AERGO
Technical White Paper

Last Updated: 17 July 2018, AERGO

ABSTRACT

AERGO is a proposed new blockchain protocol, which aims to power both public and private blockchain deployments. Based on Blocko Inc's (**Blocko**) experience in providing large-scale, production level private blockchain to recognized enterprise customers, AERGO intends to be purpose-built to enable enterprise architectures based on blockchain by incorporating both new, innovative and established technical approaches to build scalable distributed database systems.

LEGAL DISCLAIMER

This paper relates to the AERGO project and should be read in conjunction with the whitepaper available at https://AERGO.io. This and other documents may be amended or replaced at any time, without notification of any changes or access to any additional information.

This paper describes a future project

This paper contains forward-looking statements that are based on the beliefs of AERGO Limited, a private Hong Kong company limited by shares (CR No. 2713137) (**AERGO Limited**), as well as certain assumptions made by and information available to the AERGO Limited.

AERGO as envisaged in this technical whitepaper is under development and is being constantly updated, including but not limited to key governance and technical features. The native AERGO token (**AERGO Token**) involves and relates to the development and use of experimental platforms (software) and technologies that may not come to fruition or achieve the objectives specified in this whitepaper. If and when AERGO is completed, it may differ significantly from the network set out in this whitepaper. No representation or warranty is given as to the achievement or reasonableness of any plans, future projections or prospects and nothing in this document is or should be relied upon as a promise or representation as to the future.

Eligible purchasers

The information in this whitepaper is provided privately to certain prospective purchasers and is not intended to be received or read by anyone else. Eligibility is not guaranteed and is likely to be subject to restrictions.

No offer of regulated products

The AERGO platform, AERGO Token or any token that operates on it is not intended to represent a security or any other regulated product in any jurisdiction. This document does not constitute an offer or solicitation of securities or any other regulated product, nor a promotion, invitation or solicitation for investment purposes. The terms of the purchase are not intended to be a financial service offering document or a prospectus of any sort.

AERGO Token does not represent equity, shares, units, royalties or rights to capital, profit, returns or income in the platform or software in the AERGO Limited or any company or intellectual property associated with the platform or any other public or private enterprise, corporation, foundation or

other entity in any jurisdiction

This technical whitepaper is not advice

This technical whitepaper does not constitute advice to purchase AERGO Token. It must not be relied upon in connection with any contract or purchasing decision.

Risk warning

The purchase of AERGO Token and participation in AERGO Token sale carries with it significant risks. Prior to purchasing AERGO Token, you should carefully assess and take into account the risks, including those listed in any other documentation.

Views expressed in this technical whitepaper

The views and opinions expressed in this technical whitepaper are those of AERGO Limited and do not reflect the official policy or position of any government, quasi-government, authority or public body (including but not limited to any regulatory body of any jurisdiction) in any jurisdiction. Information contained in this technical whitepaper is based on sources considered reliable but there is no assurance as to their accuracy or completeness.

English is the authorized language of this whitepaper

This technical whitepaper and related materials are issued in English only. Any translation is for reference purposes only and is not certified by AERGO Limited or any other person. No assurance can be made as to the accuracy and completeness of any translations. If there is any inconsistency between a translation and the English version of this technical whitepaper, the English version prevails.

No third party affiliation or endorsements

References in this technical whitepaper to specific companies and platforms are for illustrative purposes only. The use of any company and/or platform names and trademarks does not imply any affiliation with, or endorsement by, any of those parties.

You must obtain all necessary professional advice

You must consult a lawyer, accountant, tax professional and/or any other professional advisors as necessary prior to determining whether to purchase AERGO Token or otherwise participate in the AERGO project.

This technical whitepaper has not been reviewed by any regulatory authority in any jurisdiction. References in this paper to specific companies, networks and/or potential use cases are for illustrative purposes only. Other than explicitly mentioned partners or providers such as Blocko, the use of any other company and/or platform names and trademarks does not imply any affiliation with, or endorsement by, any of those parties.

BACKGROUND

Blocko has supplied more than 20 enterprise customers with its own private blockchain implementation "Coinstack."[1] Coinstack is based on a modified Bitcoin architecture and Ethereum Virtual Machine executing smart contracts, bearing a close resemblance to QTUM[2] and RSK.[3] While Coinstack performed reasonably well for even larger-scale use-cases such as powering the authentication process for the entire customer-base of a credit card provider with millions of daily users,[4] it also provided an insight into the Bitcoin protocol's upper limit of performance and the Ethereum Virtual Machine's incompatibility with the enterprise architecture and the developers behind them.

In order to better leverage the tool chain and the application architecture of Coinstack supporting actual use-cases, Blocko started working on AERGOSQL and AERGO. AERGOSQL is an innovative, new smart contract engine capable of utilizing relational data model and developing smart contracts using tools and languages familiar to enterprise developers. For a detailed description of AERGOSQL, see the AERGOSQL technical whitepaper available at https://AERGO.io/paper/.

