SETL

The Regulated Liability Network

Whitepaper on scalability and performance

Scaling RLN to 1 million TPS

powered by aws

Contents

Authors

SETL

Anthony Culligan

Nicholas Pennington

Marjan Delatine

Philippe Morel

Amazon Web Services (AWS)

Erica M Salinas

Gloria Vargas

Nilesh Dusane

Jack lu

Saqib Sheikh

Contributors

Nick Kerigan
Head of Innovation Execution, SWIFT

Tony McLaughlin
Citi

Patrick De Courcy
Payoneer

Melvyn Low
Head, Global Transaction Banking, OCBC Bank

Kwan Hoon Park
Digital Solutions, Global Transaction Banking, OCBC Bank

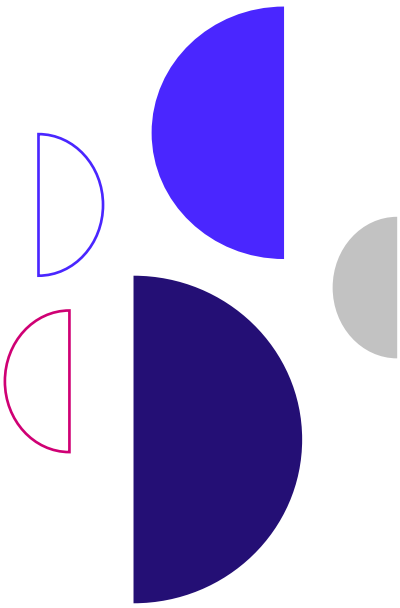
01 Introduction

By Anthony Culligan – Chief Engineer of SETL

Distributed ledger technology (DLT) can support millions of transactions per second. This core proposition has caused a high level of interest: People are used to talking about blockchains as being slow and having a high energy footprint.

This paper, which we publish today in collaboration with Amazon Web Services (AWS) focuses on speed : how can a DLT reach industrial speed, and therefore leave behind the Proof-of-Concept stage where so many DLT projects have been confined. In a follow-up study, we will focus on how the SETL blockchain is an environment-conscious choice, with a limited carbon footprint.

When reviewing the conclusions of this paper with several collaborators, it became evident that a guide would be useful for the reader to understand the design choices we made to achieve the target speed and to explain specifically how SETL blockchain compares with, and differs from, other blockchains.



First the attributes that SETL blockchain shares with other blockchains:

Stating the obvious: It is a blockchain....

SETL technology produces a globally ordered set of state updates which are grouped into sequentially numbered blocks. Each block contains the hash of the previous block thereby creating an immutable chain of state changes.

...and it is a distributed ledger

Very specifically, it is managing a subset of the liabilities of each of the participants and state updates are distributed to each ledger owner.

It has a consensus process

Each proposal to the system only succeeds if the parties to that proposal agree and sign that proposal.

Every transaction is settled real-time on-chain

There is no 'Layer 2' or 'sharding' used to reduce the burden on the core technology by rolling-up groups of transactions to increase speed. All transactions are processed and settled atomically on-chain.

Now what makes it different:

Consensus

In this technology, the inclusion of a transaction is determined by the participants in that transaction each of whom has a veto. This is orders of magnitude more efficient than proof-of-work or proof-of-stake.

Ordering

The ordering of the transactions is undertaken by a specialist component of the network – the sequencer. Other blockchains bind this into the relatively slow consensus process.

Enhanced network functionality

In most blockchains, the network between components performs no function other than connecting nodes and routing information. In our blockchain there are network components that perform functions like aggregating approvals, sequencing and routing state updates. This allows participants to focus on business tasks and not be burdened with network tasks. This approach has been adopted partially in some blockchains which implement a 'notary' service.

Parallelization of network functions

By performing certain tasks as part of the network functionality, those tasks can be horizontally scaled. For example, determining whether all parties have signed a transaction is evidentially true or not. By not pushing this function back to the business users, its technical implementation can be massively parallelized.

Optimising inter-process communication

Most blockchains connect nodes using raw tcp – the internet equivalent of a single phone line. We use massively scalable event processing of the kind used by social media platforms. These are highly distributed, resilient and allow targeted delivery of messages. Our choice in this experiment was Kafka.

Non-prescriptive approach to participation

The design assigns business functions to participants and presents a very straight forward technical connection. It anticipates that different participants will have different approval processes. For example, a decision to reject a transaction is expected to refer to a participant's own fraud/AML screening, without the need to align approval and rejection criteria with other network participants. Other blockchains seek to strictly codify approval parameters.

It has a small energy footprint

By reducing the redundant compute tasks of the participants, by avoiding wasteful consensus methods and scaling flexibly to the traffic, the network is likely to have the lowest energy consumption per transaction of all the major blockchain technologies. This will be the subject of a future whitepaper.

In Summary

When Nick Pennington, SETL's CTO, and I sat down to map this approach out, we made an active decision to design from a blank slate. We wanted to draw from the fantastic strides made in cloud scalability over the last 10 years and to create a business logic that took the best from our experience in blockchain and DLT. Most importantly, we wanted to construct a technology that can solve for real problems in regulated financial services. The team of technical experts at SETL in collaboration with AWS have brought this vision to life.

02 Abstract

In his paper 'The Regulated Internet of Value' [1], Tony McLaughlin proposes a single network to record tokens and balances, where tokens represent a promise or the liability of a regulated entity. This network, referred to as the Regulated Liability Network (RLN), would be used to issue liabilities of commercial banks, e-money providers, as well as central banks, which would, in essence, be central bank digital currency (CBDC). By supporting a wide range of issuers on the same network, coordination of value transfers between the customers of those issuers can be achieved through the movement of tokenized liabilities. To achieve this goal, the network would need to include thousands of regulated institutions and potentially process large volumes of transactions in near real-time.

In November 2021, a group of financial institutions demonstrated a working prototype based on SETL's opensource blockchain solution, as part of the Global CBDC Challenge hosted by the Monetary Authority of Singapore. The technical solution proposed by SETL was designed to scale to meet the transaction volume and throughput requirements of a practical RLN. The purpose of this whitepaper is to evaluate the scalability of that architecture.

To make this assessment, SETL collaborated with AWS to design and conduct performance tests on a simulated network aiming to validate that the SETL-based architecture could support '1000 banks, each submitting 1000 transactions per second' to the network – i.e. a throughput of one million transactions per second.

This paper begins with an overview of the RLN architecture and its various logical components. It then describes the proposed technical architecture and the performance testing methodology used. The scalability test results are then discussed in detail, including suggested areas for further design and testing.
All tests were conducted on AWS infrastructure leveraging commercially available services and SETL components purpose-built for an RLN architecture.

The paper concludes that the desired throughput is achievable. The tests undertaken were able to show that each component was scalable to at least 1m TPS and that the Kafka transport mechanism could scale to manage the inter-process communication. SETL expects to extend and refine testing as the RLN is developed into a working network in partnership with a working group of financial institutions, central banks, and other partners. SETL expects to grow the number of technology partners included in the project as value-add functionality is integrated into the network. Continued collaboration will increase the level of complexity of follow-on scenarios for subsequent phases of project testing including pilot and production scenarios.

[1] The Regulated Internet of Value, Tony McLaughlin

03 Functional Description of RLN

RLN will provide a mechanism for regulated entities to take part in a global network of tokenised regulated liabilities. Whilst such liabilities exist today and are held and traded through a variety of mechanisms, RLN will provide the following novel functionality:

a) All liabilities will be recorded on a single network
b) The network will maintain balances – i.e. ledgers of issuers and holders
c) Balances will be available to participants through a wallet system
d) End-user transactions will leverage a mapping of participant ledgers to atomically update two or more participant balances to settle payments.
e) The RLN will be programmable
f) The atomic update of all ledgers will occur in real-time.

It is anticipated that RLN will operate alongside existing account-based infrastructures and that there will be efficient mechanisms for on-boarding and off-boarding balances between RLN and traditional accounts. The following sections describe the functional components of RLN, however for the purposes of the scalability testing discussed in this whitepaper, not all the components were simulated.

Partitions

RLN will be comprised of a collection of issuer 'partitions'. An issuer (Partition Operator) can be a central bank, a commercial bank, or any institution that wishes to record a liability of that institution as a token in favour of a Token Holder. The activities of the Partition Operator will be subject to appropriate regulation, thus being a regulated liability.

Instruments

Liabilities on the RLN will be promises to deliver an Instrument to the Token Holder. Each Instrument will have a primary Partition. For example, if a commercial bank issues a token for GPB, that token represents commercial bank money in GBP, i.e. a promise to deliver GBP should the Token Holder wish to redeem it. The Bank of England, as the central bank, runs the primary Partition for GBP, and all tokens recorded in this partition are essentially a CBDC. This parallels the promises that are made to bank account holders in the account-based systems that exist today – i.e. a balance in an account at a bank is ultimately a bank's promise that an account holder can redeem their balance in GBP banknotes (notwithstanding that most often, electronic transfers are used and balances will not be 'redeemed'.

Transactions

The RLN will provide the means for Token Holders to execute certain transactions and will update balances on the RLN. An example would be the transfer of value from a Token Holder in one Partition to a Token Holder in another Partition. This would involve the extinguishing of a liability in the sender's partition and the creation of a liability in the receiver's partition. In Token terms, a token would be burned in the sender's partition and issued in the receiver's partition. Depending upon the relationship between the two Partition Operators, one or more transformations in other partitions may have to occur to settle the transaction. For example, if the two Partition Operators are both Token Holders in a common partition (say a Central Bank Partition), the RLN could be programmed to move token holdings equivalent to the newly issued

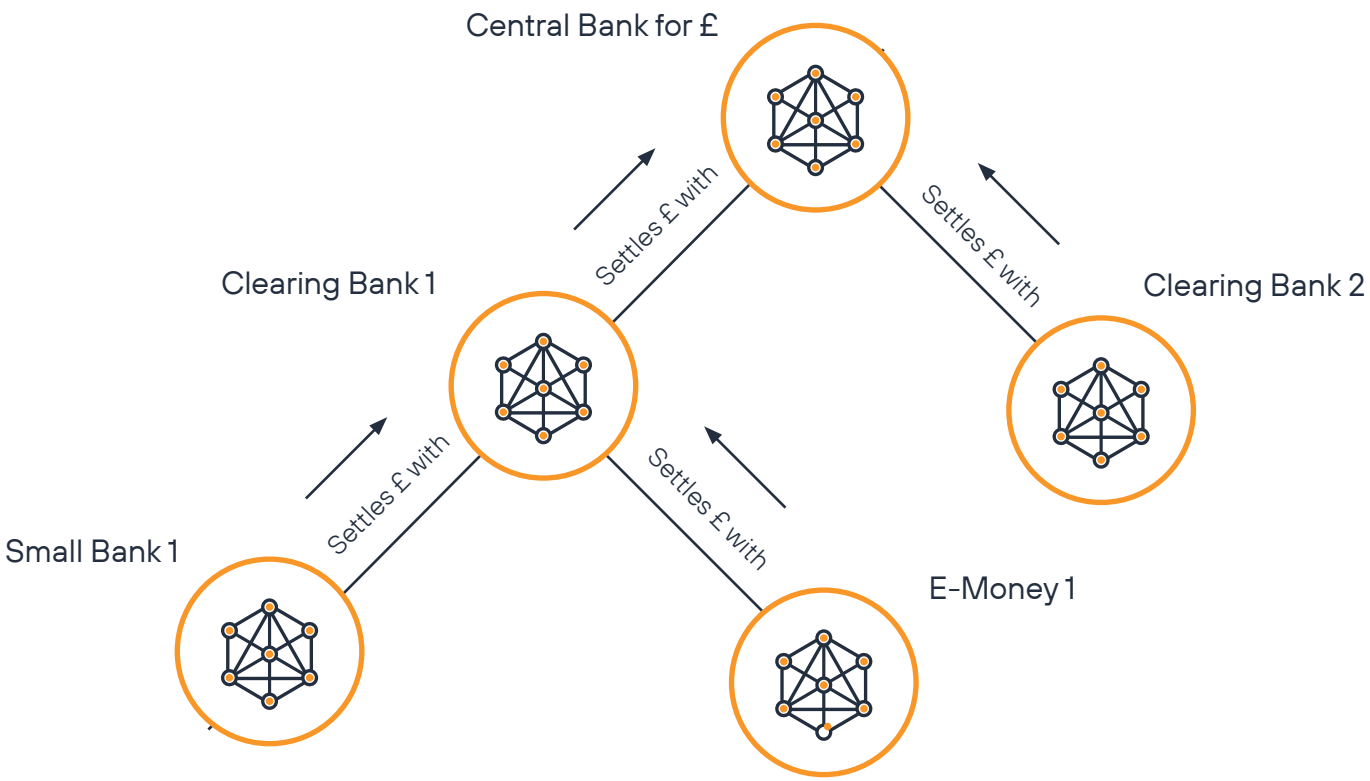
liability from the sender to the recipient Partition Operator in the common partition to settle the obligation.

