Comparative Analysis of *Dependable Storage in the Intercloud* by IBM and *Fault Tolerance Management in Cloud Computing: A System-Level Perspective* by Ravi Jhavar In The Context Of Ensuring System Dependability and Security In Cloud Computing

Abstract

Cloud computing is an attractive alternative to traditional information processing systems which offers improvements in system functionality, performance, and cost. While cloud computing has been adopted by many large enterprises, it has not reached the expected implementation rates among small and medium businesses as the choice of computing system with respect to the improvements it offers (Mujinga). The reason behind this relatively low market proliferation has been determined to be system dependability and security concerns caused by unique features of cloud computing. To address these concerns, IBM and Jhavar have offered differing innovative solutions in their papers. This paper summarises the findings of IBM and Jhavar, analyses the similarities and differences between these two solutions and discusses how these approaches can help address problems inherent to the development of other dependable and critical systems.

Introduction

Computing systems have been essential in the daily operation of many enterprises since the 1960s. These "computing systems can be characterised by fundamental properties such as functionality, performance, dependability, security and cost" (Avizienis). The type of computing system a business will use depends on the specific needs of that business. In addition, enterprises might transition to new types of computing systems if emerging technologies deliver sufficient improvements in any mentioned fundamental property as they will help a business stay competitive in their market.

One such emerging computing system type has been cloud computing which was enabled with the widespread use of the internet in the early 21st century. From a business point of view, transitioning to cloud computing offers significant improvements in functionality, performance, and cost through the delivery of more computing power, more storage, and faster network resources over the internet (Cachin; What Is IaaS? -AWS). More importantly, cloud computing offers flexibility in obtaining and releasing computing resources through the pay-as-you-go model which is crucial for scalability concerns (Jhawar).

From a technical point of view, however, new technologies present their own new and often unique set of challenges. Specifically, issues surrounding dependability and security of a system can counteract any benefits that businesses might gain by adopting newer technologies. Moreover, these issues can actually be detrimental for the success of enterprises. This is because most systems that are deployed with a cloud computing architecture will be business critical systems where failures may result in high economic losses.

Business critical systems, and critical systems overall imply there is a need for the system to utilise specific techniques and methods that will aim to achieve system dependability and security.

The concept of system dependability encompasses the concepts of system availability, reliability, safety, integrity and maintainability. In addition, system security is related and a prerequisite to system dependability that is composed of confidentiality, integrity, and availability (Avizienis).

In their paper, IBM identifies the root cause of dependability and security issues of cloud computing systems as having a system infrastructure that has a single point of failure and presents a solution to attain system dependability and security through the implementation of a two stage cloud computing stack where an additional "intercloud" layer is put on top of the already existing single-domain cloud (SDC) layer. This layer is a complementary addition to the already existing SDC layer which mitigates the issue of single point of failure. On the other hand, Jhavar argues that the root cause of the dependability and security issues of cloud computing systems arise from the high system complexity and the abstraction layers of cloud computing which makes it impossible for its users to understand the underlying infrastructure of cloud computing. Without this understanding, Jhavar states it is not possible for businesses to come up with a dependable and secure design for their systems. To solve this issue, Jhaver advocates for the implementation of a new type of service in the cloud computing stack that will enable cloud computing systems to be dependable and secure by offering systems the algorithmic implementation of different fault tolerance properties depending on what system dependency and security needs enterprises specify for their system.

This paper will summarise the findings of IBM and Jhavar, analyse the similarities and differences between these two solutions and discuss how these approaches can help address problems inherent to the development of other dependable and critical systems through evaluation real-life scenarios.

Current Cloud Computing Infrastructure's Limitations

In their papers, IBM and Jhavar discuss the current cloud computing system's infrastructure and identify its limitations in order to address them with their respective solutions.

