DRAFT

20180426 Distributed Web

2018 Apr 26 DKh

Preface

With the advent of blockchain technology (especially Ethereum) a new era of decentralized applications was born, however we posit that the current blockchain architecture is of limited value to organizations and businesses that are trying to implement their general purpose solution on a block-chain.

Blockchains provide the following benefits:

- de-centralized distributed system
- able to reach consensus about consistent data state
- create economic incentive for participants

Hype aside, why does market devour anything blockchain so voraciously? The primary reason lies in the interest in decentralization. No single governing body, except for the algorithm itself, has a full control over the chain. The very structure ensures the transparency of operation and scarcity of the resource (for cryptocurrencies), which is important for value-exchange applications. Blockchains are all about value exchange, be it cryptocurrency, parcels of land (or any other property) or tokens representing stake of some future value (think ICO). This is because they represent a chain of ledger blocks, where ledgers are basically time-ordered lists of transactions, and this imposes limits - **blockchains are basically a form of specialized event logs.**

Blockchains are not to be confused with general-purpose databases, as the later are built for:

- Business-specific data structures/data modeling - be it tables, tuples or rich documents
- Ad-hoc data querying usually built around extensive indexing of rich data models
- High throughput rates - both read (querying) and writes
- Storage and performance scalability proportional to number of nodes/servers of the database
- Ability to harvest “Big Data” loads

Are blockchains databases? Well, in a sense they are, but definitely not a general-purpose ones as blockchains:

- Lack custom data structures / data modeling (yes, you can do smart contracts in Ethereum, but that is no enough flexibility for general-purpose business apps)
- Do not provide indexing, therefore lack ad hoc querying/data analytics

- Low throughput rates, the very nature of distributed consensus precludes the write TPS rates comparable to NoSQL or RDBMS solutions (sharding/channeling does help but still is exchange-oriented and not usable for general-purpose applications)
- Blockchains have truly horrible scalability - their throughput and storage properties do NOT increase proportionally with the growth of the network size, e.g. bitcoin storage does not increase at all as all nodes needs to hold all data

What are typical business applications? These are usually data-driven solutions, with eleboarte backend and front end/UI. Business apps concentrate on their domain needs - their business. This affects the way how those apps are designed and developed. The process usually revolves around some form of DDD (Domain-Driven Design), even if it is not called that; which represents business entities in data schemas, then entity object models and eventually processes and UI screens, which are rife with data fields, custom controls, and intricate logical interconnections.

With the addition of server-side scripting on the web, companies quickly turned their inanimate static web sites into live data feeds - this is how we know internet today. Rarely does one find a life-less static website (but for software documentation sites, but even those are auto-generated from GitHub and the like). So, there is always some code on the web server that renders those HTML pages or JSON API responses.

One does not come without another. Today there is no web app without a database, and hardly any database schema without web apps that serve their data to end users. There are cases when databases are not “that important”, for example in online gaming communities the servers mostly connect the players and exchange the game character/scenery positions and other data between players in real-time. Take social networks. They have db servers, web servers that render the UI and APIs, and most importantly all sorts of queues and other middleware which dispatches those “status updates” and other messages between users - those are true n-tiered systems which need to deliver the world-class message to an audience of 100M subscribers when Kardashian goes into XYZ store shopping...

How blockchains tie-into business apps?

The blockchain concept to this day revolves around distributed ledger which is of limited value to general-purpose business tasks because:

1. General businesses produce a lot of data (contrast with smart contract storage requirements)
2. Businesses produce much data rather quickly, for example an ECommerce site tracks all order fulfillment changes as they get applied to the order; a social site generates 100s of thousands of comments every second
3. Not only the transactional throughput must be high, it must be virtually unlimited (e.g. social network)

4. Need database/datastore layer capable of representing sophisticated domain models, their relations and schema changes (contrast: patient practice management system with 20+ entities - how to model on Ethereum)
5. Need general-purpose programming language and tools for business application development - including user-facing professional UIs

Conclusions

The world likes the idea of distributed apps. People want more freedom and transparency, especially in the light of latest events with FaceBook and Google.

Businesses and not-for profit organizations want to have protection from centralized interference - that is why companies look into blockchains hoping to find there a solution for their needs, such as: databases, and server platforms which work together.

What becomes apparent, is that blockchains are really not suitable for being a single technological backbone for general purpose business apps at all, at least not for any form of near real-time OLTP. One could argue that Big Data approaches such as mapreduce may be used to “scan through” a unidirectional ledger of facts, however that is really not applicable to majority of apps.

