Design and Development of Resilient Microservices Architecture for Cloud Based Applications using Hybrid Design Patterns

Mr. J. Abdul Rasheedh[1]

Ph.D Research Scholar,
Department of Computer Science, VISTAS, Pallavaram, Chennai, Tamilnadu, India,
abdul_rasheedh@yahoo.co.in

Dr. S. Saradha[2]

Research Supervisor,
Department of Computer Science, VISTAS,Pallavaram, Chennai, Tamilnadu, India,
saradha.research@gmil.com

Abstract

Microservices is a notion that claims to enable the creation of large, complicated systems that must operate in an unpredictable environment. Microservices should therefore promote resiliency. This paper provides possible solutions on fault tolerant of microservices and solve the problem of frequent shutdown or restart of services through advance research on hybrid design patterns based on Agile – Iterative and Incremental approach through a process called dynamic fusion. Fusion of objects Microservices during runtime will ensure that there is no need for system restart. Though currently there are many techniques to achieve binding, still it is widely found that the microservice needs to be restarted whenever there is a new increment or iteration. This area of research is to find out all possible and feasible options of dynamic binding via a deep study on object-oriented analysis and design techniques which comes out with various possible technical solutions. This study is based on "SOLID" software design concepts in general and how they are applied to microservices difficulties specifically to dynamic fusion of atomic microservices in a complex microservices architecture on cloud. This research work would pave a way for the future programmers to understand how to achieve dynamic binding in the real dynamic fashion as per the principles of agile approach.

***Keywords*: Microservices, Agile Iterative and Incremental approach, dynamic fusion, binding, Object Oriented Analysis, Interfaces**

1. Introduction

There has been minimal study into a thorough evaluation of dynamic binding systems, notably in terms of system failure and dependability, because previous research has concentrated on the design and implementation of dynamic binding processes. Microservices is a phrase that has been around for a while. Dr. Peter Rodgers invented the term "micro web services based on Simple Object Access Protocol (SOAP).

Microservices design helps huge systems to avoid monolithic applications. It allows for flexible coupling between cooperating systems that operate independently in diverse settings while maintaining tight cohesion. The deployment, development, and ongoing maintenance of web applications have all changed significantly with the introduction of microservice architecture. The microservice method decays the application into numerous independently executable software components or units that coherently interoperate to deliver specific application functionality, as opposed to the traditional monolithic application architecture, which builds the entire application as a single unified system. Microservice designs handles nonfunctional features such as scalability and fault tolerance for high availability. As a result of adopting microservices, programs must be built in such a way that they can withstand the failure of individual services. As services might fail at any time, it's critical to be able to identify them immediately and, if feasible, restore them automatically. Microservice applications place a high value on real-time application monitoring, which includes evaluating both technical metrics like how many requests per second the database receives and business-relevant metrics like how many orders per minute are processed. Monitoring can serve as an early warning system when anything goes wrong, prompting development teams to investigate.

Scalable systems should be able to automatically respond to changing workloads via elastic capacity management, such as that provided by cloud infrastructures. You may dynamically replicate microservices to cloud infrastructures that are under severe strain using microservice designs. It is not essential to scale the entire system, as a monolithic system would need. Small services are easier to install and, since they are self-contained, are less likely to cause system faults. During the examination of total system Microservices can help to develop resilience. Due to the independence of independent microservices, if one microservice fails (for example, due to a non-working execution environment), it can't prevent another microservice from operating. As a result, all of the other microservices provide the functional concerns they represent. As a result, a failure that affects only one microservice does not affect the functional concerns of other microservices. In systems with other structures, the impact of a failure can be more dramatic, for example, in a monolithic system, a failure can escalate and prevent the entire system from operating (no functional anxiety is met), or in a layered system, a failing layer can cause all layers (transitively) to fail if it fails. As a result of the separation of functional concerns, a microservice system may maintain objectives described by functional concerns alive even if a single microservice fails.

A microservice also reflects one of the system's functional concerns. As a result, a microservice's complexity and size are smaller than the total system. The creation of a (resilient) system is made easier by reducing complexity and size. Other techniques that split a system into subsystems, like as component-based development or layered systems, are comparable to this advantage.

The goal of this study, which is based on SOLID principles, is to eliminate dependencies so that engineers may alter one part of the software without affecting the others. They are also meant to make designs easier to comprehend, maintain, and extend. Finally, following these design principles helps software developers avoid problems and create flexible, effective, and agile software. While the principles have numerous advantages, adhering to them usually results in longer and more complicated code. This means that the design process will take longer and development will be more complex. This extra time and effort, however, is well worth it because it makes program maintenance, testing, and extension much easier. The principles have gained popularity because they lead to better code in terms of readability, maintainability, design patterns, and testability when followed correctly. In today's world, all developers should be aware of and employ these ideas. When anything has to be added, the open-closed concept states that existing, well-tested classes will need to be updated. On the other hand, changing classes, might cause issues or bugs. You want to expand the class rather than modifying it. This guideline must be followed in order to write code that is easy to maintain and revise. 1. Open for extension, which means the behavior of the class may be modified; and 2. Closed for modification, which means the source code is fixed and cannot be altered. This study looks at agile methodology, which is a form of project management technique that is mostly employed in software development and in which requests and solutions emerge from the collaborative efforts of self-organizing and cross-functional teams as well as their clients. In order to comprehend agile project management, one must first have a thorough grasp of the agile development process.

An iterative strategy is one in which the discussion's topic, stimulus, and, in some cases, technique are modified during the course of the research project. This method is especially beneficial for time-sensitive tasks where numerous rounds of study aren't feasible. The goal of this study is to better understand and propose answers for one component of the Agile-Iterative and Incremental strategy, which is object fusion without having to restart the application or system.

The following is the structure of this paper:
The corresponding microservice survey are discussed in section 2. The proposed technique based on Hybrid Design Patterns and agile methodology is discussed in section 3. The outcomes are detailed in section 4. The conclusions are stated in Section 5

2. Literature Review

The suggested methodology by Rasheedh and Saradha [1] involves binding and arranging web services based on an assessment of this competition for mobile consumers. The first addition is the introduction of the dynamic whale optimization algorithm for the new composition framework, which allows process designers to create processes based on process and business constraints. Second, this paper show how to implement agile automation in virtual space using a derived micro-service architecture that may offer agility and flexibility to the ideal automation. In addition, this paper suggest that a Queuing Deep Neural Network (QDNN) be used to assign and update service selection probability based on the average service response time that is observed.
Mane et al. [2] recommended using a thing building framework based on microservices and small-scale frontends to assist clients in receiving geographical data and information from IoT devices in a consistent and

acceptable manner as well as a Progressive Web Application (PWA). They present a condition in which the web application is persistently developed by joining geospatial data, information guaranteed about from IoT sensors, and other relating information, and they handle the procedure of part structures utilizing a solicitation design including piece marks and other illustrative properties, similarity sharp data about the application client.

