BigchainDB 2.0
The Blockchain Database

BigchainDB GmbH, Berlin, Germany

May 2018
Paper version 1.0

Abstract

BigchainDB is software that has blockchain properties (e.g. decentralization, immutability, owner-controlled assets) and database properties (e.g. high transaction rate, low latency, indexing & querying of structured data). It was first released—open source—in February 2016 and has been improved continuously ever since. BigchainDB version 2.0 makes significant improvements over previous versions. In particular, it is now Byzantine fault tolerant (BFT), so up to a third of the nodes can fail *in any way*, and the system will continue to agree on how to proceed. BigchainDB 2.0 is also production-ready for many use cases. In this paper, we review the design goals of BigchainDB 2.0 and how they were achieved, we explore some use cases, we show how BigchainDB fits into the overall decentralization ecosystem, we follow the life of a transaction to understand how BigchainDB 2.0 works, we note ways to try BigchainDB, we outline how you can contribute, and we summarize future plans.

1 BigchainDB 2.0 Design Goals

BigchainDB was first announced in February 2016 [1], along with the original whitepaper and the first software release (version 0.1). BigchainDB is called a *blockchain database* because it has some blockchain properties and some database properties. The original design started with a database and added some blockchain characteristics such as decentralization, immutability, and owner-controlled assets. The idea was that the resulting system would inherit the desirable properties of the database such as low latency, high transaction rate and high capacity.

It worked, mostly, but there were some issues. One issue was that the resulting system couldn't withstand arbitrary faults (in a bounded subset of the nodes); it wasn't Byzantine fault tolerant (BFT) [2]. Another issue was that the underlying database has a primary (or master) node, and that node does all the writes. The other nodes just replicate whatever the primary writes. The primary node could be the primary for a long time. It was a single point of control and a single point of failure. Another issue was that there was only one logical database, so if a malicious agent managed to get admin privileges, they could delete the entire database (across all the nodes) with a single command.

BigchainDB 2.0 was designed to resolve all those issues while retaining the desirable database and blockchain properties. Below, we go through each design goal of BigchainDB 2.0 and outline how it was achieved.

	Typical Blockchain	Typical Distributed Database	BigchainDB
Decentralization	✓		✓
Byzantine Fault Tolerance	✓		✓
Immutability	✓		✓
Owner-Controlled Assets	✓		✓
High Transaction Rate		✓	✓
Low Latency		✓	✓
Indexing & Querying of Structured Data		✓	✓

1.1 Full Decentralization and Byzantine Fault Tolerance

BigchainDB 2.0 uses Tendermint [3, 4] for all networking and consensus. Each node has its own local MongoDB database [5], and all communication between nodes is done using Tendermint protocols. One consequence is that the resulting system is BFT, because Tendermint is BFT. Another result is that if a malicious hacker manages to get admin privileges to one of the local MongoDB databases, then the worst they can do is corrupt or delete the data in that local database; the MongoDB databases in the other nodes won't be affected. Tendermint still has something like a primary node (the current block proposer), but it changes with each round (using round robin) and the Tendermint developer team has an open issue to make that more secure (e.g. less predictable) [6].

If every node in a BigchainDB 2.0 network is owned and operated by a different person or entity, then it's a decentralized network because it has no single owner, no single point of control, and no single point of failure. Ideally, the nodes should be located in many countries, legal jurisdictions and hosting providers, so an issue with one doesn't affect them all. Any node can fail and the rest of the network will continue to operate. In fact, up to one third of the nodes can fail[1] in arbitrary ways, and the rest of the network will continue to work, i.e. the non-faulty nodes will agree on how to proceed.

1.2 Immutability

Once data gets stored in a BigchainDB network, it can't be changed or erased, or at least not without great difficulty. If some data somehow manages to get changed or erased, then that is detectable. One might say that the storage

[1]Technically, up to one third of the *voting power* can fail, but we usually run BigchainDB networks with all nodes having the same voting power, so one third of voting power is the same as one third of the nodes.

is "practically immutable" but in the blockchain world, one usually just says "immutable."

