Real-time Multi-granular Analytics Framework for HIT Systems

Byung H. Park*, Sangkeun Lee*, Ozgur Ozmen*, Mohit Kumar*, Merry Ward†, and Jonathan R. Nebeker†
*Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
†U.S. Federal Government
Department of Veterans Affairs

***Abstract*—Streaming analytics is the process of ingesting and digesting live data from multiple data sources. In the healthcare domain, as the importance of extracting immediate insights while data are streaming into the system grows, the focus is shifting from batch processing to streaming analytics. With data increasing dramatically at high speeds, many informatics designs have been proposed to adapt healthcare domain into this new environment. In our previous work, we introduced a prototype of health informatics technology (HIT) framework that aims to address challenges in adopting state-of-the-art technologies to enable advanced healthcare analytic tasks in new streaming environments. We recently made major updates to the framework so that anomaly from multiple streaming data sources at different granularity levels can be detected in near real-time. In this paper, we detail the implementation and deployment of the framework in Kubernetes clusters and report its performances when tested on electronic health record (EHR) data of Veterans Affairs.**

***Index Terms*—HIT, Big Data, Multi-granular, Streaming Architecture**

I. Introduction

Health Information Technology (HIT) data has shown exponential growth in the last decade, reaching zettabytes [1] in size. Most HIT data analytic platforms are based on batch data processing (e.g., offline generation of a statistical report using the entire data); however, with such a dramatic increase in the rate at which data is being accumulated, one of the new requirements of HIT today is the capability of processing large-scale streaming data in near real-time. This involves dealing with multiple data sources, performing multi-granular (i.e., temporal multi-scale aggregation) data analytics, and visualizing the results.

Although advanced big data technologies offer a wide variety of tools to meet these new HIT requirements, adoption of such state-of-the-art technologies is a challenging task, as there are hundreds of tools and thousands combinations of tools to choose from [2]. Moreover, these technologies are developed for various business use cases that may or may not fit the problem domains of interest. In this work, we built on top of our previous work [3] and demonstrate our analytic platform's capability. We first describe the updated real-time multi-granular data analytic platform that is designed and implemented by selecting technologies that showed the best fit against the functional requirements. For example, Apache Kafka and Spark Structured Streaming Engine are selected for streaming analytics and Delta Lake was selected to store multi-modal data. Second, we demonstrate the capabilities of the platform using a statistical process control-based monitoring tool that detects hazards in the electronic health record (EHR) system of the Department of Veterans Affairs. We re-implemented this detector algorithm as a streaming-oriented detector in our analytic platform. We selected this use case because it represents a very common ad-hoc slice-and-dice analytics [4] need among the corporations. Our platform can help automating those ad-hoc needs. Lastly, we discuss the statistical process control-based hazard detection algorithm deployment on Kubernetes and evaluate its performance metrics to present its potential.

The rest of this paper is organized as follows: Section 2 describes the functional requirements for our HIT framework, Section 3 introduces the parallel multi-granular streaming analytics capability, Section 4 provides details of the deployment of the statistical process control-based hazard detection algorithm on the platform and investigates the performance, and Section 5 concludes the work and discusses possible future directions.

II. HIT Analytic Framework Architecture for Streaming Analytics

A prerequisite step for constructing a new streaming platform is to identify components of legacy systems that do not support efficient streaming analytics in a scalable fashion while data from multiple sources inflows. This requires investigating the back-end data models and the front-end computational frameworks considering functional requirements for streaming analytics. To achieve that, we evaluated existing state-of-the-art technologies considering five functional aspects.

