200-206, IEEE Conference on Standards for Communications and Networking, Helsinki, Finland, 18/09/2017. https://doi.org/10.1109/CSCN.2017.8088622

http://hdl.handle.net/10138/347910
10.1109/CSCN.2017.8088622

Downloaded from Helda, University of Helsinki institutional repository.

This is an electronic reprint of the original article.

This reprint may differ from the original in pagination and typographic detail.

Please cite the original version.

NoSQL Stores for Coreless Mobile Networks

Frans Ojala[†], Ashwin Rao[*], Hannu Flinck[‡], Sasu Tarkoma[*]
[*]University of Helsinki [†]Avarko OY [‡]Nokia Bell Labs

***Abstract*—The goals of 5G networks—low latency, high bandwidth, and support for fast mobility—are non-trivial and they demand improvements across all involved technology fields. Researchers are therefore exploring approaches that leverage on network function virtualization and software-defined networking for meeting the demands of verticals expected to use 5G networks. One approach which appears promising is the concept of a coreless mobile network where the key network functions are placed at the edge of the network. In this article we focus on management of the user-context state in a coreless mobile network, and posit that these network functions can use a NoSQL data store for maintaining the user-context and other state variables. We first present an overview of promising NoSQL data stores and evaluate their suitability. We then present the results of benchmarking the Apache Geode data store as an example of a state management solution which could serve a coreless mobile network. During our tests we observe that the Apache Geode data store is, subject to its configuration, capable of delivering the data model, consistency, and high availability required by a coreless mobile network.**

Keywords*—*NoSQL, mobile networks, 5G, coreless networks.

I. Introduction

In 5G networks, the communication infrastructure used by our smartphones and tablets will be also used by a wide range of devices and verticals. These verticals are expected to generate data traffic [1], however current mobile networks were conceptualized when the volume of traditional voice traffic was comparable to the volume of data traffic. Furthermore, the key network functions serving Long-Term Evolution (LTE) networks have a convoluted control and data plane which results in a congested control plane [2], [3].

Li *et al.* [2] argue that software defined networking (SDN) can be leveraged to simplify the design and management of cellular data networks. The key insights of their design involves splitting the control and data plane. Hampel *et al.* [3], and Osmani *et al.* [4] show that such a split is indeed possible. The insights from these works have been built upon to explore different ways in which the key network functions of the LTE core can be refactored and coalesced to optimize the cellular network [5], [6]. At the high level there are two approaches for addressing some of the shortcomings of current mobile networks: a) move functionality to the cloud and create the Cloud Radio Access Network (C-RAN) [7], or b) move functionality to the edge of the network on the lines of Multi-access Edge Computing (MEC) [8].

While each of these approaches have their benefits and shortcomings, in this article we focus on the later, *i.e.*, a coreless mobile network where all the functionality is moved to the edge. We first present an overview of a coreless mobile network. This concept can be seen as an evolution of MEC, featuring among others, a fully virtualized LTE core and an interconnecting data layer for state management. We then present an analysis of the properties required of a data store serving coreless mobile networks. This is followed by an evaluation of the Apache Geode NoQSL data store as an example of a data storage system that could be leveraged by coreless mobile networks. Our key contributions are as follows.

- We enumerate the state variables stored at each network function, and discuss how they are updated across five procedures. The small number of state variables updated during the procedures motivates us to explore NoSQL stores which support delta updates.
- We present an architecture for data storage and also discuss the benefits of using a NoSQL store for the state management information in coreless mobile networks. Specifically, we explore how the eventual consistency property of NoSQL stores can be leveraged for creating high-availability and consistency zones to support mobility of UEs.
- We present an analysis of the desirable properties from a NoSQL data store serving coreless mobile networks. We also present results of preliminary experiments to quantify the performance of the Apache Geode NoSQL data store serving a coreless mobile networks.

Roadmap. In §II we motivate the requirements of a data store serving a coreless mobile networks, and we discuss the avenues opened by using a NoSQL store in §III. We then present the results of benchmarking the Apache Geode NoSQL store in §IV, and we finally conclude in §V.

II. Background and Motivation

In this section we first present an overview of the key network functions serving current mobile networks. We then discuss the user-context which is stored in each of the network functions, followed by a discussion on coreless mobile networks.

A. Architecture of Current Mobile Networks

The System Architecture Evolution (SAE), also known as the Evolved Packet System (EPS), is the current evolutionary step of the General Packet Radio Service (GPRS) core network. SAE has a flat all-IP protocol stack, supports higher throughput Radio Access Networks (RANs), and supports mobility to and from legacy systems in addition to supporting other wireless access technologies such as WiFi. An SAE based LTE network has the following two key components.

a) Radio Access Network (RAN). The Evolved Node B (eNB) is a key network function in the RAN. Each eNB is responsible for managing a number of cells and the connectivity of the UE