The process described above is similar to the correspondent banking system which operates today. It differs, however, in that the RLN will maintain all balances for Token Holders and the Token Holder will have the ability to interact directly with the RLN to see balances and initiate transactions (notwithstanding that the Partition Operator will have ultimate control over whether a transaction is allowed to proceed). Since the RLN maintains all balances, it is uniquely able to atomically perform multiple ledger changes for the completion of a transaction in a consistent manner and in real-time.

The Network Map

The RLN will maintain a logical map of settlement connections between partitions. This map allows for the simultaneous update of all impacted balances in the various partitions. For example, if the customer of an e-money provider who banks with a commercial bank sends funds to a friend who is with another e-money provider who uses the facilities of a different commercial bank, the RLN will use the Network Map to determine which partitions need to be updated and which entities need to authorise the transaction.

Network Map for £



A transaction is first presented to the RLN as a transfer between two participants. The RLN uses the network map to create a proposal to update balances in one or more partitions to settle the transfer. The RLN will publish that proposal to the relevant Partition Operators with all of the relevant information attached. Each Partition Operator will need to agree to the proposed changes to its own partition before the changes to all Partitions are implemented in a single state change.

The Network Map is a logical network of Partitions and is independent from the physical network, which will be designed to be resilient, highly available, and scalable as needed to support a global network of Partition Owners.

Signing and Verifying

Each Partition Owner has final authority over its partition and any change to that partition must be agreed to by the Partition Owner. Proposals which transfer value between partitions will, therefore, require the agreement of the sending and receiving partitions and any other partitions impacted on the route between the sender and receiver. Agreement will be indicated with the use of verifiable digital signatures. No transaction will be implemented unless all Partition Owners who are impacted approve the state changes for their respective partition.

Sequencing Transactions

Each transaction approved by the RLN will be assigned a global ordering or block number. The purpose of this is to enable all state changes required for a single transaction to be performed atomically. All ledger updates with the same block number are defined as having been executed as part of the same state change.

Persisting State Changes

Each transaction will comprise changes to one or more partitions (partition changes). After transactions have been assigned a global block number, each partition change will be selectively distributed to be recorded in one or more persistence stores. While each state store may have different criteria in respect to what it records, all state stores will persist recorded changes in the same global order. Thus, a state store for one bank at a particular block height is guaranteed to include all of the transactions stored by transaction counterparties at the same block height.

Use Cases

There are many use cases for the RLN from retail cross-border transactions to wholesale collateral movements. Cross currency movements can be broken down into two transfers – one in the first currency and one in the second currency. Those two transfers can be included within the same proposal to ensure that both are executed atomically. A similar approach can be taken with regard to Delivery vs Payment or for complex multi-lateral clearing arrangements.

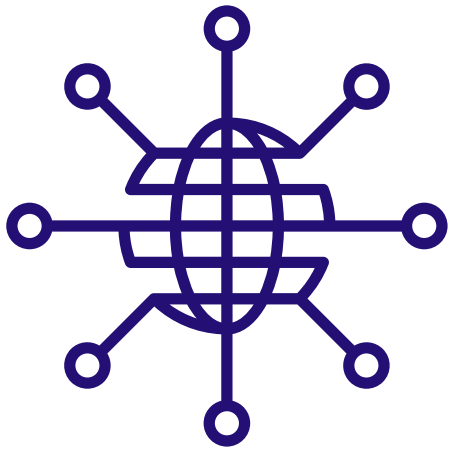
04 Technical Architecture

Key Concepts

Global State

The primary purpose of RLN is to maintain a globally sequenced set of state changes so that an unambiguous ledger of token balances can be computed. Those state changes and balances are made immutable through a chain of hashes. Users of the RLN will have permissioned access to the global state. The global state will be persisted in a highly distributed and flexible manner. For example, global state could be stored in a single or multiple state repositories. Persistence of the global state is parameterised to allow multiple repository infrastructure types, including cloud and on-premise options.

The global state will contain additional objects to assist in the operation of the network including the network map (see below) and other necessary information such as instrument identifiers.



Transaction Lifecycle

At a high level, the RLN will implement the following process for transactions submitted for settlement.

1. **Propose** – A proposal is a desired change to global state comprising changes to one or more partitions. The RLN will then determine all the required state changes to complete the proposal.
2. **Authorise** – a proposal cannot change state until it is authorised by every partition that it impacts
3. **Order** – Authorised proposals are ordered and assigned a sequential block height
4. **Persist** – Authorised and sequenced proposals are persisted by a persistence engine. That engine will feed multiple persistence stores (ledgers)

The Network Map

The network map will document settlement connections between partitions for each instrument. Each partition that issues a liability denominated in a particular asset, say GBP, will register a single settlement destination (partition) for that asset. A Partition Owner must be a member of their designated settlement partition(s) and must be authorised to hold the instrument in the settlement partition. For example, BigBankUK would nominate the Bank of

England partition as its settlement partition for GBP. EmoneyUK might nominate BigBankUK as its settlement partition for GBP. If a partition is the primary partition for an instrument it will serve as its own settlement partition. Partition owners will be able to change their settlement partition for any of its instruments at any time, but can only have one settlement partition for each instrument at a time.

The network map will be used by the Scheduler component to determine which partitions need to be changed in order to complete a value transfer over the network. For example, if a value transfer of GBP is between two partitions both of whom have nominated BigBankUK as their settlement partition, there are three partitions which need to be included in the proposal – the sender partition, BigBankUK and the receiver partition.

Proposals

A proposal can only be initiated by a Partition Owner submitting a transaction. If the Partition Owner is creating a proposal for one of its members, the partition owner is responsible for ensuring that the proposal is legitimate. A transaction submitted by a Partition Owner is expanded into a proposal by the Scheduler which determines the partitions impacted and their associated state changes by using the Network Map.

Authorisation

A proposal generated as a result of a Partition Owner transaction will include a set of state changes for one or more partitions. Each of those partitions will receive notice of the proposal and must digitally sign the proposal to indicate their agreement. The proposal should include all information or references to information required for each partition to approve the proposal. This could include identities and other objects such as information regarding the source of funds.

Participants of the RLN will be provided opensource examples of the software needed to receive proposals and to sign and submit approvals (Approval Agents), but it will ultimately be the responsibility of the partition owner to authorise proposals. The Approval Agent software may be hosted in the cloud or within the security perimeter of a Partition Owner. It is envisaged that services, such as fraud detection agents, will be developed for integration with the approval agent.

Approval Agents may need access to the current global state. Consequently, each may point at a persistence store from which they can access state information. The persistence store will publish its latest block height. Approval Agents will also have access to proposals 'in-flight' which may impact their approvals process. For example, an Approval Agent may need to be aware of proposals it has received but has yet to approve, those which it has approved but other participants have not, those which are fully approved but have yet to be persisted and those that are approved and persisted. All of these may bear on the decision to approve a new proposal it receives.

The RLN may require its members to make certain commitments to process proposals in a timely manner in order to maintain the real-time performance of the platform.

Sequencing

Each proposal will be assigned an Assembler that will be responsible for collecting and verifying the signatures of all the partitions required for the proposal to be accepted.

When a proposal has received unanimous approval, a unique hash of the proposal will be submitted to the sequencing engine by the Assembler. The Sequencer will assign the proposal a block number and a block hash. The block hash is cryptographically chained to the previous block's hash to ensure immutability. The Sequencer will pass the block hash, the proposal hash, and the block number to the State Updater component.

State Updater

The State Updater is a configurable rules-based component that collects the approved and sequenced proposals and routes each of the state change instructions to one or more persistence stores. Routes may be configured so as to filter instructions by partition or any other criteria – e.g. token holder. Each persistence store will consume and implement the state changes routed to it. The RLN will also host its own reference persistence stores that can service requests on a permissioned basis.

Process Flow

The following illustrates the processes that comprise the core elements of the RLN. It is designed to be horizontally scalable [2] wherever possible. The basic model applied is queue->process->queue so that communication between each process is guaranteed and resilient. The purpose of the flow is to take a transaction, create a proposal and to have that proposal approved by all of the relevant partitions before it is committed to every partition that it effects. This is what is required to complete, say, a correspondent banking transaction where sending, receiving and all intermediate bank ledgers need to be updated for a transfer to settle.

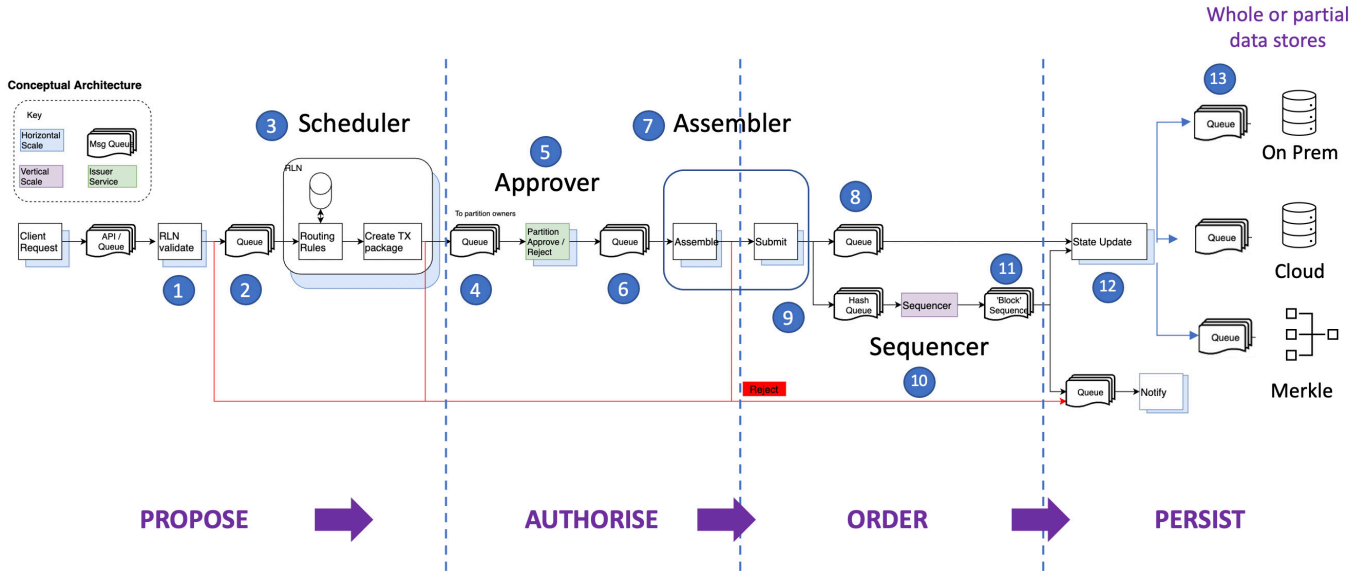


Fig 1: Core Components and Queues of RLN

[2] Horizontally scalable processes are designed so that their capacity can be increased by adding more machines running that process in parallel.

