A Survey Of Distributed Systems Software Options

John Ford, Karely Rodriguez, Kevin Offemaria
Vanderbilt University, john.c.ford, karely.m.rodriguez.galvan, kevin.p.offemaria@Vanderbilt.Edu

Abstract – **Distributed systems are becoming increasingly important in modern software development, but the coordination and management of such systems can be a complex task. To address this, several software technologies have emerged to provide distributed coordination and configuration management, including Zookeeper, etcd, Hashicorp Consul, and Traefik Proxy. This paper presents a survey of these technologies, comparing their features, strengths, and weaknesses, as well as evaluating their performance, scalability, fault tolerance, and ease of use. The paper also explores their use cases, integration with other software tools, and available client libraries. The survey aims to assist developers and system administrators in selecting the best distributed coordination and configuration management tool for their needs.**

Index Terms – etcd, consul, traefik, distributed systems, raft consensus algorithm

Introduction

The importance of distributed systems has significantly increased in recent years due to the growing demand for scalable and reliable applications. Many everyday technologies – from cellular networks, to airline flight control, to rideshare dispatch software – depend on distributed systems. However, developing and managing such systems is a complex task, requiring careful attention to the coordination and synchronization of distributed components. To tackle these challenges, various software technologies have been developed to provide distributed coordination and configuration management, such as Zookeeper, etcd, Hashicorp Consul, and Traefik Proxy.

This paper presents a survey of these four software technologies, which have gained popularity in recent years, and compares their features, strengths, and weaknesses. We evaluate their performance, scalability, fault tolerance, and ease of use, along with their support for various features such as service discovery, distributed locking, and dynamic configuration updates. Additionally, we discuss the use cases of these technologies, their integration with other software tools, and available client libraries. Our survey aims to provide a comprehensive understanding of these distributed coordination and configuration management tools, assisting developers and system administrators in selecting the best technology for their needs.

ETCD

etcd is an open-source distributed key-value store used to store and manage information for distributed systems. It is strongly consistent making it suitable for storing a system's state data such as configuration and metadata. The name is taken from the */etc* folder in Linux operating systems where all configuration files are stored, appended with the letter *d* to distinguish its usage in a distributed manner, hence *etcd* [1]. A key property of etcd is that all of its nodes, housing key-value stores, are fully replicated and highly consistent. This is achieved by using write-ahead logging (WAL), a technique popularized in database systems where persisting data to disk is paramount [2]. The purpose of WAL is to ensure that changes to the data are recorded in a log before they are written to disk. Therefore, etcd is used as the single source of truth in distributed systems. It can tolerate network partitions and hardware failures making it highly available in the event of catastrophic events.

The Raft consensus algorithm is responsible for ensuring strong consistency in etcd where its node cluster maintains a replicated log of transactions [3]. The log is divided into terms where each one has a leader node responsible for accepting client requests and replicating them to the other nodes in the cluster. A client's write request is forwarded to the current leader node who appends the transaction to its local log then distributes the information to other nodes in the cluster for replication. Once a quorum of nodes has acknowledged the transaction, the leader node sends a response to the client indicating that the transaction has been committed. In the event of a node failure, the rest of the nodes in the cluster continue to make progress as long as the system quorum holds. New or replacement nodes spun up by the controller use the disk logs to synchronize itself to the most recent state. If the leader node fails, a leader election process is initiated. The newly elected leader node ensures that all previously committed transactions are replicated to its followers, ensuring that all nodes in the cluster will have a consistent view of the key-value store. Ultimately, the Raft consensus algorithm ensures that all nodes in an etcd cluster will eventually have a synchronized key-value store with all transactions replicated across all available nodes.

The API for etcd is designed to be RESTful, utilizing HTTP methods to interact with its data store consisting of basic CRUD operations [4]. Various monitoring features such as watch mechanisms can be invoked to observe key-value changes in real-time. Another useful feature is the ability to query a range of keys returning a subset of key-value pairs. More advanced features like transactions allow multiple data store updates to be grouped together and executed atomically as a single transaction, ensuring all operations are either successful or rolled back. The leasing feature allows clients to acquire and release locks on data, which helps coordinate access to shared resources in a distributed system. Several additional features include backup and restore snapshots, cluster management, service discovery, and cluster health monitoring, among others.

etcd provides comprehensive security protocols including Transport Layer Security (TLS) data encryption and client authorization and authentication to guarantee confidence in a secure environment [5]. TLS provides secure communication between nodes and clients ensuring all data is encrypted and authenticated to protect against malicious actors. For data at rest, etcd supports a variety of encryption mechanisms such as disk encryption and file system encryption. Disk encryption protects the entire disk by encrypting data before it is written to disk, while file system encryption provides protection at the file level. A variety of authorization and authentication mechanisms allows the system to utilize Role-Based Access Control (RBAC) which grants administrators the liberty to define roles, permissions, groups. This provides a granular level of control to users. TLS protocols are designed to be flexible and are enforced in all API calls while still maintaining high performance.

