WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Enhancing Cloud-Based Big Data Streaming Applications Using Burst Flow

Namyatha M G[1], Asif Ullah Khan[2]

Assistant Professor, Department of Computer Science Engineering, CIT, Gubbi, Tumkur, Karnataka, India
U.G. Student, Department of Computer Science Engineering, CIT, Gubbi, Tumkur, Karnataka, India

ABSTRACT:

The rapid growth of stream applications in various sectors has posed new challenges for real-time Big Data processing. Traditionally, single cloud infrastructures are used for Stream Processing applications due to their extensive virtual computing resources, but this increases latency due to data sources being distant. Multi-Cloud frameworks deploy services across different clouds, communicating via high latency network links, complicating real-time requirements. Burst flow, introduced in this paper, enhances communication between edge data sources and cloud-based Big Data applications by dynamically adjusting micro-batch sizes based on communication and computation times. It also uses an adaptive data partition policy, considering memory and CPU capacities. Experiments show that burstflow can reduce execution time by up to 77% and improve CPU efficiency by up to 49% compared to state-of-the-art solutions.

INTRODUCTION:

The rapid growth of IoT has introduced significant challenges in Big Data due to massive data generation, requiring effective processing for decision-making. Data processing can be done via Batch Processing (BP) for historical datasets and Stream Processing (SP) for near real-time analysis. IoT devices often produce data at the edge, causing high latency issues when data travels to distant cloud providers, especially for time-sensitive applications like intrusion detection. SP frameworks like Apache Flink and Spark face latency and throughput issues across multiple data centers. While Multi-Cloud infrastructure reduces latency, it doesn't address micro-batch partitioning. BurstFlow addresses these issues by dynamically handling data bursts and distributing input data across SP partitions, improving communication between edge data centers and cloud computing via micro-batches based on message lifetime. It also introduces an adaptive method for stream distribution based on system capacities. Experiments show BurstFlow improves execution time by over 9% and CPU/memory utilization by more than 49%, ensuring efficient processing and supporting critical decision-making in IoT applications. BurstFlow effectively addresses real-time data processing challenges, enhancing the performance and reliability of IoT systems.

RELATED WORKS:

Cloud computing is ideal for large-scale computations, offering security, efficiency, flexibility, and a scalable pay-as-you-go model, making it suitable for data-intensive applications. Near real-time Stream Processing (SP) applications need low-latency processing for continuous data flows from distributed sources, requiring delivery guarantees. SP systems use components like a Message Queue System (MQS) and an SP framework, often across various data centers. Traditionally, frameworks like Apache Spark, Flink, and Heron assume co-located data sources, leading to high latency and resource contention. Jetstream transfers large batches via low-latency routes, while Das et al. propose adaptive batch sizing for stability. Zhang et al. adapt batch intervals to improve Spark Streaming, and Anjos et al. find best performance with larger data blocks. Fernandez et al. introduce Liquid for low latency batch processing using Kafka and Samza. Gulisano et al. improve resource utilization with an adaptive controller, and Zhao et al. enhance in-memory processing with cache management. Xiu and Tang optimize memory allocation for efficiency. BurstFlow handles data bursts, distributing input across SP partitions, optimizing communication between edge data centers and cloud infrastructure.

Techniques for Adaptative Stream Processing

		Infrastructures					Strategies														Proc	
	Author	Geo-Distributed	Cloud Infrastructure	Cluster	Multi-Cloud	Hybrid Infrastructure	Window Size	Time Interval	Throughput Evaluation	Application Evaluation	Query Evaluation	Computation Capacity	Memory Management	Cache Management	Iteration Strategy	Incremental processing	Dynamic Approach	On Demand Approach	Data Movement	Adaptive Approach	Group Strategy	Batch Monitoring	Batch	Streaming
Approaches	JetStream [12]				x				x	x										x		x		x
	Das *et al.* [14]		x				x								x					x		x	x	
	Fernandez *et al.* [22]		x						x							x							x	
	Gulisano *et al.* [25]			x						x	x							x		x				x
	Zhang *et al.* [20]		x					x									x					x		x
	Anjos *et al.* [21]	x				x			x			x							x		x		x	
	Zhao *et al.* [26]		x											x	x		x		x					x
	Xiu *et al.* [27]		x										x											x
	Tang *et al.* [28]		x						x				x				x							x
	BurstFlow	x	x		x		x		x				x				x			x		x	x	x

