EFFICIENT OPEN SOURCE DISTRIBUTED ERP SYSTEM FOR LARGE SCALE ENTERPRISE

MOHAMED ELMASSRY[1†] and SAAD AL-AHAMADI[2††],
mohamed.y.elmassry@gmail.com salahmadi@ksu.edu.sa
[1] Computer Science Department, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia, [2] Computer Science Department, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

Summary
Enterprise Resource Planning (ERP) is a software that manages and automate the internal processes of an organization. Process speed and quality can be increased, and cost reduced by process automation. Odoo is an open source ERP platform including more than 15000 apps. ERP systems such as Odoo are all-in-one management systems. Odoo can be suitable for small and medium organizations, but duo to efficiency limitations, Odoo is not suitable for the large ones. Furthermore, Odoo can be implemented on both local or public servers in which each has some advantages and disadvantages such as; the speed of internet, synced data or anywhere access. In many cases, there is a persistent need to have more than one synchronized Odoo instance in several physical places. We modified Odoo to support this kind of requirements and improve its efficiency by replacing its standard database with a distributed one, namely CockroachDB.

Key words:
Odoo, ERP, distributed ERP systems, distributed database, CockroachDB. Open Source ERP.

1. Introduction

One of the critical enablers for top managements and decision makers is the information system. We can consider information systems as the main component of nowadays' scene. Enterprise Resources Planning (ERP) is a software that introduces wide options of application functionality for all organizations' sizes while supporting a significant part of any kind of business activities.

In the large size companies (having more than 500 employee), there is a persistent need to install and operate more than one ERP instance, each of which may be customized in a different way [18]. Consequently, a need has appeared to organize data between several information systems – not only between ERP and other systems like CRM, website, E-commerce, manufacturing, billing system, accounting, warehouse, project management and inventory systems, but also between different instances of ERP systems.

Odoo is an open source platform that contains ERP system which contains a variety of applications, such as Accounting, inventory management, customer relationship management CRM and many other applications like CAD, PDM, and SCM. These applications work consistently with each other to manage companies of all sizes. One application in Odoo is made up of one or several Odoo modules which are built to work tightly with PostgreSQL as Object-Relation Database Management System (ORDBMS). With time, and as the amount of data stored in PostgreSQL DB (Database) increases, the performance of the system will be reduced, which leads to a bad customer experience [2]. Odoo can be implemented on both local and public servers. Each implantation has some advantage and disadvantage like the speed of internet and synced data or logging in to the system from anywhere, office, home or outside the country. In many cases, there is a persistent need to have two or more of Odoo instances in several physical places, all of them must be synchronized with each other. These cases like but not limited to: Having a system for both the company and its branches. The default Odoo system does not allow syncing the data neither between two Odoo instances or between two databases. Because of this limitation, Odoo cannot be useful in lots of cases.

Some companies that operate in retail sector, such as grocery chains, restaurants chain, pharmacies, etc., often need points of sell (POS). These POS should be distributed around a specific geographic area, so these companies can serve as many customers as they can. Of course, Odoo system can be installed on the cloud, and all the sales points will be connected to the cloud to complete the sales processes, but if the internet is disconnected, the sale process will stop, which will make Odoo users completely dissatisfied. In this case, the companies will install two or more of Odoo system instances; one in the main center and one for each branch. The biggest problem is the data consistency between the central system and the branches, As Odoo server does not provide synchronization between more than one Odoo instance by default.

In this paper we will modify Odoo system to support this kind of needs by replacing its standard database with a new one (CockroachDB Open Source Database) that has the distributed system standards [1]. By default, Odoo uses PostgresDB as a database solution. Nevertheless, we have chosen CockroachDB to be an alternative solution for PostgresDB for several reasons. Initially, CockroachDB is

Manuscript received December 5, 2021
Manuscript revised December 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.12.40

built primarily on PostgresDB [1] which means that CockroachDB is a new and improved version of PostgresDB, and this feature will shorten a lot of work and compatibility problems between the work-layer interface in Odoo and the database. Another reason to make CockroachDB a magnificent choice for this project is that it is built using a Google Spanner technology. Google Spanner is a scalable database in which a single instance can be run from anywhere [25]. Moreover, CockroachDB is an open source project. This means that it can be used without paying any financial costs for licenses [1].

The remainder of this paper is organized as follows: Section 2 “Related works”: In this section, we explain the theoretical problem background that we try to solve in details. To achieve that, we describe Odoo in general. Then we describe distributed systems’ pros & cons and we illustrate CockroachDB system. After that, we describe other aspects like Postgres, ERP system, Google spanner and Open source licenses. In Section 3 “Literature Review”: We will review the solutions and previous works that tried to solve the same problem in the ERP system area. We will also review what the advantages and disadvantages of each solution are. Section 4 “Suggested Design & Implementation”: First, we describe the system requirements. Then, we illustrate the benefits of the CockroachDB components that we used. Then we explain how we will implement the new Odoo system, also we show the changes that we will apply to the current Odoo system. In Section 5 “Experiments”: We will show the experiments that have been made and how the proposed design surpasses the other designs. In Section 6 “Conclusion & Future Work”: We will list the new features and improvements that we expect to be applied to Odoo system after proposed modifications; including security, reliability, speed, performance, and failure resistance and disaster resistance. Also, we summarize our experiment steps and its results. Finally, we suggest how we can enhance Odoo performance in the future.

2. Related Worktyle

Manuscripts Distributed systems are systems with multi-components that are connected through the network. The only way to communicate between those components is through message passing [15]. In each distributed system, there is a common goal that the components of this system try to communicate with each other.

There are three important features in any distributed system. First, components synchronization. Second, not having a global clock between those components. Third, the system should be failure independent (The system should not fail if any of its components fails). There are many examples of distributed systems ranging from sensitive systems of information security and financial transactions to systems of electronic games and entertainment. An application that operates on a distributed system is called a distributed program. When we write such programs, we call this process distributed programming [16]. There are other ways of passing messages in distributed systems such as using pure HTTP [16]. There are three essential aspects of a distributed system which include:

Availability: we can define availability as one of the available system components that send a request to other parties, this request must be answered by the component or components involved in it [17].

Consistency: It means that each operation in the system works as if it has the whole control on the data item while being sequenced one after another. Any read operation that begins after a write operation completes, should answer back with the confirmation of that write operation or the confirmation of any later write operation [17].

Partition: If the network components are divided into two main sets and all requests that go from one set to another are lost we can then say that it is partitioned [17].

2.1 ERP System

Enterprise Resources Planning (ERP) is a software that works in integrating the already available information all over the support or core business that has an aim to be capable planning and managing all the available resources to an enterprise so that all business areas within a project can run well. ERP system also consists of several integrated modules, such as material management, sales, distribution, production planning, financial systems and human resources system.

2.2 Odoo

Odoo is an open source ERP system known previously as OpenERP, and it is considered the highest installed business application worldwide with more than 3,000,000 users [3]. Odoo has been used in many large companies such as; Hyundai, Toyota and Danone. As it also offers both On-Premise and Cloud ERP system, in addition of consisting of 30 primary applications such as; (sales, e-commerce, invoicing, accounting and user website management). In the time of writing, around 15,612 modules were available in the Odoo app store and more than 300 modules are added per month. Odoo is developed using

Python 2.3 and 3.5 for the latest version of Odoo 11.0 At the time of writing this paper.

