**VISVESWARAYA TECHNOLOGICAL UNIVERSITY Jnana Sangama, Belagavi-590018**



**TECHNICAL SEMINAR REPORT ON**

**“Performance of Kubernetes”**

***Submitted in partial fulfillment of the requirements for the award of degree of***

**BACHELOR OF ENGINEERING**

**In**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted By**

**Prakhyat**

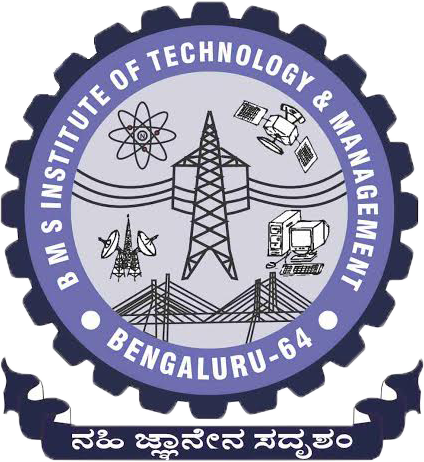
**1BY18CS108**

**Under The Guidance Of**

**Mrs Durga Bhavani A**

**Asst Professor**

**Department of CSE**

****

**BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

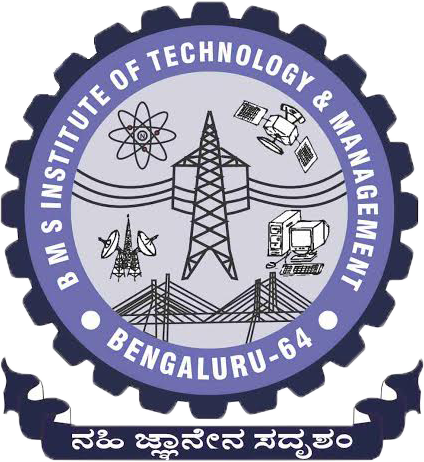
**Avalahalli , Yelahanka , Bengaluru – 560064.**

**2021-2022**

**VISVESWARAYA TECHNOLOGICAL UNIVERSITY Jnana Sangama, Belagavi-590018**

**BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**Avalahalli , Yelahanka , Bengaluru – 560064.**

****



This is to certify that the Seminar work entitled “Title” has been carried out by Mr/Ms. Student name, USN, a bonafide student of BMS Institute of Technology and Management in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belagavi during the year 2021-2022. It is certified that all corrections/suggestions indicated for assessment have been incorporated in the report deposited in department library. The Seminar report has been approved as it satisfies the academic requirements in respect of Seminar work prescribed for the said degree.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Signature of the Guide Signature of the HOD**

**Guide Name Dr. Bhuvaneshwari C M.**

**Designation Professor& Head of department**

**Department of CSE Department of CSE**

**BMSIT & M BMSIT & M**

**DECLARATION**

I, student name **[USN: ]**, student of VIII Semester BE, in Computer Science and Engineering, BMS Institute of Technology and Management hereby declare that the Seminar entitled “TITLE**”** has been carried out by me and submitted in partial fulfillment of the requirements for the *VIII Semester degree of* ***Bachelor of Engineering in Computer Science and Engineering*** *of Visvesvaraya Technological University, Belagavi* during academic year 2021-22.

Date : **Student Name Prakhyat**

Place : U**SN : 1BY18CS108**

**ACKNOWLEDGEMENT**

We are happy to present this Seminar report after completing it successfully. This seminar would not have been possible without the guidance, assistance and suggestions of many individuals. We would like to express our deep sense of gratitude and indebtedness to each and every one who has helped us make this a success.

We heartily thank our **Principal, Dr. MOHAN BABU G N,BMS Institute of Technology & Management,** for his constant encouragement and inspiration in taking up this project.

We heartily thank our **Head of the Department, Dr. Bhuvaneshwari C.M , Department of Computer Science and Engineering, BMS Institute of Technology &Management,** for her constant encouragement and inspiration in taking up this project.

We gracefully thank our Seminar Guide, **name of guide, designation, Department of Computer Science and Engineering,** for his/her guidance, support and advice.

Special thanks to all the staff members of Computer Science Department for their help and kind co-operation.

Lastly we thank our parents and friends for the support and encouragement given to us in completing this precious work successfully.

**Prakhyat(1BY18CS108)**

**ABSTRACT**

Container-based virtualisation technologies are gaining more and more traction in recent years across Cloud platforms and this will likely continue in the coming years. As such, container orchestration technologies are becoming indispensable. Kubernetes has become the de facto standard because of its robustness, maturity and rich features. To free

users of the burden of having to configure and maintain complex Kubernetes infrastructures, but still make use of its functionalities, all major Cloud providers are now offering cloud-native managed Kubernetes alternatives. The goal of this presentation is to investigate the performance of containers running in such hosted services. For this purpose, we conduct a series of experimental evaluations of containers to monitor the behavior of system resources including CPU, memory, disk and network. A baseline consisting of a manually deployed Kubernetes cluster was built for comparison. In particular, we consider the Amazon Elastic Container Service for Kubernetes (EKS), Microsoft Azure Kubernetes Service (AKS) and Google Kubernetes Engine (GKE).

The Australia-wide NeCTAR Research Cloud was used for the baseline

**CONTENT**

.

**TOPIC PAGE NO**

1. INTRODUCTION 7
2. LITERATURE SURVEY 9
3. METHODOLOGY 12

3.1 Requirement Recognition 13

3.2. Service Feature Identification 13

3.3. Metrics/Benchmarks Listing and Selection 16

3.4. Experimental Factors Listing and Selection 18

3.5. Experimental Design 18

1. RESULTS 20

4.1Computing Performance 20

4.2Memory Performance 22

4.3Disc Performance 23

4.4 Network Performance 25

1. CONCLUSIONS AND FUTURE ENHANCEMENT 27
2. REFERENCE 28

**CHAPTER 1: INTRODUCTION**

Cloud Computing is having a dominant role in the way applications are designed, built, and deployed. According to a recent Gartner Hype Cycle report, Cloud Computing has reached the stage of Slope of Enlightenment. This represents a stage where the assessed technology becomes more broadly understood, and newer generations of technology start to emerge that build upon Clouds. One such core technology that is gaining more and more traction in recent years is container based virtualisation. Container-based virtualisation, or containerisation, refers to a methodology of encapsulating an application, along with its dependencies in such a way that it can run efficiently across different computing environments. Compared to traditional virtualisation techniques (i.e., hypervisor-based solutions) where hardware and device drivers are virtualised by a virtualisation layer, container-based virtualisation offers a more lightweight alternative. Instead of running a complete Virtual Machine (VM) with its own operating system (OS) on top of virtualized hardware, containers provide an isolated environment for system resources (e.g., processes, file systems, and networks) to run at the host OS level. This means that multiple containers can share the same OS kernel, hence start much faster, and use just a fraction of the system memory compared to booting an entire virtualized OS. Several studies have demonstrated that this approach results in lower overheads and the overall performance of the containerised application is optimized. Moreover, because applications are broken down into small, independent units usually referred to as microservices, the scalability and portability of containers is increased.

However, as the use of containers at significant scale increases, the need for management tools to control the automation tasks of deployment, scaling, and overall operation of containerised applications across the infrastructure increases. An illustrative example of how indispensable orchestration tools are becoming comes from Google, the creators of the open source container orchestration tool Kubernetes. They run all of their services, across all of their business units in containers, and it has been reported that they start over two billion containers every week . Gartner identifies that container management technologies are at the peak of expectations within the Hype Cycle for Cloud Computing . This is worth noting as it indicates that the Container

Orchestration market is becoming very attractive and competitive, and will probably

continue to expand as more and more organizations consider moving towards container-based approaches. In this context, a range of solutions have arisen, especially around the two main container offerings: Docker and Kubernetes, which have rapidly become the de facto standards for container runtime and container orchestration respectively .