Ying et al. [3] presented a Dynamic QoS requirement for affiliations, which is currently a hypnotizing issue and a test for examining in the domains of affiliation suggestion and construction. Yang et al. [4] investigated SOLID difficulties with Java programming bolstered by UML outlines and dynamic execution suggestions of program synchronization in an electronic programming environment. It implies assisting understudies in bettering their understanding of the static structure and dynamic direct of java programs, as far as object-driven system exams. Through quantitative and energetic assessments, the evaluation prospective outcomes of jaguar code to look at its abundance and client endurance.

Granchelli et al.[5] show the first version of MicroART, our Architecture Recovery Tool for microservice-based systems. MicroART, which is built on Model-Driven Engineering concepts, can build models of a microservice-based system's software architecture that can be maintained by software architects for a variety of reasons. M3 is a revolutionary solution for efficient and effective monitoring of applications based on multi-virtualization (containers/VMs) multi-microservices deployed in multi-cloud settings.

Noor et al [6]. The suggested solution allows users to monitor and report on the performance of microservices running within containers and virtual machines in real time. To meet the challenges of monitoring such sophisticated services-based applications, the solution leverages an agent-based architecture to scale from a centralized to a decentralized architecture. The suggested system was put to the test in a variety of situations, with the results proving M3's usefulness in monitoring microservices in multi-virtualization multi-cloud systems. At otto.de, one of Europe's largest e-commerce platforms.

Hasselbring and Steinacker[7] show how the qualities of microservice architectures help with scalability, agility, and dependability. At otto.de, this article examines vertical deconstruction into self-contained systems, proper granularity of microservices, as well as microservice coupling, integration, scalability, and monitoring. High dependability is provided by automated quality assurance with continuous integration and deployment, while improving agility to more than 500 live deployments each week. To facilitate seamless and speedy integration of domain services.

Guo et al. [8] offer an interactive crossover service fusion technique based on microservice architecture. This technique provides a comprehensive solution for speedy crossover service integration and aids in the resolution of semantic discrepancies in service integration, hence supporting the growth of the service ecosystem. Breaking apart a monolithic system, according to Knoche [9], takes careful planning and can have a significant influence on performance. When breaking down a monolithic system into microservices, Knoche provides an eight-step gradual strategy for sustaining performance. The method contains a performance simulation to assess what kind of performance can be expected when a microservice-based architecture is implemented. Balalaie et al. [10] discuss their experiences and lessons gained when transitioning from a service to a microservice architecture step by step. They describe how they transitioned from a monolithic pipeline to a microservices pipeline with discrete services that can be deployed individually. Moving from monolithic to microservices opens up the possibility of forming many cross-functional teams to support DevOps. To analyze the "resilience" of systems in quantitative terms, according to Strigini [11], it is required to first determine what "resilience" means, whether it is a single quality or multiple, and which metric or measurements best characterize it. The different possible measures for the attributes that the word designates, with their different pros and cons in terms of ease of empirical assessment and suitability for supporting prediction and decision making. The different possible measures for the attributes that the word designates, with their different pros and cons in terms of ease of empirical assessment and suitability for supporting prediction and decision making. The commonality of these ideas, metrics, and associated difficulties across other domains of engineering, as well as how learning may be spread between them. The issues posed by the requirement to design a scalable and fault-tolerant system based on microservices are addressed by Baboi et.al[12]. This experiment considers two types of microservices such as simple and extended, and the proposed method shows to be novel, particularly in terms of dynamic behavior. The implementation of microservices systems is more productive, adaptable, and cost efficient thanks to cloud-native designs [13]. Nonetheless, Zimmermann points out that microservices are a delicate issue that is being researched by both academics [14] and industry.

The word "microservices" was originally used in May 2011 during a workshop of Software Architects in Italy to characterize what the attendees viewed as a similar architectural style that many of them had recently explored.

A year later, the same panel affirmed that the word "microservices" is correct. Microservices were created in response to issues with monolithic systems or service-oriented architectures that made complexity, scalability, and dependencies of the application under construction difficult to manage, all while employing lightweight communication protocols [15-19].

3. Research Methodology

3.1 PROPOSED SYSTEM MODELS -Agile Iterative and incremental approach

Iterative and incremental software development is a technique for creating software that is based on a cyclical release and upgrade pattern and a steady rise in feature additions. It is one of the Agile software development approaches, as well as a rational unified process and extreme programming.

Agile methodology: is a type of project management process, mainly used for software development, where demands and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customers.

The Agile Process It is required for one to have a good understanding of the agile development process in order to understand agile project management.

Iterative approach: An iterative approach is one where the content of the discussion, stimulus, or sometimes even the methodology is adapted over the course of the research programme. This approach is particularly useful for time-sensitive projects where there isn't scope for multiple rounds of research.

Iterative and incremental approach:

Iterative and incremental software development is a method of software development that is modelled around a gradual increase in feature additions and a cyclical release and upgrade pattern. It is one of the methodologies of Agile software development, rational unified process and extreme programming.

3.2 WORKFLOW OF THE PROPOSED SYSTEM

The figure 1 shows the proposed system working flow of services from Client, Interfaces, and Fusion Application Server to Micro services. The solution is based on "SOLID" software design concepts and how they apply to microservices issues, namely dynamic fusing of atomic microservices in complicated cloud Microservices architecture.

In the proposed system in figure 1, shows the micro services may be bill pay, hello services etc are provided according to the needs of the user/client by reducing their complexity and traffic time. This system provides many services in general; also if the client needs any services then it can be created and provided by means of workflow doc. These services can be fused together in the application server according to the user needs. This technique is nothing but the fusion made micro services where the client meets their requirements easily without any traffic time.