BigchainDB uses several strategies to achieve practical immutability. The simplest one is that there are no BigchainDB-provided APIs to change or erase stored data.

Another strategy is that every node has a full copy of all the data in a standalone MongoDB database (i.e. there is no global MongoDB database). Even if one node gets corrupted or destroyed, the other nodes won't be affected and will still have a copy of all the data.

Another strategy is that all transactions are cryptographically signed. After a transaction is stored, changing its contents will change the signature, which can be detected (unless the public key is also changed, but that should also be detectable because every block of transactions is signed by the a node, and the public keys of all the nodes are all known).

1.3 Owner-Controlled Assets

Like most blockchains, BigchainDB has a concept of owner-controlled assets. Only the owner (or owners) of an asset can transfer that asset. (The owners are the holders of a particular set of private keys.) Not even a node operator can transfer an asset.

In most blockchains, there's only one built-in asset (e.g. Bitcoin or Ether), but BigchainDB allows external users to create as many assets as they need. However, it's worth noting that a user can't create assets that appear to be created by someone else. All assets created by Mike are cryptographically signed by Mike. For example, a user named Joe might decide to create 1000 "Joe tokens." He would do that by building a BigchainDB CREATE transaction, signing it with his private key, and sending it to a BigchainDB network. (Later in this paper, we trace what happens to a transaction when it arrives at a BigchainDB network.) Initially, Joe might own all 1000 tokens, so only he could transfer them to others. He could transfer 37 tokens to Lisa by creating a BigchainDB TRANSFER transaction with two outputs: one with an amount of 37 tokens with a condition that only Lisa can transfer it, and the other with all remaining tokens ($1000 - 37 = 963$ tokens) with a condition that only Joe can transfer it.[2]

BigchainDB checks every transaction to make sure it's not trying to transfer an output that was already transferred (spent) by another transaction, i.e. it prevents double-spending. It also checks many other things, all of which are listed in the BigchainDB Transaction Spec v2 [7].

1.4 High Transaction Rate

One design goal of BigchainDB has always been the ability to process a large number of transactions each second. That's still true with BigchainDB 2.0.

BigchainDB 2.0 was still in Alpha at the time of writing. Performance tests were being written and started, but no concrete results were available yet. (The final, stable BigchainDB 2.0 is expected in June of 2018.) However,

[2]BigchainDB allows for quite complex conditions on the outputs. For example, an output could have a condition which says that "Jamie OR (Pat AND Kelly) OR (three of {Casey, Drew, Laurie, Riley or Whitney})" must sign.

since BigchainDB 2.0 is based on Tendermint, we can look at other Tendermint-based networks to get a sense of what can be expected. According to the Cosmos whitepaper [8]:

> Despite its strong guarantees, Tendermint provides exceptional performance. In benchmarks of 64 nodes distributed across 7 datacenters on 5 continents, on commodity cloud instances, Tendermint consensus can process thousands of transactions per second, with commit latencies on the order of one to two seconds. Notably, performance of well over a thousand transactions per second is maintained even in harsh adversarial conditions, with validators crashing or broadcasting maliciously crafted votes.

1.5 Low Latency and Fast Finality

Tendermint-based networks (such as BigchainDB networks) take only a few seconds (or less) for a transaction to be included in a new committed block. Once that happens, there's no way it can be reverted or considered defunct in the future, because Tendermint doesn't do forking.

1.6 Indexing & Querying Structured Data

Each node in a BigchainDB 2.0 network has its own local MongoDB database. That means that each node operator has access to the full power of MongoDB for indexing and querying the stored data (transactions, assets, metadata and blocks, all of which are JSON strings). Each node operator is free to decide how much of that power they expose to external users. One node operator might decide to index geospatial data and offer optimized geospatial queries via a REST API, whereas another node operator might decide to offer a GraphQL API.

