İLERİ MÜHENDİSLİK ÇALIŞMALARI VE TEKNOLOJİLERİ DERGİSİ

The Implementation of Chaos Engineering in Cloud Architecture and Applications

Mehmet Altuğ AKGÜL*[1], Hakan GÜVEZ[2]

Research Article, Received Date: 02.04.2024, Accepted Date: 23.08.2024

Abstract

The emergence of microservice architecture and the concept of Chaos Engineering, which was researched to understand how the system will behave in case of possible interruptions and service problems that may occur when software services trying to do the same job in different locations talk to each other, play an important role in determining the robustness of systems exposed to intense network traffic. In this article, the concept of Chaos Engineering, which has emerged in recent years and has started to be used especially in checking the reliability of distributed systems, is mentioned. As a result, it is emphasized that Chaos Engineering can play an important role in increasing the reliability of distributed systems running on the cloud.

Keywords: Cloud, Chaos engineering, Distributed systems, Microservices, Infrastructure.

1. INTRODUCTION

This article includes a research on while monolithic applications were common, all business logic ran in a single service and a single database, and there were actually huge applications controlled from a single place. However, especially with the increase in network speeds and traffic density of cloud systems, monolithic applications no longer respond to demand. It has become impossible for huge software teams to work on a single monolithic application. For this reason, an approach called microservices emerged where each service does only one job or has a minimum number of job responsibilities. Moreover, it was a very suitable architectural approach for distributed systems.

Chaos Engineering is a discipline that emphasizes on in-tensional injection of faults into software systems to minimize downtime while increasing resiliency. The main motivation for this approach is to overcome uncertainties prevalent in distributed systems e.g. cloud infrastructure (Kennedy et al., 2019).

Implementing chaos engineering in cloud environments is critical due to the inherent complexity and dynamics of cloud infrastructure. As companies increasingly rely on cloud services for their critical operations, the need for resilience and reliability becomes increasingly important. Chaos engineering provides a proactive approach to identifying and mitigating potential failures by intentionally introducing disruptions into the system. This approach helps uncover vulnerabilities that may not be apparent during normal operation, ensuring that the system can withstand unexpected challenges. By regularly testing the system's ability to handle failures, companies can build more robust and resilient cloud architectures. Additionally, the insights gained from chaos engineering experiments enable teams to implement improvements and develop better incident response strategies, ultimately improving overall service availability and user satisfaction. Discussing how to implement chaos engineering in the cloud is critical to fostering a culture of continuous improvement and operational excellence given the ever-evolving technology landscape.

*[1]Corresponding author mehmetaltugakgul@gmail.com,
[2]guvezhakan@gmail.com

A rapid microservices transformation has begun all over the world. Software teams have worked quickly to transform legacy monolithic applications into microservices architecture, and with the help of frameworks, many projects are no longer monolithic but serve in a microservices architectural approach and cloud native.

In addition to the advantages of having everything run on microservices responsible for a single job, the disadvantages include the difficulty in service management and the difficulty of these services communicating with each other. It becomes impossible to understand how the system will react, especially in the event of an unexpected error or service stop. Data integrity and high accessibility are two very important factors, especially in services with high transaction density, such as fintech. The possibility of losing millions of dollars even in the event of a 5-minute system crash requires these systems to operate flawlessly. Therefore, all disaster scenarios need to be created, tested and appropriate measures taken. Regardless of how distributed or complex the system is, the sources of systemic errors are generally clear and these errors must be found in advance and the fragility of the system must be determined through relevant tests.

2. EMERGENCE OF CHAOS ENGINEERING

Chaos engineering is the discipline of experimenting on a distributed system to build confidence in a system's ability to withstand unexpected failures. This method is used to increase the flexibility of complex systems by deliberately adding errors and measuring the results. Chaos engineering identifies and remedies potential system weaknesses and vulnerabilities in the design, architecture and operational applications of the system by simulating controlled malfunctions. There are many ways of adding failure to the system, including closing a service, adding delays or errors (for example, a service cancellation attack), terminating operations or tasks, or simulating a change in the environment or configuration settings. This makes it easier to measure performance and reliability measurements such as system operating time, error rates, and recovery times. This helps us better understand how the system responds to deliberate error occurrences and increases the resilience of systems by using the information obtained. Therefore, it is difficult to ensure the reliability of local applications in the cloud and requires more research. Chaos engineering simulates real-world conditions and malfunction scenarios to assess system durability and reliability. Failure injection causes errors in test systems, while chaos engineering advocates deliberately injecting errors directly into production systems. This allows teams to identify weaknesses and increase overall system resilience (Ahmad et al., 2024).