Frans Ojala did this work as a student at the University of Helsinki

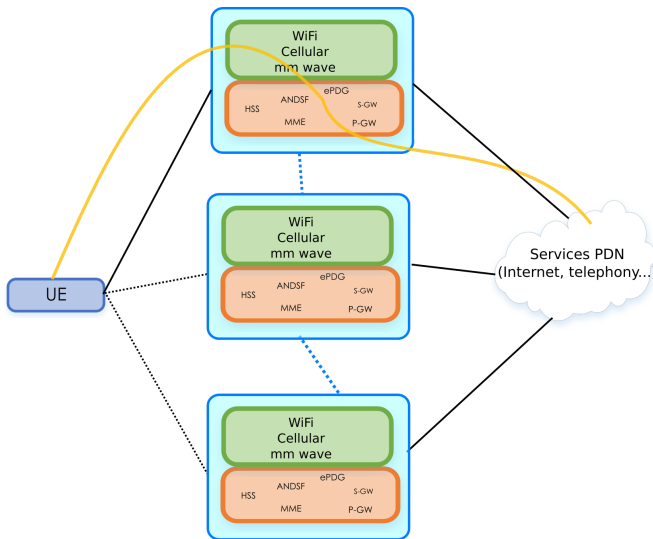


Fig. 1. **A Coreless Mobile Network.** *The network functions are virtualized and moved to the edge of the network. Each box placed at the edge is capable of providing connectivity to the UE. These boxes communicate with each other to replicate the user context in real-time, and this replicated user context can be leveraged for supporting mobility.*

within the geographical area covered. Some of the core responsibilities of the eNB include: a) radio resource control which includes activities such as scheduling and dynamic allocation of resources; b) compressing packet headers for reducing the overheads; and c) providing security by encrypting of all data sent over the radio link [9].

b) Evolved Packet Core (EPC). Some of the key network functions in the EPC are as follows.

1) The *Mobility Management Entity (MME)* is responsible for managing the mobility of the user equipment (UE), and in particular selecting the Serving gateway for the UE during the initial attach and handover procedures.

2) The *Packet Data Network Gateway (P-GW)* is the gateway to IP networks used by the UE.

3) the *Serving Gateway (S-GW)* is like a switch in an IP network, and it also acts as a mobility anchor.

4) The *Home Subscriber Server (HSS)* is largely responsible for storing information relating to the subscribed users.

5) The *Policy and Charging Rules Function (PCRF)* is largely responsible for making decisions for ensuring complying with policies and service level agreements.

6) The *Access Network Discovery and Selection Function (ANDSF)* provides the UE with information about nearby access networks, including those offering services over other technologies such as WiFi.

B. Coreless Mobile Networks

The current mobile networks are largely inflexible and they cannot be leveraged to achieve the planned advancements in 5G [10]. Furthermore, the S-GW, the P-GW, and the eNB serve both the control and data plane [2], [3]. This distribution of the control layer across several disparate entities causes a plethora of problems. To address these shortcomings, two approaches have been proposed: a) move all the functionality to the cloud [7], or b) move functionality to the edge of the network for creating a coreless mobile network.

As shown in Figure 1, in a coreless mobile network all of the network functionality is moved to the edge of the network. Our previous works which leverage the advances in software defined networking (SDN) and network function virtualization (NFV) show that it is indeed possible to build such a network [4], [5], [6]. This architecture follows along the themes of Mobile Edge Computing MEC [8], as well as the concept of a Shared Data Layer by Nokia [11]. Some of the key requirements of a coreless mobile network are as follows.

1) *In-situ computing.* The network functions are co-located at the network edge, and all computing is done locally. Contrasting to the centralized nature of the traditional EPC, this model is entirely decentralized. Each smart-box can be viewed as an access point into any connected PDN.

2) *Distributed data store for state management.* The network functions are required to maintain the state of the UEs being served, and this state information must be consistent across all the network functions. Furthermore, to support mobility, this state information needs to be propagated across the various smart-boxes serving the coreless mobile network. A coreless network will therefore require a distributed data store for state management.

We believe that such a distributed data store can be implemented using NoSQL data stores because these data stores have proved their mettle in large scale clouds [12], [13], [14]. Furthermore, in §III-A we posit that the UE context information stored at the network functions serving mobile networks can be represented as key-value pairs. In the rest of the paper we discuss our approach of using a NoSQL data store to serve a coreless mobile network.

III. NoSQL Data Stores for Coreless Networks

An objective of our work is to discover a data storage solution which can be employed to manage the state of UEs in a coreless mobile network. Clearly, distribution of data storage is necessary, however distribution brings its own problems. Specifically the problem of dealing with the CAP-theorem which states that one may be required to choose any two of a) consistency, b) availability, and c) partition tolerance [15], [16]. In this section, we first present our motivation for using NoSQL stores in coreless mobile networks. We then compare popular NoSQL stores which can be leveraged on for managing the state of the UEs in coreless mobile networks.

