International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

Distributed Data Engineering Frameworks for Real-Time Healthcare Record Integration

Dileep Valiki

Independent Researcher, India

ABSTRACT: Real-time distributed data engineering enables the concurrent ingestion, processing, and provisioning of records from multiple sources throughout both the internal and external distributed systems of healthcare organizations. A widely dispersed set of assets is harnessed: Electronic Health Records (EHRs) must be translated into a common set of clinical concept identifiers; external systems providing timely information such as medication prescribing and dispense events, clinical pathology results, and respiratory disease hazard alerts must be interfaced; and continually streaming internal sensor data are made available for clinical decision support.

Central to real-time data integration in data engineering are the principles of consistency, time versus throughput balance, and fault tolerance within a regulated domain. The low-latency ingestion and streaming processing patterns that underpin these solutions are described in terms of their event-based schema, on-the-fly schema evolution capabilities, and the inherent need for backpressure. Particularly for Electronic Health Records (EHRs), the key model elements are defined in terms of patients and their medical contacts with the institution, the observable clinical facts collected over time, and the medications associated with those facts, with a strong focus on recognition and value assignment through standardized terminologies such as SNOMED CT.

KEYWORDS: Real-Time Distributed Data Engineering, Healthcare Data Streaming Architectures, Electronic Health Records (EHR) Integration, Event-Driven Clinical Data Processing, Low-Latency Healthcare Ingestion Pipelines, Clinical Concept Standardization, SNOMED CT Terminology Mapping, Schema Evolution in Healthcare Systems, Backpressure in Streaming Systems, Fault-Tolerant Health Data Platforms, Time–Throughput Optimization, Clinical Sensor Data Integration, Interoperable Health Information Exchange, Regulated Healthcare Data Governance, Real-Time Clinical Decision Support Infrastructure.

I. INTRODUCTION

Real-time distributed data engineering has the potential to enable dynamic fusion and integration of multiple healthcare data sources and formats to assist clinical, epidemiological, and healthcare administrative operations. Timely and trustworthy delivery of information from heterogeneous systems and sources is pursued in many healthcare institutions for advanced software tools, clinical decision support services, and safety monitoring applications. These transformative capabilities have triggered development of experimental health information streaming systems that capture, create, and implement a large set of timely clinical rules. The examination of these and other emerging applications is a frenetic area of investigation in academic research.

The ideas of distributed data engineering and event-driven architecture describe both the general concept of accomplishing real-time data engineering and design principles employed in healthcare-oriented adaptation of these foundational ideas. The primary motivating aspect in this context is the need to govern and mechanize reliable content delivery chain to a diverse spectrum of heterogeneous organizational information consumers in a timely manner. Mechanization of mail delivery in current postal systems serves as a highly effective analogy for this kind of engineering and management. However, the general principle of achieving and assuring reliably large-scale service with strict delivery time mitudsmanpons through various parties that share yet never store federated content is a technical and social goal with very deep intuition and a wealth of supporting experience in shorter-scoped communities such as sailing and aviation.

1.1. Overview of Distributed Data Engineering in Healthcare

The ability to fuse data from disparate sources in a time-sensitive manner lies at the nexus of distributed data engineering technologies. Timeliness is critical for many applications in healthcare, including those supporting patient safety, infectious disease surveillance, sepsis detection, and epidemiological analysis. Modular data pipelines provide an implementation framework for real-time data fusion applications, organizing operations into decoupled components

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

that can be independently developed, deployed, governed, and scaled. Health records make up a substantial portion of clinical data, with data-sharing agreements in place among parties involved in patient care. Requirements vary depending on context, but an accurate, up-to-date patient record can greatly improve the quality of care. The ability to automatically ingest real-time updates and propagate changes among systems further enhances interoperability.

