DisasterComm: A Severity-Aware Big Data Communication Framework for Effective Disaster Notification

Animesh Giri, Anika Yadav, and Ananya V Holla

PES University, Bengaluru 560100, India
anikaprag@gmail.com,ananyaholla04@gmail.com,animeshgiri@pes.edu

Abstract. With the growing use of connected devices, geographic barriers diminish, enabling communication of critical data in near real-time. As the frequency of natural disasters increases, timely communication forms the backbone of efficient disaster management. With the number of recipients varying based on the population density, any disaster notification framework must be highly scalable. Big Data technologies like Apache Kafka, Apache Flink, RabbitMQ, Apache Spark, and Apache Hadoop can be employed to enhance scalability. This study proposes a hybrid framework that classifies disasters based on severity and then routes them through one of three pipelines. This approach ensures that the Quality of Service requirements for each disaster severity type are satisfied. The pipeline assigned to handle high-severity traffic demonstrates notification delivery with latency in the order of 0.88 seconds on average across the disaster types.

Keywords: Disasters· Apache Spark · Apache Kafka · Apache Flink · Apache Hadoop· RabbitMQ · Big Data · Quality of Service.

1 Introduction

The number of disasters, both natural and man-made has risen significantly over the last 50 years. With this increase in frequency being attributed to factors like climate change, global warming, and rapid urbanization, this trend is unlikely to reverse. Disasters that occur in densely populated cities require effective communication methods, so they can be dealt with promptly.

Since disaster communication must reach a large number of recipients in a short span of time, Big Data pipelines can be used to enhance emergency notifications. Notifications of all disasters do not need to be processed with the same urgency, and hence, it is important to classify them. This study classifies disasters into one of three severity levels: high, medium, and low and processes them accordingly.

This study proposes a novel disaster communication system that relies on one of three big data pipelines to communicate information about a disaster. A disaster is categorized as high, medium, or low severity based on its impact and

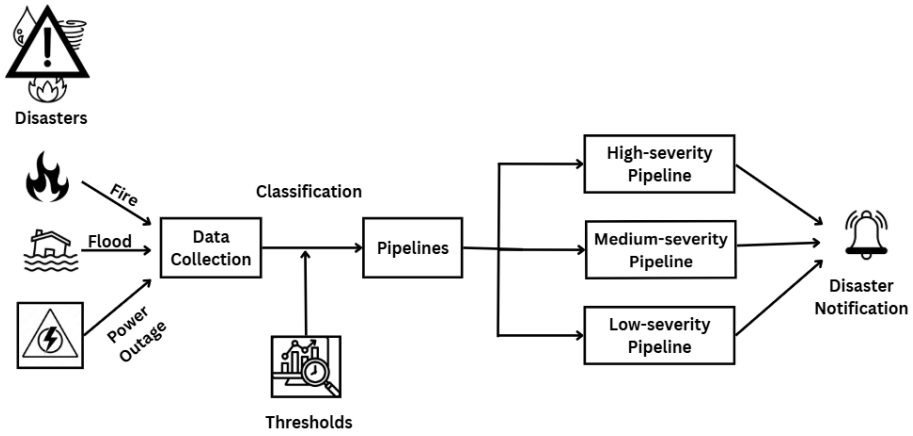


Fig. 1: Severity-based disaster notification system

the possible extent of damage. Three pipelines are set up to handle these three types of traffic as shown in Fig. 1.

Using three different pipelines to cater to three different severities ensures efficient management. Every pipeline has a single responsibility and is optimised to satisfy the Quality of Severity (QoS) requirements of that traffic only.

The primary contributions of this study are:

- **Dataset generation:** A synthetic dataset is generated to detail three kinds of disasters, each corresponding to one of three severity levels.
- **Big Data pipelines:** Three pipelines are set up. The high-severity pipeline uses Apache Kafka and Apache Flink to minimize latency. The medium-severity pipeline uses RabbitMQ and Apache Spark to balance latency with computational complexity. The low-severity pipeline uses Apache Hadoop since a higher latency is tolerable.
- **Performance Insights:** This study provides a comprehensive comparative analysis of the three proposed pipelines over metrics like throughput and latency. These metrics comprehensively highlight the differences stemming from their architectural differences.

The remainder of this study is structured as follows: Section II highlights key contributions in this domain. Section III discusses the proposed framework and architecture. Section IV discusses the performance metrics used to evaluate the pipelines. Section V discusses the results observed in this study.

