High-Performance Mining of COVID-19 Open Research Datasets for Text Classification and Insights in Cloud Computing Environments

Jie Zhao, Maria A. Rodriguez and Rajkumar Buyya
Cloud Computing and Distributed Systems Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia
Email: zhao.j4@student.unimelb.edu.au, {maria.read, rbuyya}@unimelb.edu.au

***Abstract*—The COVID-19 global pandemic is an unprecedented health crisis. Many researchers around the world have produced an extensive collection of literature since the outbreak. Analysing this information to extract knowledge and provide meaningful insights in a timely manner requires a considerable amount of computational power. Cloud platforms are designed to provide this computational power in an on-demand and elastic manner. Specifically, hybrid clouds, composed of private and public data centers, are particularly well suited to deploy computationally intensive workloads in a cost-efficient, yet scalable manner. In this paper, we developed a system utilising the Aneka Platform as a Service middleware with parallel processing and multi-cloud capability to accelerate the data process pipeline and article categorising process using machine learning on a hybrid cloud. The results are then persisted for further referencing, searching and visualising. The performance evaluation shows that the system can help with reducing processing time and achieving linear scalability. Beyond COVID-19, the application might be used directly in broader scholarly article indexing and analysing.**

I. Introduction

COVID-19 has given place to a global scale health crisis. Since the outbreak, a massive amount of research efforts have been poured into many aspects of this highly infectious disease. To help the research community, in March 2020, the White House and the Allen Institute for AI teamed up with many researchers and released the COVID-19 Open Research Dataset (CORD-19) [1]. As of July 2020, CORD-19 contained over 199,000 research papers, with nearly half of them being open access publications [2]. The rapid increase on the number of articles has posed a challenge for the research community to process these data in a timely manner. Furthermore, the general public is also interested in many aspects of the disease, especially on findings related to day-to-day life. As a result, tools and platforms that support machine learning approaches to extract knowledge from this vast amount of data are required. This is a challenging endeavor as a considerable amount of computing power is needed by Extract, Transform, Load (ETL) and ML techniques in order to produce results in a reasonable amount of time.

Cloud computing is the de facto standard to access computing resources on demand and on a pay-as-you-go manner. Hybrid Cloud Environments (HCEs) combine private data centers with resources offered by public cloud providers; thus enabling applications to save cost by using existing on-premise resources and to scale onto public clouds if the private data center's capacity is not sufficient [3]. This approach can be greatly beneficial to organizations seeking to process the CORD-19 dataset. However, building an application using HCE is also a demanding task as it requires detailed knowledge of cloud computing techniques.

In this context, we propose a system design and implementation to accelerate ETL processing and text classification based on ML techniques in an HCE. The architecture is designed with the following requirements in mind: scalability, availability, stability, high performance and portability. To achieve these goals and for ease of development, the proposed application deploys on top of Aneka [4], which is a resource management framework that provides high level Application Programming Interfaces (APIs) and Software Development Kits (SDKs) to transparently enable the deployment of applications on cloud resources. It allows developers to focus on implementing their program logic without spending too much time considering deployment and scalability. When additional resources are required, they can be seamlessly acquired from different CSPs via Aneka dynamic provisioning mechanism.

The following are the key contributions of this paper:

- We present a system architecture design that fulfils different requirements: scalability, availability, stability, high performance and portability.
- The system is implemented and tested using real world CORD-19 dataset in a real hybrid cloud environment, built using on-premise private cloud and the Melbourne Research Cloud (MRC).
- The architectural design can be easily generalized and quickly adopted in similar scenarios. The system is not only applicable for the CORD-19 dataset but also for a broader scholarly article indexing and analysing.

The rest of this paper is organized as follows: Section II presents background information about Aneka PaaS and other technologies used in the system. The system architecture is

978-0-7381-2394-3/20/$31.00 ©2020 IEEE
DOI 10.1109/UCC48980.2020.00048

detailed and discussed in Section III. Section IV explains the system design and implementation by using UML diagrams and workflows, in addition, some example queries and visualizations are given. Afterwards, section V describes the testbed built on a hybrid cloud including an on-premise private cloud and MRC, followed by the performance evaluation. Finally, section VI summarizes our work and findings and outlines proposed future work.

II. Background and Related Work

Since the CORD-19 dataset was made available, it has been downloaded over 200,000 times and many applications have been created from it [2]. For instance, Amazon Web Service (AWS) provides a search engine over the CORD-19 dataset and a question answering system powered by AWS Kendra [6]. Azure [7], TekStack [8], and COVID-Miner [9] developed full-text search engines. VIDAR-19, which extracts and visualizes risk factor from articles, was presented by Wolinski et al. [10]. COVID Seer [11] and COVID Explorer [12] were developed by the Pennsylvania State University. COVID Seer is a multi-facet search engine powered by Elasticsearch and COVID Explorer has visualization and advanced filtering features utilizing automatic unsupervised ML.

One drawback with existing systems is that they are not keeping up-to-date with the growing dataset. Some of these are still using the initial version of CORD-19, even though the current dataset has grown threefold since the initial release. Hence, we address this issue by introducing additional workflows to keep up-to-date with current version of CORD-19 dataset and ingest newly published articles. These workflows allow users to have the latest view of state-of-the-art COVID-19 research outputs.