Despite its proven efficiency at scale, configuring and main training a Kubernetes infrastructure can be a daunting task. Its multiple components and inherent complexity have a steep learning curve that can be difficult to manage. This has led to a new sub-market emerging within the container management fifield: Kubernetes hosted solutions. Since an increasing number of Cloud service providers are competing to provide better and less complicated solutions for Kubernetes, evaluating these offerings is increasingly crucial for service customers, e.g., cost-benefit analysis, as well as for providers, e.g., for direction of future improvements .

we focuses on evaluating the performance characteristics of containers running on different managed Kubernetes services. Our goal is to investigate if some level of overhead is introduced by these hosted services when Docker containers are running. In order to do so, an experimental benchmark is performed. First, a baseline is created to understand the average expected performance of manually deployed Kubernetes clusters running on the same clouds. Well-known open-source benchmarking tools are used to monitor the performance of standard computing resources such as system memory, CPU, disk, and network. To guide our evaluation, we rely on the Cloud Evaluation Experiment Methodology (CEEM).

**CHAPTER 2: LITERATURE SURVEY**

Following the rise of containers and container management frameworks, there has been increasing interest from the research and academic community on evaluating these technologies. Although container-based systems have already been proposed as a faster alternative to hypervisor-based systems for some time [5], they have gained considerable attention with the emergence of Docker (in 2013) [16]. Previous studies demonstrated that containers offer a more lightweight alter native compared to traditional hypervisor-based virtualisation [6]–[9]. In their study, Felter et al. [7] measured the CPU,

memory, storage, and network overheads of Docker containers compared to KVM hypervisors. They showed that in their experiments, containers performed equal or better than VMs in almost all scenarios. A similar comparison was made in [9], but in this case, the overall performance of containers was determined by monitoring how many requests a front end application server deployed on a container could handle in comparison to a similar application deployed on a VM. The results showed that Docker containers outperformed VM significantly (by a factor of five).

Xavier et al. [17] evaluated the performance of other Linux container-based virtualisation implementations including Linux VServer, OpenVZ and Linux Containers (LXC). Their goal was to determine the suitability of these technologies in high performance computing (HPC) environments since traditional virtualisation techniques have long been prohibitive

due to their often considerable performance overheads. In addition to conducting specific experiments for HPC applications, they used several benchmarking tools to measure

CPU, system memory, disk, and network performance. In this regard, they demonstrated that all container-based systems presented near-native performance. However, based on their findings, they also concluded that those evaluated containers technologies had certain immaturity with regards to isolation and security, especially for memory, disk, and network. Further studies by Xavier et al. [18], [19] revealed that, even though significant improvements were found in terms of performance isolation for MapReduce workloads, when evaluating disk intensive workloads, LXC failed to provide complete isolation

of resources, unlike traditional hypervisor-based systems.

Morabito et al. [6] presented a performance evaluation of KVM, LXC, and Docker based on CPU, memory, disk I/O, and network I/O. The authors ran several benchmarking tools to assess the performance of these components and concluded that the overhead introduced by container-based virtualisation technologies could be considered almost negligible, although the performance was traded-off with security. Numerous other studies have been undertaken where it was established that container-based systems have inherent advantages over traditional hypervisor-based systems due to their performance improvements, reduced start-up time, scalability, and portability [8], [20], [21]. What this suggests is that containers are playing a predominant role in the Cloud industry and will probably continue to expand their presence. [8], [19], [22]–[24] identify that container-based virtualisation is undoubtedly becoming the basis for the next generation of Cloud Computing, especially in Platform-as-a-Service (PaaS) environments.

Despite such literature, the majority of the performance evaluations has focused on comparing the performance of containerised workloads against their VM-based counterparts. However when it comes to Cloud Computing, it is most likely that a combination of these technologies will be utilized, i.e., containers running on top of VMs. Several studies have explored this topic. For example, in [25], a detailed performance comparison was made between two container-based technologies, namely Docker and Flockport (LXC), running on VMs provisioned by the Australia-wide NeCTAR Research Cloud [26]. As in other studies, several benchmarking tools were utilized to assess the performance overheads of various system resources including CPU, memory, network I/O, and disk I/O, when operating inside the container. In their results, Flockport showed slight performance improvements over Docker in some scenarios. In [27], the authors argue that running containers on top of VMs should be encouraged due to a range of benefits such as easier management and upgrading of nodes, and the possibility to overcome software driver incompatibilities present with bare metal resources. In their experiments, the performance cost was confirmed when measuring CPU, memory, disk, and network overheads.

Disregarding the performance benefits of running containerised workloads directly on physical servers, we argue that deploying containers on VMs can certainly be beneficial,

especially in Cloud environments. Since Cloud-based infrastructures rely on virtualisation technologies, this makes them a suitable environment to efficiently scale up or down nodes, e.g., nodes in a Kubernetes cluster. The elasticity offered by the different Cloud service providers allows users to provision computing resources on demand, and pay only for what they consume. Furthermore, redundancy across different geographical zones can be crucial, and this is more straightforward, affordable, and faster to provision using VMs compared to physical environments.

Cloud service providers are increasingly leveraging such aspects and an increasing number of Cloud-native options have emerged. In this regard, managed Kubernetes services are becoming mainstream offerings [28]. Having Kubernetes running on the Cloud not only facilitates auto-provisioning of worker nodes, i.e., VMs that run containers, but it also takes away the burden of having to manage and maintain complex

clusters and associated infrastructure. Moreover, these hosted services integrate with the Cloud-native features adding even more value to the already profitable Infrastructure-as-a-Service(IaaS) market.

In this study, we focus on evaluating the performance of Docker containers running on some of the most widely adopted Kubernetes-based cloud services [28]. Specifically we

consider the Amazon Elastic Container Service for Kubernetes (EKS) [29], the Azure Kubernetes Service (AKS) [30], and Google Kubernetes Engine (GKE) [31]. To the best of our knowledge, despite the popularity of Kubernetes, there is no other study that systematically compares these solutions from a performance perspective. The closest works to ours are [32] and [33]. Both of these works evaluate the performance of the two most popular open source container management tools: Docker Swarm [34] and Kubernetes. Although they undertake different benchmarking techniques, they both conclude that Docker Swarm outperforms Kubernetes. However, Swarm is not as mature and robust as Kubernetes, especially from a feature perspective. We undertake a similar approach as others and measure the performance overheads introduced by these platforms and compare them to a non-managed Kubernetes cluster deployed on the same clouds. We present the methodology to achieve this in the next section.

**CHAPTER 3: METHODOLOGY**

Evaluating Cloud services has its inherent challenges. Customers often have little knowledge or understanding about how different Cloud services are built, or how they may adjust to their particular needs . Moreover, because these services are being conceived in an ever-changing environment, they are subject to continuous upgrades with new features being added continually. Hence establishing a reproducible and robust evaluation methodology can be just as important as the results themselves. In this study, we follow the Cloud Evaluation

Experiment Methodology (CEEM) . CEEM is a well established framework, organized in a series of 10 steps that helps to conduct performance evaluations, targeted explicitly at evaluating Cloud services. By using this methodology, we aim to thoroughly describe the design of our experiments and the associated results. This facilitates the repetition of future performance evaluations, adjusting minor details or indicators where necessary. Also, it should be noted that Cloud environments are highly customisable and there is not always a direct match for a specific configuration or feature across different environments. As such, considerable effort has been made to build a homogeneous environment for each Cloud used in this performance evaluation

**3.1. Requirement Recognition**

Given the diversity of managed Kubernetes services on different Clouds, evaluating the performance of such offerings is crucial for service customers, e.g., for cost-benefit analysis, as well as for providers, e.g., for future improvements . Performance, in this context, is regarded as how well we can expect a containerised application to perform while running on a hosted Cloud service.