2 Related Work

Sarkar et al.(2024) proposed an IoT sensor, MQTT, Kafka, and Spark Streaming-based real-time urban evacuation system to determine dynamic risk scores for

the exits, with cloud-based Hadoop for storing data and a web interface for real-time notification in stadium emergency situations.

Akanbi et al.(2020) suggested a distributed stream processing middleware based on Apache Kafka for real-time processing of heterogeneous environmental data, fusing legacy systems and IoT sensors to support drought prediction with the Effective Drought Index (EDI) model.

Zhang et al.(2023) designed a real-time SHM system with Apache Kafka and Flink, where watermarks are used in time-window calculation and parallel processing for minimizing latency with 200-400 ms response times for Shanghai Tower structural health monitoring.

Dou et al.(2021) designed SeisFLINK, a near real-time earthquake monitoring system based on Apache Kafka for data stream handling and Apache Flink for parallel processing, by accomplishing two-minute rapid earthquake reports with the use of STA/LTA and FPS algorithms to detect seismic phases and correlate events.

Wang et al.(2025) designed a Flink-based system with APSO load balancing for multi-source landslide data integration, employing Kafka for real-time pipelines and LSTM for cluster load prediction, enhancing task execution time by 4.7% and throughput by 5.4%.

Gowda V et al.(2024) found that real-time data analytics through Hadoop, Spark, machine learning (e.g., decision trees, neural networks), and IoT sensors improve disaster management by providing early detection, predictive modeling, and effective resource allocation, reducing response time and resilience.

Pandey et al.(2025) discovered that sensor networks, which are supported by IoT and powered by AI, ML, and communication protocols such as MQTT and CoAP, improve manufacturing, agriculture, healthcare, and smart city process control in real-time, making scalability, power efficiency, and data security much easier to meet.

Lijuan Cao et al.(2024) found that Apache Kafka with real-time monitoring and smart alarm functions guarantees the effective and stable transfer of ocean observation data, reaching 980 entries/second throughput, less than 0.1% packet loss, and 50 ms latency, surpassing conventional methods such as RabbitMQ.

Guo Fu et al.(2021) found that Kafka performs better than RabbitMQ, RocketMQ, ActiveMQ, and Pulsar in terms of throughput (980 messages/second) and scalability but is surpassed by RocketMQ in low-latency environments, according to a benchmarking test of messaging systems using batch processing, zero-copy, and distributed architecture.

S. A. Shah et al.(2019) reported that the combination of IoT and Big Data has the potential to improve disaster resilience in smart cities. They designed a five-layer architecture including data harvesting, aggregation, preprocessing, analytics, and service platforms. Based on Apache Hadoop and Apache Spark, the system supports real-time and offline analysis for fire detection, pollution monitoring, and emergency evacuation.

Giri A. et al.(2023) proposed a disaster-resilient smart city framework using cross-layer protocol analysis for earthquake emergency response. The system in-

tegrates IoT, AI, and big data analytics for real-time detection, low-latency communication, and adaptive resource allocation. This enhances disaster resilience and recovery efficiency.

3 Proposed Work

The system uses three different pipelines to accommodate disasters of varying severity, as shown in Fig. 2. A comparison of the three pipelines is shown in Table. 1. The types of disasters considered are fires, floods, and power outages, each of which can be classified as high, medium, or low severity. The disasters are characterised by specific attributes. The threshold values for these attributes are defined in Table. 2. To ensure accurate disaster characterization, threshold values are obtained either from data published by relevant authorities or are widely accepted values. Based on these thresholds, each disaster is classified into high, medium, or low severity. The disaster is then routed to the appropriate pipeline. Each pipeline calculates the number of events processed, average latency, and throughput over a 60-second window. To ensure the pipelines stabilize, averages are taken over five such windows.