To implement the system quickly, the best solution is to select some proven technologies in both industry and academia. After evaluating many different technologies, we decide to go with the following: (a) Aneka PaaS [4] for core processing and deployment; (b) Microsoft .NET Framework and ML.net [13] for development; (c) Grobid [14] for ML-based scientific paper parsing; (d) Minio [15] for scalable S3 compatible shared storage; (e) Elastic Stack [16] including Elasticsearch for full-text indexing and Kibana for data visualization.

Aneka PaaS provides a platform for users and developers to develop and deploy distributed applications. An overview of Aneka is shown in Fig 1. It comprises of three major layers with a rich collection of components:

- Application Development and Management Layer: This layer contains the Software Development Kit (SDK) and the management kit. SDK is a collection of Application Programming Interfaces (APIs), tutorials and examples to help users and developers get started. Management kit includes a Graphical User Interface (GUI) to assist with management. It comprises of a management studio, admin portal and web services. Within the management studio, there is also an accounting view, allowing users to view various states and statistics of application executions.
- Middleware Container Layer: This layer provides many services, such as execution services, foundation services, and fabric services, among others. The execution service is the most crucial part for our application. It performs scheduling and execution based on the user's choice of QoS and strategy and supports four execution models: (1) Bag of Tasks (BoTs); (2) Distributed Threads (DTs); (3) MapReduce; and (4) Parameter Sweep Model (PSM). In our application, we want the processing to be completed as soon as possible. Therefore, we can use either BoTs or DTs model; we choose the default QoS strategy that uses all available computing resources.
- Infrastructure Layer: The bottom layer offers fundamental support to the above layers. In the current version, Aneka allows to use resources with static or dynamic provisioning. For static provisioning, user can install Aneka on fixed number of physical machines or VMs in desktop cloud, data centre, cluster, and let Aneka manages them; For dynamic provisioning, Aneka can add VMs on-the-fly when needed in public clouds, such as Amazon EC2, Microsoft Azure, GoGrid, etc.

Microsoft .NET Framework/.NET Core is a free software development framework developed by Microsoft. It is a complete platform that supports various languages, such as C#, VB, F#, etc. It provides cross-platform support including Windows, Linux and MacOS. Since Aneka is developed using .NET, it becomes a natural choice for our application. In addition, Microsoft made an extensive ML framework for .NET developers. It implements many traditional and proven ML algorithms; users who don't have detailed knowledge of ML techniques can still easily use ML technology in their application. This significantly lowers the barrier for developers to utilize ML technology.

Elastic Stack (ES) [16] is widely used in industry; it is also known as ELK Stack (Elasticsearch, Logstash, Kibana). Elasticsearch is a distributed full-text indexing and search engine based on Apache Lucene library; Logstash provides data collection and log-parsing engine through various of agents called Beats; Kibaba is a data visualization platform using searching, filtering and aggregation functions provided by Elasticsearch. In our system, Logstash was replaced by our customized ETL pipeline implemented with Aneka, we only utilize Elasticsearch and Kibana in the ELK stack.

In addition to the technologies above, we also use an open source program called Grobid [14] in our update workflow. Grobid is a ML library designed explicitly for extracting text data from technical or scientific documents. It is capable of converting PDF file to TEI/XML while maintaining section and structure format. It has been actively developed since 2008 and open-sourced in 2011.

III. System Design and Architecture

This section outlines an overview of the system architecture visualized in Fig. 2. The system utilizes a hybrid cloud environment: an on-premise private cloud for storing and