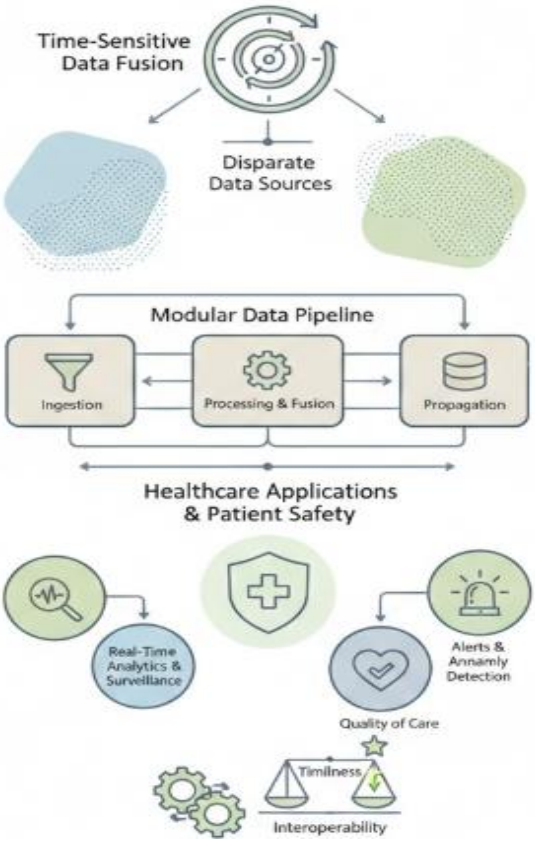


Fig 1: Modular Data Pipelines for Time-Sensitive Healthcare: A Framework for Real-Time Fusion and Patient Safety Surveillance

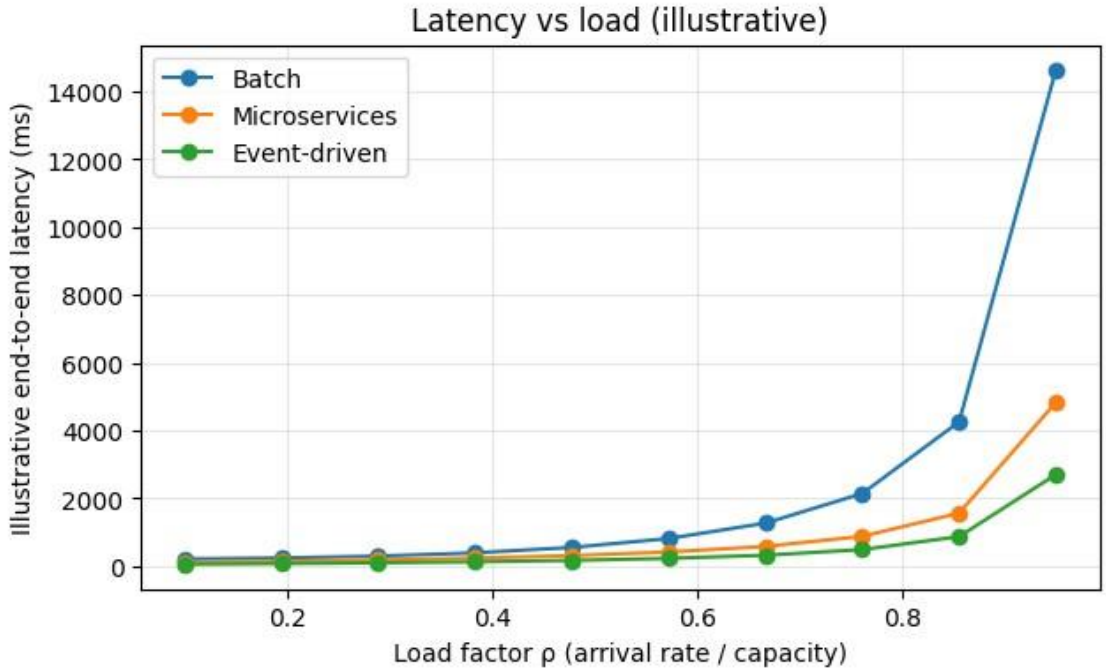
Timeliness requirements apply not only to patient records, but also to alerts and rules derived from data in motion. Detecting anomalies in real time can help avert patient safety incidents due to drug overdoses, allergic reactions, or a host of preventable causes. Timeliness joins correctness, even in the face of possible conflicting information, as an essential property of distributed systems that deal with real-time streams of clinical data. Timely is a pragmatic classification applied to many applications that can tolerate some delay, but for which lower latency is preferred. Stronger guarantees can be provided in carefully circumscribed domains, where freshness is paramount and rules are simple.