Table 1: Comparison of High, Medium, and Low-Severity Pipelines

Feature	**High-Severity**	**Medium-Severity**	**Low-Severity**
Technologies Used	Apache Kafka, Apache Flink	RabbitMQ, Apache Spark	Apache Hadoop (MapReduce)
Processing Model	Real-time streaming (event-driven)	Near-real-time streaming	Batch processing
Latency	**Milliseconds**	**Seconds**	**Minutes**
Throughput	High (lower than low-severity)	Moderate	Highest
Use Case	Immediate response (critical disasters)	Timely but non-urgent response	Delayed response acceptable
Data Handling	Processes events immediately as they arrive	Processes in micro-batches at fixed intervals	Processes large batches periodically

3.1 Data generation

This study uses a synthetic dataset that acts as the input to the pipelines. Disaster event logs are generated continuously in JSON format. The attributes for each disaster (fire, flood, and power outage) are chosen to simulate real-world scenarios. The number of high, medium, and low-severity disasters generated

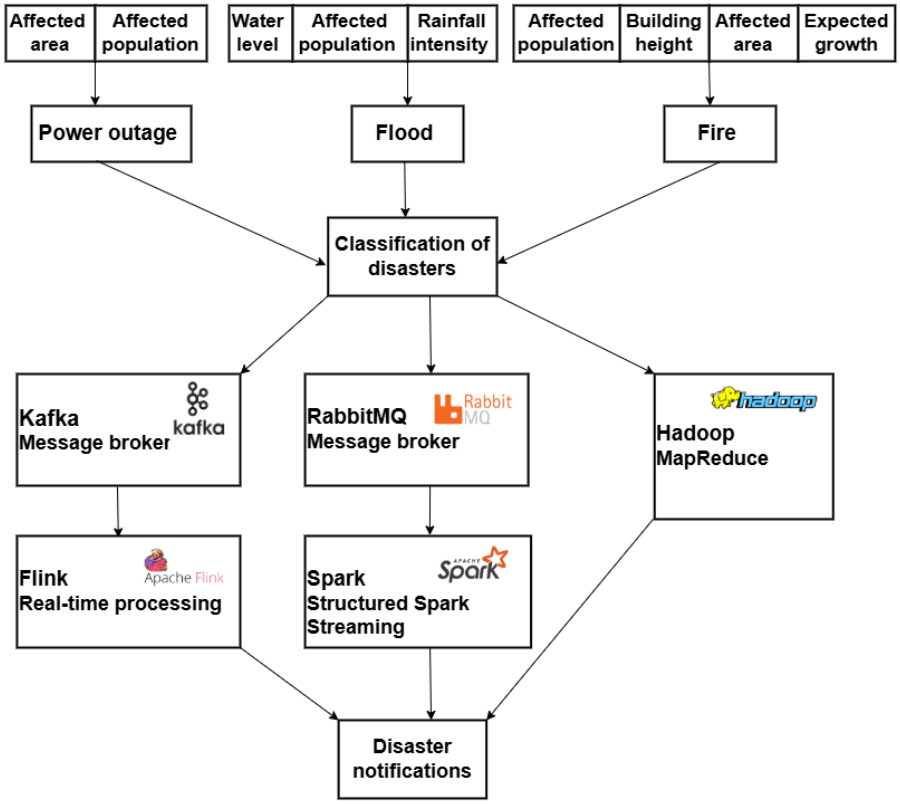


Fig. 2: Proposed system architecture for disaster notification

is approximately equal to ensure a fair performance comparison of the three pipelines. The rate of data generation is varied to study its effects on the three pipelines. The generated data is distributed to all three pipelines. This ensures that a fair comparison can be drawn between the performances of the three pipelines. A sample of the generated data is shown in Fig. 3.

3.2 High severity pipeline

When processing high-severity disasters, it is crucial to minimise latency so that immediate action can be taken. The high-severity pipeline consists of two components, namely Apache Kafka and Apache Flink. Apache Kafka publishes the data to a dedicated topic called `high_severity_disaster`. Flink's Kafka Connector enables it to subscribe to this topic and receive the relevant data. Apache Kafka