**3.2. Service Feature Identification**

Based on our requirement definition, we consider the following system resources as the main performance variables to measure and compare: CPU, memory, disk, and network. From a performance perspective, evaluating these resources inside a container, allows us to estimate how well a containerised application performs in such an environment.

**3.3. Metrics/Benchmarks Listing and Selection**

In order to measure the performance of the relevant features identified previously, several open-source benchmark tools are selected. Each of these tools stresses the system resources differently and provides insights into how each container performs. Moreover, benchmarking these system components from different perspectives help us validate the obtained results and improves the reliability of the performance evaluation. I.

**1) Computing Performance***:* In order to benchmark the processor capabilities of each environment, the first tool used in our experiments is Sysbench . This tool comprises several modules that support a thorough evaluation of various operating system resources. We use it to measure the CPU and disk performance of the evaluated systems. The main purpose of Sysbench is to benchmark a system by producing highly intensive operations, typically performed on database management systems, without the necessity of installing and configuring them individually. It is cross-platform and multithreaded. In the case of CPU, the tool stresses the system components by calculating prime numbers up to a value specifiedby the --cpu-max-primes parameter . In our tests, two threads and a limit of 300 seconds were defined, as well as a threshold of 30,000 prime numbers for the computation. The performance comparison was made by monitoring the total number of events calculated under these constraints.

Another CPU benchmarking tool utilized in this study is Y-cruncher . Y-cruncher is a common benchmarking and stress-testing application that can compute various mathematical constants, including Pi to trillions of decimal places.This benchmark consists of a multi-threaded program that is scalable to multi-core systems. We use this tool to stress the CPU and measure the time it takes to compute a value

of Pi to 500 million decimal points across the different Cloud environments. Along with the computation time, the program outputs a Multi-core Efficiency percentage, and the total execution time of the test. This includes the time it took for the computation itself and its verification. Due to the memory-intensive nature of computing some of these mathematical constants, especially Pi, Y-cruncher relies on memory to perform well.

**2) Memory Performance**: In order to test the performance of system memory, we use STREAM . STREAM is a simple synthetic benchmark program that measures sustainable memory throughput by performing four different types of vector operations: Copy, Scale, Add, and Triad. These are described in Table II. Evaluating memory throughput on a container is critical since it determines how quickly data can be read from or written to memory by the processor. Provided

that the memory bandwidth is high, the data required by the processor can be easily retrieved or written. We use this tool to evaluate the data throughput capacity from main memory. As such, performance is determined here by bandwidth not latency. The array size for our tests was defined with 100 million elements. The general rule for STREAM identifies that each array must be at least 4x the size of all the last-level caches used or 1 million elements.

**3) Disk Performanc**e: Another feature that was identified in this study as a critical performance component is disk. To assess disk performance of containerised workloads, we utilize Sysbench and Bonnie++. We use these tools to perform a combination of disk intensive operations such as reads and writes, both sequentially and randomly. In this context, Sysbench is used with the --file-test-mode parameter set to rndrw. This allows us to select the type of workload that we intend to produce, i.e., random read/write operations. In order to force these I/O operations to occur on disk and bypass the caching performed by the system main memory, a set of files with a total size larger than the amount

of RAM available is necessary. Considering that the amount of memory assigned to the worker nodes in the experiments was 8 Gb, a 16 Gb dataset was used for this purpose. Sysbench then operates by reading and copying to and from these files, and calculating the corresponding data throughput over 300 second intervals.

We use these metrics to assess the disk performance of each Cloud environment.

We use Bonnie++ [41] to further evaluate disk performance using a similar approach as in the Sysbench test. Bonnie++ is an open source suite that implements several straightforward tests to evaluate disk performance. In this test, the read and write operations on disk are executed sequentially, and the disk I/O performance is subsequently monitored and compared with specific focus on the Block Input (sequential read) and Block Output (sequential write). As previously, a dataset with a total size that is twice the size of the system memory needs to be

configured for the execution of this benchmark to avoid RAM caching mechanisms influencing the results.

**4) Network Performance**: To evaluate the network performance of the containerised workloads, we use two benchmark tools: nuttcp [42] and netperf [43]. Nuttcp operates under a client-server model. It has many applications, but it is mostly used to stress the network capabilities of a system by transferring memory buffers from a client to a server over a period of time or until a desired number of bytes is reached. This allows determination of the average network

bandwidth available between these two systems under those constraints. In our test, we use it to estimate what the average throughput that any given containerised application could expect to achieve in a given Kubernetes-based environment. A fixed 30 second limit is defined to measure both inbound

and outbound throughput. Both the nuttcp server and client are deployed in the same Kubernetes cluster under evaluation, but each container runs on a different worker node.

Another network evaluation metric that we consider relevant in this study is the communication latency, specifically the TCP and UDP transfer delay. We use the netperf request response benchmark to evaluate the round-trip network latency

of the systems under evaluation. Netperf also incorporates other predetermined tests as part of its suite to assess the network throughput between two systems thoroughly. Since it also works in a client-server model, we undertake a similar

approach from the previous test. To perform a test, a netperf server (netserver) is deployed on a worker node, and a netperf client deployed on another worker node within the same Kubernetes cluster. The netperf client then sends a request

to the netserver and waits for its response before proceeding to the next request. Therefore, only one packet is in transit at any given time. Netperf then evaluates the transaction latency. In this study, both TCP and UDP are considered (TCP RR

and UDP RR tests, respectively), and the 50th percentile, i.e., the median, is taken as a performance indicator of the various

environments that are evaluated. A fixed packet length of 200 bytes was set both for the request and the response, and for each iteration of the experiment, a 30 second limit was configured

**3.4. Experimental Factors Listing and Selection**

In this section, we assess the different factors that may have an impact on the results of the experiments. Proper identification of these factors is necessary to ensure that the evaluation is traceable and repeatable. The experimental factor framework of the Cloud services evaluation from [44] is adopted as a basis.

**1) Workload Factors:**

• **Resource Type:** All the nodes that comprise the various Kubernetes clusters are based on VM instances.

• **Number of Instances:** Each Kubernetes cluster is comprised of two VM instances used as worker nodes.

• **Geographical location:** Are described in Table I and Table II.

• **Number of Executions:** Each test is executed 20 times, and the mean calculated along with the standard deviation. To achieve more accurate results related to the performance of each environment, these 20 executions were distributed across both worker nodes in the cluster

under evaluation.

• **Activity Sequence:** This was sequential.

• **Workload Size:** The workloads were described in detail in the previous step.

**2) Computing Resources Factors**:

• **VM Instance Type and Size:** Most of the computing resources of a VM instance, e.g., vCPU, memory, storage, and network, are tightly coupled with the Type and Size of the chosen instance. Since there is not always a direct match for every resource type across the different clouds, effort has been made to build homogeneous environments in each of them. A general purpose VM type was sought in every case to ensure impartiality of the tests. Table I and Table II describe the instance types and sizes that were ultimately utilized in each Cloud environment.

• **VM Instance Operating System:** the manual Kubernetes deployments were built based on CentOS (x86 64). On the other hand, the worker nodes that used as part of the managed Kubernetes clusters, utilize the default OS recommended by each Cloud Service Provider. See Table I and Table II for a more detailed description.

