National College of Ireland

Dynamic Infrastructure Scaling Mechanism in Preallocated Resource Environments: A Practical Deployment Approach with Apache Solr

MSc Research Project
MSc In Cloud Computing

Sunil Suresh Gadhe
Student ID: X22179607

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Sunil Suresh Gadhe
Student ID:	X22179607
Programme:	MSc In Cloud Computing
Year:	2023-24
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	12/08/2024
Project Title:	Dynamic Infrastructure Scaling Mechanism in Preallocated Resource Environments: A Practical Deployment Approach with Apache Solr
Word Count:	7,493
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sunil Gadhe
Date:	12th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Dynamic Infrastructure Scaling Mechanism in Preallocated Resource Environments: A Practical Deployment Approach with Apache Solr

Sunil Suresh Gadhe
X22179607

Abstract

In today's world, it's crucial to optimize both the design and deployment strategies for applications. Traditional approach to scale out infrastructures on the cloud involves the use of the host metrics which include CPU, Disc I/O and memory usage. New nodes are brought, or nodes are removed from a node cluster based on given metrics conditions being met. However, this approach is ineffective in environments with preallocated resources, such as an Apache Solr cluster. In these configurations, a certain amount of memory is dedicated to the Solr service that implies different and constant memory usage at various queries requests. This can lead to a negative impact on the functioning of the application and results to time delay. To maintain optimal application performance, manual intervention is necessary to add and remove nodes from the cluster.Due to very high numbers of query requests, then the cluster must have maximum nodes in order to control the numerous Query requests. However, this approach results in resource wastage when the number of queries is relatively low. This paper introduces a dynamic scaling algorithm and deployment strategy that leverages a scheduler machine to address the challenge of efficient resource utilization in Apache Solr clusters. The overall goal of the study is to improve resource usage, extend the Solr cluster sustainably, avoid waste, and ensure the application's high throughput during the busiest period, where some focuses are associated with each goal. The dynamic scaling algorithm will follow chosen metrics like the rate of Solr queries and the latency over the time frame of 1 min, or 5 mins. It will then scale in/out the size of the Apache Solr cluster to the production level in the most efficient manner possible.

Keywords— Cloud computing, Dynamic resource allocation, Scheduler machine, optimal resource allocation strategy, Service Principle, Azure Cloud, Algorithm for Resource Allocation, Resource scheduling, Resource management, VMSS, Apache Solr

1 Introduction

1.1 Background And Motivation

An overall problem one can observe today is how to efficiently manage architectural design and deployment for applications in the constantly shifting context of today's digital environment. This is the chance to find the ways of efficient resource management, avoiding the creation of more waste, and building the proper infrastructure that will

allow providing high application performance. The traditional ways of scaling widely based on the usage of such indicators as CPU and memory are insufficient in cases when resources are pre-allocated, for example in the Apache Solr clusters. Apache Solr which is a search platform is a perfect example of this challenge; this is because Solr is an open source search platform that is widely reputed for its capabilities in real-time indexing and searching. This paper is focused on how to solve the scaling problem when deploying Apache Solr to Azure Cloud Platform using the Azure services, Virtual Machine Scale Sets (VMSS), Azure File Share, Azure Compute Gallery, and Azure App Registration. In this research, the focus is made on the investigation of dynamic scaling techniques, relating to preallocated resource environments in order to improve the resource usage and application performance to propagate knowledge of contemporary cloud solutions.

1.2 Objective

The main objective of the study paper is to assess the implementation of Apache Solr in the context of the Azure Cloud Platform with the focus on designing and implementing dynamic scaling method for the Apache Solr cluster and enhance application performance. The purpose of the study is to prove that solutions to the identified dynamic scaling problems are both feasible and challenging in implementation, while offering the highest levels of resource usage and application performance in contemporary cloud environments.

1.3 Research Question

1. How can we enhance the performance of an application that integrates with Apache Solr, while also optimizing resource allocation and keeping deployment costs low?

2. How does the implementation of a dynamic scaling algorithm, based on Solr query rates, impact resource utilization and application performance in preallocated resource environments?

1.4 Problem Statement

In the contemporary world of digital solutions, organizing resources and infrastructure growth are paramount to achieving high results in application handling. Preallocated resource environments are difficult to optimize as resources like Apache Solr clusters and Kubernetes clusters are pre-allocated. Other techniques like scaling using host metrics or relying on the intervention of operations personnel are not efficient in such cases. The first is to make use of a dynamic method of scaling that will allow the application to distribute resources properly, avoid wastage and activate all servers as soon as needed in the competence environment, without reference to host indicators.

1.5 Problem Solution

The proposed solution is based on the pro-active dynamic scaling of the Apache Solr cluster depending on the Solr query rate. In particular, the focus of the present investigation is to set up Apache Solr on the Azure Cloud platform by employing Azure services including Azure Virtual Machine Scale Set (VMSS), Azure File Share, Azure Compute Gallery, Service Principles, etc. This solution tries to remove the disadvantages of the old scaling ways which is the problem of the preallocated resource environments

and tries to give an effective, actual ways to scale Apache Solr on Azure Cloud Platform for improving the efficiency of the resources and performance of the application.

2 Related Work

Understanding Apache Solr and it's use in the distributed system is required for researching and finding a solution to achieve optimal performance and . This section will discuss different components such as Azure Cloud Components and Apache Solr. It will also explain what has already been done to address different resource allocation techniques used to improve the resource allocation and application performance.