Table 2: Severity classification for different disasters

Disaster Type	**Attribute**	**High Severity**	**Medium Severity**	**Low Severity**
Flood	Rainfall intensity	>124.5 mm/hour	35.6–64.4 mm/hour	2.5–35.5 mm/hour
	Water level	>1.5 meters	0.5–1.5 meters	0–0.5 meters
	Affected population	>20,000	10,000–20,000	Below 10,000
Fire	Affected area	>1000 m^2	300–1000 m^2	<300 m^2
	Building height	>45 meters	15–45 meters	<15 meters
	Affected population	>1000	300–1000	<300
	Expected growth	$t^2 < 75\,s$	$75\,s \leq t^2 \leq 300\,s$	$t^2 > 300\,s$
Power Outage	Affected area	>50 km^2	20–50 km^2	Below 20 km^2
	Affected population	>30,000	10,000–30,000	<10,000


```
{"disaster_type": "fire", "affected_area": 519.72, "building_height": 32.13, "affected_population":
 352, "fire_growth": "t^2 75-300s", "severity": "Medium", "timestamp": "2025-03-26 15:59:32.866278"
}
{"disaster_type": "flood", "rainfall_intensity": 5.98, "water_level": 0.26, "affected_population":
6739, "severity": "Low", "timestamp": "2025-03-26 15:59:32.870293"}
{"disaster_type": "power_outage", "area_affected": 57.63, "affected_population": 91842, "severity":
 "High", "timestamp": "2025-03-26 15:59:32.864067"}
{"disaster_type": "power_outage", "area_affected": 8.49, "affected_population": 3704, "severity": "
Low", "timestamp": "2025-03-26 15:59:32.868339"}
{"disaster_type": "fire", "affected_area": 504.58, "building_height": 38.22, "affected_population":
 455, "fire_growth": "t^2 75-300s", "severity": "Medium", "timestamp": "2025-03-26 15:59:32.866278"
}
{"disaster_type": "fire", "affected_area": 882.83, "building_height": 43.51, "affected_population":
 885, "fire_growth": "t^2 75-300s", "severity": "Medium", "timestamp": "2025-03-26 15:59:32.866278"
}
{"disaster_type": "power_outage", "area_affected": 8.27, "affected_population": 9010, "severity": "
Low", "timestamp": "2025-03-26 15:59:32.867368"}
{"disaster_type": "flood", "rainfall_intensity": 37.72, "water_level": 1.3, "affected_population":
18872, "severity": "Medium", "timestamp": "2025-03-26 15:59:32.866278"}
```


Fig. 3: Disaster data generated in real-time

provides high throughput, durability, scalability, real-time event streaming, and seamless integration with Apache Flink for low-latency disaster processing.

Apache Flink is responsible for processing the high-severity data, which it receives from the dedicated high-severity topic. This component processes all the data in real-time, as Apache Flink has an event-driven architecture with stateful stream processing, enabling low-latency computation. These features ensure immediate and reliable disaster notification delivery.

3.3 Medium severity pipeline

Medium-severity disasters can tolerate slightly higher latency values as compared to high-severity disasters. Hence, the medium-severity pipeline uses RabbitMQ and Apache Spark. The events labeled as medium severity are ingested

by RabbitMQ and routed to Apache Spark through a durable queue named `medium_severity_disasters`. Messages from the queue are forwarded to a socket, which serves as an intermediate component between RabbitMQ and Apache Spark.

On the consumer side, Apache Spark receives these events from the socket and processes them. Structured Spark Streaming is used, which treats incoming data as DataFrames and processes the events in micro-batches. Incoming events are broken into small, fixed-size batches and are processed at regular time intervals. Micro-batching allows Apache Spark to use checkpointing for enhanced fault tolerance and reliability. However, it introduces a slight delay compared to real-time frameworks like Apache Flink, which processes each event as soon as it is received.

This pipeline provides near-real-time processing but uses fewer computational resources than the high-severity pipeline. This trade-off makes it suitable for handling medium-severity disasters.

3.4 Low severity pipeline

As the name suggests, low-severity disasters do not pose an immediate threat to the well-being of individuals. Hence, they can be processed in batches, reducing the associated computational cost. This study proposes the use of the MapReduce paradigm for communicating notifications about low-severity disasters.

The generated data is ingested into the Hadoop Distributed File System (HDFS), where it is automatically partitioned and replicated. MapReduce is implemented as a single Mapper coupled with three different Reducers. The Mapper takes in low-severity disaster events from HDFS and outputs intermediate key-value pairs. The intermediate key is the disaster type, and the intermediate value consists of the disaster's attributes. Each of the Reducers handles a specific disaster type: flood, fire, or power outage. This allows parallel processing of different disaster types, which optimizes the throughput. This architecture ensures efficient batch processing, distributed computation, fault tolerance, and scalability, allowing the system to handle increasing amounts of data without degradation in performance.

