Not ACID, not BASE, but SALT
A Transaction Processing Perspective on Blockchains

Stefan Tai, Jacob Eberhardt and Markus Klems
Information Systems Engineering, Technische Universität Berlin, Germany

Keywords: SALT, Blockchain, Decentralized, ACID, BASE, Transaction Processing.

Abstract: Traditional ACID transactions, typically supported by relational database management systems, emphasize database consistency. BASE provides a model that trades some consistency for availability, and is typically favored by cloud systems and NoSQL data stores. With the increasing popularity of blockchain technology, another alternative to both ACID and BASE is introduced: SALT. In this keynote paper, we present SALT as a model to explain blockchains and their use in application architecture. We take both, a transaction and a transaction processing systems perspective on the SALT model. From a transactions perspective, SALT is about Sequential, Agreed-on, Ledgered, and Tamper-resistant transaction processing. From a systems perspective, SALT is about decentralized transaction processing systems being Symmetric, Admin-free, Ledgered and Time-consensual. We discuss the importance of these dual perspectives, both, when comparing SALT with ACID and BASE, and when engineering blockchain-based applications. We expect the next-generation of decentralized transactional applications to leverage combinations of all three transaction models.

1 INTRODUCTION

There is a common belief that blockchains have the potential to fundamentally disrupt entire industries. Whether we are talking about financial services, the sharing economy, the Internet of Things, or future energy markets – any application where some form of trade and agreement among different parties occurs is deemed a candidate for blockchains. And when using blockchains, so the promise, trustless interactions – in the sense that trust no longer must be managed by some central or intermediating party – and hence lowered transaction costs and other benefits of decentralization are introduced.

Is this considerable interest in blockchain technology justified? What are blockchain-based transactions? How do blockchain-based transactions compare to traditional database and distributed systems transactions? Do blockchains replace other database technology and systems or should they be used as a complement?

In this keynote paper, we take a transaction processing perspective on blockchains. With relational database management systems that implement ACID transactions, and cloud systems and NoSQL stores that favor the BASE model, we have two established models to compare blockchain-based transactions against. Using the admittedly contrived acronym of SALT, we characterize blockchain-based transactions – from a transactions perspective – as *Sequential, Agreed, Ledgered, and Tamper-resistant*, and – from a systems perspective – as *Symmetric, Admin-free, Ledgered, and Time-consensual*.

Following either ACID, BASE, or SALT naturally results in different notions of transactions and different transactional system architectures. Using two real-world application examples that use blockchains we illustrate how different business transactions map to a combination of these three transaction models and corresponding data management components.

2 BLOCKCHAINS

Since its inception in 2008 as the technology behind Bitcoin (Nakamoto, 2008), blockchains are discussed in a diversity of communities in research and practice. These include cryptocurrency and security, databases and distributed systems, IT law, entrepreneurship, and many more. In addition, blockchain developer communities exist, focusing on specific blockchain technologies such as Bitcoin, Ethereum, Hyperledger, and other. Out of the many views on blockchains and corresponding definitions that exist, we like to emphasize

Tai S., Eberhardt J. and Klems M.
Not ACID, not BASE, but SALT - A Transaction Processing Perspective on Blockchains.
DOI: 10.5220/0006408207550764
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 755-764
ISBN: 978-989-758-243-1

the following three:

1. A blockchain is a peer-to-peer protocol for trustless execution and recording of transactions secured by asymmetric cryptography in a consistent and immutable chain of blocks – the blockchain developers and technology view.
2. A blockchain is a shared append-only distributed database with full replication and a cryptographic transaction permissioning model – the IT architect and data management view.
3. A blockchain is a shared decentralized ledger, enabling business disintermediation and trustless interactions, thereby lowering transaction costs – the business executive and applications view.

With each of these three definitions, a different emphasis is set, focusing either on the protocol aspect, the data management aspect, or the decentralization aspect of blockchains. Yet, all three definitions make use of the term *transaction*. In the following, we thus take a closer look at what a transaction is and how such transactions traditionally have been supported by transaction processing systems.

3 TRADITIONAL TRANSACTIONS AND TP SYSTEMS

A (business) transaction, in its most generic sense and as used in commerce, is an instance of buying or selling something. In computer science, a transaction is a logical unit of work performed within a transaction processing system (TP system). A TP system, in turn, refers to an information processing system that divides all processing work into transactions, in a way that each transaction can be guaranteed a set of properties by the system.

