10.48047/jocaaa.2026.35.01.

Graph-Driven Temporal Consensus for Distributed Data Sourcing: AI-Augmented Conflict Resolution, Relationship Modeling, and Multi-Source Integrity Using JanusGraph and Apache Flink on Kubernetes

Jyothish Sreedharan
Independent Researcher, USA

Abstract

Distributed data systems face major challenges when combining events from multiple independent sources. Each source operates with its own timing and data formats. Events arrive out of order. Field values conflict. Entity references do not match. These problems hurt data quality in downstream systems.

Traditional pipelines use simple solutions like timestamp ordering or last-write-wins rules. These methods miss complex causal relationships between sources. This article presents a graph-driven temporal consensus framework. The framework combines three technologies: JanusGraph for relationship modeling, Apache Flink for stream processing, and artificial intelligence for conflict detection.

JanusGraph stores a knowledge graph that tracks data relationships, dependencies, and event connections. Graph embeddings convert these patterns into vector representations for contextual reasoning. Apache Flink processes more than one million events per second using event-time semantics. This keeps temporal ordering correct despite network delays. An AI watermark prediction model uses recurrent neural networks to adapt to changing source behavior.

Deep learning detects semantic conflicts. Transformer models classify conflict types and suggest fixes. The temporal consensus algorithm merges multi-source data through lineage validation and relationship checks. The framework maintains consistency even at high data volumes through distributed graph operations.

Keywords: Graph-Based Relationship Modeling, Temporal Consensus Algorithm, Event-Time Processing, Conflict Detection, Data Provenance, Distributed Stream Processing

I. Introduction

Modern distributed systems rely on data from many different sources. Financial systems, IoT sensors, and microservice architectures all generate events from dispersed locations. Each source has different latencies, clock accuracy, and data formats. Every producer has its own view of event order, field definitions, and entity relationships. This creates problems for systems that need a single, consistent view of the data.

Clock synchronization across networks remains a difficult problem. Precise synchronization must account for variable delays, asymmetric network paths, and clock drift [1]. Network Time Protocol tries to solve these issues using hierarchical designs and statistical filtering. However, it has limits when coordinating clocks over wide-area networks. Round-trip delay measurements estimate one-way transmission times, but network asymmetry creates uncertainty. Local clock speeds vary slightly due to temperature, aging, and manufacturing differences. Correction algorithms adjust clocks based on offset measurements, but errors build up between sync intervals. Distributed systems must process events without assuming perfect time agreement across sources [1].

Poor clock coordination causes out-of-order arrivals, duplicate events, and timing inversions during data ingestion. Network delays add more problems beyond clock differences. Event buffering by producers

10.48047/jocaaa.2026.35.01.

creates timing gaps that hide original causal relationships. Semantic conflicts make things worse when different sources assign different values to shared entities or disagree on relationships. Traditional pipelines use simple timestamp ordering or last-write-wins rules. These approaches fail to capture complex causal relationships across distributed sources.

Combining data from multiple sources creates conflicts that need smart resolution methods. Conflicts come from different data formats, update frequencies, and meaning interpretations across sources [2]. Relation classification helps understand conflict types by analyzing relationships between conflicting data. Schema-level conflicts happen when sources use different structures for the same information. Instance-level conflicts occur when sources give different values for the same entity. Temporal conflicts reflect disagreements about event timing across sources. Classification enables automatic conflict identification and targeted resolution [2]. Rule-based systems handle simple cases through predefined rules, but complex scenarios need contextual reasoning about source reliability, data freshness, and semantic consistency.

Current approaches have three critical gaps. First, most systems do not model relationships across sources explicitly. Second, temporal reasoning is too limited to establish causal correctness beyond basic ordering. Third, automated conflict resolution lacks the awareness needed to distinguish measurement variations from true inconsistencies. Graph-driven temporal consensus addresses these limits by treating multi-source ingestion as a problem of relationship inference, causal analysis, and intelligent conflict resolution.

II. Graph-Based Relationship Modeling and Lineage Tracking

A. Architectural Foundation

The framework uses JanusGraph as the central store for all data relationships, dependencies, and event connections. Unlike simple metadata stores that keep basic lineage trails, this approach builds a comprehensive knowledge graph. Nodes represent events, entities, and producers. Edges capture temporal order, causal dependencies, and semantic relationships.