2.1 Azure Cloud Platform

Azure, developed by Microsoft, is a comprehensive cloud computing platform that provides a wide range of cloud services including those for computing, analytics, storage, and networking. It allows users to build, deploy, and manage applications and services through a global network of Microsoft-managed data centers. Following are the Azure resources that has been used in the literature background work so far:

2.1.1 Azure Virtual Machine Scale Set (VMSS) ju shim (2024)

Azure Virtual Machine Scale Sets (VMSS) is a cloud service that a user can use to create one or more VMs and have them managed automatically by Azure, with those VMs being identical to each other and load-balanced. VMSS is supposed to help scale the number of the VMs in regards to their usage, so that the application does not experience load issues in the case of increased traffic.

2.1.2 Azure App Registration rwike77 (2023)

This is a process in Azure Active Directory, through which one can register the particular application and enables it by providing a unique identity in a tenant. This is also known as Service Principal that is used by the application to authenticate in the Azure services, to gain access to the resources and perform operations based on the authority granted.

2.1.3 Azure Compute Gallery ju shim (2023)

Azure Compute Gallery is the one of the services offered by the Microsoft Azure, which is used to store, organize and deploy the authorized Virtual Machine images in the organization. Azure Compute Gallery comes with many advanced features when used alongside VMSS and can be used to manage and deploy instances of VM efficiently.

2.1.4 Azure Virtual Machine cynthn (2024)

Azure Virtual Machine (VM) is a virtualized computing resource through the Azure services offered through a partner facility by Microsoft. It is a kind of copy or simulation of the physical computer that is hosted on the Azure cloud environment. Azure VMs can be used to execute applications, host websites, and so on, all the while giving the user the

ability to utilize computer resources while completely removing them from the necessity of having to deal with physical machines.

2.2 Apache Solr

Apache Solr is an open source search platform created as an inclusion of Apache Lucene to enable effective search and indexation. It should be noted that in the area of distributed systems, the Solr server can be employed in a special distributed node called SolrCloud.

Here's how Apache Solr fits into a distributed system:Here's how Apache Solr fits into a distributed system:

Distributed Search and Indexing: SolrCloud is a flexible system allowing data to be indexed and searched in multiple points within a Solr. This indicates that big data can be partitioned so that some parts (shards) are stored in different servers. Then concurrent queries are performed on these shards and results obtained are compiled and given to the user.
Scalability: Another advantage of Solr in a distributed setup is the ability to scale up the clusters horizontally, that is, scale up by having more nodes. Which is more important as and when data increases to a new level one or more nodes can be added to the Solr cluster to meet the new loads without stress.
Fault Tolerance and High Availability: SolrCloud also ensures that the system is always available despite the nodes which are exist in the system. This make data copies at different nodes for if one node goes down another node can be there to ensure the service is on.
Load Balancing: In a distributed setup, Solr can partition queries and can let them out to other nodes for load balancing. It assists in handling high- query rates and makes certain that none of the nodes becomes a bottleneck.
Real-time Search: Distributed Solr setups on the other hand are more suited for real time search where new data is indexed and made searchable in the real time in the cluster, it therefore fits well in to applications that require real time search.

2.3 Summary of Literature Review

The papers listed in the below table focuses on the cloud computing resource allocation, and the available literature discusses a vast number of methods that help to improve resource usage and improve the overall system performance. The adaptive resourcing has been formulated in key research works through techniques like the agent based model, genetic algorithms and heuristic in nature that learn the load as and when it is experienced. These methods have been helpful in cases where there is an attempt to reduce wastage to resources, balancing in the loads and response times. Multi-objective optimization and other methods of machine learning enhance the efficiency of resource distribution by considering workload and addressing the issues raised by urgent requests, and increase both cost and performance in the cloud services. This line of research focuses on the importance of advanced scheduling and forecasting procedures in a logical use of the cloud computing systems.

However, the research summarized here is mostly centered around the use of LP, the

so-called 'classic' method of host metric monitoring for estimating resources. Although these methods are helpful, they do not fully satisfy when it comes to the need of scaling Apache Solr. On the other hand, the proposed solution will be entirely designed to capture new metrics that are specific to Solr such as query rates over 1 min intervals of time, 5 min, and 15 min. This approach is intended to time toen offer a better solution to the dynamic nature of cloud environement for optimisation of Solr.