IBM notes that the SDC solution encompasses most of the cloud computing systems operational today such as Amazon AWS. The SDC infrastructure is optimised for the deployment of applications that have a very large number of users and has high availability needs through provider-centric distributed protocols such as internal load balancing protocols that aim to optimise the delivery of resources of a single cloud service provider's (CSP) infrastructure (Cachin).

IBM notes that while there are increasing efforts to build a dependable and secure SDC computing system where multiple users can safely and dependably can make use of the same computational resources of a single cloud service provider, there are unavoidable limitations of the SDC infrastructure since all trust in the system is only as much as the trust in the CSP. If the CSP can not be trusted, data and computation integrity can not be guaranteed. Even if the CSP is trusted and has a good reputation, IBM argues that the SDC infrastructure is prone to a single point of failure. As an example, IBM recognizes network connectivity for CSPs as a single point of failure who do not diversify their services geographically which is especially problematic in developing countries due to less stable network infrastructure. In such systems, a network outage can cause applications that are hosted on the CSPs virtual machines to experience system availability, reliability, and consistency issues even if every other system is working perfectly.

Moreover, IBM demonstrates system security issues for applications deployed on a SDC through a key management scenario. In applications that use data encryption to tackle their security threat model, key management is fundamental to ensuring system security. Such SDC applications would be required to store their keys on a storage service also offered by the same CSP in the scenario where local key management isn't possible due to data recovery concerns. This defeats the purpose of encryption which casts doubt on data confidentiality and consequently system security.

Lastly, IBM points out that the eventual consistency guarantee offered by many cloud providers which comes as a result of having a highly available design in the SDC solution may not be sufficient as a fault tolerance policy for some applications. This would incentivize businesses that have high data consistency requirements to locally store data in order to achieve better consistency which defeats the purpose of outsourcing computational power from a CSP. To summarise, IBM declares having a cloud computing system where the cloud service is provided by a single provider introduces internal system vulnerabilities that can jeopardise system dependability, security, and lead to system failures.

On the other hand, Jhavar recognizes the stale nature of the SDC solution as the primary threat for attaining system dependability and security. Specifically, Jhavar focuses on the inability of the SDC solution to handle the agile nature of an application's life cycle and its changing system dependability and security requirements through a motivating scenario.

In this scenario, Jhavar hypotheses a case where a banking company deploys its banking service as a multi-tiered app that is hosted on a SDC service. The first tier, application tier, uses the CSP's computing service to process customer input and the second tier, data tier, uses CSP's storage service to store customer data. This tiered design optimises for scalability and elasticity needs of a large banking company. However, in the case of a failure, Jhavar notes the extent of negative affect the bank would experience will be different depending on the tier that fails. This implies there are differing fault tolerance needs of these two different tiers. However, this differing requirements of system dependability within a system can not be satisfied through the SDC solution as the SDC services today offer a single constant system dependability service.

Adding Redundancy To Cloud Computing Systems

Jhaver states that the most widely adopted strategy to tolerate failures in a system is based on adding redundancy to the system which involves critical components being duplicated using additional hardware, software, and network resources.
In the context of cloud computing, there are 2 stakeholders: The CSP and the user of the cloud services. Both IBM and Jhaver tackle the solution of dependability and security of cloud computing systems through adding redundancy to the infrastructure provider's end of cloud computing. In principle, the system can be made more dependable on the user's end through deploying system dependability methods in the application layer, however, this responsibility is largely delegated to the CSP as CSPs have administrational control over the actual hardware and infrastructure of the cloud computing systems which enable the deployment of more capable and robust system dependability techniques when compared to application level solutions.
However, they apply this redundancy on the provider's end in differing ways. The reason why this difference of approach occurs is both authors make different assumptions regarding the infrastructure of the current cloud computing systems.

Assumption of IBM Regarding Cloud Computing Infrastructure

IBM assumes that a single cloud system can not be trusted to have all of the hardware, software, and network resources in place to achieve redundancy and support the implementation of system dependability techniques. However, they assume all of the necessary resources are in place between all of the CSPs.