All these properties make it a suitable choice as a core component of Kubernetes, an open-source system for automating deployment and management of containerized applications. etcd is considered to be the "brains" of a Kubernetes cluster because it reliably holds its entire state – every event, pod, customer definition, etc. As such, it appropriately fits in the control plane layer of the Kubernetes ecosystem along with other top layer components including cloud controller manager, scheduler, and API server [6]. etcd's modular architecture complements Kubernetes to be highly available by providing reliable data storage and consistent data access across all nodes in its cluster. The Raft consensus algorithm provides strong consistency and transactional updates to keep the data store serialized, thereby avoiding conflicts or race conditions. etcd's watch mechanism enables Kubernetes to monitor changes on configuration data in real-time which allows the system to quickly respond to cluster state changes ensuring a desired state. A distributed nature makes etcd highly scalable, allowing Kubernetes to support large-scale clusters with several thousands of nodes.

etcd is fully supported by the Cloud Native Computing Foundation (CNCF) as a graduated project with a vibrant open-source community dedicated to continually making improvements and maintaining the product [7]. It is written and compiled in the Go programming language making it highly performant with a strong emphasis on concurrency and parallelism. These are key characteristics that are suitable for distributed systems. Moreover, the etcd API is also available in other languages such as Java, Python, Ruby, and others. Ultimately, etcd's key features allows for a highly scalable and reliable key-value store that provides a variety of powerful features for managing distributed data. It takes the best of all sides of the CAP theorem in terms of providing strong consistency, high availability, and partition tolerance in building distributed systems.

CONSUL

HashiCorp Consul is a service networking platform that provides a service mesh, service discovery, configuration management, segmentation, and API gateway functionality across a distributed network infrastructure. “Consul is designed to be highly secure, highly scalable, and fault tolerant.” It can connect 50,000+ microservices” [8].

Service discoveries often traditionally lack a dynamic auto-discovery as such communications occur through a series of networks [9]. Though utilizing a central service registry, Consul is able to overcome these disadvantages through maintaining a real time registry of connected services and information which may be freely accessed by all services comprising the registry [10]. Controlling traffic flow is a challenge for distributive systems; a lack of verification from trusted parties is also a challenge. Consul overcomes this through service graphs, identity enforcement TLS, and load balancing [9]. Consul offers security through intentions which control access and permissions as different services communicate among each other; these intentions are enforced by the sidecar proxy or proxies. The authorized ingress and egress communication is established by specifying intentions.

Key/Value Store (KV) is a critical function of Consul and is very common in service discovery tools. KV allows for the indexing of stored objects such as configuration parameters and metadata. Though it is located in servers, KV can be accessed by either client or server agents. Access through the HTTP API and the UI is also possible. Privileges may be added by requiring tokens in order to access or store data. The KV does not restrict the stored object type so long as the object is under 512KB. Watches may be associated with paths of the KV store, allowing for data to be monitored for updates which will subsequently invoke an external handler when an update is detected [11].

Consul offers datacenters containing clusters of either server agents or client agents. A leader-follower algorithm known as RAFT is employed within server agents. There are three possible states: follower, candidate, and leader. Upon the onset of an election, all servers begin as followers, upon the onset of an election, the followers become candidates. Of the candidates one is selected to be the leader while the other candidates revert to followers and become passive replicas of the leader? Gossip protocols are a means by which consul broadcasts messages with the objective of achieving consistency throughout a distributed system. Communication within members of the consul datacenter is possible though the local area network protocol (LAN). Communication between different datacenters utilize the wireless area network (WAN) protocol [12].

Consul is platform agnostic meaning this product can integrate with existing technologies such as K8, EKS, VMs, Lambda, Nomad, etc. Service meshes such as Consul offer many benefits and features with added versatility due to being able to operate individually or utilized together in a full service mesh. Consul is also able to achieve zero-trust security. Fail handling, retries, and network observability are other benefits of this multi-networking tool. All organizations, ranging from security to improved application resiliency can also benefit from Consul’s service discovery application health monitoring, traffic management, and observability as well as traceability [13]. The availability of a UI is a unique feature which makes Consul simple and easy to use. The key/value store, the intentions, services, and health checks among others are visualized for the user. Consul allows for effortless adding and deleting without a required HTTP request [14]. HashiCorp offers user support through extensive published tutorials and documentation.

Other service meshes have been preferred over Consul because the product is considered expensive particularly in regards to the custom enterprise pricing; the website does advertise a free open source option as well as Consul HCP Development at a Standard, Plus, and Premium tiers [13]. However, the Standard is for non-production use, Plus is for larger deployments, and the Premium pricing tier is for reliability and scale [8]. The free open source option is limited with basic features while businesses will need more advanced features which are found in the enterprise option.

Traefik

Traefik Proxy is an open-source, cloud-native reverse proxy and load balancer designed to work seamlessly with microservices and container orchestration platforms such as Kubernetes, Docker, and Mesos [15]. While Traefik Proxy has gained popularity in recent years due to its ease of use, high performance, and support for dynamic configuration updates, it also has several advantages and disadvantages to consider when installing and configuring it.