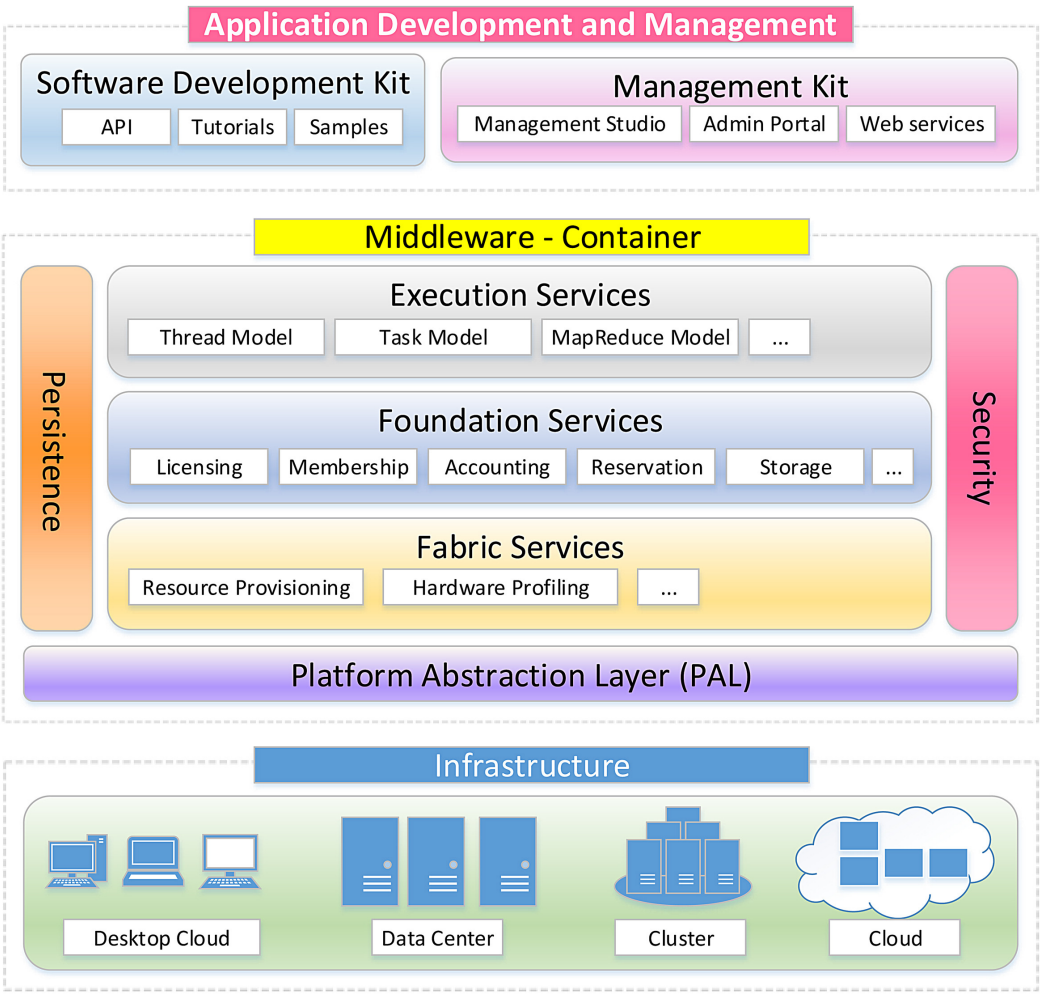

Fig. 1. Aneka Framework Overview [5]

processing data and Melbourne Research Cloud (MRC) for hosting and serving public-facing ES traffic.

While designing a data-centric processing system, the first thing to consider is how to store and distribute the data efficiently. Although Aneka PaaS supports data distribution via task payload and FTP, it is not efficient and scalable in our usecase. Therefore, we decided to use a centralized storage cluster powered by Minio [15], an open-source object storage software. Minio provides S3 compatible API and a shared-nothing architecture for scalability and availability. It is a shared file system that can be accessed by all Aneka workers/master and application server. When required, data can be easily replicated to external CSP over a VPN connection, creating a multi-cloud storage cluster. The second component is the computing service provided by Aneka PaaS as described in Section II. Aneka makes it easy to perform parallel processing by encapsulating task scheduling and execution. The third component is an ES cluster replicated to a MRC public instance over the Internet/VPN. The on-premise primary ES cluster is for data persistence. The secondary instance is configured for read-only accessing and serving public-facing traffic. Due to resource constraint, we are running on a single node at both sides with regular automated snapshot/backup.

The architecture design addresses some common system design principles as follows:

1) Scalability: All major components are horizontally and vertically scalable. When additional capability is required, the application can easily scale up by provisioning more powerful virtual machines, or scale out by adding more nodes.
2) Availability: Minio and Elasticsearch use shared-nothing architecture, which is designed for highly available systems. Aneka also has a robust mechanism to handle task/node failure automatically.
3) Stability: This can be achieved using fail-fast and idempotent processing. If a task fails for any reason, it can be rescheduled either periodically or automatically without affecting the whole system.

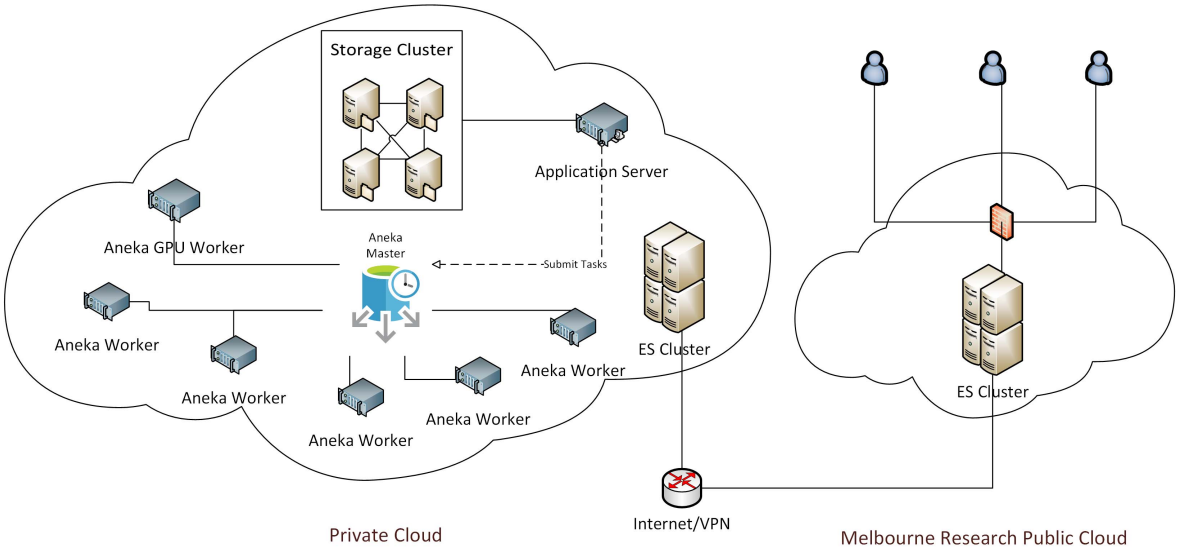

Fig. 2. Architecture Diagram

4) Performance: High performance is achieved by facilitating Aneka's distributed scheduling and execution capability.
5) Portability: An application developed with the Aneka SDK/API is portable. It can be executed in any environment that supports the .Net framework, regardless of the cloud service provider. Portability can be easily achieved by moving the application to another CSPs like AWS, Azure, GCP, etc.