This paper describes the challenges faced by enterprise blockchain deployments and the new requirements and architecture capable of addressing these challenges.

ENTERPRISE BLOCKCHAIN REQUIREMENTS

We believe that enterprise blockchains operate under different assumptions and environments from public, generic blockchains. With the deployment of Coinstack, Blocko gained first-hand exposure to the reality of enterprise blockchain adoptions. We describe a number of these general assumptions below:

- Unlike public blockchain users, who usually operate blockchain nodes on commodity hardware, businesses tend to run blockchain on server grade hardware with vast computing power and storage.
- Businesses want to run blockchain not only on cloud, but on private cloud and bare metal machines as well. The functionalities provided by private cloud and bare metal environments differ significantly from public cloud services.
- While public blockchain users run blockchain nodes at small number, businesses want to run a large number of blockchain nodes in order to take advantage of horizontal scalability and availability.
- Businesses need more control and functionalities related to the administration of blockchain than public blockchain users.
- While most of applications running on public blockchain are self-contained or dependent only

[1] http://blocko.io
[2] https://qtum.org/en/
[3] https://www.rsk.co
[4] http://www.blocko.io/news/view/39

on assets on the blockchain itself, businesses want to connect applications running on blockchain with many external and internal systems such as e-mail, SMS, databases, LDAP, and public data.

We explore below a number of other key attributes we believe that are integral to enterprise focused blockchains.

SCALABILITY

Since enterprise blockchain users typically have better access to hardware in terms of both quantity and quality, enterprise blockchain implementations need to scale both horizontally and vertically.

INTEROPERABILITY

Enterprise environments tend to depend on diverse range of technologies accumulated through years of operation and enterprise blockchain implementations need to work with both modern, standard interfaces such as OAuth and old, propriety interfaces such as Active Directory.

DEVELOPMENT ENVIRONMENT

Since the majority of enterprise development tend to be project focused, there is little room for experimenting and learning new languages and tools for the developers; instead of forcing developers to learn new languages to create smart contracts, enterprise implementations need to allow developers to leverage their existing knowledge and experience with familiar toolchain.

At the same time, certain resources web developers take for granted, such as unlimited internet access, are not available for enterprise developers. As a result, enterprise blockchain implementations need to supply a more comprehensive development environment with IDEs, SDKs, and reference architectures than public blockchain implementations.

DATA PRIVACY

Businesses face pressures to ensure stringent data security in terms of confidential information and also customer / employee personal data. Often the desire for data security is a more important consideration than the immutability and integrity of data provided by blockchain. While one way to achieve data security on public blockchains is to implement an encryption and decryption layer at the application level. enterprise blockchain implementations need to provide a more robust, holistic approach to securing data.

PROVISIONING AND ADMINISTRATION

While web developers are happy to use Vagrant or Docker on their laptops, enterprise IT is more comfortable with bigger guns like Tivoli Provisioning Manager, OpenStack, or Kubernetes. Enterprise blockchain implementations need to support integration with existing technology for provisioning and managing in enterprise IT and provide much richer suite of functionalities for

administration. Export and import, backing up the data, monitoring, logging, and data migration are typical features overlooked by public blockchain implementations, but important in the enterprise environment.

STRUCTURED AND UNSTRUCTURED DATA STORAGE

Smart contracts provide the foundation of functionality on both public blockchain and enterprise blockchain. Unlike dApps built on public blockchains with their access to cloud-based storage and CDN providers, dApps on enterprise blockchains need to be more self-reliant and enterprise blockchain implementations need to accommodate them with rich functionality for both structured and unstructured data storage.