4 Performance Metrics

The two metrics used to evaluate the performance of the three pipelines are latency and throughput. They are defined as follows:

$$Latency = ProcessingTimestamp - EventTimestamp \tag{1}$$

Processing timestamp refers to the system's time when processing for a particular event is completed.Event timestamp refers to the system's time when the event was generated. This is obtained from the timestamp attribute of the event log as seen in Fig. 3. Latency represents the net delay between the occurrence of a disaster and its processing.

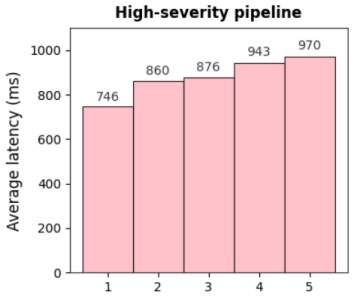
$$Throughput = \frac{Number\ of\ events\ processed}{Elapsed\ time} \tag{2}$$

Number of events processed is the total count of events processed within a stipulated window of time. Elapsed Time is the time that has passed since the last calculation of throughput. This is fixed at 60 seconds in this study. Each pipeline continuously calculates the average latency and throughput over a 60-second window.

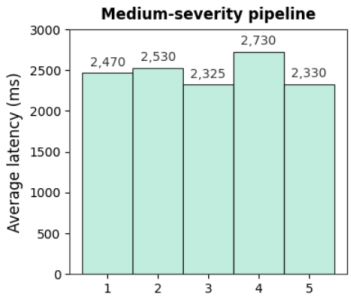
5 Results

5.1 Latency

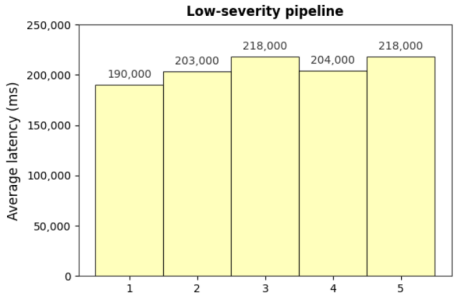
The average latency values for each pipeline are measured over 60-second windows. The results for five such windows are shown in Fig. 4a for the high-severity pipeline, Fig. 4b for the medium-severity pipeline, and Fig. 4c for the low-severity pipeline.



(a) High-severity pipeline



(b) Medium-severity pipeline



(c) Low-severity pipeline

Fig. 4: Severity-based data processing pipelines

The latency values for the high-severity pipeline range from 746 to 970 milliseconds (0.746-0.970 seconds). For the medium-severity pipeline, the values range from 2325 to 2730 milliseconds(2.325-2.730 seconds). The range of latency values for the low-severity pipeline is 190,000 to 218,000 milliseconds(3.17-3.63 minutes). Within each pipeline, there is no significant variation in the average latencies over the 5 windows. This indicates stable performance over time.

The high-severity pipeline has the lowest latency among the three. This can be attributed to Apache Flink's real-time processing framework. Apache Flink uses streaming-first architecture that processes data as it arrives, without grouping it into batches. This eliminates any delays and results in the lowest latency.

The medium-severity pipeline has a higher latency than the high-severity pipeline. This difference can be attributed to the way Apache Spark processes incoming data. In contrast to Apache Flink's event-driven processing model, Apache Spark employs micro-batching. Since,the incoming data is grouped into small-sized batches before processing,latency values are higher.

The low-severity pipeline has the highest latency among the three pipelines. This is because Apache Hadoop uses batch processing instead of stream processing. Apache Hadoop processes data in large batches which makes it unsuitable for applications requiring near real-time data processing. In contrast, Apache Flink and Apache Spark are suitable for real-time and near-real-time processing respectively. Additional overheads are introduced in Hadoop due to disk operations. Apache Hadoop reads large batches of data from the disk and writes intermediate and final results to the disk. Apache Spark utilises in-memory computation which eliminates the overhead due to disk operations.

This study also examines how latency varies across different pipelines as the rate of data generation changes. This study measures the latency for each pipeline at data generation rates of 40 events per minute, 80 events per minute, 120 events per minute, and 160 events per minute. The results for the high and medium-severity pipelines are shown in Fig. 5a, while the results for the low-severity pipeline are shown in Fig. 5b.

When the number of events generated per minute increases from 40 to 80, the percentage increase in latency is 9.23 percent for the medium-severity pipeline and 5.15 percent for the high-severity pipeline, as seen in Table. 3. Similarly, for the increases from 80 to 120 and 120 to 160, the percentage change in latency for the medium-severity pipeline is consistently higher than that of the high-severity pipeline.