A. NoSQL for Storing User Context

The distributed data store is expected to store the UE context for the use of the network functions serving the network. In Table I we summarize the transition of the key state variables of the UE context during the Initial Attach, Detach, S1-release, Service Request, and X2 handover procedures.[1]

The network functions maintain a consistent UE state and context by exchanging signaling messages. For instance, it is estimated that in regular conditions each UE may require the network functions to exchange more than 500 signaling messages per hour in the EPC [17], [18]. Furthermore, the number of signaling messages increases for always-on UEs.

[1] We obtained these values from the documentation of LTE procedures available at http://www.netmanias.com/en/?m=view&id=techdocs&no=6002 (Referenced on 14.05.2017)

TABLE I. UE CONTEXT STORED AND USED BY NETWORK FUNCTIONS

Category	Name	Initial Attach	Detach	S1-release	Service Request	X2 Handover
UE identifiers	C-RNTI	+	×	×	+	∪
	eNB S1AP UE ID	+	×	×	+	∪
	GUTI	+	•	•	•	•
	IMSI	•	•	•	•	•
	IP	+	×	•	•	•
	MME S1AP UE ID	+	×	×	+	•
Location	ECGI	+	×	×	+	∪
	MME ID	+	•	•	•	•
	TAI	+	•	•	•	•
	TAI-list	+	•	•	•	•
Security	K master	•	•	•	•	•
	NAS security	+	•	•	•	•
	AS security	+	×	×	+	∪
EPS Bearers	APN	•	•	•	•	•
	APN in use	+	×	•	•	•
	EPS bearer ID	+	×	•	•	•
	E-RAB ID	+	×	•	•	•
	DRB ID	+	×	×	+	∪
	S1 TEID (UL)	+	×	•	•	•
	S1 TEID (DL)	+	×	×	+	∪
	S5 TEID (UL/DL)	+	×	•	•	•
QoS Parameters	Access profile	•	•	•	•	•
	APN-AMBR (UL/DL)	+	•	•	•	•
	ARP	+	×	•	•	•
	UE-AMBR (UL/DL)	+	•	•	•	•
	Subscriber profile	•	•	•	•	•
	QCI	+	×	•	•	•
	TFT (UL)	+	×	•	•	•

The table summarizes the life of a state variable during the Initial Attach, Detach, S1-release, Service Request, and X2 handover procedures. The values of the symbols are as follows: + implies that the variable is added to the user-context during the procedure, and it becomes available after the procedure is completed; × implies that the variable is removed from the user-context, and becomes invalid after the end of the procedure; • implies that the variable is not modified during the procedure; ∪ implies that the variable is modified during the procedure, and the new values become available after the procedure has been completed. UL indicates the variable corresponding to the uplink, and DL corresponds to the variable corresponding to the downlink.

This is a serious shortcoming because some verticals such as hospitals can require the devices to be always-on.

As shown in Table I, the UE context is comprised of identifiers and cryptographic keys. In particular, the UE context always contains at least one identifier which does not change during the lifetime of the subscriber: the IMSI; another partially static identifier is the GUTI. These static identifiers can be used as keys and the entire context can be the value object. *Thus the data model is most easily described in terms of key-value pairs of NoSQL data stores.*

B. NoSQL for High Availability and Consistency

The context data that governs UE connectivity has a special property not seen in traditional cloud-based applications. Specifically, a UE is tightly bound to a geographic location, and can be assumed to move at a reasonable speed. Binding the UE context data to a geographic location opens new avenues for optimization. More specifically, it should be possible to relax consistency and availability constraints on the data in locations far away from the UE while still retaining the data fully consistent and highly available near the UE. Thus, we can form High availability and consistency zones, henceforth referred to as HAC-zones. The data stored in the distributed data store is eventually consistent, however the data is strongly consistent and highly available in a small subset of the global servers which are placed in geographical areas near a UE.

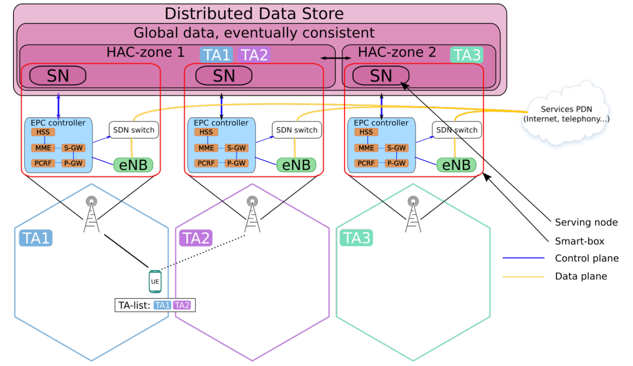


Fig. 2. **High Availability and Consistency (HAC) Zones.** *The data store consists of the HAC-zone and the Global data zone. The data is always available and consistent in the HAC zones. The data stored in the Global data zone is eventually consistent. TA denotes the tracking area, while SN denotes the serving node, i.e, the node storing the data and responding to queries.*

As shown in Figure 2, the data is available outside the HAC-zone, however it might not contain the very latest update. The data will be eventually consistent to enable the operator to maintain coherent information on its subscribers. Enabling such HAC-zones brings a new set of requirements from the data store. It must support the creation and management of data sets with varying degrees of consistency and availability. It must also support restricting the data set to any set or subset of servers that may or may not reside in the same data center or geographical region. In order to enable the UE movement between HAC-zones the system must also support mechanisms of transferring data from one set of servers hosting one HAC-zone to another set hosting the next HAC-zone.