Assumption of Jhavar Regarding Cloud Computing Infrastructure

Conversely, Jhavar assumes the computational resources to achieve redundancy are in place for a given CSP. They note, however, these resources are not being taken full advantage of.

IBM's "Intercloud" Solution: Taking Advantage Of Resources Outside Of A Single CSP

Overview

IBM proposes implementing "intercloud" on top of the existing cloud layer to form a two stage cloud computing infrastructure. Intercloud is designed to be an interface where a user can access and utilise the resources of multiple cloud providers as if it's a single service. This is illustrated in figure 1.

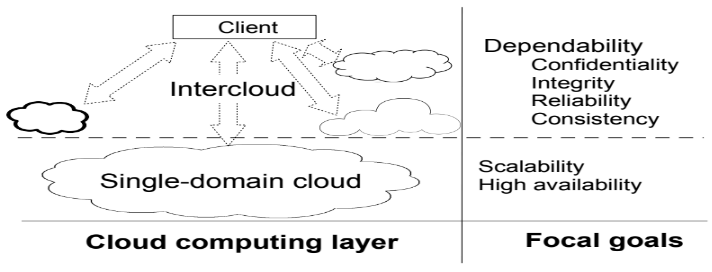

Figure 1: Dependable computing stack (Source: Cachin)

The intercloud layer addresses the concerns IBM has regarding single point of failure from two channels: Firstly, it partitions the overall trust needed for a dependable system implementation across multiple CSPs instead of a single

CSP. This makes sure in the case where a CSP fails to meet its promise of offering a computational system with confidentiality, integrity, reliability, and consistency the system can remain dependable by making up for any lost property through another CSP. Secondly, IBM recognizes that despite all measures, outages at individual CSPs can happen. The intercloud mitigates this risk of system dependability issues caused by outages at a given CSP site by making use of all different hardware, software, and network infrastructure that is available to cloud computing users around the world. In addition to this technical point of view, IBM also looks at the business point of view and notes implementing the intercloud is cost effective as all cloud computing users can absorb any additional cost together which would be negligible at scale. In contrast, it wouldn't be financially feasible for a single enterprise to achieve this level of system dependability on their own.

Fault Tolerance

One of the most important properties of dependable critical systems is to be fault tolerant (Cachin). Faults can be dormant, therefore, a presence of fault or error caused by fault activation by itself does not indicate failure. Service failure only happens when the error propagates to the system interface and causes a system behaviour deviation (Avizienis). Therefore, error detection and recovery techniques, together referred to as fault tolerance, should be deployed to intercept this chain reaction and ensure system dependability. Part of the solution to achieve fault tolerance is through failure independence. That is to say that failure in one part of the system should not affect the behaviour of the rest of the system. This requirement is usually hard to be satisfied, however, the critical system will have access to resources deployed in different geographical locations with the implementation of intercloud where each resource is supported by different power supplies, administrative domains, and internal cloud architectures. Therefore, the unprecedented diversity this solution brings is likely to achieve better failure independence and consequently fault tolerance and dependability.

Implementing Intercloud: Intercloud Storage and Protocols

In their paper, IBM describes how the implementation of intercloud might work from a lower level by detailing the outworking of intercloud storage which is the term data storage using the intercloud architecture is called.
To begin with, IBM notes that while SDC protocols are more provider centric, the intercloud storage protocols should be designed to be more client centric. From the user's point of view, these client-centric protocols will enable the client to intuitively use the system as they will only see and interact with a single intercloud storage system: ICStore Client. ICStore will use these client centric protocols to abstract away different APIs of different CSPs, the communication details of different clouds, and their arrangement for a given system. From the provider's point of view, these protocols will enable the modular addition of new CSPs to the intercloud storage service as a simple api adapter to the ICStore Client is all that is needed to complete the transition towards an intercloud architecture.
Finally, the ICStore client will have a layered design where each layer targets different system dependability and security properties: Confidentiality, integrity, reliability and consistency. This layered approach allows individual layers to be switched "on" and "off" to provide different levels of dependability. A high level view of this design can be observed in Figure 2.