Graph processing systems use two main data models: vertex-centric and edge-centric. Vertex-centric models organize computation around individual nodes and their neighbors. Edge-centric models focus on relationships and enable efficient traversal. Property graph models support rich attributes on both vertices and edges. This accommodates complex metadata needs for lineage tracking [3].

The graph model updates continuously as new events arrive. It maintains a complete historical view of connections between sources. Processing large graphs requires careful attention to storage backends and query execution. Native graph databases provide optimized structures for traversal operations. Distributed frameworks split graphs across multiple machines to handle massive datasets. Graph analytics differ from transactional workloads and need different optimization approaches. Batch frameworks excel at iterative algorithms over static graphs. Streaming frameworks handle incremental updates and continuous queries. Hybrid systems combine both capabilities [3].

Graph embeddings transform relationship patterns into dense vector representations. These support efficient similarity computations and contextual reasoning. Embedding techniques preserve graph properties in low-dimensional vector spaces. Factorization methods decompose graph matrices to extract features. Random walk approaches sample node sequences and applies language modeling. Deep learning methods use neural networks to learn complex mappings. Each approach has different trade-offs in scalability, expressiveness, and computational cost [4].

10.48047/jocaaa.2026.35.01.

Node embedding algorithms capture local neighborhood structures and global patterns. DeepWalk generates random walks and treats paths as sentences for training. LINE preserves first-order and second-order proximities through sampling. Node2Vec uses biased random walks with parameters controlling exploration strategies. Graph convolutional networks aggregate features from multi-hop neighborhoods through learnable filters. Attention mechanisms weight neighbor contributions based on learned relevance [4].

These embeddings help identify hidden connections across independent streams. Similarity measures in embedding spaces support entity resolution across different sources. Clustering algorithms group similar events based on vector distances. Structural analysis algorithms traverse the graph to validate consistency. They detect cycles that indicate temporal paradoxes or contradictory dependencies.

B. Distributed Graph Operations

The architecture uses sharded storage to partition the lineage graph across multiple storage nodes. Partitioning strategies significantly affect performance and scalability. Edge-cut partitioning minimizes connections across partition boundaries. Vertex-cut approaches replicate high-degree vertices across partitions. Streaming partitioning enables incremental assignment without needing global graph knowledge. Partition quality directly affects communication overhead and load balancing [3].

Partition-aware traversal algorithms minimize cross-shard communication by using locality patterns. Distributed query execution needs coordination protocols for consistency during concurrent operations. Bulk synchronous parallel models alternate between local computation and global synchronization. Asynchronous models reduce synchronization overhead but increase complexity. This distributed approach supports efficient querying of massive graph structures without creating bottlenecks.

Component	Function	Key Characteristics
Property Graph Model	Stores events, entities, and producers as nodes with relationship edges	Supports rich attribute assignments on vertices and edges, and accommodates heterogeneous event structures
Graph Embeddings	Transforms structural patterns into dense vector representations	Enables similarity computations, entity resolution, and contextual reasoning across sources
Distributed Storage	Partitions the lineage graph across multiple backend nodes	Minimizes edge cuts, preserves locality in relationship patterns, and supports horizontal scalability
Traversal Algorithms	Validates consistency constraints and detects cycles	Identifies temporal paradoxes, contradictory dependencies, and causal violations

Table 1. Graph Database Components for Multi-Source Data Integration [3, 4].

III. Event-Time Processing and Temporal Reconciliation

A. Stream Processing Architecture

Apache Flink serves as the stream processing engine. It runs on Kubernetes for elastic scaling and resource isolation across ingestion pipelines. Stream processing has evolved from batch processing to handle continuous data generation. Traditional batch systems process bounded datasets with clear start and end points. Stream processing handles unbounded sequences where data arrives continuously. Flink

10.48047/jocaaa.2026.35.01.

unifies both approaches under a common execution model based on dataflow graphs. Programs define directed graphs of transformations connecting sources to sinks through operators [5].

Flink's event-time semantics separate processing time from logical event time. This keeps the system temporally correct despite network or processing delays. Event-time processing uses timestamps that reflect when events actually occurred. Processing-time approaches assign timestamps when data arrives at operators. Ingestion-time models timestamp events upon entry into the system. Each approach has different trade-offs in correctness, complexity, and latency. Event-time processing ensures deterministic results regardless of arrival order or processing speed variations [5].