Paper Title	**Objective**	**Methodology**	**KeyFindings/Results**	**Implications**
Agent-based adaptive resource allocation algorithms for cloud computing Bhanuprakash and Sunitha (2018)	Develop adaptive algorithms for resource allocation in cloud computing.	Proposed algorithms include swapping and backfilling for managing resource allocation based on agent-based negotiation and counter offers.	Algorithms improve resource utilization and minimize request rejection by generating counter offers and using backfilling for efficient resource scheduling.	Enhances cloud resource management, reduces resource wastage, and improves service satisfaction rates.
A cloud resource allocation method supporting sudden and urgent demands Chen (2018)	To propose a cloud resource allocation method tailored to support sudden and urgent demands efficiently.	Develops a method that calculates new priorities for resource allocation based on user priorities and urgency levels. Utilizes a multi-objective optimization model to ensure minimal performance match distance between virtual machines and physical machines and to minimize the number of physical machines used.	The method shows advantages in reducing the number of physical machines required and improving CPU utilization. It supports rapid and optimal resource allocation during emergencies, outperforming traditional methods in urgent scenarios.	This method enhances the cloud's ability to handle emergency resource demands, making it suitable for scenarios requiring quick response times. It can potentially reduce energy consumption and improve resource utilization efficiency in data centers.
Autonomic resource allocation mechanism for service-based cloud applications Bhardwaj et al. (2019)	Design algorithms that dynamically adjust resources based on varying workloads to improve resource utilization and reduce operational costs.	Heuristic approach.	Proposed an autonomic resource allocation mechanism for cloud applications that improves resource utilization and response time using simple heuristics.	Optimizes resource utilization and improves response time for cloud applications.
Automated indexing of structured scientific metadata using apache solr Guntupally et al. (2020)	To develop an efficient automated indexing system for structured scientific metadata using Apache Solr.	Implemented an indexing system using Apache Solr, focusing on metadata extraction, indexing, and search functionalities.	Enhanced search efficiency and accuracy for scientific metadata.	Facilitates better data retrieval and management in scientific databases.
Resource allocation based on genetic algorithm for cloud computing Chen et al. (2021)	To address overloading problems in Virtual Machines (VMs)	Utilizes a Markov chain model to predict system states and GA for resource allocation to achieve load balance.	Achieved load balance for VMs and PMs by using the proposed GA method.	Improved system performance and resource utilization in cloud computing environments.
Analysis on resource allocation for parallel processing and scheduling in cloud computingKumar et al. (2021)	To analyze and improve resource allocation strategies for parallel processing and scheduling in cloud computing	Reviews and compares various resource allocation strategies including TARA and ACO.	Identified TARA and ACO as effective strategies for resource allocation in cloud environments.	Provides insights into the selection of resource allocation strategies to optimize performance in cloud computing.
Machine learning based workload prediction for auto-scaling cloud applications Singh et al. (2023)	To develop a machine learning model to predict workloads for better auto-scaling in cloud applications.	Utilizes machine learning techniques to analyze historical workload data and predict future demands.	ML models significantly improve the accuracy of workload predictions, leading to more efficient auto-scaling and resource utilization.	Enhances auto-scaling mechanisms in cloud environments, reducing costs and improving performance by anticipating and responding to workload changes more effectively.

Table 1: Summary of Previous Research Work on Cloud Computing Resource Allocation.

Some of the related works discussed in literature are Bhanuprakash and Sunitha (2018) which proposed an agent-based adaptive resource allocation algorithms for cloud computing. This is dissimilar from the present work as their work is structured mainly on agent-based negotiation not Solr query rates. Similarly, Singh et al. (2023) have come up with a machine learning-based workload prediction for auto-scaling in cloud applications. However, their research is based on historical workload and predicted workload from the machine learning approach rather than the current Solr query metrics which has been discussed here. The key difference lies in the application-specific focus and the direct in-tegration with Solr's query handling to optimize performance.

3 Methodology

3.1 Methodology

This research adopts a practical approach to developing and evaluating a dynamic scaling mechanism for Apache Solr in preallocated resource environments, specifically utilizing the Azure Cloud Platform. The methodology is divided into three primary stages: system setup, algorithm development, and evaluation.

3.2 System Setup

The system is built on the Azure Cloud Platform using several core services:

- **Azure Virtual Machine Scale Set (VMSS):** This service is used to automatically manage and scale the number of virtual machines based on demand.
- **Azure App Registration:** This process allows secure authentication and authorization of the application in Azure, facilitating interactions with other Azure services.
- **Azure Compute Gallery:** Employed to store and deploy VM images efficiently.
- **Apache Zookeeper:** It is a critical component of SolrCloud, Solr's distributed search platform. It serves as the central nervous system for managing and coordinating the Solr cluster.
- **Apache Solr:** The open-source search platform is set up in a SolrCloud configuration for distributed search and indexing.

The Solr cluster is initially configured manually on the Azure platform, where nodes are added or removed based on application demand. This setup serves as the baseline for evaluating the effectiveness of the dynamic scaling mechanism.

3.3 Algorithm Development

The core of the research lies in the development of a dynamic scaling algorithm designed to respond to changes in query rates. The algorithm is designed to monitor the query rate over intervals of 1 minute, 5 minutes, and 15 minutes, and to adjust the number of active Solr nodes accordingly. The algorithm operates as follows:

- **Data Collection:** Metrics such as query rates are continuously collected from the Solr nodes.
- **Threshold Definition:** Predefined thresholds are established for different metrics (e.g., query rate per minute) to trigger scaling actions.
- **Scaling Actions:** Based on the collected data and defined thresholds, the algorithm determines whether to scale in (remove nodes) or scale out (add nodes).
- **Deployment:** The algorithm automatically adjusts the number of nodes in the Solr cluster using the Azure VMSS API, ensuring that resources are aligned with current demand.

Figure 1 illustrates the flow diagram of the scaling algorithm and its operation.

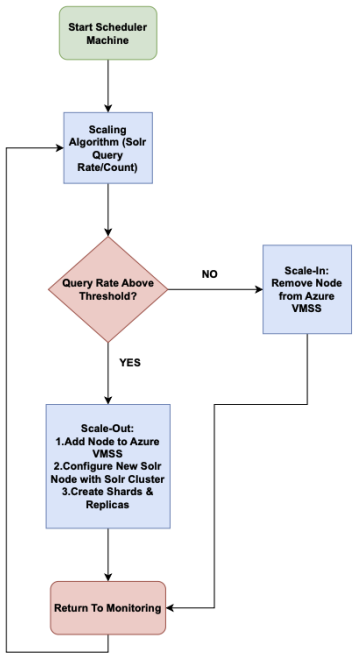

Figure 1: System Architecture With Auto-scaling Mechanism

3.4 Metrics for Evaluation

For the evaluation, the following performance metrics will be analyzed:

- Mean Rate: The average rate at which requests are processed by the system.
- Mean Response Time (ms): The average time (in milliseconds) taken to process a request.
- Median Response Time (ms): The midpoint value of response times, indicating the typical request processing time.
- 95th Percentile Response Time (p95): The response time below which 95% of requests are processed, providing insight into the system's performance under higher loads.

