Project Report

Title: Scalable Metrics Retrieval Solution using Thanos and Prometheus

Team Member 1: Sashank Agarwal (sashanka@usc.edu)
Team Member 2: Anshul Gupta (agupta84@usc.edu)

Project Summary and Goals

This project aims to analyze a scalable solution for retrieving metrics from a cluster of distributed systems using Thanos and Prometheus and efficiently address the challenges faced by large metrics consuming applications like TODS (Timeseries-based Anomaly Detection System), LLM models, Business Intelligence and Data Analytics.

Problem Statement:

- Distributed Systems Administrators currently rely on Prometheus for metrics retrieval, storage, and delivery to consumer applications.
- The growing volume of metrics data generated by the cluster increases the load on standalone Prometheus, leading to potential performance issues and instability.

Proposed Solution:

- Leverage Thanos, as an extension to Prometheus, to provide horizontal scaling capabilities for the metrics querying and delivery component (Thanos Query). This allows for on-demand scaling to handle increased loads without compromising stability.
- Utilize Azure Blob Storage for efficient data storage of Prometheus metrics.

By implementing this project, the goal is to analyze the proposed metrics retrieval solution and compare it with the traditional method in the following aspects

- Performance
- Cost
- Network Bandwidth Utilization
- Storage Utilization

Some additional metrics we would like to measure to assess the proposed system's feasibility:

- Time to Scale
- Latency

Methodology

Tools and Platform Used:

Prometheus

Prometheus is an open-source monitoring and alerting toolkit designed for reliability and scalability, primarily supporting monitoring microservices and containers in a Kubernetes environment. It collects and stores metrics as time series data, allowing users to query this data with its powerful query language, PromQL. The system also supports the configuration of sophisticated alerting rules to notify users about potential issues.

Thanos

Thanos is an extension of Prometheus designed to provide highly available and scalable Prometheus setups across multiple environments. It enhances Prometheus by enabling long-term storage, added reliability, and global querying capabilities across a network of Prometheus instances. Thanos achieves this by integrating with cloud storage solutions (like Amazon S3, Google Cloud Storage, or Azure Blob Storage) to store historical metric data, making it easier to handle large volumes of data over time without losing the granularity of short-term data.

Grafana

Grafana is an open-source platform for monitoring and observability. It allows users to query, visualize, alert on, and understand metrics no matter where they are stored. Grafana supports multiple data sources, including Prometheus, Elasticsearch, InfluxDB, and many others, allowing it to serve as a central hub for all metrics and logs. With its feature-rich data panels and interactive dashboards, Grafana is widely used to track operational metrics, understand complex systems, and troubleshoot potential issues.

Minikube

Minikube is an open-source tool that allows users to run Kubernetes locally on their computers. Minikube runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop with it day-to-day. It is designed to be as easy to use as possible and is a great way to get started with Kubernetes without requiring the resources of a full cluster.

Azure Blob Storage

Azure Blob Storage is a scalable object storage service provided by Microsoft Azure. It is designed to serve images or documents directly to a browser, store files for distributed access, stream video, and audio, write to log files, or store data for backup and restore, disaster recovery, and archiving. Azure Blob Storage is highly optimized for storing massive amounts of unstructured data, such as text or binary data, which makes it a fitting choice for applications requiring large-scale, cloud-based storage.

Architecture:

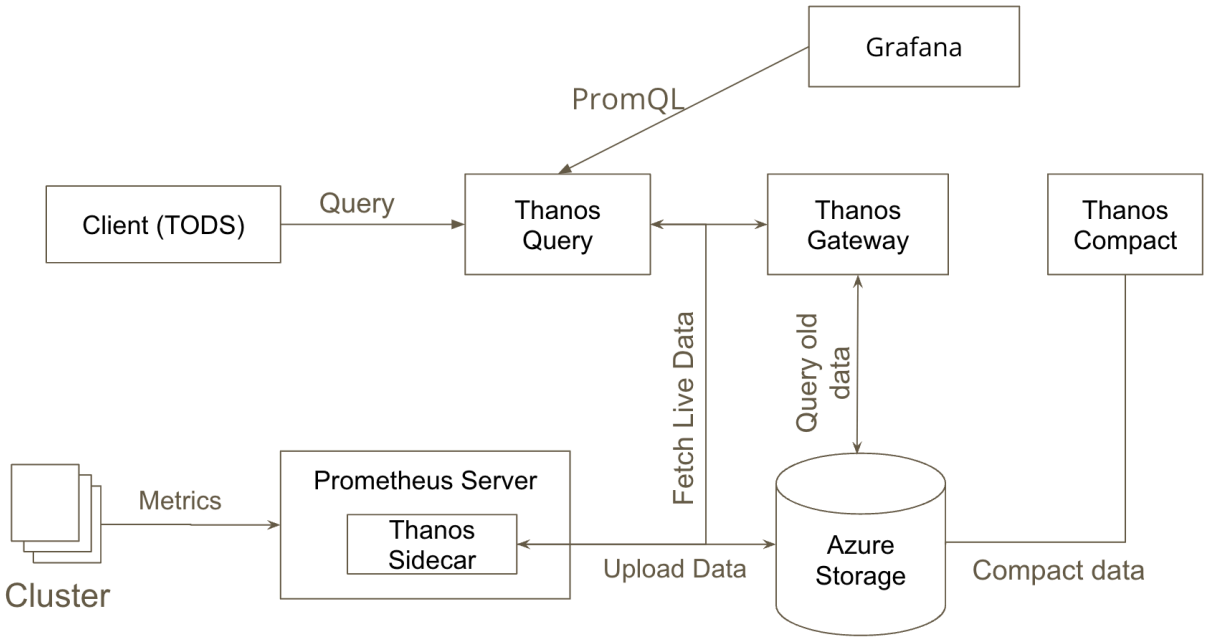

Table 1: Proposed Architecture

- Prometheus Server collects metrics from the cluster, and stores live data (past 2 hours) in its memory.
- Thanos sidecar service, which coexists alongside Prometheus, runs every hour and uploads data that is more than 2 hours old to the cloud object store (Azure Storage). It also acts as a gateway for Thanos Query to access the live data from Prometheus.
- Thanos Compact regularly runs downsampling and compacting data that is being stored.
- All the queries for metrics data are served by the Thanos Query component. It analyzes the request, fetches the live data from Thanos Sidecar and old data from Azure, combines all the data in a single response, and sends it back to the requesting application.
- When the query load increases, a new instance of Thanos Query gets spawned to balance the load among all of the available Thanos components.

Experimental Setup:

- Run 5 instances of the TODS app.
- Each instance:
    - Spawns 5 goroutines every 30 seconds.
    - Queries 10 metrics across 5 namespaces.
    - Run for a period of ~1 hour.
- Prometheus - 1 instance.
- Thanos - 1 instance (initially).

Results

Cost Estimation:

Approach / Component	Prometheus (Standalone)		Prometheus + Thanos (3VM)	
	Cost	Configuration	Cost	Configuration
Prometheus VM	$127.24 - $276.48	8 vCPU 32Gb RAM	$31.82 - $69.12	2 vCPU 8Gb RAM
Thanos VM	N/A		$6.91 * 3	2 vCPU 8Gb RAM
Storage	$76.8	1000Gb SSD	$52.41	1000Gb Blob
Total	$204.02 - $353.28		$104.96 - $140.1	

Table 2: Cost estimation summary

- Prometheus (Standalone): The cost of deploying Prometheus as a standalone server ranges from $204.02 to $353.28 per month, depending on the server's CPU, RAM, and storage specifications.
- Prometheus + Thanos (3 VM): For a more scalable solution, deploying Prometheus with Thanos on three virtual machines results in a cost between $104.96 to $140.10 per month.

Key Considerations

- The most expensive configuration utilizes a standalone Prometheus setup with high resource allocation (8 vCPU, 32 GB RAM, 1000GB SSD storage).
- The most cost-effective configuration combines Prometheus and Thanos for scalability while using lower resource tiers (e.g., 2 vCPU, 8 GB RAM, 1TB Blob storage).

Key Takeaway

- Approximately 45-50% savings in monthly cost can be achieved by using a scalable Thanos + Prometheus solution.

Standalone Prometheus Usage

- AIM:
  - Test standalone Prometheus under varying load conditions.
  - Use Prometheus to collect, store, and serve metrics to clients.
- Procedure:
  - Prometheus initially started to only collect metrics from the cluster.
  - Gradually, the client TODS application is started, and that fetches metrics from Prometheus.