Table 3: Percentage increase in latency as number of events per minute increases

Change in events per minute	**Medium-Severity (%)**	**High-Severity (%)**
40–80	9.23	5.15
80–120	17.10	15.29
120–160	8.05	4.25

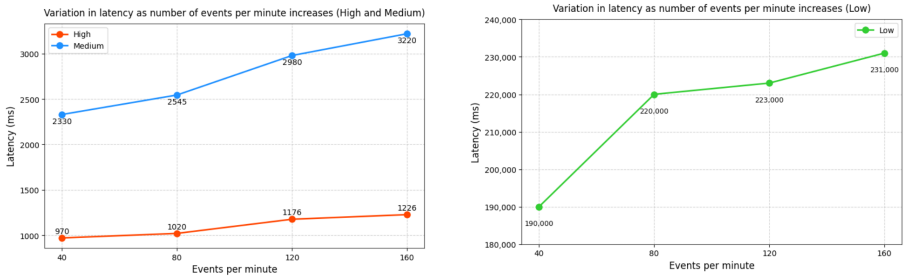
This shows that the high-severity pipeline is robust in handling large volumes of data. The high-severity pipeline fulfils the Quality of Service requirements even when larger amounts of data need to be processed. This is attributed to the fact that Apache Kafka is used as the message broker and Apache Flink is used for stream processing. This combination is optimized for low latency and real-time event processing, resulting in a smaller increase in latency as the amount of data processed increases.

5.2 Throughput

The low-severity pipeline has the highest value for throughput among all three pipelines, as shown in Fig. 6. This is because Apache Hadoop uses batch processing, which skews the throughput metric favourably. Apache Hadoop stores data over multiple nodes. Computations can be performed in parallel over these nodes, allowing more data to be processed at once and thus further inflating throughput.

The high-severity pipeline has a higher throughput compared to the medium-severity pipeline. This can be attributed to the fact that Apache Kafka partitions data based on topics and processes different topics in parallel. In contrast, RabbitMQ does not use partitioning, and all messages pass through a single queue. This could lead to bottlenecks as the amount of data to be processed increases. Hence, the high-severity pipeline can handle higher amounts of data more efficiently than the medium-severity pipeline.

As the number of events per minute increases, the throughput grows more for the high-severity pipeline as compared to the medium-severity pipeline. Thus, among the pipelines that use stream processing, which include the high-severity and medium-severity pipelines, the high-severity pipeline has better throughput. The low-priority pipeline uses batch processing, which allows it to achieve higher throughput than both stream-processing pipelines.