By default, BigchainDB 2.0 creates some MongoDB indexes and the BigchainDB HTTP API includes some endpoints for doing basic queries. However, as outlined in the previous paragraph, each node operator can add additional indexes and query APIs.

1.7 Sybil Tolerance

Some blockchain networks (such as Bitcoin) allow anyone to add their node to the network. That brings the concern that someone could add so many nodes that they effectively control the network: a Sybil attack [9]. Bitcoin makes Sybil attacks unlikely by making them prohibitively expensive. In a BigchainDB network, the governing organization behind the network controls the member list, so Sybil attacks are not an issue.

2 Use Cases

BigchainDB, with the capabilities mentioned in the previous section, can serve a multitude of use cases. Wherever there is a need of immutable, tamper-resistant data representing digital assets, BigchainDB can be used. There are several industry verticals which can directly benefit from BigchainDB features. In the following sub-sections, we will briefly touch some of these industry verticals and discuss how BigchainDB can be of help in these scenarios.

2.1 Supply Chain

In a typical supply chain scenario, there are several parties/entities collaborating and exchanging information with each other. Primarily, this information is about the processes of tracking goods being manufactured until they reach the logical end of the supply chain cycle (retailers/end-consumers). The major challenge faced by these entities collaborating in a supply chain scenario is management and security of the information being shared. Eventually, several silos of data emerge and it becomes hard to manage. That's where blockchain technology, in general, can help organizing this data in a shared system so that the overall management of information becomes easy. Because of immutability and tamper resistance, blockchain also provides a layer of trust to the collaborating entities so that they can trust the data even when they don't trust each other. While other blockchains and decentralized systems can help organize data in a shared system,they are not optimized for the high throughput and query capabilities required for a supply chain scenario. That's where BigchainDB shines, bringing along it's query capabilities and high throughput performance. When used in a supply chain scenario, like a regular blockchain, BigchainDB helps organize data in a decentralized system, but, in addition, allows the users to query the data to generate reports and do calculations on the fly.

2.2 Intellectual Property Rights Management

When it comes to IP rights management and provenance, blockchain technology has several benefits as it helps provide immutability to artists' claims. Once an art asset is registered on a blockchain with proper attribution, it can be used to prove the ownership of the IP rights. Ownership transfers can also be recorded. For this reason, we created ascribe [10] in 2014 and it was based on the Bitcoin blockchain. However, soon the throughput of the Bitcoin public blockchain became a bottleneck. That's when we envisioned BigchainDB and began building it. BigchainDB 2.0 also handles high throughput and hence is an excellent choice for IP use-cases.

2.3 Digital Twins and IoT

Digital twins are the digital representation of physical objects which can be tracked on the basis of verification of authenticity and provenance, proof of ownership, life-cycle traceability, and input data from IoT devices & sensors. To manage this scale of information, giving every product and object a story of its own, we need a high-throughput tamper-resistant system which can also serve results quickly. That's BigchainDB.

2.4 Identity

Identity is one of the most critical pieces when it comes to managing user-specific information. It has become ever more important in scenarios like IoT and Digital Twins where even machines have an identity. With identity theft becoming one of the major concerns of today, we need to make sure that identity of a human or a machine is self-sovereign and hack-proof. Because of a large amount of data associated with identities, we also need to make sure that the systems handling identity-related data are capable of handling high scale. That's when BigchainDB becomes a natural fit for solving identity-related use cases because of its combined characteristics of blockchains and databases.

2.5 Data Governance

Data governance use cases are scenarios where multiple independent organizations or entities collaborate on defining common processes and directives. Today, the major challenge with data governance is lack of collaboration and trust. Also, there are no clear incentives for the participants to collaborate on topics of governance. BigchainDB helps solve these challenges in data governance by providing an incentive-driven, easy-to-integrate platform. The approach is to model data governance topics, feedback and economic incentives as BigchainDB assets. Once we model a data governance system as a collection of BigchainDB assets, it becomes fairly straightforward to collaborate and define processes on top of them as the data is shared using a common substrate and all participants can be incentivised to participate.

