A Proposal of Change of the components for the Stacks Decentralized Application Development Architecture

Phillip Roe-Smithson, @PhillipRoeS

Paradigma CrossCheck Dapp @checkParadigma

Background

This is a high-level description of the proposal that Stacks (https://stacks.co) had built as a decentralized application development framework. Since this study required use case of a decentralized application built based on that proposal, the application developed branded as Paradigma CrossCheck (https://crosscheck.paradigma.global) was chosen, because we know it better.

Blockchains enable new types of computer programs: (a) smart contracts that can be published on a blockchain to execute in a trustless manner while anyone can verify their outputs, and (b) decentralized apps that are user-owned and avoid centralized servers.

The Stacks vision is that decentralized apps (Dapps) and varied use cases will be built on Bitcoin, the strongest and most widely used blockchain network, instead of disconnected networks.

Apps built on Stacks inherit all of Bitcoin's powers. They run their logic on the blockchain with smart contracts, are controlled by code instead of companies, and are accessible to anyone. This enables decentralized apps to do things that regular apps can't.

Stacks proposes that the different components of a decentralized app are referenced using a Name Domain Service registered in the Stacks Blockchain. For i.e., each user has its private personal ID and storage location component registered in the Stacks Blockchain.

All Stacks transactions settle on the Bitcoin blockchain, making apps and transactions as secure as a Bitcoin transaction.

When apps are built with Stacks, they can authenticate users, sign transactions and store data with the Stacks blockchain, as well as prompting users to sign and broadcast transactions to the Stacks blockchain.

Stacks proposes that by using this decentralized framework, the apps will have the following benefits:

Unstoppable: Once deployed on Stacks and settled on Bitcoin, code can't be taken down. This is essential for critical internet infrastructure.

Data ownership: Decentralized apps don't store user data on centralized servers. Instead, data is owned by the user and can be taken from app to app. The Data storage layer (Gaia) allows Dapps to save and retrieve this user data.

Open to anyone: Anyone with an internet connection can use or build Stacks apps. You don't need anyone's permission and you can't be blocked from Stacks.

Modular: Decentralized apps are open and connected by default. This enables developers to build on top of each other's apps and prevents users from getting locked in.

An incentive is created to develop new business models. Tokens enable developers to monetize open-source protocols, incentivize contribution and come up with business models that weren't possible before.

Paradigma (https://www.paradigma.global) evaluated and chose the Stacks Decentralized Development Framework for its investment in the development of a decentralized application, naming it as Paradigma CrossCheck.

Paradigma CrossCheck is a network of distributed software nodes that facilitates business communications among them, in order to do business between people, companies or organizations, protecting their privacy and data security while providing and ensuring payments and deliverables from and to any place in the world.

This solution was built using the Stacks Decentralized Development Framework. Each decentralized node is managed by a user that has created a Stacks Id. Each user manages a node in a decentralized way that establishes relations of business communications between other nodes, to send and/or receive different types of business communication messages or contents among the different nodes that are involved in a business agreement. Those business communications are integrated into one or more business agreements facilitating the accessibility of the information involved, the participants, technical specs, quotations, payment agreements, authorization or signing an agreement, the follow up process for delivery, and digital transport of deliverables.

Each user involved in a business agreement can share a private document file, image files, media files, drawings, and other types of specifications. The chat feature helps users to clarify aspects of the business agreement. When parties agree on the terms and forms of payment, they digitally sign the business agreement and produce the authorization of the forms of payments established. The payment form uses the Stacks Blockchain and its associated Clarity SmartContracts.

The business agreements are stored using GAIA and the storage repositories that are addressed by the Stacks Blockchain. The user that initiated the business agreement owns the data and the users involved in the business agreement can access the data, add complement files, and replicate when it is signed.

When the business agreement is being executed, the business agreement is used to keep track of the delivery process and the supplier can send its deliverables in a certified way.

Each business agreement by itself is independent of the rest so it is private, only the allowed users can “understand” what it is written and can interact.

Objectives

Initial objectives of the Study

Study and define the way to add a messaging component to the actual Stacks Dapp architecture, as some decentralized applications require interaction (exchange of information) between other users of the same application or to other applications. Similar to what has been discussed as collections.

Conclusions of the initial objectives of the Study

After starting the Study, we realized that this project needed a broader scope than the messaging aspect of an application. So, we decided to extend the objectives of the study starting from what the Distributed Database Management Systems are (considered the same as decentralized applications) and its considered components and functionalities. We feel that the results will more widely benefit the Stacks Community.

Extended objectives of the Study

- To study and define a prototype an evaluation framework of a Distributed Database Management System.
- Use this prototype definition to evaluate the use case of the decentralized application Paradigma

CrossCheck that was built using the Stacks proposed decentralized development framework.
- Propose changes to the Stacks proposed decentralized system development framework.

Benefits

It will serve all Decentralize Application developers that need interaction between their Dapps or with other Dapps.

Work plan

A. Study of Principles of Distributed Database Management Systems
B. Define a prototype of an Evaluation Framework of Decentralized Database Management Systems
C. Application of the prototype of Evaluation Framework to the use case Dapp Paradigma CrossCheck
D. Proposal of Component Design for Decentralized Database Management System Architecture for Stacks
E. Assessment of Potential Components for the Stacks Decentralized Application Development Framework
F. Proposal of a plan of action to continue or not with the project.

Contents

A. Study of Principles of a Distributed Database Management Systems

This is a compilation from Principles of Distributed Database System M. Tamer Özsu, Patrick Valduriez,, Springer Fourth Edition 2020 (https://link.springer.com/book/10.1007/978-3-030-26253-2) 800 pages approximately.