(a) High and medium severity pipelines (b) Low-severity pipeline

Fig. 5: Latency comparison across severity levels

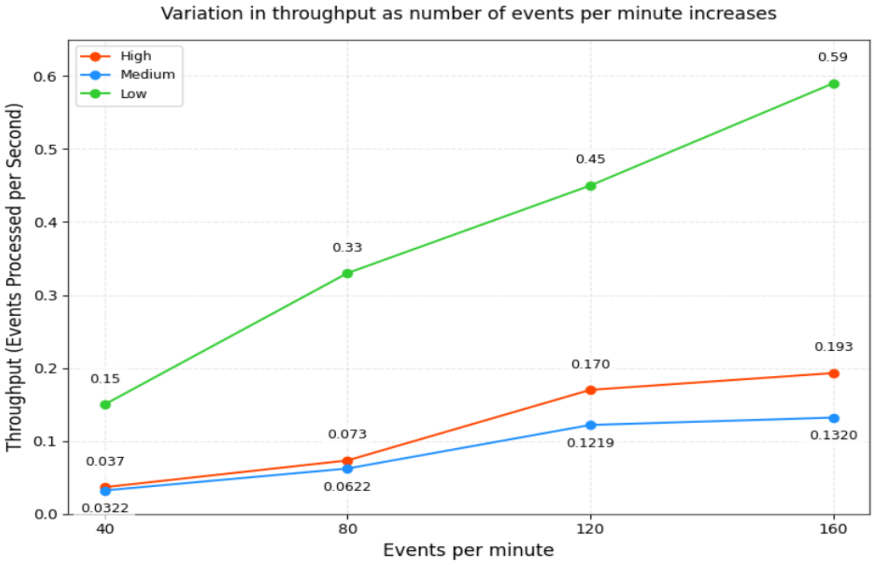


Fig. 6. Variation in throughput as the number of events per minute increases

6 Conclusion

This disaster notification system classifies disaster events into three severities: high, medium, and low, and then routes them to the appropriate pipelines for processing. This ensures that the unique Quality of Service requirements for each severity type are met. The high-severity pipeline, consisting of Apache Kafka and Apache Flink, achieves the lowest latency, making it best suited for severe disasters where an immediate response is required. RabbitMQ and Apache Spark are used for the medium-severity pipeline, which has higher latency than the high-severity pipeline but uses fewer computational resources. The low-severity pipeline employs Apache Hadoop and achieves high throughput at the cost of latency. It is best suited for less critical disasters that can tolerate delay. By tailoring each pipeline to a specific severity level, the system efficiently processes disasters while optimizing resource usage.

7 Acknowledgements

We extend our sincere gratitude to Professor Animesh Giri for his constant guidance and mentorship throughout the course of this research.

References

Akanbi A, Masinde M (2020) A Distributed Stream Processing Middleware Framework for Real-Time Analysis of Heterogeneous Data on Big Data

Platform: Case of Environmental Monitoring. Sensors 20(11):3166. doi: 10.3390/s20113166
Cao L (2024) Real-Time Transmission Monitoring and Alarm Mechanism of Big Data Ocean Observation Files Combined with Apache Kafka. In: Proceedings of the 2024 International Conference on Electronics and Devices, Computational Science (ICEDCS). doi: 10.1109/ICEDCS64328.2024.00145
Dou L, Liu S, Wang X, Wu L (2021) The application of big data real-time stream processing technology in earthquake rapid report. In: Proceedings of the 2021 7th International Conference on Hydraulic and Civil Engineering & Smart Water Conservancy and Intelligent Disaster Reduction Forum (ICHCE & SWIDR), pp 907–912. doi: 10.1109/ICHCESWIDR54323.2021.9656449
Fu G, Zhang Y, Yu G (2021) A Fair Comparison of Message Queuing Systems. IEEE Access 9:421–432. doi: 10.1109/ACCESS.2020.3046503
Giri A, Srinivas M A, Das P (2023) Disaster-Resilient Smart City Framework; A Cross-Layer Protocol Analysis for Emergency Earthquake Response. In: Proceedings of the 2023 IEEE 5th International Conference on Cybernetics, Cognition and Machine Learning Applications (ICCCMLA), Bengaluru, India, pp 413–418. doi: 10.1109/ICCCMLA58983.2023.10346620
Gowda D, Sharma A, Prasad K, Saxena R, Barua T, Mohiuddin K (2024) Dynamic Disaster Management with Real-Time IoT Data Analysis and Response. In: Proceedings of the 2024 International Conference on Automation and Computation (AUTOCOM), pp 142–147. doi: 10.1109/AUTOCOM60220.2024.10486101
Pandey S, Chaudhary M, Tóth Z (2025) An investigation on real-time insights: enhancing process control with IoT-enabled sensor networks. Discover Internet of Things 5:29. doi: 10.1007/s43926-025-00124-6
Sarkar S, Kumar SS, Giri A, Dammur A (2024) Advancing Urban Evacuation Management: A Real-Time, Adaptive Model Leveraging Cloud-Enabled Big Data and IoT Surveillance. In: Proceedings of the 2024 4th International Conference on Intelligent Technologies (CONIT), Karnataka, India, Jun 21–23, 2024. doi: 10.1109/CONIT61985.2024.10626540
Shah SA, Seker DZ, Rathore MM, Hameed S, Yahia SB, Draheim D (2019) Towards Disaster Resilient Smart Cities: Can Internet of Things and Big Data Analytics Be the Game Changers? IEEE Access 7:91885–91900. doi: 10.1109/ACCESS.2019.2928233
Wang Z, Liang H, Yang H, Li M, Cai Y (2025) Integration of Multi-Source Landslide Disaster Data Based on Flink Framework and APSO Load Balancing Task Scheduling. ISPRS International Journal of Geo-Information 14(12):12. doi: 10.3390/ijgi14010012
Zhang Q, Sun S, Yang B, Wüchner R, Pan L, Zhu H (2023) Real-time Structural Health Monitoring System Based on Streaming Data. Sensors. doi: 10.3390/s230xxx