Bitcloud Immersion

High Level Technical Description

Presentation Goal

- Provide all interested developers an understanding of the Bitcloud core architecture as currently designed
- Overview of the Bitcloud project and software, according to current state
- Give an overview of Bitcloud components, processes, and component/process interaction within key software operations
- Dive into Bitcloud’s distributed DB core: **Nodepool.sql**

Presentation Format/Agenda

•Held via Hangout and recorded for future video distribution

•Business, Project Management, and Technical presenters

ADDITIONAL INFORMATION AT:

http://bitcloudproject.org/

https://github.com/wetube/bitcloud/

Presentation Outline

1. Description of Bitcloud
2. High-level architecture and design
3. Nodepool.sql specifications
4. Questions and answers

Bitcloud Description

Bitcloud is an open source software platform currently under development for:

- Cryptographically supported,
- Community provided,
- Grid-based,
- Distributed,
- Network services (beginning with “cloud storage”)
- That are automatically maintained by random consensus synchronization operations.

Bitcloud Description (cont.)

Bitcloud may utilize, but is not, all of the following:

- Cryptocurrency, including escrow or other surety - but applications can create their own on top of the open source Bitcloud platform;
- Bitcloud has designs for automatic agreement execution functionality specific to procurement and delivery of its network services, but Bitcloud is not itself an automated agreement platform; and/or
- A Tor router, but Bitcloud can run on top of Tor if communication quality is not a strong requirement.

Bitcloud Description (cont.)

Bitcloud, however, could be leveraged for:

- Distributed Applications (Dapps) may be built on top of or integrated with the Bitcloud software platform.
  - As a result of Bitcloud's core distributed, relational, mutable database, there are few limits on the kinds of applications that can leverage the Bitcloud platform.
- Cryptocurrencies, cryptoassets, stock applicaitons or other payment mechanisms may be built on top of Bitcloud.
  - Current cryptocurrencies like Bitcoin or Dogecoin can be integrated into Bitcloud and used without modification in the automated escrow/agreement operations.

Bitcloud Description (cont.)

Additional possibilities:

- Automated agreement or “smart contract” platforms (e.g., Ethereum, OpenTransactions) may integrate with Bitcloud’s core functionality and/or services.
- Programmable or “Smart” contracts aren't the basis for the Bitcloud platform, but Bitcloud provides certain predefined contracts (supported by automated escrow functions) and capability for DAs to create new automated agreement types encoded on top of Bitcloud.

Dapps may use Bitcloud as the core of an application providing any of the above.

How Bitcloud Works: Storage

Per the white paper:

- Nodes: compose the Bitcloud Grid systems; have the roles of Storage, Gateway, and/or Owner.
- Publishers and Grids may enter into agreement for network services (i.e., storage), validated by random tests and audits (with automated consensus) whenever applicable.
- Users approved by publishers push content/data to storage grids via the Grid's Gateway Nodes with encrypted communication.
- The Gateway(s) sign and slice the file, and push each slice to its Grid's Storage Nodes, which replicates slices with N-to-K redundancy such that 60% (for example) of the Nodes storing a file may go offline with the file still able to be retrieved.

How Bitcloud Works: Storage

- Publishers certify registered users as the CA over their data, or generate unregistered users by providing them a pair of keys, to authorize and enable automatic download and decryption of the files stored on a Bitcloud grid. Those users are always attached to the Publisher's portfolio, so origin and intention of users may easily be determined.
- The Bitcloud User Interface allows automated transaction requests for Publisher authorized file(s) from the Grid Gateway, which is verified by the Grid's Owner(s).
- The transmitting Gateway Node(s) send requests to closest storage nodes containing the required slices, receives slices in parallel, reassembles, and decrypts the entire file.
- Gateway sends the file to the user via encrypted communication in a manner that the user's client can receive (e.g., HTTPS), and the request's closeout routines are run.

Sync: Data Event Routines

As part of sync maintenance operations, checks occur:

- At the event Storage Nodes attempts to provide services within a grid, the Grid's Owner and other nodes validate the requesting Storage Node has its Nodepool globally synced tables complete and valid.
- At the event Nodes or Publishers begin interaction with any other network participant, both party checks each other's certificate to ensure it is valid and comes from a mined source.
- At the time of agreement solicitation by the Grid Owner to ensure its grid has the capacity and service level requested.
- At the event of a user request, the delivering Gateway checks the key against a publisher controlled table to verify permissions.
- At the completion of a Storage Node's delivery of a requested file slice, by the Gateway to ensure data validity and to validate node quality of service.