Cloud computing supports data-intensive applications, but near real-time Stream Processing (SP) faces high latency from distant data travel. Traditional SP frameworks like Apache Spark assume co-located sources, causing resource contention and low throughput. Solutions like Jetstream and Liquid improve batch processing but have limitations. BurstFlow dynamically handles data bursts, distributing input across SP partitions to enhance communication between edge data centers and cloud infrastructure, boosting performance and efficiency.

PROBLEM STATEMENT:

Sensors at the edge produce data bursts for Stream Processing (SP) applications on distant cloud servers, causing high latency and risking data loss. SP applications use Message Queue Systems (MQS) to ensure stable data transmission and retrieve messages via TCP, optimizing performance based on network type (LAN or WAN). Cached buffers speed up retrieval, but individual message processing adds latency in Multi-Cloud (MC) environments. Micro-batching mitigates this by reducing latency, but correct batch size configuration is critical for time-sensitive SP applications. This work introduces a tool for determining batch size policies, considering network conditions and SP framework configurations to improve throughput and reduce latency. It also addresses high processing times due to imbalanced data consumption by multiple SP clients. SP applications, structured as directed graphs with operators performing data transformations, can specify parallelism degrees for operator replicas, but homogeneous cluster methods cause issues in heterogeneous infrastructures.

Burst flow: A Mechanism to Boost Throughput in Multi-Cloud SP Applications

System Overview

Burstflow improves throughput and reduces latency in Stream Processing (SP) applications by managing micro-batches in a Multi-Cloud (MC) setup. Data sources, like sensors, send data in micro-batches to a Message Queue System (MQS) in a micro data center. An SP framework on a different cloud provider consumes these batches. Burst flow's Data Orchestrator determines the optimal micro-batch size and workload distribution dynamically based on execution times, computed from message creation to processing completion.

Burst flow Architecture

Burst flow is designed for geographically distributed MC environments, optimizing micro-batch lifespans and scheduling decisions to reduce execution time. It uses algorithms to set micro-batch sizes and balance workloads across operator replicas. The system includes components like data sources, MQS, and SP frameworks, deployed to capture latencies and adjust micro-batches for optimal processing times. Messages from data sources go to the Data Orchestrator, which runs the ETAMBS (Execution Time-Aware Micro-Batch Strategy) to determine micro-batch sizes. These are then sent to the MQS queue. The SP framework's Master node uses RAPP (Resource-Aware Partitioning Policy) to distribute micro-batches based on resource utilization.

Execution Time-Aware Micro-Batch Strategy (ETAMBS)

ETAMBS dynamically adjusts micro-batch sizes at the data source,considering communication
And computation times. Messages are accumulated in a global buffer with unique ids and timestamps. Once the buffer reaches a specified size, the micro-batch is sent to the MQS.