CORE ARCHITECTURE

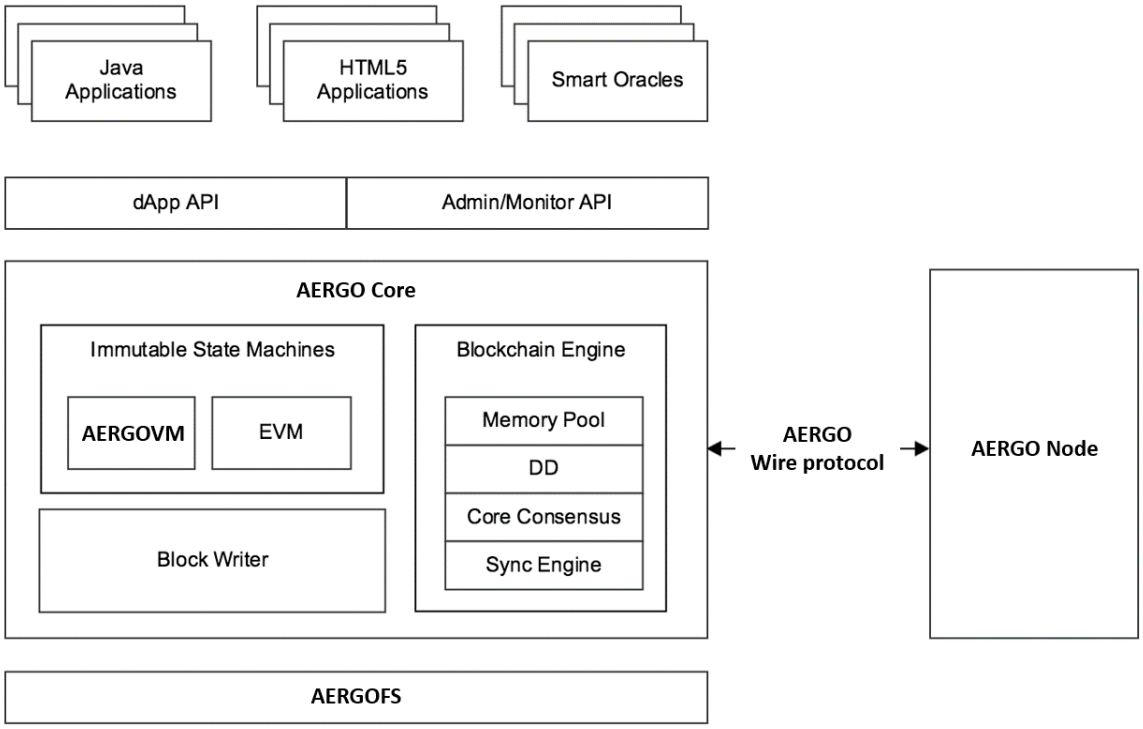


Figure 1. AERGO Architecture

AERGO is designed to be a holistic, multi-purpose platform, that bridges the gap between public blockchains and private blockchains. In order to be effective under both environments, AERGO is intended to be compact, yet flexible in design.

In order to service multi-tenant workloads with potentially millions of concurrent users accessing the same set of nodes, AERGO intends to borrow many concepts from both traditional database designs and distributed computing.

DISTRIBUTED DIRECTORY

Distributed Directory (**DD**) is a core functionality that is intended to be used as a building block for the whole AERGO implementation.

Each DD in a repository is proposed to manage an independent, isolated namespace. Each namespace contains information about different branches and tags residing in the repository, as well as the validity of various identifiers on the blockchain.

Each DD is intended to be a blockchain on its own, with its own genesis block and the best block. Unlike conventional blocks, DD blocks are limited in size with a relatively long creation interval between them; since DDs are used for managing metadata, they need to be compact.

DD is comparable to data dictionaries in databases, zookeeper for Hadoop, or etcd for CoreOS in its role and functionality.

a. Tree of Life (ToL)

The ToL namespace of a DD is proposed to contain information about all the branches in the repository, as well as their genesis blocks or root blocks. The information about tags are managed inside the ToL namespace as well. As a result, the ToL namespace contains information about the best block of each branch as well; since the HEAD tag continuously keeps track of the best block of each branch.

b. Distributed Directory Service (DDS)

The DDS namespace is proposed to contain entries for different entities on blockchain; their public keys and validity, as well as associated roles and permissions. The DDS namespace is intended to serve as the basis for access control for AERGO repositories.

Each entity can represent either a client-actor or a server certificate. For entities with server certificates, DDS can serve as both Certificate Revocation List and a DNS with routing information.

AERGOFS, the proposed distributed file system component of AERGO, is intended to be dependent on DDS, since DDS keeps track of data volumes consisting each AERGOFS instance. In turn, AERGOFS can be used for storing blocks and indexes for different branches in the repository.

The DDS namespace forms the basis of identity for nodes to participate in the core consensus process as well.

CONSENSUS ALGORITHM

a. Core Consensus

The core consensus algorithm is intended to be used for building the DDS. The core consensus algorithm and the DDS are mutually dependent, since the core consensus algorithm needs to access the DDS within the DD to enable mining new blocks.

The proposed core consensus algorithm of AERGO is Delegated Proof of Stake (DPOS)[5]. DPOS is the preferred consensus model because, in summary:

- We believe it provides the scalability and the simplicity of operation required by a core consensus; and
- DPOS operates under the assumption that block reorganizations can happen, which means it is an optimal algorithm for powering the underlying infrastructure of AERGO.

b. User-Defined Consensus

By default, each repository uses the core consensus. Since AERGO intends to provide a pluggable architecture for consensus algorithm as well, different consensus algorithm modules can be used in place of the core consensus. Notably, RAFT (for development) and PBFT (for strict-ordering) are useful for developing and running different services.

Using the same toolchain for building smart contracts, a user-defined consensus algorithm can be used for each repository as well. The user-defined logic can govern how following events are occurred and managed in the blockchain.

- Block creation and its permission
- Block transmission and priorities

Since block branching and merging can be perceived as block reorganization events as well, the same policy for block reorganization is used for distributed version control as well. From version control perspective, the block reorganization policy is called "Consistent Merging."

SMART CONTRACTS

AERGO supports a multi-paradigm, plugin-based smart contract infrastructure.