Sync: Periodic Check Routines

In addition to event-based checks, the automatic maintenance periods occur with the following:

- A random-yet-deterministic number generation process results in a node getting assigned 16 connections for synchronization activities.
- The nodes run consensus routines to arrive at valid operational DB state and update all node tables accordingly.
- Non-complying nodes are reported by the consensus majority to Grid Owners, Publishers, or other applicable agents for additional tests/audits, and or eventual removal from the network.

The above is a simplification of the basic periodic check cycle, but there are more steps and Grid Owners, Dapps, and/or Publishers may require custom periodic sync routines to ensure data validity and network operational health.

Sync: Additional Maintenance

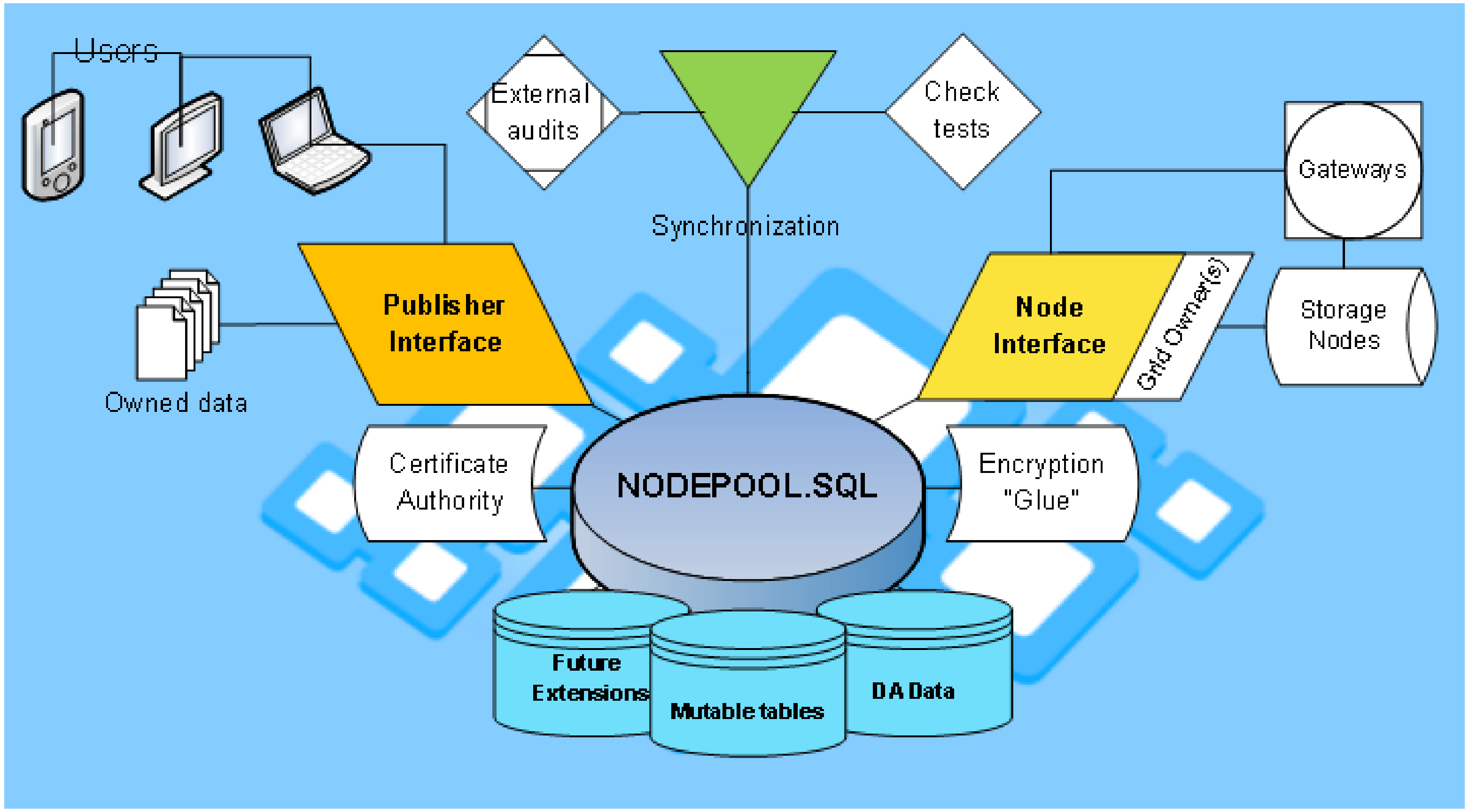
Additional checks or random-neutral-party audits may occur:

- As a result of internal grid maintenance.
- As encoded in the automated agreement.
- As required by the DA code.