For a more comprehensive and deep understanding of the Principles of Distributed Database Management Systems it is highly recommended to acquire the original publication referenced above.

The purpose of this compilation: Serve as a conceptual framework to evaluate how a decentralized application is using these principles.

1. What is a Distributed Database System?

A definition of a distributed database as a collection of multiple, logically interrelated databases located as nodes of a distributed system. A distributed database management system (distributed DBMS) is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users.
A distributed DBMS is logically integrated, but physically distributed.

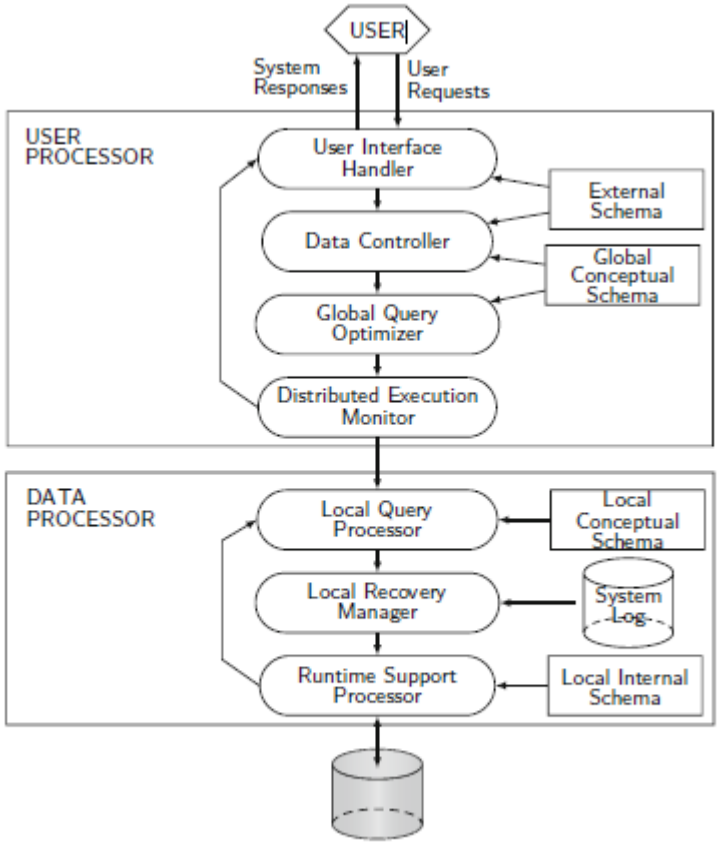


Fig. 1.11 Components of a distributed DBMS

2. Distributed and Parallel Design

The main reasons and objectives for fragmentation in distributed versus parallel DBMSs are slightly different. In the case of the former, the main reason is data locality. To the extent possible, we would like queries to access data at a single site in order to avoid costly remote data access. A second major reason is that fragmentation enables a number of queries to execute concurrently (through interquery parallelism). The fragmentation of relations also results in the parallel execution of a single query by dividing it into a set of subqueries that operate on fragments, which is referred to as intraquery parallelism. Therefore, in distributed DBMSs, fragmentation can potentially reduce costly remote data access and increase inter and intraquery parallelism.

In parallel DBMSs, data localization is not that much of a concern since the communication cost among nodes is much less than in geo-distributed DBMSs.

What is much more of a concern is load balancing as we want each node in the system to be doing more or less the same amount of work. Otherwise, there is the danger of the entire system thrashing since one or a few nodes end up doing a majority of the work, while many nodes remain idle. This also increases the latency of queries and transactions since they have to wait for these overloaded nodes to finish.

1. Distributed Data Control

An important requirement of a DBMS is the ability to support data control, i.e., controlling how data is accessed using a high-level language. Data control typically includes view management, access control, and semantic integrity control. Informally, these functions must ensure that authorized users perform correct operations on the database, thus contributing to the maintenance of database integrity.

2. Distributed Query Processing

By hiding the low-level details about the physical organization of the data, relational database languages allow the expression of complex queries in a concise and simple manner. In particular, to construct the answer to the query, the user does not precisely specify the procedure to follow; this procedure is actually devised by a module, called a *query processor*. This relieves the user from query optimization, a time-consuming task that is best handled by the query processor, since it can exploit a large amount of useful information about the data.

3. Distributed Transaction Processing

The concept of a transaction is used in database systems as a basic unit of consistent and reliable computing. Thus, queries are executed as transactions once their execution strategies are determined and they are translated into primitive database operations. Transactions ensure that database consistency and durability are maintained when concurrent access occurs to the same data item (with at least one of these being an update) and when failures occur.

4. Data Replication

As it has been presented, distributed databases are typically replicated. The purposes of replication are multiple:

1. System availability. As shown, distributed DBMSs may remove single points of failure by replicating data, so that data items are accessible from multiple sites. Consequently, even when some sites are down, data maybe accessible from other sites.

2. Performance. As we have seen previously, one of the major contributors to response time is the communication overhead. Replication enables us to locate the data closer to their access points, thereby localizing most of the access that contributes to a reduction in response time.

3. Scalability. As systems grow geographically and in terms of the number of sites (consequently, in terms of the number of access requests), replication allows for a way to support this growth with acceptable response times.

4. Application requirements. Finally, replication may be dictated by the applications, which may wish to maintain multiple data copies as part of their operational specifications.

Although data replication has clear benefits, it poses the considerable challenge of keeping different copies synchronized.

