2020

Distributed public key based computing: The super virtual computer

Zachary Monaghan
Iowa State University

Distributed public key based computing: The super virtual computer

by

Zachary Monaghan

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Cyber Security

Program of Study Committee:
Doug Jacobson, Major Professor
Jacquelyn Ulmer
Yong Guan

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2020

TABLE OF CONTENTS

Page

NOMENCLATURE iv

ACKNOWLEDGMENTS v

ABSTRACT vi

CHAPTER 1. PROBLEM DETAILS 1
- Allocation Issues 1
- Scalability Issues 2
- Reliability Issues 2
- Security Issues 3

CHAPTER 2. DISTRIBUTED COMPUTING 7
- Peering Methodology 7
- Performing Computing Tasks 9
- Distributing Tasks 9
- Securely Distributing Sensitive Data 10

CHAPTER 3. COMPUTING STRUCTURE 12
- Function as a Service 12
- Distributing Tasks 13

CHAPTER 4. PEER VALIDATION 15
- Public Key Infrastructure 15
- Blockchain 15

CHAPTER 5. COMBINATION OF TECHNOLOGIES 17
- The Combination 17
- Benefits 20
- Drawbacks 20
- Uniqueness 21

CHAPTER 6. SOLUTION EVALUATION 23
- Evaluation Criteria 23
- Experiment Steps 23
- Evaluation Results 24
- Lessons Learned 24

CHAPTER 7. MAJOR CONTRIBUTIONS 27
- Peer Validation with PKI 27
- Arbitrary Golang FaaS API Implementation 27
- Decentralized FaaS Computing 28

CHAPTER 8. ALTERNATIVE SOLUTIONS 29
Fog Computing 29
Traditional Client-Server Methods 31
Cloud Computing Models 32

REFERENCES 35

NOMENCLATURE

API	Application Program Interface
AWS	Amazon Web Services
CIA	Confidentiality, Integrity, Availability
FaaS	Function as a Service
HFT	High Frequency Trading
HTTPS	Hypertext Transport Protocol (Secure)
IaaS	Infrastructure as a Service
IT	Information Technology
IOT	Internet of Things
PaaS	Platform as a Service
PKI	Public Key Infrastructure
MQTT	Message Queuing Telemetry Transport
REST	Representational State Transfer
STUN	Session Traversal Utilities for NAT
TLS	Transport Layer Security
VDI	Virtual Desktop Infrastructure

ACKNOWLEDGMENTS

I would like to thank my committee chair, Doug Jacobson, and my committee members, Jacquelyn Ulmer, and Yong Guan, for their guidance and support throughout the course of this research.

In addition, I would also like to thank my friends, colleagues, the department faculty and staff for making my time at Iowa State University a wonderful experience.

And finally, a special thanks to my MBA cohort and MBA faculty and staff for their support during my graduate studies at Iowa State University.

ABSTRACT

A problem with modern computing architecture is allocation of resources. When distributed systems are scaled, the workload is rarely shared amongst nodes efficiently. At an organizational level, there are even more allocation issues; workstations are typically oversized and underutilized. The extra capacity of these workstations could be utilized to assist servers with computing tasks. The underutilized resources can be redistributed towards server tasks in order to solve the modern problems of under sizing, redundancy and scalability.

My approach to this problem is to use a combination of modern methodologies and technologies to reallocate unused resources towards collaborative computing efforts. To do this, we can combine any number of devices into a resource pool to be used by those who need additional computing resources. To achieve this, I used a compiled language that reaches many platforms. My solution uses public key infrastructure to validate nodes and their computation results. To perform the computing, I used a cloud-first approach based on the FaaS model. Using these nodes in a distributed architecture, the computing can be allocated more effectively with built-in redundancy and scalability. This technology can ease computational shortcomings and reduce capital expenditures in organizations.

CHAPTER 1. PROBLEM DETAILS

Allocation of resources is a common problem in many organizations and even in individual computing systems. This problem is normally solved by adding more resources, but this approach is unsustainable and inefficient. For example, when you pay per use in a cloud environment, adding more resources only adds cost. A more efficient allocation of existing resources, taking into account workstations, could increase computing output without increasing costs.

Allocation Issues

There have been many allocation issues discussed at a CPU-interaction level. However, these issues can be seen at a network level scale as well. Many enterprise systems networks are built with ample resources to handle the required workloads. The majority of users in a typical organization will only use one fourth of their computer's capacity. According to Joe Weinman in Cloudonomics, the cost of underutilization is significant, and he claims that buying wine would be better than oversizing the computing power of an organization because wine appreciates in value while servers depreciate [1]. With this in mind, there is never a situation where all computers and servers are reaching their full capacity. The issue here is that servers can run out of computing power, while workstations have plenty to spare. The opposite side of this issue, where workstations lack power, has been solved with virtualization servers, using technologies such as VDI. Notably, my approach is also opposite of this as VDI is a centralized infrastructure that can easily lack scalability whereas my solution is distributed and further scalable.

Scalability Issues

The problem with scaling is that organizations need to have large amounts of resources available to meet peak demand while under normal conditions there is excess capacity. The software I've created while doing my research would allow this excess to be used to perform other computing tasks; or allow other resources to be pooled to meet peak demand without oversizing production servers. The solves the scalability issue of computing power by spreading out the work amongst workstations as well as servers.

The issue created by my solution is network bandwidth consumption and the scalability of a distributed computing network. To be practical and efficient, my solution needs to be deployed on a state-of-the-art network that has the extra capacity for sending the additional data all over the network, from workstation to workstation. To mitigate this issue, I used a RESTful framework that minimizes the request data sent over the network. In addition to minimizing this data, I would deploy this network in production using TLS 1.3, as it would also reduce network bandwidth and improve security. "TLS 1.3, a new encryption protocol that improves both speed and security for Internet users everywhere" [2]. In 2018, the protocol was officially finished, but we are still waiting on vendors to finish their implementations so we can take full advantage of the benefits of TLS 1.3.