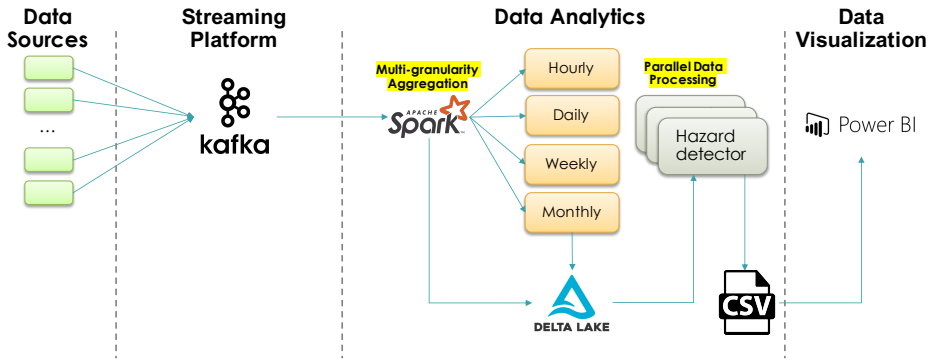


Fig. 1: Architecture of the multi-granular streaming analytic framework applied to detect hazards (anomalies) from multiple data sources. Spark structured streaming engine and multiprocess data processing module detects hazards more efficiently compared with the legacy system. The front-end user interface including visualization of output is implemented using the Power BI tool

A. *Functional Requirements*

The five functional requirements include:

- Processing data from multiple data sources (potentially heterogeneous)
- Ingesting and digesting streaming data
- Efficient data processing
- Ensuring ACID properties
- Flexible data visualization

We analyzed all the functional requirements and came up with the parallel multi-granular streaming analytic framework shown in Fig. 1. This section describes the components of the framework and different big data technologies the framework takes advantage of.

Since we assume that data are generated from multiple sources, we chose Apache Kafka [5] as the data landing platform. It is an open-source tool for handling streaming data in a distributed manner that can ingest fast inflow data with negligible latency. To serve as a data source, each data producer only needs to implement a message publisher before sending data stream to our framework. We then connected Apache Kafka to Apache Spark [6] which supports various programming languages and functionalities including stream processing. This connection allows event-at-a-time processing with millisecond latency.

Apache Spark is configured to receive streaming data from Kafka by subscribing to it. Once consumed, the data is processed and stored in Delta lake [7]. Delta lake is a storage layer that guarantees ACID properties for computation engines like Spark. All data in this storage layer are stored in Apache Parquet [8] format, providing efficient compression and encoding schemes. We chose Delta Lake over other data storage options because Delta Lake is 100% compatible with Apache Spark, and can be configured with other storage backends like Amazon S3 and Azure Data Lake Storage.

For the front-end visualization and user interactions, we chose Power BI [9], a commercial BI tool that converts data into coherent, visually immersive, and interactive insights. To make it compatible with Delta Lake, the framework produces results in CSV format which Power BI tool consumes.

III. Implementation and Deployment of the Framework

This section details implementation of the framework and deployment of hazard detection algorithm for healthcare data of Veterans Affairs (VA).

A. *Modules*

Our framework is implemented as four modules: Data Sources, Streaming Platform, Data Analytics, and Data Visualization.

a) Data Sources: A data source is any stream source that implements Kafka producer. Technically, it needs to publish data to one or more topics in Kafka cluster. The Veterans Health Information Systems and Technology (VISTA)[1] is a nationwide health information system used in all VA facilities. All regional VISTA data, which includes clinical, administrative, and financial records, are integrated into the central Corporate Data Warehouse (CDW) [2]. An update to CDW is currently not real-time; it is a daily batch update. However, since VA is transitioning into real-time updates from multiple data sources including Department of Defense, hospitals, laboratory systems, medical devices, new EHR implementation, and consumer devices [10], we decided to adapt our framework to such an environment. For this, we emulated data sources using a snapshot of VA clinical data from January 2018 to October 2019. More specifically, we created a Kafka Producer implemented with Kafka Python library [11] to send data continuously with a time delay between data rows, where the delay is adjustable.

b) Streaming Platform: We implemented the streaming platform with a Kafka cluster, i.e. Kafka with more than one broker with an intention to provide resiliency even during the downtime of the main broker. For the evaluation of the framework on the VA data, we created multiple emulated data sources that each has a different amount of delay between data rows.

[1]https://www.data.va.gov/dataset/veterans-health-information-systems-and-technology-architecture-vista.

[2]https://www.data.va.gov/dataset/corporate-data-warehouse-cdw.

