Towards A Low-Cost Stateless 5G Core

Umakant Kulkarni
Purdue University
ukulkarn@purdue.edu

Amit Sheoran
AT&T Labs - Research
asheoran@research.att.com

Sonia Fahmy
Purdue University
fahmy@purdue.edu

***Abstract*—We propose an optimization to reduce the latency incurred by a stateless 5G control plane. The key idea is to avoid redundant database read operations. We achieve this by reading the user's state only once and sending it to successive network functions in a chain. Experimental results show that this optimization can reduce the total cost by 33% on average.**

I. Introduction

Considering the time critical applications of 5G, the telecommunications standards have set the upper limit on the control plane (core) latency to be 10 ms [1]. At the same time, the 3GPP standards recommend stateless 5G network functions (NFs). In our recent work [2], we investigated two types of statelessness, procedural and transactional, and proposed a number of optimizations to mitigate the performance cost of transactional statelessness. We found that sharing database state among 5G functions reduces the cost of transactional statelessness by an average of 10%.

In this poster, we propose a new optimization on top of database sharing, where we reduce the number of read operations to a single read operation. We achieve this by sending the data to the next NF in a service function chain (SFC) over the standard end-user state creation/modification request. Our preliminary results indicate that this single-read mechanism reduces the cost of transactional statelessness by more than 33%, which is 22% more than the previously proposed state-sharing optimization. We also show that how simply replacing a database document instead of updating it in-place can reduce the cost of transactional statelessness by more than 6%.

II. Reducing Stateless NF Latency

With transactional statelessness, an NF stores the end-user's state after completing each individual transaction. Hence, when an NF receives a trigger to modify the user's state, it first fetches its latest state from the database, then uses that to process the triggered request and finally writes the modified state back into the database.

The total cost of transactional statelessness is $4 \times n$ messages, where n is the number of NFs in an SFC. This is because we need $2 \times n$ messages for the write operation, and $2 \times n$ messages for the read operation. The goal of this work is to reduce the $2 \times n$ read cost to just two messages, by taking advantage of the sequential execution of requests and responses in an SFC which has the following properties. (1) A transaction between two NFs consists of exactly one request and the corresponding response. Procedures are executed as a sequence of transactions. (2) All NFs except the last in an SFC send a request to the next NF in an SFC on receiving an event trigger. (3) Once the last NF receives a request, it processes it and sends a response to previous NF in the SFC. The responses propagate back to the first NF. (4) When an NF receives the response for a request it sent, it stores the new state to the database. Before then, it maintains the state in the cache or main memory. (5) For a transaction between two NFs, both NFs read the exact same state from the database. This ensures that state is synchronized.

Since each NF reads the exact same state, we can avoid redundant read operations. We take advantage of the serial execution of transactions, so that only the first NF (NF_1) needs to read the user's state from the database. NF_1 can then embed this data in the underlying 5G control-plane request. The receiving side NF (NF_2) extracts the data from the request and uses it to process the remaining part of the request, as shown in Figure 1. NF_2 can then simply use the same data and embed it into the request being sent out to the next NF (NF_3). NF_2 need not perform another read operation from the database, since the previously read data remains valid because no other NF has written new data for that particular user.

The process of embedding the data into the requests continues until the last NF in the SFC. This reduces the number of database read messages to just two: one corresponding to the request from the first NF to the database and one to the response from the database to the NF.

III. Evaluation

We now compare the costs of transactional statelessness, including the previously proposed state sharing optimization, with the new embedding mechanism. We conduct our experiments on CloudLab, where we create a network with nine nodes, five of which constitute a Kubernetes cluster with one master and four worker nodes. We deploy `open5gs` v2.4.3 (an open-source 5GS C implementation) along with mongoDB v5.0.3 on the cluster. The remaining four CloudLab nodes run `UERANSIM` v3.2.5 (a C++ open-source tool): two nodes as two separate gNBs and two as UEs. All nodes are of type m510, equipped with Intel Xeon D-1548 processor supporting x86_64 architecture consisting of 16 CPUs with maximum speed of 2 GHz. The nodes run on 5.4.0-77-generic kernel with Ubuntu 20.04 and Kubernetes