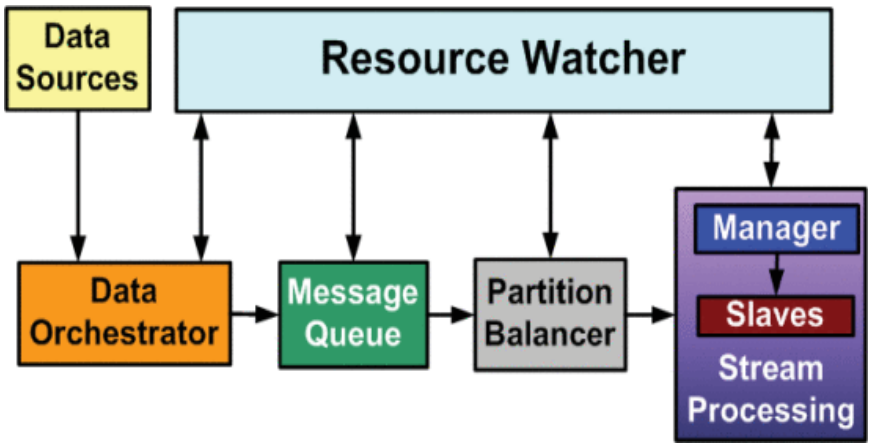

Fig 1. Burst Flow Architecture

Illustrates the architecture, showing message flow from data sources to the SP framework, Detailing Each component's role.In summary, burstflow enhances SP application performance by dynamically managing micro-batches and optimizing resource use across MC environments, ensuring efficient data processing and reduced latency.

Aggregation Algorithm (Algorithm 1)

1. Procedure SEND(msg)
2. If buffer.size=0 then

3. Buffer.setid()
4. Buffer.settimestamp(msg.gettimestamp())
5. Buffer.append(msg)
6. If buffer.size()>=getmaxbuffersize() then
7. Send_micro_batch(buffer)
8. Buffer.clear()

The get max buffer size function starts with aggregating 2 messages and interacts with the Resource Watcher to check execution time. The function adjusts micro-batch size based on the execution rate, aiming for optimal performance. If the variance in execution rate exceeds 30%, the convergence phase reinitializes to detect anomalies.

Resource-Aware Partition Policy (RAPP)

RAPP distributes micro-batches across operator replicas based on the ratio of available memory and CPU. It creates an affinity list to prioritize the most idle machines for micro batch assignment, Updating the list every five seconds. This ensures efficient workload distribution and reduces Processing times by considering system resources dynamically.

Prototype, Experimental Setup, and Performance Evaluation

Burst flow Prototype: The burst flow prototype uses Python for the Data Orchestrator and Apache Kafka for the MQS. Apache Flink with Hadoop processes the data, and Hadoop Distributed File System (HDFS) manages storage. The setup uses Java Development Kit (JDK) 8.181 and JMX for monitoring.
Experimental Setup: Experiments were conducted on Microsoft Azure Cloud's data centers. Data
Sources are 10 A3 instances in Brazil South, producing data via 100 threads each. MQS runs on Apache Kafka in East US, and the SP framework runs on 5 vms in West US with varied configurations. Evaluated with sentiment analysis of tweets, burst flow is compared against the Baseline approach.
Performance Evaluation: Using Jain Methodology, metrics include execution time, throughput, event time, and resource usage.

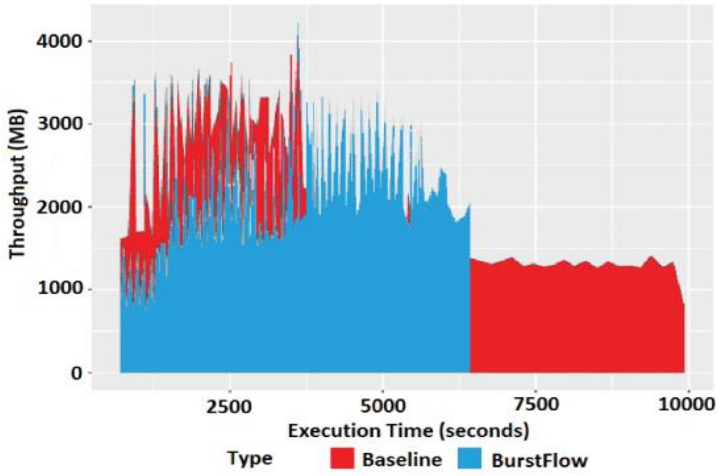

Fig 2. Baseline Vs Burst flow

Burst flow vs. Baseline: burst flow's ETAMBS initially has lower throughput due to message accumulation but outperforms Baseline once micro-batches reach the SP framework, achieving stable throughput of 1.5GB/s.