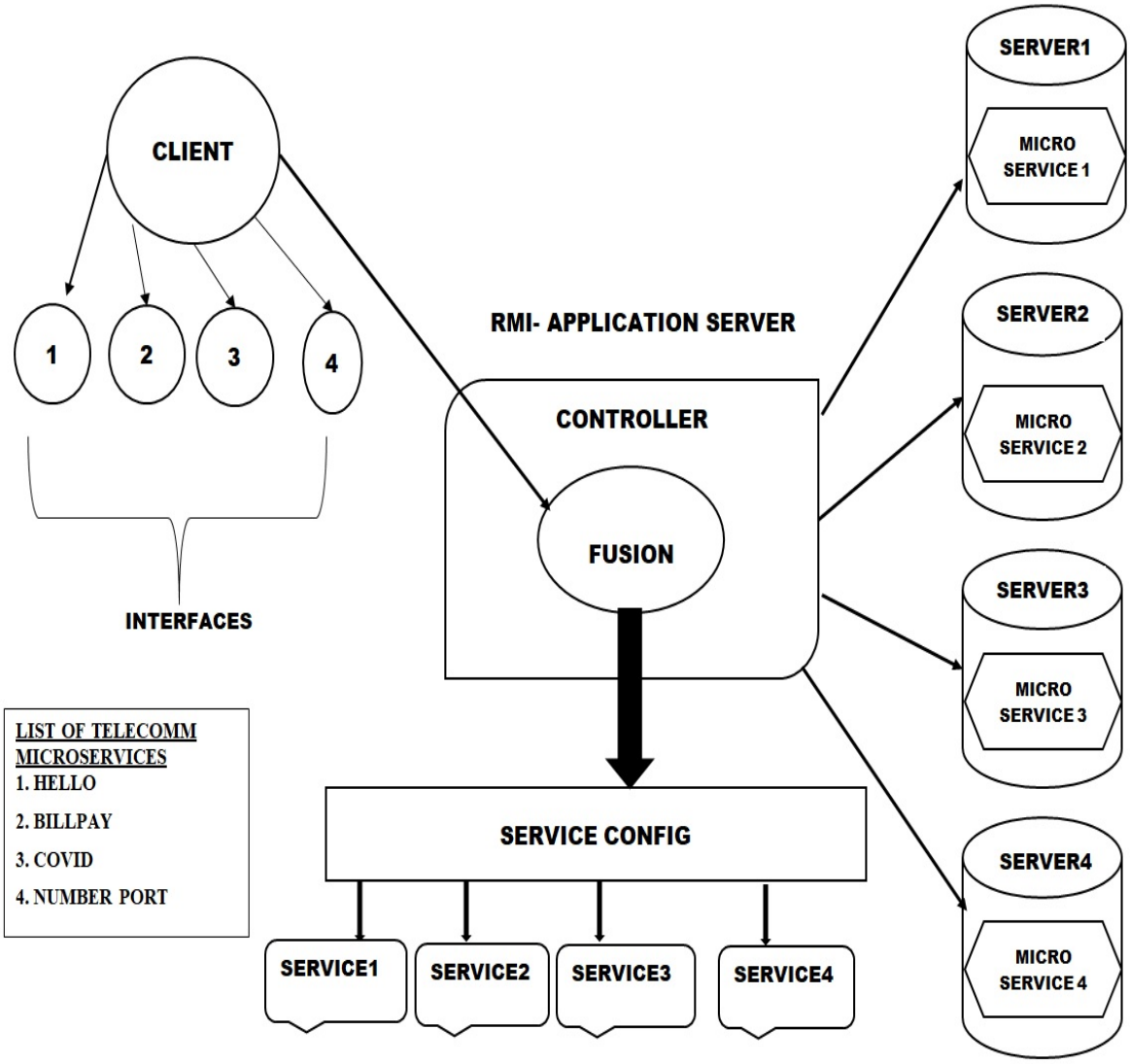


Fig 1: Telecomm microservices –Working Flow Diagram

3.3 MICROSERVICES DYNAMIC FUSION USING HYBRID DESIGN PATTERN

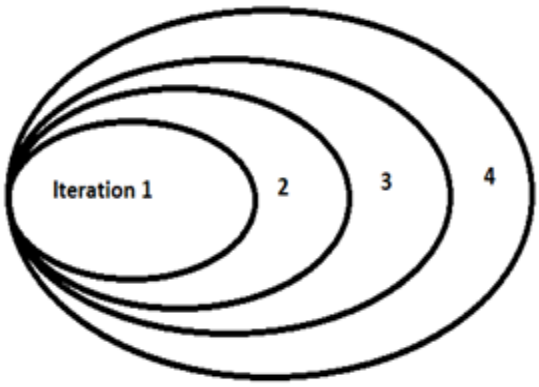


Fig 2: Proposed microservices Dynamic Fusion using Hybrid Design Pattern

3.3.1 This hybrid approach proposed is to have the following features

a) When a fusion happens in a mother object, the mother object undergoes a seamless transformation from existing to a new system.

b) The transformation should be at runtime to ensure that there is no system restart.

This dynamic fusion using hybrid design pattern shown in figure 2 work on the SOLID principle.

The proposed design pattern apply the SOLID software design principles per service with this extreme approach, where each service is totally isolated, even at the level of its codebase. Because this system don't want to share a codebase, which will consider each microservice as a standalone application. It will have its own interfaces, class structure, and dependencies as a result.

Each letter in the word "SOLID" represents one of the following principles:

S - Single Responsibility
O - Open/Closed
L - Liskov Substitution
I - Interface Segregation
D - Dependency Inversion

3.3.2 The Single Responsibility Principle:

The single responsibility principle states that a class should have a single reason to change.

Many people have added their own understanding of this principle, which translates to “a class should only perform a single task”.

3.3.3 The Open/Closed Principle:

The SOLID concept of "Open/Closed" asserts that software entities should be open for extension but closed for change. This suggested hybrid design dynamic fusion is based on this principle.

3.3.4 The Liskov Substitution Principle:

The Liskov Substitution Principle states that if objects of type T are replaced with objects of type S, when S is a subtype of T, in a program, the program’s characteristics should not change (e.g. correctness).

3.3.5 The Interface Segregation Principle:

This principle sounds so simple, yet is so powerful:

An object should only depend on interfaces it requires and should not be enforced to implement any method (or property) it doesn’t require.

“Breaking down” interface described behaviors allows us to improve our codes readability, testability, and maintainability.

3.3.6 The Dependency Inversion Principle:

The principle of Dependency inversion is described as follows:

A. Higher-level modules should not depend on low-level modules. Both should depend on abstractions.

B. Abstractions should not depend on details. Details should depend on abstractions.

Interesting, right?