Elements

The numbered elements in Figure 1 were included in the scalability test and represent the technical components and Queues within scope:

Process	Type	Name	Description
1	Producer	Transaction Generator	Will generate transactions for the test
2	Queue	Transaction Queue	Queue of generated transactions
3	Consumer/ Producer	Scheduler	Consumes transactions from transaction queues and uses the network map to generate proposed RLN partition changes. Pushes proposal onto the Proposal Queue, assigns an assembler, and sends proposal to assembler
4	Queue	Proposal Queue	Queue of the generated proposals with a separate topic for each Partition
5	Consumer/ Producer	Approver(s) – Each partition has its own approver	Consumes proposals from its associated partition topic. If approved, signs a hash of the proposal and places it on the Assembler Queue
6	Queue	Assembler Queue	Queue of signatures related to proposals
7	Consumer/ Producer	Assembler	Consumes signatures from the Assembler queue and matches with associated proposal until a proposal has collected a full set of signatures. When completely signed, the proposal and signature block are pushed onto the Approved Proposal Queue and pushes the hash of the proposal onto the Sequence Queue
8	Queue	Approved Proposal Queue	Queue of approved proposals
9	Queue	Sequence Queue	Queue of hashes generated by the Assembler
10	Consumer/ Producer	Sequencer	Consumes the hash of the proposal from the sequence queue, creates a chained block hash by concatenating the hash of the proposal with the last chained block hash and hashing the result. Assigns block height. Pushes the proposal hash, the chained block hash, and the block height to the Hash queue

11	Queue	Queue	Queue of sequenced proposal hashes, chained block hashes, and associated block heights
12	Consumer/ Producer	State Updater	Consumes from both the Hash Queue and the Approved Proposal Queue. Unbundles the proposal into constituent partition state changes. Pushes partition state changes to one or more Persist Queues according the parameters for each queue
13	Queue	Persist Queue(s)	Queue of ordered and approved state changes for each persistence store

Table 1. RLN Elements

05 Testing

Test Scenarios

As outlined above, SETL, with AWS support, implemented a subset of the technical components of RLN to conduct scalability and resiliency testing. This testing was designed to validate the proposition that an RLN could successfully process 1 million transactions per second on the network. The tests executed assumed that each transaction would impact three partitions and would therefore comprise three state changes in total (one in each partition). In reality, certain aspects of the system will be naturally highly distributed, such as the submission of transactions and the approval process. The test scenarios devised for this exercise are based upon a smaller set of actors and are therefore more concentrated. If anything, this puts a higher burden on certain parts of the system.

Definition of Key Metrics

Each component of the RLN consumes from one or more Kafka topic, conducts a process, and produces output to a Kafka topic. For example, the scheduler component consumes a transaction from the Kafka-hosted transaction queue, conducts the process to develop a proposal for that transaction, and then produces the proposal that is pushed to the Kafka-hosted approval and assembler queues.

The key metric is the message throughput in transactions per second that can be consumed, processed and produced in respect to each component of the system. Each of the processes from proposal generation through to persistence was independently tested.

Test Components

Referring to the core components of RLN numbered in (Fig. 2), the components selected for the simulated network and scalability tests were as follows:

- **Generator:** creating (validating) transfer requests (1)(2)
- **Scheduler:** converting a transfer request into a proposal of partition changes (3)(4)
- **Approver:** Partition owner approval of associated partition changes (5)(6)
- **Assembler:** assembling the proposal and approvals into a package with signatures (7)(8)(9)
- **Sequencer:** sequencing the package hashes into a determined order (10)(11)
- **State Updater:** taking approved proposals and sequenced proposal hashes and persisting to a permissioned queue (12)(13)

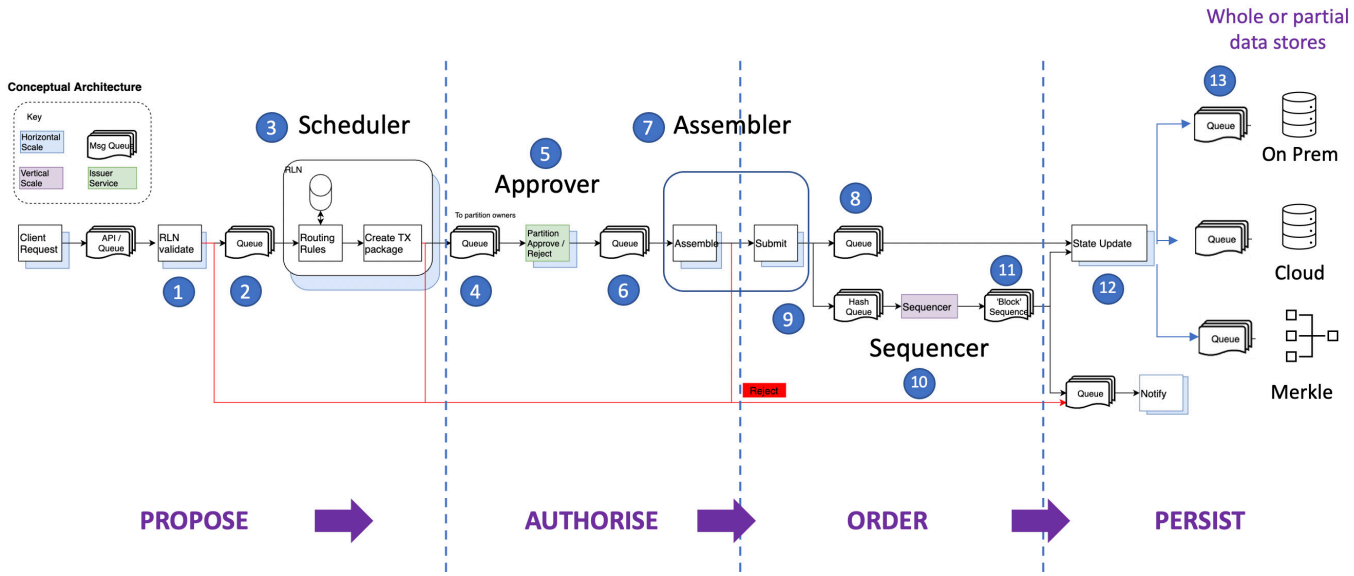


Fig 2: Core Components of RLN

The architecture components in Fig 3 represents the implementation of the selected components and the simulated network. Each of the selected components is a consumer and/or a producer and the components are connected in sequence by scalable Kafka clusters.

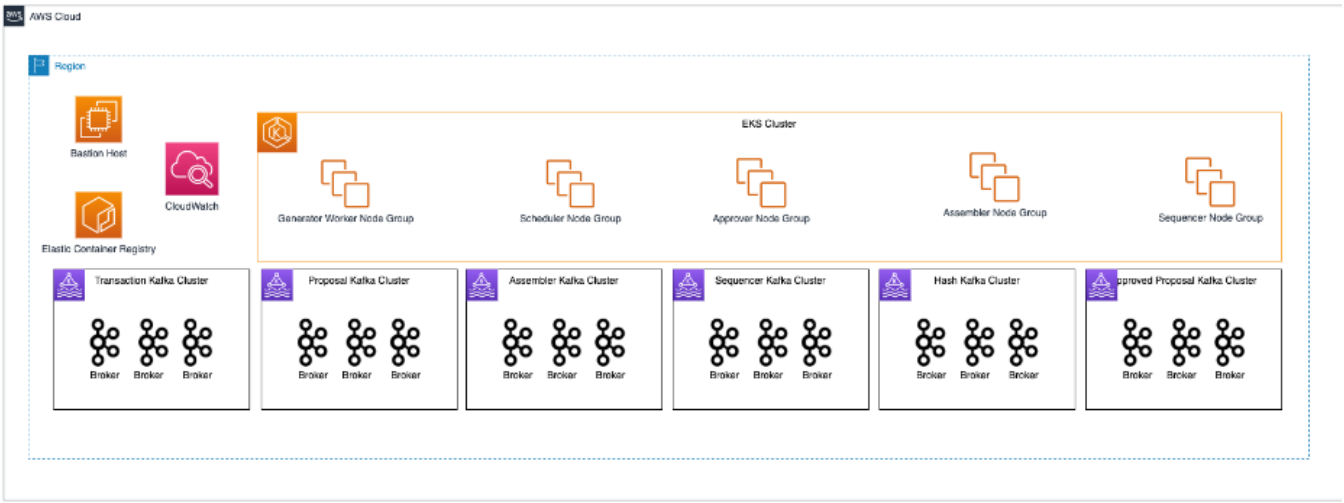


Fig 3: Simulated RLN Architecture on AWS

Component Interactions

The Transaction Generator (1) creates requests which are posted onto a Topic called 'validtx'(2).

The Scheduler (3) reads from 'validtx' (2) and publishes the full Proposals (i.e. with partition changes added) to both the Approvers (topics 'approver-0' through 'approver-<n>')(4) and the assigned Assembler (topics 'assembler-0' through assembler -<n>')(6). An approval is required from each RLN partition owner that requires a partition change. In the test, a smaller number of approvers were used than there were partitions – with each approving more transactions than they would in a production scenario.

The approvers(5) read the approval request (topics 'approver-0' through 'approver-<n>'), add signatures for their associated Partition owners, and publish the approval, with signatures, to the Assembler (topics 'assembler -0' through assembler -<n>')(6).

The Assembler(7) groups the proposal received from the Scheduler and individual approvals until the all of the required signatures have been received. Once complete, the Assembler publishes the signed proposal to the State Updater (topics '**stateupdate-0**' through '**stateupdate-<n>**')(8) and the Sequencer (topic '**sequencer**')(9)

The Sequencer (10) groups hashes, taken from the 'sequencer' topic(9), into groups, each of which represents an atomic Block of partition changes. A Block may contain one, or many, scheduled proposal hashes. The Blocks of Hashes are published to the 'block' queue(11).

The State Updater(12) consumes from the Block Queue (11) and the Approved Proposal Queue(8), filters proposals according to defined criteria (e.g. partition or instrument) and creates globally ordered Kafka queues (i.e. Persist Queues)(13) of the filtered proposals.

Test Environment

The AWS cloud was selected for these scalability tests as it provides the elasticity, agility, flexibility, and scalability needed to achieve future RLN scalability requirements. The testing was conducted entirely in the AWS Ireland (eu-west-1) region leveraging three Availability Zones. The testing region was selected purely based on team location and the testing could have been conducted in another AWS region where the relevant AWS services are available.

The infrastructure components and AWS services [3] used for conducting the testing can be divided into three groups:

1. Main message flow – Amazon Elastic Kubernetes Service (EKS), Amazon Elastic Cloud Compute (EC2), Amazon Managed Streaming for Apache Kafka (MSK)
2. Infrastructure deployment and configuration – Elastic Container Registry (ECR), EC2 (Bastion Host)
3. Infrastructure monitoring – Amazon CloudWatch.