Reliability Issues

The issue with reliability is that resources need to be available at all times; without failure. The standard practice is to have an entire copy of production systems online and available at a secondary physical location; with further redundancy built into each site to prevent total failure. This redundancy multiplies capacity requirements by a factor of two or four at minimum. Server redundancy levels depend on the availability requirements of the

data involved. A basic redundant server setup will have two servers, one that runs normally, and one for failover in case the primary node fails. In more advanced setups, this basic redundancy will be scaled across multiple locations, for a minimum of four nodes. This means that an entire location can fail, and the service will still be available. With a piece of technology that allows flexible relocation of services to workstations, there is as much redundancy as there are workstations. My research can demonstrate this flexibility; as any number of nodes can perform the same set of computing tasks; creating a redundant system.

This redundancy certainly creates more opportunities for users to get their computing tasks accomplished, but that may not always lead to reliability. If the user were to request information from the furthest away node every time, it would take longer to get the response they are looking for, and the system would feel slower and less reliable. This can be countered by using closer nodes for computation tasks, but the user may not know of their nearest nodes. Additional work needs to be done on this issue. The most common way to approach this issue is to send a request to each node and sort them by response times, but this isn't a scalable solution for a flexible network in which nodes can enter and leave at any given time. That being said, having many more nodes capable of handling the servers' tasks can improve response times from the server and allow it to work as intended under peak user activity, improving reliability. Overall, the redundancy improves reliability by providing additional nodes that can handle the servers' workload. As long as there are nodes (workstations) up and running, users will be able to access their resources.

Security Issues

Since peer-to-peer computing is a solved problem, I have focused on the security of scaling computing resources across arbitrary networks of nodes. The primary issue here is

confidentiality; as any node in the network could be used to leak information if used maliciously. A common research area in forensics is tracking and investigating malicious nodes in peer to peer networks. A proper, traceable validation system solves many of these issues.

The security problem of information leakage has proved to be difficult in highly secured environments. That is why many organizations with highly sensitive information are using their own private cloud environments, and Amazon created their own government-specific cloud for their government customers. “AWS GovCloud (US) Regions are subject to FedRAMP High and Moderate baselines and allow customers to host sensitive Controlled Unclassified Information (CUI) and all types of regulated workloads” [3]. The best way to combat this issue is to keep all data encrypted at all times.

For proper security, data needs to be encrypted in transit, at rest, and during computation. The first two are fairly straightforward, and are much easier to accomplish, than maintaining encryption during computation. These encryption considerations are discussed below. The added cost of this security comes in the form of computation complexity. By encrypting information at rest or in transit, we must add computation steps to every process ran through the distributed computer. This computational cost will slow down computing, and may increase power consumption of devices, reducing efficiency. For any application that requires higher levels of security, especially in a distributed design, this added cost is highly warranted and will be highly recommended. For less sensitive computation, it could be removed in order to increase efficiency.

For encryption in transit, my solution uses TLS protocols, such as TLS 1.3 via use of PKI. My use of PKI serves that each node (workstation) has its own peer-validated

certificate, managed by my software. This certificate can be used to encrypt the connections to other nodes and allows nodes to be easily and reliably verified by peers. For further verification, a node must be approved by a parent node in order to join the computing network. That is, it creates a certificate request, and has it validated by the parent node when it joins. This creates a traceable verification tree that can be used to track when a node joined the network and who let them in.

The security issue of protecting data at rest is typically solved by using strong encryption on files being stored on disk. This is generally accomplished using AES 256 with today's technology. However, my solution bypasses this issue entirely by not storing information on disk. It creates too much risk for a node to hold onto computational remnants such as this, so the nodes store information in memory instead. It is possible for a given node's memory to overflow and write some information to disk, but this would be part of a temporary cache, paging table, or swap space. In addition to this precaution, the FaaS component of the node, that runs requests, is always run in a Docker container.

The containerization of arbitrary computation tasks is critical to keeping the nodes themselves safe from compromise. If there is ever an issue in an individual container on an individual node, the Docker container can be thrown away and the node can create a new one. This keeps the node safe from arbitrary code execution exposure.

To address the issue of securing information during computation, homomorphic encryption can be implemented. I did not integrate this into my solution as it is an active research area of its own. Homomorphic encryption allows computation to be done on ciphertexts in memory, such that information is never fully decrypted when being processed. According to Maha TEBAA et. al., homomorphic encryption is additive, such that it can only

be proven with addition. Multiplication can be applied, but only once [7]. This means that applications of homomorphic encryption are possible, but inefficient. Because of the limitations of current research on the topic, it is impractical to apply to my solution. Therefore, to maintain the highest level of security, sensitive computation should only be done on private computation networks, if at all. This would be the final piece in allowing sensitive data to be run on a distributed computing network such as the one I have created.

CHAPTER 2. DISTRIBUTED COMPUTING

Peering Methodology

Since the focus of my research is resource allocation at an organization level, I used basic peering methods. To perform larger scale computing; at a global level, for example; protocols such as UPnP and STUN can be used to allow communication between nodes behind NAT points. This is how Microsoft performs their updates. “Windows 10 clients can source content from other devices on their local network that have already downloaded the updates or from peers over the internet” [4]. Using this model would allow for further expansion and improved scalability for my software.

My peering methodology requires new nodes to know about existing nodes in the network. The primary network nodes are Master nodes. They would be listed in a registry, that new nodes can access to join the computing network. To join the computing network, a new node will contact an existing node, likely one of the primary nodes listed in the registry and send its information to be validated. Once validated, the new node will be associated with nearby peers, and may be given code to run as FaaS tasks. Once it has been validated and peered, a user can contact it directly to perform requests. Notably, once validated, the peer will have no further need to contact the Master node unless it needs to be revalidated or unless the Master node is participating in the computation network.