Odoo provides a standardized way for developers to develop new Odoo modules or customize and modify the already existed modules. Odoo modules consist of several modules which interact with each other's and with other modules to achieve the goal of the developed module. Model inheritance and View inheritance are the main features in Odoo which allows the developer to add new features to a model or view and modify an already existed model.

We can consider Odoo as a multitenant architecture application. There are three main tiers in Odoo. First, the database tier. Obviously, this tire is for data storage. Second, the application tier. This tire is responsible for processing and functionalities as it contains all business roles. Third, the presentation tier which provides the user interface. Inside Odoo server, Odoo treats those tires as separate layers where its core is the application tier itself. In order to create a particular instance of Odoo, there are many other modules that can be installed. We can adapt this instance to specific needs and requirements through installing a combination of those modules. Moreover, Odoo framework is based on the View and Controller (MVC) architectural organization model.

Figure 1 shows an instance of Odoo deployment. As it also shows that an Odoo system consists of three main tires bases on Web embedded deployment: A PostgreSQL database tire server: This tire contains all Odoo database. The whole application data and the biggest part of the Odoo instances configuration elements like menus and privilege are stored in those databases, this tire can possibly be deployed using other deployment methodologies like clustered databases. The Odoo Server tire: This tire ensures that Odoo's logic runs optimally. This tire also includes all the business requirements or logics. There is also a dedicated layer for communicating with the PostgreSQL database called ORM engine.
It is possible to have more than one server instance for multilabel reasons like in our project. The client tire: It is a JavaScript application that runs on the client-side device like a (web browser). PostgreSQL database tire, also called data layer. PostgreSQL relational database provides the main components of this tire. However, SQL queries can be explicitly executed directly from Odoo modules. The whole application data and most Odoo elements settings are stored in databases. Clustered databases can be used to deploy this tire as we propose in this paper. We can build business requirements, or applications in top of Odoo application server (Odoo server tire). Furthermore, we can consider Odoo application server as a comprehensively framework for development that provides a wide range of options to develop the business requirements. Also, Odoo ORM (Object Relational Mapping) offers functionalities and interfaces included in those features provided by Odoo application server. In order to communicate between the standard browser applications used by the client and the Odoo application server, Odoo provides a dedicated layer to organize communication between the two parties.

From the developers' point of view Figure 2, Odoo provides these properties in the form of a programming library (API) that invokes all the benefits and properties of the upper layer and hides all the complex details in the bottom layer. Object Relational Mapping Server tire – ORM. This tire is not a noticeable feature by the developers of the Odoo apps. Odoo ORM Offers more benefits and important features above PostgreSQL server layer. The description and identification of the objects (data models) in Odoo are written in Python language and then converted into tables and fields in the database. The whole advantages of RDPMS such as; relational queries and active queries are used by Python language through this layer. For example, any convention developed in Python language can be append to any Odoo data form. Odoo also provides scalability mechanisms for different modules. It is very important to understand the working mechanism and responsibilities of ORM before starting to use it in order to be able to deal with it and with clean SQL lines. When we use the ORM layer, Odoo ensures that the data stay clean without any deformities. For example, if we use ORM layer, no data can be created through any module without using ORM layer tire.

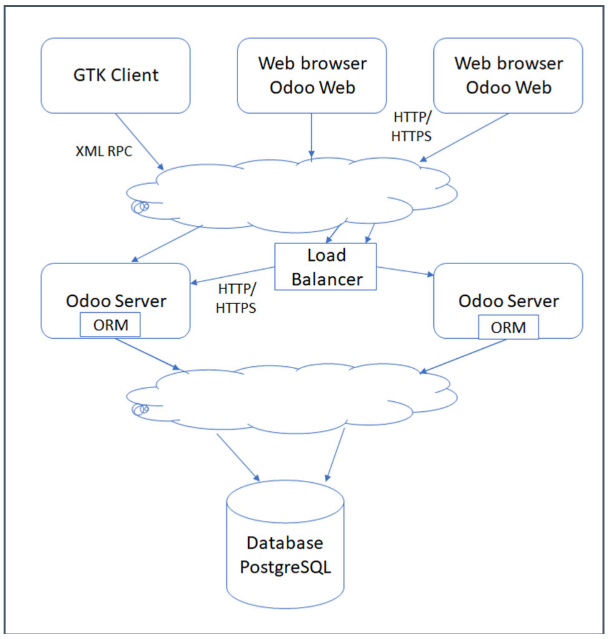

Figure 1. Web deployment Odoo architecture [2]

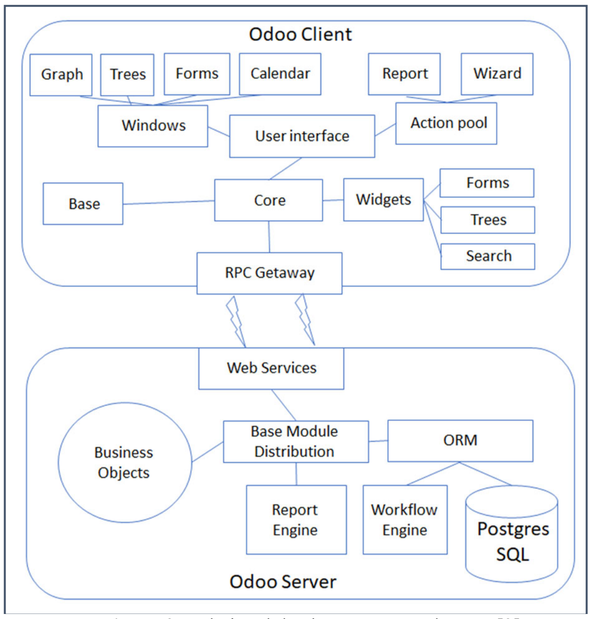

Figure 2. Relational database server and ORM [3]

2.3 CockroachDB Database

Databases can be split into two types: The relational databases and NoSQL databases. In the mid 1970's, the concept of relational databases was first developed [18]. On the other hand, the concept of NoSQL was developed at the beginning of the 20th century [19]. Nowadays, there are more than 255 different kinds of NoSQL databases [20][19]. Database systems are constantly being developed due to high inflation in the data itself.

Machine power growth can be done in X or Y axes, either vertical or horizontal. If we add more processors, hard drives space or random memory RAM in the same machine, that means we are expanding our system vertically. We will not only get a too large and expensive machine, but we will even reach a point where we will not be able to expand further. So, the best way is to expand horizontally by assembling more than one machine in one organized system (no matter if it's small or cheap). There is no doubt that horizontal expansion is more effective than vertical expansion in terms of cost and performance.