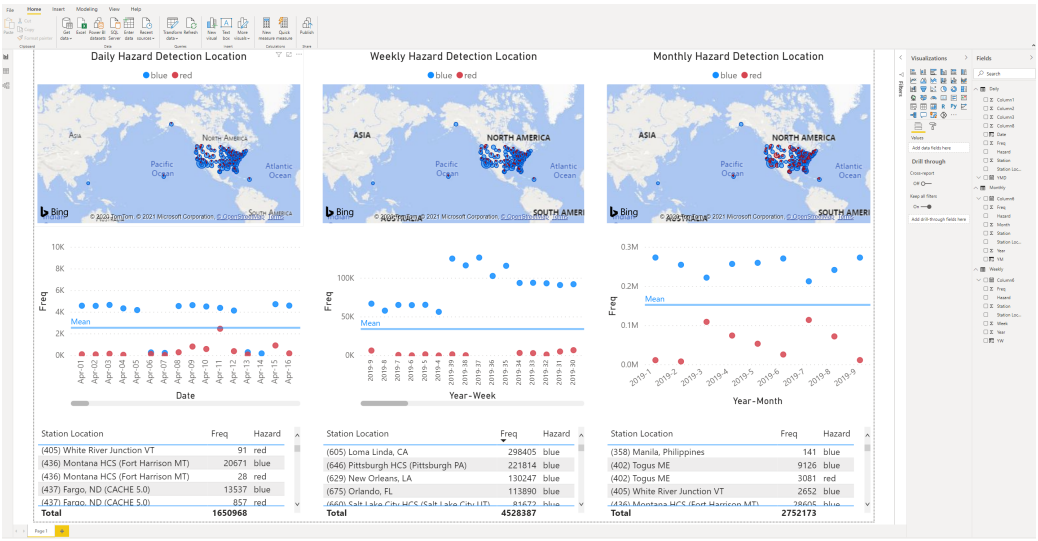


Fig. 2: Power BI front end (user interface) designed for hazard (anomaly) detection.

c) Data Analytics: For flexible and scalable analysis of streaming data, we designed the data analytics module to perform data aggregation at different granularity levels in parallel. By implementing multiple *Kafka Consumer*, Spark is configured to consume multiple streaming data and compute hourly, daily, weekly, and monthly statistics to detect anomalies at different resolutions. Both consumed results and statistics are stored in Delta Lake.

d) Data Visualization: Power BI, the interactive data visualization module is configured to access and display results stored in CSV format and display as illustrated in Fig. 2, where each anomaly detection type (e.g., hourly, weekly, monthly aggregation) is separately reported.

For each detection type, we show:

- Interactive map to show station location with embedded pie chart
- Interactive scatter plot to show station frequency
- Interactive table to look at the details

B. Kubernetes Deployment

The application is deployed in Kubernetes. Kubernetes is a system for automatic deployment, management and scaling of containers. A Kubernetes cluster consists of multiple nodes which run application container, where a node is a worker machine which can be a virtual or physical.

Helm is a package manager for Kubernetes. Helm charts are the collection of YAML files which describe Kubernetes resources for a specific application. Fig. 3 shows the helm chart for HDFS setup in cluster.

We first deployed our application on an enterprise cluster. We configured the cluster with 68 cores and 272 GB of memory (RAM) for the deployment. To mitigate complexities in managing Kubernetes, we adopted Lens IDE for management. Fig. 4 shows the home page of Lens for our application.

Once the Kubernetes cluster was configured and tested on ORNL enterprise cluster, we then moved it to the secured

job-dfssetup.yaml 824 Bytes

```
{{- if .Values.hithd.hdfs.manage_permisisons }}
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ include "hit-hd.fullname" . }}-dfssetup"
  labels:
    {{ include "hit-hd.labels" . | nindent 4 }}
  annotations:
    "helm.sh/hook": post-install,post-upgrade
spec:
  completions: 1
  backoffLimit: 10
  ttlSecondsAfterFinished: 3600
  template:
    spec:
      restartPolicy: OnFailure
      containers:
        - name: dfssetup
          image: gradiant/hdfs:2.9.2
          command:
            - bash
            - -exc
          args:
            - |
              if ! hdfs dfs -test -d {{ include "hit-hd.hdfsBase" . | quote }}; then
                hdfs dfs -mkdir {{ include "hit-hd.hdfsBase" . | quote }}
              fi
              hdfs dfs -chown hit:hit {{ include "hit-hd.hdfsBase" . | quote }}
{{- end }}
```

Fig. 3: Helm chart used to setup our framework with HDFS (Example).