Because of how the code works, any node could technically be used for validation, but involving the Master nodes in the initial validation process improves the integrity of the network. These nodes are deployed with a trusted certificate that is signed to identify them as Master nodes, indicating trustworthiness. Without this step, a malicious actor could deploy a

node, begin validating new nodes, and compromise the security of computation done by those nodes. Notably, this would not be easy, but it is a risk.

The simplicity of this design allows every workstation in an organization the be a part of the computing network. This model allows all users to query their own computer first, because localhost will always have the best latency. If a user's workstation doesn't know how to solve a problem, or access a specific resource, it can pass the request on to its peers. This may slow down the access, but it adds a layer of proxy to servers and adds at most two hops to the actual server to retrieve a resource or perform a compute task.

Further work could be done to expand the software to work at a global or multi-organizational level. As mentioned previously, the technology to do this is UPnP or STUN. UPnP, Universal Plug and Play, and STUN, Session Traversal Utilities for NAT, are both used for bypassing NAT points, such as firewalls or home network routers. Though they work differently, either protocol could be used to distribute my network globally or in a multi-organizational collaboration. The advantage of having a more widespread computing network is added scalability and redundancy. The drawback of this is the added security risk of allowing arbitrary nodes, potentially around the world, to access sensitive information. Because of this risk, the focus of my testing has been to create organization-level computing networks, so that each organization can control their own network. If opened to the world, any computer on the internet would be able to participate. By opening computation to anyone, it becomes impossible to track who has temporary access to information used during computation. An example of this that involves high-risk would be payment processing. This impact could be mitigated by limiting the scope of sensitive computation tasks to internal use only, like is done on servers today. Without mitigation, it would be challenging for an

attacker to gather information, but a motivated attacker could do it. A real-world example of this is the abuse of the Heartbleed vulnerability. Attackers gathered small bits of information from websites and reassembled the fragments to find sensitive information on the webservers. [16]

Performing Computing Tasks

At a fundamental level, computing tasks are performed as microservices. These microservices are created by parsing the user-uploaded code and creating RESTful APIs for them. Since all user functions are served as microservices, they are accessible from any HTTPS client. This makes the platform very user-friendly and open; allowing any user to easily access network compute resources. As mentioned above, these nodes may also be the workstations users are using anyway, meaning they have even easier access to resources than accessing a nearby peer.

Distributing Tasks

Since all nodes function in the same fashion, performing tasks as microservices; one node can call another node to perform it's given computing task input by a user. This redistribution of tasks allows a user to interact with the node nearest to them (based on network latency) and have other nodes assist with the computing tasks.

Tasks are distributed using a weighted random selection algorithm that uses weights based on performance. If a node performs tasks quickly, it will pick up more of them. If multiple nodes have similar performance, they will receive a similar number of tasks; giving more value to more complex tasks. Effectively, a node can complete multiple small tasks to equal one more complex task. This task distribution model is rudimentary and functions as a mean to distribute tasks to nodes, but a more effective model would be to implement proof of

work in order to monetize the computation and reward compute nodes for their contributions to the network.

Securely Distributing Sensitive Data

When handling sensitive data, it must be secured in transit and at rest. These are fundamental ideas of the CIA triad. This can be difficult for a computing platform, but there are many options for securely handling data in a distributed system. The firm precaution that was taken was encrypt data in transit from the user to the network and between nodes in the network. This was implemented with HTTPS because it ties in well with the node verification component as both use PKI. The certificate used to validate the authenticity of a node can be used to encrypt the data with TLS 1.3 in transit.

Encryption at rest is not needed with the FaaS implementation because data is not stored locally on the nodes. Data is stored in memory only, when it is being processed by the node; and does not need to be stored on disk. To protect data while it is on the node, the functions are executed in Docker containers. This separates the data from users with normal permissions on the host. It is worth noting that an advanced user with administrator privileges can dump memory from processes running under this architecture. This is ultimately a risk that needs to be mitigated by storing as little information on the node as possible. If information is highly sensitive, it should be processed in a more secure environment.

To protect the node, which may be an organization's owned device, from malicious execution, all arbitrary code is executed in Docker containers. As mentioned previously, this creates a degree of separation from the user, so they cannot interfere with data being processed. In addition to this benefit, the containerization prevents malicious code execution.

The issue with running arbitrary functions on an arbitrary node is that the code may try to take over the host or install other potentially malicious software on it. This is a huge risk in a distributed computing network, so protecting the nodes is important to maintaining the integrity of data processed with its capabilities.

By protecting the data processed by nodes and protecting the integrity of node's and distributed computing software, we create a safe environment for utilizing additional compute resources on arbitrary devices. This helps to solve the underutilization and scaling issues created by modern IT infrastructure implementations.

CHAPTER 3. COMPUTING STRUCTURE

Function as a Service

Since computing tasks are served as microservices, the most effective method to serve them is using a FaaS model. FaaS is a new concept introduced with cloud technologies. It is a model that allows users to define actions for a server to use. The cloud provider will typically handle all other parts of the computation stack. I chose to use this model because it allows the nodes in my distributed computer to serve arbitrary functions of code with arbitrary inputs from the user.

The FaaS implementation allows users to upload code to the network of compute nodes and run it from any node. When code is uploaded, the first node parses out functions and structs and exposes them as microservices. Then it passes working, compliable functions to peers in the network. At this point, the user can call their functions as microservices. Once a user calls a function, the requested node may pass it onto a peer if it is busy with other computation tasks.

This model was used to create an efficient, scalable platform for distributed computing. Since multiple nodes can share a load of tasks and distribute the load on a per-function-call basis, the platform provides reliability and scalability.