Netflix moved its traditional architecture systems to the cloud environment in the 2010s, and this migration process took an average of 7 years. During this long process, there were no tools to detect systemic errors and to test the consequences of these errors in the cloud environment. The Chaos Monkey tool was a tool that emerged from this deficiency. People who work professionally in the field of testing are familiar with the testing method called "monkey test". The purpose here was based on the idea that a monkey left in the system room would randomly damage the relevant system components, network cables, servers and other parts. In such a situation, an attempt was made to measure how systemic outages would affect the main system. In this way, precautions could be taken to reduce the fragility of the system against unexpected outages, and these outages occurred randomly on an unexpected day and time. Real-life problems often arise unexpectedly.

Following these developments, the Chaos Engineering concept and the Chaos Engineer position as a position in professional business life began to become widespread and used more. Industry giants such as Facebook, Google, LinkedIn and Amazon started working on Chaos Engineering and creating employment.

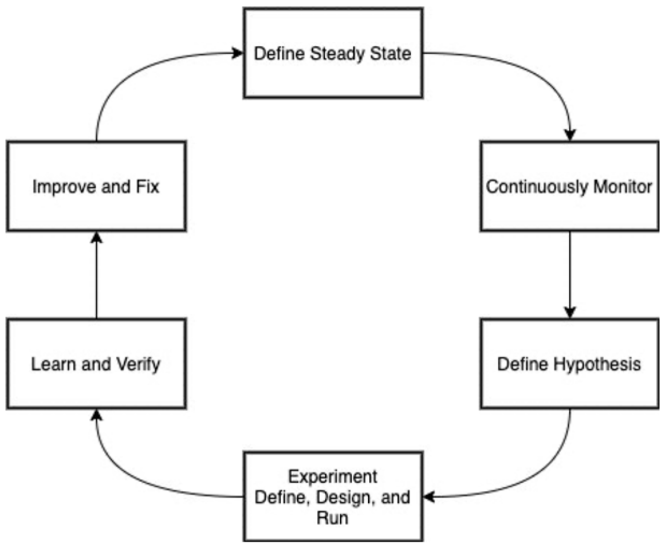


Figure 1. Cycle of chaos engineering experiments

3. APPLICATIONS OF CHAOS ENGINEERING

Chaos engineering, as in many projects, starts from the planning phase, and this planning generally consists of considering the errors that may occur in the system and the preliminary preparation process. These errors can cover a wide range from very common errors to the rarest problems.

After the planning phase, the Exploding phase comes into play and a small-scale error is created in the system. In other words, the necessary information can be obtained at the point where the system crashes and becomes unresponsive. If information cannot be obtained, this blasting scale is enlarged and scaled. If the problem is solved, a different trial can be started; if the problem is not solved, the system can be scaled until this problem is solved.

In fact, deliberately detonating the system adopts the approach of making the system immune to possible problems, just like our body's immune system. The more familiar the system is with such situations, the weaker the effect it will have (IBM., n.d.).

The probability of error is much higher, especially in distributed systems. Because the system consists of the combination of different subsystems working remotely from each other. Here, many problems such as network problems, packet transfers, delays, bandwidth, and dissimilarity of devices in the system are the problems that come with distributed systems. At this point, there may be different misconceptions that all system designers get caught up in. Peter Deutsch collected these misconceptions in a list of 8 items (Wikic2., n.d.).

1. **The network is reliable.**
2. **Latency is zero.**
3. **Bandwidth is infinite.**
4. **The network is secure.**
5. **Topology doesn't change.**
6. **There is one administrator.**
7. **Transport cost is zero.**
8. **The network is homogeneous.**

In fact, all the items in this list summarize the problems we encounter in modern distributed systems. In general, system architects accept most things under normal conditions when designing systems, but generally all problems arise after these mistakes. With cloud use, system management in general is provided through cloud providers that we call hyperscaler, and shared responsibility models are generally clearly stated in user agreements, and the responsibilities of the infrastructure owner and the customer are different. For this reason, although there are not many infrastructure, maintenance and operational problems in modern cloud architectures, problems that may occur after user error are much more serious. Especially chaos engineering on the cloud is very different from on-premise systems.