Throughput, Execution Time, and Resource Consumption Evaluations

Execution Time and Throughput : The evaluation measures throughput, CPU, and memory usage when consuming messages from the MQS and distributing them across operator replicas. Using Apache Flink's default configuration, each core corresponds to one slot, with 24 slots available. The experiments compare state-of-the-art policies in Apache Flink, burst flow's RAPP, and Baseline.

A. Execution Time and Throughput:

Table 1 shows that burst flow's ETAMBS reduces execution time by over 8% by dynamically adjusting micro-batch sizes. Without ETAMBS, each operator consumes one tweet at a time, adding an average of 65 ms due to network latency. Table 5 compares burst flow's RAPP with other solutions, showing it is 37%, 70%, and 77% faster than Baseline, Flink's Random, and Flink's Rebalance, respectively, due to its algorithm for heterogeneous workloads.

Resource Consumption

Resource consumption evaluation identifies bottlenecks impacting performance. Figure 3 shows that burstflow's ETAMBS and RAPP are up to 49% more efficient in CPU usage compared to the one-at-a-time approach, which has higher execution delays and lower throughput. Memory usage mirrors CPU trends; burstflow's RAPP optimizes memory use for stream processing, leading to more homogeneous memory usage and higher CPU utilization. Flink's Broadcast, compared to burst flow, has 32% lower CPU use but 36% higher memory consumption on average. Overall, burst flow's RAPP improves CPU and memory resource utilization, enhancing performance compared to other algorithms, providing better resource management and higher throughput.

Algorithms	w/o ETAMBS	w/ ETAMBS	Gain (%)
BurstFlow's RAPP	6276	5400	16
Baseline	7380	7380	-
Flink's Broadcast	6364	5880	8
Flink's Random	10200	9180	11
Flink's Rebalance	10756	9540	13

Table 1. Execution Time in Seconds With and Without ETAMBS

Algorithms	Execution time (s)	Gain (%)
BurstFlow's RAPP	5400	-
Flink's Broadcast	5880	9
Baseline	7380	37
Flink's Random	9180	70
Flink's Rebalance	9540	77

Table 2. Average Execution Time Duratio

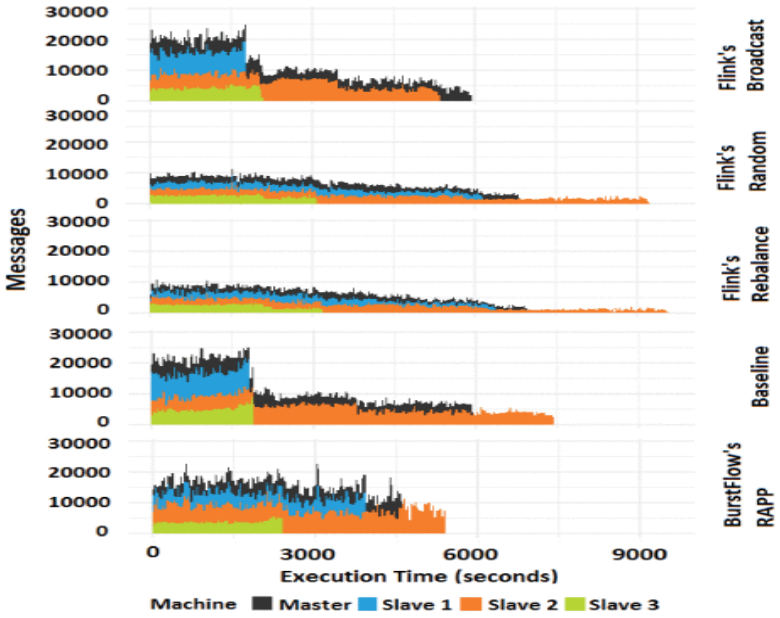

Fig 3. Throughput per node.

Threats to Validity

Experiments were conducted on the Microsoft Azure Cloud Computing Platform, assuming stable machines with high SLA levels and secure environments not exposed directly to the Internet.