5. Database Integration—Multidatabase Systems

It is focused on distributed databases that are designed in a bottom-up fashion—referred as multidatabase systems. In this case, a number of databases already exist, and the design task involves integrating them into one database. The starting point of bottom-up design is the set of individual local conceptual schemas (LCSs). The process consists of integrating local databases with their (local) schemas into a global database and generating a global conceptual schema (GCS) (also called the *mediated schema*).

Querying over a multidatabase system is more complicated in that applications and users can either query using the GCS (or views defined on it) or through the LCSs since each existing local database may already have applications running on it.

Database integration, and the related problem of querying multidatabases, is only one part of the more general *interoperability* problem, which includes non-database data sources and interoperability at the application level in addition to the database level.

6. Parallel Database Systems

Many data-intensive applications require support for very large databases (e.g., hundreds of terabytes or exabytes). Supporting very large databases efficiently for either OLTP or OLAP can be addressed by combining parallel computing and distributed database management.

A parallel computer, or multiprocessor, is a form of distributed system made of a number of nodes (processors, memories, and disks) connected by a very fast network within one or more cabinets in the same room. There are two kinds of multiprocessors depending on how these nodes are coupled: tightly coupled and loosely coupled. Tightly coupled multiprocessors contain multiple processors that are connected at the bus level with a shared-memory.

7. Peer-to-Peer Data Management

By distributing data storage and processing across autonomous peers in the network, P2P systems can scale without the need for powerful servers. P2P has also been successfully used to scale data management in the cloud, e.g., DynamoDB key-value store. However, these applications remain limited in terms of database functionality. Advanced P2P applications such as collaborative consumption (e.g., car sharing) must deal with semantically rich data (e.g., XML or RDF documents, relational tables, etc.). Supporting such applications requires significant revisiting of distributed database techniques (schema management, access control, query processing, transaction management, consistency management, reliability, and replication).

When considering data management, the main requirements of a P2P data management system are autonomy, query expressiveness, efficiency, quality of service, and fault-tolerance. Depending on the P2P

network architecture (unstructured, structured DHT, or superpeer), these requirements can be achieved to varying degrees.

However, more recently with blockchain, there has been much more work on update management, replication, transactions, and access control, yet over relatively simple data. P2P techniques have also received some attention to help scaling up data management in the context of Grid Computing or to help protecting data privacy in the context of information retrieval or data analytics.

8. Big Data Processing

The past decade has seen an explosion of "data-intensive" or "data-centric" applications where the analysis of large volumes of heterogeneous data is the basis of solving problems. These are commonly known as *big data applications* and special systems have been investigated to support the management and processing of this data—commonly referred to as *big data processing systems*. These applications arise in many domains, from health sciences to social media to environmental studies and many others. Big data is a major aspect of data science, which combines various disciplines such as data management, data analysis and statistics, machine learning, and others to produce new knowledge from data. The more the data, the better the results of data science can be with the attendant challenges in managing and processing these data.

9. NoSQL, NewSQL, and Polystores

Many different data management solutions have been proposed, specialized for different kinds of data and tasks, and able to perform orders of magnitude better than traditional relational DBMSs. Examples of new data management technologies include distributed file systems and parallel data processing frameworks for big data.

An important kind of new data management technology is NoSQL, meaning "Not Only SQL" to contrast with the "one size fits all" approach of relational DBMS. NoSQL systems are specialized data stores that address the requirements of web and cloud data management.

10. Web Data Management

The World Wide Web ("WWW" or "web" for short) has become a major repository of data and documents. Although measurements differ and change, the web has grown at a phenomenal rate. Besides its size, the web is very dynamic and changes rapidly. For all practical purposes, the web represents a very large, dynamic, and distributed data store and there are the obvious distributed data management issues in accessing web data.

The web, in its present form, can be viewed as two distinct yet related components.

The first of these components is what is known as the *publicly indexable web* (PIW) that is composed of all static (and cross-linked) web pages that exist on web servers. These can be easily searched and indexed. The other component, which is known as the *deep web* (or the *hidden web*), is composed of a huge number of databases that encapsulate the data, hiding it from the outside world.

The difference between the PIW and the hidden web is basically in the way they are handled for searching and/or querying. Searching the PIW depends mainly on crawling its pages using the link structure between them, indexing the crawled pages, and then searching the indexed data.

B. Define a prototype of an Evaluation Framework of Decentralized Database Management Systems

In order to have a conceptual framework to evaluate how a decentralized application has been built, some key questions were extracted from the Principles of Distributed Database Management Systems presented before in A.

These questions can give us an idea how a decentralized application is applying the different attributes for each principle.

In order to facilitate the evaluation of a decentralized application, a form with questions and attributes was prepared and it is presented below. Probably, this evaluation form will be a start to help understanding the different decentralized applications that will be created in the near future by the information technology industry.

Following the questions forms, the compiled answers of the evaluation framework of the decentralized application use case Paradigma CrossCheck is presented.

Distributed Database System (DBSM) Evaluation Form

Produced by Phillip Roe-Smithson @philliproes 2021-02-15

DBSM Principles	Question?	Option	Option
1. Distributed Databases System			
Is it a Distributed Database System?	Yes	No	
A. Data delivery alternatives	Delivery Modes	Frequency	
From servers to client sites	Push only /Pull Only / Hybrid	Periodic/ conditional/ Adhoc	
Between multiple servers or nodes	Push only /Pull Only / Hybrid	Periodic/ conditional/ Adhoc	
B. Distributed Database Design	Partioned/ Replicated		
2. Distribute and Paralell Design			
A. Data Fragmentation	Horizontally / Vertically/ Hybrid		
B. Allocation	Partition database/ Fully replicated/ Partially replicated		
C. Combined approaches	Combine the fragmentation and allocation steps in such a way that the data partitioning algorithm also dictates allocation, or the allocation algorithm dictates how the data is partitioned?		
D. Adaptive approaches	Any measurements for?		
	1. How to detect workload changes that require changes in the distribution design?	Yes, How?	No
	2. How to determine which data items are going to be affected in the design?	Yes, How?	No
	3. How to perform the changes in an efficient manner?	Yes, How?	No
E. Data Directory	Any Global Conceptual Schema (GCS)?		
	Any Local Conceptual Schema (LCS)?		
	Integrated by means of global DBMS functions?		
	Single or multiple copies?		