Again, it seems rather intuitive, nowadays, as we got used to “connecting” our components using interfaces.

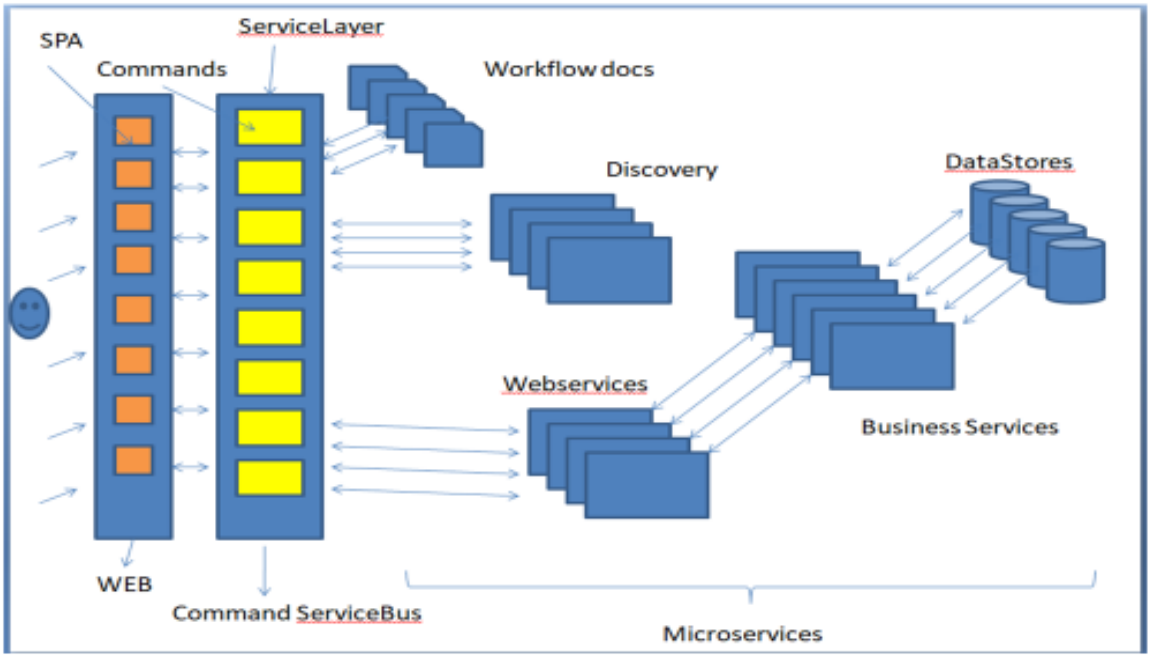
However, many beginner-level developers make this mistake again and again, as (apparently) our human mind tends to think hierarchically.

The following “bad” example is not that uncommon:

By extracting the “contract” between each of the dependent entities, we’ve decoupled each entity from its former dependencies. A contract is now being shared between each entity and its dependency.

3.4 SCHEMATIC DESIGN OF PROPOSED SYTEM

The figure 3 displays the schematic diagram of the microservices using dynamic object binding. The system consist of components such as SPA, Web, Commands, Service layer, Command Services Bus, Workflow docs, web services, business services, data stores, microservices. Each session is explained in detail below:



F

Fig 3: Schematic Diagram Microservices Using Dynamic Object Binding

i) COMMANDS

It is the secret secured code which is selected/discovered by the user depending on their application what should they do actually.

ii) COMMAND SERVICES BUS

It consists of several commands which act as the communicating link between discover, web services and business services.

iii) WEB SERVICES

This will provide the gateway for the production of the services according to the needs of the user in the web layer.

iv) DISCOVERY SERVICES

These services will bring the end product to the user through advertising about the services provided by the system

v) BUSINESS SERVICES

This service works as the gateway for the enterprises. It is similar to marketing of the end product.

vi) SINGLE PAGE APPLICATION (SPA)

SPA where the service are selected by the user according to their needs.

vii) MICROSERVICES

A microservice should be deployable on its own. Our software system can be installed in its entirety, although this is not required. While the system is functioning, services (microservices) can be added (and withdrawn). If a microservice has dependents, they should be deployed as well (e.g. database, 3rd-Party components). Although the complexity of microservices is significant, it may be lowered by making the services transparent, i.e. concealing the system's codes.

3.5 ALGORITHM OF PROPOSED SYSTEM

The propose system algorithm have four different phases. They are 1) Interface 2) Base Object 3) Molecular framework 4) Mixing engine

i. INTERFACE CLASS

It is nothing but simply a role. In other words just identifying a way or creating route or making a role to do work is known as interface or role.

ii. IMPLEMENTATION CLASS

It is nothing but explaining how to do that work or schematic workflow of the process is known as implementation class.

In our proposed model the implementation class is not given to user instead we identify their user needs and provide them the end product alone this is main unique feature of the system.

Actually the figure explains that the user decides what role he has to do? After that he will start implementing those interfaces into end product. Then the user will approach the business services man where the business

services man study about the interfaces, understand the concept of it and decide to be sale/market the end product. Then the molecular framework will provide the legal agreement between the user and business services. The business services sale the product through micro services platform in the web.

iii. MOLECULAR FRAMEWORK

This is the art of the architecture. It is the legal agreement or tie up details or handshakes between user/client and business services. Mostly in the existing system if any service is to be added newly then the entire system should be shutdown or stop the web running time. So in order to overcome this proposed system use the methodology of micro services molecular framework in which the system run continuously without any break during addition of any new services. There is no need to close the entire system and restart or reboot the system again. The new service will be added along with the run time of the system. Thus it helps to expand the system without any shutdown or loss of old data. The algorithm of the proposed system is as follows:

Phase 1: Interface

The phase one is interface in order words it is defined as roles. To start with the services first we need to identify the role of those particular services. This roles/ interface will associate with behaviours of the client needs which in turn result in implementation process of the service.

Step 1: Create/Identify Interfaces / Roles
Step 2: Associate Behaviours to Interface
Step 3: Provide Implementation for Interfaces/Roles

Phase 2: Base Object

The phase two is base object. In this phase according to the user role, the lookup table is created for the base object and that object is implemented

Step 1: Create/Lookup/Retrieve the Base Object
Step 2: Set Interface to Molecules Framework
Step 3: Pass the General object and the implementation object of the set Interface to Molecule Framework.
Step 4: Retrieve the fused object from molecular framework in return.
Step 5: Cast the Object with corresponding interface Invoke the method of the interface on the retrieved fused object.