Conclusions and Future Work:

This paper introduces BurstFlow, a tool that enhances communication between multiple cloud providers by addressing orchestration issues in cloud-based stream processing frameworks. Unlike Apache Flink, which deploys applications on a single cloud provider, BurstFlow dynamically manages computing resources in a geographically distributed multi-cloud (MC) infrastructure. It improves latency and throughput by adjusting micro-batch sizes

using a feedback loop and controls data distribution with ad-hoc partitioning policies. This flexibility allows BurstFlow to be applied to various scenarios without compromising memory usage or causing context swaps. Evaluated in an MC deployment with a real-world application, BurstFlow reduces execution time by up to 77%, improves CPU and memory usage by up to 49%, and achieves a throughput of approximately 1.5GB/s. Future work includes estimating micro-batch sizes based on stream processing operators and further evaluations using benchmarks from various fields to showcase BurstFlow's benefits.

REFERENCES:

[1] M. Hilbert, Big data for development: A review of promises and chal lenges, Develop. Policy Rev., vol. 34, no. 1, pp. 135174, Jan. 2016, doi: 10.1111/dpr.12142.

[2] N. Miloslavskaya and A. Tolstoy, Application of big data, fast data, and data lake concepts to information security issues, in Proc. 4th Int. Conf. Future Internet Things Cloud Workshops, Apr. 2016, pp. 148153, doi: 10.1109/W-FiCloud.2016.41.

[3] T. Mohammed, A. Albeshri, I. Katib, and R. Mehmood, Ubipriseq Deep reinforcement learning to manage privacy, security, energy, and qos in 5g iot hetnets, Appl. Sci., vol. 10, no. 20, pp. 118, 2020, doi: 10.3390/app10207120.

[4] N. Janbi, I. Katib, A. Albeshri, and R. Mehmood, Distributed arti cial intelligence-as-a-service (daiaas) for smarter ioe and 6g environments, Sensors, vol. 20, pp. 128, May 2020, doi: 10.3390/s20205796.

[5] M. T. Tun, D. E. Nyaung, and M. P. Phyu, Performance evaluation of intrusion detection streaming transactions using apache kafka and spark streaming, in Proc. Int. Conf. Adv. Inf. Technol. (ICAIT), Nov. 2019, pp. 2530, doi: 10.1109/AITC.2019.8920960.

[6] J. Abawajy, Comprehensive analysis of big data variety landscape, Int. J. Parallel, Emergent Distrib. Syst., vol. 30, no. 1, pp. 514, Jan. 2015, doi: 10.1080/17445760.2014.925548.

[7] K. J. Matteussi, B. F. Zanchetta, G. Bertoncello, J. D. D. Dos Santos, J. C. S. Dos Anjos, and C. F. R. Geyer, Analysis and performance evaluation of deep learning on big data, in Proc. IEEE Symp. Com put. Commun. (ISCC), Jun. 2019, pp. 16, doi: 10.1109/ISCC47284.2019. 8969762.

[8] J. C. S. dos Anjos, K. J. Matteussi, P. R. de Souza, A. S. da Veith, G. Fedak, J. L. V. Barbosa, and C. F. R. Geyer, Enabling strategies for big data analytics in hybrid infrastructures, Proc. Int. Conf. High Perform. Comput. Simulation, 2018, pp. 869876, doi: 10.1109/HPCS.2018.00140.

[9] R. Fernandez, P. R. Pietzuch, J. Kreps, N. Narkhede, J. Rao, J. Koshy, D. Lin,C.Riccomini,andG.Wang, Liquid:Unifyingnearlineandof ine big data integration, in Proc. 7th Biennial Conf. Innov. Data Syst. Res., 2015, pp. 18.

[10] (2020). Zookeeper. [Online]. Available: https://zookeeper.apache.org/

[24] Z. Zhuang, T. Feng, Y. Pan, H. Ramachandra, and B. Sridharan, Effec tive multi-stream joining in apache samza framework, in Proc. IEEE Int. Congr. Big Data, Jun. 2016, pp. 267274, doi: 10.1109/BigData Congress.2016.41.