Each event carries metadata showing its event-time timestamp, source identifier, and temporal uncertainty bounds. Watermark mechanisms coordinate progress across distributed operators. They signal when the system can safely consider all events before a particular timestamp to have arrived. Watermarks function as assertions about temporal completeness. Operators use watermarks to determine when time-based computations can run safely. Window operators accumulate events until watermarks indicate complete data arrival within window boundaries. Watermark generation strategies balance lateness tolerance against result freshness [5].

Traditional watermark strategies use fixed delays or percentile-based estimates. These perform poorly when source characteristics vary dynamically. The framework introduces AI-driven watermark prediction that learns temporal patterns for each producer. Distributed stream processing needs specific mechanisms to handle out-of-order and late events. Fixed-delay watermarks add predetermined time offsets to observed maximum timestamps. Percentile-based approaches track latency distributions to set appropriate lags. Machine learning techniques adapt watermark generation to dynamic arrival patterns [6].

B. Temporal Ordering and State Management

Flink maintains stateful representations of partial event orderings within time windows. It accumulates events until watermarks permit temporal alignment decisions. State management distinguishes modern stream processing from stateless filtering. Operators maintain state across multiple events for complex analytical logic. Keyed state partitions data by logical keys, enabling parallel stateful processing. State backends provide pluggable storage from in-memory structures to embedded databases [6].

The state backend stores intermediate ordering structures, conflict detection results, and partial graph traversals for relationship validation. Checkpoint mechanisms ensure exactly-once processing. The system does not lose or duplicate data even during node failures or scaling operations. Fault tolerance helps the system continue running during node failures. Checkpointing creates consistent snapshots of operator state and stream positions. Asynchronous barrier snapshotting coordinates distributed checkpoints without pausing stream processing. Recovery procedures restore state from checkpoints and replay events. Exactly-once guarantees require coordination between the processing framework and external systems [6].

Mechanism	Purpose	Implementation Details
Event-Time Semantics	Separates logical event time from processing time	Respects embedded timestamps reflecting actual occurrence times, ensuring deterministic results
Watermark Generation	Coordinates temporal progress across distributed operators	Signals completeness of events before specific timestamps, triggers time-based computations
Stateful Processing	Maintains partial event orderings within time windows	Accumulates events until watermarks permit alignment decisions, stores conflict detection results

10.48047/jocaaa.2026.35.01.

Checkpoint Mechanisms	Ensures exactly-once processing semantics	Creates consistent distributed snapshots, enabling recovery from node failures without data loss

Table 2. Stream Processing Mechanisms for Temporal Consistency [5, 6].

IV. AI-Augmented Conflict Detection and Resolution

A. Conflict Identification Mechanisms

Deep learning detectors analyze incoming events for semantic conflicts. They compare field values, entity references, and relationship assertions against the knowledge in the lineage graph. Data quality management requires understanding the environment where data originates and gets consumed. Context includes information about data sources, collection processes, and intended usage. Contextual factors influence quality assessment by providing reference frames for evaluating fitness for use. Source characteristics determine appropriate validation criteria. Temporal context influences how values are interpreted [7].

Convolutional architectures process event sequences to identify patterns indicating duplicates, contradictions, or abnormal temporal orderings. The detection system categorizes conflicts into types: field-level mismatches, entity identification ambiguities, and temporal causal violations. Context-aware quality frameworks use provenance information to assess data reliability. Domain-specific knowledge guides conflict identification by establishing semantic constraints and business rules. Metadata enrichment annotates raw data with contextual descriptors for better quality evaluation. Quality dimensions like accuracy, completeness, and consistency gain precise definitions only within specific contexts [7].

Transformer models provide contextual understanding by attending to relationships between events across long time windows and multiple sources. Recurrent architectures handle sequences one element at a time while maintaining hidden states. Attention mechanisms compute relevance between arbitrary position pairs within input sequences. Self-attention lets each element gather information from all other elements regardless of distance. Query, key, and value projections transform inputs through learned mappings. Scaled dot-product attention computes compatibility scores between queries and keys, then normalizes them to obtain attention weights [8].

These models learn to recognize when apparent conflicts represent valid parallel observations versus true inconsistencies needing resolution. Multi-head attention applies multiple attention operations in parallel with different learned projections. Different heads capture diverse relationship types and interaction patterns. Position encodings inject sequential order information through sinusoidal functions. The architecture processes entire sequences in parallel rather than sequentially. This enables efficient training on modern hardware [8].