    - The CPU usage during the time is measured and visualized using Grafana, as shown in Fig 1.
- Outcome:
    - A steady 2x to 3x growth in CPU usage is observed.

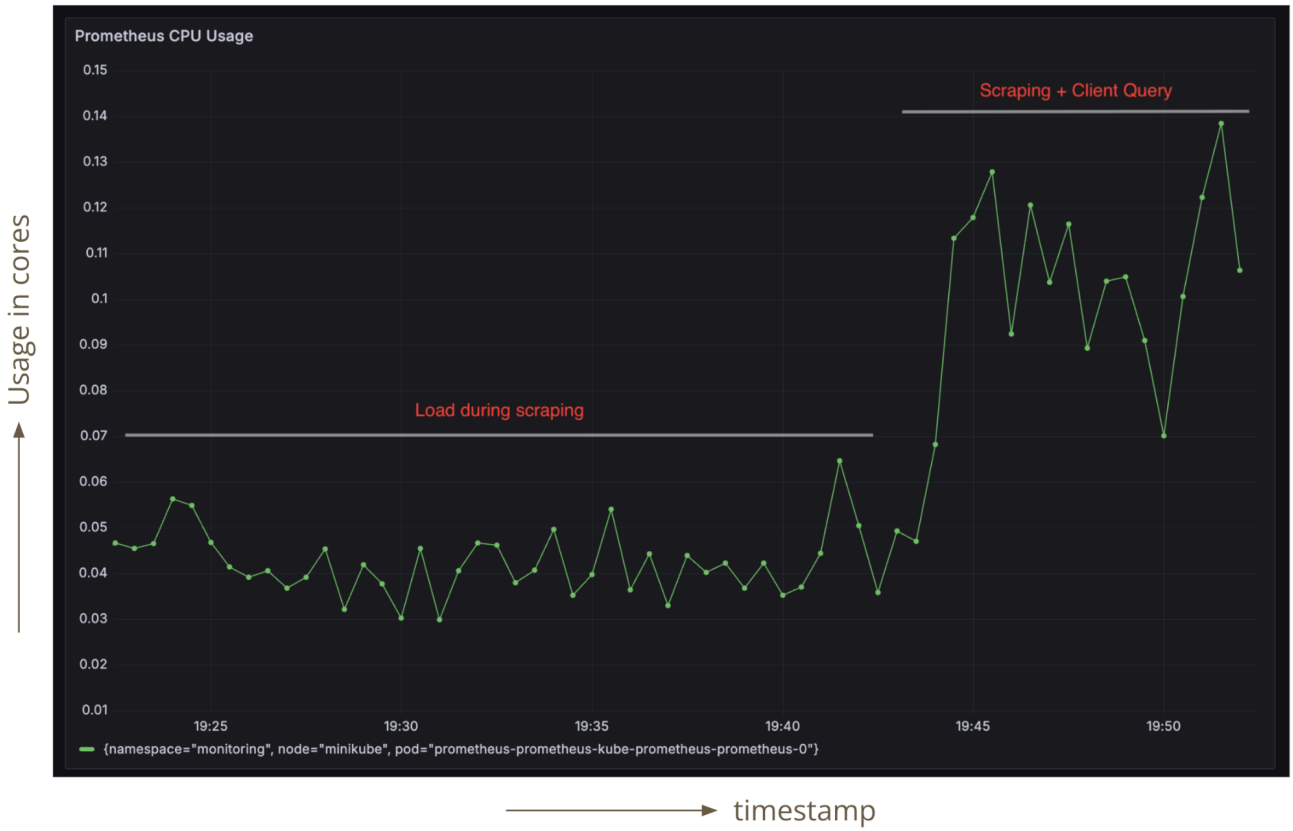

Fig 1: CPU usage of Prometheus

Prometheus + Thanos Usage

- AIM:
    - Test Prometheus + Thanos setup under varying load conditions.
    - Use Prometheus to collect metrics.
    - Use Thanos Gateway to store metrics to Azure Storage after downsampling and compaction using Thanos Compact.
    - Use Thanos Query to serve the client's requests for metrics.
- Procedure:
    - Prometheus initially started to only collect metrics from the cluster.
    - Thanos Query component is started to serve the clients for metrics.
    - The TODS application is started, and an increase in the load was observed over a period of time, as shown by Fig 2, where instances 2 and 3 of the Thanos Query component get spawned up.
    - After a brief period of time, 3 out of 5 instances of TODS are stopped, and the load is observed, as shown by Fig 2, where instance 3 of the Thanos Query component dies.
- Outcome:

- New instances get spawned based on on-demand load.
- The load is balanced between all the instances to utilize the CPU efficiently.
- As the load decreases, instances die down.