One of the main advantages of Traefik Proxy is its simple configuration process. Traefik Proxy has a declarative configuration file format that allows users to specify the desired configuration in a human-readable format [16]. Additionally, Traefik Proxy supports multiple configuration backends, including Kubernetes, Docker, and file-based configuration [15]. This flexibility makes it easy to configure Traefik Proxy to work with different deployment environments and orchestration platforms. Furthermore, Traefik Proxy has built-in support for Let's Encrypt SSL/TLS certificate issuance, which simplifies the process of securing your services [17].

Another advantage of Traefik Proxy is its ability to perform automatic service discovery and load balancing. Traefik Proxy can automatically discover and route traffic to services based on their labels or annotations [18][19]. This feature reduces the amount of manual configuration needed, making it easy to scale and manage microservices-based applications. There is also a wide selection of middleware available to handle tasks such as rate limiting, circuit breaking, buffering, compression, and more [20].

On the other hand, Traefik Proxy has some disadvantages to consider. One major issue is that it lacks advanced load balancing features, only supporting round-robin at this time [19]. Another drawback is that Traefik Proxy can have a steep learning curve for users who are not familiar with container orchestration platforms or microservices architecture. Traefiklabs' community forum is full of posts requesting help or clarification regarding installation, configuration, and usage [21]. Users need to understand how Traefik Proxy works with their specific environment and how to configure it correctly to achieve optimal performance and reliability.

Another thing to consider before adopting Traefik Proxy is that it is a relative newcomer in the world of software proxies. Traefik's most common comparison is to nginx, a proxy that was first released back in 2004 [22]. In comparison, Traefik Proxy's first release was in 2015 [23]. The team behind Traefik Proxy, Traefiklabs, is still growing its financial revenue streams by providing software-as-a-service offerings Traefik Enterprise [24] and Traefik Hub [25]. Traefik Enterprise doesn't currently disclose pricing publicly and only makes it available through a contact form [26]. Traefik Hub is currently in early access and not ready for production use [25].

Traefik Proxy is a powerful and easy-to-use tool for managing microservices-based applications. While it has several advantages, including a simple configuration process, automatic service discovery, and integrated TLS management, it also has some drawbacks, such as a lack of advanced load balancing features, a potentially steep learning curve, and uncertainty about its future development. Overall, the decision to use Traefik Proxy will depend on the specific needs and requirements of your application, and careful consideration should be given to the pros and cons of this tool before installing and configuring it.

Publication Experiments

Key/value store lock experiments

As an artifact of the Communications in Computer and Information Science book series (CCIS, volume 1018), Grzesik and Mrozek evaluate the performance of the distributed locking mechanisms in etcd, Consul, and Zookeeper. This paper presents a comparative analysis of two different experiments to assess their ability to meet safety standards, preserve a deadlock-free state, and evaluate fault tolerance performance [27]. The authors developed a Python application that can use each of the stores to acquire a distributed lock. The application then simulated a short computation that requires the lock to be held and then released. The application measures the time it took to acquire a distributed lock.

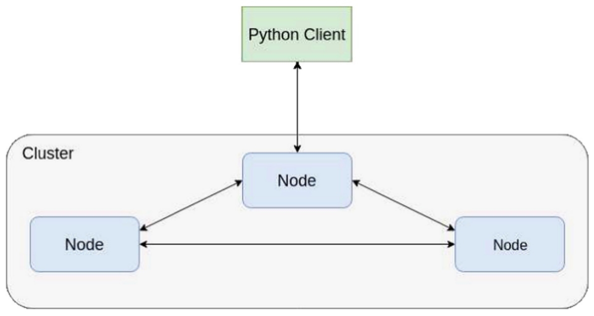


Figure I
Environment Configuration Diagram [27].

Each distributed locking mechanism was configured as a 3-node cluster and deployed in the AWS cloud. All of the configuration of the EC2 instances were sent up with identical parameters and specifications [27].

In the first experiment, the specifications of simulation did not allow for the process to compete for the same locks on concurrent traffic from 1, 3, and 5 different processes. The lock acquisition times increased as the number of processes increased. In the results, Consul consistently had the slowest times while Zookeeper outperformed both its competitors. For the results of etcd and Consul, the 99th percentile proved to be 2 times higher than average.

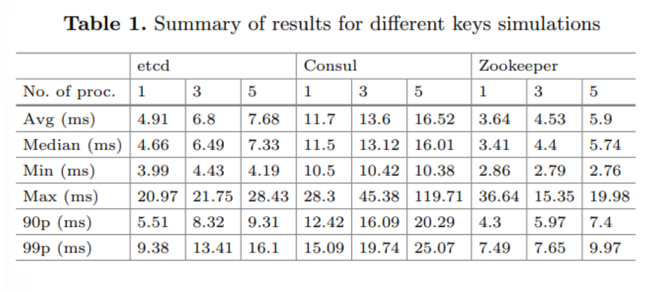


Table 1. Summary of results for different keys simulations

	etcd			Consul			Zookeeper		
No. of proc.	1	3	5	1	3	5	1	3	5
Avg (ms)	4.91	6.8	7.68	11.7	13.6	16.52	3.64	4.53	5.9
Median (ms)	4.66	6.49	7.33	11.5	13.12	16.01	3.41	4.4	5.74
Min (ms)	3.99	4.43	4.19	10.5	10.42	10.38	2.86	2.79	2.76
Max (ms)	20.97	21.75	28.43	28.3	45.38	119.71	36.64	15.35	19.98
90p (ms)	5.51	8.32	9.31	12.42	16.09	20.29	4.3	5.97	7.4
99p (ms)	9.38	13.41	16.1	15.09	19.74	25.07	7.49	7.65	9.97

FIGURE II
COMPETING FOR DIFFERENT LOCK RESULTS [27].