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002



II. FOUNDATIONS OF DISTRIBUTED DATA ENGINEERING FOR HEALTHCARE

Data consistency, the trade-off between latency and throughput, and fault tolerance in regulated environments form key foundations of real-time distributed data engineering for health data integration. All data engineering entails some trade-off between the latency of individual operations and the throughput achievable under load. In the case of real-time data engineering, there has long been a belief that heuristic choices—eventual consistency rather than formal guarantees—could mitigate the impact of full consistency checks on latency while offering acceptable reliability most of the time Santoro and Muccini3. However, the extensive regulation of healthcare poses significant challenges to the pragmatic adoption of soft-state consistency models. Where especially sensitive data is at play, or for other reasons data persistence policies require strict transactional handling, a focus on batch processing operations with consequently larger latencies may be warranted. In addition, workload-specific burn-in periods defining the service-level agreements become even more important when a timing-dependent function such as a decision-support rule has to be reevaluated after every data update.

Large-scale distributed fault tolerance is therefore indispensable in systems that must remain online so that health and safety assurance remains intact. Distributed processing frameworks based on dedicated service assemblies or external hosting spheres share many features but also expose specific requirements and bottlenecks in health data streams. In particular, automatic schema evolution and backpressure handling can be key needs in the medical domain. The subtleties of distributed ingestion, be it of historical imports into presentation instances or of hot fresh data from clinical activities, deserve attention. Event-driven streaming systems, functionally similar to pub–sub models, marry well with the technical demands of low-latency processing and fulfilment of service-level agreements. In addition, by enabling careful decoupling of the architecture's components and an incremental horizontal scaling facility, they mitigate the complexëity and operational costs associated with microservice pipelines.

Equation 1) Core definitions → equations (step-by-step)

1.1 End-to-end latency

Step-by-step derivation

1. Let an event occur at timestamp t_{event}.
2. Let the result (alert, updated record, decision-support output) be delivered at t_{deliver}.
3. **End-to-end latency** is the difference:

$$L = t_{\text{deliver}} - t_{\text{event}}$$

4. If the pipeline has stages (ingest → broker → process → sink), and stage i contributes time L_i, then:

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

$$L = \sum_{i=1}^{k} L_i$$

1.2 Throughput

Step-by-step derivation

4. Let N = number of completed events in a time window of length Δt.
5. Then throughput is:

$$X = \frac{N}{\Delta t}$$

3. If you observe multiple windows, average throughput over total time T with total completed events N_T is:

$$\bar{X} = \frac{N_T}{T}$$

1.3 Latency–throughput relationship (Little's Law)

A foundational relationship is **Little's Law** (queueing identity), linking:

- W: average time in system (≈ latency),
- λ: arrival rate (events/s),
- N: average number of in-flight items (WIP, "work in process").

Step-by-step

1. Let λ be the average arrival rate (events/s).
2. Let W be average time an event spends in the system (s).
3. Let N be average number of events concurrently in the system.
4. Then:

$$N = \lambda W$$

2.1. Real-Time Data Ingestion and Streaming

Real-time distributed data engineering frameworks ingest information in motion and provide it in motion for immediate processing and publishing. These systems result in fast-moving data streams subject to backpressure, incremental processing using event-driven microservices, and high-throughput immediate processing using big-data paradigms.