The main system logic is implemented in the application server component. The application server works as a collaborator performing some housekeeping tasks. It periodically checks for dataset updates, downloads PDF files from various sources and submits tasks to the Aneka master. It also monitors the ML model bucket. When a new training set becomes available that can improve the ML model, the application server submits a task to the Aneka master for training. The actual data processing tasks are performed by worker nodes.

IV. System Implementation

This section describes system implementation shown in Fig. 3. The diagram comprises six UML swimlanes showing the whole application workflow, which we categorized to three stages, will be explain from left side.

Firstly, the data update stage is the three swimlanes from the left. The application fetches the latest version from CORD-19 dataset and other sources like PubMed or Semantic Scholar. If new data is available, depending on the available data type, it ingests PDF to the storage bucket labelled as RAW or CORD-19 to the Staging bucket. The RAW bucket stores PDF files that need to be extracted at later stage; and the Staging bucket contains TEI/XML, metadata and JSON files that have been extracted from PDF files or the CORD-19 dataset. Since every article is binary checksummed with SHA1 and all processed articles are stored in a separate bucket, an incremental update is possible by comparing checksum. Because checksum values are used as object keys in S3/Minio, complexity for checking is O(1).

The swimlane labelled as ML is the second stage. The ML Models bucket stores training data, test data and pre-trained ML model. This bucket will be checked periodically by the application. We decided to use a separate bucket for the ML models for following reasons: a) For classification problem, high quality dataset is crucial for achieving higher quality output. If higher quality labelled data becomes available in the near future, it can be put into this bucket manually and the application will train a newer model; b) Object versioning can be enabled for pre-trained ML models. Hence, the application can evaluate all past and current models to decide which one to use for inference base on their quality. Also, if something goes wrong with a newer model, it can always fallback to the previous version. For ease of development, we use SdcaMaximumEntropy multi-class trainer provided by ML.net to train our model and save it to the bucket.

Finally, the main data processing stage is described in Algorithm 1. The UML class diagram in Fig. 4 shows the classes that comprise the application. To define an Aneka task, developers only need to make the class serializable and implement Aneka's task interface. This simplifies the implementation of the application logic with just a single method to program. After processing, the processed files are moved from the staging bucket to the Completed bucket for archiving and referencing purposes. The benefits for the moving are two-fold: 1) it avoids duplicate processing; 2) data can be moved back if required for various reasons, e.g. re-processing with better model.

We carried out two queries with the purpose of demonstrating the use and applicability of our application:

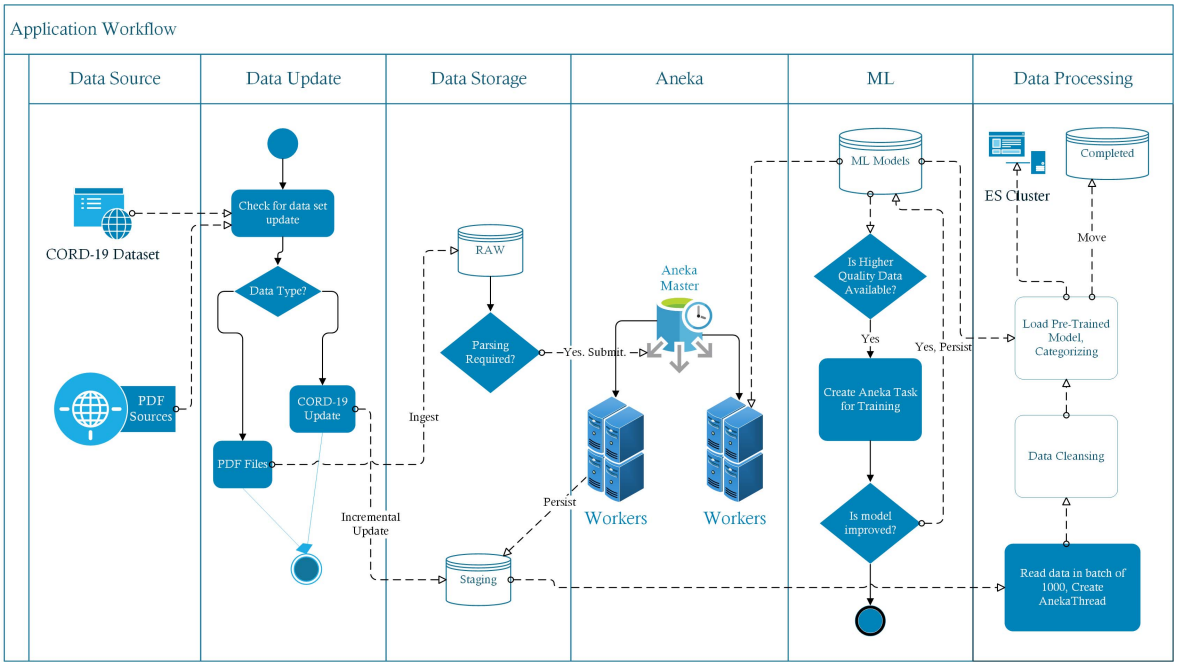

Fig. 3. Application Workflow

Algorithm 1: Data Processing Stage.