When someone says “we have built our XYZ system on a blockchain” what they probably mean is that they store some limited set of data such as agreements or transactions in a distributed ledger, however the majority of data is still kept in the classic RDBMS and or NoSQL/NewSQL.

We are not saying that blockchains are bad. We are saying that they should be used for what they have been designed for - accounting of value exchange.

Our Approach

We are suggesting to look at the problem from a bit of different angle. The so much coveted freedom from central dictatorship is attainable with other techniques, some similar in a way to block chains, some different.

We start from the distributed system design as a whole. Everything is distributed: database, web servers, middle tiers etc. This solves one part of the puzzle. Now we could serve XYZ site from all over the world as no single central cybercenter holds the data and the code that presents this data to end users. What remains to be tackled is the 2nd quality of blockchains - their ability to reach a consensus - keep the data in consistent state.

GOS

GlobOS, **GOS**, GlobalOS is a computer software framework (more correctly a software-mode operating system) which promotes creation of global applications. Global applications are defined as software deployed on a planetary (hence the name) scale.

The primary objective of the project is to allow the world to create physically de-centralized applications of the Facebook/Twitter scale, in such a way that no single authority/country/government/group shall have access to sabotage the application, alter its data or disrupt the service in any way.

Simply put, we are building GOS on the ideas of share-nothing, eventually consistent, distributed data stores operating on a Fact Chain (FACT.db) which derives its roots from blockchain and document-oriented database with a built-in application platform which takes its code from the very same Fact.db and provides a distributed web/UI facade, so no single datacenter is vulnerable to political disfavour or terrorist attack.

The approach is based on the so-popular blockchain ideology which decentralizes currency/ledgers, however blockchains fall short of being able to be used as a general-purpose platform (e.g. Bitcoin lacks Turing-complete language) . Etherium made a leap forward with its smart contracts and ability to run general purpose business code to the very limited extent (storing strings as byte[]? Solidity devs easy to find?). For example, the current blockchain approaches are:

- Slow in reaching consensus - incapable of processing multi million trans/sec globally (think Facebook users posting comments - those are transactions too)
- Lack general-purpose high level data modeling, storage, querying capabilities (compare RDBMS/NoSQL/Document databases to Etherium or Ripple)
- Lack general-purpose application platform similar to n-tiered approach (queues, batch jobs, agents , etc all operating on backend data sources)
- Lack decentralized Web architecture (e.g. server side-scripting) - serving dynamic content from DB

So what are we saying? To summarize, simply put, we want to have:

- a “blockchain-insh” (consensus-based conflict resolution) data sources with properties close to “regular” queryable databases; reaching eventual consistency; able to swallow multi-millions of changes world-wide
- Ability to write dynamic websites fed from the aforementioned database, serving them from all over the world, not a single datacenter
- Have a simple high-level language known to millions of developers already (such as javaScript, Java, C#, Python but not Rust or Erlang)

- Allowing them to write any distributed n-tiered software, be it a simple web to multi-tiered service stack
- Store that software on a replicated database itself, so no malefactor can manipulate the system

We have studied systems like free.net for web, Etherium for blockchain and IPFS as a distributed store and came to conclusion that it is hard if not practically impossible to use these approaches for building end-to-end business/data-driven applications, as:

- All distributed web systems lack normal server-side scripting/application programming, so it is virtually impossible to create dynamic web apps (and that is what is needed in 90% of cases!). Had they had some scripting abilities built-in, the operation model would still have been missing the distributed Database/store. Who needs a static web site/bulletin board in 2018?

- Blockchains were originally meant for ledgers - a very narrow applicability in the accounting field. For example, blockchains can not answer the question: “Who has Green eyes and taller than 5’10” - they are simply not built for that, there is no indexing in blockchains and no simple way to “query blockchains” (yes there are some incipient developments in the field but they lack simplicity and common mindset of a typical CRUD)

- IPFS and similar DHT/DFS projects are too of a low level for “an average Joe” to develop on. One can not expect to write a social app that has a friend list or eCommerce app using IPFS (at least today), furthermore thats is not in the scope of IPFS project altogether.