Real-time data ingestion is concerned with streams of input data from sources such as electronic health records (EHRs), streaming feeds, social media, and clinical systems within external organizations. The real-time requirement rests on the impact of data latency on preventive care. The risk associated with a particular patient event is based not only on the patient's situation at that moment but also on the situation within the relevant patient cohort. That cohort is comprised of the patient and other temporally close patients with similar medical conditions. Therefore, the cohort situation must be considered within a sufficiently fine time granularity. The data sources generate patient record information internally through normal operations. Organizations that have access to these records for their users provide updates to other organizations with external users. Further patient records can be created and subsequently requested by other organizations, again from sources such as Facebook, Twitter, and news feeds. Although the availability of social media data and news alerts may not facilitate preventive care directly, it may provide indirect information that is still valuable to the users.

Backpressure is the classic approach that limits the rate of incoming data by resource capacity. The aim is not to remove message-processing bottlenecks. Healthy backpressure dynamics arise under normal operational levels where incoming data volumes are well within bounds.

2.2. Data Modeling for Electronic Health Records

Reliable and maintainable integration at both record and event levels requires careful design of data models around core data types used in electronic health records, supported by proven industry-wide standards for coding and terminology. Four core data types are commonly found (or closely related in specializations): patients, encounters, observations, and medications. Data-modeling work starts by defining these, while positioning patient as the primary structural entity.

Focusing on patients first, a standard-profiling approach maps Electronic healthcare record system data dictionaries onto an interoperable model by bridging local, state, and other repositories. Patient metadata is expressed in an interoperable event schema using a slightly extended version of the FAST data-integration profile, built on the FHIR R4 Resource Patient. Several property groups — identifying information, gender and ethnicity, address, contact details, and insurance — are handled in accordance with recommended guidelines, the Data Übermenmodell (DUM) that maps best-practice design with SQL build compliance, and the Rupa profile. Sources cited include software-engineering

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

textbooks, general-government data-consent frameworks, British and Australian standards, the World Health Organization, and the World Wide Web Consortium.

Load factor ρ (arrival/capacity)	Latency - Batch (ms)	Latency - Microservices (ms)	Latency - Event-driven (ms)
0.1	208.9	107.8	55.6
0.19	237.5	140.3	73.8
0.29	293.9	181.6	96.9
0.38	390.6	235.4	127.0
0.48	549.7	308.7	168.1
0.57	812.4	414.4	227.3

III. ARCHITECTURAL PARADIGMS FOR REAL-TIME INTEGRATION

Real-time architecture choices affect integration modalities. Simple data streams flowing to target systems without special handling are suitable for many use cases, but applications sometimes require more complex logic. Consequently, specialized paradigms have emerged that cater to those needs. Such paradigms can be grouped into two high-level classes: microservices and service-oriented architecture for modularizing deployment; and event-driven architecture and streaming platforms for low-latency data processing. Although they can be applied individually, they work exceptionally well in conjunction.

Microservices and service-oriented architecture support the modularization of applications into autonomous services. A data-streaming engine such as Apache Kafka can act as the shared external communication medium for the services supporting a distributed architecture while ensuring operational governance with security and privacy. Microservices support the independent deployment of services, which allows specialized scaling. Health-care information systems are often subjected to periodic audits, and microservices ease operational governance by allowing compliance management in smaller, independent deployments. Distributed data pipelines can thus leverage the advantages of microservices and service-oriented architectural design while limiting operational governance complexity.

Equation 2) Backpressure → queue growth equations

2.1 Minimal deterministic model

Let:

- λ = incoming event rate (events/s),
- μ = processing capacity (events/s),
- $Q(t)$ = queue length (events).