• **CPU Index:** in every scenario, two vCPUs were configured for the worker nodes. A detailed description of the CPU characteristics is provided in Table Iand Table II.

• **Memory Size:** The VM instance sizes that were chosen all had 8 GB of RAM. Additionally, in the infrastructure for the baseline, the swap was disabled as recommended by Kubernetes [45].

• **Storage Size, Type and Geographical location:** In all worker node instances, sufficient storage was attached to meet the requirements of the different benchmarks. Particularly, when benchmarking disk I/O, both Bonnie++ and Sysbench require at least 16 GB of free space to provide more accurate results. The disk-related tests were performed on SSD disks (Table I and Table II indicate the types of disks that were used). The volumes were deployed in the same geographical location as their associated VM instances.

*•* **Communication Scope:** All the communication-related benchmarks were performed inside a cluster in the same availability zone, i.e., intra-cloud. This was done to avoid any external factors influencing the metrics. Consequently, the results reflect the maximum throughput achieved with the selected resources, i.e., the type and size of instance etc.

**Communication Ethernet Index:** Table III and Table IV show the network performance for the VM instances utilized in the experiments for each Cloud provider.

**3.5. Experimental Design**

All tests follow the same procedure. Each selected benchmark was wrapped up as a microservice into a Docker image and uploaded to a public registry. It is worth noting that all of the benchmark tools use the latest CentOS image available as a base image. The installation and configuration of all the Kubernetes clusters for the baseline was performed according to the documentation from the official Kubernetes site [45]. Once the Kubernetes environments, both hosted and self-managed ones were established and taking into account the experimental factors from the previous subsection, a deployment definition vfile (.yaml) was created for each benchmark. Single-container pods and two replicas were defined for each deployment object so that the benchmark tools could collect performance information from two different worker nodes. As stated previously, the experiments were performed in an isolated and sequential manner, and no resource limit was configured at any point to avoid resource contention when performing the tests.

Lastly, in order to automate the execution of the tests (20 times for each test to validate the results), several scripts were configured following each tool documentation. By following these steps, we were able to stress the different system components under evaluation from within the containers and monitor their performance. The results allowed us to contrast the performance of containers running in each managed Kuber netes environment and understand which one performed better and under which scenario. Moreover, by comparing the results against manually deployed clusters, we could determine if any overheads were introduced by these Cloud services.