The simulation of the second experiment was similar but introduced competition for the same lock. Like in the first experiment, lock acquisition times increased as processes increased, however the average run times for lock acquisition minimally changed. The final results were the same as Zookeeper once again out did its competitors while Consul held up the rear.

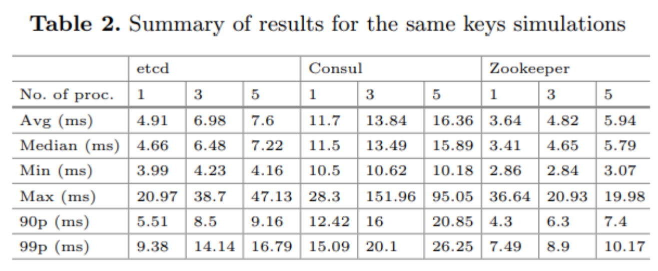


Table 2. Summary of results for the same keys simulations

	etcd			Consul			Zookeeper		
No. of proc.	1	3	5	1	3	5	1	3	5
Avg (ms)	4.91	6.98	7.6	11.7	13.84	16.36	3.64	4.82	5.94
Median (ms)	4.66	6.48	7.22	11.5	13.49	15.89	3.41	4.65	5.79
Min (ms)	3.99	4.23	4.16	10.5	10.62	10.18	2.86	2.84	3.07
Max (ms)	20.97	38.7	47.13	28.3	151.96	95.05	36.64	20.93	19.98
90p (ms)	5.51	8.5	9.16	12.42	16	20.85	4.3	6.3	7.4
99p (ms)	9.38	14.14	16.79	15.09	20.1	26.25	7.49	8.9	10.17

FIGURE III
COMPETING FOR THE SAME LOCK RESULTS [27].

KEY/VALUE STORE KEY CREATION EXPERIMENTS

The experiment below was conducted by Gyuho as a distributed database basemark tester [28]. Etcd, Consul, and Zookeeper Key/Value (KV) store experiments analyze the creation of KV pairs to compare throughput of 1 million keys, which were 256-bytes each, and had a respective value size of 1KB. The throughput was calculated for 100, 300, 500, 700, and 1000 concurrent clients.

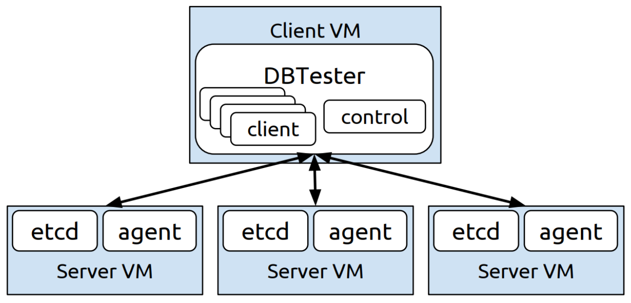


FIGURE IV
ENVIRONMENT CONFIGURATION DIAGRAM [28].

The virtual machines were deployed on Google Cloud Compute Engine where all machines had the same configurations of 16 CPUs, 60GB memory, and 300GB SSD [28].

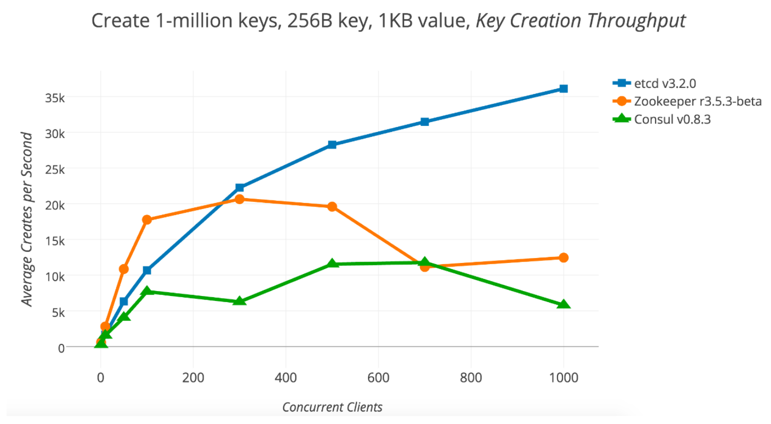


FIGURE V
KEY CREATION THROUGHPUT RESULTS [28].

As the amount of concurrent clients increased, etcd was the only one that experienced an increase in the number of keys created. Additionally, etcd excelled when supporting 1,000 concurrent clients as it was able to perform about 35 thousand key creations per second which is approximately 25 thousand more creations per second than its competitors. When supporting 100-500 concurrent clients, Zookeeper performed its best, however it performed negatively when the number of clients exceeded 500. Across all tests, Consul produced the least amount of keys per second.