These metrics provide a comprehensive understanding of how well the dynamic scaling mechanism optimizes resource usage, maintains application performance in a cloud environment and eventually reduce infra costs.

3.5 Justification of Methodology

The chosen methodology is justified by the need to address the inefficiencies of traditional scaling methods in preallocated resource environments, particularly for applications like Apache Solr that exhibit non-linear resource usage patterns. Traditional scaling methods, which rely on host metrics such as CPU and memory usage, are insufficient for environments where resources are preallocated and fixed.
By focusing on query rates and response times as key metrics, the dynamic scaling algorithm is better aligned with the operational characteristics of Apache Solr, leading to more efficient resource usage and improved application performance. The use of Azure's robust cloud services further supports the practical implementation and evaluation of the proposed solution in a real-world cloud environment.

4 Design Specification

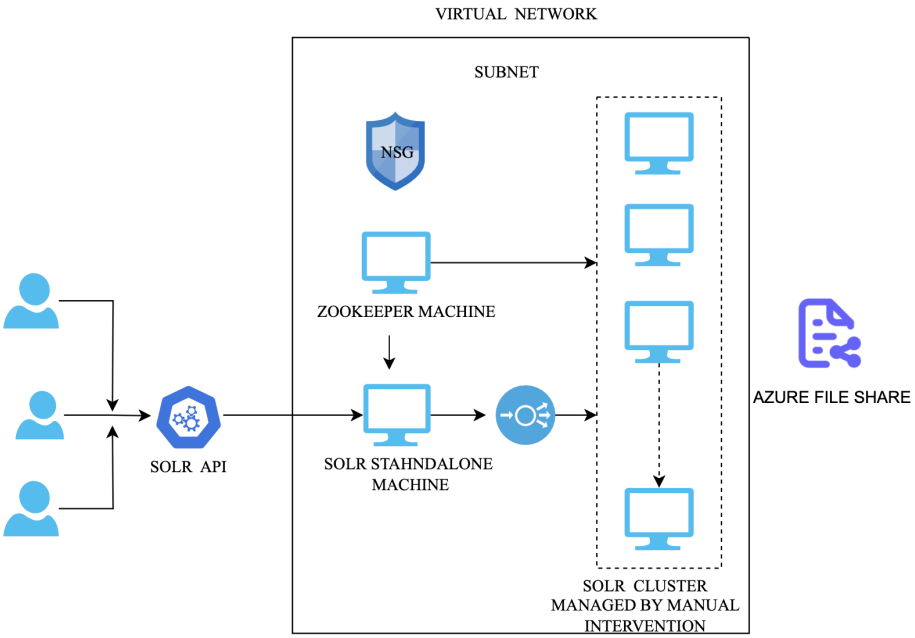

Figure 2: System Architecture Without Auto-scaling Mechanism

System architecture plays significant role in resources opimization and cost optimization. The Figure 2 below; which shows the original Solr cluster scaling manually when nodes are added or removed in order to correspond with application performance. While this method may help in the short-term resolve some performance problems it is manual

and time-consuming and thus cannot be used in the long run. Futhermore, if the cluster is not scaled down, it leads to additional cost during periods where the load on the system is low and is considered or wastage of resources.