For relational databases, horizontally expansion faces many difficulties. This is because relational databases are not designed (from the beginning) to work on distributed systems [32]. In the other hand, we can store a large volume of data as we expand distribution and use NoSQL databases that allow horizontal expansion. Nevertheless, it is not easy to achieve ACID (Atomicity, Consistency, Isolation, and Durability) operations. Over time, many developers have developed the advantages and features of NoSQL databases. Therefore, CockroachDB is an example of NoSQL databases that gain more advanced features and benefits by the time from transactional databases. CockroachDB is a distributed NoSQL database and is currently a production stage. CockroachDB supports distributed system features including ACID operations and principles [22]. Developers of the CockroachDB database say that many data centres can be saved if they use a CockroachDB database because they provide the lowest rate of failure [1]. There are some similarities between CockroachDB Sqlite4 and MySQL, because all of them are using key/value store approach [24]. But CockroachDB is able to install on a single instance as much as the multi node clusters system. That means it can expand in both directions horizontally by joining the other available nodes or vertically by adding more resources.

CockroachDB provides a highly available and fault tolerance features and is designed to store three images of any data model. In the case of machine crash, data is automatically redistributed to other versions to achieve 3 data instances again [24].

Furthermore, CockroachDB, which is a version of Google Spanner, is a database with an open source license. The most beneficial feature of CockroachDB database has overcome the disadvantage of Spanner (Spanner cannot work without Google infrastructure). Because of that, CockroachDB can be implemented anywhere even on the local machine (not like Google Spanner which limits the choices for the ERP implementers).

The main difference between CockroachDB database and Spanner is that CockroachDB database does not rely on Google API's like TrueTime. CockroachDB relies on techniques built within its design to coordinate the clock between different CockroachDB nodes. In the next section, we get more about Google Spanner.

Spanner database is a scalable and global distributed database that Google designed, built, and deployed at Google infrastructure internally as one of Google projects. At the highest level of abstraction, it is typically a database that shards data across many sets of Paxos state machines in data centres that are spread all over the world [25]. Replication is used for global availability and geographic locality; clients automatically failover between replicas. Spanner automatically migrates data across machines (and across data centres) to balance load and in response to failures.

Furthermore, Spanner automatically re-shards data across machines as the amount of data or the number of servers' changes. Spanner is designed to scale up to multi-millions of machines across multi-hundreds of data centres and multi-trillions of database lines. Applications (like Odoo) can use Spanner for high availability, even in the face of wide-area natural disasters, by replicating their data within or even across continents. Google spanner is the first system to do that at a global scale [1]. Spanner assigned a global timestamp of the transactions across a distributed set of nodes. The key to those global timestamps is the TrueTime API (one of Google's APIs) and its implementation. The TrueTime API abstracts and exposes clock uncertainty and allows applications to reason with uncertainty, while the TrueTime API implementation in Google's data centres restricts the uncertainty to less than ten milliseconds. The uncertainty is very small compared to other systems where the delay between different clocks across a distributed system can reach 250 milliseconds. By having two physical clocks on each node atomic and GPS, Google's TrueTime API implementation can be achieved [25]. Because of that, Spanner is tied and restricted to Google infrastructure.

3. Literature Review

Many studies have address the ERP and information system architecture. With the fact that they are not directly mapped to a single database of a unique ERP server for a company anymore, but instead requires to be modelled inside an organization, distributed systems requirements have to be supported by both ERP and the information system [30][31]. Gattiker and Goodhue [26] discussed ERP systems' adoption in a distributed organization that consists of 20 units. In the first trial to perform ERP, the units were given much autonomy to configure the ERP instances. This strategy failed as the project quickly ran out of budget.

MHD Fawaz [27] tried to make an experimental performance comparing NoSQL and RDBMS data storage systems in Odoo ERP system to increase the performance of Odoo by replacing Postgres database with Hadoop ecosystem, but the experiment results were not as promising as expected. However, integrating an application with big data technologies opens new opportunities by providing a robust data processing framework and increase system availability (Figure 3).

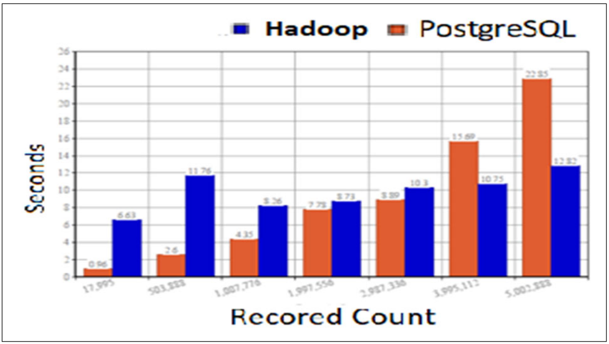

Figure 3. Comparison between PostgreSQL and HBase "Hadoop" in filter messages process by body content [27]

An N-safe technique was proposed by Frank [36]. In general, replication methods have "n" copies of data where "n" have to be more than 1. The primary replication designs that store "n" in different copies of data are n-safe, 2-safe, 1-safe or 0-safe respectively. When these n, 2, 1 or 0 of the N copies are consistent and are up-to-date at the usual operation. In some cases, it is not possible but to use the 1-safe or 0-safe replication designs. However, we cannot prefer on of the replication designs over the others as they all have different properties. [28].

Frank has once again proposed a structure that tells us the possibility of applying the concepts and properties ACID (Atomicity, Consistency, Isolation, and Durability) of the distributed systems on any ERP system as one of the trends in the enterprise application architecture [37]. And as he theoretically said, such a system can exist by using a distributed DBMS (Data Base Management System) instead of using the regular database Systems. Frank just describes the way of designing a distributed ERP system by using databases with relaxed ACID properties. The described techniques are general and in the theoretical level. These techniques need more efforts to be implemented practically on a real ERP product.

Alanne, et. al. proposed a solution based on peer-to-peer networks and web services for the distributed ERP system for small and medium enterprises [39]. Their proposed solution was using an "out-of-the-box" computer with a preinstalled software, where web services must be used to expand the functionality of the whole system. Their work shows that their solution is very much cheaper to be installed and maintained than the already available solutions. But the author did not discuss information security problems to share data among network participants. Moreover, one of the most important features of the system is that if a user requests a piece of data and this part is not available in the local area, the system will automatically search for it in other objects' databases [39]. This creates a serious problem in information security. Moreover, the authors were describing their solution in general and in the theoretical level. They hope that this design will bring a new generation of ERP systems, which may be easier to install and maintain than the traditional ERP systems [39].

Gerhard and Michael described the effect of the usefulness of the distributed ERP systems according to the quality of the material master data [28]. The author presented several issues that are in relation with the quality of the master data (data of the customers, suppliers, employees, or products). The master data comes from several systems in large organizations (not just ERP, but other systems like CRM). In addition, the problems of data entry from different sources or wrong entries are the biggest problems, along with a different number of ERP instances.

Elmasri and Navathe [40] showed the scientific concepts of distributed database systems, distributed databases, distributed database management systems, and the way the client-server architecture is used as a platform for database application development in their book " Fundamentals of Database Systems " [40] in which they describe how distributed databases bring the advantages of the distributed computing to the management domain of the database. They also, describe the possible mechanisms by which the data can be divided into distributed databases systems. They also proposed a mechanism for implementing queries in distributed databases. Finally, they reviewed Oracle distributed databases.

Tamer and Valduriez described more principles of distributed database systems in their design and structure [42]. Moreover, they have discussed deeper concepts in distributed databases such as; parallel databases, linking distributed databases with cloud computing, peer-to-peer databases and other advanced concepts.