This whitepaper focuses on the infrastructure configuration for the Main message flow in the following sections. Table 1 contains the detailed specifications used for the testing:

[3] Links to relevant AWS services: Amazon Elastic Kubernetes Service (EKS), Amazon Elastic Cloud Compute (EC2), Amazon Managed Streaming for Apache Kafka (MSK), Elastic Container Registry (ECR), Amazon CloudWatch

Table 2: Testing specifications

	Type	vCPUs	Memory (GiB)	Disk (GiB)	Network Performance	Quantity
Generator Node Group	c5.4xlarge	16	32	100	Up to 10 Gigabit	2
Scheduler Node Group	c5.xlarge	8	16	100	Up to 10 Gigabit	100
Approver Node Group	c5.xlarge	4	8	100	Up to 10 Gigabit	100
Assembler Node Group	c5.xlarge	8	16	100	Up to 10 Gigabit	100
Sequencer Node Group	c5.2xlarge	16	32	100	Up to 10 Gigabit	1
State Updater Node Group	Kafka.m5.4xlarge	16	32	500	10Gbps	10
Transaction Kafka Cluster	Kafka.m5.4xlarge	16	64	500	10Gbps	6
Proposal Kafka Cluster	Kafka.m5.4xlarge	16	64	500	10Gbps	6
Assembler Kafka Cluster	Kafka.m5.4xlarge	16	64	500	10Gbps	3
Sequencer Kafka Cluster	Kafka.m5.4xlarge	16	64	500	10Gbps	3
Hash Kafka Cluster	Kafka.m5.4xlarge	16	64	500	10Gbps	3
Approved Proposal Kakfa Cluster	Kafka.m5.4xlarge	16	64	500	10Gbps	3

The purpose of having distinct EC2 node groups for different RLN components was to accommodate the scaling characteristics for each component. Also of note is that each EC2 in the node group houses 1 or 2 Kubernetes Pods rather than having a few large EC2 nodes with many Kubernetes Pods. This made it much easier to fine tune and identify bottlenecks in the system. The same philosophy was applied to the Kafka clusters Each RLN Queue component had its own dedicated Kafka cluster. The software versions used for the key components were Kafka version 2.6.2 and Kubernetes version 1.20.

Test Methodology

Hypothesis

The hypothesis being tested is that the components described above can be deployed in a horizontally scalable configuration as seen in Table 3 to achieve an end-to-end throughput of 1 million transactions per second.

The proposition was tested by:

a) Measuring the performance of each component individually. Those were;
   a. (1) Generator
   b. (3) Scheduler
   c. (5) Approver
   d. (7) Assembler
   e. (10) Sequencer
   f. (12) State Updater

b) Testing the combined performance of a cluster of each component to determine if each task can be scaled to reach the target throughput

c) Testing the ability of the proposed Kafka clusters to service the replicated components

d) Testing whether the single sequencer component could consume, process, and produce 1 million approved proposal per second organising them into blocks.

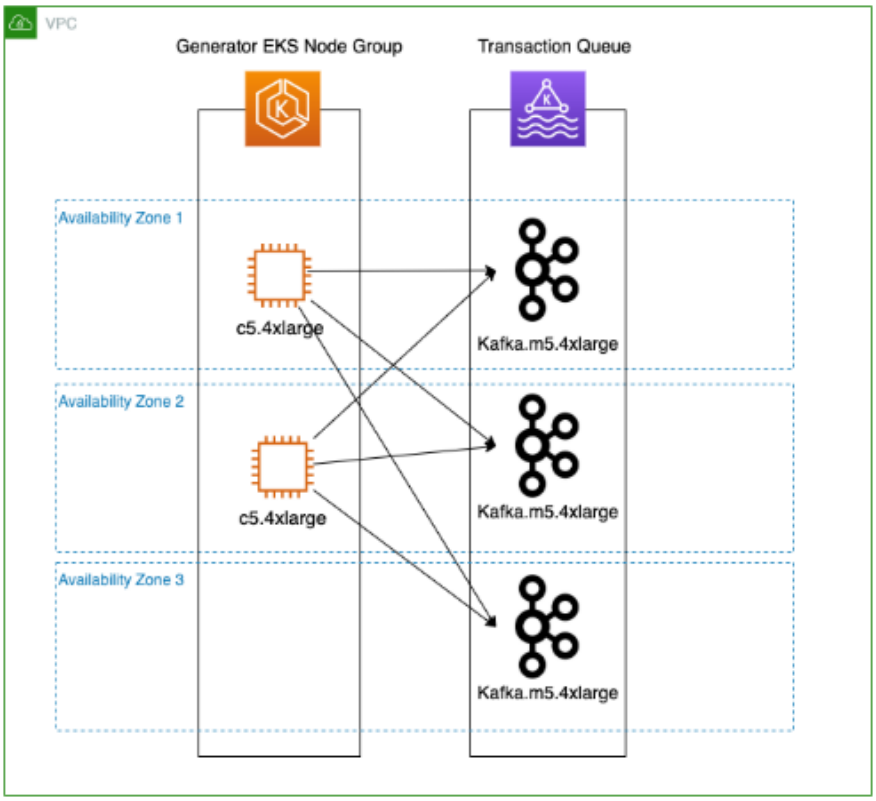
Table 3: Testing configurations

Process	Type	Name	Number of Instances	Pod	Topic	Partition
1	Producer	Transaction Generator	2	4		
2	Queue	Transaction Queue	3		1	100
3	Consumer/Producer	Scheduler	100	100		
4	Queue	Proposal Queue	3		10	10 each
5	Consumer/Producer	Approver	100	100		
6	Queue	Assembler Queue	3		10	20 each
7	Consumer/Producer	Assembler	100	100		
8	Queue	Approved Proposal Queue	3		10	10 each
9	Queue	Sequence Queue	3		1	10
10	Consumer/Producer	Sequencer	1	1		
11	Queue	Hash Queue	3		1	10
12	Consumer/Producer	State Updater	10	10		
13	Queue	Persist Queue	3			

In this proposed configuration, only the Sequencer component is vertically scaled, while other components can scale horizontally.

Detailed Configurations

Transaction Generator (producer) / Transaction Queue



The transaction Generator acted as the customer entering transactions into the system. The test was simulated with 100 assetcount and 1000 partitioncount. The kafka producer property buffer.memory was changed to 536870912 to increase the producer TPS:

```
- name: buffer.memory
  value: '536870912'
```

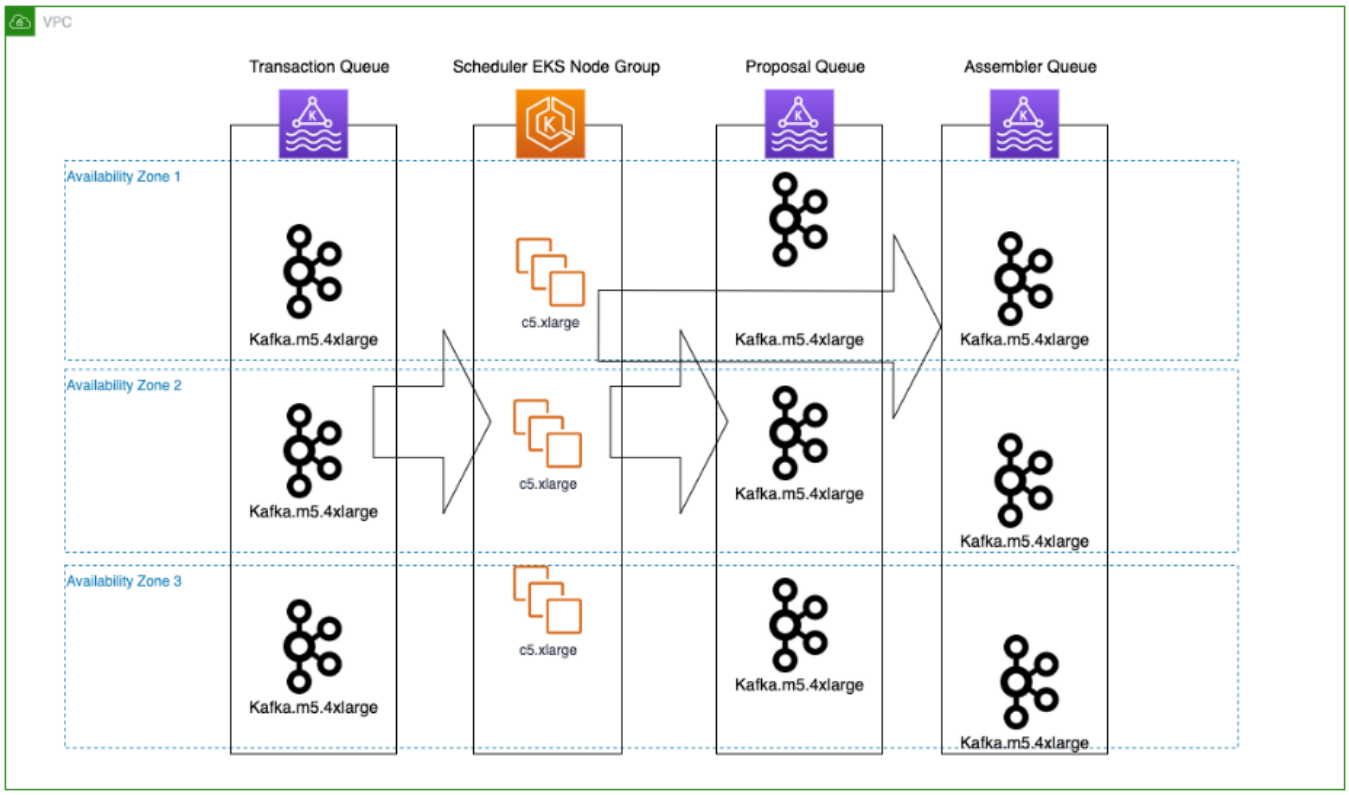
"validtx" topic was created on Transaction Queue with replication-factor 3 and partitions 100.

```
bin/kafka-topics.sh --create --zookeeper "z-3.rln-ingestion-msk-clus.dy8o5n.
c3.kafka.eu-west-1.amazonaws.com:2181,z-2.rln-ingestion-msk-clus.dy8o5n.c3.kafka.
eu-west-1.amazonaws.com:2181,z-1.rln-ingestion-msk-clus.dy8o5n.c3.kafka.eu-west-1.
amazonaws.com:2181" --replication-factor 3 --partitions 100 --topic validtx
```

Kakfa cluster configuration (same settings for all Kafka clusters)

```
auto.create.topics.enable=false
default.replication.factor=3
min.insync.replicas=2
num.io.threads=8
num.network.threads=5
num.partitions=1
num.replica.fetchers=2
replica.lag.time.max.ms=30000
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
socket.send.buffer.bytes=102400
unclean.leader.election.enable=true
zookeeper.session.timeout.ms=18000
log.retention.hours=24
log.retention.bytes=1073741824
log.segment.bytes=107374182
```

Scheduler (producer/consumer) / Transaction Queue / Proposal Queue/Assembler Queue



The Scheduler component consumed messages from validtx topic (Transaction Queue), and produced messages to both the approver topics (Proposal Queue) and assembler topics (Assembler Queue). The scaling technique for the scheduler was to have dedicated compute (one EC2/Pod) that listened to one Kafka partition in the validtx topic. The following is the consumer property:

```
- name: max.partition.fetch.bytes
  value: '419430400'
- name: fetch.min.bytes
  value: '1048576'
- name: receive.buffer.bytes
  value: '1048576'
- name: max.poll.records
  value: '10000'
```

Tuning on gzip compression on the consumer solved an initial network bottleneck for the downstream Kafka cluster.