3. Distributed Data Control	
A. Ability to support data control	Is data accessed using a high-level language?
	Can control …
	View management?
B. Access Control	Insert?
	Delete?
	Update?
	Count?
C. Discretionary Access Control (DAC)	Subject, operation, object?
D. Mandatory Access Control	Security levels?
E. Distributed Access Control	Remote node user authentication?
	Access rules?
	Handling views and user groups?
	Enforcing Mandatory Access Control (MAC)?
	Authentication information maintainced at central site?
	Authentication information replicated all sites?
	All sites identify and authenticate themselves. Site password?
F. Semantic Integrity Control	Rules of integrity of the Global and local data model?
4. Distributed Query Processing	Any query processing in distributed environment?

5. Distributed Transaction Processing	
	ACID (Atomicity, Consistency, Isolation, Durability) transactions in system?
A. Distributed Concurrency Control	Serializable?
B. Locking-Based Algorithms	Locking?
	Centralized Deadlock Detection?
	Distributed Deadlock Detection?
C. Timestamp-Based Algorithms	Timestamp-based concurrency control algorithms?
D. Multiversion Concurrency Control	Maintain the versions of data items?
E. Optimistic Algorithms	Follows phases: read, execute, write, validated and commit?
F. Distributed Concurrency Control using Snapshot Isolation	Multiversioning approach, allowing transactions to read the appropriate snapshot?
G. Distributed DBMS Reliability	The sender of a message will set a timer and wait until the end of a timeout period when it will decide that the message has not been delivered?
G.1 Two-Phase Commit Protocol (2PC)	Uses 2PC?
G.2 Site Failures	Recovery protocols?
G.3 Network Partitioning	Viable to occur?
6. Data Replication	
Database design	Transactions that access replicated data items have to executed at multiple sites?
Database consistency	Strong consistency criteria?
	Weak consistency criteria?
Update propagation	Eager techniques?
	Lazy techniques?
Degree of replication transparency	Full replication transparency?
	Limited replication transparency?

7. Database Integration - Multidatabase Systems	
	Are the databases design top-down or bottom-up?
	Is there a need to integrate them as one database?
	Any formal individual local conceptual schemas (LCS)?
	Any formal global conceptual schemas (GCS) integrating local LCS?
A. Database Integration	Integration physical or logical?
	Using ETL(extract-transform-load)?
B. Multidatabase Query Processing	
	Does it have Multi Database System (MDBS) layer to communicate with component DBMS?
	New requirements for query processing technologies?
8. Paralell Database Systems	
	Is the the app data-intensive requiring support for very large databases?
	If so, does it use any paralell databases?
9. Peer-to-Peer Data Management	
	Does the app use Stacks ID?
	Does the app use the GAIA Storage for data management?
	Does the app use Clarity Smartcontracts?
10. Big Data Processing	
	Is considered as big data application (volumen, variety, velocity, veracity)?
	Does it use big data processing systems?

11. NoSQL, NewSQL, and Polystores	
	Does the app require the use of Not Only SQL systems?
	Considers the use of Polystores?
12. Web Data Management	
	Is the app a repository of data and documents accessed as web?
	Can be considered as publicly indexable web (PIW)?
	Can be considered as deep web (or the hidden web)?
	XML, or RBF?

C. Application of the prototype of an Evaluation Framework to the use case Dapp Paradigma CrossCheck

Distributed Database System **(DBSM) Evaluation Form 15/02/2021**

Case of Study:Paradigma CrossCheck

DBSM Principles	Answer	Answer Description	Desired to have
1. Distributed Databases System			
Is it a Distributed Database System?	Yes		
A. Data delivery alternatives			
From servers to client sites	Pull Only	Conditional, the users have to enter the application to get updated	Push notifications of inserts/ updates to other nodes
Between multiple servers or nodes	Pull Only	Conditional, the users have to enter the application to get updated	Push notifications of inserts/ updates to other nodes
B. Distributed Database Design	Partioned	Each node manages its own data.	Some data must be replicated.
2. Distribute and Paralell Design			
A. Data Fragmentation	Horizontally		
B. Allocation	Partition Database		It should be partially replicated
C. Combined approaches	Data partitioning algorithm also dictates allocation.		
D. Adaptive approaches			
	No detection of workload changes		Not sure
	No data items determination to be affected by design.		Not sure
	No strategies to perform changes in an efficient manner.		Not sure
E. Data Directory	No, Global Conceptual Schema		Yes
	No, Local Conceptual Schema		Yes
	Not integrated to global DBMS functions		Yes
	Single copies		Multiple copies