3.1 Understanding ACID

In the 1980s, relational database management systems (RDBMS) based on Codd's relational model were first introduced. With the addition of TP technologies to RDBMS a few years later, originally proposed by Jim Gray, the acronym "ACID transaction" was born (Gray and Reuter, 1992). ACID refers to a set of guarantees for each transaction to be processed by the TP system: *Atomicity, Consistency, Isolation*, and *Durability*.

ACID has since been understood as a very convenient model:

- A transaction consisting of multiple operations is executed as a whole or not at all ("all or nothing").
- Each transaction transforms the database from one consistent, valid state to another, adhering to all validation rules and database integrity constraints.
- Concurrent transactions are executed by maintaining isolation, that is, by executing them as if they were sequential.
- Once a transaction has been committed, the results become permanent.

The responsibilities to achieve the ACID properties are spread across different components of a TP system (Bernstein and Newcomer, 2009). A *transaction manager* component typically is required to drive coordination protocols among the *resource managers*, for example, the 2PC completion protocol. Consistency is a responsibility of components that perform validation checks, for example, by using the rules of the database itself. Isolation requires some *concurrency control*, which typically relies upon locking protocols such as the 2PL. And durability typically is a responsibility of the database itself.

Figure 1 illustrates a TP system in support of ACID transactions. An application interacts with the transaction manager to begin (1) and end (3) a transaction as a logical unit-of-work. Each transaction groups a set of operations (2) on one or more databases (resources). For each resource involved in a transaction, a resource manager component exists. The resource managers must be registered with the transaction manager and must understand protocols and provide interfaces in support of transaction processing, e.g., the X/Open XA interface, so that the transaction manager can run completion protocols when committing the transaction (4).

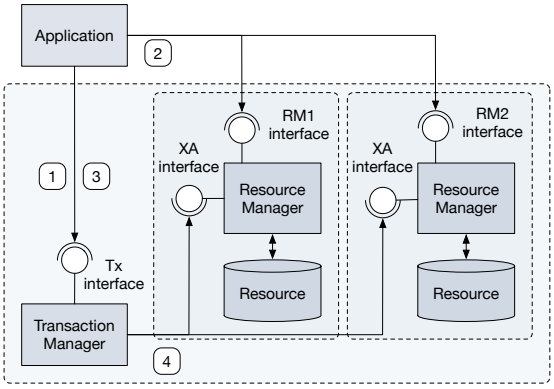

Figure 1: Architecture of a TP system supporting ACID transactions.

Guaranteeing the ACID properties implies that

whenever things do go wrong, the TP system will detect the failure and rollback any intermediate steps taken, if necessary.

3.2 Understanding BASE

With the emergence of unprecedented scalability needs of modern Web applications, an alternative to the rather expensive and consistency-focused ACID model was introduced: BASE – *Basically Available, Soft state, Eventually consistent*. The term was coined around the year 2000, deliberately constructed to describe a model that is diametrically opposed to ACID (Brewer, 2000).

Today, BASE is a model commonly favored by cloud systems and NoSQL stores. BASE captures the following properties (Pritchett, 2008):

- A system is basically available when supporting partial failures without total system failure.
- The state of the system is 'soft' in that it can change over time even if no further updates are made.
- The system will eventually become consistent, if no new updates are made to the system.

Generally, BASE systems follow a peer-to-peer architecture, where each peer is equally responsible for a subset of the overall data, a so-called shard. An incoming client request is forwarded to a responsible node based on a sharding function, e.g., a distributed hashtable. No notion of a transaction manager exists with BASE systems; all nodes can instead process incoming requests, which dramatically increases scalability and throughput. Shards themselves are usually replicated for increased fault tolerance. Hence, not only the data itself, but also the control over it is distributed.

BASE systems typically use optimistic replication techniques and thereby trade some consistency for availability. Instead of requiring availability of all components that are participating in a logical transaction (and thereby coupling all components when processing the transaction), a BASE design encourages interaction only with select nodes and components, thereby tolerating partial failures (of other nodes and components) and accepting that the overall consistency will be in a state of flux. Read and write operations by different clients can be performed by different nodes in the system, thereby ensuring high availability. Updates written to a node typically are propagated asynchronously to other required nodes after returning to the client – the system therefore is in soft state and, if no new updates are made to a given data item, will only eventually converge to a consistent state.