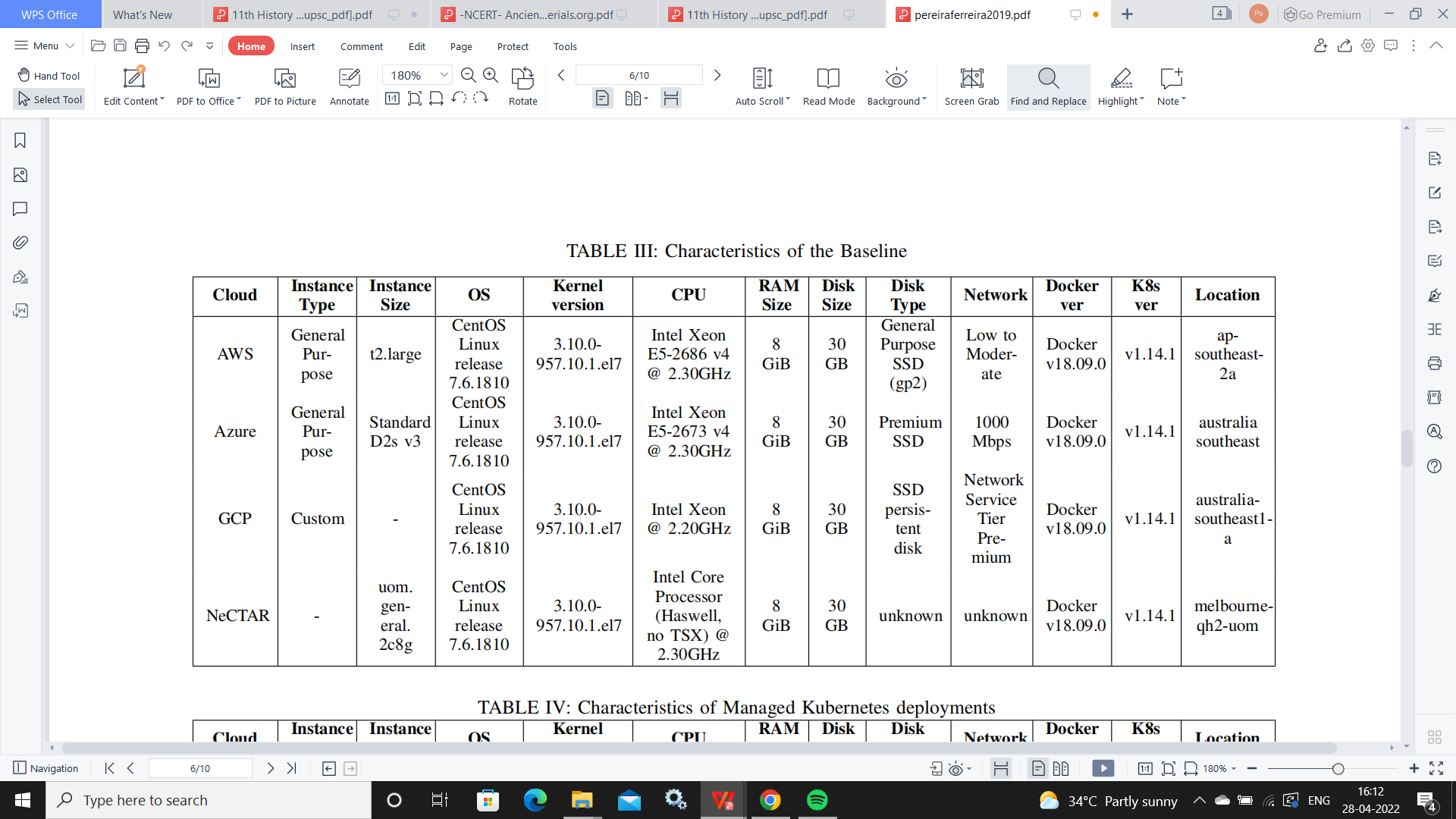


Table 3.1 Characteristics of Baseline

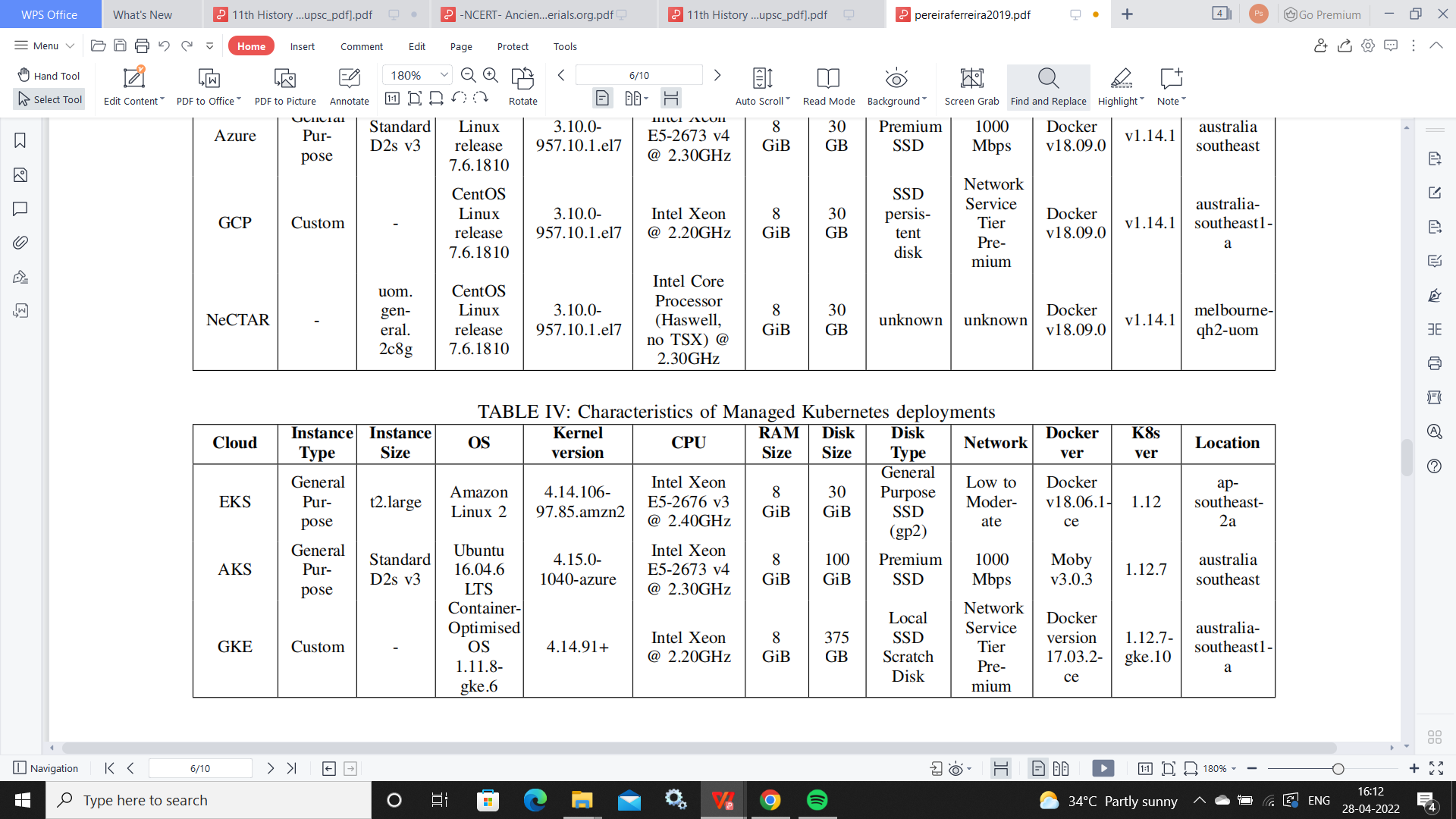


Table 3.2characteristics of managed kubernetes deployment

**CHAPTER 4: RESULTS**

In this section, the results of our evaluations are presented, and analyses of the primary outcomes are provided. For each experiment, the mean and the standard deviation was computed and drawn in the corresponding graphs. As stated previously, the performance of containers running on the NeCTAR Research Cloud was used as the baseline.

**4.1. Computing Performance**

The first results that we explore correspond to Sysbench . Since this tool stresses the system by computing prime numbers, a higher number of events is thus important. As shown in Figure 2, it can be seen that amongst the man aged Kubernetes services, the AWS offering, herein EKS, outperforms both Google GKE by 17% and Azure AKS by almost 30%. Although there are slight variations between the average performance of the hosted services and the manually configured Kubernetes services. It is also noticeable that the standard deviations and distribution of results were quite spread. This was mostly due to the execution of the tests across various worker nodes. In some scenarios, the performance difference ranged between 8% to 14% when evaluating the performance of the same cluster but using different containers running on different worker nodes. Therefore, the difference between the results of the two types of environments, i.e., hosted vs manually configured, was not significant.

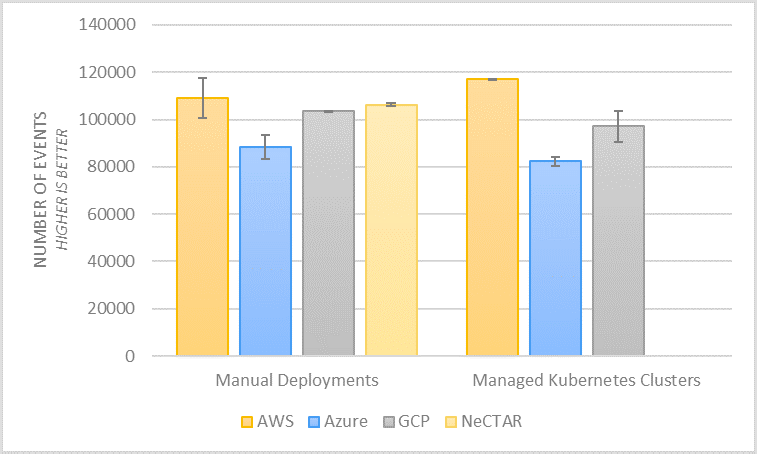


Fig 4.1. Computing Performance Results using Sysbench

The second CPU benchmarking tool used to analyze performance was Y-cruncher [39]. In this case, since we compute the time it takes the system to calculate a Pi to 500 million decimal places, a smaller result is sought. The comparison shown in Figure 3 is based on the total time, which refers to the total computation time plus the time dedicated to verification. Here again, it can be seen that the best performance in both scenarios was achieved by AWS, followed closely by NeCTAR (with the manually configured Kubernetes

cluster). AWS EKS outperforms Google GKE by 59% and Microsoft AKS by 117%. Additionally, by comparing all of the managed Kubernetes environments against their respective manually configured Kubernetes clusters, we can see that the performance is almost the same. The multi-core efficiency results indicate how effective the CPU is utilized for the computation. We can see that the improved performance of AWS with regard to CPU, is reflected in this metric, as shown in Table V. It is worth noting that in both tests, EKS had a small standard deviation. This indicates a consistent performance throughout the 20 times it was tested. An interesting insight from this is that both environments that received two cores (AWS and NeCTAR) outperformed the other two environments that only received one core with two threads (Azure and GCP). In all scenarios, we only requested two vCPU when provisioning the worker nodes.