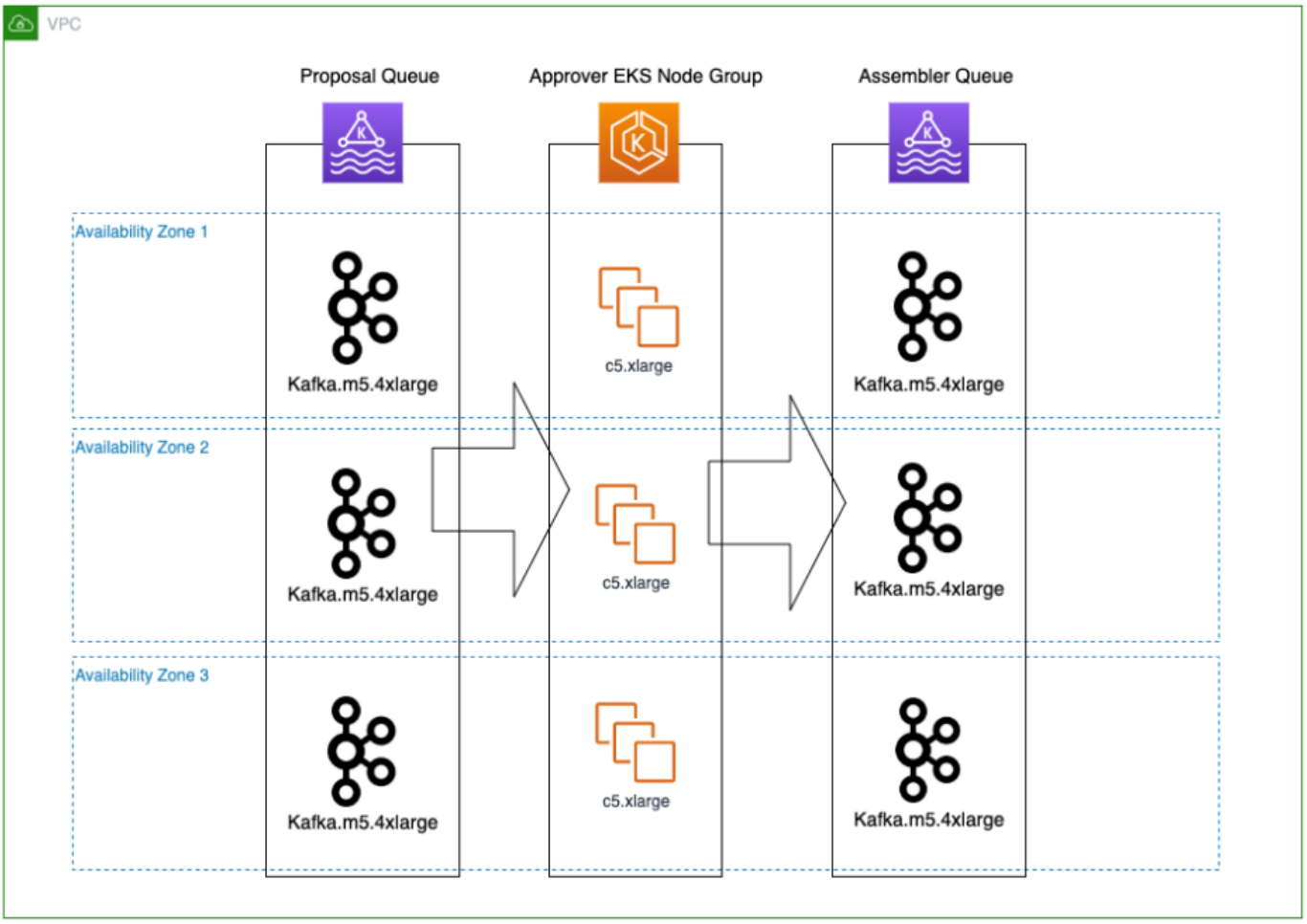
```
- name: compression.type
  value: 'gzip'
```

Similar to the validtx topic, here is the script for creating the first approver and assembler topic that ends with -0 suffix.

```
bin/kafka-topics.sh --create --zookeeper "z-1.rln-proposal-msk-clust.47kixt.
c3.kafka.eu-west-1.amazonaws.com:2181,z-3.rln-proposal-msk-clust.47kixt.c3.kafka.
eu-west-1.amazonaws.com:2181,z-2.rln-proposal-msk-clust.47kixt.c3.kafka.eu-west-1.
amazonaws.com:2181" --replication-factor 3 --partitions 10 --topic approver-0
```

```
bin/kafka-topics.sh --create --zookeeper "z-1.rln-assembler-msk-clus.rc1fmj.
c3.kafka.eu-west-1.amazonaws.com:2181,z-2.rln-assembler-msk-clus.rc1fmj.c3.kafka.
eu-west-1.amazonaws.com:2181,z-3.rln-assembler-msk-clus.rc1fmj.c3.kafka.eu-west-1.
amazonaws.com:2181" --replication-factor 3 --partitions 10 --topic assembler-0
```

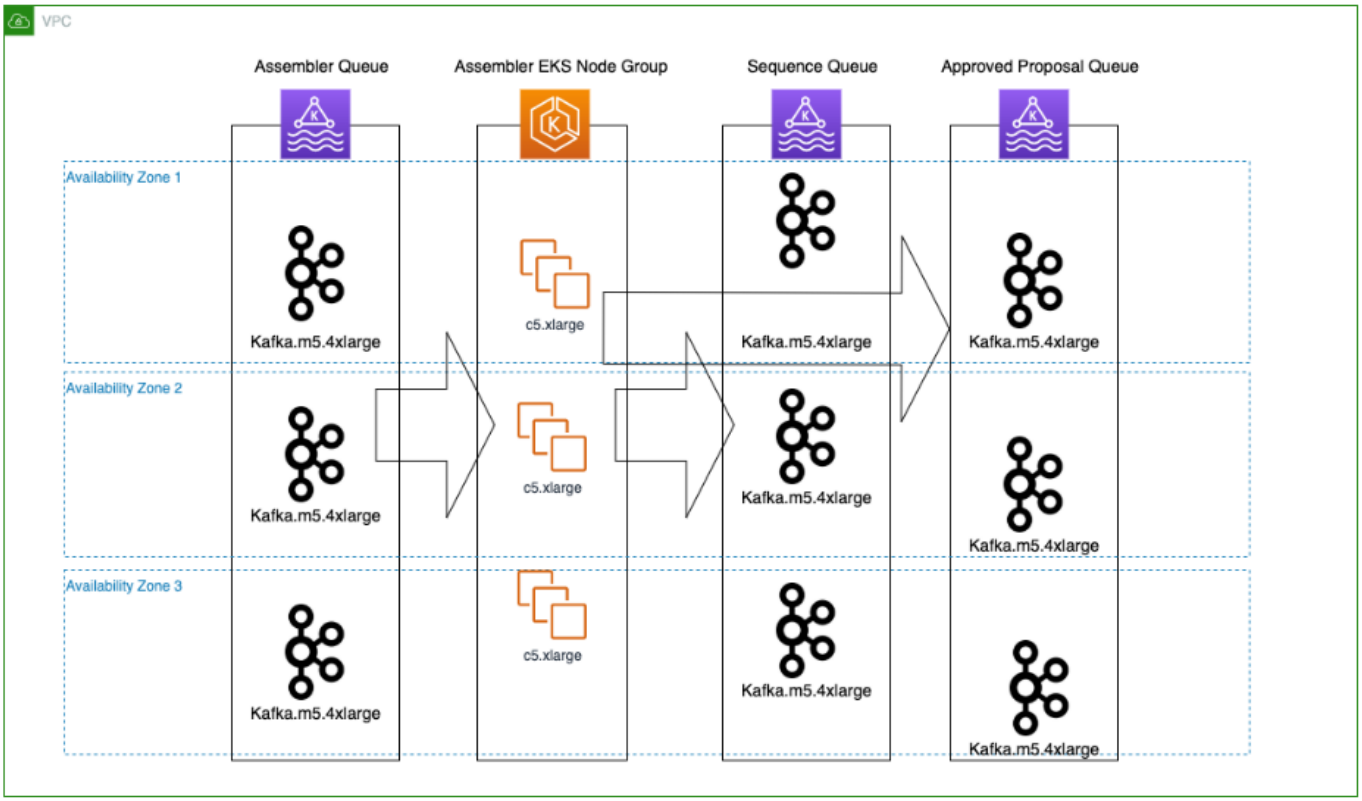
Approver (producer/consumer) / Proposal Queue/Assembler Queue



The Approver component consumes messages from approver topic (Proposal Queue), and produces messages to assembler topics (Assembler Queue). The scaling technique for the approver was very similar to the Scheduler where the instance only listened to its own partition. The difference was that the proposal queue had 10 topics, while the transaction queue only had 1 topic. The Approver would randomly pick a topic and a partition to listen to. The test was structured so that each approver had a similar load. In reality, loads will differ. The test, however, put a very high load on each approver as each serviced multiple partitions. Also, in production, the approval process for any individual partition could be horizontally scaled.

The kafka consumer and producer configuration of the Approver was the same as the Scheduler, therefore same considerations apply.

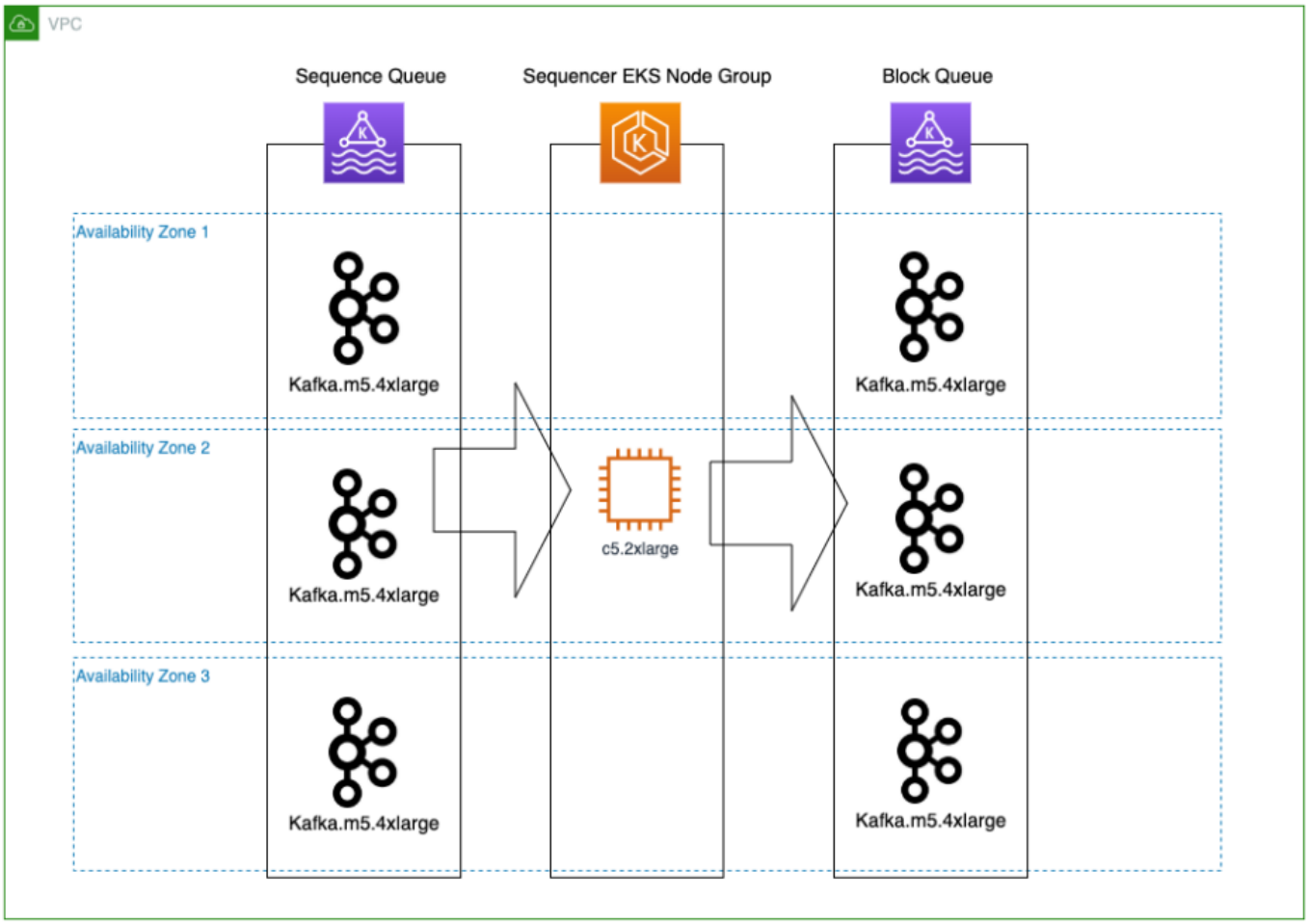
Assembler (producer/consumer) / Transaction Queue / Sequence Queue / Approved Proposal Queue



The scaling considerations for the Assembler follow the same combination of the last two blocks: 1/Consume from multiple topics with multiple partitions and 2/ Produce to two Kakfa clusters.