2.6 Immutable Audit Trails

Immutable audit trails are one of the generic use cases of BigchainDB. While not directly linked with any industry vertical, they helps solve a lot of track and trace challenges across verticals. From banking to supply chain, from utilities to access control, audit trails are heavily relied upon. It really adds a lot of value if these audit trails are immutable and easy to query. BigchainDB, because of high throughput and query capability becomes a natural fit for maintaining immutable audit trails.

2.7 Some Closing Remarks on Use Cases

In general, BigchainDB can be used in almost all the scenarios where there is a need of immutable and tamper-proof storage of data assets at high throughput with ability to search and query. BigchainDB can also be used by groups of people or organizations who want to have a shared database, even if they don't trust (or know) each other. One must be careful about the data one stores in a BigchainDB network. For example, storing personally-identifiable information (PII) is discouraged, because many countries have regulations that require PII to be erasable upon request.

3 BigchainDB in the Decentralization Ecosystem

As a blockchain database, BigchainDB is complementary to other decentralized systems, such as decentralized file storage (e.g. IPFS [11]), decentralized data exchange protocols (e.g. Ocean Protocol [12]), smart-contract blockchains (e.g. Ethereum [13, 14] or Hyperledger Fabric [15]), and decentralized processing (e.g. TrueBit [16]). BigchainDB works with centralized computing systems as well. Figure 1 illustrates some ways BigchainDB could be used in various technology stacks.

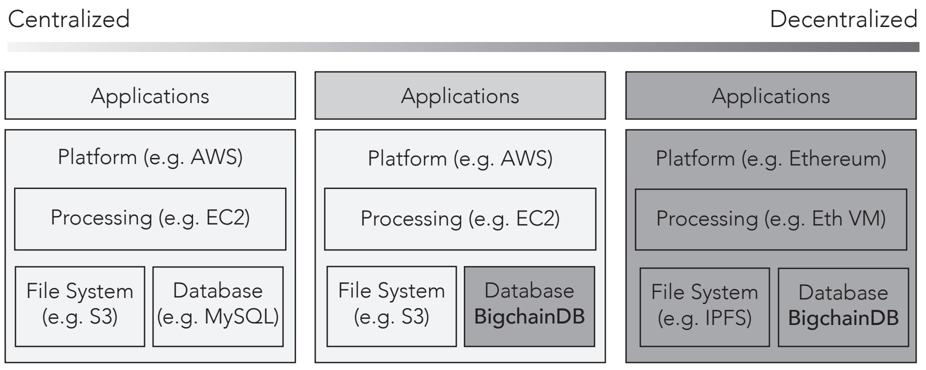

Figure 1: From a base context of a centralized cloud computing ecosystem (left), BigchainDB can be added as another database to gain some decentralization benefits (middle). It also fits into a fully-decentralized technology stack (right).

4 The Life of a BigchainDB Transaction, or How BigchainDB 2.0 Works

One can get a good sense of how BigchainDB 2.0 works by following the life of a BigchainDB transaction.

4.1 BigchainDB Transactions

A BigchainDB transaction is a JSON string that conforms to a BigchainDB Transactions Specification (Spec). At the time of writing, there were two such specs: v1 and v2. Transactions conforming to BigchainDB Transactions Spec v1 were accepted by BigchainDB versions 1.0–1.3; such transactions are no longer supported. Transactions conforming to BigchainDB Transactions Spec v2 [17] are accepted by BigchainDB version 2.0 (i.e. the latest version at the time of writing). Each transactions spec explains the expected keys and values (including what they mean), instructions for how to construct a transaction, a list of checks that must be done to check if a transaction is valid, and details of the cryptographic primitives used. Listing 1 shows an example BigchainDB transaction (v2).