Result: Performing ETL and categorizing articles in parallel
Aneka Initialisation;
totalNumberOfFiles ← Count number of files in staging bucket;
foreach *Batch of 1000 files* **do**
  keys[1000] ← file object keys;
  Create an AnekaThread wrapping the keys and process logic;
  Submit to Aneka Master to schedule the processing;
end
|| Worker Nodes in parallel:
model ← Load pre-trained ML model;
foreach *key in keys* **do**
  Read file from storage server;
  Clean unnecessary texts, we only keep metadata, abstract and full-text;
  Predict the category with model and add labels;
  Send data to ES cluster;
  Move the processed file to the completed bucket;
end

- Query 1: What are the hottest research areas?
- Query 2: Which country is contributing the most efforts to these researches? How many articles are contributed by Australia and the University of Melbourne?

For both queries, the data was searched, aggregated and visualized using Elasticsearch and Kibana. The results are shown in Fig. 5 and Fig. 6. These results are based on the data that existed at the time of querying and are intended only for illustration purposes.

V. Performance Evaluation

In this section, we introduce our testing environment, present benchmark results, and also share our observations based on the experimental findings.

A. Testing Environment Setup

Table I lists the specifications of our test environment. The Aneka master runs in a converged mode, which means it also acts as a worker node. In our single node performance benchmark, the master node processed all data on itself; in multi-node benchmark, the workload was sent to workers over network. In our test-bed, the nodes are virtualized instances with Linux KVM running on top of three physical hosts. Each host has 2x1 GBE configured with LACP (802.3ad), layer 2+ hashing and connected to the same gigabit managed switch.

Testing Environment	
Role x Qty	CPU/RAM/OS
Aneka Master x1	Xeon E5-2690v3 2.6GHz(4C)/8G/Windows 10
Aneka Worker x3	Xeon E5-2690v3 2.6GHz(4C)/8G/Windows 10
Minio x1	Xeon E3-1245v5 3.5GHz(2C)/8G/Ubuntu 20.04
ES Private x1	Xeon E3-1245v5 3.5GHz(4C)/16G/Ubuntu 20.04
ES Public x1	MRC vCPU(2C)/8G/Ubuntu 18.04