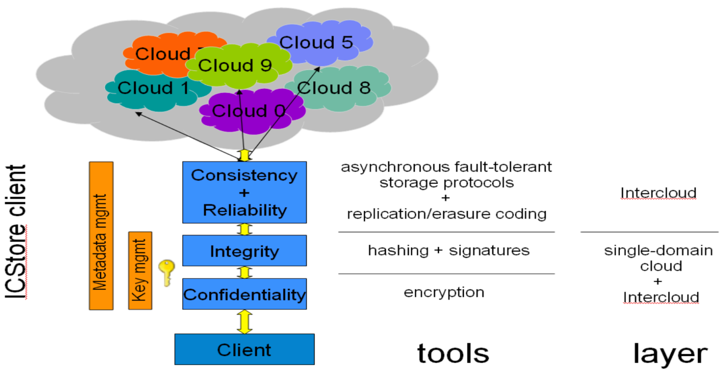

Figure 2: Intercloud Storage (Source: Cachin)

Jhaver's "Third Party Service Provider" Solution: Taking Advantage Of Resources Within A Single CSP

Overview

Similar to IBM, Jhaver also proposes a solution that deploys fault tolerance methods to achieve system dependability. Jhaver envisions the provision of a new system dependability service from a separate stakeholder in the cloud computing stack called a fault tolerance service provider (SP). This stakeholder will be in charge of providing fault tolerance support to entities based on the requirements of the system that will be deployed on the cloud.
In order to achieve this goal, Jhaver details what the SP must be capable of doing as maintaining a consistent view of the availability status of all resources in the cloud through a resource manager, develop different classes of fault tolerance algorithms to be utilised for a system and develop an interface that can seamlessly integrate with the existing cloud infrastructure.

Resource Manager

Jhavar envisions the resource manager (RM) to be implemented as a comprehensive database that keeps track of all log information such as event logs, error logs and performance logs as well as the physical and virtual resources' working state. Moreover, the inventory information such as the layout of physical machines, their identifying serial numbers and runtime state information such as used/free memory or used/free CPU cores should also be tracked with this database in order to make the resource manager aware of the full state of the cloud infrastructure of a CSP. To achieve this, the RM must introduce a graph representing the topology and working state of resources in the CSP's system. This ensures that in the case of a failure of a resource, the system can efficiently and effectively manage the remaining available resources. An example of such a graph can be seen in figure 3

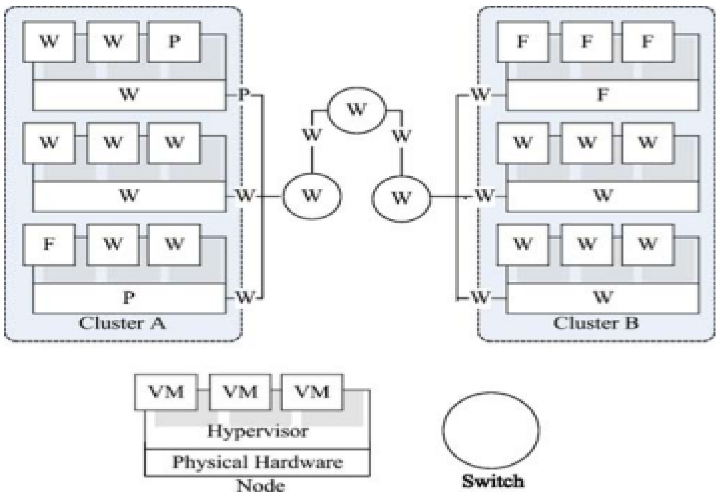

Figure 3: The graph that maintains topology and state details of physical and virtual CSP resources (Source: Jhavar)