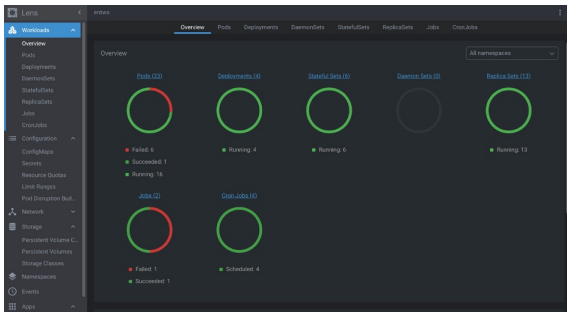


Fig. 4: Snapshot of Lens IDE shows deployment status of Kubernetes cluster for our streaming analytic framework.

environment of ORNL-VA enclave called BlueRidge from which VA healthcare data can be accessed.

IV. Use Case Evaluation

Our use-case is a hazard detection algorithm that monitors the clinical order cancellations throughout VHA hospitals.

The legacy system conducts daily aggregation of the order cancellations and identifies anomalous number (very high frequency) of order cancellations using statistical process control technique. Every day, one year past window of time-series data is leveraged to calculate mean and standard deviation and new daily data is compared against mean + 3 standard deviations threshold. If the cancelled order count is higher than the threshold, then that station and date are tagged (see [12] and [13]). In our deployment, we run the same algorithm to evaluate different components of our framework at various granularities (other than daily). We leverage a snapshot of clinical data that ranges from Jan. 2018 to Oct. 2019 for all the evaluations. We tested our framework deployed in three different cluster configurations.

1) Baseline configuration on a local machine of Quad-Core processor and 16 GB of memory
2) Configuration on ORNL Enterprise cluster of 68 cores with 272 GB of memory
3) Configuration on ORNL-VA enclave cluster of 68 cores with 272 GB of memory

A. Scenarios

We calculate various statistics of the data to get a better picture of the streaming input. This frequency helps us to determine the data rate when emulating streaming data. We iterate the snapshot of the data and send it to Apache Kafka one at a time. We adjust the sleep time between messages to control the data rate. The Table I shows different statistics at an aggregation rate of minute, hour, and day. The temporal distribution of the data is shown in Fig. 5. The maximum number of messages in a minute is around 7,937 with a mean of 18. To achieve the same type of frequency, we performed sleep operation between sending messages at a delay of 3 s, 0.3 s, and 0.007 s that represent mean, worse, and the worst cases, respectively.

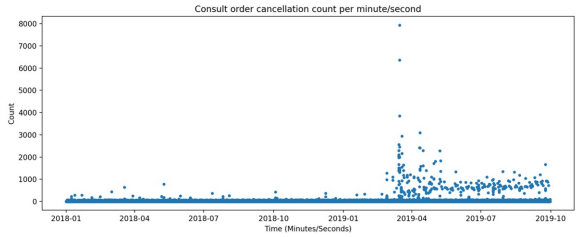


Fig. 5: Temporal distribution of VA EHR data occurrences aggregated at minute level.

B. Producer

We start sending the data with different frequencies from producer to Apache Kafka topic. Producer script ingests time-series order cancellation data and feed it to Apache Kafka. To explore this time series data to understand its nature, we ran the producer code for six hours and measured the message counts of the Kafka topic at a five minute interval. Fig. 6 shows the increase in message counts in Kafka overtime in log scale on the cluster and local machine. We observed a rapid increase in the number of messages in Kafka topic as the sleep interval time decrease on both deployments.

TABLE I: Statistics of data aggregated at different granularity levels.

Statistics	Minute	Hour	Day
Count	579188.000000	15251	637.000000
Mean	18.8	713.98	17094.031397
Std	31.86	1076.19	10656.934648
Min	1	1	496.000000
25%	2	21	2356.000000
50%	7	87	22243.000000
75%	36	1433	24228.000000
Max	7937	36987	58632.000000