Phase 3: Molecule Framework

The phase three molecular framework, in which the interface from the base object lookup table are accepted such that the first object will be mother object and other will be implementation object. So it will create a newProxyInstance object. Thus the new object will be fused to mixing engine.

Step 1: Accepts the Interface/interfaces and set the interface/interfaces in a collection.
Step 2: In the getObject object its accepts an array objects - first object will be the mother/base object and other objects will be implementation objects.
Step 3: Creates an object of Proxy class by calling 'newProxyInstance' method.
Step 4: Passes the class loader of the base object, the interfaces class and the objects to be fused to the MIXING Engine.
Step 5: In return the getObject method return a fused object.

Phase 4: Mixing Engine

The final phase is mixing engine which implements the invocation handler, accepts the objects and store in the object array using reflection method

Step 1: Mixing engine implements Invocation Handler interface of Java
Step 2: Mixing engine accepts all the objects which need to be fused and store it in an object array.
Step 3: Overrides the invoke method of the invocation Handler.
Step 4: The invoke method gets called whenever any method is called on the fused object returned by the molecular frameworks getobject method (note: molecule framework as already passed the objects to the invocation handler in Mixing engine).
Step 5: The invoke method searches the method called in all the objects stored in the mixing engine, based on the availability it invoked the method using reflection process

3.6 Software Specification

Tools : **Eureka** server
Framework : Spring boot
Communication Tools : Spring cloud
Language : Java
Architecture : Microservices

i. SPRING BOOT

Spring Boot makes it simple to develop a Spring application, whereas Spring Cloud gives you a collection of tools to make communication between microservices go more smoothly.

ii. EUREKA SERVER

Eureka Server is a program that keeps track of all of the client-service apps. Each Micro service will register with Eureka, and Eureka will be aware of all client applications operating on each port and IP address. Discovery Server is another name for Eureka Server. Eureka server usage: Eureka is a Representational State Transfer (REST)-based service that is largely used in the AWS cloud to locate services for load balancing and middle-tier server failover.

4. Results and Discussion

This research work presents an evaluation of eight major problems in software development where the proposed fusion based Agile iterative model is quite a flexible method which allows changes to be made in the project development requirements even if the initial planning has been completed than in traditional acquisition programs. Estimating costs in a proposed Agile environment requires a more iterative, integrated, and collaborative approach than in existing waterfall model. Contrary to the myth that Agile is an undisciplined approach that downplays cost aspects, cost estimation is a critical activity in programs that use Agile practices. The proposed Agile development may lead to a net decrease in total life cycle costs, especially because the largest area of cost decrease is in sustainment. The cost estimate for integration and testing must consider the impact of the short, frequent, integrated real-time integration and testing characteristics of Agile developments, and determine if the costs differ from those for a traditional waterfall approach. Integrating and testing large batches of functionality, as in traditional waterfall approaches, is inherently more complex than performing these tasks on small batches of functionality, and developing testing techniques and detecting errors during tests are more difficult on larger batches of software. Frequent testing also allows for earlier detection of errors, when they are easier and less costly to fix than fixing those same errors found in tests performed at the end of development. An Agile release deploys new capabilities in short, frequent releases (typically every 6–12 months). The degree of change and level of complexity of each release may require further consideration of deployment costs. The proposed Agile iterative development model depends on a high degree of automated deployment usage to deliver short, frequent releases of software effectively. Table.1 illustrates analysis of cost benefit based on effort of 100 days average between existing waterfall model and fusion based agile iterative model. Agile methodology works exceptionally well with Time & Materials or non-fixed funding. It may increase stress in fixed-price scenarios. The Agile team members are interchangeable, as a result, they work faster. There is also no need for project managers because the projects are managed by the entire team.

Major problem in software improvement	**Effort(100 days average)**	
	Existing method (waterfall model)	**Fusion method(Agile iterative model)**
Understanding Deliverables	20	20
Understanding Existing code	20	0
Developing New Code	30	30
Testing	3	3
Integration of new code with existing code	20	19
Integration Testing	5	4
Deployment	1	1
Go Live	1	1
NO.OF DAYS	100	78