Figure 2 illustrates a system that follows the BASE model. An application interacts either directly with a resource managing component (1.b), or, more typically, through some load balancer (1.a) that distributes incoming requests to peers of nodes with symmetrical responsibilities and capabilities. Requests are served by one or more resource managers directly, and changes are propagated (2) to other required nodes subsequently. The number of resource managers required depends on the sharding model and the system configuration chosen, ranging from a single node (single replica) to a quorum of nodes to, theoretically, all replicas of a given data record.

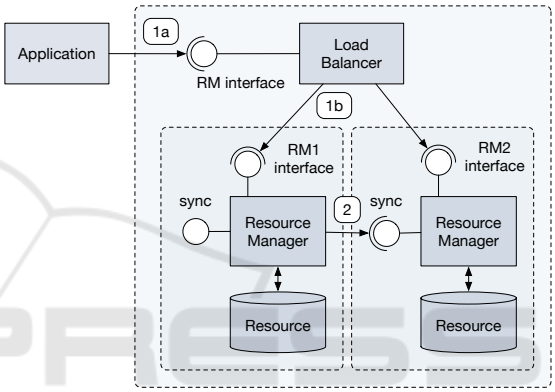

Figure 2: Architecture of a System following the BASE model.

Notice that BASE refers to properties of the TP system more so than of an individual transaction, and that BASE hardly describes any guarantee: being basically available, soft-state and eventually consistent are very weak guarantees, if any. In comparison to ACID, BASE does not address atomicity and does not support isolation. BASE compromises consistency to eventual consistency. Durability (persistence) of changes to the system is the only property common to both models.

In the context of BASE systems, the term transaction (and transaction processing, correspondingly) is misleading. A "BASE transaction" – if at all – probably is best understood as the unit of work comprising an initial client request to one system node and all subsequent steps to propagate the effects of that request to all other required nodes. However, we prefer referring to "ACID transactions" (rather than "ACID systems") and to "BASE systems" (rather than "BASE transactions") instead, reflecting the focus of each of the acronyms.

4 SALT: THE BLOCKCHAIN ALTERNATIVE TO ACID AND BASE

While ACID transactions provide convenient consistency guarantees and BASE systems scale to meet the demands of the largest of applications, they both require the users' trust. A blackbox to the clients, the correct execution of transactions can rarely be controlled. For example, when you send money to a bank account, you cannot be sure that this account was actually credited. You rather trust your bank to play by the rules and to keep balances correct.

Turning to blockchains now, an alternative to ACID and BASE is introduced. Blockchains address this trust concern and propose a system design that enables transactions which do not involve trust in a central party (therefore, the interactions become *trustless*). These transactions neither fit the ACID nor the BASE model, which is why we introduce SALT as a new acronym to describe the unique properties of blockchain-based transactions and systems.

Whereas ACID compromises availability in favor of consistency, and BASE compromises consistency in favor of availability, blockchains compromise scalability in favor of trustless interactions (Tai, 2017).

As explained before, ACID focuses on properties of transactions while BASE focuses on properties of the system. Hence, we provide two SALT perspectives: a transactions perspective, and a systems perspective.

4.1 SALT – The Transactions Perspective

Blockchain-based *transactions* can be characterized as *Sequential, Agreed, Ledgered*, and *Tamper-resistant*.

Notice that blockchains use two main types of data records: *transactions* and *blocks*. A block groups a set of transactions. The SALT transaction properties also apply to blocks, however, for now, we will concentrate on transactions and discuss blocks in section 4.2 when defining the SALT systems properties.

Sequential
All transactions are processed sequentially. Unlike with ACID transactions, there is no parallel processing of transactions in the SALT model. However, just like the isolation property in ACID, all transactions are processed in a sequential order.

Agreed
A transaction is accepted when the majority of the network agrees on its validity. Essentially, community consensus determines the system state – unlike in ACID and BASE, where it is solely controlled by a central authority.

Ledgered
All agreed-on transactions are added to an append-only transaction ledger and cannot be revoked. State changes cannot be undone; compensating transactions can be processed, however.

Note, though, that the ledgered property is a bit weaker than the durability property in ACID. Due to the majority-based agreement process, a part of the ledger may be overwritten by another, later version. The likeliness of this happening for a given transaction quickly diminishes with the number of transactions succeeding said transaction.

Tamper-Resistant
A transaction cannot be manipulated or censored. This has two dimensions: Pending and agreed-on transactions.

Unlike with ACID and BASE, there is no central access management. The access control model is completely decentralized and tied to asymmetric cryptography. Pending transactions contain a signature that only the original sender can create. There are no privileged users which could alter or block pending transactions. Furthermore, the ledger data structure is designed in a way that any alteration to an agreed-on transaction would invalidate the data structures. Hence, other nodes in the network could never be convinced of such an invalid history.