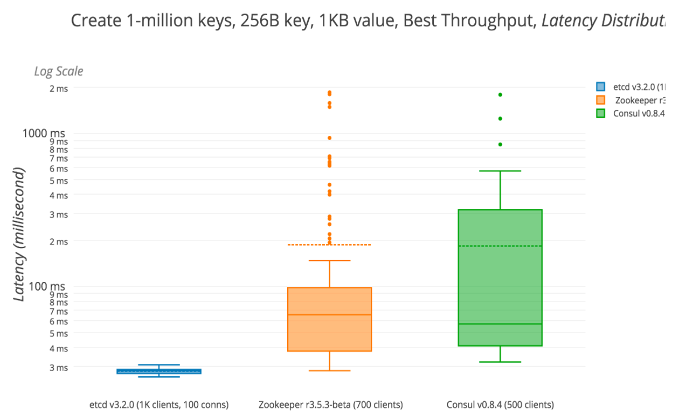


FIGURE VI
LATENCY DISTRIBUTION RESULTS [28].

Etcd also proved to hold the most consistent latency times which resulted in no significant visible outliers and a narrow range between the 1st and 3rd quartiles. The data represents Zookeeper as the key/store with the most outliers and experienced a large gap between the maximum value and first outlier. Zookeeper also is shown to have a right skew to the right. Consul, overall, had very few outliers and had a skew to the right. The range between the 25th and 75th percentile was of a much larger scope then Consuls too competitors, Zookeeper and etcd. In the end, etcd was the clear frontrunner, Zookeeper had a comparatively moderate performance, and Consul performed the worst.

PUBLICATION EXPERIMENT SUMMARY

Both the experiment conducted by Grzesik and Mrozek as well the experiment by Gyuho compared and contrasted Etcd, Consul, and Zookeeper. While these two studies had different methodologies and variables they each yielded similar results. In both cases, the three key/stored measured based on efficiency which was defined as completing a task in the fewest seconds. In the distributed lock test, Zookeeper did the best and Etcd performed well, but not as well as Zookeeper. Etcd still outperformed Consul. In the key/value creation experiment, Etcd was the fastest of the three key/stores in all tests involving the creation of pairs. Consul, would be varying margins performing the slowest. Both Etcd and Zookeeper were notably more efficient than Consul.

SELF-CONDUCTED EXPERIMENTS

KEY/VALUE STORE CREATION EXPERIMENTS

The latency and throughput associated with a key/value creation request to Consuls and etcds Key/Value store is featured in our self-conducting experiments. Both experiments utilize the same Virtual Machine running Ubuntu 22.04 inside of Oracle VM VirtualBox. To conduct the experiment, a python program was created to generate 100,000 key/value requests for varying numbers of concurrent clients [20, 40, 60, 80, 100]. To achieve this, a Python program was executed that utilized the Requests and concurrent.futures libraries to determine the latency time and total time taken for each number of concurrent clients to complete all 100,000 requests.

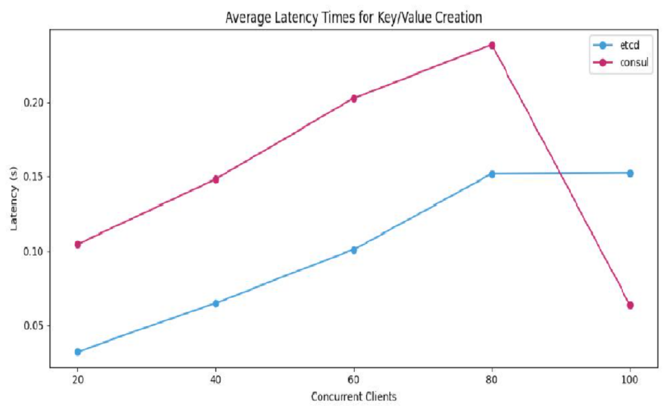


FIGURE VII
ETCD VS CONSUL LATENCY TEST RESULTS [I].

Both systems aim to provide low-latency access to key-value storage and service discovery and are designed to minimize the impact of network latency. etcd's highly optimized storage engine provides low-latency access to data with most requests typically taking only a few milliseconds to complete. Consul utilizes a gossip-based membership protocol which helps ensure that nodes in the cluster are always updated. It can also quickly detect and recover from failures thus improving overall cluster stability and reducing the impact of latency. In the latency test results graph [Fig. VII], average write latency performance tests for both key-value stores yield fairly similar results, albeit etcd having slightly better results when scaling up from 20 to 80 concurrent clients.

Consul's latency is averaging +0.06 milliseconds higher than etcd during that span. However, Consul's latency drops significantly after 80 concurrent clients. This can be attributed to its design prioritizing strong consistency at the cost of a minimal sacrifice on latency performance. These expected results are in line with benchmark data as performed in previous experiments. Both systems use the Raft consensus algorithm which significantly reduces latency, though etcd implements an additional technique for linearizability which allows reads from its data stores to be processed quicker through bypassing Raft [29].