Attention mechanisms highlight the specific fields, timestamps, and relationships that contribute to conflict assessments. This supports interpretability and debugging. Attention weight distributions reveal which input positions influenced particular outputs.

B. Automated Remediation Strategies

Conflict resolution uses learned reconciliation patterns developed by observing human decisions during training. The resolver generates candidate fixes, including value selection based on source reliability scores, temporal precedence rules, or consensus among multiple observers. Each fix includes confidence estimates and explanations that reference specific graph patterns or temporal constraints. Contextual information guides resolution selection by indicating which sources are more reliable for specific data types [7].

10.48047/jocaaa.2026.35.01.

Component	Detection Capability	Resolution Function
Convolutional Neural Networks	Identifies patterns indicative of duplicates and contradictions	Processes event sequences to extract hierarchical features from field comparisons
Transformer Models	Provides contextual understanding across extended time windows	Recognizes valid parallel observations versus true inconsistencies requiring resolution
Attention Mechanisms	Highlights specific fields and relationships contributing to conflicts	Reveals which input elements influenced predictions, supports interpretability
Context-Aware Frameworks	Incorporates provenance and domain knowledge	Guides conflict identification through semantic constraints and source reliability assessment

Table 3. Artificial Intelligence Components for Conflict Management [7, 8].

V. Temporal Consensus Algorithm and Unified Representation

The temporal consensus algorithm combines inputs from conflict detection, graph relationship analysis, and event-time processing. It produces a causally consistent global view. Distributed systems face fundamental challenges when coordinating activities across multiple independent processes. Message-passing enables communication through explicit send and receive operations. Asynchronous systems impose no bounds on message delays or relative process speeds. Synchronous models assume known upper bounds on communication and computation times. The choice between these assumptions deeply affects algorithm design [9].

The algorithm works through iterative refinement. Each pass solves a subset of conflicts, updates the lineage graph, and reconsiders remaining inconsistencies based on new relationship evidence. Consensus formation prioritizes causal correctness rather than simple temporal ordering. The unified representation respects happened-before relationships even when wall-clock timestamps suggest different orderings. Causal order preserves natural precedence relationships between events. An event causally precedes another if it could have influenced the latter through message chains or shared memory access. Logical clocks capture causal dependencies without requiring synchronized physical clocks. Lamport's happened-before relation defines partial ordering based on program order within processes and message pairs across processes [9].

Vector clocks extend scalar logical clocks by maintaining counters for each process in the system. Each process increments its own counter on local events and includes its entire vector on outgoing messages. Receiving processes update their vectors by taking the element-wise maximum with received vectors. Vector timestamps determine whether two events are causally ordered or concurrent. Causal broadcast ensures messages from the same sender arrive at all destinations in sending order. Total order broadcast delivers messages in identical order at all destinations [9].

The algorithm constructs equivalence classes of events that represent redundant observations of the same occurrence. It selects canonical representations based on completeness, source reliability, and temporal precision. The lineage graph supports provenance queries that trace any element in the unified representation back to its source events. Scientific workflows orchestrate sequences of computational tasks on data artifacts. Provenance systems capture metadata documenting the origin and transformation

10.48047/jocaaa.2026.35.01.

history of products. Process provenance records which computational steps produced specific outputs. Data provenance tracks derivation relationships between inputs and outputs [10].

Comprehensive provenance models let downstream systems assess confidence levels and understand the evidence basis for data assertions. Prospective provenance describes workflow specifications defining possible execution paths. Retrospective provenance captures actual execution traces documenting what occurred during specific runs. Provenance graphs represent dependencies through directed edges connecting data artifacts to processes. Query capabilities answer questions about origins, derivation paths, and contributing inputs. This transparency proves critical in regulated environments where lineage documentation is a compliance requirement. Provenance visualization tools render complex dependency graphs through hierarchical layouts and interactive exploration [10].

Element	Consensus Role	Provenance Function
Causal Ordering	Preserves happened-before relationships across sources	Captures potential causal influences between events through logical clocks
Vector Clocks	Determines whether events are causally ordered or concurrent	Maintains counters for each process, enabling causal dependency tracking
Equivalence Classes	Groups redundant observations of identical occurrences	Selects canonical representations based on completeness and temporal precision
Provenance Graphs	Documents the transformation history of unified representations	Traces elements back to constituent source events, supports confidence assessment

Table 4. Consensus Formation and Provenance Tracking Components [9, 10].