To apply this technology to the problem of underutilization, we consider the option of using desktop computers as nodes in an organization. An organization's desktop computers could contribute a small amount of their resources to create a fairly sizeable pool for the organization's servers to use as a supplement under strain. That is, when the servers become overwhelmed with complex computing tasks, they can use the distributed computing network of desktop computers to share the load. This utilizes unused resources in the organization and

allows better performance for customers during peak operation without requiring massively scaled and dedicated customer-facing systems. This model is effective, because each desktop computer will perform a small subset of the customer tasks without compromising the security of information transmitted to or processed by the node.

Distributing Tasks

The method used for distributing tasks is weighted-average task forwarding based on complexity and response time. The weighting is biased for nodes to complete tasks themselves; and tasks are only forwarded once at most. Since I am using a proxying methodology for distributing tasks, we can look at Crowds and Mix-Nets as alternatives to my chosen solution. These alternatives are focused on anonymity, so they are not a perfect fit, but they accomplish the same forwarding of requests. In comparison with these distribution algorithms, mine is more efficient in terms of computation because it limits the number of hops to two. By limiting the number of redirects, we can maintain a reasonable level of efficiency. Because of the microservice infrastructure, the total computation time can vary substantially based on network conditions and the environment in which the software is deployed. In my test deployment, basic tasks were completed in around 50-90 microseconds. This was far better than I expected, but that was all on a single machine. For normal deployments, I would expect the request structure to add anywhere from 100 milliseconds to a full second to computation tasks.

From the node side, there will be little overhead in running the node service outside of performing computing tasks. The microservice is a small web service that is compiled with Golang, making it very lightweight and thus efficient. The docker component that runs on the backend will consume more resources, however. Ideally, monitoring will be put in

place to minimize the impact of resource usage for end-users. The goal of my application is to use excess capacity, such that my distributed computing network is unnoticed by normal users, and they may can be unaware of its usage on their machines. In practice, the overhead of running the web-server component of my solution did not create a significant impact for the host. However, scaling up computing usage can cause concerns with this model, so restrictions do need to be in place for production use.

CHAPTER 4. PEER VALIDATION

Public Key Infrastructure

With multiple ways to validate nodes, there were several advantages of using PKI. These advantages include low resource consumption, simple interfacing, universally verifiable, data integrity, and encryption. According to Cisco, "Public Key Infrastructure (PKI) offers a scalable method of securing networks, reducing management overhead, and simplifying the deployment of network infrastructures" [5]. This technology is extremely popular with web technologies and is the base for HTTPS. This makes it a good fit for my web-based application framework. Since my computation requests are performed using web technology, the same certificate for the web application is used for peer validation. This creates a layer of transparency for users to see that nodes are validated. To accomplish this, the consumer would simply need to have the Master node certificate trusted by their browser.

Blockchain

Notably, blockchain is a strong competitor for validation in this use-case and there are firms using the technology for their enterprise-grade solutions. With blockchain, it is very easy to validate nodes and computing tasks. However, blockchain is a new and complex technology; so it may have been difficult to integrate into my project.

An issue with Blockchain validation is that as the Blockchain grows, it takes more time to fully validate. In practice, shortcuts can be taken to limit this drawback, but there is still the risk of a 51% attack. A 51% attack being the situation where a malicious actor or group of malicious actors takes over the validation process to compromise the integrity of data written in the Blockchain. This is possible because it is a consensus verification process,

and a 51% group would comprise of more than half of the network and thus be able to decide on what makes it into the Blockchain. This would allow them to modify the Blockchain or control new entries.

CHAPTER 5. COMBINATION OF TECHNOLOGIES

The Combination

This combination of technologies aims to solve under-utilization and large capital expense problems discussed previously in this thesis. These issues are faced in every organization with a technology presence. Obviously, the large capital expenditure of a single laptop in a restaurant does not compare to a full desktop replacement period in a large corporation. And likewise, the benefits of my research are not as significant for a restaurant as a large organization. This is because of the large technology footprint of these larger organizations. That being said, my combination of technologies can be deployed on any substantial corporate network to ease capital expenditure needs and allow the organization to properly utilize their technology resources.

Here is a list of technologies I have included in my project:

- Golang – Programming language chosen for concurrency and efficiency.
- Web APIs – Microservices that allow implementation of FaaS.
- FaaS – Function as a Service design pattern.
- PKI – Public Key Infrastructure used for peer validation.
- TLS – Encryption between clients and nodes (HTTPS).
- Docker – Used to protect nodes from arbitrary code injection (malicious clients).

The platform was written in Golang, using web APIs for communication between the user interface and backend, and between the nodes on the backend. This platform implements FaaS to serve client code as functions back to them, so they can run programs on the network

of nodes running the code. PKI, TLS, and Docker were added for security in order to protect endpoints running the mesh network, and to protect the clients running code on the network.

In a corporate environment, this combination of technologies could be deployed to all hosts on the network, and servers that run out of computing resources can utilize the network of hosts to serve functions. This would reduce the strain on an overworked server and allow the organization to refrain from upgrading too early. Upgrading early to scale and meet demand is one of the largest reasons for huge IT capital expenditures for many organizations. Financial institutions, for example, require highly redundant servers with low latencies for quick and reliable financial transactions. Traditional banks require redundancy to keep their services available to customers, so their customers can do business and generate revenue for the banks. If these systems lack redundancy and fail, the bank and customers can both lose out on money-making opportunities. To maintain streams of revenue and customer relations, it is standard for financial institutions to make their systems redundant and fail-proof. For other financial institutions that participate in high-frequency trading, low latency is key. "Since lower latency equals faster speed, high-frequency traders spend heavily to obtain the fastest computer hardware, software, and data lines so as execute orders as speedily as possible and gain a competitive edge in trading" [8]. This competitive edge allows HFT firms to make money. Firms like Tradebot rely entirely on systems like this and failing systems could cause firms to lose their ability to generate revenue while down. As the example, Tradebot states "Technology is our weapon" on their website [9]. Other organizations, of course, have similar needs, and can also benefit from having extra computing power made available from the mass of workstations they have already purchased.

