Posted November 22, 2015 by Gideon Greenspan (http://www.multichain.com/blog/author/gdg/) in Private blockchains (http://www.multichain.com/blog/category/private-blockchains/).

Avoiding the pointless blockchain project

How to determine if you've found a real blockchain use case

Blockchains are overhyped. There, I said it. From Sibos (https://www.sibos.com/conference/conference-programme/new-kids-blockchain-platform) to Money20/20 (http://www.coindesk.com/bitcoin-blockchain-money2020-2015-pictures/) to cover stories of The Economist (http://www.economist.com/printedition/covers/2015-10-29/ap-e-eu-la-me-na-uk) and Euromoney (http://www.euromoney.com/Article/3501936/Getting-to-grips-with-blockchain.html), everyone seems to be climbing aboard the blockchain wagon. And no doubt like others in the space, we're seeing a rapidly increasing number of companies building proofs of concept on our platform (http://www.multichain.com/) and/or asking for our help.

As a young startup, you'd think we'd be over the moon. Surely now is the time to raise a ton of money and build that high performance next generation blockchain platform we've already designed. What on earth are we waiting for?

I'll tell you what. We're waiting to gain a clearer understanding of where blockchains *genuinely* add value in enterprise IT. You see, a large proportion of these incoming projects have **nothing to do with blockchains at all**. Here's how it plays out. Big company hears that blockchains are the next big thing. Big company finds some people internally who are interested in the subject. Big company gives them a budget and tells them to go do something blockchainy. Soon enough they come knocking on our door, waving dollar bills, asking *us* to help *them* think up a use case. Say what now?

As for those who do have a project in mind, what's the problem? In many cases, the project can be implemented perfectly well *using a regular relational database*. You know, big iron behemoths like Oracle (http://www.oracle.com/) and SQL Server (http://www.microsoft.com/en-us/server-cloud/products/sql-server/), or for the more open-minded, MySQL (https://www.mysql.com/) and Postgres (http://www.postgresql.org/). So let me start by setting things straight:

If your requirements are fulfilled by today's relational databases, you'd be insane to use a blockchain.

Why? Because products like Oracle and MySQL have decades of development behind them. They've been deployed on millions of servers running trillions of queries. They contain some of the most thoroughly tested, debugged and optimized code on the planet, processing thousands of transactions per second without breaking a sweat.

And what about blockchains? Well, our product (http://www.multichain.com/) was one of the first to market, and has been available for exactly 5 months, with a few thousand downloads. Actually it's extremely stable, because we built it off Bitcoin Core (https://bitcoin.org/en/download), the software which powers bitcoin. But even so, **this entire product category is still in its diapers**.

So am I saying that blockchains are useless? Absolutely not. But before you embark on that shiny blockchain project, you need to have a very clear idea of **why you are using a blockchain**. There are a bunch of conditions that need to be fulfilled. And if they're not, you should go back to the drawing board. Maybe you can define the project better. Or maybe you can save everyone a load of time and money, because you don't need a blockchain at all.

1. The database

Here's the first rule. Blockchains are a technology for **shared databases**. So you need to start by knowing why you are using a database, by which I mean a structured repository of information. This can be a traditional relational database (https://en.wikipedia.org/wiki/Relational_database), which contains one or more spreadsheet-like tables. Or it can be the trendier NoSQL (https://en.wikipedia.org/wiki/NoSQL) variety, which works more like a file system or dictionary. (On a theoretical level, NoSQL databases are just a subset of relational databases anyway.)

A ledger for financial assets can be naturally expressed as a database table in which each row represents one asset type owned by one particular entity. Each row has three columns containing: (a) the owner's identifier such as an account number, (b) an identifier for the asset type such as "USD" or "AAPL", and (c) the quantity of that asset held by that owner.

Databases are modified via "transactions" which represent a set of changes to the database which must be accepted or rejected as a whole. For example, in the case of an asset ledger, a payment from one user to another is represented by a transaction that deducts the appropriate quantity from one row, and adds it to another.

2. Multiple writers

This one's easy. Blockchains are a technology for **databases with multiple writers**. In other words, there needs to be more than one entity which is generating the transactions that modify the database. Do you know who these writers are?

In most cases the writers will also run "nodes" which hold a copy of the database and relay transactions to other nodes in a peer-to-peer (https://en.wikipedia.org/wiki/Peer-to-peer) fashion. However transactions might also be created by users who are not running a node themselves. Consider for example a payments system which is collectively maintained by a small group of banks but has millions of end users on mobile devices, communicating only with their own bank's systems.