In a coreless network, all signaling messages are exchanged within the smart-box hosting the eNB and core network entities. Only fetches and updates to the distributed data storage will traverse the interconnect. For instance, during the Initial attach the UE context does not exist. Only the subscription profile is installed, and the EPC controller in a smart-box fetches the profile from the data store and constructs the context and installs forwarding rules. During the subsequent procedures, the updated UE context is pushed into the global data store for ensuring eventual consistency.

C. Comparison of NoSQL stores

Numerous NoSQL stores exist, and discussing each one of them is beyond the scope of this paper.[2] In Figure 3 we present a short comparison of some of the popular NoSQL data stores. We restrict the comparison to the chosen set of data stores because their data model most closely resembles the one desired by the coreless mobile network architecture.

Specifically, we compare Apache Casandra,[3] Memcached,[4] Voldemort,[5] Google Spanner [14], Infinispan,[6] Apache Ignite,[7]

[2] A comprehensive listing of NoSQL data stores is available at http://nosql-databases.org (referenced on 14.05.2017).
[3] http://cassandra.apache.org (referenced on 14.05.2017).
[4] https://memcached.org (referenced on 14.05.2017).
[5] http://www.project-voldemort.com/voldemort/ (referenced on 14.05.2017).
[6] http://infinispan.org (referenced on 14.05.2017).
[7] https://ignite.apache.org (referenced on 14.05.2017).

HBase,[8] Redis,[9] Amazon Dynamo [13], and Apache Geode.[10]

For our work, we chose Apache Geode primarily because it supports delta updates, geolocation of data, and strong consistency. From Table I, it is clear that during the procedures only a small number of state variables are updated. Delta updates can therefore be leverage for optimal usage of the network bandwidth for ensuring consistency: when a state variable is updated, only the difference with the previous state is sent over the network. Furthermore, Apache Geode also supports geolocation of data and strong consistency, making it appropriate for the coreless mobile networks. In contrast, one can argue that because in Apache Geode the data is unavailable in the minority quorum, a more high availability failure model should be adopted. However, we assume that the interconnecting network is stable and that there are redundant paths to keep it functioning properly even in the event of network partitions. To reiterate: we assume that the consistency of the UE context state in the local geographic region is more important than extremely-high availability.

IV. Benchmarking Apache Geode

One of the standard methods to test a system is benchmarking it with appropriate workloads. For evaluating the performance of the Apache Geode NoSQL Store we chose the Yahoo! cloud serving benchmark (YCSB) [19]. We had to modify parts of YCSB testing framework to incorporate a) the data model for user context, b) the workloads corresponding to UE state changes, and c) the tests for delta updates which are natively not supported by YCSB. Furthermore, we incorporated changes to demonstrate and evaluate HAC-zoning.[11]

A. Apache Geode Topology for Experiments

As shown in Figure 4, an Apache Geode cluster is comprised of two types of processes: locators and serving nodes. The locators orchestrate the cluster. They connect to the serving nodes for receiving load information and also for load balancing. The client applications first connect to the locators to request connections to serving nodes. The locator responds with references to serving nodes in a least-loaded first fashion. The serving nodes respond to the requests made by the clients for adding, modifying, and removing data entries from the NoSQL store. Each client pools the connections and uses the available connections for single-hop querying of data to and from the serving nodes. Please note that this cluster topology is the same for replicated and partitioned regions discussed in §IV-C. For our experiments, we use the YCSB client to add, remove, and modify the data entries in the Apache Geode store.

B. Evaluation Setup

We used up to five Dell C6320 servers, and each server had the following configuration.

1) *CPU.* Two Intel(R) Xeon(R) CPU E5-2680 v3 operating at 2.50 GHz providing a total of 48 logical cores with hyper-threading enabled.

[8]http://hbase.apache.org (referenced on 14.05.2017).

[9]http://redis.io (referenced on 14.05.2017).

[10]http://geode.apache.org (referenced on 14.05.2017).

[11]Our code is publicly available at the following URL: https://github.com/Virta/YCSB/tree/geode-updates

TABLE II. Procedures and size of delta updates.

Procedure	Fraction	Update size (bits)
Initial attach	0.011	3744
Detach	0.011	2400
Service request	0.313	1280
S1 release	0.313	1280
TA update	0.094	736
Handover	0.124	1088
Cell re-selection	0.065	0
Session management	0.069	152

Fraction represents the fraction of total procedures, and the update size represents the number of bits of user context which were updated during the procedure. For example, 0.011 procedures emulated were of type Initial attach and each Initial Attach resulted in the modification of 3744 bits.

2) *Main memory.* 256 GB DDR-4 dual rank 2133 MHz main memory.