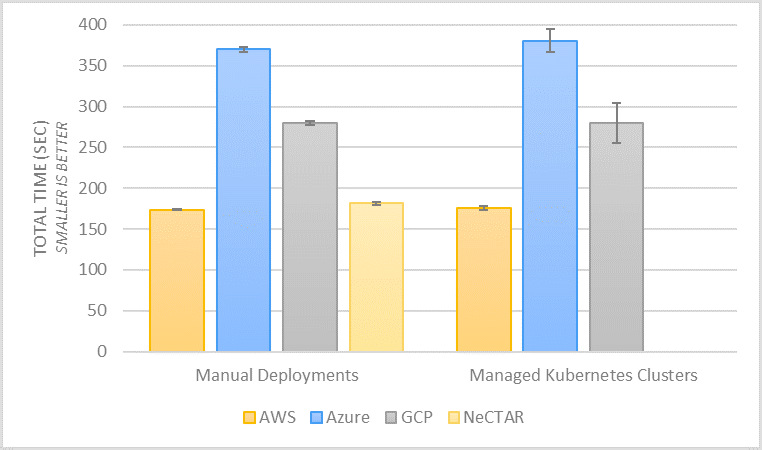


Fig 4.2: Computing Performance Results using Y-cruncher

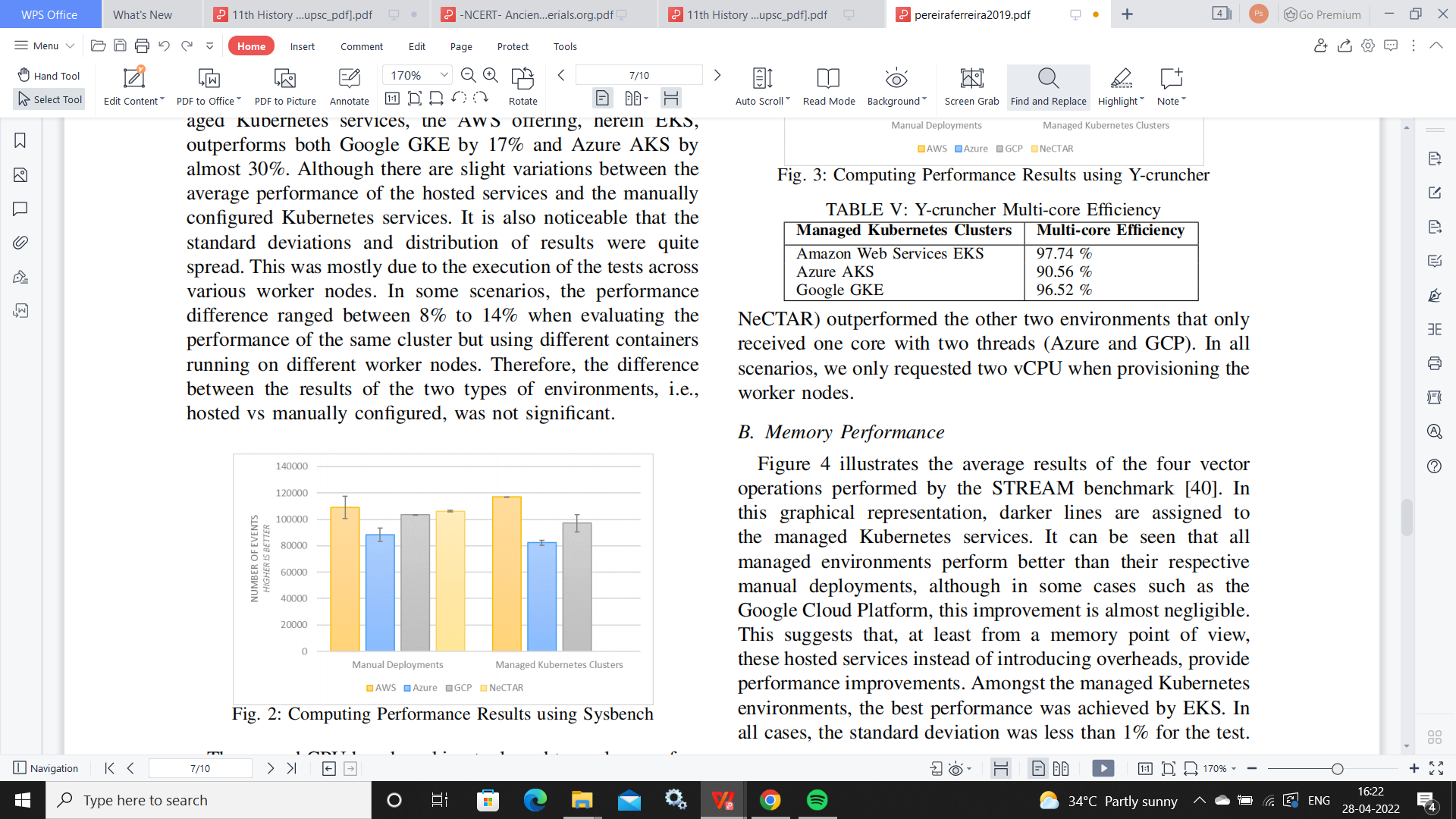


Table 4.1 Y-cruncher multi core efficiency

**4.2. Memory Performance**

Figure 4 illustrates the average results of the four vector operations performed by the STREAM benchmark [40]. In this graphical representation, darker lines are assigned to

the managed Kubernetes services. It can be seen that all managed environments perform better than their respective manual deployments, although in some cases such as the

Google Cloud Platform, this improvement is almost negligible. This suggests that, at least from a memory point of view, these hosted services instead of introducing overheads, provide performance improvements. Amongst the managed Kubernetes environments, the best performance was achieved by EKS. In all cases, the standard deviation was less than 1% for the test.

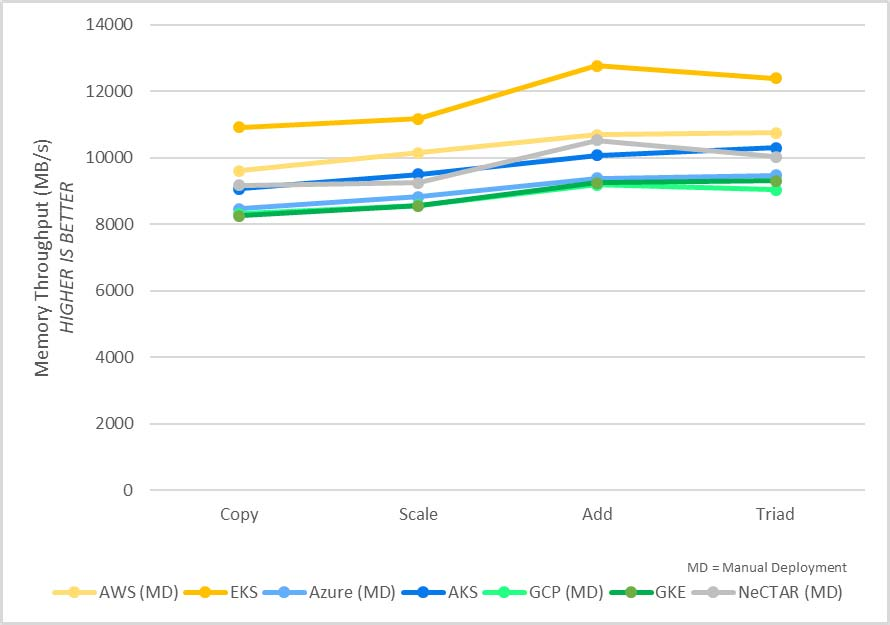


Fig 4.3 Memory Performance Results using STREAM

**4.3. Disk Performance**

In this subsection, we analyse the performance of the different Kubernetes-based environments from a disk perspective. For this purpose, we measure both sequential and random I/O. First, we examine the results from Bonnie++ [41] as shown in Figure 5a and Figure 5b. The single most important conclusion we can draw from these results is the huge performance improvement implemented by GKE with regard to its baseline. It represents an improvement of approximately 850% for read operations and 1244% for write operations. After investigating

this effect, we found out that although both Google-based environments utilise SSD volumes, each environment was created with a different type of SSD, as described in Table III and Table IV. In GCP, the worker nodes that are part of the manually confifigured cluster utilise ”SSD persistent disks” while the Managed Kubernetes cluster makes use of ”Local SSD”. This is a clear example that indicates that most of the performance flfluctuations are tightly coupled with the underlying resources selected (e.g., VM Instance Size, Storage Type, etc.), and that the mere utilisation of managed cloud services does not necessarily imply the existence of performance overheads associated with them. A detailed explanation of the types of SSD disks in the Google Cloud Platform can be

found in [46]. The other platforms exhibit similar performance compared to their respective baselines. We note that NeCTAR is not considered in this analysis since we were not able to attach SSD disks on the VM instances.