Step-by-step

5. In a small time step Δt, arrivals add $\lambda \Delta t$ events.
6. Processing removes up to $\mu \Delta t$ events (cannot remove more than you have, but assume backlog exists).
7. Net change:

$$\Delta Q \approx (\lambda - \mu)\Delta t$$

5. Take the continuous limit:

$$\frac{dQ}{dt} = \lambda - \mu$$

5. Integrate (assuming constant λ, μ):

$$Q(t) = Q(0) + (\lambda - \mu)t$$

6. Key consequence:

- If $\lambda > \mu$: $Q(t)$ grows linearly → latency explodes.
- If $\lambda \leq \mu$: queue does not grow unbounded.

2.2 Backpressure condition

Backpressure is essentially enforcing an **effective** arrival rate λ_{eff} such that:

$$\lambda_{\text{eff}} \leq \mu$$

3.1. Microservices and Service-Oriented Architectures

Microservices and service-oriented architectures facilitate modular development, deployment, and governance of healthcare data pipelines. Service-oriented architecture (SOA) relies on networked, loosely coupled, and interoperable

International Journal of Science, Research and Technology (IJSRAT)
|Bimonthly | Peer-Reviewed | Scholarly Journal|
|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|
DOI: 10.15662/IJSRAT.2020.0306002

functional units. Its fundamental principles are extensive reuse and sharing of stable, well-defined APIs. Semantic description of services establishes a common understanding across involved parties.