978-1-6654-8353-7/22/$31.00 ©2022 IEEE

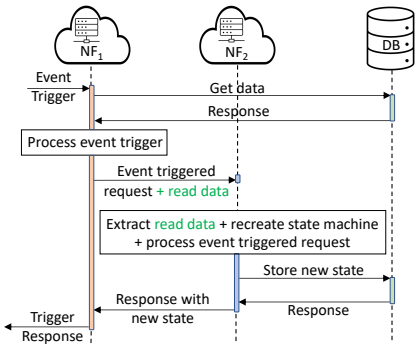

Fig. 1: Embedding data in request

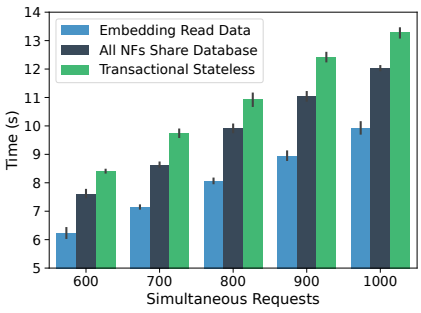

Fig. 2: Transactional stateless optimizations

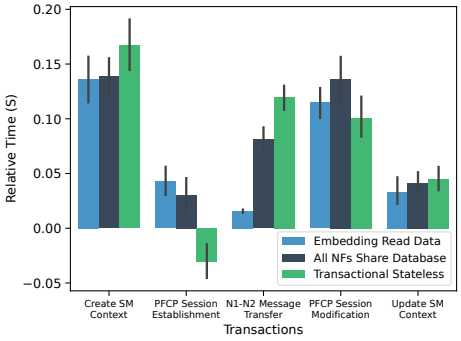

Fig. 3: Time taken by each transaction

v1.23. We use Helm charts from the publicly available repository `opensource-5g-core-service-mesh` to manage these cloud-native `open5gs` functions[1].

We experiment with the UE-initiated PDU session establishment request procedure, defined in section 4.3.2.2 of [4]. We trigger this procedure through "UE registration," which involves communication between almost all functions within the 5G system (UE, RAN, AMF, SMF, UPF, PCF, UDR, UDM, NRF, NSSF and AUSF). We compare four alternatives: (1) Fully transactionally-stateless NFs as the baseline, (2) Our previously proposed optimization where NFs share the database [2], (3) The proposed optimization of embedding read data, and (4) Using the replace API. We vary the number of simultaneous requests made between 100 and 1000, in steps of 100. We repeat each set of experiments ten times, and compute statistics for successful runs.

Figure 2 shows that the baseline transactional statelessness incurs the highest latency, followed by the state sharing optimization. Our proposed embedding optimization yields the lowest cost in terms of time taken to complete a 5G control plane procedure. In the embedding optimization, each NF sends additional data, and the serialization/de-serialization of this data incurs additional processing cost. The total time taken, however, is still the lowest. Sending a request to the database to read a document and waiting for the response is more expensive than serializing/deserializing additional data and sending it. The average reduction in time for the embedding optimization is 33% over the standard transactionally stateless baseline, and 22% over the state-sharing optimization. This saving can go up to 44% over transactionally stateless baseline, and 29% over the state-sharing optimization.

Interestingly, these cost savings are not uniform across transactions. Figure 3 shows the relative time between two consecutive transactions, averaged over 100 users. A negative value for the PFCP establishment transaction for fully transactional statelessness shows that the SMF finished processing the PFCP establishment message before the AMF finished processing the create SM context message. In the state sharing paradigm, multiple NFs try to access the user's unique state in the database, increasing simultaneous read/write operations on an object. A similar behavior is seen in case of PFCP modification, but not for the three other transactions. This implies that the SMF-UPF (N4) interface has higher transaction times for state sharing, because the SMF needs to serialize/deserialize the (shared) data twice; once for REST and once for PFCP. We can conclude that individual transaction times are influenced by simultaneous database operations as well as the amount of data and its encoding. The optimization proposed in this work is best suited for the AMF-SMF (N11) interface.

We have also explored using an alternate database API. Typically, an update request is sent to the database, with a patched document consisting of modified or new fields as key-value pairs [5]. The database checks a key-value pair against the matched document ID, and if a key already exists, it replaces that value. An alternative approach we explored is sending a new document to replace the existing document. Since the NF has already read the user's state, it concatenates the read state with the new state; i.e., patches the previously read document with the updated key-value pairs. This reduced the procedure completion time by up to 9%.

IV. Future Work

We have deployed a single microservice (container) for each 5G NF, and deployed all 5G NFs in a single namespace. As future work, we plan to break down 5G control-plane signaling cost with multiple microservices for each NF, in order to better understand the overhead of inter-microservice communication and state management as the number of microservices in an SFC increases.

[1]Our changes to the open-source tools are available [3].

References

[1] 3GPP, "Study on scenarios and requirements for next generation access technologies," Technical Report (TR) 38.913, 4 2022, version 17.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38913.htm

[2] U. Kulkarni, A. Sheoran, and S. Fahmy, "The Cost of Stateless Network Functions in 5G," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, 2021, p. 73–79. [Online]. Available: https://doi.org/10.1145/3493425.3502749

[3] "GitHub," https://github.com/UmakantKulkarni/LowCostStateless5G.

[4] 3GPP, "Procedures for the 5G System (5GS)," Technical Specification (TS) 23.502, 6 2021, version 17.1.0. [Online]. Available: http://www.3gpp.org/DynaReport/23502.htm

[5] "MongoDB Manual," https://www.mongodb.com/docs/manual/.