TABLE I
Specification for Testing Environment

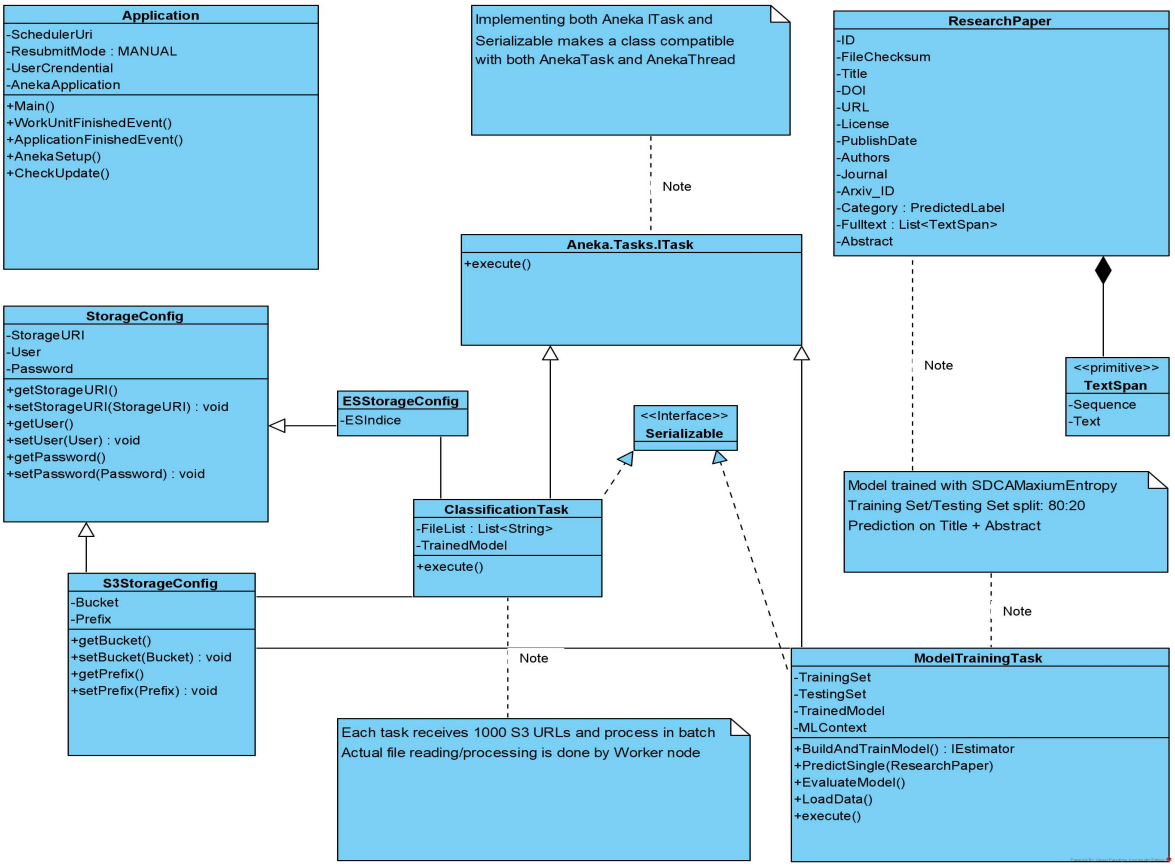

Fig. 4. UML Class Diagram for the Application

B. Benchmark Results and Observations

The experiments were done with smaller sets of articles $N = \{10000, 20000, 30000, 40000, 50000\}$ randomly taken from the CORD-19 dataset, running with $M = \{1, 2, 3, 4\}$ nodes to demonstrate performance and scalability. Fig. 7 and Fig. 8 show data processing time for a single node and multiple nodes.

Since each article in the dataset is independent, for a single node, we assume the execution time increases linearly in relation to the size of input. It is demonstrated in Fig. 7. For multiple nodes, the processing time can be significantly reduced; we are able to achieve near linear scalability with minor overhead. There are few observations worth note here:

1) Processing time of each article is near constant, the fluctuation is relatively small from 18.2ms to 18.7ms. This is largely caused by I/O bound and network latency since each article is read from storage server before processing and persisted to the ES cluster afterwards.
2) Based on single node result, assumedly, using two nodes reduces processing time by half comparing to a single node. However, the actual result is less than 2x since there is communication and task scheduling overhead.
3) When dataset is not large enough, after reaching the diminishing return point, further increasing number of nodes will not reduce processing time much further as theoretically expected due to overheads (e.g., for 10000-20000 articles, the benefit of using more than two nodes is less rewarding, meaning using two nodes is the best in this problem size). For larger input, using more nodes is still beneficial.

VI. Conclusions and Future Work

Extracting knowledge and providing meaningful insights from an extensive collection of literature remains a non-trivial task, especially when time is a constraint under circumstances such as the COVID-19 crisis. Hybrid clouds are well suited for these scenarios because of its scalable yet cost-effective nature. In this paper, we proposed a system architecture for indexing, analysing and extracting insights from scholarly articles. In particular, we conducted our experiment with the CORD-19 dataset. We choose many technologies from academia and open-source community to create an scalable, highly available, stable, high-performance and portable application. By using the Aneka PaaS solution, parallel data processing application can be effortlessly developed. It significantly reduces entry barrier for a developer to develop such a distributed application.