As for users, if their own workstations, or workstations on their subnet can serve content in place of a centralized server, the connection will be faster and they will have a better user-experience as well. Notably, by spreading functions to workstations, they can also be made redundant more easily. If one host goes down, others can fill its place. This is a perfect example of the adage "Treat servers like livestock, not pets." By applying this adage to workstations in addition to servers, we can expand the herd and increase our technology productivity yields. This increase of availability and technology productivity can easily benefit users in an organization, as applications will be more responsive and reliable, creating a better user-experience for internal technology customers. For example, in my industry, insurance; this would be our Claims and Underwriting departments, because they need technology to do their jobs.

Since workstations can be desktops, laptops, or thin clients, some considerations can be made for each case. Desktops are the most likely target for usage of my solution because they are not impeded by the same constraints as laptops and thin clients. The most important consideration for laptops is power. I would not recommend deploying any solution like mine on a battery powered device because of parasitic battery drain. While plugged in, the software could be used safely like a traditional desktop workstation. Notably, if used on a laptop, heat can also be an issue. Laptops typically do not have the same amount of cooling power as desktops, so they can retain heat more easily. By increasing resource usage (CPU, RAM, HDD, etc), a laptop could face heat-related issues. Finally, thin clients are a unique case as well. Some thin clients have enough excess computing capacity that they could easily become nodes in a computing network. Others, however, have very limited specifications and would not make good candidates. Therefore, I would primarily apply my solution to

desktop computers, and apply special considerations to laptops and thin clients. If applied to laptops, I would only run the software while plugged in; and if applied to thin clients I would make sure they have enough resources available to be effective nodes.

Benefits

There are many benefits to having a mesh computing network within an organization. Worth mentioning here, these benefits could be seen at a global scale for global organizations, or global collaborations between organizations.

Here are some of the benefits of my combination of technologies:

- Scalability
- Redundancy
- Low Latency
- Reduced Capital Expenditures
- Better Resource Utilization

Drawbacks

Though there are many benefits to my tool, there are some drawbacks. The primary drawback to address is the size requirement. My solution may be completely ineffective in a small organization; and may be hard to scale properly in a larger organization where benefits can be realized. A special consideration can be applied to human capital requirements. Since it's a new framework, developers would need to be trained to apply the technology to existing software solutions and build in functionality with new solutions.

Some other drawbacks to consider are the following:

- Overwork Workstations

- Potentially Reduced Reliability
- Potentially Increased Latency
- Function Management
- Cost Allocation (who is spending what)
- Improper Fit
- Human Capital

Uniqueness

This research may seem like something that has been done before, but I could not find any evidence of it being done effectively. To be discussed in more detail below, the primary alternatives to my solution are SONM [6], a fog computing network; and the cloud. Both offerings have strong potential, but each solution has challenges.

My solution is different from SONM because it does not rely on blockchain technologies and infrastructure to run. In addition to this dependency, SONM is run on a single global network instead of having the potential to be unique to each organization like my solution [6]. This effectively shows that my solution is more targeted, but more useful because of the flexibility of the design.

The only benefit my solution has over traditional cloud offerings is that it can be deployed on existing hardware an organization owns; and it can interface with cloud offerings to offer even more capacity and scale. That being said, my code runs a lot like Amazon's Lambda and Microsoft's Azure Functions and there is potential for cross-integration. That is, an organization could rely on their owned hardware first and use cloud computing as a backup for when workstations are offline or do not meet demand for function calls. This has been described as Edge Computing in recent years; and my software would be

a great solution to enabling an organization to create a real Edge Computing environment. Edge Computing environments are designed to place resources closer to users. [15] These systems are widely spread in order to reduce the latency between server and client. In recent years, the Internet of Things (IOT) has become popular, and Edge Computing has been a significant contribution to the wide expanse of internet connected devices. [15] This is because IOT devices require an internet connection and rely on servers for control, monitoring, and updates. Because of this, my research could be applied to Edge Computing in order to improve the usability of IOT devices.

Though there are alternative solutions, and they have similar aspects to my research; I believe my solution is unique enough to be deployed independently and could be used to create direct competition to even the primary alternative solutions.

CHAPTER 6. SOLUTION EVALUATION

Evaluation Criteria

To determine the effectiveness of my solution I tested the software I wrote on my home network using three nodes. The goal of my experiment was to be able to demonstrate running a simple Golang program on multiple devices. Each node should be able to upload code, discover and validate peers, and share code with peers. Then the user should be able to run code on the network from the interface and APIs of a single node. Some simple assumptions can be made, such as: clients only connect to secure nodes, nodes only accept code from trusted peers, and computation requests are forwarded at most one time.

Experiment Steps

1. Install Golang on host
2. Download code from private Github repository
3. Compile source code into binary
4. Run binary on each host
5. Upload code to one host via web interface
6. Run code via API on origin host
7. Record response from API
8. Repeat steps 6-7 several times
9. Check logs on each node and verify computation is distributed and responses are accurate

Evaluation Results

This experiment yielded results as expected. I was able to upload a file of Golang source code to a single node, with multiple functions. Then I was able to run each function independently as a service. To more easily verify that code was distributed, one of the functions returned the compute node's hostname. By running this function multiple times, it was easy to compare results to show that other nodes participated in computation. I was able to reuse this test when verifying new code uploads because when the Docker containers were rebuilt they got a new, randomized hostname.