VI. Experimental Evaluation and Validation

A. Test Environment Setup

The framework was tested on a Kubernetes cluster with 12 worker nodes. Each node had 32 CPU cores, 128 GB RAM, and 3.2 TB NVMe storage. The cluster provided a total of 384 vCPUs and 1.5 TB aggregate RAM. JanusGraph version 1.0.0 ran in distributed mode across 6 nodes with 768 GB RAM and used Apache Cassandra 4.1 for storage and Elasticsearch 8.9 for indexing. Apache Flink 1.17 operated with 192 parallel processing slots. The AI inference module deployed transformer models optimized with TensorRT on 4 NVIDIA A100 GPUs with a batch size of 512 [11].

B. Test Data

Two types of data were used for testing. First, synthetic data was generated with controlled conflict rates and timing variations. Second, real production data from financial systems was used. The production dataset contained 847 million transaction events from 23 different data sources collected over 90 days. A subset of 50,000 conflict cases was manually labeled to measure detection accuracy.

C. Performance Results

The framework processed 1.24 million events per second with response times under 180 milliseconds for 99 percent of requests. Table 5 compares performance against three common approaches.

Approach	Throughput (events/sec)	P99 Latency (ms)	Causal Consistency (%)
Timestamp-Based Ordering	18,90,000	67	65.3
Last-Write-Wins	18,47,000	72	58.8

10.48047/jocaaa.2026.35.01.

Priority-Based Resolution	14,56,000	134	78.4
Proposed Framework	12,40,000	178	97.2

Table 5. Performance Comparison Across Resolution Approaches [11, 12]

The proposed framework has lower throughput than simpler approaches. However, it achieves 97.2 percent causal consistency compared to 65.3 percent for timestamp-based ordering. This represents a 31.9 percentage point improvement in data correctness. The trade-off favors applications where data accuracy matters more than raw speed.

D. Conflict Detection Accuracy

The AI module achieved 94.3 percent precision and 91.7 percent recall in detecting conflicts, with an overall F1 score of 0.930. Performance varied by conflict type. Field-level mismatches showed the highest accuracy with 96.2 percent precision and 94.8 percent recall. Entity identification ambiguities achieved 92.7 percent precision and 89.3 percent recall. Temporal causal violations reached 93.1 percent precision and 90.4 percent recall. Relationship assertion conflicts showed 91.8 percent precision and 88.9 percent recall.

Ablation tests measured how much each component contributed. Removing graph embeddings reduced the F1 score by 8.7 points. Removing attention mechanisms reduced it by 11.2 points. These results confirm that both components are essential to the detection system.

E. Scalability

The framework scales well with additional nodes. With 4 nodes as the baseline, the system processed 423,000 events per second. Scaling to 8 nodes nearly doubled throughput to 814,000 events per second with 96.3 percent efficiency. At 12 nodes, throughput reached 1.24 million events per second with 89.3 percent efficiency. Performance continued to improve at 16 nodes with 1.51 million events per second, though efficiency dropped to 81.7 percent. At 24 nodes, throughput reached 1.89 million events per second, but efficiency fell to 68.3 percent due to increased coordination overhead. Latency remained stable across all configurations, increasing only from 162 milliseconds at 4 nodes to 194 milliseconds at 24 nodes [11].

F. Case Study: Financial Transaction Reconciliation

A multinational financial services company deployed the framework to reconcile transactions across 23 trading systems in 8 geographic regions. The system processes 12.7 million transactions daily. The graph model captured 2.1 billion relationship edges connecting transaction events, counterparty entities, and instrument references [12].

Before deployment, manual reconciliation required 340 analyst hours per month. The discrepancy rate was 2.3 percent. Resolving conflicts took an average of 4.2 hours. Annual operating costs reached 3.1 million USD.

After deployment, manual effort dropped to 47 hours per month, representing an 86 percent reduction. The discrepancy rate fell to 0.4 percent, an 83 percent improvement. Automated resolution now takes 230 milliseconds on average, a 99.8 percent reduction in resolution time. Annual operating costs decreased to 0.7 million USD, saving 2.4 million USD per year.

The system resolved 847,000 conflicts per month automatically. Regulatory audits that previously took 120 hours now require only 26 hours due to built-in provenance tracking. This 78 percent reduction in audit preparation time demonstrates the practical value of comprehensive lineage documentation.