Google launched Google Spanner by disclosing it in its scientific paper " Google's globally distributed database" [25]. Google says it is to be the first data distribution system all over the world. They explained the details of their design, the way it works, its characteristics and its features. But its main disadvantage is that Spanner was tied with Google's infrastructure (TrueTime API) [25].

H. Daudi & j. Vora in Serpent Consulting Services Pvt. Ltd. have developed a module Odoo that provides a solution for synchronizing two Odoo databases together. They call it Multi-DB Synchronization [9].

Toolkt Co. did a similar work called OpenERP-Base_Synchro for Odoo v7. The OpenERP-Base_Synchro module provides the merging or the transferring the data from one database into another. It also takes care of all the defined constraints over the objects of other reserves' databases. [10]

BrowseInfo Co. uses H. Daudi & j. Vora work and adds more features to it like the automatic scheduled action, report after synchronization and details of the database [11]. They called it Auto Multiple Database Synchronization or bi_base_synchro.

However, all the three-previous works have some obvious disadvantages or problems. First, the three solutions depend on the Odoo application layer, and this requires more resources for the application servers. Also, if one of the application servers has a failure, the whole syncing process will stop. Second, the admin user needs to identify the objects one by one in both databases to be synced. However, this is not practical from the user's point of view, and if we consider the hidden object (Odoo define everything as object menus, action, and even views, etc. not just a natural data object like a student, employee, product, and others. Therefore, there are thousands of objects), it will be unpractical at all. Third, this solution is acutely to union the records in both database processes. It will work fine if our requirement is to import our old recurred to the new database.

4. System Design and Implementation

To optimize Odoo performance, availability, failure resistance, disaster resistance, reliability and security, we need to adjust Odoo to store data into database support distrusted system features to get the benefits of a scalable data management system. As we described in Section 2.1 "Odoo system architecture", there are several components and layers (see Figure 2) such as PostgreSQL database, Odoo server, Server – ORM, Server–Web, Modules, Clients and Odoo MVC, each of these components is connected as Figure 2 describes.

As we previously proposed, we will replace PostgreSQL database with our selected DB, which is CockroachDB database. The Odoo architecture will change, Odoo ORM will be connected with CockroachDB instead of PostgreSQL. This change will not affect either the representation of the Odoo models nor the views or controllers Because CockroachDB is PostgreSQL [1]. CockroachDB will do the rest of the distributed system work. Our new Odoo system will gain more layers from CockroachDB. At its highest levels, CockroachDB converts the SQL statements of the clients rinto key-value (KV) data, which gets distributed to the nodes and then written to the disk. In our case, SQL statements come from Odoo ORM. CockroachDB design and architecture is the process by which we use to accomplish that, which is manifested as some layers that interact with those directly connected with it (both up and down it) as relatively not transparent services. The performed functions by each layer are described in the following Table. Some interactions occur between layers which are not explicitly articulated and require an understanding of the function of each layer to understand the entire process.

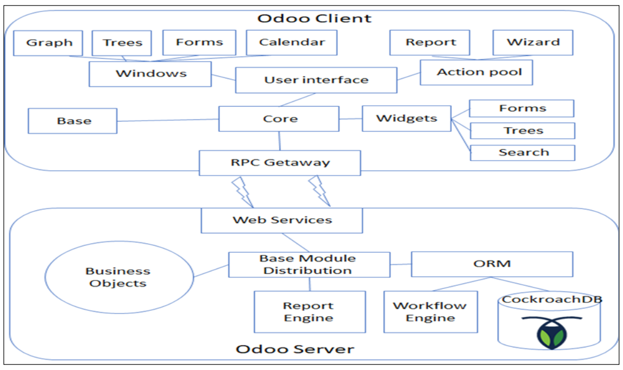

Figure *4.* As Figure 2 but Odoo ORM connected with CockroachDB

Table 1. New CockroachDB layers added to Odoo ORM

#	Layer	Purpose
1	SQL	Translate client SQL queries to KV operations.
2	Transactional	Allow atomic changes to multiple KV entries.
3	Distribution	Present replicated KV ranges as a single entity.
4	Replication	Consistently and synchronously replicate KV ranges across many nodes. This layer also enables consistent reads via leases.
5	Storage	Write and read KV data on disk.

SQL: This layer helps the developers to run SQL queries as in a traditional environment. It provides all the familiar terms and concepts such as schema, tables, and indexes. All feature sets are used by CockroachDB and its own SQL.

Distributed Key-Value Store: We can develop large tables and indexes as HBase, BigTable, and others. Because the SQL layer communicates with the distributed key-value store.

Distributed Transactions: we can consider transactions as the core part of our Odoo application. The implementation of this feature manages the transition from SQL to stores and ranges.

Nodes: They can either be virtual or physical machines. Nodes are the servers that store our data. Routes messages to different nodes of our cluster are done by the distributed key-value store.

Store: Each store can hold many ranges, and each node can contain one or more stores. RocksDB, is an open source storage engine that manages ranges.

Range: The lowest level of key-value data. Each store contains ranges, and each range covers a segment of the more important key-spaces.

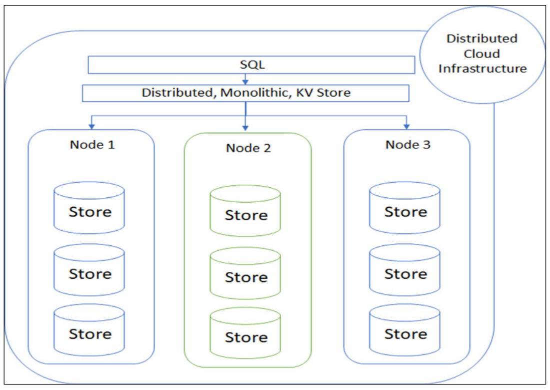

Figure 5. Architecture Diagram of CockroachDB

In the Figure above, every store potentially contains of a number of ranges. These ranges are replicated by using the Raft consensus protocol. The diagram below is a blown-up version of stores from four of the five nodes in the previous figure. Each range is replicated in three ways using Raft. The color coding shows the associated range replicas

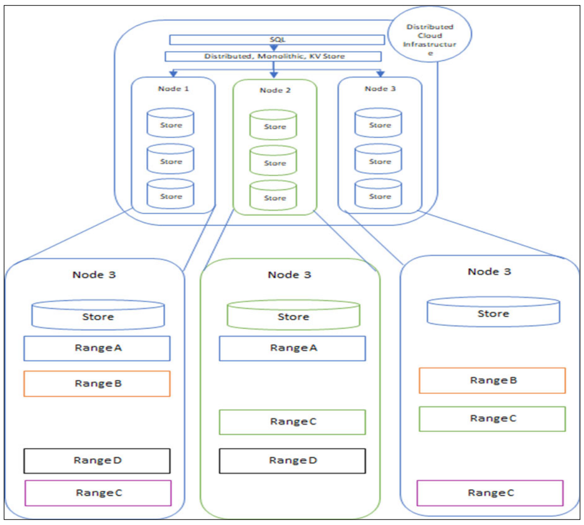

Figure 6: Blown up version of stores from Figure 5

Once we have CockroachDB installed in each node, we will connect each of them with Odoo instance application layer. Figure 7 to simplifies Figure 4.