If someone wants to construct a valid BigchainDB transaction, then they'll typically use a BigchainDB driver (software package). There's a list of BigchainDB drivers in the BigchainDB docs [18].

```
{
  "id": "3667c0e5cbf1fd3398e375dc24f47206cc52d53d771ac68ce14d⌟
  ↪  df0fde806a1c",
  "version": "2.0",
  "inputs": [
    {
      "fulfillment": "pGSAIEGwaKW1LibaZXx7_NZ5-V0alDLvrguGLyL⌟
      ↪  RkgmKWG73gUBJ2Wpnab0Y-4i-kSGFa_VxxYCcctpT8D6s4uTGOO⌟
      ↪  F-hVR2VbbxS35NiDrwUJXYCHSH2IALYUoUZ6529Qbe2g4G",
      "fulfills": null,
      "owners_before": [
        "5RRWzmZBKPM84o63dppAttCpXG3wqYqL5niwNS1XBFyY"
      ]
    }
  ],
  "outputs": [
    {
      "amount": "1",
      "condition": {
        "details": {
          "public_key":
          ↪  "5RRWzmZBKPM84o63dppAttCpXG3wqYqL5niwNS1XBFyY",
          "type": "ed25519-sha-256"
        },
        "uri": "ni:///sha-256;d-_huQ-eG-QQD-GAJpvrSsy7lLJqyNh⌟
        ↪  tUAs_own7aTY?fpt=ed25519-sha-256&cost=131072"
      },
      "public_keys": [
        "5RRWzmZBKPM84o63dppAttCpXG3wqYqL5niwNS1XBFyY"
      ]
    }
  ],
  "operation": "CREATE",
  "asset": {
    "data": {
      "message": "Greetings from Berlin!"
    }
  },
  "metadata": null
}
```

Listing 1: An Example BigchainDB Transaction (v2)

4.2 Sending a Transaction to a BigchainDB Network

Once one has a transaction, they can send it to a BigchainDB network using the BigchainDB HTTP API [19]. More specifically, one would use one of the following endpoints, with the transaction in the body of the HTTP request:

```
POST /api/v1/transactions
POST /api/v1/transactions?mode=async
POST /api/v1/transactions?mode=sync
POST /api/v1/transactions?mode=commit
```

Later, we'll see what the different modes mean. A BigchainDB driver could also be used to post a transaction. The HTTP request (containing the transaction) can be sent to any of the nodes in the BigchainDB network, or even more than one. Figure 2 illustrates the main components in a four-node BigchainDB 2.0 network, and how they communicate with each other.