3) *Network interface.* Two 10 Gbps NICs with bonding disabled.

5) *Disk Storage.* Two 600 GB 10000 rpm disks configured as RAID 1.

6) *Operating System.* Each machine ran the Ubuntu 16.04 operating system.

Furthermore these servers were in an isolated network and they were disconnected from the rest of the Internet during the experiments. The Apache Geode cluster was built from the incubating.M2-source[12] with Oracle Java 1.8 as the compiler and runtime. The same Java version was also used for compiling and running our customized YCSB. During the benchmarking test we emulated 1 million UEs, each making 3 million operations. Note that each operation corresponds to a procedure—initial attach, detach, *etc.*—which involve signaling messages exchanged between the network functions, and also between the network functions and the UE. Furthermore, we also scaled the number of load generating threads of YCSB incrementally to find the maximum throughput as it was not possible to achieve maximum throughput with a single load generating thread. While not strictly necessary, we included a locator on each physical machine for two reasons: it eased the automation of the scaling benchmark, and it provides extra redundancy for the clients pool requests in the event of a locator failure.

The workload used for the benchmarking was based on signaling load distribution observed by Nokia in a tracking area of 15 eNBs [17]. We use this dataset to obtain the fraction of procedures which might be observed in a typical cellular network. We then uses this fraction to emulate the signals generated by one million UEs, where each UE has a context of 4596 bits.[13]. The procedures emulated and the update sizes are presented in Table II. Please note that this table is not a comprehensive list of procedures, but we use it as an example of a typical workload in a cellular network.

C. Results

We performed tests to evaluate the performance of Apache Geode under the following three scenarios

1) *Partitioned regions.* In Apache Geode, each partitioned region with redundant copies of data contains a copy called the primary copy. This primary copy is the update anchor: all writes first go to this copy from where they are synchronously

[12]https://github.com/apache/incubator-geode/tree/rel/v1.0.0-incubating.M2

[13]The details of information stored in each bit are available Table 6 of [20].

	Type	Consistency model	Availability model	Partitioning model	Distribution model	Data model	Replication model	Data localisation	Failure model	Data storage
Apache Cassandra	Distributed database	Write quorum, read single/ quorum	High availability through replication	Consistent hashing, order preserving	Coordinated, symmetrical	Structured, column store	Zookeeper orchestration	Rack (un)aware, datacenter aware	Accrual FD, Scuttlebutt	Commit log, in-memory, disk dump
Memcached	Cache	Locked single read/write	Least recently used	None	Symmetrical	Key-value store	None	None	None	In-memory, no persistence
Voldemort	Data store	Vector clock versioning, read-repair	High availability through replication	Consistent hashing	Symmetrical	Key-value store	N, where N is user def ned	None	Consistent hashing replacement	Cache, disk
Google Spanner	Database / data grid	Paxos groups, tuneable reads	High availability through replication	Tuneable	Symmetrical storage, hierarchical orchestration	Bag of (key, timestamp, value), relation-like	Tuneable, directory shared configuration	Common pref x directory, aff nity collocation	Automatic failover between replicas	Tuneable
Infinispan	Data store / cache	Optimistic, pessimistic locks, total order	High availability via simple clustering	Consistent hashing	Symmetrical	Binary/object representation	Local, invalidation, replicated, distributed	None	Degraded/normal with rebalance, reduced availability	In-memory, disk persistence
Apache Ignite	Data grid, compute platform	ACID, deadlock-free, transactions, locks	Tuneable	Depends on replication and availability conf guration	Tuneable	Key-value store	Local, partitioned or replicated	Pluggable hashing, aff nity collocation	Unknown	In-memory, client side (near) cache
HBase	Distributed database on HDFS	Strong consistency (ACID)	Highly available reads (HDFS)	HDFS	HDFS	Column-oriented key-value store	HDFS	None	HDFS, data unavailability	HDFS: disk
Redis	Cache, database, message broker	Reasonable consistency	Reasonable availability	Depends on client, consistent hashing	Master-slave hierarchy	Key-value store	Master-slave, asynchronous	Hash tags	Minority unavailable, replica migration	In-memory, snapshots, journaling
Amazon Dynamo	Distributed database	Vector clock versioning	High available through vector clock and reconciliation	Consistent hashing	Symmetrical	Key-value store	Tuneable per instance	None	Sloppy quorum, hinted handoff, merkle-tree anti entropy, gossip	Caches and write buffers to persistence
Apache Geode	Distributed cache	Strong consistency	High availability through replication	Consistent hashing, custom	P2P, client-server, multi-site	Key-value store	Tuneable, custom, redundancy zones	Fixed custom partitioning	Member weights for quorum, minority unavailable	In-memory, tuneable persistence

Fig. 3. **Comparison of popular NoSQL stores.** *We use the following parameters to compare the solutions: a) technique for ensuring consistency across hosts, b) the availability of the data for reading and writing in the event of network partitions, c) technique for partitioning, d) are all components the same (symmetrical), or is there a hierarchy of dedicated controlling and serving components, e) how the data is seen by the system, f) technique for data replication across multiple hosts, g) support for localization of data and support for geographical regions, h) the failure model, and i) the supported persistent storage options.*