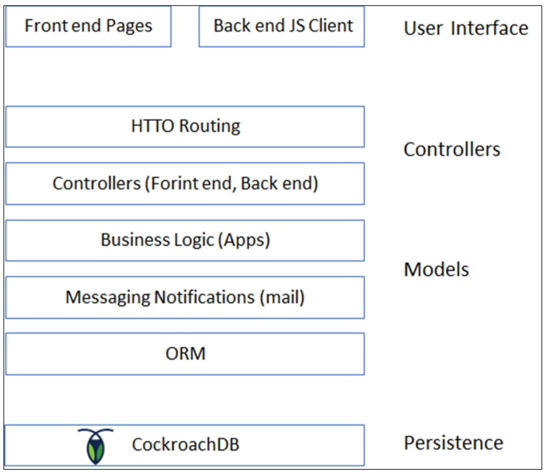

Figure 7: CockroachDB and Odoo layers simplify Figure 4

In the next Figure, we can see how four Odoo instances synced with each other's through CockroachDB.

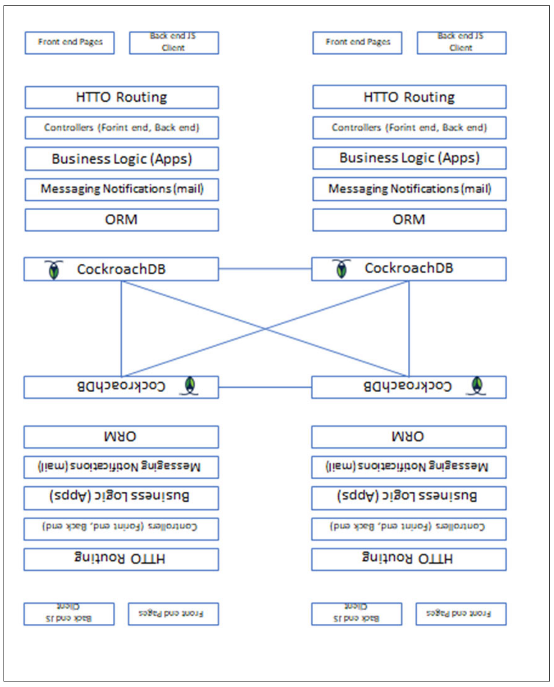

Figure 8: Four Odoo instances are synchronize with each other by Cockroach database.

5. Experiments

A. Structures Installing Experiments

We have installed Odoo in four different structures. It is noteworthy that we faced a number of challenges in the practical implementation. We will mention them in the next few lines.

Structure A "Odoo as it is": The purpose of this experiment is to install Odoo as it is using PostgreSQL databases in order to compare it later with the proposed structure "new Odoo distribution ERP system". In this experiment, we installed Odoo on a local Ubuntu server (2 GB MEMORY, 1 CPU, 50 GB SSD DISK and 2 TB TRANSFER. These specifications have been standardized for all later testing processes). By using the following commands:

```
# apt-get install postgresql -y
wget -O - https://nightly.odoo.com/odoo.key | apt-key add -
#   echo   "deb   http://nightly.odoo.com/8.0/nightly/deb/   ./"   >>
```

Figure 9: Normal Ubuntu installing Odoo commands

We did not have any installation problems in this experience because we used the pre-installed Odoo bags as they are.

Structures B "Enhanced Odoo with CockroachDB": The objective of this experiment is to know whether Odoo system can work on the CockroachDB databases in a practical way and to identify what the potential problems and difficulties are. We installed cockroachDB [1] and Odoo, but this did not work, and we received many error messages. The error messages were saying that the Odoo system could not work because there were no PostgreSQL databases. We then changed the Odoo database configuration file, to connect with CockroachDB databases. The error messages still appeared, reporting that there were errors in executing SQL queries. After investigating and tracking the error, we found that this was because there were differences between the CockroachDB databases and PostgreSQL databases syntax. Although the difference was very simple in some queries, the system was not able to work. At that point, we had two choices. Whether to review all PostgreSQL databases SQL queries in Odoo's framework and modify them to CockroachDB databases version which takes a lot of time and effort, or to create a database using PostgreSQL and Odoo framework and install all the Odoo modules that we need. Then, we take a back-up of this database using the dump tool and the pg_dump command. Then, we transfer them to the CockroachDB database using the IMPORT command, which is available in the CockroachDB database, and will convert the hole PostgreSQL backup into a compatible CockroachDB database backup which shortens a lot of effort and time. This option is good for researches and tests, but not for the production and operation, because there will be more errors when we want to install more Odoo modules or update the old ones. In this project, we chose the second option for the above reasons. At the moment, we have a single Odoo system running on CockroachDB databases that work locally as well.

Structures C "n Enhanced Odoo instance with centralized load balancer (n=4)": In this experiment, we are trying to connect four Odoo instances of enhanced Odoo systems that were built in the previous experiment to test what benefit we gained from the CockroachDB databases. To make it easier in this experiment, we have proposed a fifth server to serve as a load balancer in order to facilitate the testing process later, as we need to target this server only and it will distribute the queries to the other four Odoo instances or servers. The first problem we faced was that in the previous two experiments, we installed the system locally, so we had to do many settings in the LAN to connect the nodes together. For that reason, we transferred the experience to the cloud. After we installed four enhanced Odoos on four virtual machines (VMs) on DigitalOcean environment, we have activated a fifth pre-configured server VM by Digital Ocean to act as a load balancer. At this moment, we have “n” Odoo instances connected to each other by the load balancer where n=4 (n is the number of Odoo instance).

Structures D "n Enhanced Odoo instance with decentralized load balancer (n=4)": In the previous experiment, we have already linked four enhanced instances of Odoo systems by the load balancer successfully, but we have not yet achieved the proposed structure (the distributed Odoo system). One of the most important features of distributed systems is not having a central node, which is the opposite in experiment C. So, we have to remove the fifth node and distribute its task (load balancing) to the other four VMs. We distribute the task by installing one of the load balancer tools on each node such as HAProxy. This structure will enable us to target any node in our system and this node will distribute the load evenly between the four nodes (It takes itself into account). If one of them is destroyed, the rest will not be affected and the system will continue to function as if it had not been destroyed.