We also benchmark the effificiency of random I/O operations using Sysbench [37]. One important observation to note in this case is that the scale is signifificantly smaller since accessing the data randomly is not as fast as accessing it sequentially. Figure 6a and Figure 6b show the outcome for the baseline as well as the different managed Kubernetes services that were evaluated. We see that the average performance obtained in the hosted Kubernetes services is slightly better than our baseline. The only exception in this case GKE. Interestingly, although GKE utilises a ”Local SSD” which performs better sequentially, it does not show the same improvement when the data is accessed in a random fashion. Hence the other type of SSD available in Google-based environments (SSD persistent disks), perform better for this type of workload. Overall, the results show that AWS always performs better than the rest, and the worst performance was Azure

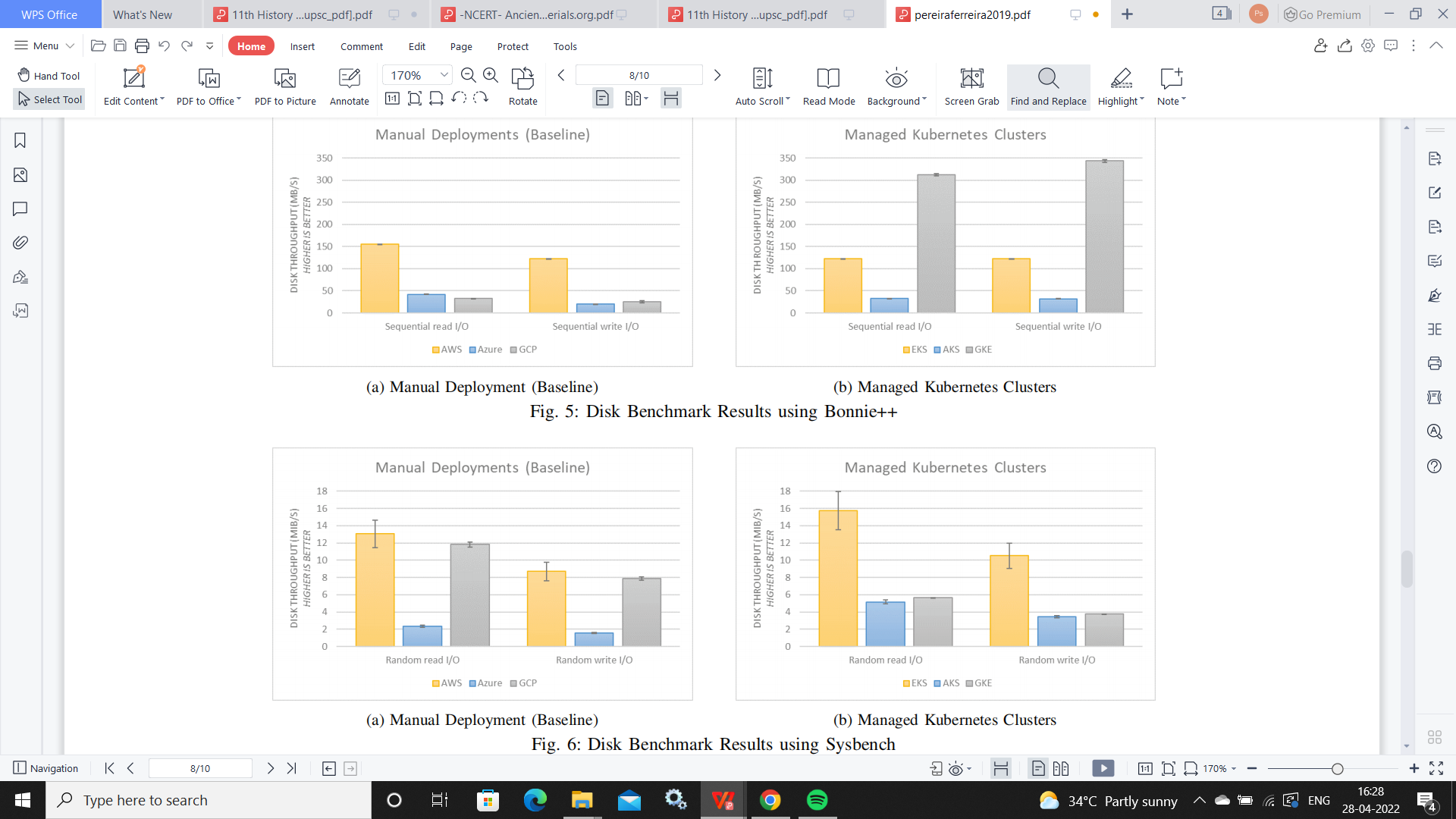


Fig 4.4 Disc benchmark result

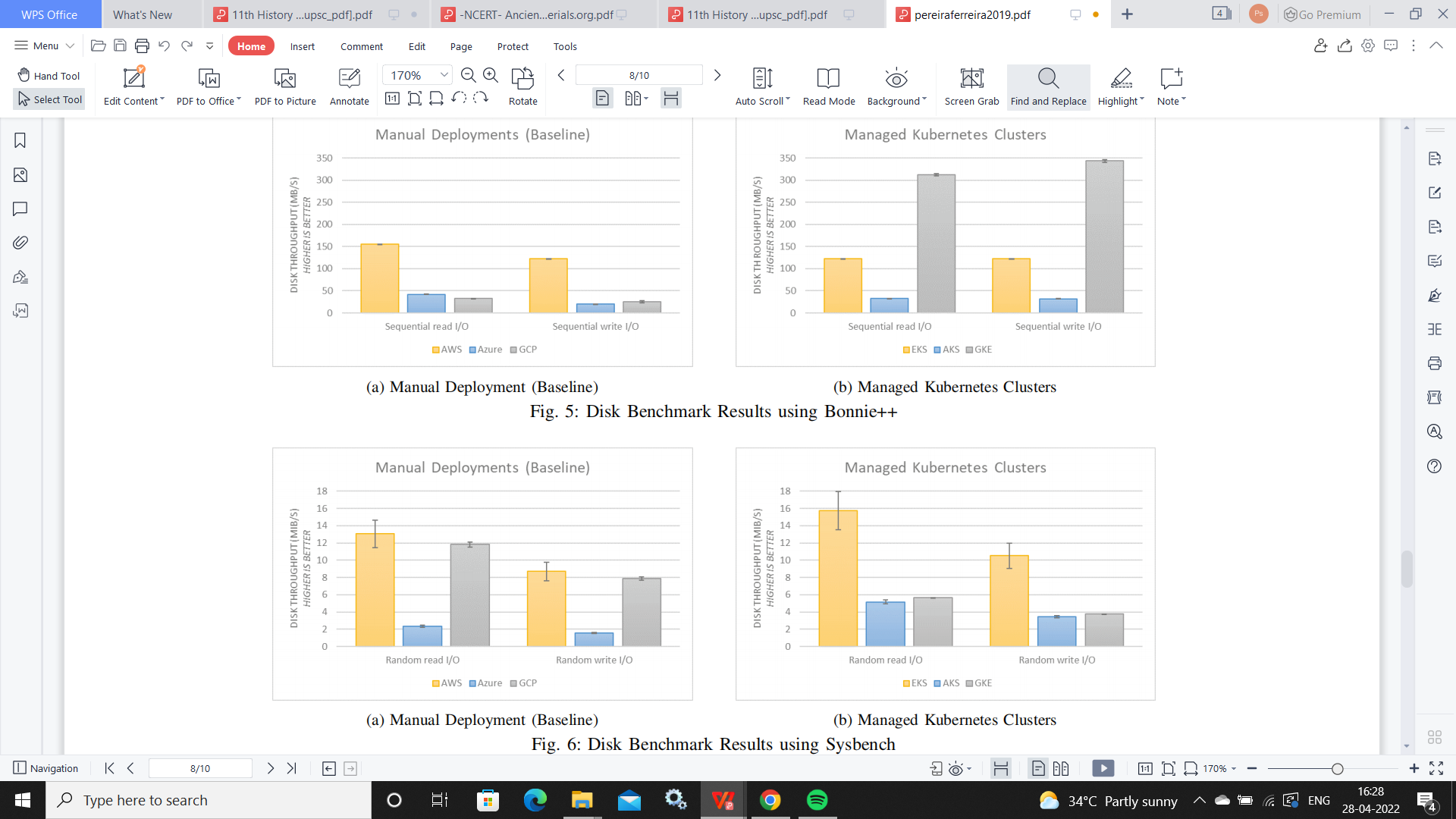


Fig 4.5Disc benchmark result using sysbech

**4.4. Network Performance**

Network performance was evaluated by measuring the communication throughput and latency present in each environment. First, we examine the results of network throughput provided by nuttcp [42]. It is worth noting that we deployed the same infrastructure for all network-related tests. This consisted of two containers; the first one acting as a server in one worker node, and the other acting as a client in another worker node within the same Kubernetes cluster. Therefore with this model the results correspond to the best performance that can be expected in Clouds adopting these resources (as detailed in Table III and Table IV). It can be seen in Figure 7 that all the managed Kubernetes alternatives outperform the containers running in the manually installed and configured

Kubernetes clusters, although the difference can be quite small with AWS and Azure (less than 1% in AWS and 14% in Azure). Here again, a considerable improvement exists in the Google alternative, GKE reports an 82% improvement over its baseline. There are no network configuration differences between those Google-based environments. Reverse TCP was also explored and shown in every scenario to be similar to TCP (*±*0.05).

Lastly, we examine the communication latency with the netperf benchmark tool [43]. From Figure 8a and Figure 8b we identify noteworthy situations. First, it is highly noticeable that the containers running on the AWS managed Kubernetes cluster perform worse than the containers running in the manually deployed cluster on AWS. From a latency perspective, there is approximately 257% in overheads introduced for TCP and 265% for UDP. It is also noted that both environments were built using the same underlying resources, in this case, t2.large

EC2 instances. However, the documentation only mentions that the network performance we can expect to achieve with these types of instances is ”Low to Moderate” [47]. Another observation that can be made from these outcomes is that Google-based environments achieve the best performance, followed closely by NeCTAR (based on the manual deployments). Notably, amongst the managed Kubernetes alternatives, our experiments show that GKE outperforms AKS by 600% and EKS by 1060%. AKS was similar to its baseline.

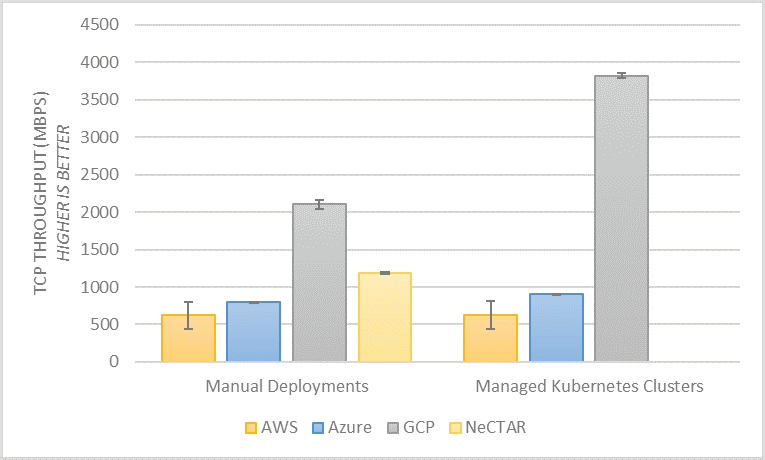


fig 4.6Network Throughput Performance Results using Nuttcp

**CHAPTER 5: CONCLUSION AND FUTURE WORK**

In this presentation , we evaluated the performance of containers running on various managed Kubernetes environments across different mainstream Cloud environments including Amazon Elastic Container Service for Kubernetes (EKS), Azure Kubernetes Service (AKS), and Google Kubernetes Engine (GKE). Since an increasing number of Cloud-based solutions have emerged recently, our goal was to investigate if significant overheads were introduced when using these services.