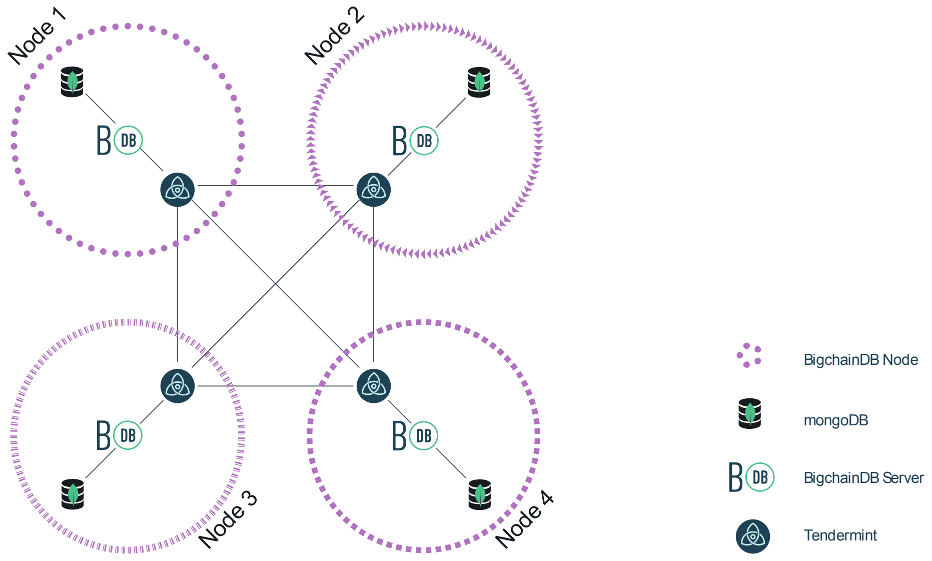

Figure 2: Communications in a Four-Node BigchainDB 2.0 Network

4.3 Arrival of a Transaction at a Node

The details of what happens next can vary. Let's skip those details and assume that the HTTP request arrives successfully at the Gunicorn web server inside a BigchainDB node, because that's where all incoming HTTP requests should get routed. Gunicorn exposes a standard interface (Web Server Gateway Interface [WSGI]) which enables Python applications to talk to it. (WSGI is a Python standard; the spec is in Python Enhancement Proposal 3333 [20].) BigchainDB uses the Flask web application development framework to simplify working with WSGI/Gunicorn.

Flask is used to route the request to a Python method for handling that endpoint. That method checks the validity of the transaction. If it's not valid, then that's the end of the story for the transaction, the HTTP response status code is 400 (Error), and the response body gives some information about what was invalid. If the transaction is valid, then it's converted to Base64 and put into a new JSON string with some other information (such as the mode). BigchainDB then sends that string to the local Tendermint instance in the body of an HTTP POST request. That request uses the Tendermint Broadcast API [21]. (Tendermint has other APIs.)

4.4 Arrival of a Transaction at a Tendermint Instance

To learn what happens to the transaction once it arrives in the local Tendermint instance, one should read the Tendermint docs about the Broadcast API [21]. At the time of writing, those docs said:

> When a transaction is sent to a Tendermint node, it will run via `CheckTx` against the application. If it passes `CheckTx`, it will be included in the mempool, broadcast to other peers, and eventually included in a block.
>
> Since there are multiple phases to processing a transaction, we offer multiple endpoints to broadcast a transaction:
>
> `/broadcast_tx_async`
>
> `/broadcast_tx_sync`
>
> `/broadcast_tx_commit`
>
> These correspond to no-processing, processing through the mempool, and processing through a block, respectively. That is, `broadcast_tx_async`, will return right away without waiting to hear if the transaction is even valid, while `broadcast_tx_sync` will return with the result of running the transaction through `CheckTx`. Using `broadcast_tx_commit` will wait until the transaction is committed in a block or until some timeout is reached, but will return right away if the transaction does not pass `CheckTx`...
>
> The benefit of using `broadcast_tx_commit` is that the request returns after the transaction is committed (i.e. included in a block), but that can take on the order of a second. For a quick result, use `broadcast_tx_sync`, but the transaction will not be committed until later, and by that point its effect on the state may change.

The above text requires some explanation:

- `CheckTx` is an API that Tendermint expects BigchainDB to implement. It's explained below.
- If someone uses BigchainDB's `POST /api/v1/transactions?mode=async` endpoint to send the transaction, then the Tendermint `/broadcast_tx_async` endpoint will be used. Similar things can be said for the sync and commit modes. If no mode was specified, then the default is async.

- Every Tendermint instance has a local mempool (memory pool) of transactions which have passed initial validation, but haven't been included in a block yet.

When Tendermint wants to determine if a transaction is valid, it sends the transaction to BigchainDB using a `CheckTx` request. It expects BigchainDB to implement `CheckTx`, and several other message types, all of which are explained in the ABCI Specification [22]. (ABCI stands for Application BlockChain Interface.) In particular, BigchainDB implements:

- `InitChain`
- `Info`
- `CheckTx`
- `BeginBlock`
- `DeliverTx`
- `EndBlock`
- `Commit`

Tendermint takes care of proposing new blocks (each of which contains a set of transactions) and making sure that all the nodes agree on the next block in a Byzantine fault tolerant way. Each BigchainDB instance keeps track of the block-under-construction by collecting the transactions delivered (by `DeliverTx`) between the `BeginBlock` and `EndBlock` calls.