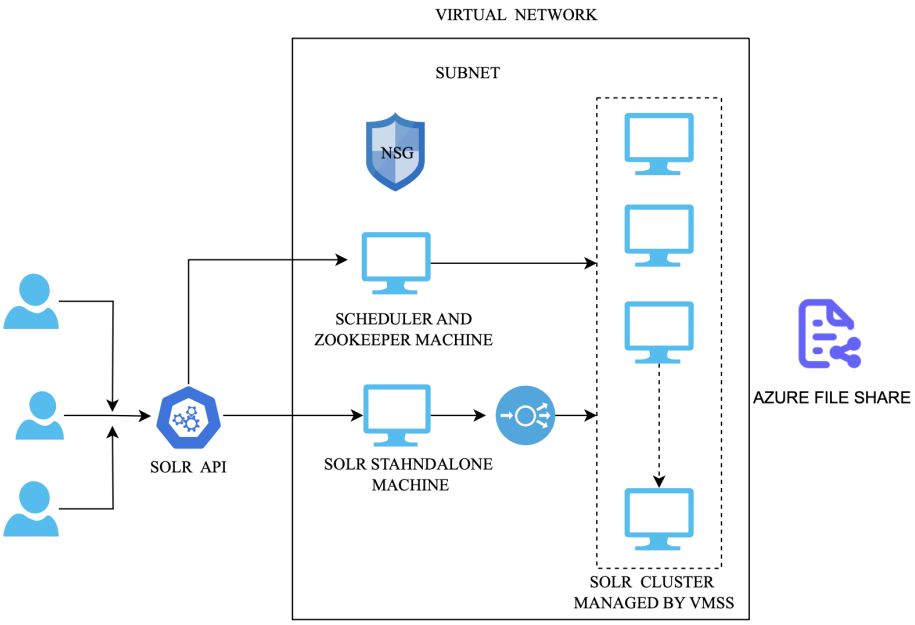

Figure 3: System Architecture With Auto-scaling Mechanism

Figure 3 highlights the system architecture that contains an auto-scaling mechanism. In this setup, there is a monitoring algorithm/script resident on the scheduler node that monitors one or several Solr metrics, in this case the query rate. When these metrics get to these set thresholds, the algorithm captures it and adds or removes Solr nodes on the Apache Solr Cluster which is a Azure VMSS to ensure resource optimization as well as avoid wastage without the need for the administrator to do it. Also, the algorithm is responsible for creation of new shards and replicas on each Solr node, which again improves the performance and scalability of the system.

4.1 Scaling Algorithm/Script

This script is designed to manage and automate scaling for a Solr search cluster based on various performance metrics. It uses Azure Virtual Machine Scale Sets (VMSS) and Solr utilities to monitor performance, make scaling decisions, and perform scaling operations. The script also integrates with Kibana for additional performance insights.

The script automates the process of scaling a Solr cluster by:

- Monitoring performance metrics from Solr and Kibana.
- Determining scaling needs based on these metrics.

- Executing scaling operations (both scale-in and scale-out) on the Azure VMSS.
- Performing cleanup to remove unused resources and log the operations.

Below is the code snippet for the monitoring algorithm:

```
# Import necessary modules and utilities
IMPORT necessary modules for handling JSON, time, date, collections, and custom utilities like AzureVmss, SolrUtil, etc.

# Define a function to get the timestamp of the last 'n' minutes
FUNCTION get_timestamp_last_n_mins(n):
    RETURN current_time_minus_n_minutes in format 'YYYYMMDDHHMMSS'

# Define a function to handle Kibana response time scaling
FUNCTION kibana_resp_time_scaling(key):
    LOG the input key
    INITIALIZE a result dictionary with 'scale_in' and 'scale_out' set to True
    INITIALIZE avg_key_res_time as None
    IF key is 'clustering':
        CALCULATE last_n_min_timestamp
        FETCH avg_key_res_time for clustering from Kibana logs
        SET low_threshold and high_threshold for clustering
    ELSE IF key is 'suggestion':
        CALCULATE last_n_min_timestamp
        FETCH avg_key_res_time for suggestion from Kibana logs
        SET low_threshold and high_threshold for suggestion
    ELSE:
        SET avg_key_res_time to 0.0
        SET low_threshold and high_threshold to 0 and 100 respectively

    IF avg_key_res_time is None:
        SET avg_key_res_time to 1
        SET low_threshold and high_threshold to 0 and 2

            DETERMINE scaling actions based on avg_key_res_time, low_threshold, and high_threshold
    RETURN avg_key_res_time and result dictionary

# Define a function to calculate average search rates for shards
```

```
FUNCTION avg_search_rates_for_shards(collection_search_rates):
    INITIALIZE shard_rates as an empty dictionary
    LOG the collection search rates

    FOR each shard_replica and rates in collection_search_rates:
        SPLIT shard_replica to get shard and replica
        APPEND rates to shard_rates[shard]

    LOG shard rates

    INITIALIZE avg_shard_rates as an empty dictionary
    FOR each shard and rates_list in shard_rates:
        INITIALIZE lists for one_mins, five_mins, and fifteen_mins
        FOR each dict_obj in rates_list:
            APPEND corresponding rate values to lists

        CALCULATE average rates and store in avg_shard_rates
    RETURN avg_shard_rates

# Define a function to find hot and cold shards for scaling
FUNCTION find_hot_cold_shards(azure_obj, solr_obj):
    TRY:
        GET current machines from Azure
    EXCEPT:
        LOG error and SET machines_ip_ids to an empty dictionary

    LOG current VMs

    IF machines_ip_ids is not empty:
        FETCH VM shard status and shard IP mapping from Solr
    ELSE:
        SET vm_cores and shard_ip_mapping to empty dictionaries

    LOG current shard to VM distribution

    FETCH search rates from Solr
    INITIALIZE avg_search_rates and scale_op as empty dictionaries

    FOR each collection in collections_to_monitor:
        FETCH search rates for the collection
        CALCULATE avg search rates for shards
        STORE avg search rates in avg_search_rates
```

```
75
76          LOG avg search rates
77
78      FOR each shard and rates in shard_search_rates:
79          DETERMINE the number of required replicas based on
                search rates
80          STORE the result in scale_op
81      RETURN scale_op and avg_search_rates
82
83  # Main execution block
84  IF __name__ == '__main__':
85      INITIALIZE start_time and as_log dictionary
86
87      TRY:
88          SETUP logger and log initial configurations
89          UPDATE as_log with initial settings
90          IF SKIP_KIBANA is False:
91          TRY:
92              PERFORM Kibana response time scaling for
                    clustering and suggestion
93              DETERMINE scale_in_flag and scale_out_flag based
                    on results
94          EXCEPT:
95              LOG error and SET scale_in_flag and
                    scale_out_flag to True
96      ELSE:
97          SET scale_in_flag and scale_out_flag to True
98      UPDATE as_log with scaling decisions
99
100     IF scaling is needed (scale_in_flag or scale_out_flag):
101         INITIALIZE AzureVmss and SolrUtil objects
102         FIND hot and cold shards
103         UPDATE as_log with scaling operations
104
105         IF scale_in_flag:
106             PERFORM scale in operation
107         ELSE:
108             LOG no scale in required
109         UPDATE as_log with scale in results
110
111         IF scale_out_flag:
112             SLEEP for a short duration
113             PERFORM scale out operation
114         ELSE:
115             LOG no scale out required
116         UPDATE as_log with scale out results
117     ELSE:
118         LOG no scaling needed
```

```
    CLEAN UP if required:
        SLEEP for a short duration
        FIND and remove empty nodes
        DELETE VMs if necessary
        UPDATE as_log with removed VMs
    GET current machines and update as_log
    LOG the results using clog

    EXCEPT:
        LOG error and update as_log with error details
        LOG the results using clog
    LOG finished execution
```