4.2 SALT – The Systems Perspective

Blockchain-basesd *systems* supporting transactions can be characterized as *Symmetric, Admin-free, Ledgered*, and *Time-consensual*.

Symmetric
All nodes in the peer-to-peer network symmetrically share their responsibilities. This does not only refer to transaction processing, but also to state storage and propagation of information.

Admin-free
As a fully peer-to-peer system, there is no concept of a system administrator who is responsible for maintenance, infrastructure provisioning or access control. Updates of peers happen on an individual basis and by that are subject to community consensus.

Ledgered
All peers in the system maintain an append-only data structure of transactions which we refer to as the

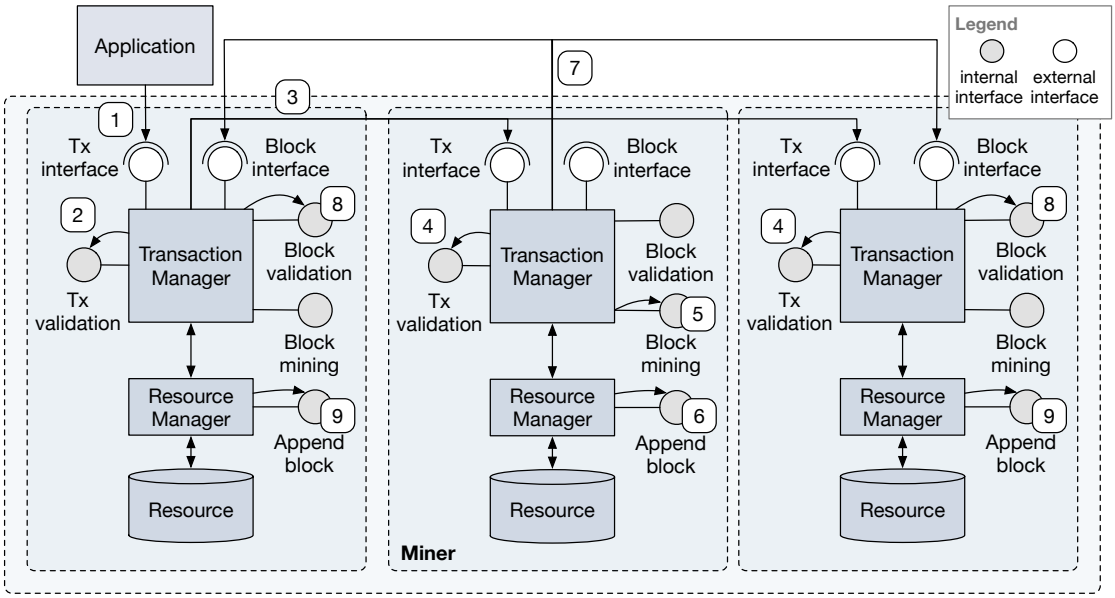

Figure 3: Transaction-oriented Architecture of the SALT system perspective.

ledger. Before transactions can be appended to that ledger, all nodes in the network need to agree on an order to maintain the ledgers consistency.

Since finding consensus is expensive, blockchains introduce *blocks*, a data structure that groups a set of transactions. Blocks are then used as the unit of consensus, thus reducing the number of consensus rounds relative to the number of transactions. When consensus is found, the agreed-on block is appended to the ledger and with it the contained transactions.

Time-consensual To ensure timely processing of transactions while tolerating network latencies due to geo-distribution of the networks' peers, the consensus algorithm targets a defined average time between the creation of two blocks. This is called the block interval. Hence, while not guaranteed, a transaction is on average processed and included into the ledger within half the block interval.

4.3 Understanding SALT

Figure 3 illustrates a TP system in support of SALT transactions. An application can begin a transaction with any of the peers in the system (1). Each peer maintains a local copy of the entire blockchain. The transaction is validated locally at the chosen node and – if valid – added to its list of "pending transactions" (2). Furthermore, the node propagates the validated transaction to other nodes in the network (3). The receiving nodes perform the same steps themselves (4), which together leads to a gossip-like spread of valid transactions and them being in pending state on many nodes.