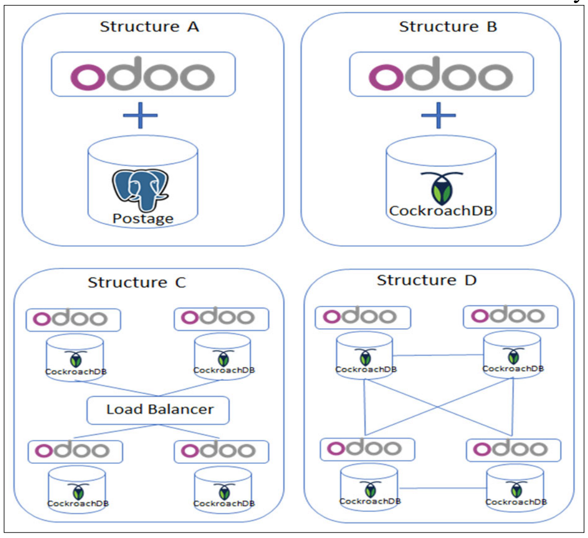

Figure 10: Four experiments structures A,B,C and D

Table 2. Four structures description summarization

#	Server (VM) Description
A	Default Odoo with Postgre DB
B	Enhanced Odoo with CockroachDB
C	n Enhanced Odoo instance with centralized load balancer (n=4)
D	n Enhanced Odoo instance with decentralized load balancer (n=4)

B. Experiments

Comparative criteria: There is a set of criteria and specifications that will improve in Odoo system when we replace PostgreSQL databases with the CockroachDB databases; Normal user level of effort: How many techniques and steps should be taken in case the average user wants to get this system? We will measure this standard on the easy, medium, hard and challenging scale, where easy means that the average user does not need to learn any technique or do anything unusual to get the system (All actions will be downloaded and fully prepared). Fault-tolerance: If one of the nodes fails in the system, will the system be able to resist the failure and complete its normal operation? If we say yes, it means that there is no central node in the system. In other words, the system is able to continue working even if any node in that system failed as long as there is at least one node still working. Availability - DDOS attack resistance: What is the size of the attack that the system can withstand without falling? We will measure it in Giga Bytes (GB). Scalability: Can the system increase or decrease the number of nodes in an unnoticed manner by the end user or not? Response time: The time the system needs to answer the user's query. We will measure it by the number of seconds. Cost: We will calculate the system cost by the number of nodes used to configure this system multiplied with x (x is a certain amount of money measured in a financial unit). Assuming that all used nodes have a uniform specification (as we mention in installing experiment A, 2 GB MEMORY, 1 CPU, 50 GB SSD DISK and 2 TB TRANSFER) and each of them will cost x$.

We have done a number of experiments to compare the four structures based on the previous six criteria. The following table shows the results of these experiments.

Table 3: Shows the results of 6 experiments For Structure A and B

#	Description		A	B
1	Normal user level of effort		Easy	Medium
2	Fault-tolerance		None	None
3	Availability - DDOS attack		2 GB	2 GB
4	Scalability		None	None
5	response time (seconds)	1 user * 200 records load	6	2.7
		2 users * 200 records load	11	5
		3 users * 200 records load	16	8
		4 users * 200 records load	22	11
		5 users * 200 records load	26	14
		6 users * 200 records load	31	17
		7 users * 200 records load	37	19
		8 users * 200 records load	45	21
		9 users * 200 record load	50	24
		10 users * 200 records load	55	27
6	Cost		x $	x $

Table 4: Shows the results of 6 experiments For Structure C and D

#	Description	C	D
1	Normal user level of effort	Hard	Challengin g
2	Fault-tolerance	None*	Yes
3	Availability - DDOS attack	7.5 GB	8 GB
4	Scalability	Yes	Yes

5	response time (seconds)	1 user * 200 records load	3	2.7
		2 users * 200 records load	3	2.7
		3 users * 200 records load	3	2.7
		4 users * 200 records load	3	2.7
		5 users * 200 records load	6	5.2
		6 users * 200 records load	6	5.2
		7 users * 200 records load	6	5.2
		8 users * 200 records load	6	5.2
		9 users * 200 record load	9	8.2
		10 users * 200 records load	9	8.2
6	Cost		5 x $	4 x $

Experiment 1: As shown in Table 3 and 4, structure A is the easiest in terms of installation and preparation, because the programs are already-made by Odoo, all you have to do is download and install it through a simple and easy interface, while in the rest of the structures, users must take several steps to obtain the desired system as described in Chapter 6.1.

Experiment 2: In Structures A and B, the whole system will fail if one of its nodes fails because these structures have only one node. While structure C have four nodes, each of which is connected to all the other nodes. That means if one of them fails, the other three will remain connected. Structure D can actually resist failure if one of the Odoo instances fails, but will not resist the failure if the load balancer fails. For this reason, we cannot say that structure D has fault tolerance criteria.

Experiment 3: Using a tool to generate many requests to make pressure on the four systems such as TPC Benchmark (TPC Benchmark is a dataset with over than 2 terabytes in size). We used this tool to simulate the denial of service attack and test the four systems. This tool allows you to control the size of the desired attack and measures when the system stops responding at the same time. The best result was for structure C where it stopped responding when we hit it with 8 GB attack.

Experiment 4: In this experiment, we try to test the possibility of increasing the number of nodes without affecting the performance of the system. It is clear that the number of nodes cannot be increased in structures A and B. This is because these systems are based on a single node. Also, there is no protocol to link those nodes with each other's. Therefore, we cannot increase the number of nodes. On the other hand, structures C and D can increase or decrease nodes easily and without end user notes.

Experiment 5: In this experiment, we tested the response time of the four structures by simulating the number of users (scaling up from 1 to 10) trying to call 200 records from each system. Table 5 shows that structure D achieves the best and shortest response time for all numbers of users, which is even better than structure C that contains 5 nodes. We can also observe that structure B responds better than structure A with a 50% less response time. This means that when using the CockroachDB database with the Odoo system, the system efficiency increases by two times.

Experiment 6: In this experiment, all we have to do is count the number of nodes in each system and multiply them by the number x and then compare them to each other. The results show that structures A and B are the cheapest, while C is the most expensive structure, and structure D is the third in terms of cost. Also, we can note that because structure C consumes one node as load balancer, it is always more expensive than structure D by one x$.

C. Response Time of the Extended Comparative Experiments

In this section, we focused on the response time of the four structures. In the fifth comparative experiment, we fixed the variable number of records by 200 records (we will name this variable later as r) and we made an increase in the number of users from 1 to 10 (we will name this variable later as u). This comparison gives us a general indication that there is a difference in performance between the four structures but does not describe what the actual efficiency function for each structure is. To find out, we had to extend this test and try the values of records' number and users' number on each structure to see what the efficiency function (growth function) for each structure is. The following Figure shows a scaling up in the number of users from 1 to 100 for each of the four structures and with 200 records as constant (from u=1 to u=100 where r=200 and n=4 in structures C and D). The following figure illustrates the efficiency of each structure by drawing the growth function.

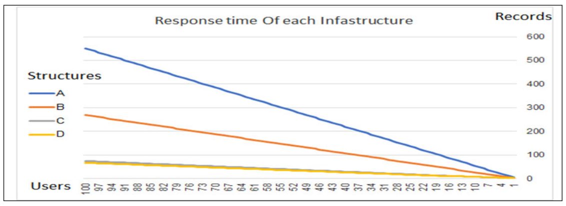

Figure 11: Illustrates the efficiency of each structure by drawing the growth function for structure A, B, C and D

Figure 11 illustrates the efficiency of each structure by drawing the growth function. As it also shows that all structures have a linear growth function, while there is a significant improvement in slope "a". where "a" is represented as in the following equation:

$$f(x) = ax + b,$$

But what if we increase the number of records and users at the same time? Figures 12, 13, 14 and 15 represent the growth function of increasing users and records numbers at the same time in the experiments for the A, B, C and D structures respectively.