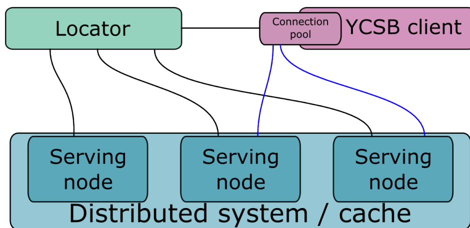


Fig. 4. **Experiment Topology**. *The Apache Geode Locator orchestrates the cluster while the Apache Geode Serving Nodes are responsible for storing the data and responding to the queries. We use the Yahoo! Cloud Serving Benchmark (YCSB) to benchmark Apache Geode using workloads which emulate the queries made by the network functions serving mobile networks.*

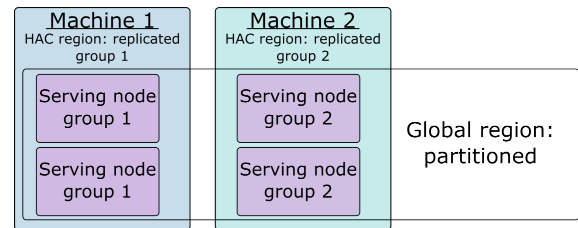


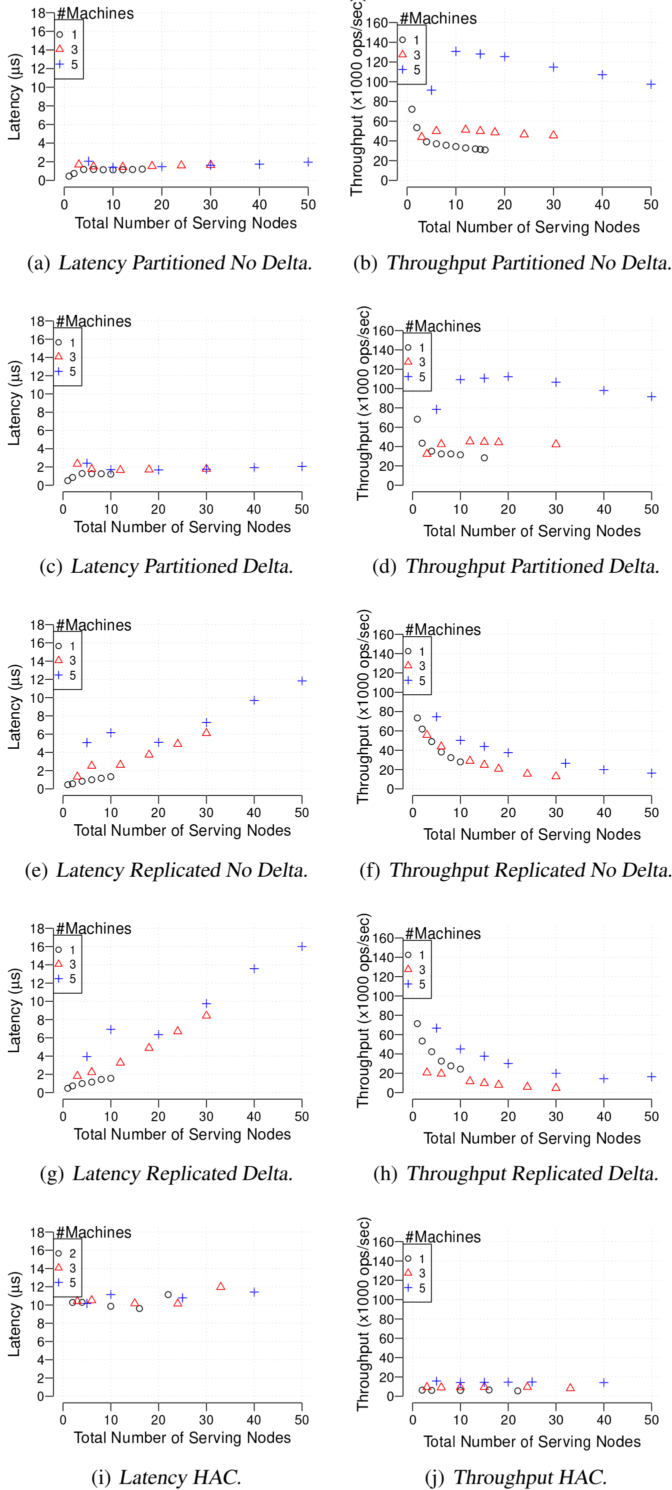
Fig. 5. **Example of HAC Grouping with Apache Geode**. *This example contains two machines, each containing one HAC-zone which are hosted by two serving nodes on each machine while the global region is hosted by all four serving nodes.*

propagated to all secondaries. In contrast, the reads go to any serving node with a copy of the data. During our evaluation, we used a three-way redundancy policy which the locators try to uphold in the event of a loss of serving nodes. Furthermore, Apache Geode also tries to place the three copies of data into serving nodes on different physical machines.