Notably, the compute times were very fast and efficient; but this is most likely due to being done on a single local network. On a broader network, such as a corporate WAN or the internet, response times would be greater and the network would not be as efficient. Further testing would be required to verify this, but it would come down network latency and the application's requirements to determine usability.

As far as functionality, I noticed no impact when I added Docker to the technology stack other than obfuscating hostnames. This would be an easy issue to fix, but I prefer the randomized container name to revealing the host's actual name. Adding Docker did not significantly impact the computing ability of the host, but it was a small-scale test. I believe this model would add significant computing overhead at scale, but it would still be less impactful than deploying the framework using a full virtualization stack.

Lessons Learned

By experimenting with my ideas and the solution I created, I was surprised how efficient the computation was on my local network, how effectively my Golang shenanigans was able to recompile itself on the backend without usability issues, and how efficiently the

certificate validation component acted. It was challenging to implement some of the components of my application, but the language's computing and concurrency capabilities made it worthwhile.

The first lesson learned fits the description of Edge Computing. The low latency of local computing worked very well for my solution, and the idea is closely related to the field of Edge Computing. According to the Wiley Series on Parallel and Distributed Computing there are around 50 billion devices connected to the internet. And with the massive amount of data flowing through the internet backbone, we need to pay attention to latency between users and servers. [15] While the internet has a massive backbone with substantial bandwidth, I believe the authors and editors of this series are correct in that we will hit a limit of bandwidth as more devices are connected and more data-intensive services are consumed. By placing my nodes closer to users than origin servers, we can save on bandwidth and improve response times. This helps to solve big data bandwidth issues on a large-scale and improves the user experience on the small-scale end. According to the same authors, a fog computing design applied to edge computing could be used for brokerage services (like MQTT), data analytics engine, or as a traditional server. [15]

Since Golang is a compiled language, I had to take additional steps to allow users to upload code and have the web server component of the node implementation to compile it on the fly for immediate user interaction. I was able to accomplish this with metaprogramming and implementing some basic Docker container management as part of my node code. When I created this solution, I knew it would work in theory, but I was still surprised how quickly it was able to accomplish code uploads and propagation during my experiment.

Finally, I learned how efficient using PKI for peer validation really is. For my implementation, it greatly reduced the computational steps and time required to validate peer nodes like other decentralized networks.

CHAPTER 7. MAJOR CONTRIBUTIONS

Peer Validation with PKI

The hierarchy I created for my decentralized computing network based on PKI is a major contribution because it is an efficient way to establish trust between nodes. Traditional methods of establishing trust between nodes can require multiple, complex computations that must be repeated often to ensure that trust is maintained. By reducing the number of computations and frequency of reverification, I am able to provide a more efficient method of verification.

In addition to this efficiency, it is worth mentioning that nearly all computers have PKI built in because of web browsers and software verification programs. This makes my contribution widely useable at scale in any number of environments.

Arbitrary Golang FaaS API Implementation

This contribution is incidental, but since Golang is a compiled language I consider this to be an important part of my project because it is unique and was challenging to design. Being able to accomplish this flexibility with a compiled language is a major contribution because it combines the efficiency of a compiled binary with the flexibility of an interpreted language.

This contribution allows me to upload code to a web server, compile it, and serve it as functions via API calls in a single user action. This simple use of web APIs expands the functionality of the compiled language Golang. If used independently of the rest of the project, it could be applied to testing environments to more efficiently test new Golang code.

Decentralized FaaS Computing

This contribution has been done by others in a few specialized cases, but since the field is new and my implementation is unique, I believe it is significant. Most FaaS implementations are cloud-based, so there are some similarities, but at the end of the day the user must always contact the cloud to get a response. By making my solution decentralized and giving the user the ability to run their own code if they choose, I have created an even more flexible option. This contribution may be valuable to field of Edge Computing because it could be deployed on any number of workstations, servers, and cloud hosts to provide an enterprise a widely spread computing network that provides low latency access to information and the redundancy of a highly distributed computing model.

This computing model could be used as part of an edge computing environment. As discussed earlier, this usage would provide capability for data analytics, IOT connectivity, brokerage services, or other traditional server functionality. There is already a lot of research in this environment, but it is fragmented; so, a platform like mine could unify the research space and reduce the barrier to entry for new competitors.

CHAPTER 8. ALTERNATIVE SOLUTIONS

Fog Computing

Fog computing has been talked about in many groups, but it has not been implemented effectively. “SONM is a powerful distributed worldwide system for general-purpose computing, implemented as a fog computing structure” [6]. SONM is marketed as a direct competitor to cloud offerings. Their primary offerings are IaaS and PaaS. The way their decentralized fog computing network operates is that miners are performing real computing tasks for customers. It’s built on blockchain and mining technologies but offered as a computing platform.

The purpose of the SONM technology is to offer a decentralized marketplace for compute resources. An interesting feature of this technology and marketplace is that it scales to meet demand and allows miners to select more profitable tasks automatically. This is a fundamental concept in economics, so it is an absolute requirement for a project like this to succeed.