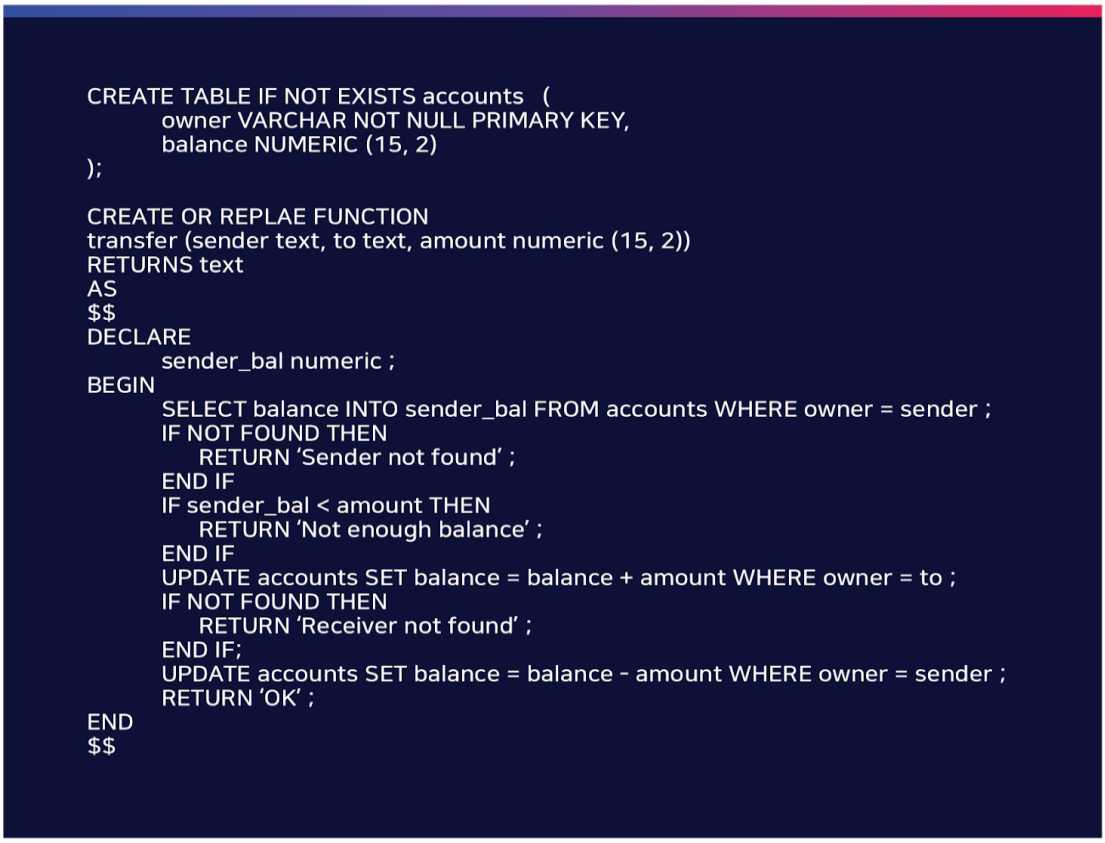
Each contract can be executed or queried by a client-actor or another smart contract instance. Since AERGO provides a permissive interface with maximum interoperability between smart contract implementations, contracts written for Ethereum Virtual Machine, Fabric Chaincode, or AERGOSQL can be used with each other.

[5] https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper

a. AERGOSQL

The canonical way to write a smart contract for AERGO is provided by AERGOSQL. AERGOSQL provides a relational data model for storing and accessing data and SQL-like scripting language for writing smart contracts.

Using AERGOSQL, smart contracts can be written using the familiar SQL syntax.



```
CREATE TABLE IF NOT EXISTS accounts   (
        owner VARCHAR NOT NULL PRIMARY KEY,
        balance NUMERIC (15, 2)
);

CREATE OR REPLAE FUNCTION
transfer (sender text, to text, amount numeric (15, 2))
RETURNS text
AS
$$
DECLARE
        sender_bal numeric ;
BEGIN
        SELECT balance INTO sender_bal FROM accounts WHERE owner = sender ;
        IF NOT FOUND THEN
            RETURN 'Sender not found' ;
        END IF
        IF sender_bal < amount THEN
            RETURN 'Not enough balance' ;
        END IF
        UPDATE accounts SET balance = balance + amount WHERE owner = to ;
        IF NOT FOUND THEN
            RETURN 'Receiver not found' ;
        END IF;
        UPDATE accounts SET balance = balance - amount WHERE owner = sender ;
        RETURN 'OK' ;
END
$$
```

Figure 2. AERGOSQL Coding model extract

For maximum performance, AERGOSQL leverages technologies such as LLVM to utilize JIT compilation[6] and high-performance b-tree implementations such as WiredTiger[7]for data storage.