Table 1: Cost Benefit Analysis

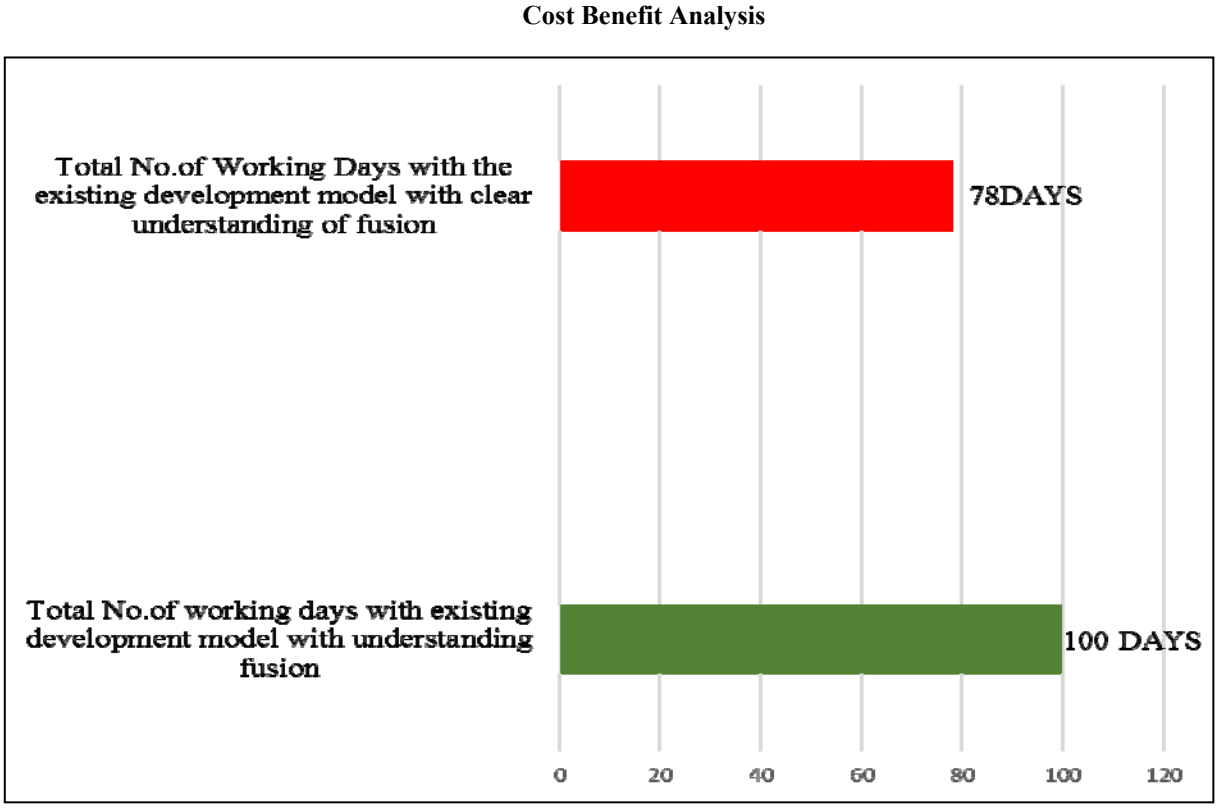


Fig 4: Graphical representation of cost benefit analysis between waterfall and Agile iterative model

Figure.4 illustrates total number of working days with the proposed agile fusion model based on understanding fusion takes 78 days whereas total number of working days with the existing waterfall model takes 100 days.

4.1 Performance analysis

Our study environment will imitate a whole system, including a user interface, REST API, business logic, and database. The many functions and services in this system will be explained in the sections that follow. In Scala [16], the main key is created automatically using the UUID Java class [17]. After that, the data will be kept in Cassandra [18], our system storage technology of choice. For this experiment, the following gear was used: 2x Intel NUC5i7RYH with 16GB RAM, an Intel Core i7-5557U processor, and Ubuntu Server 16.04 as the host operating system. JMeter was ran on a computer connected to the same local network as the other devices, which was running Windows 10 Pro with an Intel Core i54690K CPU and 16GB RAM.

4.2 Performance of CPU Usage

The CPU usage of the proposed hybrid microservice is less while compared with monolithic and Eureka-Zuul microservice. Therefore, the memory usage will be less in the case of proposed agile microservice are illustrated in table 2 and figure 5

Service Application Type	Average CPU usage (%)
Monolithic (PostgreSQL)	53.2
Eureka-Zuul_Microservice	45.7
Proposed Agile_Microservice	42.1

Table 2: Average CPU Usage of various service applications

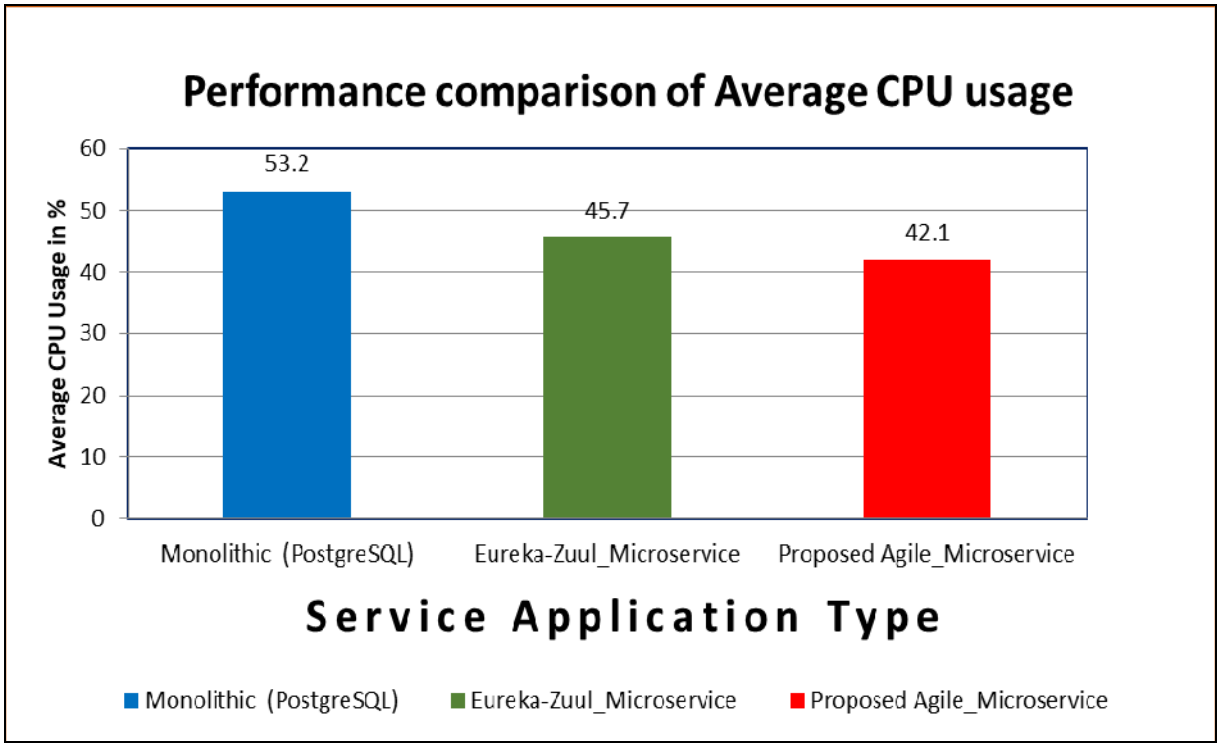


Fig 5: Performance comparison of Average CPU Usage

4.3 Throughput performance

The request per second of the various users from 1, 10, 20, 40, 50 are considered for this experimental research. The throughput is evaluated in term of request done successfully per second whereas the proposed agile microservice is 27 for 1 user case that is higher while compared with other monolithic and Eureka-Zuul microservice as 20 and 24 respectively. Similarly, the successful throughputs are high for proposed agile microservice compared with existing monolithic and microservice applications in the case of 10, 20, 40 and 50 are shown in table.3 and figure 6.