Various platforms and technologies facilitate chaos engineering experiments on cloud systems. Gremlin and Litmus are notable examples of error addition and the ability to evaluate the flexibility of systems. The techniques used by Netflix engineers to test the system's resistance to unforeseen failures led to the emergence of the chaos engineering field. To identify weaknesses and increase system resilience, use concepts such as error addition and edited trial (Rahul, 2024).

In addition, chaos engineering in systems should be planned by considering all components of the system. With a transaction-based intensive payment system, scheduled systems that operate at certain times of the day are not subject to the same interruptions and fragility. Network, storage, database and architecture teams are expected to work together and make chaos engineering planning appropriate to their systems.

We can also separate errors according to the types we encounter.

Known errors: Errors that we are aware of, familiar with, and can easily understand are included in these errors.

Known and unknown errors: These are errors that we are aware of and familiar with, but cannot easily understand.

Unknown known errors: These are errors that we are not aware of or familiar with, but can easily understand.

Unknown unknown errors: Errors that we are neither aware of, familiar with nor can we easily understand also fall into this class.

4. WHICH TYPES ARE AVAILABLE?

While preparing chaos engineering test environments, different types of methods can be used. These methods are listed below. Methods can be diversified, but in general, proceeding with these headings is appropriate in terms of best practices and compatible with current life practices.

Load Generation: It can be summarized as performing a load test on the system by creating a traffic load much higher than the traffic on the media received by the system. Thus, the degree of fragility of the system can be measured.

Canary Testing: Canary testing is a method used to open an application or a systemic change to a selected small group, rather than to all users, and to help them understand whether there is any problem in the system. The remaining users will not be informed of the update until the test is finished.

Error injection: This type is a technique that deliberately injects an error into a system that generally has no errors and makes it easier for us to understand how the system works. Types of error injection can be very diverse. In general, there may also be physical difficulties such as hardware malfunction or increasing the ambient temperature. Or deliberately corrupting the software also falls under this type of injection.

Latency injection: Testing the instability of the system by deliberately creating low-speed network or network connection errors. These levels of slowness can be repeated in a certain principle.

5. AREAS OF IMPLEMENTATION FOR CHAOS ENGINEERING IN CLOUD

Chaos Engineering, an innovative approach aimed at enhancing the resilience of software systems through deliberate fault injection, has become a cornerstone in ensuring system robustness within the tech industry. This proactive methodology is spearheaded by dedicated teams, often small in number, whose mission is to embed these practices across their organizations. These teams not only apply Chaos Engineering techniques directly but also empower and motivate other engineering departments to adopt these practices in their daily workflow.

The implementation of Chaos Engineering typically begins with crucial service-oriented teams, who lead the charge in its adoption and serve as advocates for its benefits across their organizations. These teams are responsible for crucial areas such as:

Managing Traffic Infrastructure (examples include HAProxy, Squid, Load Balancers)

Streaming Services (such as RabbitMQ, ActiveMQ, Stream Processing Platforms)

Storage Solutions (for example, Cassandra, Redis, Distributed File Systems)

Data Processing Frameworks (e.g., Spark, Flink, Big Data Analytics Platforms)

Database Management (including Oracle, SQL Server, NoSQL Databases)

By focusing on these essential components, organizations can significantly improve the resilience and reliability of their foundational services.

Forward-thinking companies have integrated Chaos Engineering into their regular release cycles, positioning it alongside other established testing methodologies. This approach ensures that considerations of reliability are woven into the fabric of the development process from the outset, embedding durability into every feature prior to its release. This shift in perspective highlights the evolution of Chaos Engineering from a specialized technique to a vital element of software development, emphasizing its critical role in constructing robust systems capable of withstanding the complexities of the modern digital landscape.