2) *Replicated regions.* A replicated region in Apache Geode is a data set that is symmetrically stored at every serving node that hosts the region. During our evaluation we configured a replicated region to use the *distributed-ack update procedure*: for each update the client receives an acknowledgement of reception from all serving nodes before continuing. While this technique is expected to be slower than no acknowledgement, it is known to dramatically improve the cache consistency.

3) *High Availability and Consistency (HAC).* We also demonstrate of a type of HAC-zoning discussed in subsection III-B. Specifically, for HAC-zoning we combine the partitioned regions and replicated regions in the following manner. For our evaluation, we split the 1 million UE objects into 10 disjoint chunks of equal size. Each of these chunks was assigned to a specific member group. As shown in Figure 5, only the serving nodes initiated with the specific group identifier hosted the data allocated to the said region. Each group was made host of a replicated region, within which the chunk of data is hosted. Further, all serving nodes were made host of a global region of type partitioned with three copy redundancy. In this scenario, we emulated the handover procedure as follows. Once a UE object had been fetched and

mutated, a new group was randomly selected, a new connection pool was established to the new group and the object was put there and the global region, and finally the UE object was removed from the old group.



(a) *Latency Partitioned No Delta.* (b) *Throughput Partitioned No Delta.*

(c) *Latency Partitioned Delta.* (d) *Throughput Partitioned Delta.*

(e) *Latency Replicated No Delta.* (f) *Throughput Replicated No Delta.*

(g) *Latency Replicated Delta.* (h) *Throughput Replicated Delta.*

(i) *Latency HAC.* (j) *Throughput HAC.*

Fig. 6. **Evaluation Results.** *Latency represents the average time required for a given operation, and throughput represents the average number of operations performed per second. Each operation emulates the state changes which take place during a procedure (Initial attach, Detach, etc.). The low latencies are a consequence of using the in-memory implementation of Apache Geode.*

We present the outcome of our experiments in Figure 6. The throughput represents the average number of operations per second, and the latency represents the average amount of time required to perform an operation. An operation corresponds to a state change which take place during a procedure.

1) *Partitioned regions* (Figures 6(a), 6(b), 6(c) and 6(d)). With only one machine in the cluster the throughput decreases quite rapidly with the addition of serving nodes. This is largely due to resource contention. Clearly, adding more machines increases the overall throughput. However, contrary to expectations the non-delta update mechanism performs constantly better. This is highly likely due to the overheads incurred by the delta updates for our workloads. We discuss these overheads later in this section. Furthermore, we observe smaller throughputs when we increase the number of physical machines but when each machine has a small number of serving nodes. For example, when we have three machines and a total of three serving nodes, or when we have five machines and a total of five serving nodes. This behavior is expected because when the data is partitioned and the number of serving nodes is comparable to the number of physical machines, the likelihood of a data item to be residing on another machine is higher than with more serving nodes per machine.

2) *Replicated regions* (Figures 6(e), 6(f), 6(e) and 6(h)). Similar to the behavior of the partitioned region, for the replicated regions we observe that the non-delta updates performs better than the delta updates in most cases. Furthermore, our observations for the latencies and throughput match expectations: for a given number of physical machines, the increase of serving nodes causes the latency for operations to steadily increase and the throughput to decrease. The throughput decreases because an increase in the number of serving nodes increases the number of update acknowledgements, consequently the latency increases.

3) *HAC (Figures* 6(i) and 6(j)). For this scenario, we kept all the parameters the same but we started with two physical machines to emulate HAC zones. Even though relatively constant in throughput, we observe that the HAC-zoning performed poorly in comparison to the other scenarios. The poor performance in latency and throughput is most likely due to our implementation of emulating workloads; we were forced to introduce some synchronization of the YCSB threads to emulate signals exchanged during handoffs in HAC zones. Without the aforementioned locking, we expect that the HAC-zone could perform as the replicated region because it is largely implemented as one.

D. Discussion

The partitioned region with three copy redundancy demonstrated good scaling properties with the addition of new machines into the cluster, and retention of throughput when adding serving nodes. Replicated regions demonstrated expected properties of diminishing throughput when new serving nodes and machines were brought into the cluster. Between both the partitioned and replicated regions the non-delta update mechanism performed better than the delta counterpart. Even though the resulting throughput in our experimental HAC-zoning is poor compared to the other two types of regions, it serves as a demonstration of the capabilities of Apache Geode.

We implemented the delta update using the internal delta-interface recommended by Apache Geode. We believe that

the limited benefits of delta updates we observed during our evaluation is a consequence of the processing overhead of the delta update mechanism and our small UE object. Apache Geode serializes the delta updates and this incurs additional processing overheads. As shown in Table I and Table II, the small size of the UE object can easily fit in a packet and also in large segments created by offloading engines [21]. As a consequence, Apache Geode's delta updates in their current form do not significantly reduce the number of packets exchanged.