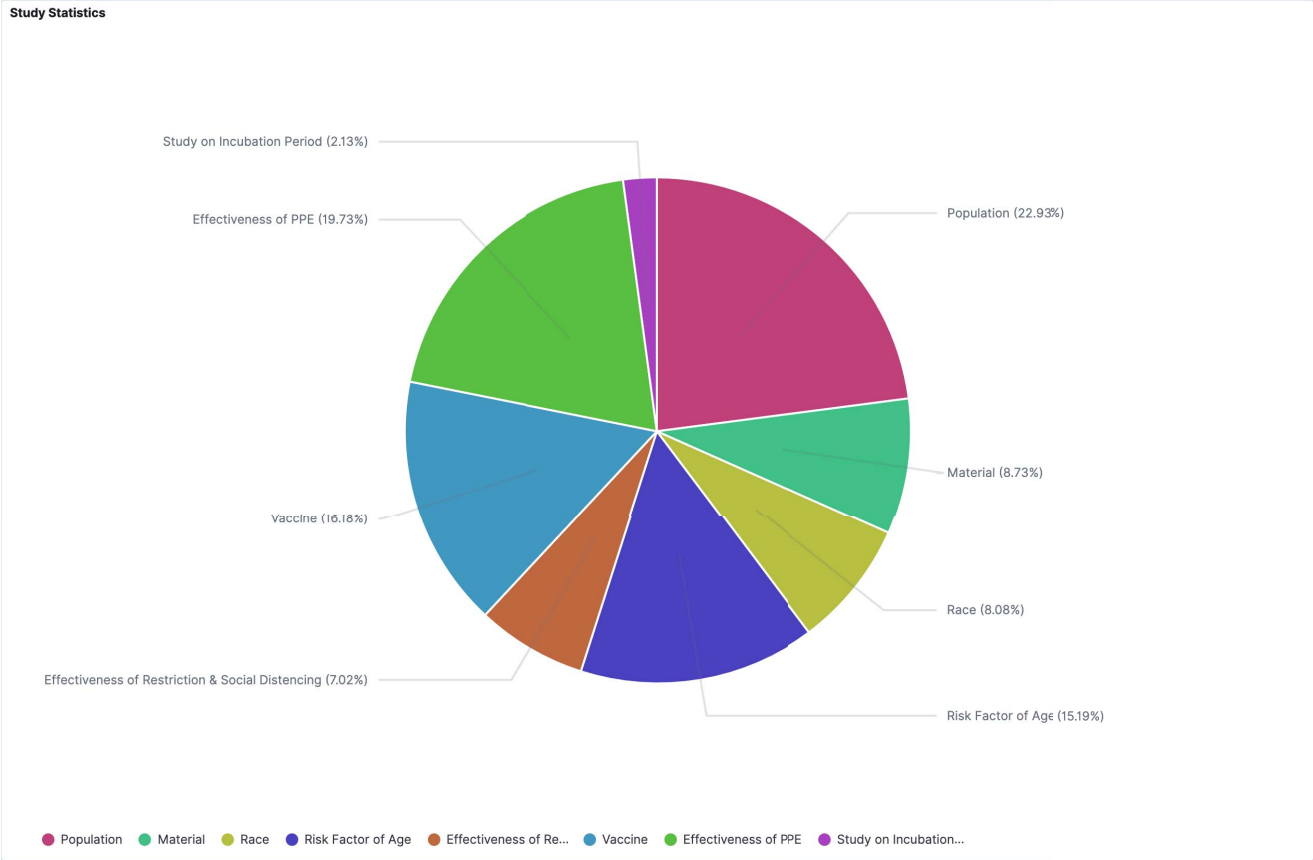

Fig. 5. Example Query 1: Field of Studies Visualized

For future work, the ML model can be improved with higher quality labelled datasets. A significant contribution is the CODA-19 dataset [17]. Huang et. al. used 248 human workers provisioned by AWS Mechanical Turk and created a human-annotated dataset. We plan utilize this dataset to improve our model in the near future.

Also, we are planning on performing more comprehensive tesits with a more extensive data collection. S2ORC [18] is a general-purpose corpus containing 136M+ paper nodes with 12.7M+ full-text papers (as of 27/July/2020) from many different sources. As a future work, we will be testing and benchmarking the system design with a much larger dataset with the aim of iteratively improving the framework.

ACKNOWLEDGEMENT

The public Elasticsearch and Kibana instance is hosted on Melbourne Research Cloud (MRC). We thank MRC for providing computing resource and bandwidth. We also thank Mohammad Goudarzi for proof-reading and suggestions for improvement.

REFERENCES

[1] L. L. Wang, K. Lo, Y. Chandrasekhar, R. Reas, J. Yang, D. Eide, K. Funk, R. Kinney, Z. Liu, W. Merrill, P. Mooney, D. Murdick, D. Rishi, J. Sheehan, Z. Shen, B. Stilson, A. D. Wade, K. Wang, C. Wilhelm, B. Xie, D. Raymond, D. S. Weld, O. Etzioni, S. Kohlmeier, D. Burdick, D. Eide, K. Funk, Y. Katsis, R. Kinney, Y. Li, Z. Liu, W. Merrill, P. Mooney, D. Murdick, D. Rishi, J. Sheehan, Z. Shen, B. Stilson, A. D. Wade, K. Wang, N. X. R. Wang, C. Wilhelm, B. Xie, D. Raymond, D. S. Weld, O. Etzioni, and S. Kohlmeier, "CORD-19: The Covid-19 Open Research Dataset," in *Proceedings of the Workshop on {NLP} for {COVID-19} at {ACL 2020}*. Association for Computational Linguistics, Jul 2020. [Online]. Available: http://arxiv.org/abs/2004.10706https://arxiv.org/abs/2004.10706

[2] "COVID-19 Open Research Dataset Challenge (CORD-19) — Kaggle." [Online]. Available: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge/data

[3] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds," *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861–870, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2011.07.005

[4] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: A Software Platform for .NET Based Cloud Computing," *Advances in Parallel Computing*, vol. 18, pp. 267–295, 2009.

[5] A. Nadjaran Toosi, R. O. Sinnott, and R. Buyya, "Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using Aneka," *Future Generation Computer Systems*, vol. 79, pp. 765–775, Feb 2018.

[6] "CORD-19 Search." [Online]. Available: https://cord19.aws/{#}

[7] "COVID-Miner." [Online]. Available: https://dain.research.cchmc.org/covidminer/

[8] "TEKStack Health - COVID-19 Research Portal." [Online]. Available: https://covid-research.tekstackhealth.com/

[9] "COVID-Miner." [Online]. Available: https://dain.research.cchmc.org/covidminer/