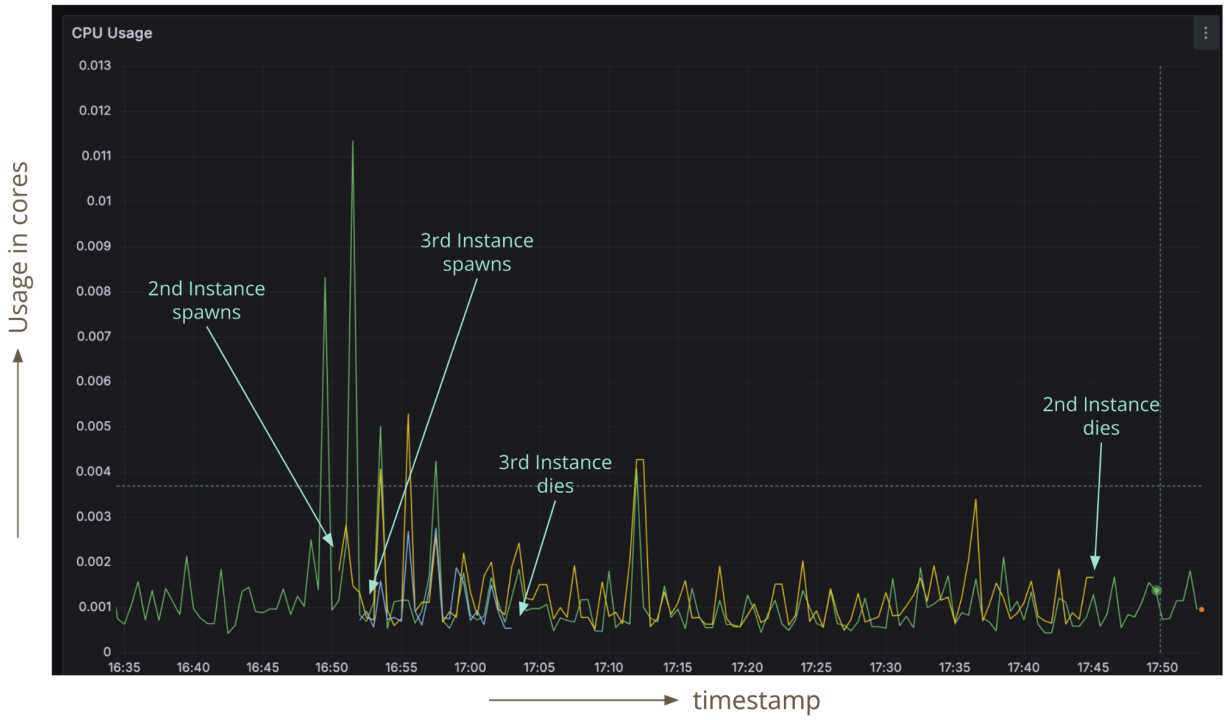

Fig 2: CPU usage of Thanos Query

Time to Scale and Stabilization Period of Thanos Query Instances

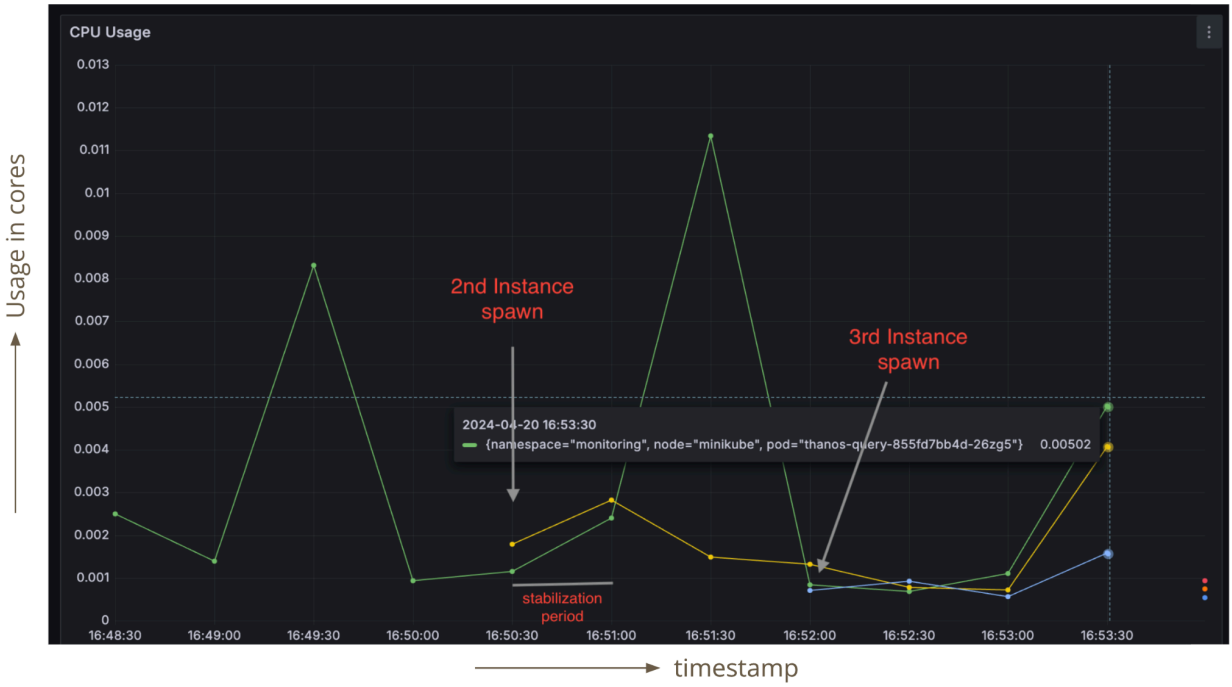

Fig 3: A detailed view of 2nd and 3rd instances spawning

Time to Scale

- The 2nd instance takes approximately 40s to 60s from the time of load increases to the time it starts serving. This is basically known as time to scale, as shown in Fig 3.
- This is indeed more for 2nd instance when compared to 3rd instance, which has time to scale approximately between 20s and 30s.
- The key reasons for this increase in time with 2nd instance are:
    - Since only one instance is available initially, when we send a burst of traffic by starting all 5 instances of client application simultaneously, the instance becomes overwhelmed. It fails to efficiently decide on which task to do first, i.e., whether to serve the client request or fulfill the need to spawn another instance.