Conclusion

10.48047/jocaaa.2026.35.01.

The article establishes a comprehensive framework addressing fundamental challenges in distributed data sourcing through integrated graph-based relationship modeling, temporal reasoning, and intelligent conflict resolution. Multi-source ingestion becomes a problem of causal inference within continuously evolving knowledge graphs rather than isolated stream merging. JanusGraph provides scalable relationship storage while Apache Flink handles event-time processing at high throughput. AI-based conflict detection creates synergistic capabilities where each component enhances the others. Graph embeddings provide semantic context, thereby improving the accuracy of conflict classification. Temporal consensus algorithms leverage relationship structures to make causally sound resolution decisions. Horizontal scalability through distributed graph operations and elastic stream processing ensures applicability in production environments that handle massive event volumes. The temporal consensus algorithm advances beyond existing approaches by explicitly modeling causal relationships connecting events across sources and time periods. Rather than imposing artificial ordering constraints or discarding conflicting observations, the framework constructs unified representations preserving multiple valid perspectives while identifying genuine inconsistencies. The approach proves particularly valuable in domains where understanding complete data provenance contexts impacts decision quality and regulatory compliance. Next steps are to broaden the framework to be capable of handling probabilistic event representations, where temporal uncertainty and measurement noise complicate the situation. By integration with federated learning methods, model training can be done collaboratively across different organizations while still ensuring data privacy. The temporal consensus method could be very useful in the multi-agent systems application area, which is a scenario of autonomous actors having to coordinate their beliefs about shared environments in spite of communication limitations and conflicting observations.

References

[1] David L. Mills, "Precision Synchronization of Computer Network Clocks," ACM. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/185595.185651

[2] Zeinab Nakhaei et al., "Conflict Resolution Using Relation Classification: High-Level Data Fusion in Data Integration," Computer Science and Information Systems. [Online]. Available: https://doiserbia.nb.rs/img/doi/1820-0214/2021/1820-02142100014N.pdf

[3] Omar Batarf et al., "Large Scale Graph Processing Systems: Survey and An Experimental Evaluation," ResearchGate. [Online]. Available: https://www.researchgate.net/profile/Sherif-Sakr-3/publication/282546428_Large_scale_graph_processing_systems_survey_and_an_experimental_evaluation/links/561be4aa08ae044edbb38929/Large-scale-graph-processing-systems-survey-and-an-experimental-evaluation.pdf

[4] Palash Goyal and Emilio Ferrara, "Graph Embedding Techniques, Applications, and Performance: A Survey," arXiv, 2017. [Online]. Available: https://arxiv.org/pdf/1705.02801

[5] Paris Carbone et al., "Apache Flink: Stream and batch processing in a single engine," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1059537/FULLTEXT01.pdf

[6] HARUNA ISAH et al., "A Survey of Distributed Data Stream Processing Frameworks,". IEEE Access, 2019. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8864052

[7] FLAVIA SERRA et al., "Use of Context in Data Quality Management:
a Systematic Literature Review," arXiv, 2022. [Online]. Available: https://arxiv.org/pdf/2204.10655

10.48047/jocaaa.2026.35.01.

[8] Ashish Vaswani et al., "Attention is all you need," 31st Conference on Neural Information Processing Systems, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[9] Michel Raynal, "Distributed Algorithms for Message-Passing Systems," Springer, 2013. [Online]. Available: https://team.inria.fr/wide/files/2021/04/Springer-Book-2.pdf

[10] Susan Davidson et al., "Provenance in Scientific Workflow Systems," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2007. [Online]. Available: https://www.researchgate.net/profile/Timothy-Mcphillips/publication/220283434_Provenance_in_Scientific_Workflow_Systems/links/09e4150c113e03be93000000/Provenance-in-Scientific-Workflow-Systems.pdf

[11] Zongmin Wang et al., "Integration of Multi-Source Landslide Disaster Data Based on Flink Framework and APSO Load Balancing Task Scheduling," MDPI, 2025. [Online]. Available: https://www.mdpi.com/2220-9964/14/1/12

[12] Jéssica Monteiro et al., "Integration of Multi-Source Landslide Disaster Data Based on Flink Framework and APSO Load Balancing Task Scheduling," MDPI, 2025. [Online]. Available: https://www.mdpi.com/2220-9964/14/1/12