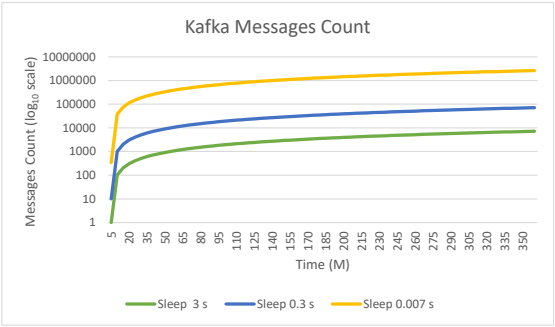
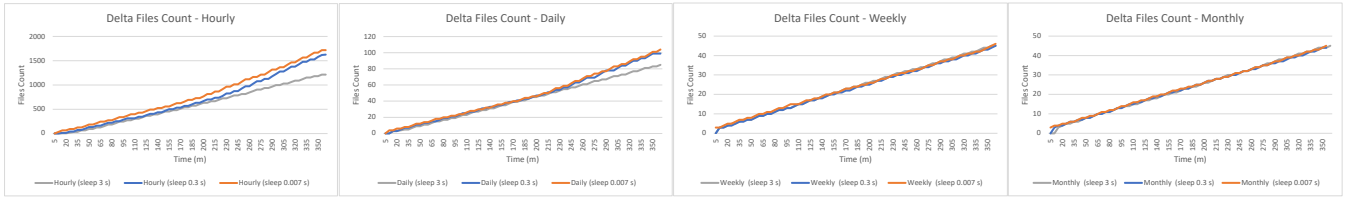


Fig. 6: Accumulation of Kafka message counts with different delay lengths between messages.

C. Aggregation

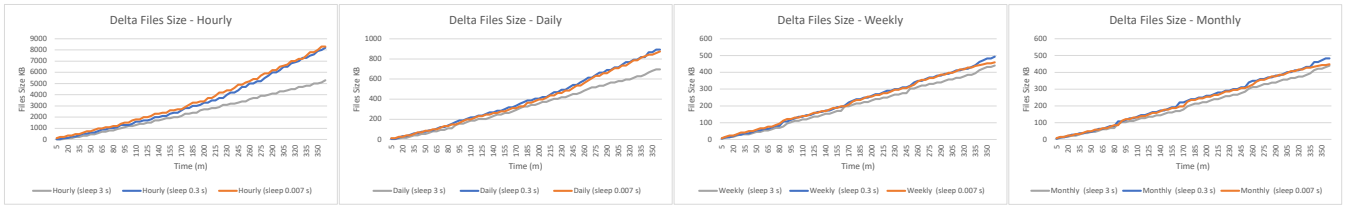
We measured the amount of aggregated data generated by Apache Spark while different sizes of data are sent to Apache Kafka. We observe this at the Apache Spark Streaming tab, which provides various metrics to measure the streaming performance. The aggregation was performed for six hours and we calculated the file counts at a five minute interval. Fig. 7 presents the increase in delta lake file counts overtime. We noticed that the number of delta lake files increase at a lower granularity. For hourly, the max delta lake files count is 1,717 whereas for monthly, the max delta lake files count is 46. We can also see that a lower sleep interval result in more delta lake files. The difference is more visible at a lower granularity that is hourly as more data is generated while performing hourly aggregation.

We also investigate the size of delta lake files. We measure the delta lake file sizes for the same six hours at an interval of five minutes. Fig. 8 shows the increase in delta lake files size overtime. Delta lake files size presents a better picture of how lower granularity in aggregation and sleep interval results in more file size.



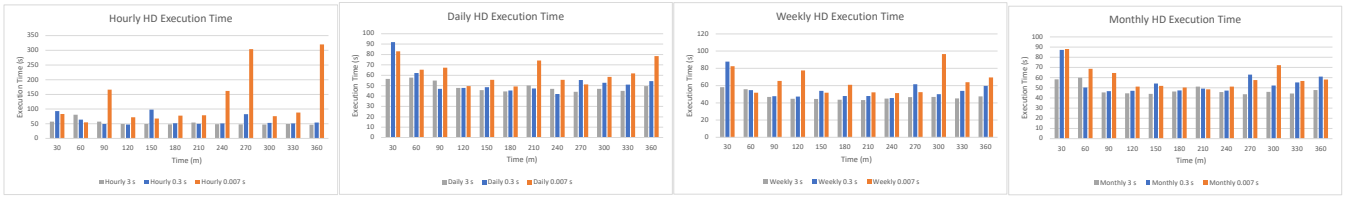
(a) Hourly (b) Daily (c) Weekly (d) Monthly

Fig. 7: File counts of Delta files measured at different aggregation levels.