5 Implementation

This implementation plan outlines the steps for setting up and configuring the components necessary for the project, including the Scheduler/Zookeeper VM, Standalone Solr VM, Solr base image creation, Azure VM Scale Set (VMSS), and App Registration. This plan ensures the proper setup and configuration of all necessary components for effective scaling and management of the Solr deployment.

5.1 Scheduler And Zookeeper Virtual Machine

Package Installation: Download and extract ZooKeeper packages. Install OpenJDK using the system package manager.
Configuration Setup: Verify and create the ZooKeeper configuration file (zoo.cfg). Ensure it includes necessary parameters like tickTime, dataDir, and clientPort.
Permissions and Directory Setup: Check and set appropriate permissions for the ZooKeeper directories and files.
Service Verification: Start ZooKeeper and check its status. Troubleshoot any issues by reviewing the ZooKeeper logs.

5.2 Setup Standalone Solr Machine

Java Installation: Install OpenJDK on the standalone Solr VM.
Solr Installation: Download and extract the Solr package. Adjust the configuration to allow remote access to the Solr Admin Dashboard.
Service Configuration: Start the Solr service in cloud mode and configure it to connect to the ZooKeeper instance.

5.3 Create Solr Base Image on Azure

Machine Deployment: Deploy a new VM for Solr setup to avoid interfering with the original VM. Perform the standalone Solr setup steps on this VM.
Automation Setup: Configure the rc.local file to ensure Solr starts automatically on VM deployment. Set necessary permissions and start the rc-local service.

Image Creation: Capture an image of the Solr VM through the Azure portal. Ensure it is shared to an Azure compute gallery and configured as "Specialized."

5.4 Virtual Machine Scale Set (VMSS) Using Solr Base Image

Image Selection: Access the Azure compute galleries dashboard, choose the Solr base image, and select the relevant image definition and version.
VMSS Creation: Create a new VMSS with the selected image. Set the orchestration mode to Uniform and manually adjust the scaling capacity. Ensure the VMSS is in the same virtual network (VNet) as the standalone Solr VM for proper communication.

5.5 App Registration to Access and Control Solr VMSS

App Registration: Register a new application in Azure to manage and control the VMSS. Create and securely store application secrets.
CLI Configuration: Install Azure CLI on the scheduler machine and configure it using the application registration details for authentication.

5.6 Schedule Cron Job to Run Monitoring Script Every 15 Minutes

Create a cron job on the scheduler machine to run the monitoring script (`solr_autoscaling.py`) every 15 minutes.

5.7 API URL for Monitoring Solr Admin Dashboard and Solr Performance Metrics

Apache Solr Admin Dashboard URL: `http://<Solr-StandAloneIP>:8987/solr/#/~cloud`
Apache Solr Performance Metrics URL: `http://<Solr-StandAloneIP>:8987/solr/admin/metrics?regex=QUERY%5C./select%5C.requestTimes.*`

6 Evaluation

The evaluation of our system focuses on comparing the performance metrics under different configurations: with and without the auto-scaling mechanism. We conducted experiments to understand how auto-scaling impacts query performance and resource utilization. Below, we present and analyze the results from our two key experiments.

6.1 Experiment / Case Study 1

Performance metrics at different query rates without Auto-Scaling Mechanism:

Figures 4 and 5 depict the system configuration, highlighting the nodes and shards setup without an auto-scaling mechanism. The cluster consists of a single Solr node with 6 replicas. Due to the static configuration, the system is prone to performance bottlenecks

as the load increases. This underscores the importance of implementing dynamic scaling solutions to ensure optimal performance.

Avg Query Request Per Minute	500 Requests/Min	1000 Requests/Min
Avg meanRate	1.4	2.08
Avg mean_ms	12.64	15.09
Avg median_ms	10.37	13.166
p95_ms	19.40	23.87

Table 2: Performance metrics at different query rates without Auto-Scaling Mechanism.

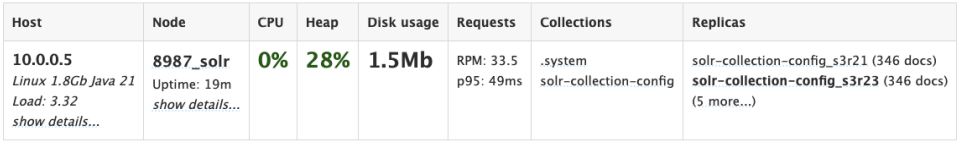

Figure 4: Solr Cluster Nodes Without Auto-scaling Mechanism

Figure 5: Shards and Replicas Without Auto-scaling Mechanism

In this experiment, we analyzed the performance metrics of our system operating without an auto-scaling mechanism. The metrics were evaluated at two different query rates: 500 requests per minute and 1000 requests per minute.
The data shows that as the query rate increased from 500 to 1000 requests per minute, there was a noticeable increase in the average meanRate, mean response time (mean_ms), median response time (median_ms), and 95th percentile response time (p95_ms). Specifically, the mean response time increased from 12.64 ms to 15.09 ms, and the 95th percentile response time rose from 19.40 ms to 23.87 ms.