So what we posit with GOS is that, in 2018 the world needs to:

- Have an open-source solution which
- Anyone can install for their “private/under the desk” needs OR
- Anyone can install a GOS node and participate in global de-centralized applications (e.g. FactBook social network operated by the world);
- Create de-centralized applications easily - no need to hire rocket scientists - we want to reuse existing web skills/mindset: JavaScript, templates, JSON, REST, MVC/Actions, Data Model, CRUD, SQL
- Have a general-purpose “normal mind” data store akin to Document/record-based CRUD/Active record pattern with built-in blockchain “magic” - write SQL-like/LINQuesque.
- Serve dynamic Web Content, usually taken from Database as the result of REST/HTML FORM calls and render the response

- Enjoy global services likes distributed queues, actors, worksets, background services, agents (think Azure/AWS) without any coupling to these providers

[TBD Economical of the participant?]

So what is GOS? Instead of building blockchain separately from the distributed web - we decided to first start from building a new kind of a database-oriented distributed consensus system - a FACT CHAIN, which operates on Facts - documents similar to the ones used in MongoDB. The ideology of a document tree used for data objects is very widely accepted (think JSON, Thrift, Protobuf et.al.). Blockchains are based around blocks of records of limited purpose (ledger mostly or just byte blobs). GOSs fact chain does the same per database record identified by a GDID key (2^96 id).[tbd more consensus reaching details, especially REVs]

So once we have a database service that provides consensus for the data “documents”/FACTS stored in it, we can now query it for data, and code - actually this is where from the system takes the applications that it runs. The code is written in CLR .NET-compatible language (C#) and runs natively on Linux, Mac and Windows (we build our stack on .NET Standards 2.0 and execute on .NET Framework 4.7+ or .NET Core 2+)

Users install nodes which host the system. A node has a role, primarily: compute node or data node. Data nodes hold information and require disk, whereas compute nodes require mostly CPU. Regardless of configuration a node requires 8GB + ram for performance.

In a nutshell, GOS is a framework that developers get using Nuget (or copying files). The development is done in any code editor using dotnet cli or fancy IDEs such as Visual Studio (free), VS Code or Rider.
No paid technologies are used - only open source software which is to be free to use for days to come.
The only technological coupling in the system is to CLR (Common Language Runtime) - which was open sourced by Microsoft, and not possible to be subverted in future (there are other runtimes, like Mono).
The implementation of GOS is a clean one - bypassing patented code bases with dubious licenses. The platform has the whole UNISTACK (Unified Stack) of solutions (including the ES6 Component-Based Web UI, webserver + MVC, data mapper, serialization, logging, security) etc…

Fact.db

It is a database service offered to nodes situated in the GOS zones.
It is an eventually-consistent data service, the changes will not be visible instantly to all consumers as the changes need to be agreed-upon (consensus) and propagate around the network.

Every fact is identified by a unique name, for example “Person”.
Facts have effective schema, and can change their schema with time (transparent upgrade). Of course, in the systems of this magnitude one needs to completely exclude notions of “DDL” as applied in SQL RDBMSs (create table, alter table) as GOS has to deal with possibly millions of data nodes storing data for “Person”. This ideology is not new and was used extensively in business apps written around MongoDB which uses the “document” approach.

So think of a fact as a table(or collection), named “Person” per our example. Every fact has a mandatory set of system fields, first and foremost is an “ID” which uniquely identifies any fact when concatenated with the fact name: person://23:3:34563 <- exemplifies a “Person” fact with GDID (23:3:34563). You can read on GDIDs here:
http://agnicore.com/products/agnios/book/global_id_generation.htm
Per-shard querying is permitted as on normal table as the data is indexed and exists locally. Data modeling with Fact.db is simple and will be covered elsewhere, but briefly it is based on a

Dynamic Distributed Web

This is where GOS start to drastically differ from anything available. Distributed web is formed by N nodes running GOS applications. Those nodes run web servers - they run web applications which until today have only been run centrally (think Facebook) in a large datacenters controlled by a single company. Freenet is not the same, as it is a static web. GOS is all for dynamic application web.

In GOS, the apps are written in the similar way to writing regular MVC web sites. Controllers, models, views. The big difference is that the DATA which is much needed in for live apps comes from the Fact chain - so it is the data verified by other nodes.

The applications get installed/deployed from the “metabase” which is a master metadata feeder itself hosted on a fact chain.

Engineering Issues to Address

Metabase Must Run on FactChain

It is important to make Metabase itself run on the fact chain because:

1. The overall system topology is compromised IF the Metabase catalogs are filled with garbage data
2. Only good code must be able to execute, we need to have “verifiable code”

Looks like the SYSTEM metabase would need to run the Metabase Cluster - of which the metabase for application will be served.
Reach consensus using Proof of Merit

Code Security

Suppose some nad code gets deployed to Metabase. How do we ensure CAS (dropped in .NET Core)?
https://docs.microsoft.com/en-us/dotnet/framework/misc/code-access-security-basics

What to do we high RAM requirement?