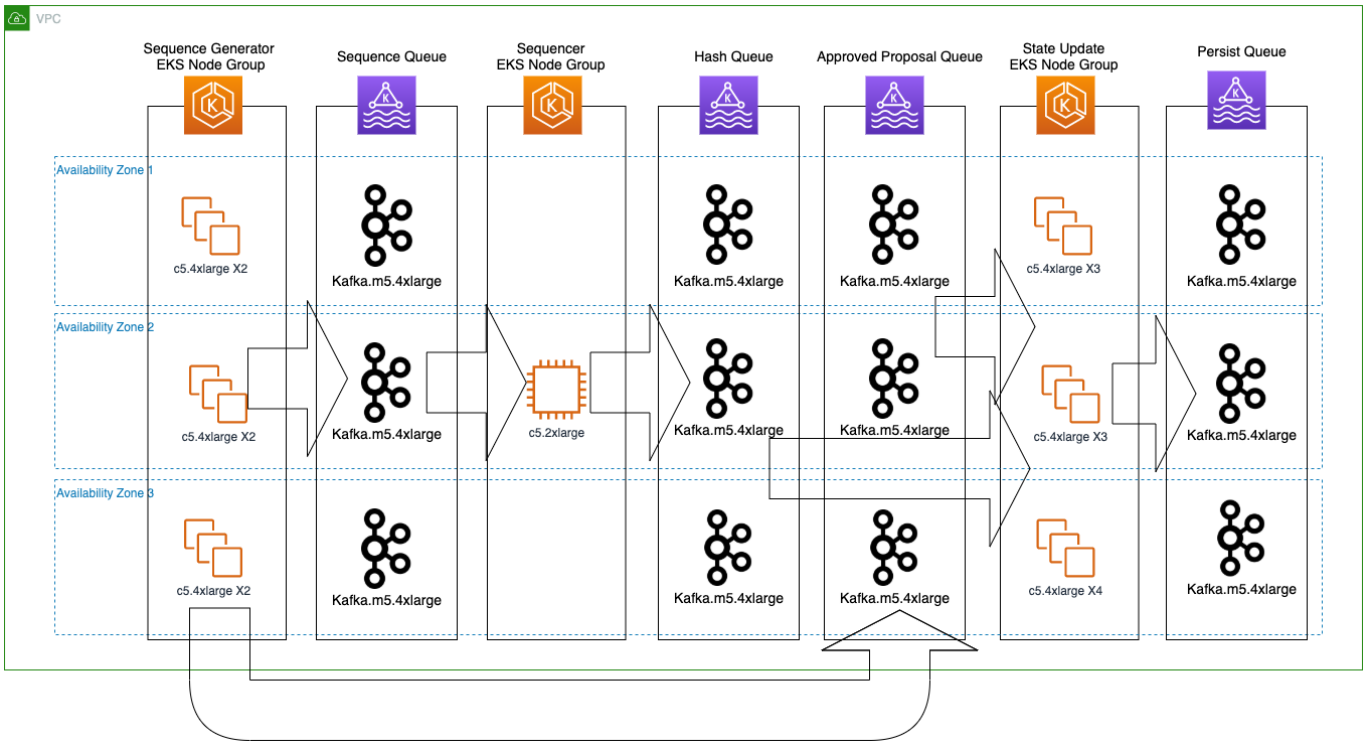
Again, the configurations for the Scheduler component apply here.

Sequencer (producer/consumer)/Sequence Queue/ Hash Queue



This component, as written for this test, cannot be scaled horizontally due to the nature of the Sequencing algorithm deployed. However, without any special tuning on the Kafka consumer configuration or any significant increase of the sizing for the EC2 instance, it was easily able to achieve 1 Million TPS throughput. The number of partitions of the sequence topic was set to 200 for the purpose of testing.

State Updater



Testing the State Updater throughput relies on the generator that simulates the behaviour of the Assembler. This generator sends Transactions IDs to the Sequencer Queue and sends approved transactions to the Approved Proposal Queue. Each State Updater pod/EC2 listens to its own assigned Approved Proposal topic (one-to-one mapping), while each compute node receives all the sequenced hashes from the Hash Queue. Achieving 1 million tps throughput requires 6 nodes configured with c5.4xlarge instances as the generator, 1 node configured with c5.2xlarge instances as the Sequencer, and 10 nodes configured with c5.4xlarge as the State Updater. Sequence, Hash, and Approved Proposal topics have 10 partitions each.

Test Results

Individual component tests

Table 4 summarizes the test results for each of the individual network components tested. The element numbers are aligned to those in Table 1 in Section Elements.

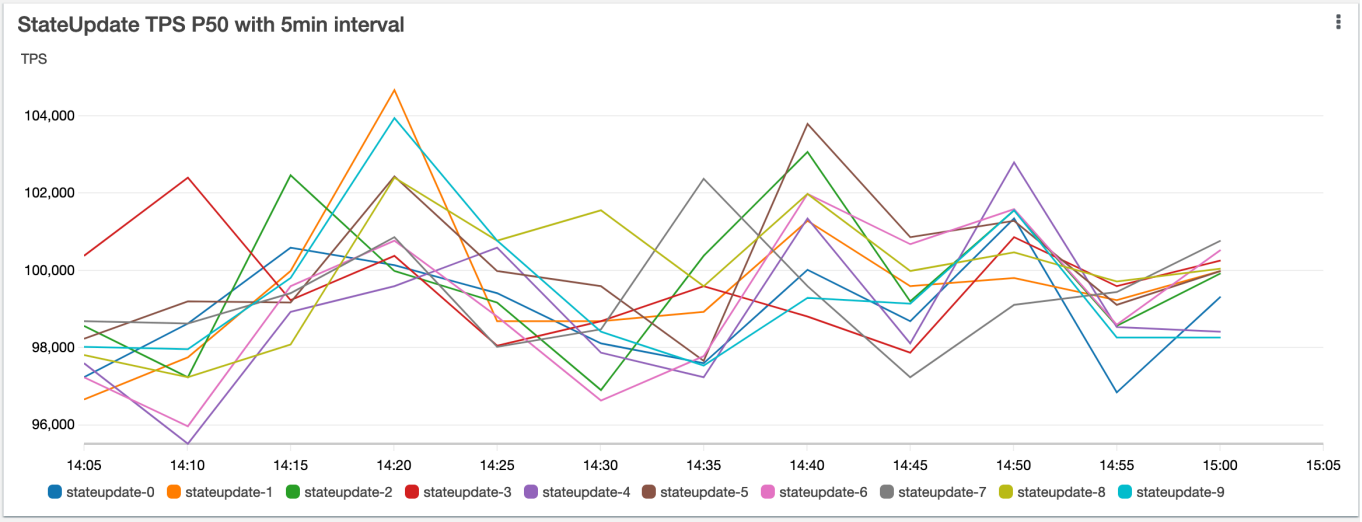
Element	Component	Single Instance (tps)	# of Instances	Multiple Instance (tps)
1	Generator	~629,219	2	1,258,438.4
3	Scheduler	~11,019	100	1,101,928
5	Approver	~10,348	100	1,034,868
7	Assembler	~10,691	100	1,069,100
10	Sequencer	1,052,845	1	NA
12	State Updater	~100,256	10	1,002,569.4

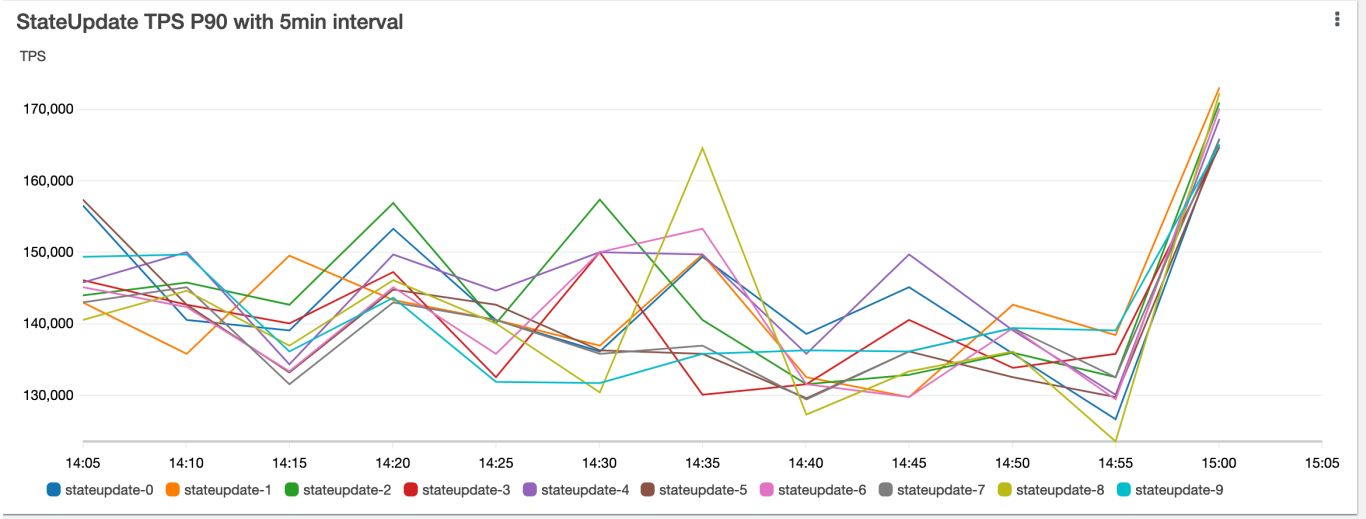
Table 4. Test result per individual component

As demonstrated, each component was able to sustain over 1 million tps through the provision of multiple instances running. The Scheduler, Approver and the Assembler required the most instances, however, even a single instance was able to achieve over 10,000 tps. Thus, the total throughput each component can process is over 1 million tps.