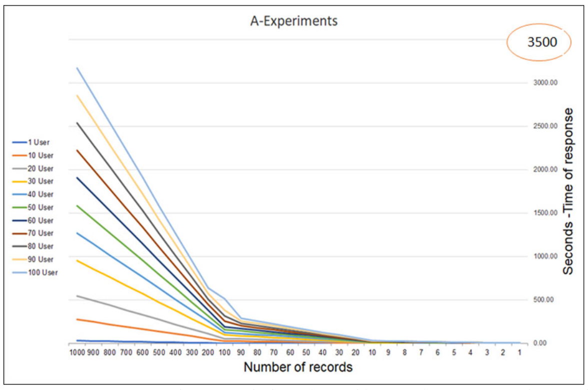

Figure 12: Represent the growth function of increasing users and records numbers at the same time for structure A

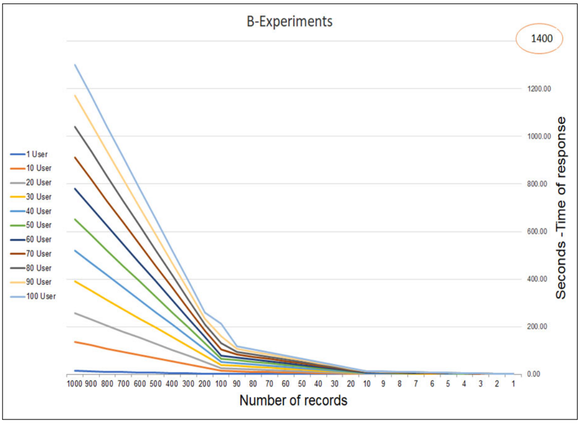

Figure 13**:** Represent the growth function of increasing users and records numbers at the same time for structure B

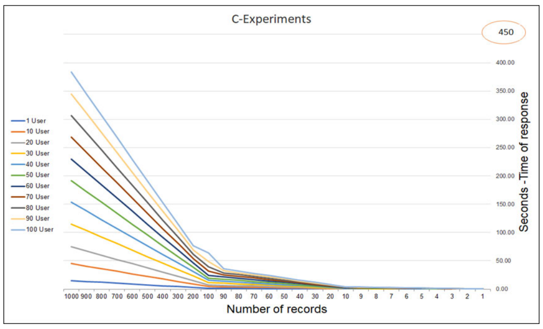

Figure 14: Represent the growth function of increasing users and records numbers at the same time for structure C

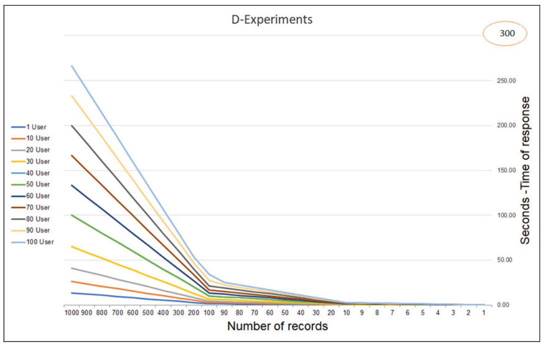

Figure 15: Represent the growth function of increasing users and records numbers at the same time for structure D

We have also noticed that all structures in the previous experiments have an exponential growth function. Structures C and D show better performance. (Note that we will fix the number of users and records by the upper limit of the previous experiments of 100 users and 1000 records.)

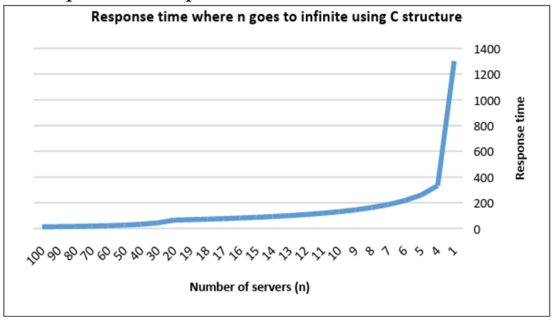

Figure 16: Shows the growth function of the results of the experiments increase the number of n in the structure C (r=1000 u=100)

As Figure 16 shows, there is a great improvement in the performance of Odoo system when using structure C (our new structure) which increases the number of "n" from 1 to 100, where the growth function has become a logarithmic function of log(n).

6. Conclusion and Future Work

The Distributed ERP was early mentioned and concerned by many types of research. Researchers used many different techniques to access the ERP system with the characteristics of the distributed systems. Some of them succeeded in achieving a part of this goal, while some wished to reach it. In this paper, we have chosen several systems according to several criteria. We chose the CockroachDB database for many reasons. Most importantly, this new version because it was built by using an open source technology, making it secured and with low-cost and high quality. Moreover, it was built based on PostgreSQL and PostgreSQL backend databases, which made it easy for us to replace it, in addition to using the Google Spanner technology, which makes it gain all the features of the distributed systems. Furthermore, we also chose it because it is a vibrant system with many features and more than 15 thousand modules with 300 new modules each month [3]. This feature makes Odoo system attractive for a lot of users in all sectors; government, private, commercial and non-profit sectors. Finally, Odoo system already has a broad audience of more than three million users [3]. All these reasons made Odoo our best choice for this project. We have proposed a new structure for system installation (distributed Odoo ERP system), and conducted a number of experiments and showed that Odoo's performance could be significantly improved when the number of "n" was increased.

We still see that we can optimize Odoo. In fact, to render a requested page from Odoo application, Odoo front-end sends several requests to the back-end to retrieve the information. Some of these requests are limited to a specific piece of information. As a result, the front-end needs many requests to present the page. Nevertheless, these multi-requests are linked to each other and can be grouped in one request, which may have a good impact on Odoo performance.

We also aspire to simplify the process of replacing CockroachDB Database in Odoo structure that we mentioned in this paper by making an installation package for Windows OS and Ubuntu OS.

References

[1] CockroachDB - For Global Cloud Services." Cockroach Labs, Odoo SA., 6 Mar. 2018, www.cockroachlabs.com/product/cockroachdb/#distributed-sql.

[2] Technical Architecture." Technical Architecture, Odoo SA., 6 Mar. 2018, oc.odoo.com/6.0/developer/12_module_development/1_server_module/. March 06, 2018

[3] Homepage." Odoo S.A., Odoo SA., 4 Mar. 2018, www.odoo.com/.

[4] Nicolas, B. Odoo Community Association, what is GeoEngine Cited 2.3.2018 http://oca.github.io/geospatial/what_is_geoengine.html

[5] Pinckaers, F., Gardiner, G. & Vossel, E. 2011. Open ERP a modern approach to integrated business management Release 6.0.0.

[6] Gartner, 75% of all ERP projects Fail but why? Cited 1.3.2018 http://officeoffinance.com/gartner-75-of-all-erp-projects-fail-but-why/

[7] ERP Systems-Popularity Ranking Open Source, Cited1.3.2018 https://erp-systems.zone/ranking/licence-open-source

[8] "Electronic News" Overview, Yesser gov SA, 9 Mar. 2018, www.yesser.gov.sa/EN/programdefinition/pages/overview.aspx.