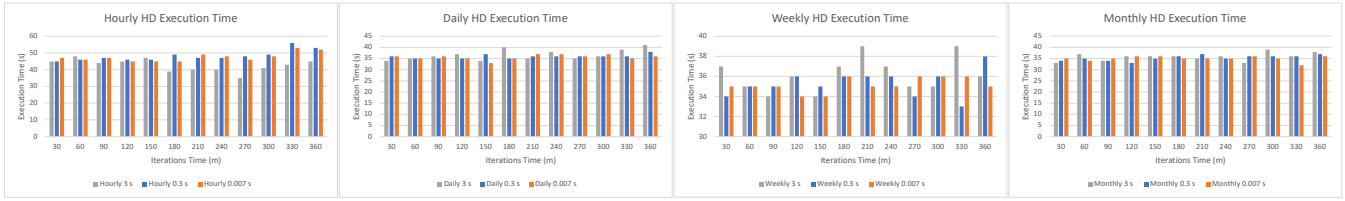
(a) Hourly (b) Daily (c) Weekly (d) Monthly

Fig. 8: Sizes of Delta files measured at different aggregation levels.



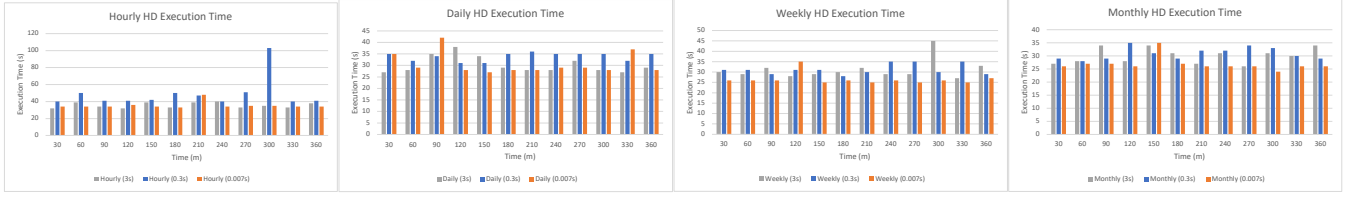
(a) Hourly (b) Daily (c) Weekly (d) Monthly

Fig. 9: Baseline Performance of hazard detection on a Local computer.



(a) Hourly (b) Daily (c) Weekly (d) Monthly

Fig. 10: Performance of hazard detection of the ORNL Enterprise Kubernetes cluster.



(a) Hourly (b) Daily (c) Weekly (d) Monthly

Fig. 11: Performance of hazard detection of the BlueRidge Kubernetes cluster (VA-ORNL Enclave).

TABLE II: Average execution time of hazard detection on baseline, enterprise cluster, and BlueRidge.

Scenario	Scenario 1 (message delay 0.007 s)				Scenario 2 (message delay 0.3 s)				Scenario 3 (message delay 3 s)			
	Hourly	Daily	Weekly	Monthly	Hourly	Daily	Weekly	Monthly	Hourly	Daily	Weekly	Monthly
Baseline	53.099	49.147	47.314	48.102	62.051	53.807	54.789	55.069	129.241	62.496	64.614	59.956
Enterprise Cluster	42.666	36.666	36.166	35.750	48.250	35.916	35.333	35.333	47.583	35.667	35.167	35.000
BlueRidge	35.416	30.666	26.500	26.833	48.833	33.833	31.250	30.916	35.583	30.250	31.083	30.083

TABLE III: Average execution time reduction in percentage (%) on enterprise cluster, and BlueRidge compared with the baseline.

Scenario	Scenario 1 (message delay 0.007 s)				Scenario 2 (message delay 0.3 s)				Scenario 3 (message delay 3 s)			
	Hourly	Daily	Weekly	Monthly	Hourly	Daily	Weekly	Monthly	Hourly	Daily	Weekly	Monthly
Enterprise Cluster	19.647	25.394	23.560	25.679	22.241	33.249	35.510	35.838	63.182	42.929	45.574	41.624
BlueRidge	33.301	37.602	43.991	44.216	21.301	37.121	42.963	43.858	72.467	51.597	51.894	49.824

D. Hazard detection

We analyze hourly, daily, weekly, and monthly runs of the hazard detection algorithm as different scenarios. Fig. 9 - 11 shows the hourly, daily, weekly, and monthly hazard detection execution time for different sleep intervals on a local computer (baseline), on the enterprise cluster, and inside the enclave. We schedule different hazard detection as a cron job and ran it for six hours at an interval of half hour. Notice that the hourly hazard detection code took longer when the hazard detection algorithm is performed each hour of the available data compared to monthly aggregation. We can also see that for almost all scenarios, the hazard detection is performed in less then one minute on the enterprise cluster and inside the enclave. However, on the local machine, it increased to almost six minutes which is due to less computational power.