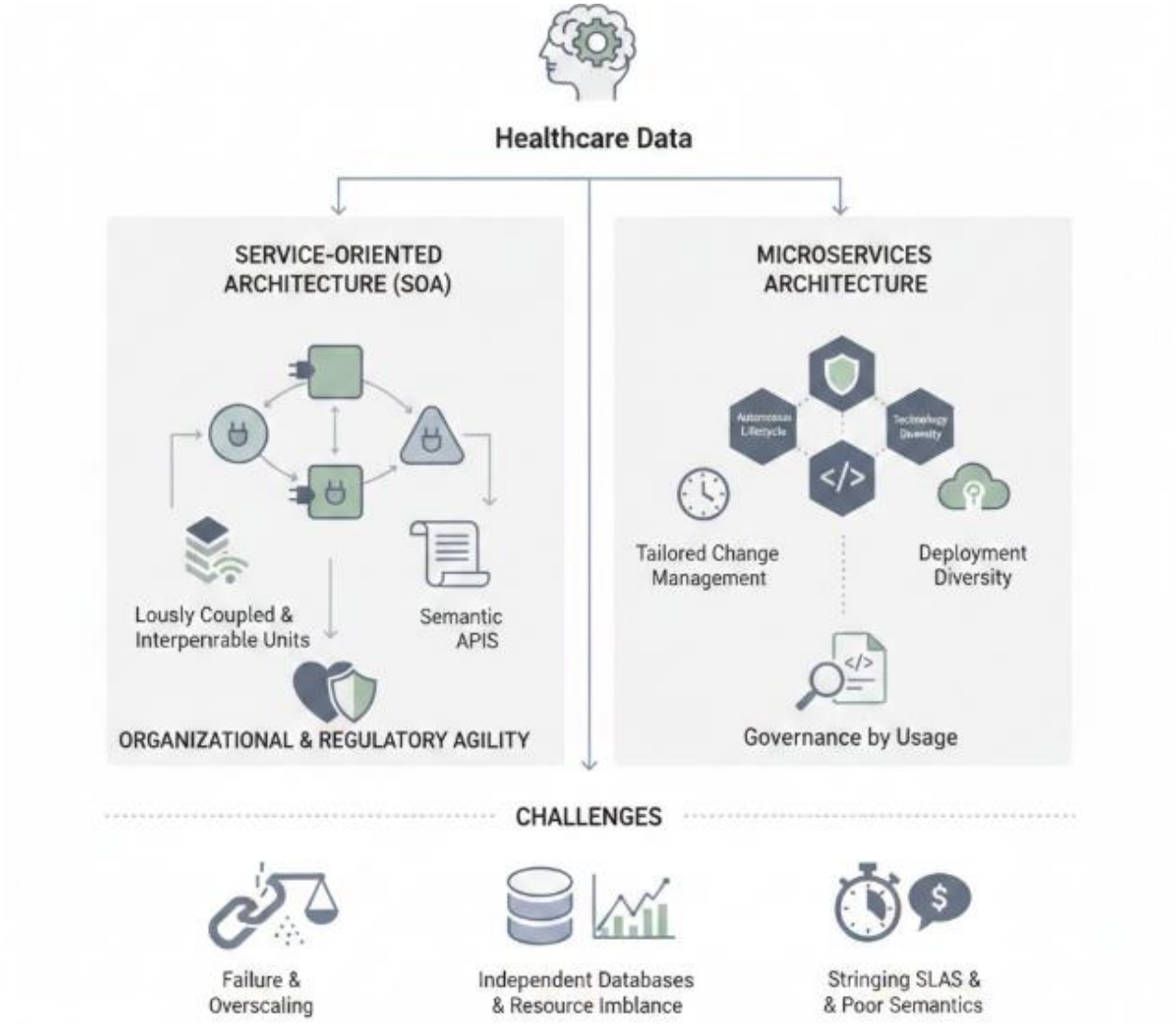


Fig 2: Decoupling Governance: The Microservice Architecture Paradox in Healthcare Data Pipelines

Microservices represent an SOA characterized by coarse-grained services and a single purpose. Granularity and independence enhance organizational and regulatory agility by enabling autonomous life-cycle management, allowing smaller teams to take full responsibility for specific services. Change management policies can also be tailored to risk, permitting a rapid-release testbed for urgent updates. Architectural independence of services supports technology and deployment diversity—for example, service coding in different languages, and testbed support of alternative databases. Thus, operational overhead is reduced, and venues for governance can be grounded in actual usage patterns. Services tend to demand a long-term commitment from consumers because providing an exact substitute is costly.

Microservices warrant careful design because they facilitate failure and overscaling. Independent database instances increase complexity, and wide-varying resource utilization is difficult to balance. Moreover, a stringent SLAs impose governance costs that interrupt local innovation; overhead becomes significant if the service is used only occasionally; and poorly documented semantic assumptions can lead to failures.

3.2. Event-Driven Architectures and Streaming Platforms

For complex and latency-sensitive applications that require data processing within small time windows, decoupled components that communicate through asynchronous events enable new online monitoring or decision-support features

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

and provide opportunities for horizontal scaling. Continuous distributed processing is often implemented based on platforms such as Apache Kafka or Apache Flink. These frameworks can also support more relaxed applications with larger time windows, employing techniques such as stateful processing and event time semantics. Event-time processing caters for out-of-order events based on external timestamps, while stateful stream processing handles maintaining, updating, and querying internal state volumes. By internally freeing application developers from the details of distributed processing, these platforms make it feasible to accommodate complex applications behind a single topic or event-stream schema.

Information from other electronic health record (EHR) systems or external clinical systems can also be made available in real-time, allowing incoming patients with recent admissions elsewhere to be considered simultaneously with ongoing patients. A common simulated-live feed integrated using these techniques is a continuous stream of patients arriving at the hospital by ambulance. A monitoring service simulates ambulances calling in with details of real-time events, and the patient candlestick visualisation indicates which services are available, along with the duration each service was busy. Events triggering alerts for safety, such as an admitted stroke patient arriving at a hospital where all stroke physicians are busy for a prolonged period, and alerts necessary for decisions, such as a dedicated child anaesthetist being busy, are also integrated.