Our experimental evaluations indicate that most of the performance fluctuations obtained were tightly coupled with the underlying resources selected (e.g., VM Instance Type/Size,

Storage Type, Network Tier), and that the mere utilization of managed Cloud services does not necessarily imply the existence of a performance overhead associated with the

managed Cloud service itself. The baseline was key to help understand that some of the poor performances obtained when evaluating the managed Kubernetes services, were also poor for the manually confifigured Kubernetes clusters. Moreover, the results show that there was no single optimal solution for every scenario. For CPU-intensive containerised applications, either AWS or NeCTAR provided the best choice. For Network intensive applications, undoubtedly GKE (with the Network Service Tier Premium) had the best network performance, even enhancing the manually deployed cluster in its own Cloud. One of the most important aspects to notice was that the selection of adequate underlying Cloud resources was key to achieving a good performance.

There exist many other experiments that could be made with different VM Instances Types/Sizes, Volumes and other resources. The evaluation logic of our experiments was fully documented and could be adapted to different scenarios directly, especially since it follows the Cloud Evaluation Experiment Methodology (CEEM) to support traceable and reproducible experimental evaluation results. Exploring the performance impact and costs associated with scaling of Kubernetes clusters in Cloud environments would be something further to address in future work, e.g., scaling to hundreds or thousands of containers and measuring pod start up time, or scaling across different availability zones/regions, or even across different Cloud providers for inter-Cloud scenarios.

CHAPTER 6: REFERENCES

[1] G. Research, *Hype cycle for cloud computing, 2021,* https:// www.gartner.com/en/documents/3884671, [Online; accessed 11-June-20201, 2021.

[2] R. Hat, *Red hat global customer tech outlook 2020: Automation, cloud, security lead funding priorities*, https://www.redhat.com/en/blog/red/hat- global- customer-tech- outlook- 2020- automation- cloud- security lead-funding-priorities?source=bloglisting, [Online; accessed 11-June-2020], Dec. 2020.

[3] Diamanti, *2020 container adoption benchmark survey*, https://diamanti.com/wp- content/uploads/2020/07/WP Diamanti End- User Survey 072818.pdf, [Online; accessed 11-June-2019], 2020.

[4] Portworx, *2019 container adoption survey*, https://portworx.com/wpcontent / uploads / 2019 / 12 / Portworx - Container - Adoption - Survey -Report-2019.pdf, [Online; accessed 11-June-2019], 2019.

[5] S. Soltesz, H. P¨otzl, M. E. Fiuczynski, A. Bavier, and L. Peterson,“Container-based operating system virtualization: A scalable, high performance alternative to hypervisors,” in *ACM SIGOPS Operating Systems Review*, ACM, vol. 41, 2007, pp. 275–287.

[6] R. Morabito, J. Kjallman, and M. Komu, “Hypervisors vs. lightweight virtualization: A performance comparison,” in *2019 IEEE International Conference on Cloud Engineering*, IEEE, 2015, pp. 386–393.

[7] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in *2020 IEEE international symposium on performance analysis of systems and software (ISPASS)*, IEEE, 2020, pp. 171–172.

[8] R. Dua, A. R. Raja, and D. Kakadia, “Virtualization vs containerization to support paas,” in *2018IEEE International Conference on Cloud Engineering*, IEEE, 2018, pp. 610–614.

[9] A. M. Joy, “Performance comparison between linux containers and virtual machines,” in *2015 International Conference on Advances in Computer Engineering and Applications*, IEEE, 2015, pp. 342–346.

[10] Kubernetes, *Production-grade container orchestration*, https : / /

kubernetes.io/, [Online; accessed 11-June-2019].