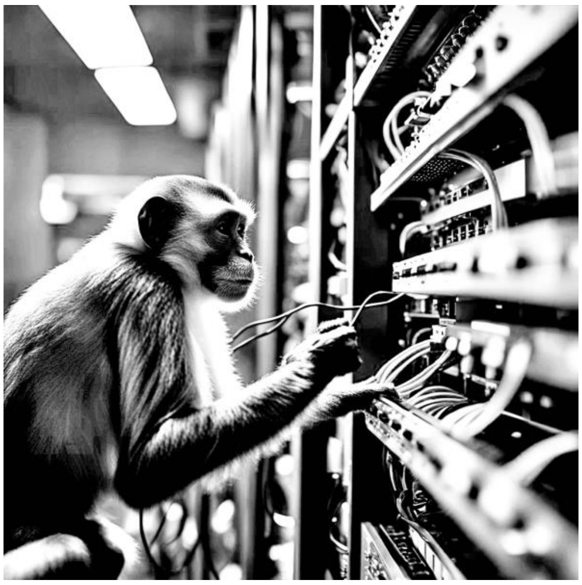


Figure 2. Chaos engineering tests can be thought of as a monkey being left in a system room and randomly interfering with servers and cables

6. CHAOS ENGINEERING FOR LOAD BALANCERS

In our online world, load balancers work like junctions on a busy road, making sure internet traffic goes where it should. Chaos Engineering is like testing those junctions with unexpected problems to make sure they can handle the pressure and keep the traffic flowing smoothly.

Known-Knowns: The awareness of an infrastructure's capability to reroute traffic in the event of a load balancer failure is well-established. Systems are equipped with health check mechanisms to identify inactive or malfunctioning load balancers.

Experiment: The simulation of a single load balancer's failure during off-peak hours is conducted to observe the efficacy of traffic rerouting and to quantify the detection and rerouting duration.

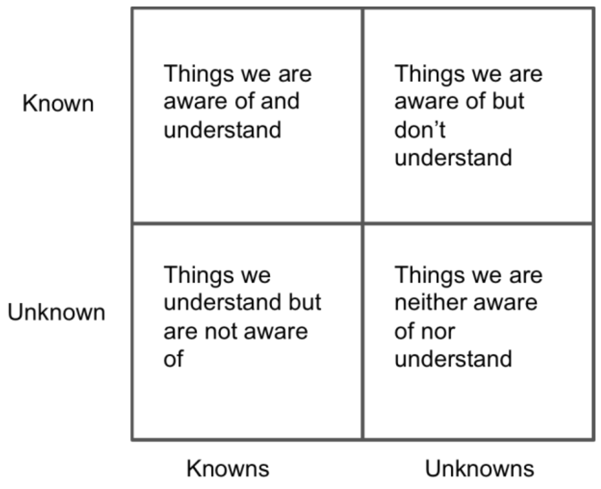

Known
Unknown
Things we are aware of and understand
Things we are aware of but don't understand
Things we understand but are not aware of
Things we are neither aware of nor understand
Knowns
Unknowns

Figure 3. Chaos engineering things table (Mesh, 2024).

Known-Unknowns: While load balancers are configured to manage a specified volume of traffic, the precise limitations of the current setup and the consequences of abrupt traffic surges remain uncertain.

Experiment: The load on load balancers is incrementally increased beyond anticipated capacity to ascertain the threshold at which failure or significant performance degradation occurs. The system's response and recovery duration are meticulously measured.

Unknown-Knowns:

The premise that load balancers distribute traffic according to specified rules, such as least connections or round-robin algorithms, is understood. However, the efficiency of these algorithms under specific conditions is not fully known.

Experiment: Controlled scenarios are crafted to simulate particular traffic patterns that might result in disproportionate load distribution. The effectiveness of the current algorithm in managing these patterns is evaluated.

Unknown-Unknowns:

Potential vulnerabilities may arise from the complex interdependencies between load balancers and other infrastructure components like DNS services or internal network configurations.

Experiment: A comprehensive shutdown of all load balancers is executed to examine the infrastructure's reaction. Such a rigorous test has the potential to reveal latent dependencies or failure points within the system's architecture that have not been previously considered.

Through methodical progression across these categories with targeted experiments, the opportunity arises to detect and rectify potential frailties within the Load Balancer configuration, thereby reinforcing the system's overall robustness. This approach necessitates meticulous planning, execution, and analysis of each experiment to accrue valuable insights while curtailing the impact on the operational environment.