[6] https://llvm.org
[7] http://www.wiredtiger.com

b. Interoperability

With its pluggable architecture, AERGO is designed to support different smart contract implementations. AERGO inherits the Ethereum Virtual Machine compatibility from Blocko Coinstack out of box. Fabric Chaincode is supported through lightweight virtualization such as Docker.

The initial release of AERGO is dependent on go-Ethereum's EVM implementation. The use of evmjit for higher performance is planned in the future.

SMART ORACLES

AERGO supports integrating smart contracts inside the walled garden of blockchain, as well as smart contracts that have regard to external events and factors through implementing smart oracles. Smart oracles seek to provide following functionalities:

- Allow smart contracts to consume data from legacy systems such as Active Directory
- Allow smart contracts to trigger events in external services such as e-mail or SMS

From the perspective of a smart contract, smart oracles are external factors that are coupled to a specific smart contract; smart oracles react to changes to the coupled smart contract and inject data as a response. In some cases, smart oracles can trigger smart contracts autonomously.

From the perspective of a dApp, smart oracles implement micro-services that expose external functionalities required by the dApp. Since smart oracles and dApps can communicate off-chain, the micro-services provided by smart oracles can be used to implement an out-of-band communication required by the smart contract; a common use-case includes exchanging an authentication token between a smart oracle and dApp.

Isomorphic Contracts

AERGO development toolkit intends to support the isomorphic execution of a smart contract through automatic code generation. The isomorphic code generated from a smart contract can be accessed by both dApp and smart oracles, enabling a transparent access to the smart contract and the underlying data structure. The isomorphic execution of a smart contract is critical to the productivity of developing a smart contract and applications or services based on it.

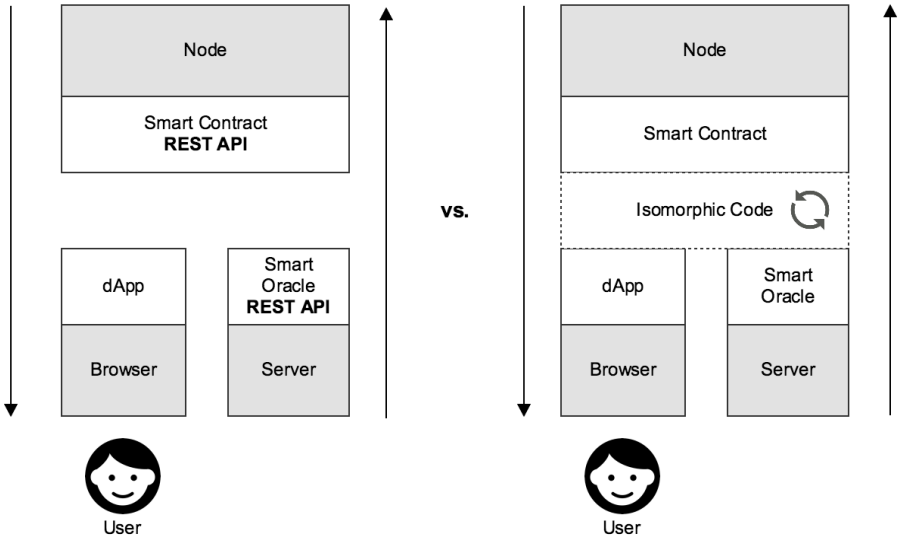


Figure 3. Conventional dApp vs. Isomorphic dApp Architecture

Not all smart contract languages support isomorphic contracts; the support for isomorphic contracts is limited to contracts written for AERGOSQL.

DISTRIBUTED FILE SYSTEM

AERGOFS is a core component of the AERGO platform, providing distributed file system functionalities.

AERGOFS is dependent on the DD for managing metadata related to files; metadata about each file including physical location, hash value, and various statistics is stored within the DD.

While smart contracts provide structured data storage with data schema and indexes for faster query, AERGOFS intends to provide the unstructured data storage capability of AERGO.

AERGOFS provides a simple HTTP interface, enabling access from both smart oracles running on server environment and dApps running on web browsers.

DISTRIBUTED VERSION CONTROL

Unlike traditional blockchain systems, AERGO views chain forks and block reorganizations as core features of blockchain, rather than annoying side effects. By adopting git-like data models and command structure, AERGO seeks to enable collaborating on data as easy as it is to collaborate on source code.