3. Distributed Data Control		
A. Ability to support data control	No data access using high-level language. Using json structures.	Desirable
	View management Partially, readonly and encription. Physical accesibility.	Ideally, separate the physical accessibility with views
B. Access Control	Only node can insert	
	Only node can delete	
	Only node can update	
	It can be counted, as records, not as data	
C. Discretionary Access Control (DAC)	No	Yes
D. Mandatory Access Control	No security levels	Yes
E. Distributed Access Control	No remote node user authentication.	Yes
	No access rules	Yes
	No handling views and user groups.	Yes
	No enforcing Mandatory Access Control	Yes
	Not centralized Auth Decentralized using Stacks ID	
	Auth info replicated on Stacks Blockchain nodes	
	No sites identification and sel autheintcation. No site password	Yes
F. Semantic Integrity Control	No rules of integrity of the local and Global model	Yes, basic rules
4. Distributed Query Processing	No query processing in distributed environment.	Yes, desirable. Too dependent on JS

5. Distributed Transaction Processing		
	No ACID transactions in system	
A. Distributed Concurrency Control	Yes, using Stacks to serialize transactions and operations	Not yet
B. Locking-Based Algorithms	Yes, locked by same node.	Not yet
	No, centralized deadlock detection.	
	Yes, distributed lock deadlock detection. Locked by same node.	
C. Timestamp-Based Algorithms	Yes, using Stacks to serialize timestamp-based transactions and operations	
D. Multiversion Concurrency Control	No maintanance of versions of data items	Not yet
E. Optimistic Algorithms	It does not follow phases: read, execute, write, validated and commit.	Not yet
F. Distributed Concurrency Control using Snapshot Isolation	No, multiversioning approach.	Not yet
G. Distributed DBMS Reliability	No times and wait until the end of timeout period to decide that message has not been delivered.	Yes
G.1 Two-Phase Commit Protocol (2PC)	No use of 2PC	Not yet
G.2 Site Failures	No recovery protocols	Yes
G.3 Network Partitioning	No networking partitioning	
6. Data Replication		
Database design	Yes, using Stack Blockchain smartcontracts based in Clarity to be executed at multiple sites	If data is requiered to be executed at multiple sites, data has to replicated
Database consistency	No consistency criteria	Not yet
	Yes weak consistency criteria	
Update propagation	No eager techniques	
	Yes lazy techniques	
Degree of replication transparency	No full replication transparency	
	Yes limited transparency	

7. Database Integration - Multidatabase Systems		
	Top-down design	Yes
	No need yet to integrate as one database	Yes
	No formal local conceptual schema	Need to add logical integration
	No formal global conceptual schema	Not yet
A. Database Integration	Physical integration	
	No ETL (extract-transform-load)	
B. Multidatabase Query Processing		
	No Multi Database System	Yes
	No need for new requirements for query processing technologies	No
8. Paralell Database Systems		
	Not a data-intensive requiring support of very large databases	No
	No paralell databases	No
9. Peer-to-Peer Data Management		
	Yes uses, Stacks ID	
	Yes, uses GAIA Storage	
	Yes, uses Clarity Smartcontracts	
10. Big Data Processing		
	It is not considered as bid data application	No
	It does use big data processing systems	No

11. NoSQL, NewSQL, and Polystores	No requirement to use Not Only SQL Systems No consideration of using Polystores	No No
12. Web Data Management	No web access of the repository of data and documents Not to be considered as publicly indexable web (PIW) Yes, can be considered as deep web (or hidden web) No XML or RBF	Yes, part of the app should accessible as web Yes, part of the app should accessible as web Yes, it should have RBF

D. Proposal of Component Design for a Decentralized Database Management System Architecture for Stacks

The evaluation shows that several components are needed for the Dapp Paradigma CrossCheck to have a functional Decentralized Application as envisioned by Tamer Özsu, and Patrick Valduriez on their publication in 2020.

From the applied evaluation in C., a selection of the functionalities desired or required are summarized:

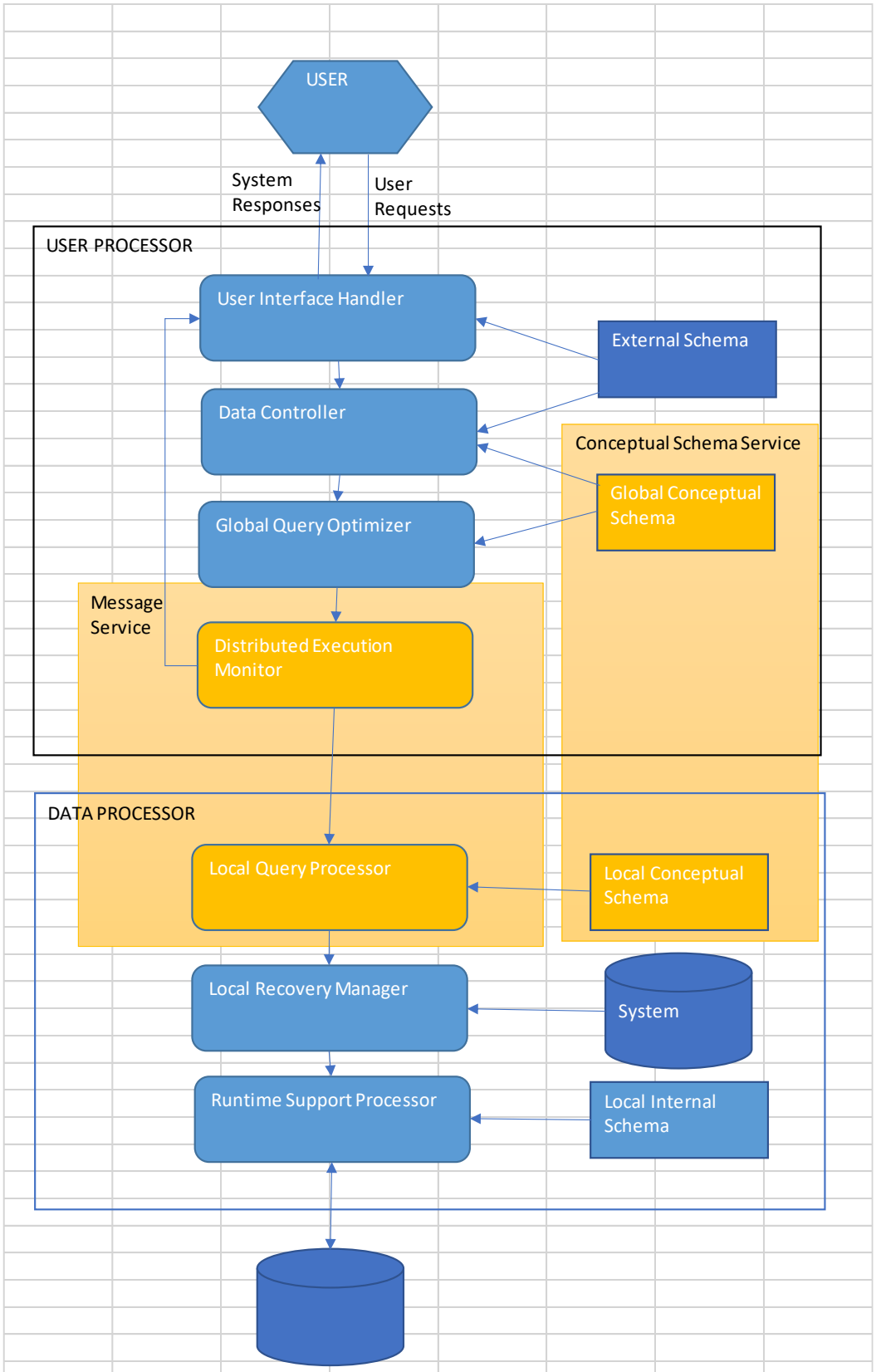
1. To have Push notifications of inserts/ updates to other nodes. Value proposition: Add interactivity and dynamism to the application. Better synchronized data.
2. To have ways to define to have some data replicated. Value proposition: Quick access to data that has authoritative level.
3. Need to formalize a Global Conceptual Schema to foster multi-application integration. Value proposition: Potential integration with other applications.
4. Need to formalize a Local Conceptual Schema to foster multi-application integration. Value proposition: Potential integration with other applications.
5. Include data access using high-level language and query processing. Value proposition: separate the physical implementation with the logical implementation.
6. Ideally, separate the physical accessibility with views. Value proposition: Control visualization of the data through logic, not physical.
7. Access control. Value proposition: control accessibility of users to the local or global data.
8. Establish standard and tools for recovery protocols for site failures. Value proposition: Have the capacity to recover from site failure in a fast way.
9. Establish components or standards for web access. Value proposition: Make the decentralized application accessible to the web, especially for massive search
10. It should have RBF. Value proposition: facilitate integration to other applications through data semantics.

Below a Diagram presents the components (shown in yellow) that should be considered priority for the improvement of the Dapp Paradigma CrossCheck.

The first component is a **Message Service** to facilitate the communications between nodes.

The second component is a **Conceptual Schema Service** that helps to define the Global and Local Conceptual Schemas in order to foster integration.

Other components can be identified in this evaluation like high-level and query processing, recovery protocols, webaccess and views accessibility among some. The study of those components can be addressed after the first two components have been integrated.



Components of a Distributed DBMS based on Tamer Özsu, and Patrick Valduriez

E. Assessment of Potential Components for the Stacks Decentralized Application Development Framework

1. Messaging Services

A messaging server is a middleware program that handles messages that are sent for use by other programs using a messaging application program interface (API). A messaging server can usually queue and prioritize messages as needed and saves each of the client programs from having to perform these services.

Messaging and messaging servers usually are organized in one or both of two models: point-to-point messaging and subscribe/publish messaging.

Source https://whatis.techtarget.com/definition/messaging-server

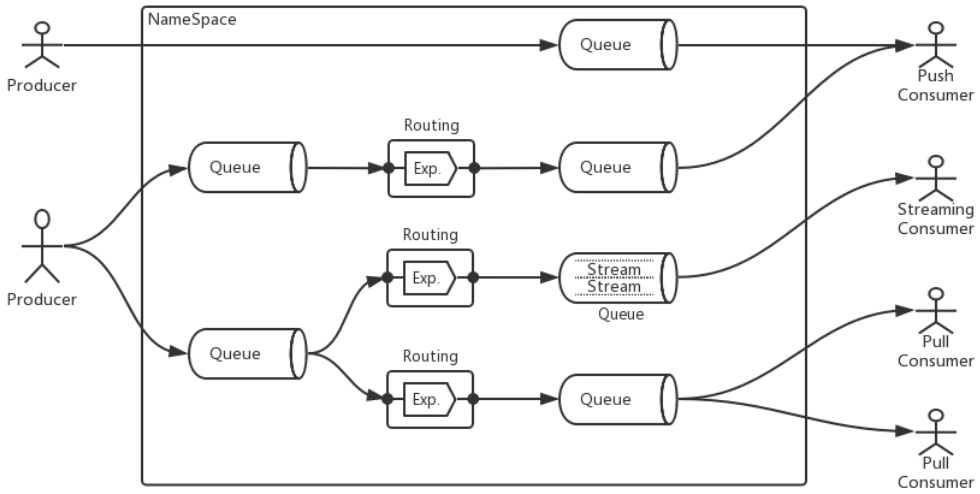
1.1 Open specification for distributed messaging

The industry join force to write an open specification for distributed messaging. It is useful to have a basic understanding of the specificactions that have been considered. This effort is called Open Messaging (http://openmessaging.cloud/).