Some of the nodes – called the miners – group their pending transactions into candidate blocks, the unit of consensus. This happens in parallel on many nodes and a consensus algorithm, (e.g., Proof-of-Work, Proof-of-Stake) is used to decide which node's candidate block becomes a valid block accepted by the network without communication between peers. This is referred to as mining (5). This agreed-on block is then appended to the miner's blockchain (6) and broadcast (7) so it spreads to all nodes in the network in a gossip-like way. The receiving nodes check the validity of the block (8) by validating all contained transactions as well as the contained proof, that the block was actually selected in the consensus process. A valid block is then appended to the node's local blockchain. As this happens on all nodes in the network, the contained transactions are now reflected in the overall system state.

4.4 Comparing SALT, ACID and BASE

A SALT transaction conceptually is simpler than an ACID transaction; there is no concept of grouping multiple operations into an atomic unit-of-work from a client perspective. Hence, no SALT property directly relates to the atomicity property of ACID. Note, however, that grouping multiple operations is possible inside a SALT system by creating stored procedures. We expand on this idea in section 4.5. Neither of both approaches is commonly supported in BASE systems.

The isolation property of ACID is closely related to the sequential processing of transactions in the

SALT model. In both models, transactions have a well-defined order that is consistent within the network. In BASE systems, in contrast, a defined global order of transactions is not enforced by the system and does not have to exist, which is why BASE systems are soft-state.

Both, the agreed-on property of SALT and the consistency property of ACID, refer to the validity of transactions and are thus closely related. While essentially the same guarantee, the parties enforcing the gurarantee differ: In the ACID case, specific components of the TP system check the post transaction database state for integrity and reject transactions in case of violations. In the SALT case, all nodes in the network independently validate transactions based on an agreed-on ruleset and acknowledge them depending on the result.

Note, that this notion of consistency in ACID and SALT is different from the system-centric consistency notion in BASE. With BASE systems, consistency refers to (reaching) equivalence of state across replicas. This notion of consistency is a pre-requisite for distributed systems in support of ACID transactions, also. Yet, ACID makes no direct statement about replica consistency. In systems supporting BASE, replicas can be in conflicting state because they each process a different set of transactions. The same can happen in systems supporting SALT, when two nodes mine different blocks and hence temporarily maintain different state. Whereas systems supporting BASE are guaranteed to converge in the absence of new transactions, in SALT systems the conflict is resolved by deciding for exactly one block to be agreed-on by all peers.

Another interesting relation exists between the ledgered property of SALT and the durability property of ACID, as already briefly discussed in 4.1. Not considering factors external to the system, a transaction written to a TP system supporting ACID or BASE is guaranteed to be persistent and cannot be lost. This guarantee is weakened to a probabilistic version in the SALT model. It can happen, that a transaction is no longer represented in the system state although it has been processed successfully at an earlier point in time. With the number of subsequent transactions, the probability of this happening quickly diminishes (Nakamoto, 2008). While having to rely on probabilistic state appears unreliable, it is barely a problem in current networks (e.g., Bitcoin or Ethereum). As a best practice, users wait for a certain number of blocks (based on their risk threshold) succeeding the block containing their transaction until they consider the transaction to be final.

4.5 T for Turing Complete?

Traditional TP systems know the concepts of stored procedures. These procedures group a sequence of database operations and can be exposed to clients. This reduces the permissions clients require and can also improve usability.

With blockchains, the same concept becomes much more powerful. The procedure is executed on every node in the network and the resulting state changes are applied. Hence, a stored procedure on a blockchain is essentially a piece of software that is executed in a fully decentralized way without anyone being able to control it.

Applying the concept of stored procedures to blockchains has two challenges, though. Because the state changes caused by the stored procedures on every node have to be consistent, the code needs to be deterministic. Furthermore, for the network to make progress and for block times to average the targeted block interval, there can be no infinite loops or long running transactions.

Determinism can be achieved by not allowing any non-deterministic operations in stored procedures. External API calls, for example, cannot be allowed, as they might yield different results when performed on different nodes in the network.

The second problem, preventing infinite loops, theoretically reduces to the halting problem: Does a given program terminate or will it run forever? It is well known, that this problem is undecidable for turing-complete languages, so another way of preventing infinite loops has to be found.

One approach is to write stored procedures in a less powerful language so they are guaranteed to terminate after a well-defined number of steps. While this constrains overall expressiveness, it enhances security by reducing potential vulnerabilities. A prominent example of a blockchain following this approach is Bitcoin, which provides a rather limited scripting language mainly used for signature verification.