Though SONM has some good ideas and a solid foundation to build a future in the cryptocurrency market, there are many problems with this solution. Here are some of the primary issues that are not addressed with a decentralized fog computing technology:

- Lack of Demand
- Lack of Supply
- Infrastructure Design
- Fraudulent Claims

Though SONM has a lot of issues, they are working through them. It is worth noting some of the merits of their ideas and technologies. Here are some of these merits:

- Anonymity
- Peer-to-peer Networking
- On-Demand Resources
- Compute Isn't Wasted (like other blockchain applications)

Overall, fog computing and the example SONM have many upsides, but the technology available is not designed for organizational use. Because of the reliance on Blockchain and the basis of cryptocurrency and anonymity being so prevalent these fog computing networks are targeting a completely different audience. The blockchain infrastructure backend creates a lot of overhead and eliminates FaaS as an option with current technology. The SONM example uses Smart Contracts, which take extra time to validate, reducing the effect of having resources on demand. "Blockchain smart contracts run all nodes on blockchain simultaneously, and it's quite ineffective way of wasting resources. Moreover, if we consider some big data processing tasks, they hardly could be implemented on blockchain" [10]. Directly from the source, SONM admits that Smart Contracts are inefficient for large computation because of the resource overhead of the blockchain. They are doing their own research into other blockchains to find a more efficient means of transacting Smart Contracts. "Distributed consensus could be implemented, but Ethereum is too slow, a fast blockchain or DAG consensus algorithm with smart contracts is to be chosen and adopted" [10]. Any latency in this validation is a significant detriment to the usability of the network as is, and even more if it were to support FaaS.

Traditional Client-Server Methods

In this rudimentary model, everyone has a workstation and connects to internal servers. Organizations have been deploying systems using this model for many years, so it is still a very popular computing model. I would say this also applies to mainframe technologies; though modern server stacks are much more reliable and scalable than a traditional mainframe.

The benefit of this model is that it gives the organization complete control over the hardware they purchase, and complete control over the software they choose to run on the hardware. This additional control gives leaders peace of mind, so long as they trust their team. If their administrators are sufficient; the team can create a very secure and reliable infrastructure for internal staff and external customers to rely on. An additional benefit, that is becoming less important as cloud technologies and the internet backbone become stronger, is low latency. The latency of an employee to an internal system is typically much lower than the same connection being made to the cloud because of geographical distance.

The drawback of this approach is that it does not scale smoothly. It requires large capital expenditures periodically to maintain and upgrade functionality of legacy systems [13]. That is, it tends to be slow and costly to replace systems; and many organizations are running systems far longer than originally intended. The impact of these large capital expenditures is shown in the chart on the following page. In my previous organization, they were still dealing with Windows XP and Windows Server 2003 just a few months ago. In my current organization, they dealt with these systems just before I started. The primary difference here is that my former organization was government and had less funding for IT

upgrades. This lack of funding for capital expenditures becomes a huge problem with traditional client-server setups.

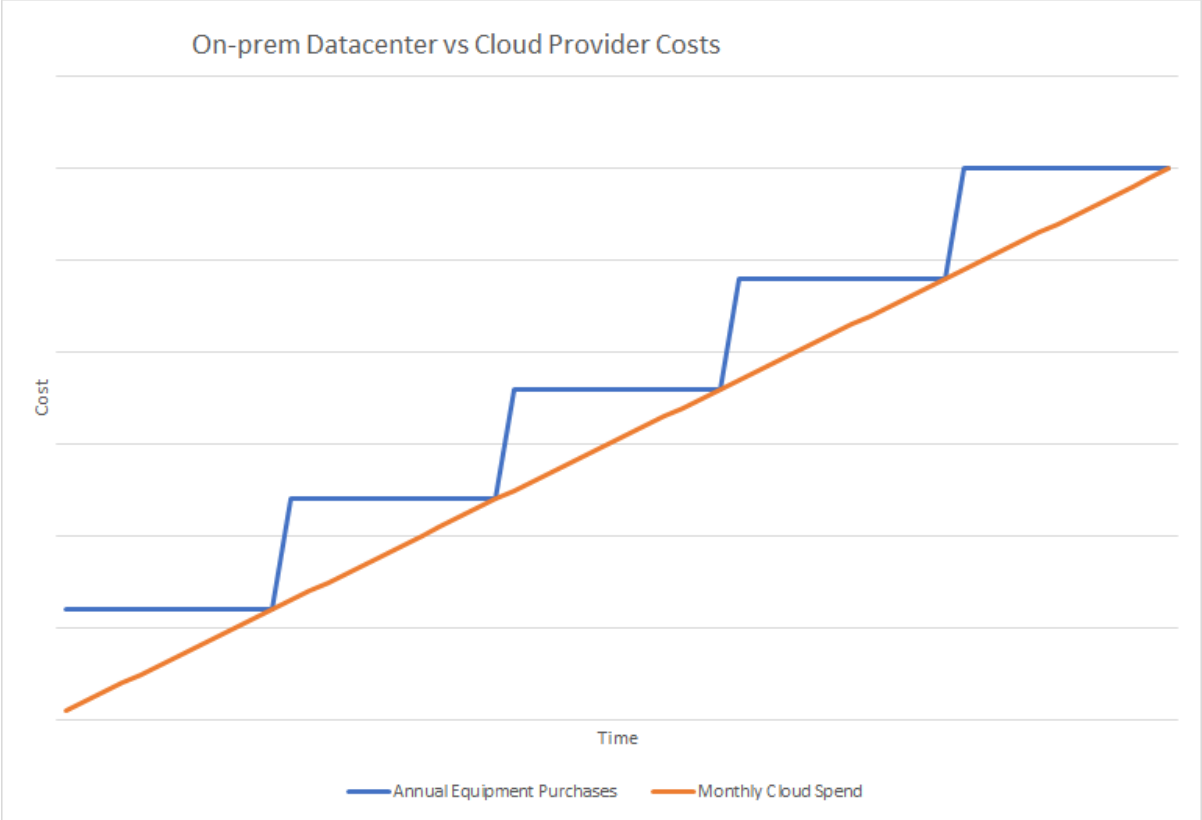

Figure 1. Cost comparison of on-prem datacenter vs cloud provider.

Cloud Computing Models

This approach is very new and involves selling services in bite-size chunks instead of selling large suites under long contracts like legacy systems have been sold previously. This low barrier to entry makes it possible for organizations and even individuals to dip their toes into the cloud in order to learn about its benefits before making a large move. Amazon AWS has more than 200 services available as of 2019. [14].