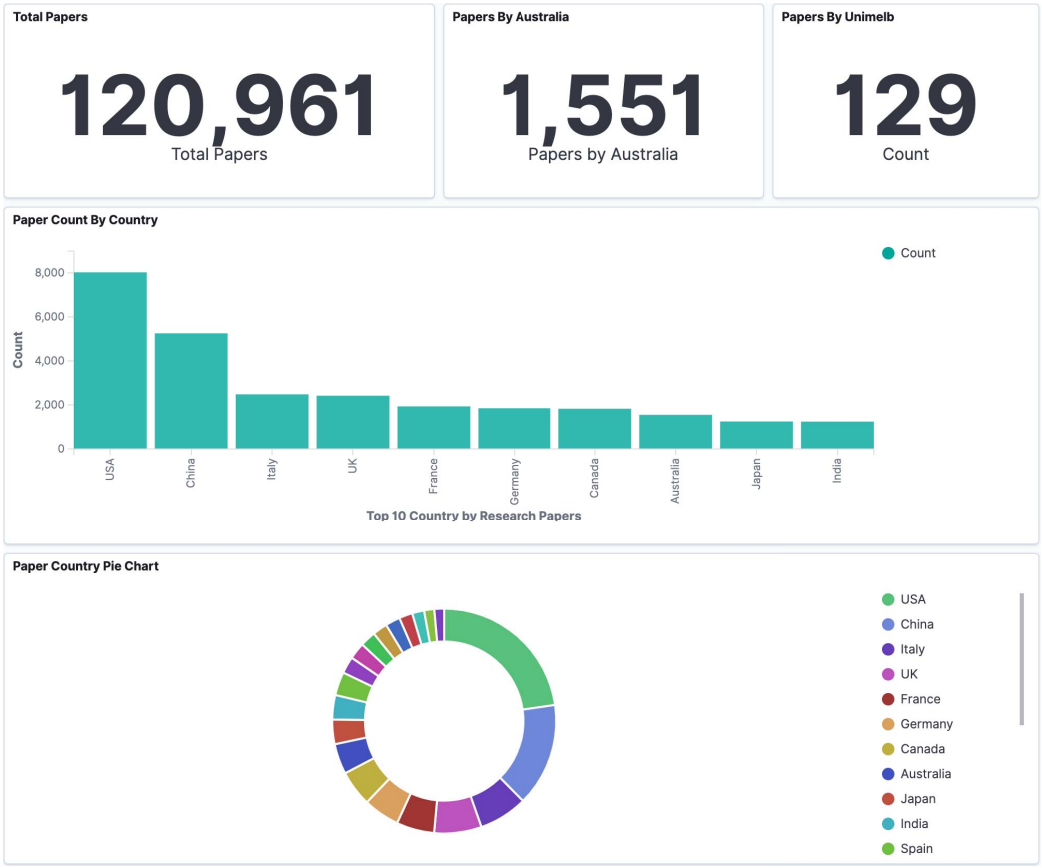

Fig. 6. Example Query 2: Contribution by Country

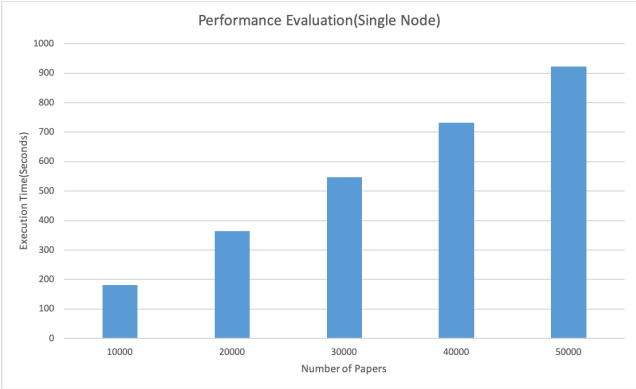

Fig. 7. Single Node Benchmark

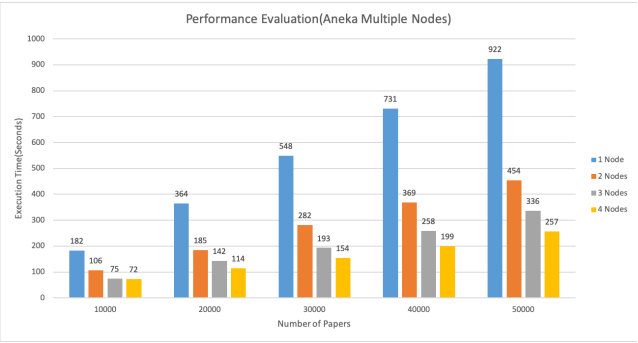

Fig. 8. Single vs Multi Node Benchmark

[10] F. Wolinski, "Visualization of Diseases at Risk in the COVID-19 Literature," May 2020. [Online]. Available: http://arxiv.org/abs/2005.00848

[11] "COVIDSeer." [Online]. Available: https://covidseer.ist.psu.edu/search?query=covid

[12] "COVID Explorer." [Online]. Available: https://coronavirus-ai.psu.edu/database

[13] J. McCaffrey, "ML.NET: The Machine Learning Framework for .NET Developers." *MSDN magazine*, no. 13, p. 24, 2018.

[14] "GROBID." [Online]. Available: https://github.com/kermitt2/grobid

[15] "MinIO — High Performance, Kubernetes Native Object Storage." [Online]. Available: https://min.io/

[16] "Elastic Stack: Elasticsearch, Kibana, Beats & Logstash — Elastic." [Online]. Available: https://www.elastic.co/elastic-stack

[17] T.-H. K. Huang, C.-Y. Huang, C.-K. C. Ding, Y.-C. Hsu, and C. L. Giles, "CODA-19: Reliably Annotating Research Aspects on 10,000+ CORD-19 Abstracts Using a Non-Expert Crowd," May 2020. [Online]. Available: http://arxiv.org/abs/2005.02367

[18] K. Lo, L. L. Wang, M. Neumann, R. Kinney, and D. S. Weld, "S2ORC: The Semantic Scholar Open Research Corpus," in *Proceedings of ACL*, Nov 2019. [Online]. Available: http://arxiv.org/abs/1911.02782