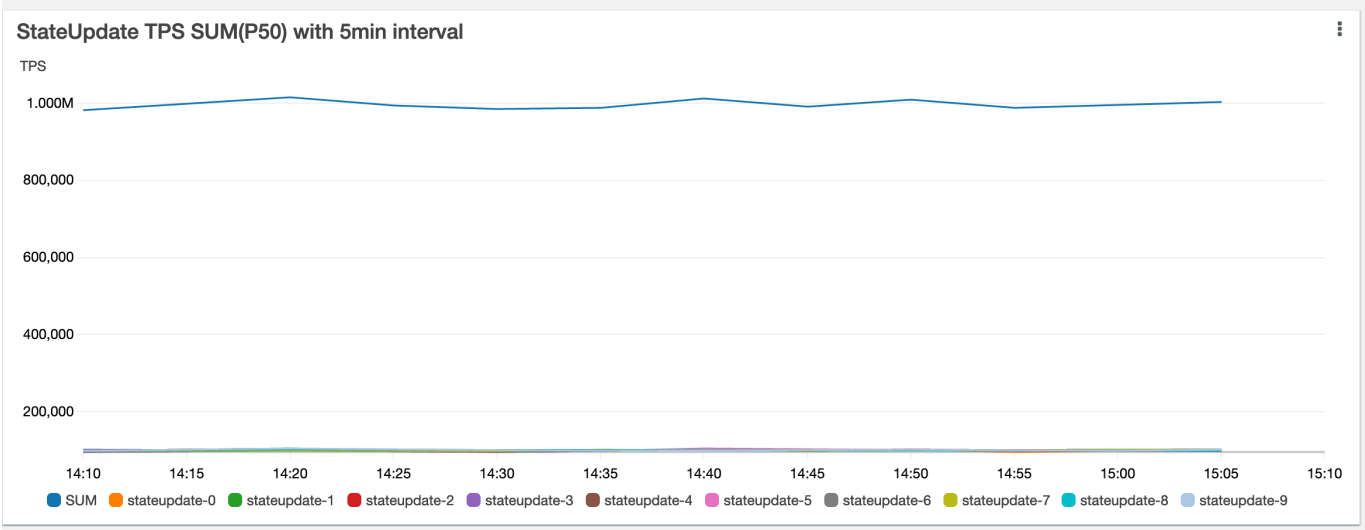
End to End tests

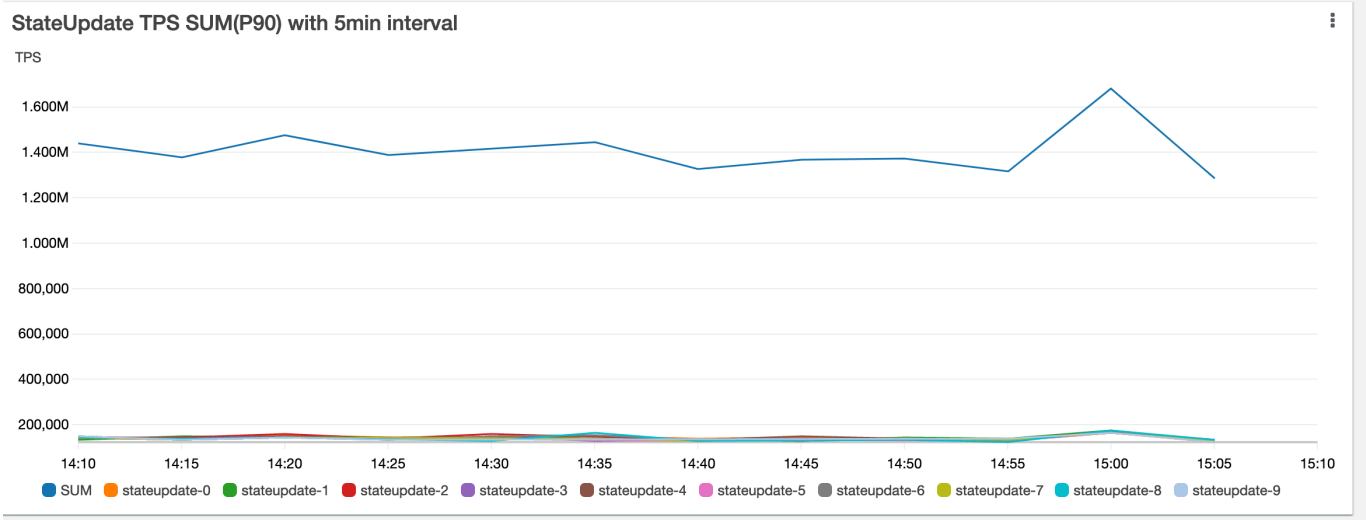
End to end tests were also performed with all components integrated to simulate the RLN network as described in section Test Components. The End-to-End tests were executed over periods of 1 hour with sustained traffic generation of ~1,000,000 tps. The tps data points were collected every 5 seconds and sent to Amazon CloudWatch for visualization. Figure X and Y below show the metrics graphs for P50 (median) and P90 (90 percentile) of the hash tps processed for the each of the State Updater instances.





Graphs X and Y below show the aggregated sum of P50 and P90 of all the State Updater instances. It clear that the RLN network can sustain around 1 million tps end-to-end.


StateUpdate TPS SUM(P50) with 5min interval
TPS
1.000M
800,000
600,000
400,000
200,000
14:10
14:15
14:20
14:25
14:30
14:35
14:40
14:45
14:50
14:55
15:00
15:05
15:10
SUM
stateupdate-0
stateupdate-1
stateupdate-2
stateupdate-3
stateupdate-4
stateupdate-5
stateupdate-6
stateupdate-7
stateupdate-8
stateupdate-9


StateUpdate TPS SUM(P90) with 5min interval
TPS
1.600M
1.400M
1.200M
1.000M
800,000
600,000
400,000
200,000
14:10
14:15
14:20
14:25
14:30
14:35
14:40
14:45
14:50
14:55
15:00
15:05
15:10
SUM
stateupdate-0
stateupdate-1
stateupdate-2
stateupdate-3
stateupdate-4
stateupdate-5
stateupdate-6
stateupdate-7
stateupdate-8
stateupdate-9

06 Conclusion

Testing Findings

The tests performed with the technical components and architecture described in the previous sections demonstrate the feasibility to scale the proposed RLN network to process 1,000,000 tps at every component as well as in the overall process flow while maintaining the business requirements of atomicity and finality. The components leveraged for this test and the reference architecture implemented will serve as the base for the continued evolution of the RLN concept to support production-ready use cases.

The technical implementation tested for scalability was deployed using the AWS global infrastructure of regions and availability zones, which inherently enabled a resilient environment.

The successful scalability test of the RLN components for the simulated scenario gives the community a solid foundation to continue evolving the proposed architecture towards a production-ready infrastructure that can support the volumes of partitions and transactions required to support a global regulated liabilities network.

Security and Resiliency

Whilst this study was designed to review capacity and throughput, the tests did replicate Kafka partitions across the three Availability Zones in the selected AWS region demonstrating that the network can utilize geographical resilience within the AWS network whilst meeting the throughput benchmark. A more in-depth study focused on security and resilience would be required before this network architecture could be considered production ready.

Areas for Evolution

The architecture described in this paper represents the initial implementation of RLN components, however several areas have been identified for further design and testing:

- Digital Signing and Verification: All operations in RLN require the agreement of the sending and receiving partitions and also any partitions that are impacted on the route between the sender and receiver. Agreement will be indicated with the use of verifiable digital signatures. Which digital signatures to leverage, as well as the standards to be implemented, are a key area for collaboration by potential partition owners.

- Interoperability with Partition owners' transactional environment: Secure integration patterns from RLN to the partition owners' transactional environments in cloud and/or on-premise.

07 Perspectives

SWIFT

The RLN

As a global industry co-operative, collaboration is in SWIFT's DNA. And innovation is at our heart. For more than 40 years, SWIFT has brought together financial institutions from around the world to identify industry-wide challenges, and developing solutions that benefit the entire global financial system. Our approach to digital currency exploration is no different. SWIFT has embarked on a journey to look at the opportunities and challenges presented by CBDCs and a wider digital asset ecosystem on our business and our members. In May 2021, we published a paper alongside Accenture, Exploring central bank digital currencies: How they could work for international payments, setting out the roles SWIFT could play in a world with CBDC, as well as detailing the experimentation we have conducted to date.

SWIFT welcomes the inclusive approach taken in developing the RLN, bringing together a range of participants from across the financial community. We also note the desire to both operate within current regulatory frameworks and requirements, and to leverage the correspondent banking model that SWIFT enables today. Alongside technology, establishing clear roles and governance will important to the success of this initiative. SWIFT stands ready to support its clients in further development and help ensure this initiative meets the needs of our unique global community.

The Technology

SETL, in collaboration with AWS, have tested an approach that leverages the advancements in cloud computing, as well as use of a blockchain, to create a new network. Dynamic and secure scaling, together with containerisation, has benefits in resilience and availability. These techniques and mature foundational technologies are employed by SWIFT in our new platforms, and within the infrastructures of many of our members. The exercise performed to demonstrate scalability serves to show what is possible.

Potential areas for further research include:

- The incorporation of the standardised ISO20022 messaging format, which would likely help adoption
- The participants need to rely upon a trusted PKI system in order to benefit from no-repudiation and other authentication characteristics. It would be good to understand how this could be incorporated into the system
- The experiment completes at state changes being delivered to an ordered Kafka queue. How those state changes are persisted into one or more databases or ledgers could be investigated further

Overall, SWIFT regards this type of exploration as beneficial to the financial community and supports further collaborative innovation.

Citi

The Regulated Liability Network (RLN) concept is an attempt to upgrade national currencies using the latest methods, including distributed ledger technology. RLN is different from existing payment systems which are built on messaging between institutions. It is the idea of a regulated Financial Market Infrastructure (FMI) where the values are represented and processed within the network itself. If that network can operate 24*7, has finality of settlement built in, can represent multiple types of asset and is programmable through 'smart contracts' then we have something new to world... the next generation of settlement based on national currencies. We encourage these efforts to drive technical assessment.

Payoneer

Payoneer welcomes the prospect of continuing to work with SETL, SWIFT, Citi and others to bring forward the novel concept of a network of regulated liabilities on the blockchain. We fully support the approach taken of defining and testing a broad range of use cases, whilst demonstrating the robustness of the supporting technology.

The ultimate test will be in the value that can be actually delivered to customers. The idea of central bank and privately issued tokens on a blockchain that is interoperable, programmable and has the potential to remove friction between silos shows real promise. The prospect of being able to issue such tokens within tried and tested regulatory frameworks that exist today could also remove a key hurdle to broad adoption by banks, payment providers, other financial institutions and their customers around the globe. Finally, a shared technology infrastructure that can operate at scale and is demonstrably – to regulators and financial institutions – robust and safe will address another key barrier to broad adoption.

Further areas of focus should include:

- Deploying a focused proof of concept with real-life transactions within an existing regulatory framework;
- Exploring how interoperability can be achieved across currencies and jurisdictions.

OCBC Bank

The RLN provides a balanced way for central banks and commercial banks to constructively work together to introduce CBDCs in the financial system today. This study demonstrates that the RLN can be built with the speed and scalability it requires. We would like to see how the various use cases can be further developed and come alive on the RLN.

08 Appendix A: Technical Description of Components

Component Details

The technical aspect of each component is described in Appendix A below.

Template

All of the Java projects were based on a broadly similar approach: They were Gradle projects and used ENV variables for configuration.

There were a few parameters specific to individual projects, which are detailed below, and they all had a consistent approach to configuring the Kafka Consumers and Producers.

Kafka Configuration

Kafka clients have a number of configuration parameters, which can be found in the following links: https://kafka.apache.org/documentation/#producerconfigs and https://kafka.apache.org/documentation/#consumerconfigs
Any of these settings could be provided as an environment variable where the '.' values of the Kafka configuration names are replaced with either '-' or '_' characters.

Parameters provided in this way applied to both Kafka Consumers and Producers if they were valid configuration properties for the respective components. Consumers and Producers did not have many configuration properties in common, generally just those relating to common areas such as SSL configuration and Bootstrap Brokers.