Other notes:

- Every node and publisher has to mine its own certificate, and functions as its own Certificate Authority. The mining requires solving computing intensive problems that will take modern high-end computers 1-2 days to complete.
- **Many key processes are left yet unmentioned to keep the description high-level.**

High-level Design: Components



The above diagram illustrates key Bitcloud components and interactions, but not all.

High-level Design: Operations

- Bitcloud is a platform, and not in itself an application. Bitcloud may be used to build any number of applications.
- Each participant may choose how they participate in the Bitcloud network. Participants may be Nodes or Publishers, they may choose to run instances of both roles.
- A node instance may only be one type of role (e.g., gateway, storage, etc.) at a time, but multiple node instances may be run on a single machine at the same time (limited by machine capabilities).
- A node or publisher may only run one Bitcloud-based Dapp per Bitcloud instance.
- Nodes may participate in multiple grids, one per instance, each with different service capacity and levels (as configured).
- Random assignment is a cornerstone to Bitcloud health.

Other Bitcloud Design Elements

Bitcloud has been designed to be built with generic tools:

- Generic SQL as the core platform language. Future Bitcloud iterations may incorporate or expand to allow for Postgre or DB types.
- The first version will be programmed using generic C.
- Heavily deployed SQLite DB is used to implement the generic SQL core.
- The design requires distributed CA infrastructure, but architecture remains cryptographically agnostic. (May the best solution protect!)
- Standard communication library is used – communication agnostic.

Although much of the team has a functional programming preferences, the above generic toolset was selected to attract as many devs as possible.

Future versions must strictly enforce consistency rules (e.g., sync, cert mining,audit, etc.), but may use other tools for performance and feature improvements (e.g., Lisp and/or other languages, Postgre SQL, etc.).

Bitcloud Immersion

Core Architecture: Nodepool

Nodepool.sql Globally Synced Tables

nodes	
•id	Blob(20)
·public_key	Blob(32)
·signature	Blob(80)
·proof_of_creation	Integer
·creation_date	Integer
·address	Text
°storage_capacity	Integer
°bandwidth_capacity	Integer

grids	
•id	Blob(20)
·signature	Blob(80)
°storage_capacity	Integer
°bandwidth_capacity	Integer
°service_reputation	Integer
°availability	Integer

table_rules
·elements related to sync

gateways
°elements to regulate
°and configure gateways

publishers	
•public key	
°address	Text
·creation_date	date
°proof_of_creation	
·public_metadata	Boolean
·public_files	boolean
·trust_all_users	Boolean

publisher_trusts
·elements related to
·publisher associations

users	
•public key	
·publisher	
·publisher_signature	
°address	Text
°nick	Text
·fixed_address	Boolean
°reovcation_date	Date
·storage_quota	Integer
·bandwidth_quota	Integer
·files_quota	Integer
·folder_quota	Integer
·root_folder	

user_requests
°immutable table for user
°storage requests

publisher_requests
°immutable table for
°publisher storage requests

files
·elements related to
°stored files

pubisher_grid_contracts	
•id	Blob(16)
·publisher	Blob
·grid	Text
·pubisher_sig	Text
·grid_sig	Text
·min_storage	Integer
·min_bandwidth	Integer
·start_date	Date
·end_date	Date
°availability	Integer
°average_latency	Integer
·coin	Text(4)

Nodepool.sql Internal and Private Tables

grid_node_contracts	
•id	Blob(16)
•grid	Blob
•mode	Text
•grid_sig	Text
•node_sig	Text
•min_storage	Integer
•min_bandwidth	Integer
•start_date	Date
•working_time	Integer
◦availability	Integer
◦bandwidth_block_size	
◦price_per_block	
•coin	Text(4)

CAs	
•public key	
•private_key	
◦proof_of_generation	
•ssl_extra	text

logs
•logging data

config
•configuration data

The Nodepool design is still in development, but it is mutable and extensible so it may always be in development.

Nodes Table

- Synced globally
- To insert a record, the inserting node must have performed a CPU/memory-intensive mining operation as the process to generate valid keys/certificate.
- Every X days, the node must renew its registration in the database by updating the record, otherwise it is removed and have to mine again. (This process does not require additional CPU/memory-intensive mining.)

Main attributes are:

- The entire record is signed by the node with its private key, so no other nodes can modify its data. At the time of sync, unsigned rows are rejected and the node trying to cheat is audited. This applies to many tables in the nodepool too.
- Each node has an ID, similar to Kademlia ID, with the purpose of uniquely identifying the node in the net and its state “proximity” to other nodes.
- Nodes can choose the networking protocol to use - for now it can be IP (with or without DNS), Tor, or both. In the future, meshnets will be supported too.
- The node states its storage and bandwidth capacity, which results in a test/audit flag if they cannot support a request within its stated capacity.