Table II shows the average execution time of hazard detection algorithm on the local computer, on the enterprise cluster, and inside the enclave. Table III compares the performance (execurtion time reductions) on the Kubernetes cluster inside the BlueRidge and on the enterprise cluster with the baseline performance. The hourly runs on enterprise cluster and BlueRidge are resulted in up to 63% and 72% performance improvements, respectively, compared to the baseline. In most cases, hazard detection on BlueRidge performs better than enterprise cluster.

V. Conclusion

In this work, we introduce a parallel framework with streaming capability for HIT systems that can ingest data from multiple data sources and perform multi-granular analytics to provide real-time insights. We started by identifying functional requirements and then designed a generic architecture. We then tailored it towards a use case and configured it. In this paper, we demonstrated its use for an operational hazard detection algorithm to show its runtime benefits and potential for future use-cases. Our evaluation shows that the Kubernetes deployment of the framework result in up to 72% lower execution time compared to single machine batch processing of the same tasks. Please note that the data that was evaluated include a single domain (consult) cancelled orders data. CDW has numerous domains (i.e., radiology, laboratory, outpatient medications) that generate millions of orders per day in approximately 130 Veterans Affairs hospitals throughout the US. Besides the latency in analytics, the use of batch processing for all orders on local computers exacerbates the runtime problem and our parallel implementation can scale significantly for larger use-cases when more compute resources are provided. In future work, we will evaluate our work on all data domains for multiple granularities in an operational setting.

Acknowledgment

This work is sponsored by the US Department of Veterans Affairs under Inter-Agency Agreement number VA118-17-M-2015.

References

[1] W. Raghupathi and V. Raghupathi, "Big data analytics in healthcare: promise and potential," *Health information science and systems*, vol. 2, no. 1, p. 3, 2014.

[2] A. Foundation, "Apache software foundation," *URL https://www.apache.org*, 2022.

[3] M. Kumar, S. Lee, B. H. Park, J. Blum, M. Ward, and J. R. Nebeker, "Advanced health information technology analytic framework and application to hazard detection," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 1–4.

[4] M. Stacey, J. Salvatore, and A. Jorgensen, "Slice and dice: Ad hoc analytics," *Visual Intelligence: Microsoft Tools and Techniques for Visualizing Data*, pp. 269–298, 2013.

[5] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, 2011, pp. 1–7.

[6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica *et al.*, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[7] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H. van Hovell, A. Ionescu, A. Łuszczak *et al.*, "Delta lake: high-performance acid table storage over cloud object stores," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, 2020.

[8] A. Parquet, "Apache parquet," *URL https://parquet.apache.org/*, 2022.

[9] B. Power, U. Excel, P. B. Desktop, and P. Tiles, "Microsoft power bi," *Available here: https://powerbi. microsoft. com/en-us*.

[10] R. E. Gliklich, M. B. Leavy, and N. A. Dreyer, "Tools and technologies for registry interoperability, registries for evaluating patient outcomes: A users guide, addendum 2 [internet]," 2019.

[11] D. Powers, "kafka-python," *https://pypi.org/project/kafka-python/*.

[12] O. A. Omitaomu, O. Ozmen, M. M. Olama, L. L. Pullum, T. Kuruganti, J. Nutaro, H. B. Klasky, H. Zandi, A. Advani, A. L. Laurio *et al.*, "Real-time automated hazard detection framework for health information technology systems," *Health Systems*, vol. 8, no. 3, pp. 190–202, 2019.

[13] O. A. Omitaomu, H. B. Klasky, M. Olama, O. Ozmen, L. Pullum, A. M. Thakur, T. Kuruganti, J. M. Scott, A. Laurio, F. Drews *et al.*, "A new methodological framework for hazard detection models in health information technology systems," *Journal of Biomedical Informatics*, vol. 124, p. 103937, 2021.