Another approach is to introduce a cost model to a turing-complete language: Every operation is assigned a price and an initial endowment is specified for a stored procedure when it is called. With that, every stored procedure is guaranteed to halt, because even if it contains an infinite loop, the endowment will be used up at some point which causes execution to stop. Although it involves additional risk from a security perspective, this approach is certainly more expressive and enables more powerful applications. However, the cost calculation and tracking during execution involve considerable overhead. The Ethereum blockchain adopts this approach and refers

to its stored procedures as Smart Contracts (Buterin, 2014), (Wood, 2014).

5 DESIGNING SALTy APPLICATIONS

With an understanding of SALT transactions and SALT systems, we now turn to the question of how to design the next generation of decentralized transactional applications that use blockchains. Innovative companies are already building such applications today, which allows us to discuss SALT in the context of specific industry use cases. Besides financial applications in the banking and insurance industry, there are numerous examples of non-financial, blockchain-based applications, such as notary services, digital content monetization, decentralized storage, decentralized IoT, and so on (Crosby et al., 2015).

We propose to approach the design and analysis of blockchain-based applications along the two SALT perspectives: the transactions perspective, and the systems (data management components) perspective. We selected two non-financial use cases to illustrate the basic approach.

5.1 Monegraph Use Case

The first use case, Monegraph, is a content distribution and monetization platform that enables media creators to transfer rights of digital artwork to media owners (Monegraph, 2017). With the platform, media creators, such as photographers, videographers, and musicians, can monetize their work, and media owners can manage their media's use rights, distribute content, and share revenue with media creators, publishing partners, and affiliates.

5.1.1 The Transactions Perspective

The main business transactions from the point of view of the two main stakeholders, the media creators and the media owners, are the following, as depicted in figure 4.

- Media creators can, for example, create and upload digital artwork (as a file), register the artwork as a public record, and list their artwork in a private catalog.
- Media owners use the platform to agree to terms (in the form of a Smart Contract), make payments using Bitcoin or traditional payment transactions, and transfer ownership of the public record.

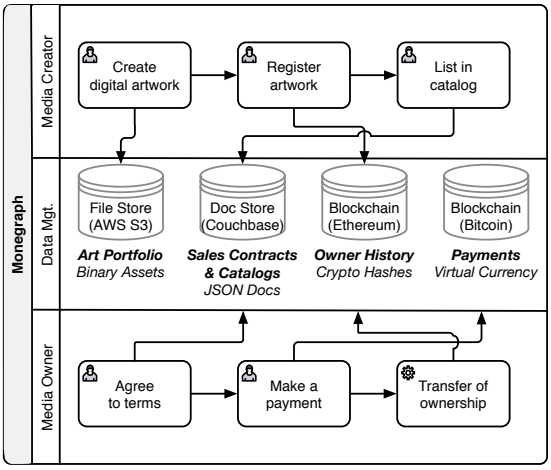

Figure 4: Monegraph use case from a combined transactions and data management systems perspective.

Using blockchain technology, Monegraph gains the following SALT transaction properties.

Sequential
Transactions are guaranteed to be processed in a sequential order so that the transfer of media rights from creators to owners is performed without "double selling", i.e., selling the same rights to different media owners.

Agreed
Since an agreement among the majority of the blockchain network is required, it becomes more difficult to illegitimately claim media rights ownership. Once an agreement on a transaction has been reached in the network, there is a certain level of trust that media rights that have been exchanged really belong to the agreed-on media owner.

Ledgered
Appending all agreed-on transactions to a public ledger asserts that media ownership records cannot be revoked by a single party or a small part of the network. Thereby, media owners gain trust that their digital rights ownership cannot be single-handedly annulled by a seller.

Tamper-resistant
All transactions are tamper-resistant, making it difficult to forge transactions, e.g., by replacing the media owner in a transaction through a man-in-the-middle attack.

5.1.2 The Systems Perspective

Monegraph uses three different types of data stores for enabling the business transactions described be-

fore:

1. A blob store for digital artworks, which stores the art portfolio as binary files (jpeg, mp4, mp3, etc.). The blob store (using AWS S3) is suitable for storing multiple Petabytes of unstructured data in a flat namespace.
2. A NoSQL data store for storing sales contracts and catalog data in JSON format. The NoSQL document store (using Couchbase) is suitable for storing Terabytes of semi-structured data.
3. Two blockchain data stores for storing Gigabytes of data:
   (a) One blockchain, Ethereum, is used for storing the ownership history of digital artworks as crypto hashes.
   (b) Another blockchain, the Bitcoin system, is used for facilitating payments with the Bitcoin virtual currency.