With this graph, The SP will be able to represent the state of resources at different granularities as the SP will gain awareness whether a resource is working, partially faulty or completely faulty through the log information in the database. This is crucial for the correct working of fault tolerance mechanisms. For instance, a redundancy mechanism by replication may have constraints on relative placement of individual replicas and requirements on resource characteristics of each replica. Correct implementation of this mechanism is only possible by organisation of resources which will be enabled by the RM. Jhavar further notes that the resource manager significantly contributes toward balancing the constraint of cost against performance for the fault tolerance mechanisms offered by the service provider by aiding the determination of the minimum necessary algorithms and resources necessary for the implementation of system dependability needs.

Fault Tolerance Delivery Mechanism

Jhavar notes that in order to offer fault tolerance as a service, the SP will have to accommodate a wide range of system dependability needs in different contexts for different enterprises. This is only possible with the utilisation of different fault tolerance algorithms and techniques that cover both error detection and recovery mechanisms. Jhaver realises the individual techniques of fault tolerance as the crux of the system dependability and refers to these mechanisms as ft_unit. A ft_unit offers a coherent fault tolerance mechanism to a recurrent system failure which can be applied directly on a virtual machine (VM) rather than the application. For instance, the fault tolerance of the banking service can be increased by having a ft_unit that does recovery by replicating the entire VM instance that hosts the application tier across different physical nodes, and having another ft_unit that does error detection of the nodes that fail through a heartbeat protocol. This approach doesn't require any change to the application's source code, therefore, the fault tolerance delivery mechanism can be abstracted away from the end user.
Jhavar suggests implementing different ft_units using a two stage protocol: Design stage and runtime stage

Design Stage

The goal of the design stage is to identify the client's system dependability needs and accordingly match the necessary fault tolerance techniques offered by the SP to this system.
The matching process of the ft_units is done by comparing the different characteristics of each ft_unit using its functional, operational and structural attributes and determining whether these characteristics can efficiently address the dependability requirements of the system. These attributes can represent the resource consumption costs, and QOS parameters of the solution. The characteristics of a ft_unit can be denoted as follows:

Equation 1: <fault_tolerance_property>= (<ft_unit's name>, < {attribute_name}= {attribute_value or type}>)*

Where, a total or partial order relationship between an attributes name and its either value or type is used to represent the characteristics of a ft_unit. For instance, fault tolerance property of a ft_unit: u1 can be denoted as p=(u1, {mechanism=active_replication, no_of_replicas=4, fault_model=node_crashes}) This representation enables the binding of the attribute set of a given ft_unit to the ft_unit as its metadata. An algorithm that takes advantage of this binding scheme can partially automate the process of matching the dependability solutions to the dependability requirements of the system by creating a shortlist of possible solutions. The task of the SP from this point would be to analyse the set of requirements and decide on a solution that best optimises robustness, cost and performance with respect to the system's nature.
Although a ft_unit can serve as the nucleus of dependability, a comprehensive dependability solution ft_sol that can be dynamically changed will combine a set of ft_units in a specific execution logic. For example, a heartbeat test (ft_unit1) can be applied only after the client's application is replicated on multiple nodes (ft−unit2), and a recovery mechanism (ft_unit3) can be applied only after a failure is detected. This logic can be represented as follows:

```
ft_sol[
invoke:ft_unit(VM-instances replication)
invoke:ft_unit(failure detection)
do{
execute(failure detection ft_unit)
}while(no failures)
if(failure detected)
invoke:ft_unit(recovery mechanism)
].
```

Figure 4: A System Dependability Solution Offered By A SP Through Fault Tolerance (Source: Jhavar)

Due to the modular nature of the ft_units, the power and capability of fault tolerance mechanisms can be changed via swapping in and out different ft_units at run time which provides flexibility and optimises the cost the business incurs with the dependability solution. For instance, a robust failure detection mechanism (u3) can be replaced with a less robust one (u1).