REPOSITORIES

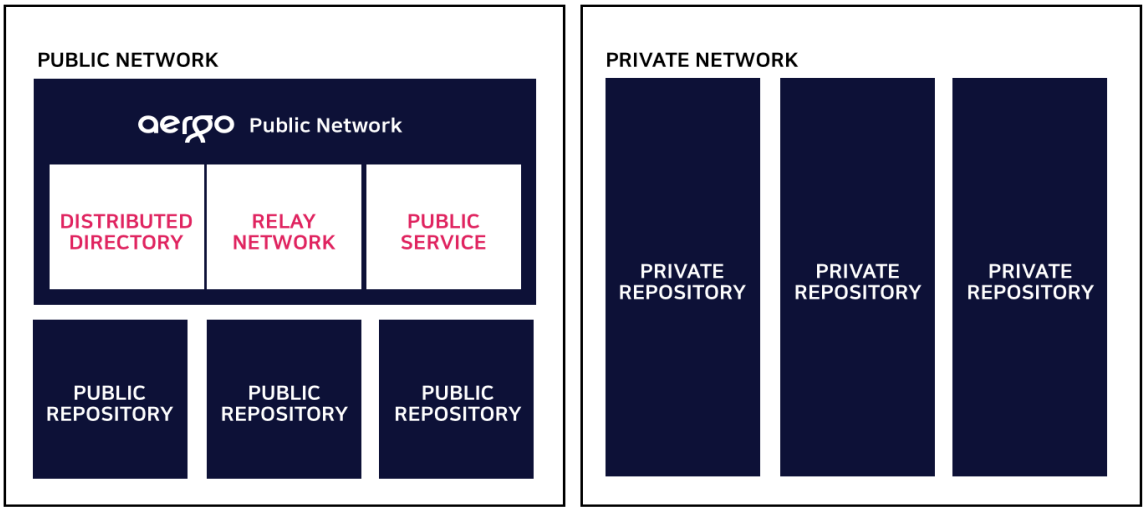


Figure 4. Public and Private Repositories

AERGO supports the creation of public and private repositories. Each repository can be either named or unnamed. A named repository has an associated public entity on AERGO Public Network's Distributed Directory. An unnamed repository has no such association.

Much like a public Git repository, a public AERGO repository is intended to be transparent to read and write, or selectively allow different permissions to anonymous users. A common configuration is to create a public AERGO repository with read-only anonymous access.

A private repository is intended to be an AERGO repository with full access control enabled, both for reading and writing the repository. A public or private repository is effectively a private blockchain in a sense that it operates independently from AERGO Public Network. As a result, AERGO Token does not have any utility within public or private repositories.

BRANCHES

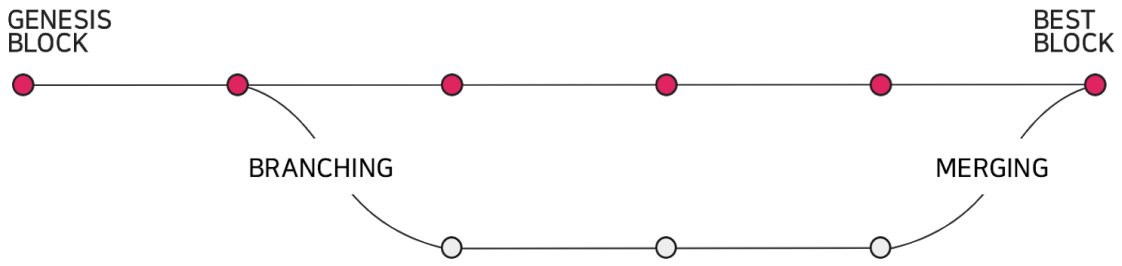


Figure 5. Branching and Merging Blocks

Within each repository, different branches pointing to a different snapshot in the blockchain status can be created. In fact, the concept of “best chain” in AERGO is analogous to the master branch.

SYNTAX AND SEMANTICS

AERGO seeks to provide friendly syntax and semantics for users accustomed to version control systems such as Git. Such functionalities can be accessed through AERGO CLI client, as well as RPC APIs.

a. Basic Commands

Set out below are illustrations of the basic usage of AERGO for distributed version control.

```
aergo branch <new branch> [--block=<block hash>]
```

The above command creates a new branch. Without an implicit block hash as a parameter, the current branch's best block is used as the root block for the new branch. The new branch functions as an independent chain, with the ability to acquire new blocks. Without user-created branches, the master branch exists by default.

```
aergo tag <block hash> [--block=<block hash>]
```

The above command creates a new named tag. Without an implicit block hash as parameter, current branch's best block is used as the root block for the new tag. Unlike a branch, a tag is unable to acquire new blocks.

```
aergo checkout <branch | tag>
```

The above command checkouts an existing branch or tag for examination or manipulation.

```
aergo pull <repository:branch>
```

This command merges the changes in the remote branch to the local repository branch. As a result, the remote transactions are applied to the local repository as well. In the process, the named tags are synchronized as well.

```
aergo push <repository:branch>
```

The above seeks to merge the changes in the local branch to the remote repository branch. As a result, the local transactions are applied to the remote repository as well. In the process, the named tags are synchronized as well.

b. Branching and Merging

One of the most complicated concepts in distributed version control systems is the process of merging branches. For blockchains with real-time data, merging is even more difficult to achieve. Due to its non-destructive process, branching is a simple and straightforward process.

However, merging requires two different approaches.

Automatic Merging
By default, Automatic Merging is the expected process for merging two branches. Automatic Merging is similar to the block-reorganization process in blockchains. In this case, the merging source's blocks are dissolved into transactions and absorbed in the merging target's merging-pool. Ultimately, the merging pool results in a new block attached to the merging target's best block. In the process, transactions inconsistent to the merging target branch are automatically excluded from the new block.