This work was inspired by the work of Kuhlenkamp *et al.* [22] who used YCSB to benchmark the Cassandra and HBase data stores. They show that the purpose of the data storage system, whether it be write or read optimized, has a tremendous impact on the performance of the system in different scenarios. For the workloads we tested, we observe that the Apache Geode system can perform well in both read and write intensive workloads. Furthermore, the in-memory characteristic of the Apache Geode is clearly visible in the extremely low latencies we observe in Figure 6.

V. Conclusion

A coreless mobile network opens avenues for meeting some of the requirements of future mobile networks. While a coreless mobile network can be seen as an evolution of MEC [8], it demands the network functions to be placed at the edge of the network. These network functions need to keep the state information of the UEs they serve, and we therefore discuss the benefits of using a NoSQL store for managing the state of UEs in a coreless mobile network. We explore how the eventual consistency property of NoSQL stores can be leveraged for creating HAC zones to support mobility of UEs. We compare different NoSQL stores and chose the Apache Geode store for its support of delta updates and geolocation of data. With the help of a dataset from a real cellular network and the YCSB tool, we evaluate the performance of the Apache Geode NoQSL data store. While the results of benchmarking Apache Geode are promising we believe that there is room for optimization. Specifically, it is important to explore approaches for implementing HAC-zoning to meet the mobility requirements from various verticals which demand services from coreless mobile networks.

Acknowledgment

This work has been supported by the Nokia Center for Advanced Research (NCAR) and the Tekes TAKE-5 project.

References

[1] G. Press, "Internet of Things By The Numbers: Market Estimates And Forecasts," https://www.forbes.com/sites/gilpress/2014/08/22/internet-of-things-by-the-numbers-market-estimates-and-forecasts; Referenced on 14.05.2017.

[2] L. E. Li, Z. M. Mao, and J. Rexford, "Toward software-defined cellular networks," in *Proceedings of the 2012 European Workshop on Software Defined Networking*, ser. EWSDN '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 7–12.

[3] G. Hampel, M. Steiner, and T. Bu, "Applying Software-Defined Networking to the telecom domain," in *IEEE INFOCOM Workshops*, 2013, pp. 133–138.

[4] L. Osmani, H. Lindholm, B. Chemmagate, A. Rao, S. Tarkoma, J. Heinonen, and H. Flinck, "Building Blocks for an Elastic Mobile Core," in *Proceedings of the 2014 CoNEXT on Student Workshop*. ACM, 2014, pp. 43–45.

[5] H. Lindholm, L. Osmani, H. Flinck, S. Tarkoma, and A. Rao, "State Space Analysis to Refactor the Mobile Core," in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, ser. AllThingsCellular '15. New York, NY, USA: ACM, 2015, pp. 31–36.

[6] M. Pozza, A. Rao, A. Bujari, H. Flinck, C. E. Palazzi, and S. Tarkoma, "A Refactoring Approach for Optimizing Mobile Networks," in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 4001–4006.

[7] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud RAN for Mobile Networks:A Technology Overview," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 405–426, Firstquarter 2015.

[8] ETSI, "Mobile-edge computing," *Introductory Technical White Paper*, 2014.

[9] Alcatel-Lucent, "The LTE Network Architecture: A Comprehensive Tutorial," 2009.

[10] F. Khan, "Coreless 5G Mobile Network," *CoRR*, vol. abs/1508.02052, 2015.

[11] Nokia, "Creating a new data freedom with the Shared Data Layer," http://resources.alcatel-lucent.com/asset/200238, Referenced on 14.05.2017.

[12] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 4:1–4:26, Jun. 2008.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-value Store," in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007, pp. 205–220.

[14] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's Globally Distributed Database," *ACM Transactions on Computer Systems*, vol. 31, no. 3, pp. 8:1–8:22, Aug. 2013.

[15] A. Fox and E. A. Brewer, "Harvest, Yield, and Scalable Tolerant Systems," in *Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems*, ser. HOTOS '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 174–.

[16] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, Feb 2012.

[17] D. Nowoswiat, "Managing The Signaling Traffic in Packet Core," https://insight.nokia.com/managing-lte-core-network-signaling-traffic; Referenced on 14.05.2017.

[18] I. Widjaja, P. Bosch, and H. L. Roche, "Comparison of MME Signaling Loads for Long-Term-Evolution Architectures," in *2009 IEEE 70th Vehicular Technology Conference Fall*, Sept 2009, pp. 1–5.

[19] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154.

[20] F. Ojala, "State management in coreless mobile networks," Master's Thesis, University of Helsinki, December 2016. [Online]. Available: https://helda.helsinki.fi/handle/10138/201587

[21] J. C. Mogul, "TCP Offload is a Dumb Idea Whose Time Has Come," in *Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9*, ser. HOTOS'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 5–5.

[22] J. Kuhlenkamp, M. Klems, and O. Röss, "Benchmarking Scalability and Elasticity of Distributed Database Systems," *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1219–1230, Aug. 2014.