Grids Table

- It is a collection of nodes with a shared economical or social interest that choose to operate together and provide network services (e.g., storage).

Attributes:

- Each table record must have at least one owner, as validated by cryptographic signature process.
- Grids table records contain grid configuration and disposition information.
- Grids make contracts with publisher and storage/gateways nodes, as stored in the related "publisher_grid_contracts" internal table.
- A grid is responsible for choosing its gateways from the pool of its grid nodes with signed records in the related "gateways" globally synced table.
- Grid owners are responsible to assign secondary owner(s), in case the primary owner node goes offline.

Gateways Table

- Gateway services are solicited within the grid using the “grid_owner_requests” internal table, and contracted between Grid Owner and Gateway Grid using the “grid_node_contracts” internal table.
- Gateways transform the data from Bitcloud storage nodes into something receivable by the user, like an HTTP or FTP transmission.
- Gateways can be storage nodes at the same time, but those records are contained on a different table. However, if they don't store the content, they can be considered routers and cannot be liable by current western laws.
- Gateways are in possession of the keys to decrypt slices coming from the storage nodes.
- They slice the data files and distribute slices across its grid's storage nodes.
- Gateways perform various checks (e.g., node operation, file validity, etc.) as part of their general operation and grant permission to users based on validated publisher certification.

Publishers

Many useful and needed tasks include but are not limited to:

- They can be considered analogous to the “webmasters” of Bitcloud.
- Curate data and/or content uploaded by users.
- Any registered user can upgrade to publisher at any moment (i.e., after they mine the authority).
- They contract network services (i.e., storage) with grids under the conditions of related distributed applications.
- They grant service (i.e., storage and/or data) access to users.
- Association of publishers are possible, making it possible for one publisher to grant access to another publisher's users.

Users

- Registered users: they autonomously generate a pair of keys (i.e., “mining the authority”) and request approval from any desired publisher.
- A user may be registered in many publishers at the same time.
- On each publisher's rolls, the user may have a different nickname but his/her keys remain the same.
- User interface should allow the use of multiple identities/key pairs.
- Unregistered users: usually they come from browsers and their pair of keys are provided by a publisher. Only recommended for users wanting to view only, like people using embedded content in web pages without even noticing they are using Bitcloud.

File storage

- File storage is always encrypted and sliced such that storage nodes cannot know what local they are storing using any available inspection technique/tool.
- Only gateways and grid owners know the data composition on storage nodes, and content remains unknown.
- Data contents may be accessible to gateways and grid owners only in the case that the user/publisher didn't encrypt the data in advance of transmitting it to the grid.
- Storage is sliced and distributed redundantly using a k-of-n redundancy so that, if most of the storage nodes storing a single file go off line, the data is still retrievable.
- Grid owners can decide upon it's k and n redundancy (and may offer different levels via different contract vehicles). For example k=7 and n=10 means that 70% of the nodes must go off line before data is not retrievable.
- Files and folders may be assigned to Users, and curated by Publishers.
- Files and folders have permissions in a similar way to a filesystem, with write/read access and quota functionality available for users and publishers.

Database for Dapps

- Bitcloud intentionally allows use of its relational database (Nodepool.sql) by Distributed Applications build on top of Bitcloud software.
- It is relational, accepting normal SQL queries.
- It is mutable, so Dapps can update or remove content from Dapp controlled tables. Globally synced tables are not available for modification by Dapps.
- Garbage collection may optionally run according to Dapp specified rules.
- Dapps may be dynamically added to a running node at any time, or optionally as static extension at compile time.
- Different Dapps can share the same network, so content is not fragmented and may be universally available (according to permissions).
- Dapps can point to data of other Dapps if they set access access to do so. For example, a social Dapp may embed a video from a media Dapp that allows it.
- Each Dapp provides its own search engine implementation, if required. Bitcloud does not provide any as part of the platform.

Generic API

- Dapps can be constructed on top of Bitcloud with a simple C language interface.
- Interface consists of generic functions like “insert”, “delete”, “update”, “check”, that the programmer may use, or otherwise use default Bitcloud code (i.e., no interface).
- Languages other than C (e.g., Python, Java, Lisp, etc.) may be used for the interface with FFI mechanisms. Bitcloud developers will port the API to those languages.
- Dapps are loaded dynamically, or statically built, in the form of modules similar in fashion to Apache HTTP server modules.
- Dapps can be distributed in a common repository Dapp that the Bitcloud team will generate after initial Bitcloud release. Each Dapp package must be signed, and there may be dependences in a similar fashion to Debian APT repositories.

Bitcloud Immersion

FINIS

(Thank you.)