When Tendermint delivers a transaction to a BigchainDB instance using `DeliverTx`, BigchainDB checks the validity of the transaction again. If it's valid, it keeps the transaction around (in memory), but it *doesn't* write anything to MongoDB yet. BigchainDB waits for the `Commit` message before writing the new block (and all contained transactions) to MongoDB.

When storing a transaction in MongoDB, BigchainDB removes the asset.data value (JSON string) and stores it in a separate MongoDB collection of assets. It does that to facilitate the text search of assets. Similarly, the metadata value (JSON string) is also removed and stored in a separate collection.

Tendermint writes the block to the blockchain (stored in a local LevelDB database) after getting a response to the `Commit` message.

We glossed over some details above, because our goal was to give a big-picture overview. If you want all the details, then you're in luck! All the code in question is open source, so you can look it up and read exactly what it does.

5 Further Reading (Technical Reading)

- BigchainDB Transactions Spec v2 [7]
- BigchainDB HTTP API [19]
- BigchainDB Enhancement Proposals [23]
- Tendermint Documentation [24]

- Tendermint ABCI Specification [25]
- Papers about Tendermint found by Google Scholar [26]

6 How to Try BigchainDB

If you want to try BigchainDB, then you need a BigchainDB network to connect to. You could deploy your own network, or you could use the BigchainDB Testnet, a live BigchainDB network operated by the BigchainDB development team.

If you go to the Get Started page on the BigchainDB website [27], you can enter some text and click "Off you go." An in-page JavaScript app will build a BigchainDB transaction and send it to the BigchainDB Testnet. You can see the constructed transaction and check if it was actually stored.

You can also write your own app to write to and read from a BigchainDB network. It can be written using any programming language, but there are official BigchainDB drivers (software packages) for JavaScript and Python, so we recommend using one of those. Links can be found in the Drivers & Tools page of the docs [18]. The docs for those drivers include example code.

Once you've written an app, you could test it against the BigchainDB Testnet, but first you'll need access credentials. You can get those (for free) by signing up for an account at testnet.bigchaindb.com.

7 How to Contribute Ideas or Code

We changed our processes to make it easier for anyone to contribute ideas or code to BigchainDB. Questions can be asked in one of the BigchainDB chat rooms on Gitter [28]. Bug reports can be submitted by creating a new issue in the relevant GitHub repository. Changes to code can be made by submitting a pull request to the relevant GitHub repository. We use a variation of the Collective Code Construction Contract (C4) [29] to guide how we handle pull requests. Detailed feature requests can be made by submitting a BigchainDB Enhancement Proposal (BEP) as a pull request in the bigchaindb/BEPs repository [30]. The format of a BEP should follow the outline given in our variant of the Consensus-Oriented Specification System (COSS) [31].

8 The BigchainDB Roadmap

We aim to release the final, stable BigchainDB 2.0 in June of 2018. No new features will be added before then; we're mostly doing testing (of all kinds). We'll fix any issues that arise and report the results of performance tests once we have them.

Starting with BigchainDB 2.0, there will always be tools and documentation to help with migration (including data migration) to future versions. BigchainDB 2.0 is production-ready for many uses cases.

There's an online BigchainDB Roadmap [32] that lists our medium-term goals. It's updated fairly often. One goal is to create some example projects illustrating how one can use BigchainDB with other decentralized systems, such as Ethereum. Longer-term, we want to make it possible to create a public BigchainDB network where anyone can add a node without getting permission.