It is a cloud native, vendor-neutral open specification for distributed messaging

Messaging and Streaming products have been widely used in modern architecture and data processing, for decoupling, queuing, buffering, ordering, replicating, etc. But when data transfers across different messaging and streaming platforms, the compatibility problem arises, which always means much additional work. Although JMS was a good solution during the past decade, it is limited in java environment, lacks specified guidelines for load balance/fault-tolerance, administration, security, and streaming feature, which make it not good at satisfying modern cloud-oriented messaging and streaming applications.

The OpenMessaging Domain Architecture is described in the following diagram.



The Stream element is an abstract concept of partition, shard, message group, etc.

The following concepts are part of the OMS Domain Architecture.

Namespace

Namespace like a cgroup namespace, to create an isolated space with security guarantee. Each namespace has its own set of producer, consumer, topic, queue and so on. OpenMessaging uses MessagingAccessPoint to access/read/write the resources of a specified Namespace.

Producer

OpenMessaging defines two kinds of Producer: Producer and BatchMessageSender.

Producer, provides various send methods to send a message to a specified destination which is a queue in OMS.

BatchMessageSender, focuses on speed, the implementation can adopt the batch way, send many messages and then commit at once.

Consumer

OpenMessaging defines three kinds of Consumer: PullConsumer, PushConsumer and StreamingConsumer. Each consumer only supports consume messages from the Queue.

PullConsumer, pulls messages from the specified queue, supports submit the consume result by acknowledgement at any time. One PullConsumer only can pull messages from one fixed queue.

PushConsumer, receives messages from multiple queues, these messages are pushed from the MOM server. PushConsumer can attach to multiple queues with separate MessageListener and submit consume result through ReceivedMessageContext at any time.

StreamingConsumer, a brand-new consumer type, a stream-oriented consumer, to integrate messaging system with Streaming/BigData related platforms easily. StreamingConsumer supports consume messages from streams of a specified queue like a iterator.

Queue

Queue is the basic and core concept of OMS, the source of a Queue can be a producer or a routing.

It is noteworthy that a Queue may be divided into streams, a message will be dispatched to a specified stream by MessageHeader#STREAM_KEY.

Queue also accepts messages from Producer directly, sometimes, we want to the shortest path from Producer to Consumer, for performance.

Routing

The Routing is in charge of processing the original messages in source queue, and routing to another queue. Each Routing is a triple consists of source queue, destination queue and expression. The messages will flow through the expression from source queue and destination queue.

The expression is used to handle the flowing messages in Routing. There are many kinds of expression, the filter expression is the most commonly used in many scenarios and is the only type defined by this OMS version. In the future, the OMS may support deduplicator exp., joiner exp., rpc exp., and so on.

What's more? Routing can cross the network, message can be routed from a network partition to another partition.

1.2 Some leading Message Server Initiatives

A search was conducted of some of the actual leading Message Servers available worldwide.

a) Apache Kafka

More than 80% of all Fortune 100 companies trust, and use Kafka.

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications

Source https://kafka.apache.org/

b) Apache Pulsar

Apache Pulsar is a cloud-native, distributed messaging and streaming platform originally created at Yahoo! and now a top-level Apache Software Foundation project

Source https://pulsar.incubator.apache.org/

c) Apache RocketMQ

Apache RocketMQ is a distributed messaging and streaming platform with low latency, high performance and reliability, trillion-level capacity and flexible scalability. It consists of four parts: name servers, brokers, producers and consumers. Each of them can be horizontally extended without a single Point of Failure. As shown in screenshot above.

Source https://rocketmq.apache.org/docs/rmq-arc/

d) SAP Message Server

The SAP Message Server is a part of the ABAP Server central services instance (ASCS instance).

Since the ASCS instance in an SAP system is running occurs only once, only one message server in each system. The SAP Web Dispatcher carries out the following tasks:

• Central communication channel between the individual application servers (instances) of the system

• Load distribution of logons using SAP GUI and RFC with logon groups

• Information point for the SAP Web Dispatcher and the ABAP instances (each instance first logs on to the message server)

When an instance is started, the dispatcher process contacts the message server so that it can announce the services it provides (DIA, BTC, SPO, UPD, and so on). If the connection setup to the message server fails, an entry is made in the system log (syslog).

If the message server stops working, it must be restarted as quickly as possible, to ensure system continues to operate trouble-free.

Source https://help.sap.com/viewer/77b3972f873044acb3a70258f3984c64/7.52.4/en-US

e) RabbitMQ

RabbitMQ is the most widely deployed open source message broker.

With tens of thousands of users, RabbitMQ is one of the most popular open source message brokers. From T-Mobile to Runtastic, RabbitMQ is used worldwide at small startups and large enterprises.

RabbitMQ is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements.

RabbitMQ runs on many operating systems and cloud environments, and provides a wide range of developer tools for most popular languages.

Source https://rabbitmq.com/

f) Genesys Message Server

The Genesys Framework component that provides centralized processing and storage of every application's maintenance events. Events are stored as log records in the Centralized Log Database, where they are available for further centralized processing. Message Server can also be set up to produce outbound messages (alarms) that are triggered by configured log events. If it detects a match, it sends the alarm to Solution Control Server for immediate processing.