Runtime Stage

The goal of the runtime stage is to ensure correct and efficient delivery of the fault tolerance techniques in the dynamic nature of the cloud computing environment through system monitoring. In order to achieve this, a set R of rules over attributes of a given ft_unit is defined. These rules specify what quantitative metrics are supposed to be met by the SP's solution. After these rules are determined, they are continuously checked against the system at runtime to ensure the client's dependability requirements are met. When the check fails, or the dependability requirements are changed by the end user, the service provider must initiate a process where the design phase is done again to select a new set of ft_units to form a ft_sol

Implementing Service Provider: An Service Oriented Framework

Jhavar discusses the implementation of the SP through a conceptual framework named Fault Tolerance Manager (FTM) which delivers fault tolerance as a service. This framework addresses the issues of heterogeneity of computing resources and delivers a transparent system dependability solution to the end user via a dedicated service layer that sits in between the virtual machine manager and the client's application.
The FTM follows the principles of service-oriented architecture in order to achieve granularity and abstraction, where each ft_unit is realised as an individual web service, and a ft_sol is formed using the business process execution language(BPEL) constructs. The FTM is composed of the following services: Resource manager, a client interface, a computing component called FTMKernal, replication manager, Fault Detection Or Prediction Manager, Fault Masking Manager, Recovery Manager, and messaging manager. In the following sections, we take a closer look at these different services that make up the FTM.

Resource Manager

The resource manager service's aim was discussed above. Accordingly, this service provides a status operation that takes a resource such as a processing node, storage, or memory as input and outputs the state of that resource. This status along with the update operation, generates a database that overviews the availability of resources.

Client Interface

Clients will need a way to interact with the SP to obtain dependability support for their applications. This is done through the service provider. While it is essential to include an interface that allows clients to specify their requirements through a specification language, this interface can be enhanced with an automated configuration tool that allows clients to obtain dependability support just by selecting the application they want to be dependable along with values of desired availability, reliability, response time and cost. If this interface is implemented with a high level language for dependability metrics such as percentages range and numbers, this automated configuration tool would lessen human errors and allow non-technical clients to obtain easier dependability support

FTMKernal

The central computing component of FTM framework is the FTMKernel that is responsible for composing a fault tolerance solution based on client's requirements using ft_units, delivering the composed service on client's applications, and monitoring each service instance to ensure its QoS. FTMKernel is composed of a service directory, a composition engine, and an evaluation unit.

Service Directory

The service directory is a collection of all ft_units realised by the service provider in the form of web services. As discussed, this modular component that applies different fault tolerance techniques also registers the metadata that represents the fault tolerance property of each ft_unit. When FTM receives input from the client interface, the FTMKernal first performs a matching between the client's preferences and properties of each ft_unit in the service directory, to generate the set of ft_units that satisfy the requirements. This set is then ordered based on the client's preferences and provided to the composition engine. To achieve agility, this process is triggered again if any change in requirements or failure occurs in runtime.

Composition Engine

This engine is responsible for the generation of the comprehensive fault tolerance solution ft_sol using ft_units upon receiving the ordered set of ft_units from the service directory as the input.

Evaluation Unit

As discussed before, the service provider must monitor the delivery of fault tolerance solutions at runtime in order to assert correct and efficient service. To this end, the evaluation unit continuously monitors all composed fault tolerance solutions at runtime using the validation mechanisms and the set of rules defined corresponding to each ft_sol.

Replication Manager

This manager provides support to ft_units that employ replication mechanisms by managing the details of individual replication techniques constrained by the type of replication the client needs such as the replica location or synchronisation process between them . This abstracts away the tedious mechanisms of the replication process which aids with achieving correct service behaviour and ensures the client perceives a single service

Fault Detection Or Prediction Manager
This manager provides failure detection as part of the error detection aspect of fault tolerance to the FTM from both a global level that provides failure detection across all the nodes in the cloud and local level which detects failures among individual VM replicas.