No of Users	Monolithic _PostgreSQL	Eureka-Zuul_Microservice	Proposed Agile_Microservice
1	20	24	27
10	36	39	41
20	45	47	50
40	60	65	67
50	62	66	68

Table 3: performance Comparison of Successful Throughput

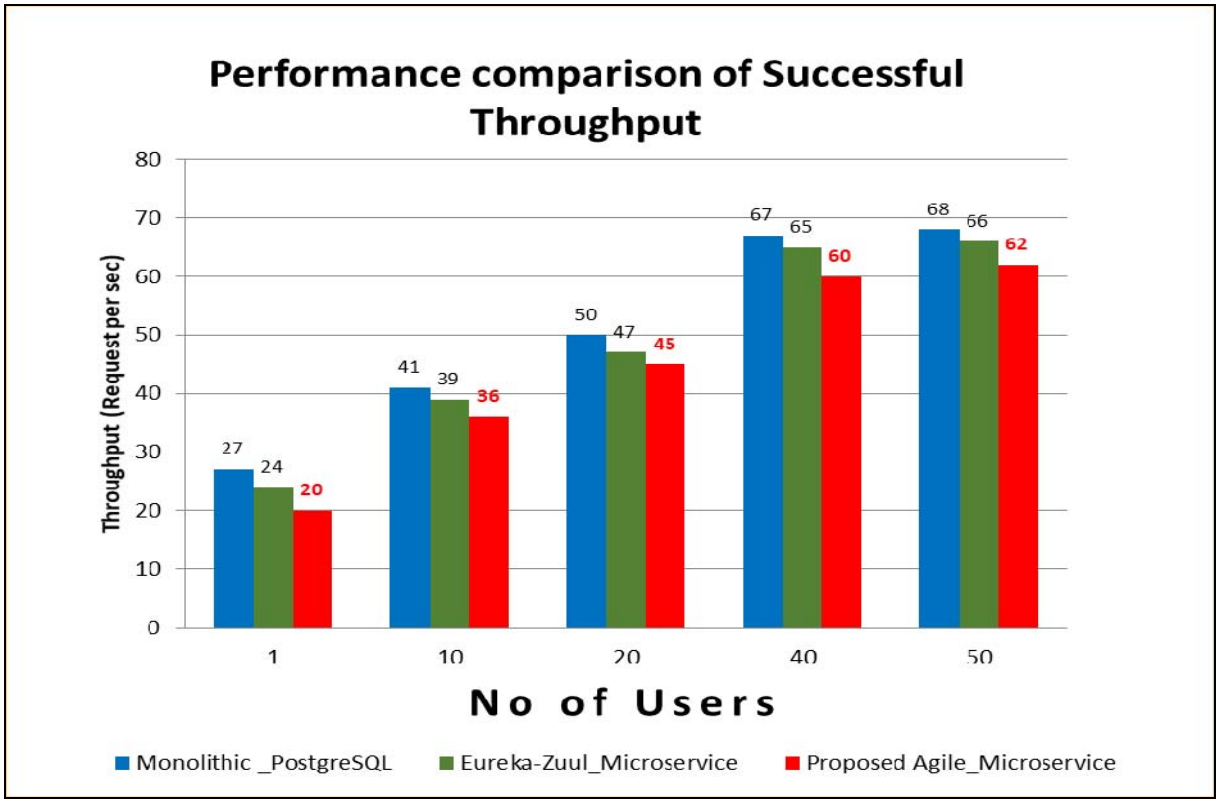


Fig 6: Performance comparison of successful throughput

4.4 Performance of average error rate

The average error occurs for the various users from 1, 10, 50, 100, 200 are considered for this experimental research. The average error rate is evaluated in term of error occurred per user whereas the proposed agile microservice is 1% for 1 user case that is less while compared with other monolithic as 2% but in the case of Eureka-Zuul microservice also obtained with 1%. Similarly, the average error rates are less in proposed agile microservice compared with existing monolithic and microservice applications in the case of 10, 50, 100 and 200 are shown in table 4 and figure 7.

No of Users	Monolithic _PostgreSQL	Eureka-Zuul_Microservice	Proposed Agile_Microservice
1	2	1	1
10	10	8	7
50	13	12	10
100	15	13	12
200	18	15	13