7. CHAOS ENGINEERING TOOLS

1-Litmus / Harness Chaos Engineering:

Platforms: Kubernetes

Release year: 2018

Creator: MayaData

License: Open source (with a managed option)

Litmus started as a testing tool for OpenEBS and has since grown into one of the largest open-source Kubernetes-native Chaos Engineering tools. It provides a library of faults for testing containers, hosts, and platforms such as Amazon EC2, Apache Kafka, and Azure.

2-AWS Fault Injection Simulator and AWS Resilience Hub

Platforms: AWS

Release year: 2021

Creator: Amazon Web Services

License: Commercial

AWS Fault Injection Simulator (FIS) lets you introduce faults into AWS services, including Amazon RDS, Amazon EC2, and Amazon EKS. AWS Resilience Hub evaluates your AWS environment, compares them to reliability policies, and provides improvement recommendations.

3-Azure Chaos Studio

Platforms: Azure

Release year: 2021

Creator: Microsoft

License: Commercial

Azure Chaos Studio is a Chaos Engineering solution for running faults directly on the Azure API. It supports faults on Azure Compute instances, CosmosDB, and Azure Cache for Redis. It also supports Kubernetes via integration with Chaos Mesh.

4-Steadybit

Platforms: Docker, Kubernetes, Linux hosts

Release year: 2018

Creator: Steadybit

License: Commercial

Steadybit is a commercial Chaos Engineering tool that aims to build remediation into its experiments.

5-Chaos Monkey

Platform: Spinnaker

Release year: 2012

Creator: Netflix

Language: Go

It was one of the first Chaos Engineering tools and kickstarted the adoption of Chaos Engineering outside of large companies.

6-ChaosBlade

Platforms: Docker, Kubernetes, bare-metal, cloud platforms

Release year: 2019

Creator: Alibaba

Language: Go

ChaosBlade is a CNCF sandbox project built on nearly ten years of failure testing at Alibaba. It supports many platforms, including Kubernetes, cloud platforms, and bare-metal, and provides dozens of attacks, including packet loss, process killing, and resource consumption.

7-Chaos Mesh

Platform: Kubernetes

Release year: 2020

Creator: PingCAP

License: Open source

Chaos Mesh is a Kubernetes-native tool that lets you deploy and manage your experiments as Kubernetes resources. It's also a Cloud Native Computing Foundation (CNCF) sandbox project (Wikic2., n.d.).

8-Chaos Toolkit

Platforms: Docker, Kubernetes, bare-metal, cloud platforms

Release year: 2018

Creator: ChaosIQ

License: Open source

Chaos Toolkit will be familiar to anyone who's used an infrastructure automation tool like Ansible. Instead of making you select from predefined experiments, Chaos Toolkit lets you define your own.

9-Toxiproxy

Platforms: Any

Release year: 2014

Creator: Shopify

License: Open source

Toxiproxy is a network failure injection tool that lets you create conditions such as latency, connection loss, bandwidth throttling, and packet manipulation. As the name implies, it acts as a proxy that sits between two services and can inject failure directly into traffic.

10-Istio

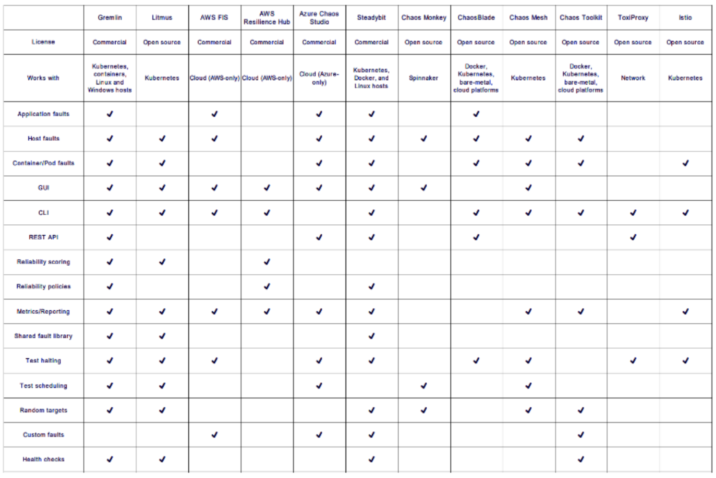
Platform: Kubernetes

Release year: 2017

Creators: Google, IBM, and Lyft

License: Open source

Istio is best known as a Kubernetes service mesh, but not many know it natively supports fault injection as part of its traffic management feature.