    - The 1st instance might assume that the increase in load is temporary and decide to continue serving the requests with one instance only and wait a little longer before spawning a new instance.

Stabilization Period

- Each instance takes approximately 10-20 seconds before they can start serving the client's requests.
- This time period is termed as Stabilization Period. During this period:
    - The instance gets scheduled on the node.
    - Resources are allocated to the instances.
    - The new instance registers itself with the Kubernetes service for efficient load balancing.

Latency

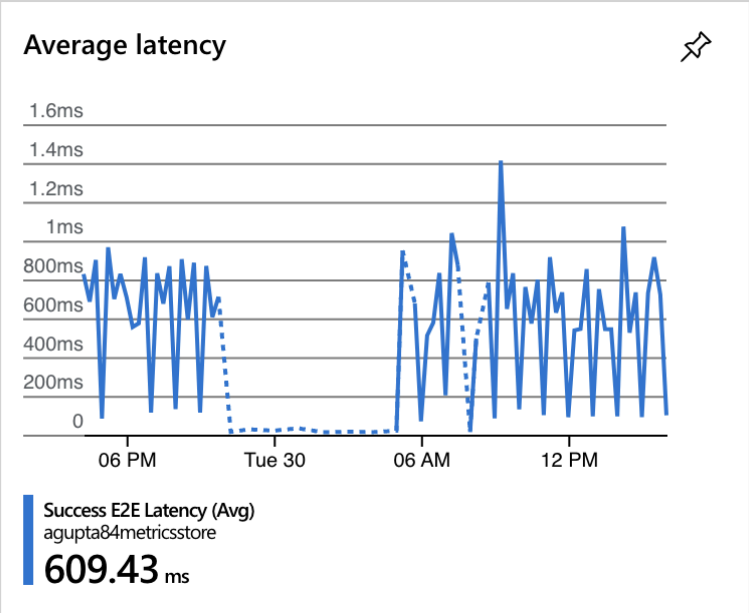

Fig 4: Average latency at Azure Cloud Storage

- Writing data to Azure Storage incurs additional latency, as shown in Fig 4.
- The average latency is 609 ms, which doesn't degrade the performance of the Prometheus + Thanos solution we recommend.