Consistent Merging
Consistent Merging happens only when a branch is created with a specified consistent merging logic. Consistent merging is similar to the merge functionality provided by version control systems like Git. Unlike Automatic Merging which discards inconsistent transactions by default, Consistent Merging relies on the predefined conflict resolution logic to manage inconsistent transactions. The conflict resolution logic is implemented as a system-level smart contract.

SCALABILITY

AERGO employs three different approaches for achieving scalability.

- Domain partitioning
- Scale up
- Scale out

DOMAIN-BASED PARTITIONING

Domain-based partitioning is the most basic scalability strategy used by AERGO. Domain-based partitioning is achieved through the distributed version control (DVC) functionality of AERGO.

Unlike conventional blockchain implementations, AERGO is able to fork and merge its data through branches freely. As a result, the distributed ledger can be partitioned both logically and physically through different repositories.

Such approach is already used successfully by distributed version controls such as Git and Mercurial. For instance, a gigantic service like GitHub is able to host tens of millions of repositories.

However, the effectiveness of domain-based partitioning is dependent on the structure and usage of data. When a single repository needs to handle unbounded expansion of data, partitioning data through branching is very difficult. As a result, two additional scalability approaches are provided by AERGO for handling huge amount of data for a single repository.

SCALE OUT

AERGO's scale out strategy depends on the functionality provided by AERGOFS. AERGOFS fulfils two roles for achieving scalability:

> (1) AERGOFS can serve as a storage layer for each node's blocks and indices. The manner AERGO nodes utilize AERGOFS is very similar to how HDFS is used by HBase. With AERGOFS, each node is able to store unlimited number of blocks and indices and function as a gigantic uber-node.
>
> (2) AERGOFS is able to function as an object storage similar to S3 as well. In this configuration, AERGOFS provides immutable and durable access to binary data. In this case, AERGO's smart contracts need to store locators to access files stored on AERGOFS.

SCALE UP

The most direct and simple approach that AERGOFS seeks to utilize for scalability is through optimizing a single node.

While horizontally scaling out works well for large amount of data, it fails to meet realistic benchmarks. With the advent of cheap memory, fast storage such as SSD, and limited network throughput, optimizing a single node is very effective for everyday systems. Blocko learned this lesson dearly while providing a real life blockchain implementations in the enterprise world, and AERGO, with Blocko's assistance, seeks to borrow many ideas and techniques from Blocko's Coinstack in this regard.

In order to make each node as efficient as possible, AERGO nodes are intended to be equipped with an efficient networking stack and an optimized storage engine for enhanced I/O.

- AERGO networking stack provides an out of order, highly parallel networking fabric that is able to serve a high number of nodes with complex topology on both bare metal environment and cloud environment.
- AERGOSQL forms the basis of the high-performance storage engine required by AERGO.
- AERGO nodes use multi-thread architecture to take advantage of a multi-core environment.

CONCURRENCY CONTROL

AERGO seeks to provide two mechanisms for transaction serialization.

BLOCK LEVEL SERIALIZATION

Since each branch of blockchain consists of a series of blocks, the transactions can be serialized through stacking after one another.

AERGO aims to provide Multi Version Concurrency Control (MVCC) based on block heights. As a result, with a branch and block height specified, it is possible to provide [consistent reads]

across different nodes in the repository.

AERGO's MVCC functionality aims to provide both a snapshot isolation for consistent reads and a form of optimistic locking through row or document versioning. However, MVCC works only for block-level serialization.

POOL LEVEL SERIALIZATION

Clients accessing AERGO nodes can take advantage of the deterministic, scheduled creation of blocks by delegates, a characteristic provided by DPOS and core consensus, to execute transactions synchronously, with a strong guarantee on transaction finality.

Since each delegated node can apply a uniform serialization ordering to process new transactions into the memory pool and to create new blocks, clients do not have to wait for the block interval to retrieve the result of transactions. As a result, the latency of executing a transaction decreases from seconds to milliseconds.