The two different NoSQL data stores (S3 and Couchbase) that Monegraph uses are both BASE systems which enable scalable and highly-available data storage of unstructured items in a blob store, and semi-structured JSON items in a document store, respectively. It would be prohibitively expensive to use the Ethereum blockchain as a data store for these items. Using the Ethereum blockchain as part of the data management system, however, enables, in particular, the following SALT system properties that simplify revenue sharing and payment distribution.

Ledgered
Transparently storing all agreed-on transactions in a public ledger simplifies revenue sharing between multiple parties, such as publishers, media creators, and affiliates, without layers of intermediaries that use their own complex accounting and rights management processes. For example, a third-party that wants to license media can more easily verify media ownership of the party that claims to be the media owner by looking up the ownership record in the public blockchain.

Time-consensual
The time-consensual property implies that transactions are appended to the ledger after a defined average time period. Although this could be problematic for some applications, a certain delay involved in transferring media rights could still be acceptable in this use case.

5.2 Provenance Use Case

The second use case, Provenance, is a supply chain traceability application that provides insight into the provenance of items across different supply chains (Provenance, 2017). Thereby, Provenance makes it possible for consumers to transparently retrace the origin of products that they purchase. A pilot project shows how blockchain technology can be used for tracing yellowfin and skipjack tuna fish in Indonesia from catch to consumer.

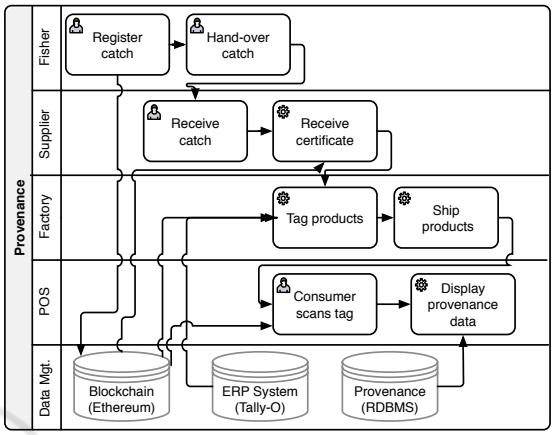

Figure 5: Provenance use case from a combined transactions and data management systems perspective.

5.2.1 The Transactions Perspective

Provenance uses blockchain technology to integrate different stakeholders across the supply chain, enabling them to support the following business transactions, as shown in figure 5.

1. Local fishermen in Indonesia register their catch (by sending an SMS message to a Provenance application), which triggers the creation of a public digital certificate that represents the physical good in the blockchain.
2. Along with the physical transaction of the catch from fisher to supplier, suppliers receive the digital certificate from the blockchain.
3. In the factory that processes and packages the fish, the products are tagged (e.g., using QR codes, barcodes, NFC tags, etc.), and can thereby be traced along the supply chain by linking the tag to the digital certificate.
4. At the Point of Sale, consumers scan the tag on a product (e.g., a QR code) with their mobile phones and retrieve information regarding the product's provenance.

In the tuna fish tracing pilot project of Provenance, NGOs perform audits to validate compliance along the supply chain. Therefore, the system is not completely trustless. However, audits are simplified by

using the blockchain as a shared data layer for certificates.

Using blockchain technology for building the certification system enables, in particular, the following SALT transaction properties.

Agreed
There is no single party that governs the certification process. Instead, an agreement between all stakeholders is automatically formed based on defined conventions of the blockchain consensus protocol. The certification data is thereby not prone to arbitrary actions of a single party.

Ledgered
The agreed-on transactions in a blockchain cannot be revoked, thereby providing a perfectly auditable system which is particularly important for NGOs who are responsible for auditing the uninterrupted chain of custody from raw fish to packaged products at the point of sale. It is practically impossible for a single party to modify transactions once they have been appended to the blockchain.

5.2.2 The Systems Perspective

Provenance uses the Ethereum blockchain as a secure, auditable, shared data layer, thereby avoiding data silos in heterogeneous systems with limited interoperability. With the blockchain, data can be accessed and verified by all involved parties, rather than solely by the original certificatory.

By using a blockchain-based system in the tuna tracing pilot project, Provenance gains two main advantages. First, the blockchain serves as an integration layer between different stakeholders with heterogeneous IT systems that range from the mobile phones of fishermen to complex ERP systems of manufacturers. Second, the peer-to-peer setup and cryptographic properties of the system make it impossible for a single governing party to manipulate data at some point in the supply chain, thereby strengthening the confidence of consumers into the provenance information they retrieve at the point of sale.