3. Absence of trust

And now for the third rule. If multiple entities are writing to the database, there also needs to be some degree of *mistrust* between those entities. In other words, blockchains are a technology for **databases with multiple non-trusting writers**.

You might think that mistrust only arises between separate organizations, such as the banks trading in a marketplace or the companies involved in a supply chain. But it can also exist *within a single large organization*, for example between departments or the operations in different countries.

What do I specifically mean by mistrust? I mean that one user is not willing to let another modify database entries which it "owns". Similarly, when it comes to reading the database's contents, one user will not accept as gospel the "truth" as reported by another user, because each has different economic or political incentives.

4. Disintermediation

So the problem, as defined so far, is enabling a database with multiple non-trusting writers. And there's already a well-known solution to this problem: **the trusted intermediary**. That is, someone who all the writers trust, even if they don't fully trust each other. Indeed, the world is filled with databases of this nature, such as the ledger of accounts in a bank. Your bank **controls the database** and ensures that every transaction is valid and authorized by the customer whose funds it moves. No matter how politely you ask, your bank will never let you modify their database directly.

Blockchains remove the need for trusted intermediaries by enabling **databases with multiple non-trusting writers to be modified directly**. No central gatekeeper is required to verify transactions and authenticate their source. Instead, the definition of a transaction is extended to include a proof of authorization and a proof of validity. Transactions can therefore be **independently verified and processed by every node** which maintains a copy of the database.

But the question you need to ask is: **Do you want or need this disintermediation?** Given your use case, is there anything wrong with having a central party who maintains an authoritative database and acts as the transaction gatekeeper? Good reasons to prefer a blockchain-based database over a trusted intermediary might include lower costs, faster transactions, automatic reconciliation (https://en.wikipedia.org/wiki/Reconciliation_(accounting)), new regulation or a simple inability to find a suitable intermediary.

5. Transaction interaction

So blockchains make sense for databases that are shared by multiple writers who don't entirely trust each other, and who modify that database directly. But that's still not enough. Blockchains truly shine where there is some **interaction between the transactions** created by these writers.

What do I mean by interaction? In the fullest sense, this means that transactions created by different writers often depend on one other. For example, let's say Alice sends some funds to Bob and then Bob sends some on to Charlie. In this case, Bob's transaction is dependent on Alice's one, and there's no way to verify Bob's transaction without checking Alice's first. Because of this dependency, the transactions naturally belong together in a **single shared database**.

Taking this further, one nice feature of blockchains is that transactions can be created **collaboratively by multiple writers**, without either party exposing themselves to risk. This is what allows delivery versus payment (http://www.multichain.com/blog/2015/09/delivery-versus-payment-blockchain/) settlement to be performed safely over a blockchain, without requiring a trusted intermediary.

A weaker case can also be made for situations where transactions from different writers are cross-correlated with each other, even if they remain independent. One example might be a shared identity database in which multiple entities validate different aspects of consumers' identities. Although each such certification stands alone, the blockchain provides a useful way to bring everything together in a unified way.

6. Set the rules

This isn't really a condition, but rather an inevitable consequence of the previous points. If we have a database modified directly by multiple writers, and those writers don't fully trust each other, then the database must contain embedded rules **restricting the transactions performed**.

These rules are fundamentally different from the constraints (https://en.wikipedia.org/wiki/Relational_database#Constraints) that appear in traditional databases, because they relate to the legitimacy of transformations (http://www.multichain.com/blog/2015/10/private-blockchains-shared-databases/) rather than the state of the database at a particular point in time. Every transaction is checked against these rules by every node in the network, and those that fail are rejected and not relayed on.

Asset ledgers contain a simple example of this type of rule, to prevent transactions creating assets out of thin air. The rule states that the total quantity of each asset in the ledger must be the same before and after every transaction.

7. Pick your validators

So far we've described a distributed database in which transactions can originate in many places, propagate between nodes in a peer-to-peer fashion, and are verified by every node independently. So where does a "blockchain" come in? Well, a blockchain's job is to be the **authoritative final transaction log**, on whose contents all nodes provably agree.

Why do we need this log? First, it enables newly added nodes to calculate the database's contents from scratch, without needing to trust another node. Second, it addresses the possibility that some nodes might miss some transactions, due to system downtime or a communications glitch. Without a transaction log, this would cause one node's database to diverge from that of the others, undermining the goal of a shared database.