The primary advantage of this approach is that it scales much more linearly than the traditional client-server methodology. Customers can choose the amount of a service they

want to pay for and have it available instantly; and if they only need it for a short period of time, they can stop easily as well. A good use of this type of scalability is training machine learning models or scaling up a web service to outpace a Denial of Service attack. Both efforts require a large amount of additional computing power for only a short period of time. Another significant advantage is managed services. Most customers are opting to use managed services to drastically reduce IT staff overhead. If they don't need as many people to run updates on operating systems and install webservers or manage databases; they can spend that time or money elsewhere.

Some drawbacks of this infrastructure are lack of expertise, cost, and vendor-lock-in. Some leaders are hesitant to migrate to cloud technologies because they are relatively unproven, and their employees don't know enough about implementing and maintaining services in the cloud. With this lack of expertise, some organizations are "forklifting" their services to the cloud in an effort to eliminate their need for a datacenter; and this can become very cost ineffective. According to the CTO of Dell, John Roese, "[Cloud is] way more expensive than buying from us" [11]. To support this argument, Dropbox moved from AWS to their own infrastructure in 2018 and cut operating costs by nearly $75 million [12]. Without meeting certain efficiency thresholds, cloud services can quickly become more costly than legacy solutions. An additional problem with this migration and cost, is vendor-lock-in. Vendor-lock-in is defined as "a situation where a customer may be unable to leave, migrate, or transfer to an alternate provider due to technical or nontechnical constraints" [17. Basically, when an organization starts using services from one cloud vendor, they may have a hard time switching to another vendor or moving back to an on-premise datacenter; and this can be risky for large organizations.

A new setup that organizations are using with the expansion of cloud computing is thin clients. Instead of having powerful workstations, most users in an organization can have thin clients that use web applications and virtual desktops to access resources that allow them to do their jobs. Combined with cloud technologies, this can be done in a widespread deployment.

REFERENCES

[1] Weinman, Joe. (2012). "Cost of Excess Capacity" in *Cloudonomics: The Business Value of Cloud Computing*. [E-book]. Available: https://www.oreilly.com/library/view/cloudonomics-the-business/9781118282885/xhtml/sec108.html. [Accessed Sept. 14, 2019].

[2] Sullivan, Nick. (2016). Introducing TLS 1.3. [Online]. Available: https://blog.cloudflare.com/introducing-tls-1-3/. [Accessed Sept. 15, 2019].

[3] Introduction to the AWS GovCloud (US) Regions. [Online]. Available: https://aws.amazon.com/govcloud-us/. [Accessed Oct. 5, 2019].

[4] Mariano Gorzelany, Andres; Kim, Ashley; Hermansen, Baard; Lich, Brian; Halfin, Dani; Gallagher, Ed; Lindsay, Greg; Ondrusek, Jaime; Decker, Jeanie; Poggemeyer, Liza; Brower, Nick. (2019). Optimize Windows 10 update delivery. [Online]. Available: https://docs.microsoft.com/en-us/windows/deployment/update/waas-optimize-windows-10-updates. [Accessed Dec. 4, 2019].

[5] Cisco IOS Public-Key Infrastructure: Deployment Benefits and Features. [Online]. Available: https://www.cisco.com/en/US/tech/tk583/tk618/technologies_white_paper09186a0080179739.shtml. [Accessed Dec 4, 2019].

[6] About SONM. [Online]. Available: https://docs.sonm.com/. [Accessed Dec. 8, 2019].

[7] El Hajji, Said; El Ghazi, Abdellatif; Tebaa Maha. (2012). "Homomorphic Encryption Applied to Cloud Computing," Proceedings of the World Congress on Engineering 2012 Vol 1 WCE 2012. [Online]. Available: http://www.iaeng.org/publication/WCE2012/WCE2012_pp536-539.pdf. [Accessed Dec. 10, 2019].

[8] Picardo, Elvis. (2019). Understanding High-Frequency Trading Terminology. [Online]. Available: https://www.investopedia.com/articles/active-trading/042414/youd-better-know-your-highfrequency-trading-terminology.asp. [Accessed Dec. 10, 2019].

[9] BEAT WALL STREET FROM KANSAS CITY. [Online]. Available: https://tradebot.com. [Accessed Dec. 12, 2019].

[10] White-box application. [Online]. Available: https://docs.sonm.com/home/use-cases/white-box-application. [Accessed Dec. 13, 2019].

[11] Wagner, Mitch. (2018). Dell CTO: Public Cloud is 'Way More Expensive Than Buying From Us,' in *CLOUD MAKING BUSINESS LIGHTER THAN AIR*. [Online]. https://www.lightreading.com/enterprise-cloud/infrastructure-and-platform/dell-cto-public-cloud-is-way-more-expensive-than-buying-from-us/d/d-id/741523. [Accessed Dec. 20, 2019].

[12] Houston, Andrew. (2018). Dropbox, Inc. FORM S-1 REGISTRATION STATEMENT. [Online]. Available: https://www.sec.gov/Archives/edgar/data/1467623/000119312518055809/d451946ds1.htm. [Accessed Jan. 4, 2020].

[13] Mitchell, Matthew. (2019). SHIFT HAPPENS LEADING THROUGH CHANGE. *AWS Innovation Day*.

[14] Gustafson, Chad. (2019). AWS Cloud Security and Networking Fundamentals. *AWS Innovation Day*.

[15] Buyya, Rajkumar; Srirama, Satish Narayana. (2019). Fog and Edge Computing Principles and Paradigms. *Print.*

[16] Synopsys, Inc. (2014). Heartbleed Bug. [Online]. Available: https://heartbleed.com/ [Accessed Mar. 19, 2020].

[17] O'Hara, Brian; Malisow, Ben. (2017). Certified Cloud Security Professional Official Study Guide. *Print*.