Table 4: Average error rate of various service applications

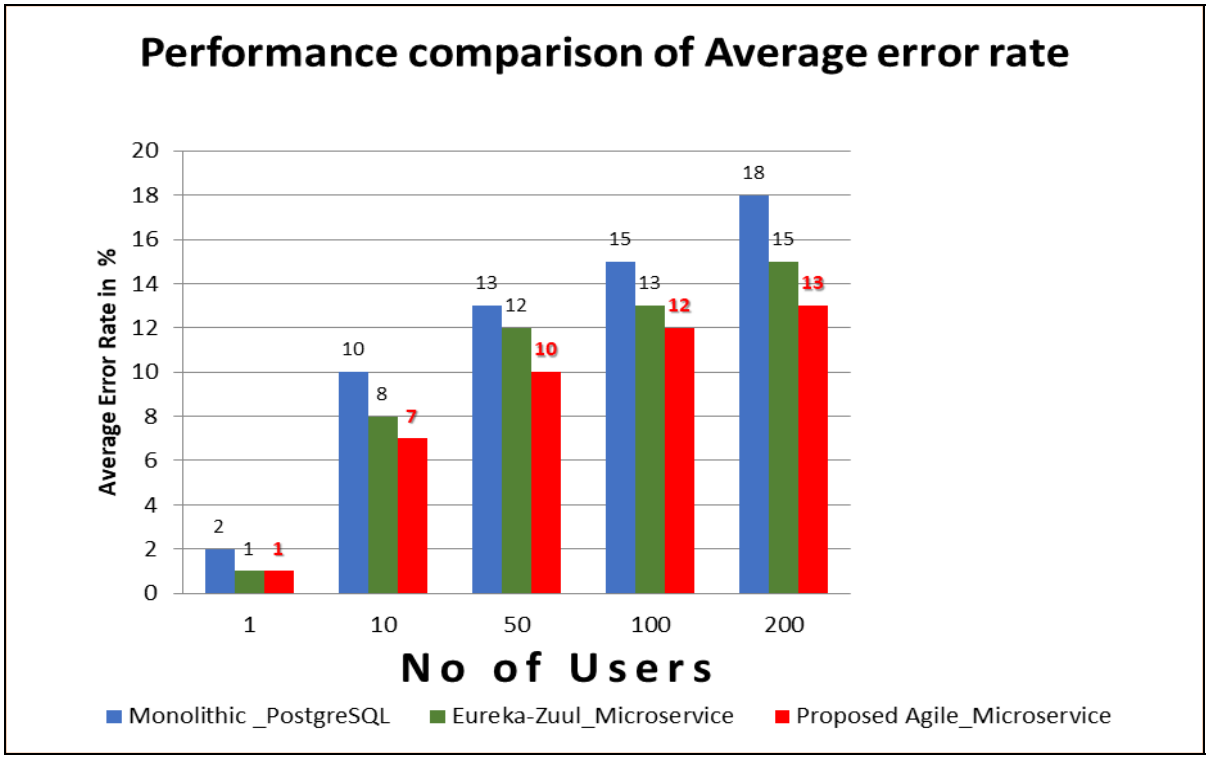


Fig 7: Performance comparison of average error rate

5. Conclusion

The proposed hybrid approach allows for the creation of new microservices in a very simple way, which are then automatically integrated into the built system. The innovative elements of this architecture are: it can scale very easily, each new client being assigned a task by the server according to the strategy pursued. The server handles the dynamic distribution of tasks between clients, providing a dynamic fusion to ensure that there is no system restart as per the principles of Agile methodology. The SOLID software design principles are the code abstractions which were used to implement our objects whether it's classed for an object template or functions in which had written a specific behavior of a fusion of objects without the system restart. This research will help especially mission critical services and applications which guarantees no downtime and 24/7 support on all 365 days. If we can achieve real dynamic binding, then there won't be any service downtime nor maintenance down time, this way we can achieve smiles on millions of faces throughout the globe.

The total number of working days for the proposed agile fusion model based on understanding fusion takes 78 days whereas total number of working days for the existing waterfall model takes 100 days. The proposed fusion based Agile iterative model is quite a flexible, faster, iterative method which allows changes to be made in the project development requirements even if the initial planning has been completed than waterfall model. The utilization of CPU and error rate in the proposed hybrid microservice is less compared to the existing microservices. The throughput is evaluated in term of request done successfully per second whereas the proposed agile microservice is 27 for 1 user case that is higher while compared with other monolithic and Eureka-Zuul microservice. This research also focuses on how re-usable components can be integrated, there by bringing a new concept called "Programming Code Inventory", which in turn can bring smiles on every face in the software Industry.

References

[1] Abdul Rasheedh and Dr. S. Saradha, "An Optimal Micro-Services Architecture for Runtime Dynamic Binding in Web Services Using Hybrid Heuristic Algorithms", International Journal of Advanced Science and Technology, Vol. 29, No. 8s, 2020, pp. 3550-3564.
[2] Mena, M., Corral, A., Iribarne, L. and Criado, J., "A progressive web application based on microservices combining geospatial data and IoT", IEEE access, Vol-7, pp-104577-104590.
[3] Jin, Y., Guo,W and Zhang, Y., "A time- aware dynamic service quality prediction approach for services", Tsinghua science and technology, 2019, Vol-25, issues-2,pp-227-238.
[4] Yang, J., Lee, Y and Chang, K.H., "Evaluations of JaguarCode: A web-based object-oriented programming environment with static and dynamic visualization", Journal of system and software, 2018, vol-145, pp-147-163.
[5] Giona Granchelli, Mario Cardarelli and Paolo Di Francesco, "MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-based Systems", IEEE, 2017.

[6] Ayman Noor, Devki Nandan Jha, Karan Mitra, Prem Prakash Jayaraman, Arthur Souza and Rajiv Ranjan, "A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments", IEEE, 2019.
[7] Wilhelm Hasselbring and Guido Steinacker, "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce", IEEE, 2017.
[8] Siying Guo, Chao Xu, Shizhan Chen, Xiao Xue, Zhiyong Feng and Shiping Chen, "Crossover Service Fusion Approach Based on Microservice Architecture", 2019, IEEE.
[9] Holger Knoche, "Sustaining runtime performance while incrementally modernizing transactional monolithic software towards microservices", In Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering, ICPE '16, pages 121–124, New York, NY, USA, 2016. ACM.
[10] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture", May 2016.
[11] Strigini, L, "Fault tolerance and resilience: meanings, measures and assessment", In: Wolter, K., Avritzer, A., Vieira, M. and van Moorsel, A. (Eds.), Resilience Assessment and Evaluation of Computing Systems,2012, Berlin, Germany: Springer.
[12] Mihai Baboi, Adrian Iftene and Daniela Gîfu, " Dynamic Microservices to Create Scalable and Fault Tolerance Architecture", International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, 2019, Vol-159, pp-1035-1044.
[13] Xia, C., Zhang, Y., Wang, L, Coleman, S., and Liu, Y, "Microservice-based cloud robotics system for intelligent space", In: Robotics and Autonomous Systems 110, 2018, DOI: 10.1016/j.robot.2018.10.001.
[14] Bogner, J., Fritzsch, J., Wagner, S., and Zimmermann, A, "Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality". At the 2019 IEEE International Conference on Software Architecture Workshops (ICSAW) At: Hamburg, Germany.
[15] Akentev, E., Tchitchigin, A., Safina, L., and Mzzara, M, "Verified type checker for Jolie programming language",2017, https://arXiv.org/pdf/1703.05186.pdf.
[16] Černý, T., Donahoo, M.J., and Trnka, M, "Contextual understanding of microservice architecture: current and future directions". ACM SIGAPP Applied Computing Review, 2018, Vol-17, issues-4, pp- 29-45, DOI: 10.1145/3183628.3183631.
[17] The Scala Programming Language. https://www.scala-lang.org/. Accessed: 2017-05-19.
[18] UUID (Java Platform SE 8). https://docs.oracle.com/javase/8/docs/api/java/util/UUID.html. Accessed: 2017-05-19.
[19] Apache Cassandra. http://cassandra.apache.org/. Accessed: 2017-05-19.

Mr. J.Abdul Rasheedh MCA in 2004 at University of Madras, Chennai and at present working as an **Assistant Professor in the P.G.Department of Computer Science, The New College, Chennai.** He is currently pursuing PhD in Computer Science at VISTAS, Pallavaram. His Area of interest and research includes Cloud Computing and Microservices. He has been actively taken part and presented and published various papers in International and National Conferences in his research area.
SCOPUS ID: https://www.scopus.com/authid/detail.uri?authorId=57220899979
ORCID ID: https://orcid.org/0000-0003-3460-290X
VIDWAN ID: https://vidwan.inflibnet.ac.in/profile/253606

Dr.S.Saradha, working as a Assistant Professor in the Department of Computer Science, Vels Institute of Science, Technology and Advanced studies (VISTAS), Pallavaram, Chennai. She has more than 10 years of experience in Educational Institute. Her area of research includes Data Mining, Artificial Neural Networks, Machine Learning, Big Data Analytics. She has published more than 17 Research Papers in National and International journals. She is interested in writing textbooks for the students to make them understand any concept in an easy way. She is a recognised supervisor, guiding M.Phil and PhD scholars.