Storage Utilization

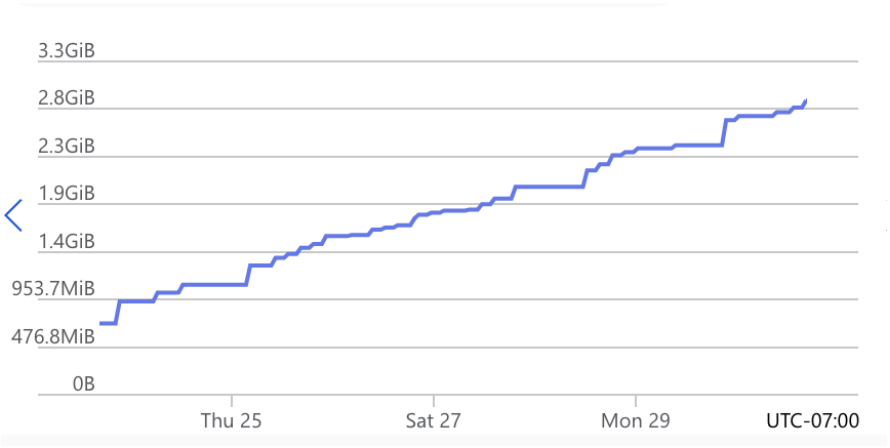

Fig 5: Average Azure Cloud Storage Utilization for 7 Day Period

- Prometheus: Since it only stores data from the past 36 hours, prorating the storage required to store 7 Days' worth of data comes to 10GB - 12GB.
- Prometheus + Thanos: As per Fig 5, on average, we needed 3 GB of storage space on the cloud. The storage benefit we are getting is due to the Thanos Compaction component actively competing with the stored data.

Key Insights

Prometheus Only:

- Incurs a higher monthly cost
  - This is mainly due to large instance sizes.
  - Local SSD for data retention incurs higher costs.
- The trade-off here is:
  - High performance metrics processing and storage but with increased costs.

Prometheus + Thanos:

- The cost is significantly reduced
  - Mainly due to smaller instances of both Prometheus and Thanos
  - Cost-effective Azure Blob storage for long-term data retention.
- The trade-off here is:
  - Minimal degradation in performance compared to standalone but a cost-efficient solution suitable for many use cases.

Recommendation

- Consider Prometheus with Thanos in places where:
    - Organizations prioritizing cost optimization without compromising essential functionalities, integrating Thanos with Prometheus presents a compelling option.
    - It offers a balance between cost-effectiveness and functionality, making it suitable for various monitoring and analytics requirements.
- Many of the use cases, like Anomaly Detection, Generative AI, Large Language Models (LLMs), etc, don't rely on live data. Rather, they need very old data in an efficient time. In such scenarios, the proposed stack can be very helpful as it is fast and cost-effective.

Challenges

Cost of running in the Cloud

- While initially running applications on free-tier-eligible VMs in Azure, we encountered high costs due to Azure Kubernetes Service (AKS) fees required to manage server cluster and the non-volatile premium data storage requirements of Prometheus, which quickly exceeded the free tier, amounting to $80 for a week's operation. To mitigate these costs, we transitioned to running our applications locally.
- This transition involved scaling down our computing resources for each pod and cluster size. Additionally, to manage our distributed cluster, we used Minikube due to its ease of configuration, rapid setup, and support for the latest features.

Low volume of metrics

- Since we had to scale down the cluster size and operations to run locally, we noticed that the volume metrics generated were low and insufficient to see any substantial change in the two approaches we are comparing in this Project.
- To tackle this challenge, we ran 5 instances of the application in each Kubernetes pod.