Fault Masking Manager

This manager supports ft_units that realise fault masking mechanisms so that occurrence of faults in the system do not evolve to failures.

Recovery Manager

This manager helps FTM achieve system-level resilience by minimising the downtime of the system during failures. To this aim, this component supports ft_units that deploy recovery mechanisms so that an error-prone node can be resumed back to a normal operational mode

Messaging Monitor

The messaging monitor works over all the components of our framework (as shown in Figure 5) and offers the communication infrastructure in two different forms: message exchange within the replicas of a client system, and inter component communication within the framework. Since ft_units, and other components in FTM, are realised as web services, the communication between any two components or the replicas of the client system must be reliable even in the presence of component, system or network failure. Therefore, this component is critical in providing maximum interoperability, and consequently overall expected service behaviour.

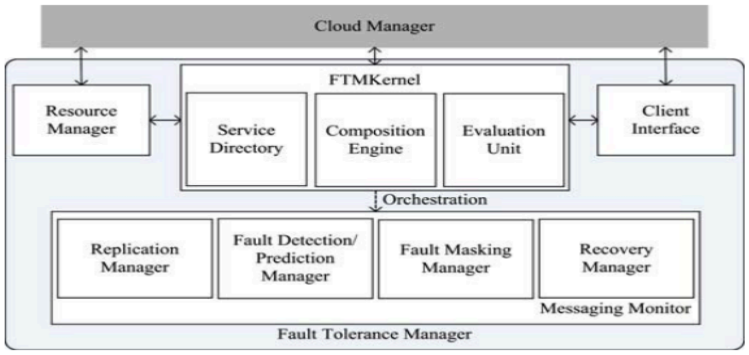

Figure 5: The FTM architecture (Source: Jhavar)

Critical Reflection of IBM and Jhavar's Solutions In The Context Of Critical System Development

Critical system engineering of a software system always requires hardware support. Without this hardware support, achieving system dependability is not possible. Both papers were written with the common goal of making cloud computing a dependable system. However, they made differing deductions regarding the underlying hardware infrastructure of cloud computing systems. IBM assumed that a single cloud system won't have sufficient diversity of hardware and network resources in place to achieve redundancy. On the other hand Jhavar did not identify a lack of resource diversity as an issue. Therefore, IBM first addressed this resource insufficiency through the intercloud solution that grants access to cloud computing systems different resources around the globe.
Similarly, however, both IBM and Jhavar tried to achieve system dependability through deployment of fault tolerance mechanisms. IBM implemented these mechanisms in the reliability and consistency layer of their intercloud storage model by extending RAID techniques for the cloud with asynchronous communication and fault-prone client proxies. While IBM presents a more high level solution, Jhavar's work is more detailed offering solutions and implementation techniques that cover both error detection and recovery aspects to build a fault tolerant solution.
Moreover, while both solutions discussed deploying dependability techniques at the provider's end, dependability techniques should also be deployed on client's end. For instance, Netflix has a microservice architecture alongside cloud computing and where they use a concept called "chaos engineering" to increase the dependability of their service. In this technique, they identify core features they see as business critical such as being able to start a video, and turn off cloud nodes or introduce network failures to ensure the system keeps functioning even in the event of failure. This cycle of introducing an error and observing the output continues until a system failure occurs in which case the engineers then correct the code to handle the specific type of fault events (Basiri).

Furthermore, while IBM recognizes security issues as part of system dependability and takes steps to tackle this through ensuring data confidentiality and integrity with encryption, hashing, and signature techniques this is neglected by Jhavar's solution. None of the papers, however, present solutions encompassing fault prevention, fault removal, or fault forecasting which would enable the development of a more dependable cloud computing system. One challenge both papers face is deploying this system over the scale of the internet. This issue is better illustrated by Reason's swiss cheese model which recognizes that each layer of countermeasure against system failure will still have vulnerabilities, and at the scale of the internet, these vulnerabilities will align to cause a failure. Despite this fact, these papers still present an effective solution against system failures in the cloud.