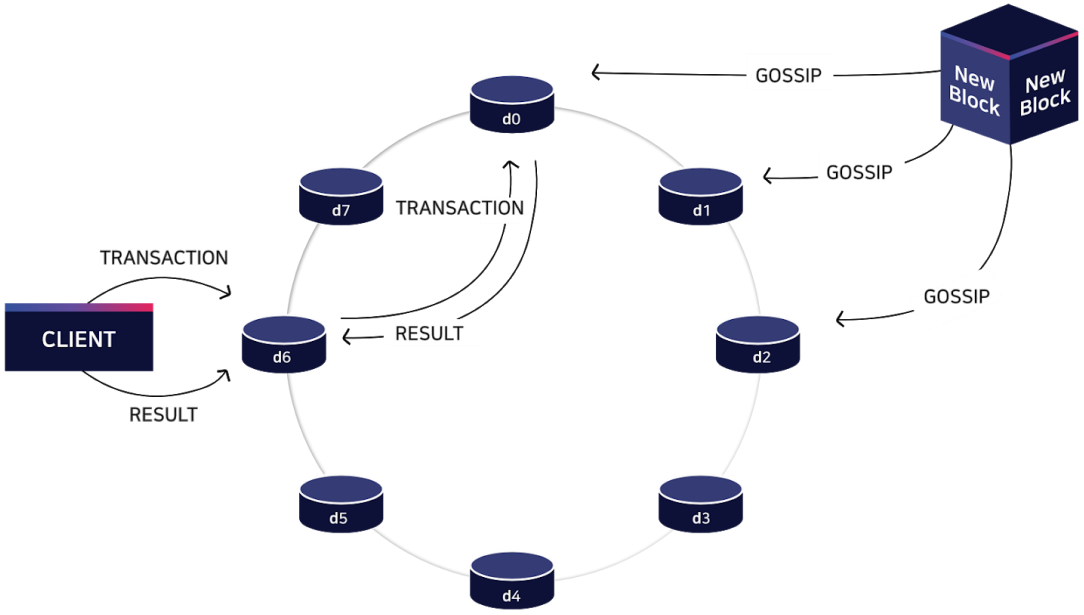


Figure 6. Pool Level Serialization

However, with block reorganizations and chain partitioning in play, as well as the presence of ill-intentioned clients, pool level serialization provides only a probabilistic level of consistency. On the other hand, with optimistic workloads, pool level serialization works well for solving real life problems.

PRIVACY

DATA ISOLATION

AERGO intends only to allow users with adequate permission to access ledger data by providing git-like private repositories.

By creating a new branch from a remote parent branch, users are able to keep newly created blocks in a private branch, such that they are isolated from the public. Only with those permission to the specific repository housing the branch are able to access the blocks.

DATA SHARING

A specific branch can be synced with remote repositories to exchange data. In this case, the private branches of the repository can either cherry-pick relevant commits from the public repository or merge the whole change set automatically.

PARALLELISM

The performance of a specific blockchain depends on the efficiency of creating and sharing new blocks, and the time it takes for each node to validate the new blocks.

The block creation process involves a consideration of the whole distributed consensus protocol of blockchain. It is submitted that the block validation process used as part various distributed consensus protocols is sometimes poorly designed and implemented.

While underperforming nodes are acceptable for consumer-grade blockchain implementations such as bitcoin or Ethereum, enterprise-grade blockchains like AERGO require much robust performance on a near real-time basis. As a result, each node needs to be implemented with as much efficiency as the consensus protocol itself.

AERGO intends to introduce the concept of parallelism to various stages of processing blocks to maximize the performance.

The parallelism involves the careful analysis of dependencies between transactions included in each block and an efficient architecture inspired by SEDA[8].

DEPENDENCY ANALYSIS

In order to guarantee consistency between nodes, blockchain implementations usually employ the policy of serializing the execution of all the transactions and the blocks available.

As a result, the rate of blocks a blockchain node can process depends on the time it takes to process each transaction, regardless of the number of processing units or memory available.

[8] https://en.wikipedia.org/wiki/Staged_event-driven_architecture

In order to enable the parallel validation of transactions and blocks, AERGO intends to perform a dependency analysis between transactions and blocks and create a data structure known as Deterministic Transaction Tree.

Deterministic Transaction Tree

A Deterministic Transaction Tree (**DTT**) can be viewed as a formal representation of the execution order of transactions to result in deterministic results to the state machines affected by the transactions.

As a result, for a set of transactions, there can be more than one viable and correct DTT.

Each branch of a DTT can be processed and applied to the underlying state machines related to the transactions in parallel with deterministic resulting states. A typical DTT will have a number of branches with varying lengths.

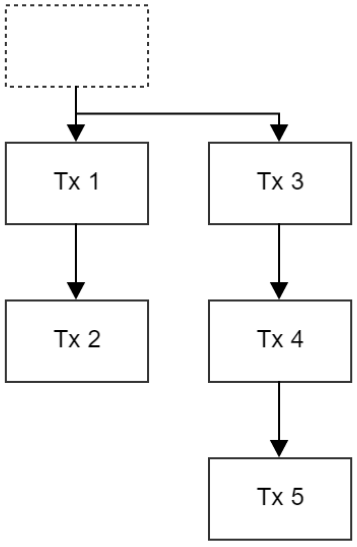


Figure 7. Deterministic Transaction Tres

Depending on the size of blocks, each DTT can have branches from with couple of transactions in length to thousands of transactions in length. Similarly, a DTT can have varying number of branches as well.

The validity of a DTT can be only verified by actually executing a DTT against a set of state machines. A version of DTT can be optimized into another version by transforming the tree as well.

In order to create a DTT for a set of transactions in a realistic time frame, AERGO employs a rule-based approach to analyze the transactions. More sophisticated approaches including machine learning are planned to be tested in the future releases of AERGO.