9 Conclusion

BigchainDB is a blockchain database: it has both blockchain properties and database properties. That combination makes it useful for a wide variety of use cases, including supply chain, IP rights management, digital twins & IoT, identity, data governance and immutable audit trails. BigchainDB 2.0 includes significant improvements, including the integration of Tendermint for inter-node networking and Byzantine fault tolerant (BFT) consensus. BigchainDB 2.0 is now production-ready for many uses cases. For more information, see the BigchainDB website at bigchaindb.com.

References

[1] Carlo Thomas. ascribe announces scalable blockchain database BigchainDB. *CoinReport*, February 2016. `https://coinreport.net/ascribe-announces-scalable-blockchain-database-bigchaindb/`.

[2] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, July 1982. `http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf`.

[3] Jae Kwon. Tendermint: Consensus without Mining, fall 2014.

[4] Tendermint. `https://tendermint.com/`.

[5] MongoDB. `https://www.mongodb.com`.

[6] Anton Kaliaev (melekes on GitHub). tendermint/tendermint Issue #763: Introduce randomness into proposer selection? `https://github.com/tendermint/tendermint/issues/763`.

[7] BigchainDB Transactions Spec v2. `https://github.com/bigchaindb/BEPs/tree/master/13`.

[8] Cosmos: A Network of Distributed Ledgers. `https://cosmos.network/resources/whitepaper`.

[9] John R Douceur. The Sybil Attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002. `http://research.microsoft.com/pubs/74220/IPTPS2002.pdf`.

[10] ascribe. `https://www.ascribe.io/`.

[11] J. Benet. IPFS – Content Addressed, Versioned, P2P File System. `http://static.benet.ai/t/ipfs.pdf`, 2014.

[12] Ocean Protocol: A Decentralized Substrate for AI Data & Services. `https://oceanprotocol.com/tech-whitepaper.pdf`.

[13] Ethereum. https://ethereum.org/.

[14] Vitalik Buterin. Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform. http://blog.lavoiedubitcoin.info/public/Bibliotheque/EthereumWhitePaper.pdf.

[15] Hyperledger Fabric. https://www.hyperledger.org/projects/fabric.

[16] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. November 2017.

[17] BEP-13: BigchainDB Transactions Spec v2. https://github.com/bigchaindb/BEPs/tree/master/13.

[18] BigchainDB Drivers & Tools. https://docs.bigchaindb.com/projects/server/en/latest/drivers-clients/index.html.

[19] BigchainDB HTTP API. https://docs.bigchaindb.com/projects/server/en/latest/http-client-server-api.html.

[20] PEP 3333 – Python Web Server Gateway Interface v1.0.1. https://www.python.org/dev/peps/pep-3333/.

[21] Tendermint Broadcast API. http://tendermint.readthedocs.io/projects/tools/en/master/using-tendermint.html#broadcast-api.

[22] Tendermint ABCI Specification. https://github.com/tendermint/abci/blob/master/specification.rst.

[23] BigchainDB Enhancement Proposals. https://github.com/bigchaindb/BEPs.

[24] Tendermint Documentation. http://tendermint.readthedocs.io/projects/tools/en/master/index.html.

[25] ABCI Overview. http://tendermint.readthedocs.io/projects/tools/en/master/introduction.html#abci-overview.

[26] Papers about Tendermint found by Google Scholar. https://scholar.google.de/scholar?hl=en&as_sdt=0%2C5&q=Tendermint&btnG=.

[27] BigchainDB – Get Started. https://www.bigchaindb.com/developers/getstarted/.

[28] BigchainDB chat rooms. https://gitter.im/bigchaindb/home.

[29] BEP-1: The BigchainDB Variant of the Collective Code Construction Contract (C4). https://github.com/bigchaindb/BEPs/tree/master/1.

[30] The bigchaindb/BEPs Repository on Github. https://github.com/bigchaindb/BEPs.

[31] BEP-2: The BigchainDB Variant of the Consensus-Oriented Specification System. https://github.com/bigchaindb/BEPs/tree/master/2.

[32] BigchainDB Roadmap. https://github.com/bigchaindb/org/blob/master/ROADMAP.md.