6.2 Experiment / Case Study 2

Performance metrics with Auto-Scaling Mechanism:

In the second experiment, we evaluated the performance metrics of our system with the auto-scaling mechanism enabled, again at a query rate of 1000 requests per minute.

Figure 6 illustrates the system's behavior after implementing the scaling algorithm. When the meanRate metric met the scaling threshold, the algorithm responded by adding a new node and creating 2 additional replicas to accommodate the increased demand. This demonstrates the effectiveness of auto-scaling in optimizing resource allocation and performance.

Host	Node	CPU	Heap	Disk usage	Requests	Collections	Replicas
10.0.0.5 *Linux 1.8Gb Java 21* *Load: 1.42* *show details...*	8987_solr Uptime: 30m *show details...*	0%	53%	1.5Mb	RPM: 40.98 p95: 40ms	.system solr-collection-config	solr-collection-config_s3r21 (346 docs) **solr-collection-config_s3r23** (346 docs) (5 more...)
10.0.0.8 *Linux 1.8Gb Java 21* *Load: 0* *show details...*	8987_solr Uptime: 10m *show details...*	0%	62%	516.6Kb	RPM: 0.01 p95: 19ms	solr-collection-config	solr-collection-config_s1r29 (334 docs) solr-collection-config_s2r31 (320 docs)

Figure 6: Solr Cluster Nodes With Auto-scaling Mechanism

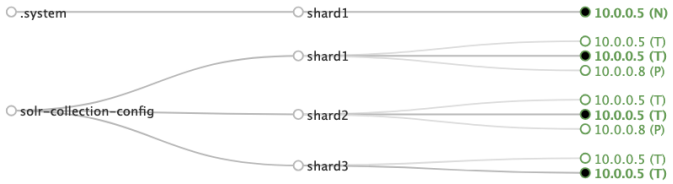

Figure 7: Shards and Replicas With Auto-scaling Mechanism

With the auto-scaling mechanism enabled, the performance metrics show improvements compared to the non-auto-scaling scenario. The average meanRate increased to 3.11, while the average response time (mean_ms) decreased to 11.46 ms, and the 95th percentile response time improved to 18.7 ms.

Avg Query Request Per Minute	**1000 Requests/Min**
Avg meanRate	3.11
Avg mean_ms	11.46
Avg median_ms	10.22
p95_ms	18.7

Table 3: Performance metrics with Auto-Scaling Mechanism.

6.3 Comparison Evaluation of two case studies

The evaluation clearly shows that the auto-scaling mechanism enhances the system's ability to handle higher loads more efficiently compared to a static configuration.

Figures 8 and 9 illustrate the significant improvements in performance metrics following the implementation of the dynamic scaling mechanism. These enhancements are evident in both response time and throughput, underscoring the effectiveness of this approach in maintaining optimal system performance.

- Increased Mean Rate: The addition of more nodes and replicas has led to an increase in the mean rate, enabling the cluster to handle a greater number of queries in parallel. This enhancement allows the system to process more queries per minute, provided the load balancer effectively distributes the queries across the nodes.
- Decreased Response Times: The metrics for mean, median, and p95 response times have all decreased, indicating a higher likelihood of faster query responses. This improvement suggests that the dynamic scaling mechanism has successfully enhanced the system's ability to serve queries more efficiently.