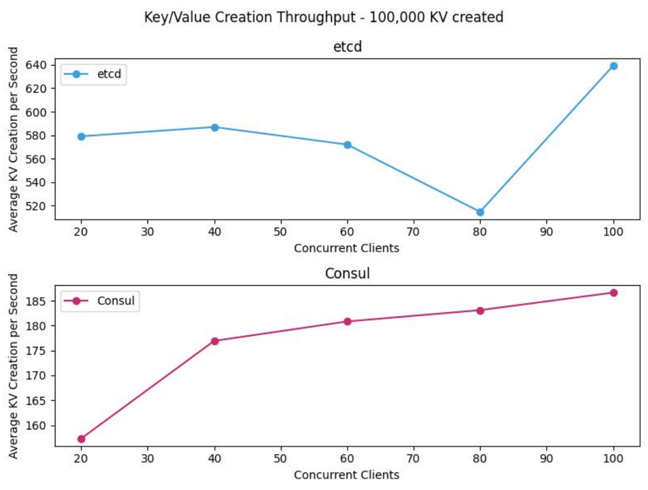


FIGURE VIII
ETCD VS CONSUL THROUGHPUT TEST RESULTS [I].

The founders of etcd made sure to prioritize high throughput performance in its architecture of the product [30]. A highly optimized storage engine is designed to easily accommodate substantial volumes of data. Etcd is fully capable of performing thousands of writes per second which also easily scales for larger deployments. Again, the Raft algorithm is credited for ensuring that writes are replicated efficiently across the cluster, therefore minimizing the impact on throughput. This can be seen as etcd throughput averages in the range of +/- 500 key-value writes per second as shown in the test results graph [Fig. VIII]. These numbers generally stay stable as more concurrent clients are available, though a slight dip can be noticed once 80 threads before climbing back up to +600. This demonstrates the system's resiliency as the workload increases. Comparatively, Consul has considerably lower average throughput in the range of +/- 180 key-value writes per second. As such, it may not be the most optimal choice for extremely high-volume workloads, though still capable in its own right.

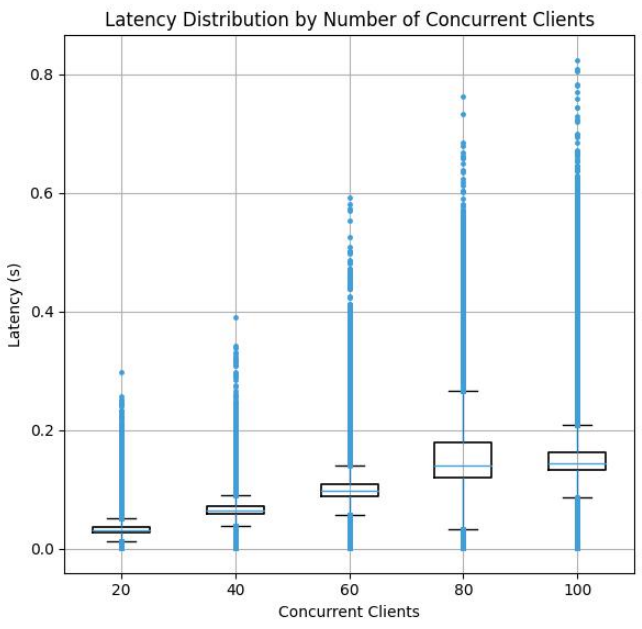


FIGURE IX
ETCD LATENCY DISTRIBUTION TEST RESULTS [I].

Analyzing Etcd's latency distribution box plot we can see the data for each one of the concurrent clients are rather close to symmetrical. This means the standard deviation of the results is not screwed, the data is equivalently spread on either side of the median. The 80 concurrent clients had the greatest variation between the 25th and 75th percentiles. The difference between the minimum and lower quartile, as well as the difference between the maximum and upper quartile are much greater than the differences presented by the other concurrent clients. In other words, the whiskers were the longest for the 80 concurrent client mark. The 100 concurrent client results had the most outliers and had the outlier representing the longest time for Etcd to complete the key creation request. Compared to 20 concurrent clients which had outliers of nominal value. Its furthest outlier was the most rapid of outliers. The 20 concurrent clients results were the most consistent.

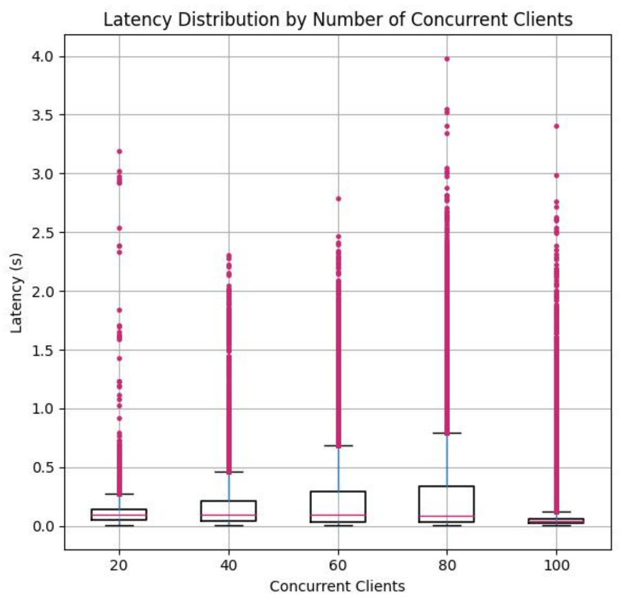
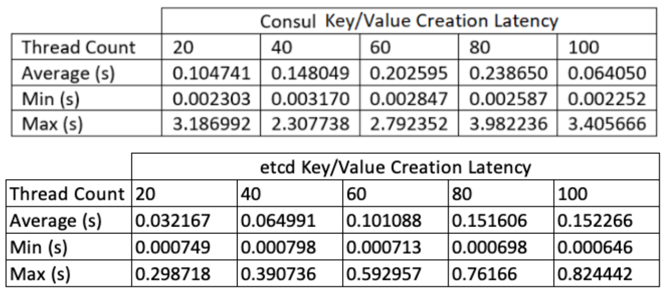