Some of the RLN components used multiple Consumers or Producers. These components could be targeted individually by prefixing the Env variable with a specific tag, detailed in each project.

The Specific tagged properties took priority over general configuration values if there was a conflict. By default, the projects looked for a broker '`kafka:9092`'. This could be changed by configuration or host file as needed. Most often, the Kafka bootstrap broker needed to be set by defining the BOOTSTRAP_SERVERS environment variable.

Threads

Generally, each project attempted to fully utilise as many processors available to it by creating a worker thread for each logical processor, as identified by the Java '`Runtime.getRuntime().availableProcessors()`' system call.

The number of threads could be overridden by use of an Env parameter, detailed in each component below.

Transaction Generator (1)

Purpose	The Generator will create transfer requests. By default, it will create an unlimited quantity, as fast as it can, and post them to a Kafka topic 'validtx'. The Generator will create arbitrary transfers of a random quantity of one of <**assetcount**> assets, between Random clients of <**partitioncount**> partitions, where <**assetcount**> defaults to 100 and <**partitioncount**> defaults to 1000. It is anticipated that these defaults will be sufficient for the purposes of the performance test.
Environment Parameters	assetcount — Default to 100. Sets number of Assets to use. Named 'Asset-<n>'. partitioncount — Defaults to 1000. Sets the number of Partitions to use. 'Partition-<n>'. txcount — Defaults to -1 (unlimited). May be set to produce a defined number of requests targettps — Defaults to -1 (unlimited). May be set to target a specific tps generation rate. threadcount — If set, will specify the number of worker processes writetopic — Defaults to 'validtx'. May be changed to specify a different Kafka Topic.

Scheduler (3)

Purpose	The Scheduler calculates partition updates from a transfer request then publishes to the Assembler and One-or-more Approver topics. Write topics cannot be changed. They are always in the form 'approver-<n>' and 'assembler-<n>'.
Environment Parameters	assetcount — MUST match values used in generator partitioncount — MUST match values used in generator threadcount — If set, will specify the number of worker processes readtopic — Defaults to 'validtx'. May be changed to specify a different Kafka Topic. approvertopiccount — Defaults to 10. Defines how many approver topics are used (0 to n-1) assemblertopiccount — Defaults to 10. Defines how many assembler topics are used (0 to n-1)

Consumer	Specific Consumer config (such as Kafka Bootstrap) can be set with the VALIDATOR_ prefix, though this can be set using the default parameters and specific values for the Producers if required.
Producer	There are two Kafka Producers created: One for publishing Approvals and one for publishing to the Assemblers. Specific Approvals config (such as Kafka Bootstrap) can be set with the APPROVALS_ prefix. Specific Assembler config (such as Kafka Bootstrap) can be set with the ASSEMBLER_ prefix.

Approver (5)

Purpose	The Approver takes approval requests from 'approver-<n>' topics, creates signature messages based approvals received from the required partition owners, and then publishes signature packages to the 'assembler-<n>' topics. The required assembler topic is not configured here as it is provided in the message coming from the scheduler.
Environment Parameters	For testing purposes, there is a defined sequence of topics that the Scheduler will post messages to, named in the style 'approver-<n>' where n is a number in the defined range. Where the largest n is less than 1000, some partitions are approved by the same approver, carrying a higher workload than they would in real life. For example, where approvertopcicstart = 0 and approvertopicend = 9, the Approver will expect a sequence of topics to exist starting with 'approver-0', 'approver-1' and so on to 'approver-9'. Where 'topiccount' = 1, one of these topics will be chosen to be listened to. If 'topiccount' were to = 3, then a sequence of three (starting at a random offset) topics would be chosen. topiccount — Default 1. Each approver can listen to one or more approver topics. This parameter defines how many topics this worker will listen to. A random range of <topiccount> topics is selected. approvertopcicstart — Default 0. This parameter sets the 'start' of the topic range that this instance will try to select from. approvertopicend — Default 9. This parameter sets the 'end' of the topic range that this instance will try to select from. threadcount — If set, will specify the number of worker processes readtopic — Topics read by this instance are auto-generated by reference to 'approvertopcicstart' and 'approvertopcicend'. This may be overridden by directly specifying one or more topics (csv) in this parameter value.

Consumer	Specific Consumer config (such as Kafka Bootstrap) can be set with the **SCHEDULER_** prefix.
Producer	Specific Producer config (such as Kafka Bootstrap) can be set with the **ASSEMBLER_** prefix.

Assembler (7)

Purpose	The Assembler takes partition updates from the Scheduler and approvals from its 'assembler-<n>' topics, applies the approval signatures to ledger updates, and, once all signatures have been received, sends hash details to the sequencer and sends partition updates with signatures to the 'stateupdate-<n>' topics.
Environment Parameters	For testing purposes, there is a defined sequence of topics that the Scheduler and Approvers will post messages to, named in the style 'assembler-<n>' where n is a number in the defined range. For example, where assembler topcicstart = 0 and assembler topicend = 9, the Assembler will expect a sequence of topics to exist starting with 'assembler -0', 'assembler -1' and so on to 'assembler -9'. Where 'topiccount' = 1, one of these topics will be chosen to be listened to. If 'topiccount' were to = 3, then a sequence of three (starting at a random offset) topics would be chosen. updategroupcount — Default 10. State Updates are grouped for processing and publishing purposes. 'updategroupcount' specifies the number of stateupdate topics that will be written to ('stateupdate-0' -> 'stateupdate-<n>') and also the number of logical groupings for state update messages.
	topiccount — Default 1. Each worker can listen to one or more assembler topics. This parameter defines how many topics this worker will listen to. A random range of <topiccount> topics is selected. assemblertopicstart — Default 0. This parameter sets the 'start' of the topic range that this instance will try to select from. assemblertopicend — Default 9. This parameter sets the 'end' of the topic range that this instance will try to select from. threadcount — If set, will specify the number of worker processes readtopic — Topics read by this instance are auto-generated by reference to 'approvertopcicstart' and 'approvertopcicend'. This may be overridden by directly specifying one or more topics (csv) in this parameter value.

	hashtopic statetopic	Default 'sequencer'. Topic to publish hash details to. Default 'stateupdate'. Prefix to use for state-update publish topics.
Consumer	Specific Consumer config can be set with the **ASSEMBLER_** prefix for the consumer that reads from the 'assembler_<n>' topics.	
Producer	Specific Producer config can be set with the **HASH_** prefix for the producer that publishes hashes for sequencing. Specific Producer config can be set with the **STATE_** prefix for the producer that publishes state-update packages.	

Sequencer (10)

Purpose	The Sequencer takes update hashes from the Assembler, packages them into signed, linked, blocks and then published these blocks to the 'blocks' topic.	
Environment Paramenters	threadcount readtopic writetopic	If set, will specify the number of worker processes Default 'sequencer'. Defines what topic name is used to read update hashes from. Default 'blocks'. Topic to publish block details to.
Consumer	Specific Consumer config can be set with the **SEQUENCER_** prefix for the consumer that reads from the 'sequencer' topic.	
Producer	Specific Producer config can be set with the **BLOCKS_** prefix for the producer that publishes blocks.	

State Updater (12)

Purpose	The StateUpdater is designed to isolate state movements relating to particular criteria by extracting matching transfers from each Block and then persisting the relevant state movements to a customer update topic from which they will be able to update their chosen persistence store. The test StateUpdater is designed to extract state movements relating to a given Asset though future developments will be designed to match other criteria.

Environment Parameters	podname	When given in the format XXX-<n>, this will automatically se configuration by reference to the value given in <n>. Designed for ease of scaling in a Kubernates environment.
	txreaderthreadcount	Limit threads used to read transactions.
	blockreaderthreadcount	Limit threads used to read Blocks.
	workerthreadcount	Limit threads used to process Blocks.
	blocktopic	Set topic to read Blocks from
	tradetopics	Set topic(s) to read Transactions from, if not set with 'podname' config.
	groupids	Set GroupIDs to process, if not set with 'podname' config. MUST match with 'tradetopics' configuration if given.
	assets	The test StateUpdater looks for transactions relating to one or more assets. This configuration can be used to determine which Assets(s) are checked for.
Consumer	The State Updater creates two sets of Consumers relating to the consumption of Blocks and Transaction Packages. Specific Block Consumer config (such as Kafka Bootstrap) can be set with the **BLOCKS_** prefix and specific Package Consumer config can be set with the **STATE_** prefix. Either consumer will use generic Consumer configuration parameters, though specific settings will take precedence	
Producer	There is a single Kafka Producer created for publishing state changes to a RLN client. Specific producer config (such as Kafka Bootstrap) can be set with the CUSTOMER_ prefix or with the generic Producer configuration settings	

09

Appendix B: Glossary of Terms

Approval Agent	The Approval Agent is a software component that is run by or on behalf of a Partition Owner. This component completes the checks necessary for a partition to approve any change to their partition. Typically it would do AML, sanctions and fraud checking. It may also include a check on the balance or credit of the account being changed.
Global State	Global State is a synchronised data set that is the same for every participant of a system. In this system Global State evolves in discrete steps by applying blocks of state updates in an the same order for every participant. This does not mean that every participant sees every update – just that the updates they see are ordered identically for everyone. This means that any state updates that belong to a particular transaction are grouped into the same state update block so the whole transaction can be seen by all participants to have all happened or to have all not happened.
Instrument	An instrument is something that is the object of a promise made by a participant on the RLN. Promises are made by partition owners in respect of some 'thing' such as GBP or BP Shares. Those promises are represented by Tokens on the RLN. They are also liabilities on the balance sheet of the partition owner. Many participants may promise the same instrument. For example, any number of banks may promise GBP as tokens. The RLN is a network that orchestrates tokens to allow a movement of value between different partition owners' customers
Network Map	The Network Map is a collection of records that allows the RLN to determine the relationship between the sending and receiving partitions so that settlement can be effected through the intermediating partitions
Partition	A Partition is a logical component of the RLN that is able to issue promises in the form of tokens. Those promises are liabilities on the balance sheet of the Partition Owner.
Partition Changes	Partition Changes comprise the redemption, issuing and transferring of tokens within a Partition. As Tokens are representations of liabilities on the Partition Owner's balance sheet, these are effectively balance sheet changes.
Partition Owner	A Partition Owner is the regulated entity that controls a Partition on the RLN

Persistence Engine	A Persistence Engine is a piece of software whose function it is, is to write date to a non-volatile store, such as a database. The RLN will produce a globally ordered set of state changes. Those changes will impact balances of Tokens. The Persistence Engine will consume those state changes and update some record of balances.
Persistence Store	A Persistence Store is the non-volatile store which is updated by the Persistence Engine.
Proposal	A Proposal is the collection of Partition Changes that are required to settle a transaction. This is computed from the transaction and the Network Map. A Proposal can only proceed if all the impacted Partition Owners agree to it.
Regulated Liability	A Regulated Liability is a promise from a regulated institution. Bank accounts, for example, are promises from banks. Because such promises are so important to our daily lives, regulators monitor that banks have sufficient assets to meet the promises they make to their customers.
Regulated Liabilities Network	The RLN is a network which allows regulated institutions to issue tokens which represent their promises. The RLN orchestrates value transfers between different partitions on the RLN by orchestrating the issuing, redemption and transfer of those tokens.
Settlement Partition	A settlement partition is an outside agent selected by a Partition Owner for completion of settlement for a particular asset. It is the first partition in a chain of partitions that are required to settle a sender's partition with a recipient's partition. If a partition is the primary partition for an instrument it will serve as its own settlement partition. If the sender and recipient of a transaction are both on the same partition, a settlement partition is not needed.
Token Holder	A Token Holder is a beneficiary of a promise made by a regulated institution on the RLN.