Third, it's possible for two transactions to be in conflict, so that only one can be accepted. A classic example is a double spend (https://en.bitcoin.it/wiki/Double-spending) in which the same asset is sent to two different recipients. In a peer-to-peer database with no central authority, nodes might have different opinions regarding which transaction to accept, because there is **no objective right answer**. By requiring transactions to be "confirmed" in a blockchain, we ensure that all nodes converge on the same decision.

Finally, in Ethereum (https://www.ethereum.org/)-style blockchains, the precise *ordering* of transactions plays a crucial role, because every transaction can affect what happens (http://www.multichain.com/blog/2015/11/smart-contracts-slow-blockchains/) in every subsequent one. In this case the blockchain acts to define the authoritative chronology, without which transactions cannot be processed at all.

A blockchain is literally a chain of blocks, in which each block contains a set of transactions that are confirmed as a group. But who is responsible for choosing the transactions that go into each block? In the kind of "private blockchain" which is suitable for enterprise applications, the answer is a closed group of validators ("miners") who digitally sign the blocks they create. This whitelisting is combined with some form of distributed consensus scheme to prevent a minority of validators from seizing control of the chain. For example, MultiChain uses a scheme called mining diversity (http://www.multichain.com/developers/blockchain-parameters/), in which the permitted miners work in a round-robin (https://en.wikipedia.org/wiki/Round-robin) fashion, with some degree of leniency to allow for non-functioning nodes.

No matter which consensus scheme is used, the validating nodes have far less power than the owner of a traditional centralized database. Validators cannot fake transactions or modify the database in violation of its rules. In an asset ledger, that means they cannot spend other people's money, nor change the total quantity of assets represented. Nonetheless there are still two ways in which validators can unduly influence a database's contents:

- **Transaction censorship**. If enough of the validators collude maliciously, they can prevent a particular transaction from being confirmed in the blockchain, leaving it permanently in limbo.
- **Biased conflict resolution**. If two transactions conflict, the validator who creates the next block decides which transaction is confirmed on the blockchain, causing the other to be rejected. The fair choice would be the transaction that was seen first, but validators can choose based on other factors without revealing this.

Because of these problems, when deploying a blockchain-based database, you need to have a clear idea of **who your validators are and why you trust them**, collectively if not alone. Depending on the use case, the validators might be chosen as: (a) one or more nodes controlled by a single organization, (b) a core group of organizations that maintain the chain, or (c) every node on the network.

8. Back your assets

If you've got this far, you may have noticed that I tend to refer to blockchains as shared databases, rather than the more common "shared ledgers". Why? Because as a technology, blockchains can be applied to problems far beyond the tracking of asset ownership. Any database which has multiple non-trusting writers can be implemented over a blockchain, without requiring a central intermediary. Examples include shared calendars, wiki-style collaboration and discussion forums.

Having said that, for now it seems that blockchains are mainly of interest to those who track the movement and exchange of financial assets. I can think of two reasons for this: (a) the finance sector is responding to the (in retrospect, minuscule) threat of cryptocurrencies like bitcoin, and (b) an asset ledger is the most simple and natural example of a shared database with interdependent transactions created by multiple non-trusting entities.

If you do want to use a blockchain as an asset ledger, you need to answer one additional crucial question: What is the nature of the assets being moved around? By this I don't just mean cash or bonds or bills of lading, though of course that's important as well. The question is rather: **Who stands behind the assets represented on the blockchain?** If the database says that I own 10 units of something, who will allow me to claim those 10 units *in the real world*? Who do I sue if I can't convert what's written in the blockchain into traditional physical assets? (See this asset agreement (http://coinspark.org/create-asset/?file=contract&id=sample&template=uk) for an example.)

The answer, of course, will vary by the use case. For monetary assets, one can imagine custodial banks accepting cash in traditional form, and then crediting the accounts of depositors in a blockchain-powered distributed ledger. In trade finance, letters of credit and bills of lading would be backed by the importer's bank and the shipping company respectively. And further in the future, we can imagine a time when the primary issuance (https://en.wikipedia.org/wiki/Primary_market) of corporate bonds takes place directly on a blockchain by the company seeking to raise funds.

Conclusion

As I mentioned in the introduction, if your project does not fulfill **every single one of these conditions**, you should not be using a blockchain. In the absence of any of the first five, you should consider one of: (a) regular file storage, (b) a centralized database, (c) master–slave database replication (https://en.wikipedia.org/wiki/Replication_(computing)#DATABASE), or (d) multiple databases to which users can subscribe (https://en.wikipedia.org/wiki/Publish–subscribe_pattern).