[9] S.A., Odoo, and Serpent Consulting Services Pvt. Ltd. "Multi-DB Synchronization." Odoo S.A., Odoo S.A., 9 Mar. 2018, www.odoo.com/apps/modules/8.0/base_synchro/.

[10] "OpenERP- Base_Synchro (Synchronization)" The toolkit, Toolkit inc., 9 Mar. 2018, toolkt.com/site/openerp-base_synchrosynchronization/.

[11] BrowseInfo. "Auto Multiple Database Synchronization." Odoo S.A., BrowseInfo Odoo S.A., 9 Mar. 2018, apps.openerp.com/apps/modules/10.0/bi_base_synchro/.

[12] HallFeed12votes, Jim, et al. "Homepage." Opensource.com, 9 Mar. 2018, opensource.com/.

[13] Start a Local Cluster (Insecure)." Cockroach Labs, 9 Mar. 2018, www.cockroachlabs.com/docs/stable/start-a-local-cluster.html.

[14] Anatoly E Doroshenko and Vlad Romanenko. Object-relational mapping techniques for. net framework. In ISTA, pages 81-92, 2004.

[15] Coulouris, George; Jean Dollimore; Tim Kindberg; Gordon Blair (2011). Distributed Systems: Concepts and Design (5th Edition). Boston: Addison-Wesley. ISBN 0-132-14301-1.

[16] Magnoni, L. (2015). "Modern Messaging for Distributed Sytems." Journal of Physics: Conference Series. 608 (1): 012038. DOI:10.1088/1742-6596/608/1/012038. ISSN 1742-6596.

[17] Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33, 2 (June 2002), 51-59. DOI: https://doi.org/10.1145/564585.564601.

[18] Edgar F Codd. \A relational model of data for large shared data banks". In: Communications of the ACM 13.6 (1970), pp. 377{387.

[19] Pramod J Sadalage and Martin Fowler. NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Pearson Education, 2012.

[20] Prof. Dr. Stefan Edlich. NoSQL Databases. http://nosql-database.org/. Accessed: 2018-03-30.

[21] PC freak. What is Vertical scaling and Horizontal scaling Vertical and Horizontal hardware/ services scaling.http://www.pc-freak.net/blog/vertical-horizontal-server-services-scaling-vertical-horizontal-hardware-scaling/. Accessed: 2018-04-04.

[22] Diego Ongaro and John Ousterhout. "In search of an understandable consensus algorithm." In: 2014 USENIX

Annual Technical Conference (USENIX ATC 14). 2014, pp. 305{319.
[23] RocksDB. A persistent key-value store for fast storage environments. http://rocksdb.org/. Accessed: 2018-03-30.
[24] Tamir Duberstein Peter Mattis. SQL in CockroachDB: Mapping Table Data to Key-Value Storage. https://www.cockroachlabs.com/blog/sql-in-cockroachdb-mapping-table-data-to-key-value-storage/. Accessed: 2018-03-30.
[25] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., and Woodford, D. 2013. Spanner: Google's globally distributed database. ACM Trans. Comput. Syst. 31, 3, Article 8 (August 2013),22 pages. DOI:http://dx.doi.org/10.1145/2491245
[26] Gattiker, T.F., Goodhue, D.L.: Understanding the local-level costs and benefits of ERP through organizational information processing theory. Information & Management 41(4), 431–443 (2004)
[27] Enaya, MHD Fawaz: An Experimental Performance Comparison of NoSQL and RDBMS Data Storage Systems in the ERP System Odoo, November-2016.
[28] Knolmayer, Gerhard F. - Quality of Material Master Data and Its Effect on the Usefulness of Distributed ERP Systems - Berlin, Heidelberg SN - 978-3-540-47704-4 2006.
[29] C. (n.d.). Deploy CockroachDB On-Premises (Insecure). Retrieved April 21, 2018, from https://www.cockroachlabs.com/docs/stable/deploy-Cockroachdb-on-premises-insecure.html
[30] P. Maheshwari, Enterprise application integration using a component-based architecture, in 27th Annual International Computer Software and Applications Conference, Dallas, USA, November 03–06, 2003.
[31] D. Smith, L. O'Brien, M. Barbacci, A roadmap for enterprise integration, in 10th International Workshop on Software Technology and Engineering Practice, Montre´al, Canada,October 6–8, 2002.
[32] James Cowling and Barbara Liskov. "Granola: Low-Overhead Distributed Transaction Coordination." Proc. of USENIX ATC.2012, pp. 223–236.
[33] Alexander Shraer et al. "Dynamic Reconfiguration of Primary/ Backup Clusters." Proc. of USENIX ATC. 2012, pp. 425–438.
[34] Ashish Thusoo et al. "Hive — A Petabyte Scale Data Warehouse Using Hadoop." Proc. of ICDE. 2010, pp. 996–1005.
[35] Jeff Shute et al. "F1—The clearly Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business". Proc. of SIGMOD. May 2012, pp. 777–778.
[36] L. Frank, 'Architecture for Integration of Distributed ERP Systems and E-commerce Systems,' Industrial Management and Data Systems (IMDS), Vol. 104(5), 2004, pp 418-429.
[37] L. Frank, 'Trends in Enterprise Application Architecture, chapter 7, Architecture for Distributed ERP Systems' Industrial Management and Data Systems (IMDS), 2016, 978-3-540-32735-6.
[38] Alanne, Aki; Pekkola, Samuli; and Kähkönen, Tommi, "CENTRALIZED AND DISTRIBUTED ERP DEVELOPMENT MODELS: OPERATIONS AND CHALLENGES" (2014). PACIS 2014 Proceedings. 337
[39] Nico Brehm, Jorge Marx Gomez, and Claus Rautenstrauch. 2016. An ERP solution based on web services and peer-to-peer networks for small and medium enterprises. Int. J. Inf. Syst. Chang. Manage. 1, 1 (November 2006), 99-111. DOI=http://dx.doi.org/10.1504/IJISCM.2006.008288
[40] Ramez Elmasri and Shamkant Navathe. 2010. Fundamentals of Database Systems (6th ed.). Addison-Wesley Publishing Company, ISBN:0136086209 9780136086208, SA.
[41] O. (2008, March 13). Database Administrators Guide. Retrieved May 5, 2018, from https://docs.oracle.com/cd/B28359_01/server.111/b28310/ds_concepts001.htm#ADMIN12078
[42] Özsu, M. T., & Valduriez, P. (2011). Principles of Distributed Database Systems, Third Edition [recurso electrónico]. Estados Unidos: Springer New York

Mohamed Elmassry is a researcher and interested in open source ERP systems. He is a member of the technical committee and the administrative board of the system of planning the facilities resources for government agencies in King Abdul Aziz City for Technical Sciences. Contributing to open source technical product development communities. He has Master degree of Computer Science from King Saud University. He served as IT Director in more than one organization. He has three patents from the US and Saudi office. He has several published technical papers. He is technical consultant in several organizations.

Saad Al-Ahmadi received the MS and PhD degree in computer science from King Saud University, Saudi Arabia. He is Associate Professor in the Department of Computer Science, King Saud University. Also, he serves as part-time consultant in many public and private organizations. His current research interests include cybersecurity, IoT, machine learning for healthcare, and future generation networks.