FIGURE X
CONSUL LATENCY DISTRIBUTION TEST RESULTS [I].

In the Consul Latency Distribution box plot we observed that Consul has a positive screw. This is due to no data points that are considered outliers, are present and are of a lower value than the minimum. The sum of concurrent clients which demonstrates the most consistent results is the 100 concurrent client amount due to the lower and upper quartile having a small range followed by the 20 concurrent clients. The 80 concurrent clients had the outlier representing the longest time for Consul to complete the key creation request. Consul's performance was shown to be the worst at 80 clients. The sum of the data points with the least amount of outliers was at 20 concurrent clients, but it was at 40 concurrent clients that the results demonstrate the smallest range of outliers.



	Consul Key/Value Creation Latency				
Thread Count	20	40	60	80	100
Average (s)	0.104741	0.148049	0.202595	0.238650	0.064050
Min (s)	0.002303	0.003170	0.002847	0.002587	0.002252
Max (s)	3.186992	2.307738	2.792352	3.982236	3.405666

	etcd Key/Value Creation Latency				
Thread Count	20	40	60	80	100
Average (s)	0.032167	0.064991	0.101088	0.151606	0.152266
Min (s)	0.000749	0.000798	0.000713	0.000698	0.000646
Max (s)	0.298718	0.390736	0.592957	0.76166	0.824442

FIGURE XI
ETCD AND CONSUL LATENCY DATA RESULTS [I].

SELF-TEST SUMMARY

To summarize, etcd and Consul performances prove each system's capability in handling large workloads. While both systems use the Raft consensus algorithm, they differ in their approach to prioritizing strong consistency versus throughput performance. These are further illustrated in the latency data results [Fig. IX] where side-by-side comparisons of their performance statistics are displayed through increasingly higher concurrency workloads. It shows etcd having the clear advantage in latency performance with its overall min/max much lower than Consul. Although etcd's documentation itself admits, "Both etcd and Consul solve different problems", with the former yielding to Consul's end-to-end cluster discovery features [1]. Application requirements for each use case will ultimately determine the ideal choice ranging from workload type, performance, consistency, scalability, and reliability.

SUMMARY

We conducted a survey of different distributed system technologies that focused on coordination and focus management. This paper analyzed Zookeeper, Consul, etcd, and Traefik Proxy. Each of the last three technologies are presented along with their core functionality, individual unique offers and benefits and drawbacks. We also explored each technology through two published experiments and analyzed the findings. Finally, we conducted our own latency experiments and documented the results. While etcd and Consul both offer key/value storage functionality and can be compared directly from that commonality, Traefik falls into a different class of distributed systems software and was not sufficiently comparable for the purpose of our experimental study. The concluding evidence showed that etcd had the fastest latency times while Consul was the slowest in fulfilling a key/value creation request.

References

[1] "ETCD versus other key-value stores," etcd, 27-Nov-2022. [Online]. Available: https://etcd.io/docs/v3.5/learning/why/. [Accessed: 02-May-2023].

[2] "Administration," etcd, 27-Apr-2021. [Online]. Available: https://etcd.io/docs/v2.3/admin_guide/. [Accessed: 02-May-2023].

[3] M. Fazlali, S. M. Eftekhar, M. M. Dehshibi, H. T. Malazi, and M. Nosrati, Raft Consensus Algorithm: an Effective Substitute for Paxos in High Throughput P2P-based Systems. 2019.

[4] "ETCD API," etcd, 26-Apr-2023. [Online]. Available: https://etcd.io/docs/v3.5/learning/api/. [Accessed: 02-May-2023].

[5] E. Evans, "How ETCD works with and without kubernetes," Learnk8s, 21-Jul-2021. [Online]. Available: https://learnk8s.io/etcd-kubernetes. [Accessed: 02-May-2023]

[6] "ETCD," Cloud Native Computing Foundation, 16-Mar-2022. [Online]. Available: https://www.cncf.io/projects/etcd/. [Accessed: 02-May-2023].