Source https://docs.genesys.com/Glossary:Message_Server

g) Apache ActiveMQ

Apache ActiveMQ™ is the most popular open source, multi-protocol, Java-based messaging server. It supports industry standard protocols so users get the benefits of client choices across a broadrange of languages and platforms. Connectivity from C, C++, Python, .Net, and more is available. Integrate your multi-platform applications using the ubiquitous AMQP protocol. Exchange messages between your web applications using STOMP over websockets. Manage your IoT devices using MQTT. Support your existing JMS infrastructure and beyond. ActiveMQ offers the power and flexibility to support any messaging use-case.

Source https://activemq.apache.org/

h) Matrix

Matrix is an open source project that publishes the Matrix open standard for secure, decentralised, real-time communication, and its Apache licensed reference implementations.

Maintained by the non-profit Matrix.org Foundation, aims to create an open platform which is as independent, vibrant and evolving as the Web itself... but for communication.

As of June 2019, Matrix is out of beta, and the protocol is fully suitable for production usage.

Source https://matrix.org/

i) Dcomms

Dcomms - open source, fully decentralized P2P messenger with encryption and authentication

Source http://dcomms.org/

j) Firebase Your server environment and FCM

The server side of Firebase Cloud Messaging consists of two components:

- The FCM backend provided by Google.

- Your app server or other trusted server environment where your server logic runs, such as Cloud Functions for Firebase or other cloud environments managed by Google.

Your app server or trusted server environment sends message requests to the FCM backend, which then routes messages to client apps running on users' devices.

Source https://firebase.google.com/docs/cloud-messaging/server

1.3 Conclusion of the Assessment of Message Servers

The message server is a component that is being integrated widely among the multiple developed applications. The observed difference is that most of these message servers are more focused to centralized computing, just a few are following the path to decentralization. In this arena, Matrix is standing out of the rest in that respect. At the same time, the footprint to deploy some of these message servers are quite large. Probably a smaller footprint for the message servers could serve better specially when a massive deploy is needed.

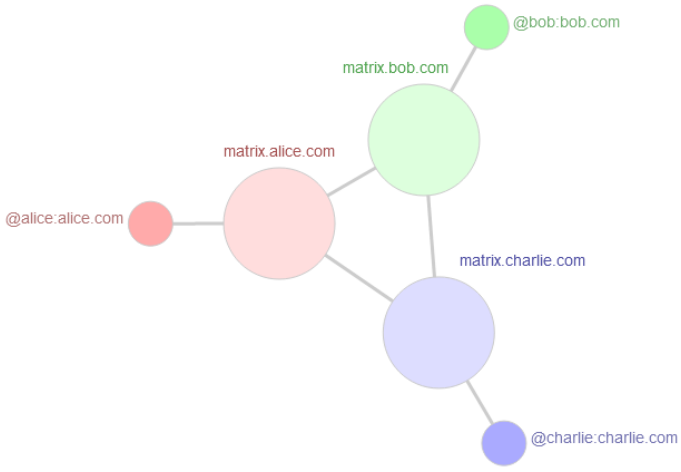
Matrix makes an emphasis about that difference and the suggestion is that this message serve should be tested as a component for the Stacks Decentralized Development Framework.

“Matrix is really a decentralised conversation store rather than a messaging protocol. When you send a message in Matrix, it is replicated over all the servers whose users are participating in a given conversation - similarly to how commits are replicated between Git repositories. There is no single point of control or failure in a Matrix conversation which spans multiple servers: the act of communication with someone elsewhere in Matrix shares ownership of the conversation equally with them. Even if your server goes offline, the conversation can continue uninterrupted elsewhere until it returns.

This means that every server has total self-sovereignty over its user’s data - and anyone can choose or run their own server and participate in the wider Matrix network. This is how Matrix democratises control over communication.

By default, Matrix uses simple HTTPS+JSON APIs as its baseline transport, but also embraces more sophisticated transports such as WebSockets or ultra-low-bandwidth Matrix via CoAP+Noise.”

The illustration that follows shows the nodes that are interacting through the message servers when they are exchanging messages.



2. **Conceptual Schema Service**

Conceptual schema design is the process of generating a description of the contents of a database in high-level terms that are natural and direct for users of the database. The process takes as input information requirements for the applications that will use the database, and produces a schema expressed in a conceptual modelling notation, such as the Extended Entity-Relationship (EER) Data Model or UML class diagrams. The challenges in designing a conceptual schema include: (i) turning informal information requirements into a *cognitive model* that describes unambiguously and completely the contents of the database-to-be; and (ii) using the constructs of a data modelling language appropriately to generate from the cognitive model a conceptual schema that reflects it as accurately as possible.

Borgida A., Mylopoulos J. (2009) Conceptual Schema Design. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_642

2.1 Some leading Conceptual Schemas

Several Conceptual Schemas were found and it requires further research about this component. Its study should be considered as continuation of this study.

F. Proposal of a plan of action to continue or not with the project.

It became clear during this study the usefulness of creating the prototype of an Evaluation Framework of Decentralized Database Management Systems. It facilitated the understanding of the particular needs of the Dapp Paradigma CrossCheck that could be covered using a decentralized application approach.

This is a suggestion of plan of action to continue with the project:

1. Look into ways to distribute among the Stacks Community the publication Principles of Distributed Database System M. Tamer Özsu, Patrick Valduriez,, Springer Fourth Edition 2020 (https://link.springer.com/book/10.1007/978-3-030-26253-2) and/or educational content about the subject.
2. This study has to be expanded considering the rest of the potential components for a decentralized development framework as Stacks.
3. Consider designing and developing a prototype of integration of the Matrix Message Server to the Stacks Development Framework and Domain Name Service.
4. Promote developer of Dapps to use this integration.
5. Evaluate how this component can be massively used by the Stacks Community as the GAIA server and S3 storage components.