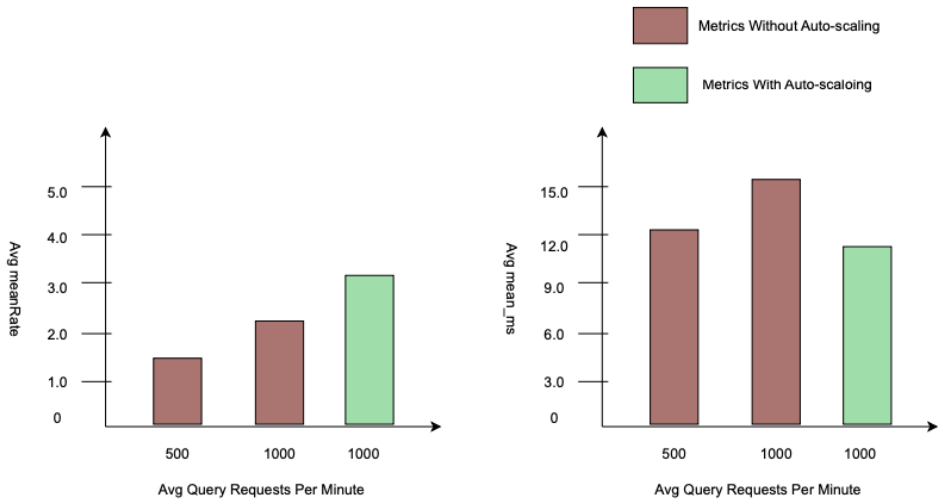

Figure 8: Solr Performance Metrics

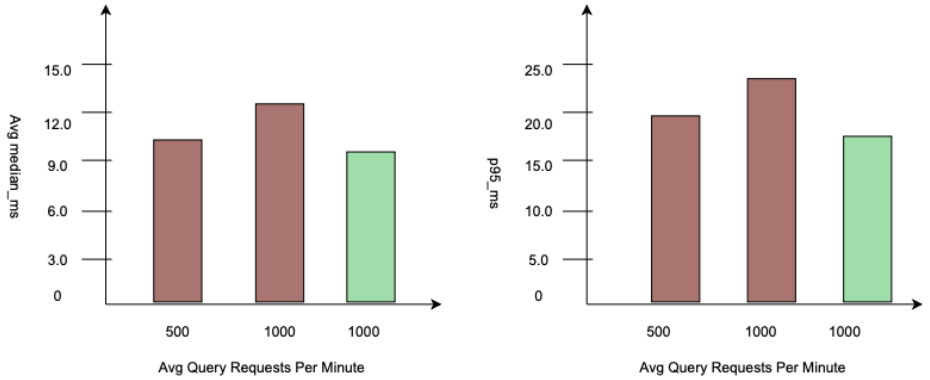

Figure 9: Solr Performance Metrics

7 Conclusion and Future Work

In this research, we have successfully implemented a dynamic infrastructure scaling mechanism within a preallocated resource environment using Apache Solr on the Azure Cloud Platform. The primary objective was to enhance resource utilization and improve application performance by dynamically adjusting the size of the Apache Solr cluster based on query metrics. Our approach addresses the limitations of traditional scaling methods, which often rely on static host metrics and require manual intervention, leading to resource wastage and suboptimal performance.

The proposed dynamic scaling algorithm demonstrated a significant improvement in resource efficiency, ensuring that the Solr cluster could handle varying query loads without unnecessary resource consumption. By automating the scaling process and integrating

key Azure services such as Virtual Machine Scale Sets (VMSS), Azure File Share, and Azure Compute Gallery, we provided a robust solution that not only maintains high performance during peak periods but also reduces operational costs during low-demand periods.

This research contributes to the field of cloud computing by offering a practical deployment strategy that can be adapted to other similar environments requiring efficient resource management. Future work could explore the application of this dynamic scaling approach in different cloud platforms or with other distributed systems beyond Apache Solr, as well as the integration of more advanced machine learning techniques to further optimize scaling decisions.

Overall, the dynamic infrastructure scaling mechanism presented in this study offers a sustainable and efficient approach to managing resources in cloud-based applications, providing a foundation for further innovation in the field of cloud resource management.

Building on the current research, several avenues can be explored to enhance the system's capabilities and address its limitations. One promising direction is the integration of Apache Solr with advanced monitoring tools such as Prometheus, Grafana, and ELK (Elasticsearch, Logstash, Kibana). These tools would offer more sophisticated historical data analysis and provide a more intuitive user interface, enabling more informed decision-making. By visualizing metrics over time and setting up real-time alerts, the system could become more responsive to performance changes, leading to more effective resource scaling.

Another area for future improvement lies in addressing the current system's limitations with load distribution. The existing setup does not evenly distribute the load across all Solr replicas, which can lead to suboptimal performance and resource utilization. Implementing advanced load balancing techniques could ensure a more even distribution of query loads, thereby enhancing the overall efficiency and reliability of the system. Exploring these enhancements would significantly contribute to the robustness and scalability of Apache Solr in cloud-based environments, paving the way for more efficient data management solutions.

References

Bhanuprakash, T. and Sunitha, N. (2018). Agent based adaptive resource allocation algorithms for cloud computing, *Proceedings of the 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 845–851.

Bhardwaj, T., Upadhyay, H. and Sharma, S. C. (2019). Autonomic resource allocation mechanism for service-based cloud applications, *2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pp. 183–187.

Chen, J. (2018). A cloud resource allocation method supporting sudden and urgent demands, *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, pp. 66–70.

Chen, Y.-L., Huang, S.-Y., Chang, Y.-C. and Chao, H.-C. (2021). Resource allocation based on genetic algorithm for cloud computing, *2021 30th Wireless and Optical Communications Conference (WOCC)*, pp. 211–212.

cynthn (2024). Overview of virtual machines in azure - azure virtual machines.
URL: *https://learn.microsoft.com/en-us/azure/virtual-machines/overview*

Guntupally, K., Dumas, K., Darnell, W., Crow, M., Devarakonda, R. and Giri, P. (2020). Automated indexing of structured scientific metadata using apache solr, *2020 IEEE International Conference on Big Data (Big Data)*, pp. 5685–5687.

ju shim (2023). Overview of azure compute gallery - azure virtual machines.
URL: *https://learn.microsoft.com/en-us/azure/virtual-machines/azure-compute-gallery*

ju shim (2024). Azure virtual machine scale sets overview - azure virtual machine scale sets.
URL: *https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview*

Kumar, P., Tharad, A., Mukhammadjonov, U. and Rawat, S. (2021). Analysis on resource allocation for parallel processing and scheduling in cloud computing, *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*, pp. 1–6.

rwike77 (2023). Apps service principals in microsoft entra id.
URL: *https://learn.microsoft.com/en-us/entra/identity-platform/app-objects-and-service-principals?tabs=browser*

Singh, S. T., Tiwari, M. and Dhar, A. S. (2023). Machine learning based workload prediction for auto-scaling cloud applications, *2022 OPJU International Technology Conference on Emerging Technologies for Sustainable Development (OTCON)*, pp. 1–6.