	Gremlin	Litmus	AWS FIS	AWS Resilience Hub	Azure Chaos Studio	Steadybit	Chaos Monkey	ChaosBlade	Chaos Mesh	Chaos Toolkit	ToxiProxy	Istio
License	Commercial	Open source	Commercial	Commercial	Commercial	Commercial	Open source	Open source	Open source	Open source	Open source	Open source
Works with	Kubernetes, containers, Linux and Windows hosts	Kubernetes	Cloud (AWS-only)	Cloud (AWS-only)	Cloud (Azure-only)	Kubernetes, Docker, and Linux hosts	Spinnaker	Docker, Kubernetes, bare-metal, cloud platforms	Kubernetes	Docker, Kubernetes, bare-metal, cloud platforms	Network	Kubernetes
Application faults	✓		✓		✓	✓		✓				
Host faults	✓	✓	✓		✓	✓	✓	✓	✓	✓		
Container/Pod faults	✓	✓			✓	✓		✓	✓	✓		✓
GUI	✓	✓	✓	✓	✓	✓	✓		✓			
CLI	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓
REST API	✓				✓	✓		✓			✓	
Reliability scoring	✓	✓		✓								
Reliability policies	✓			✓		✓						
Metrics/Reporting	✓	✓	✓	✓	✓	✓			✓	✓		✓
Shared fault library	✓	✓				✓						
Test halting	✓	✓	✓		✓	✓		✓	✓		✓	✓
Test scheduling	✓	✓			✓		✓		✓			
Random targets	✓	✓				✓	✓		✓	✓		
Custom faults			✓		✓	✓				✓		
Health checks	✓	✓				✓				✓		

Figure 4. Chaos engineering tools comparison matrix (*Comparing chaos engineering tools)*

8. WHAT ARE THE BENEFITS

We have talked about many features of chaos engineering, but it would be good to talk about its benefits in general. First of all, determining and testing this approach shows how well the system can survive against problems that may occur in live environments and increases the reliability of the system. If errors are known in advance, preliminary studies can be carried out on this issue to ensure that the system is resistant to major errors. It prevents major problems in remote systems of users and customers and reduces repair and maintenance activities. It saves money and time for the customer and provides long-term reliability to the company. It prevents unexpected interruptions in distributed systems and creates a working area for high availability. In addition, it gives an idea about all the weaknesses of the system and how to act in case of disaster.

9. CONCLUSION

In this article, we have discussed Chaos Engineering is a concept that has started to be used to check the reliability of distributed systems and systems running on the cloud, especially distributed systems running on the cloud. In this context, it is stated that Chaos Engineering involves creating errors around the service in a controlled manner in order to verify the durability of the service and find weaknesses. Chaos Engineering is used to understand the behavior of systems by simulating real-world scenarios, detect weaknesses and strengthen the system against unexpected failures. This continuous improvement process ensures that cloud systems become more robust and reliable. You can also strengthen the fragility of your systems by using best practices and relevant tools.

REFERENCES

Al-Said Ahmad, A., Al-Qora'n, L.F. & Zayed, A. Exploring the impact of chaos engineering with various user loads on cloud native applications: an exploratory empirical study. Computing 106, 2389–2425 (2024). https://doi.org/10.1007/s00607-024-01292-z

Chaos Mesh. (2024). Chaos Mesh Overview. Chaos Mesh. https://chaos-mesh.org/docs/

Comparing Chaos Engineering tools. (2024, February 19). Gremlin. https://www.gremlin.com/community/tutorials/chaos-engineering-tools-comparison

IBM. (n.d.). What is Chaos Engineering? Retrieved from https://www.ibm.com/topics/chaos-engineering

Torkura, Kennedy & Sukmana, Muhammad Ihsan Haikal & Cheng, Feng & Meinel, Christoph. (2019). Security Chaos Engineering for Cloud Services. https://doi.org/10.1109/NCA.2019.8935046.

Yadav, Rahul. (2024). Harnessing Chaos: The Role of Chaos Engineering in Cloud Applications and Impacts on Site Reliability Engineering. 72. 25-30. https://doi.org/10.14445/22312803/IJCTT-V72I6P104.

Wikic2. (n.d.). Eight Fallacies of Distributed Computing. Retrieved from https://wiki.c2.com/?EightFallaciesOfDistributedComputing