And if you do fulfill the first five, there's still work to do. You need to be able to express the rules of your application in terms of the transactions which a database allows. You need to be confident about who you can trust as validators and how you'll define distributed consensus. And finally, if you're looking at creating a shared ledger, you need to know who will be backing the assets which that ledger represents.

Got all the answers? Congratulations, you have a real blockchain use case. And we'd love to hear from you (http://www.multichain.com/contact-us/).

Please post any comments on LinkedIn (https://www.linkedin.com/pulse/avoiding-pointless-blockchain-project-gideon-greenspan). See also this follow up: Four genuine blockchain use cases (http://www.multichain.com/blog/2016/05/four-genuine-blockchain-use-cases/).

Recent Posts

- Explaining zero knowledge blockchains (http://www.multichain.com/blog/2016/11/explaining-zero-knowledge-blockchains/)
- First MultiChain Partners Announced (http://www.multichain.com/blog/2016/10/first-multichain-partners-announced/)
- Introducing MultiChain Streams (http://www.multichain.com/blog/2016/09/introducing-multichain-streams/)
- Announcing the new MultiChain wallet (http://www.multichain.com/blog/2016/07/announcing-the-new-multichain-wallet/)
- Smart contracts and the DAO implosion (http://www.multichain.com/blog/2016/06/smart-contracts-the-dao-implosion/)
- Four genuine blockchain use cases (http://www.multichain.com/blog/2016/05/four-genuine-blockchain-use-cases/)
- Beware the impossible smart contract (http://www.multichain.com/blog/2016/04/beware-impossible-smart-contract/)
- Blockchains vs centralized databases (http://www.multichain.com/blog/2016/03/blockchains-vs-centralized-databases/)
- Recent Features and 2016 Roadmap (http://www.multichain.com/blog/2016/03/recent-features-2016-roadmap/)
- Moving on from big blockchains (http://www.multichain.com/blog/2016/01/moving-on-from-big-blockchains/)
- Avoiding the pointless blockchain project (http://www.multichain.com/blog/2015/11/avoiding-pointless-blockchain-project/)
- Smart contracts make slow blockchains (http://www.multichain.com/blog/2015/11/smart-contracts-slow-blockchains/)

- Smart contracts: The good, the bad and the lazy (http://www.multichain.com/blog/2015/11/smart-contracts-good-bad-lazy/)
- Private blockchains are more than “just” shared databases (http://www.multichain.com/blog/2015/10/private-blockchains-shared-databases/)
- Delivery versus payment on a blockchain (http://www.multichain.com/blog/2015/09/delivery-versus-payment-blockchain/)
- Ending the bitcoin vs blockchain debate (http://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/)

Recent Comments

- Patrick Lismore on Moving on from big blockchains (http://www.multichain.com/blog/2016/01/moving-on-from-big-blockchains/#comment-79)
- Vitalik Buterin (http://ethereum.org) on Smart contracts: The good, the bad and the lazy (http://www.multichain.com/blog/2015/11/smart-contracts-good-bad-lazy/#comment-60)
- Nexus on Private blockchains are more than “just” shared databases (http://www.multichain.com/blog/2015/10/private-blockchains-shared-databases/#comment-49)
- Gideon Greenspan on Private blockchains are more than “just” shared databases (http://www.multichain.com/blog/2015/10/private-blockchains-shared-databases/#comment-33)
- romanix (http://www.orderbook.info) on Private blockchains are more than “just” shared databases (http://www.multichain.com/blog/2015/10/private-blockchains-shared-databases/#comment-32)

Archives

- November 2016 (http://www.multichain.com/blog/2016/11/)
- October 2016 (http://www.multichain.com/blog/2016/10/)
- September 2016 (http://www.multichain.com/blog/2016/09/)
- July 2016 (http://www.multichain.com/blog/2016/07/)
- June 2016 (http://www.multichain.com/blog/2016/06/)
- May 2016 (http://www.multichain.com/blog/2016/05/)
- April 2016 (http://www.multichain.com/blog/2016/04/)
- March 2016 (http://www.multichain.com/blog/2016/03/)
- January 2016 (http://www.multichain.com/blog/2016/01/)
- November 2015 (http://www.multichain.com/blog/2015/11/)
- October 2015 (http://www.multichain.com/blog/2015/10/)
- September 2015 (http://www.multichain.com/blog/2015/09/)
- July 2015 (http://www.multichain.com/blog/2015/07/)