[7] Cloud Native Computing Foundation. (2023, March 16). Transport security model. etcd. Retrieved May 2, 2023, from https://etcd.io/docs/v3.4/op-guide/security/

[8] P. Bournhonesque, "Hashicorp Consul: Because dynamic workloads call for Dynamic Service Networking," *Devoteam*, 20-Apr-2023. [Online]. Available: https://www.devoteam.com/expert-view/hashicorp-consul-because-dynamic-workloads-call-for-dynamic-service-networking. [Accessed: 03-May-2023].

[9] N. Sabharwal, S. Pandey, and P. Pandey, "Infrastructure-as-code automation using terraform, Packer, vault, Nomad and consul : Hands-on deployment, configuration, and best practices," *O'Reilly Online Learning*. [Online]. Available: https://learning.oreilly.com/library/view/infrastructure-as-code-automation-using/9781484271292/html/492265_1_En_7_Chapter.xhtml. [Accessed: 03-May-2023].

[10] Khatri, Anjali, et al. "Understanding the Consul Service Mesh." Mastering Service Mesh, Packt Publishing, Limited, 2020.

[11] "Key/value (kv) store overview: Consul: HashiCorp developer," *Key/Value (KV) Store Overview | Consul | HashiCorp Developer*. [Online]. Available: https://developer.hashicorp.com/consul/docs/dynamic-app-config/kv. [Accessed: 03-May-2023].

[12] "Consul - introduction," *Tutorials Point*. [Online]. Available: https://www.tutorialspoint.com/consul/consul_introduction.htm. [Accessed: 03-May-2023].

[13] "Service Mesh explained: Consul: HashiCorp developer," *Service Mesh Explained | Consul | HashiCorp Developer*. [Online]. Available: https://developer.hashicorp.com/consul/docs/concepts/service-mesh. [Accessed: 03-May-2023].

[14] "Hashicorp Consul Simple Breakdown: Quickreads," *CD Cloud Logix*, 15-Jun-2022. [Online]. Available: https://cdcloudlogix.com/hashicorp-consul-simple-breakdown-quickreads/. [Accessed: 03-May-2023].

[15] "Providers Overview," *Traefik*. [Online]. Available: https://doc.traefik.io/traefik/providers/overview/. [Accessed: 03-May-2023].

[16] "Configuration Introduction," *Traefik*. [Online]. Available: https://doc.traefik.io/traefik/getting-started/configuration-overview/. [Accessed: 03-May-2023].

[17] "HTTPS & TLS," *Traefik*. [Online]. Available: https://doc.traefik.io/traefik/https/overview/. [Accessed: 03-May-2023].

[18] "Routers," *Traefik*. [Online]. Available: https://doc.traefik.io/traefik/routing/routers/. [Accessed: 03-May-2023].

[19] "Services - Load Balancing," *Traefik*. [Online]. Available: https://doc.traefik.io/traefik/routing/services/#load-balancing. [Accessed: 03-May-2023].

[20] "Middleware," *Traefik*. [Online]. Available: https://doc.traefik.io/traefik/middlewares/http/overview/. [Accessed: 03-May-2023].

[21] *Traefik Labs Community Forum*. [Online]. Available: https://community.traefik.io/. [Accessed: 03-May-2023].

[22] "Nginx Changelog," *nginx*. [Online]. Available: http://nginx.org/en/CHANGES. [Accessed: 03-May-2023].

[23] Traefik, "Releases · Traefik/Traefik," *GitHub*. [Online]. Available: https://github.com/traefik/traefik/releases. [Accessed: 03-May-2023].

[24] "Traefik Enterprise, the API Gateway Cloud Natives Trust," *Traefik Labs: Say Goodbye to Connectivity Chaos*. [Online]. Available: https://traefik.io/traefik-enterprise/. [Accessed: 03-May-2023].

[25] "Traefik Hub, your apis deserve better," *Traefik Labs: Say Goodbye to Connectivity Chaos*. [Online]. Available: https://traefik.io/traefik-hub/. [Accessed: 03-May-2023].

[26] "Compare Traefik Products & Pricing: Traefik Labs," *Traefik Labs: Say Goodbye to Connectivity Chaos*. [Online]. Available: https://traefik.io/pricing/. [Accessed: 03-May-2023].

[27] Kozielski, Stanisław, et al. "Evaluation of Key-Value Stores for Distributed Locking Purposes." Beyond Databases, Architectures and Structures. Paving the Road to Smart Data Processing and Analysis, vol. 1018, Springer International Publishing AG, 2019, pp. 70–81, https://doi.org/10.1007/978-3-030-19093-4_6.

[28] Etcd-Io, "Dbtester/test-results/2018q1-02-etcd-zookeeper-consul at master · ETCD-IO/DBTESTER," *GitHub*. [Online]. Available: https://github.com/etcd-io/dbtester/tree/master/test-results/2018Q1-02-etcd-zookeeper-consul. [Accessed: 03-May-2023].

[29] Cloud Native Computing Foundation. (2021, August 17). KV API guarantees. etcd. Retrieved May 2, 2023, from https://etcd.io/docs/v3.3/learning/api_guarantees/#linearizability

[30] Cloud Native Computing Foundation. (2021, June 14). Performance. etcd. Retrieved May 2, 2023, from https://etcd.io/docs/v3.5/op-guide/performance/