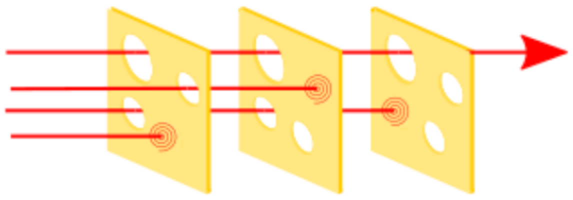

Figure 6: Reason's Swiss Cheese Model (Source 'Swiss Cheese Model'. Wikipedia)

As discussed before, these solutions can be collectively classed as dependability by service. This concept can help mitigate other system dependability concerns in different domains especially when data storage is a crucial part of the critical system. The benefit of this concept is its tendency to abstract away technical details which can help provide better system dependability. For instance, the mental health care patient management system had concerns regarding dependability issues caused by conflicting concerns. If there was a service that offered dependability as a service, they would be better equipped to handle these conflicts as they would be more likely to provide dependability solutions at finer granularities.

Lastly, in the context of socio-technical critical systems where individuals directly interact with the system, their inability to understand system details does lead to system dependability or security issues caused by misuse. Again, the dependability as a service approach can help mitigate this issue. For instance, many people use a single password to secure their different profiles which makes them vulnerable to reuse attacks in case a password is compromised. The authenticator applications which provide 2 factor authentication can be viewed as a dependability by service app as security is part of dependability.

Conclusion

In conclusion, both papers present valid solutions to achieve dependability in the cloud. While the approaches taken differ at certain instances, both papers deploy fault tolerance techniques to achieve system dependability. However, we also note that in the context of cloud computing, these dependability features offered by the cloud computing providers can be further improved by deploying dependability methods in the application layer. The solutions presented in these papers can be viewed as a dependability by service framework which has potential to lead to improvements in system dependability in other domains, and future work is encouraged to explore this possibility more in depth.

References

Basiri, Ali, et al. ‘Chaos Engineering’. *IEEE Software*, vol. 33, no. 3, May 2016, pp. 35–41. *DOI.org (Crossref)*, https://doi.org/10.1109/MS.2016.60.

Brazhenenko, Maksym, et al. ‘Cloud Based Architecture Design of System of Systems’. *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, IEEE, 2019, pp. 1–5. *DOI.org (Crossref)*, https://doi.org/10.1109/CADSM.2019.8779307.

Vogels, Werner. ‘Eventually Consistent’. *Communications of the ACM*, vol. 52, no. 1, Jan. 2009, pp. 40–44. *DOI.org (Crossref)*, https://doi.org/10.1145/1435417.1435432.

Avizienis, A., et al. ‘Basic Concepts and Taxonomy of Dependable and Secure Computing’. *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, Jan. 2004, pp. 11–33. *DOI.org (Crossref)*, https://doi.org/10.1109/TDSC.2004.2.

‘What Is IaaS? - Infrastructure as a Service Explained - AWS’. *Amazon Web Services, Inc.*, https://aws.amazon.com/what-is/iaas/. Accessed 26 Oct. 2023.

Cachin, Christian, et al. *Dependable Storage in the Intercloud.*

Jhawar, Ravi, et al. ‘Fault Tolerance Management in Cloud Computing: A System-Level Perspective’. *IEEE Systems Journal*, vol. 7, no. 2, June 2013, pp. 288–97. *DOI.org (Crossref)*, https://doi.org/10.1109/JSYST.2012.2221934.

Mujinga, Mathias. ‘Cloud Computing Inhibitors Among Small and Medium Enterprises’. *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*, IEEE, 2020, pp. 1385–91. *DOI.org (Crossref)*, https://doi.org/10.1109/ICISS49785.2020.9315905.