We compile 64 bit only and use bigmemory, many users have 16gb, even 8gb. Running large footprint nodesd will hamper performance.
Possible solution, use NFX’s memory footprint setting to make IN-MEMORY stores “Tiny/Small”

Q and A

How is GOS different from Cosmos DB?

From Microsoft site https://docs.microsoft.com/en-us/azure/cosmos-db/local-emulator

“The Azure Cosmos DB Emulator provides a local environment that emulates the Azure Cosmos DB service for development purposes. Using the Azure Cosmos DB Emulator, you can develop and test your application locally, without creating an Azure subscription or incurring any costs….”

Cosmos DB is a Microsoft-proprietary product which runs on Azure, furthermore it has very little if anything to do with distributed consensus/fact chain approach.

How is GOS different from Etherium “World Computer”?

People sometimes describe Etherium as a “state machine” on a blockchain. It still remains at this level of abstraction. It is not possible to author true applications on such a low-level.

Etherium runs on EVM - a special purpose virtual machine which is hard to fathom conceptually for some developers.

Also, it lacks the whole general-purpose computational framework, for example - suppose you want to create an agent that cross-checks the records of facts between various parties in a civil dispute. What if you want to verify some facts in 1 day, 2 days, 2 weeks? There is no mechanisms for agents, batch jobs, general purpose REST APis etc..
There is in GOS.

How is this different from IPFS?

IPFS is an interesting project of somewhat similar ilk, however vastly different, for:

a. It is not a general-purpose application platform
b. It is not a CRUD-like database platform with a blockchain nature

How is this different from Akasha Project?

GOS is a library, Akasha is an app built on top of IPFS and Etherium.
In the case of GOS such app would be built on GOS alone, not various different solutions

What languages and tools can one use to develop for GOS?

Any language and compiler compatible with CLR and .NET Standard 2 or above will do.
Examples: C#, VB.net, F# and many more…
You can develop in rich Visual Studio 2017 on Windows or be as minimalistic as vim + dotnet cli on Linux - it is up to you. VS Code, Rider and other .NET tools are welcome since it is all standard lean .NET stuff (without typical bloat).
GOS is only a set of nuget packages

Can you run GOS on MAC?

Yes, since .NET Core runs on a MAC

Is there a performance disparity running GOS between Windows and Linux?
No, there is none, since .NET Core is designed to run natively on Linux. You are probably thinking of Mono 5-6 years ago, when .NET code could not run on Linux as fast as it could on Windows. This has been long gone and fixed by Microsoft since .NET Core. We do not use Mono.

Isn’t .NET/Microsoft a single point of failure in your “free the world” saga?

No, they are not. They have open-sourced the runtime, compilers and toolchain. There is no turning back as we only use basic C# and basic class libraries from MSFT.

If I host a node, and since GOS installs the new software by itself, can I contract a “virus”/bad code?

You absolutely could if you install an application developed by a rogue developer/group, with shady metabase history and no peer-reviewed code. This question is akin to “can i get a virus if I install WInZip” - yes you can if you get a bad code on your system. Only install applications of trusted lineage - the fact chain ensures the consensus on the app deployment cycle.

References

https://www.bigchaindb.com/developers/getstarted/
The problem is, the blockchain as a database is awful, measured by traditional database standards: throughput is just a few transactions per second (tps), latency before a single confirmed write is 10 minutes, and capacity is a few dozen GB. Furthermore, adding nodes causes more problems: with a doubling of nodes, network traffic quadruples with no improvement in throughput, latency, or capacity. Plus, the blockchain essentially has no querying abilities.

https://ipdb.io/

As of February 2nd, 2018, IPDB will be shutting down.

We've thought long and hard about this decision. Our plan to be an internet-scale blockchain database for the world started in mid-2015, and those two and a half years are an eternity in blockchain time. The world has changed, and funding to maintain and operate IPDB while maintaining its core values became an insurmountable struggle.

https://blog.bigchaindb.com/bigchaindb-raises-3-million-series-a-investment-71e8065940df
https://www.zdnet.com/article/blockchains-in-the-database-world-what-for-and-how/

https://medicalchain.com/en/news/
https://monax.io/
https://www.hyperledger.org/
https://flur.ee/
https://www.bigchaindb.com/developers/getstarted/

https://en.bitcoin.it/wiki/Proof_of_burn
https://gochain.io/
https://medium.com/gochain/proof-of-reputation-e37432420712