The following SALT system properties are of particular importance in the Provenance use case.

Symmetric
Fragmentation along the chain of custody makes a certification system vulnerable to fraud. Furthermore, no single organization can be trusted to broker all data about every product's supply chain. Relying on a single party creates an inherent bias that eventually could weaken trust of consumers regarding the certified provenance information that they receive at the point of sale. A blockchain-based peer-to-peer networking approach with symmetrically shared responsibilities takes the role of an "honest broker" in such a distributed certification system.

Admin-free
The blockchain serves as an integration layer between different stakeholders with heterogeneous IT systems that are ranging from the mobile phones of local producers to the ERP systems of manufacturers. Traditionally, a governing third-party, such as an NGO, would have been required to integrate all IT systems along the supply chain with their own auditing system, thereby increasing cost of operations. Cost sharing is difficult and could weaken the trustworthiness of the system due to adverse incentives of the governing third-party who might be more interested in pleasing the sponsor rather than performing proper oversight.

Moreover, by using a blockchain, the authentication system is drastically simplified compared to centrally managed public key infrastructure. This removes a critical single-point-of-failure and security risk from the certification system.

6 SUMMARY AND OUTLOOK

Different notions of transactions and transaction processing systems exist, most notably ACID transactions and BASE systems. With the emergence of blockchain technology, an alternative to ACID and BASE is introduced: SALT. In this paper, we introduced SALT along the two perspectives of transactions and TP systems. We discussed the importance of these dual perspectives.

When engineering blockchain-based applications, the SALT model can be used to explain the blockchains under consideration, that is, to study both the business transactions and their supporting systems. While there are specific transactions that are exclusively SALT, we expect the systems to always encompass different data management components so that SALT, ACID, and BASE can all be supported at the same time. The two use cases discussed are first examples of such systems that benefit from using multiple transactional models and systems.

This immediately raises two main research questions that require further exploration:

1. Given a combination of different transaction models and systems, including SALT, are there system-wide application properties that can be guaranteed?

2. With blockchain technology advancing at a rapid pace, will ACID, BASE and SALT continue to coexist as alternatives, or can frameworks and solutions stacks be designed that impose an integrated data management using all three models?

Further, a number of related research challenges emerge:

1. In this paper, we suggested a business transactions and a systems perspective when engineering blockchain-based applications. Other perspectives – for example, a consensus-perspective, or a blockchain technology (platforms and tools) perspective – may also be beneficial.
2. Private and permissioned blockchains, as opposed to the public blockchains discussed in this paper, maintain some of the SALT properties, while compromising others. Whenever closed networks with different consensus algorithms and asymmetric peers are suggested – is less SALT actually healthy?
3. Scalability (of blockchains and blockchain-based applications) is a major limitation of current blockchain technology. Heterogeneous data management, as exemplified with the two use cases above, may help to improve scalability (and address related concerns of data volume, distribution, and processing needs). How, and at what costs, can scalability be improved?

The list above is by no means complete and final; our discussion in this paper is largely driven from a data management and distributed systems interest, while several other communities (including security, privacy, legal, economics) also formulate a number of open issues related to blockchains and blockchain-based applications. As blockchain technology and blockchain-based applications continue to evolve at a rapid pace and more lessons are learned, we expect both a fine-tuning of specific open issues and a more integrated, inter-disciplinary big picture to become important next steps.

REFERENCES

Bernstein, P. A. and Newcomer, E. (2009). *Principles of Transaction Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition.

Brewer, E. A. (2000). Towards robust distributed systems. In *PODC*, volume 7.

Buterin, V. (2014). Ethereum: A next-generation smart contract and decentralized application platform. *https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-White-Paper*.

Crosby, M., Nachiappan, Pattanayak, P., Verma, S., and Kalyanaraman, V. (2015). Blockchain technology: Beyond bitcoin. Technical report, Sutardja Center for Entrepreneurship & Technology, Berkeley, CA.

Gray, J. and Reuter, A. (1992). *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

Monegraph (2017). https://monegraph.com/. Accessed: 2017-02-07.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Pritchett, D. (2008). BASE: An acid alternative. *ACM Queue*, 6(3):48–55.

Provenance (2017). https://www.provenance.org/. Accessed: 2017-02-07.

Tai, S. (2017). Continuous, trustless, and fair: Changing priorities in services computing. In *Advances in Service-Oriented and Cloud Computing (ASOCC)*. Springer.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*.