IV. DATA QUALITY, GOVERNANCE, AND SECURITY IN REAL-TIME STREAMS

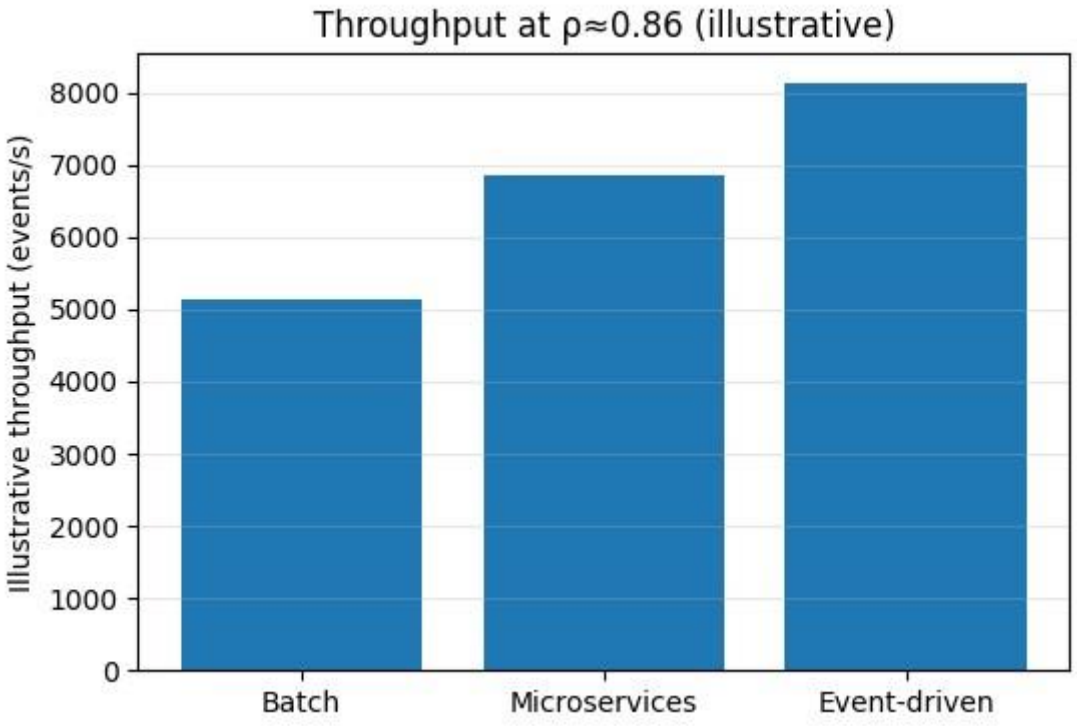
Reliable, trustworthy data are paramount for operational, clinical, and research purposes. Regulatory measures govern data quality in clinical record repositories; for external contributors such as data marketplaces or EHR vendors, compliance with standards, protocols, and testing enables third-party certification. Timeliness and adherence to service-level agreements are critical for important operations such as alert delivery.

When integrating data from distributed flows, it is essential to verify its quality and fitness prior to using or considering it authoritative. Anomalies may arise from data provenance, application errors, originating system malfunctions, or malfunctioning ports/boxes. Validation and cleansing steps can be executed on pipelines before writing to sinks. Overlapping sink contents often necessitate conflict resolution. Various nodes must either share a common configuration or synchronize periodically to avoid accumulating disparity.

Governance of real-time streaming data is more challenging because quality concerns and data traces are much less controlled. Yet, quality and fitness remain as important as in batch architectures. Copying data from its operational source may incur multiple copies in sinks, and appropriately timed deduplication can remedy this. Backpressure mechanisms, monitoring tool alerts, and streaming-data testing help identify preventive actions.

Given the open nature of stream data, provenance and lineage are necessary for safety monitoring. Provenance captures the origin, context, and history of data items to assess integrity and credibility, while lineage denotes the sequence and transformation of a data item throughout the workflow. These paths are important for performing judicious analysis, communicating insight reliability, and course correcting when multiple streams collide. Auditing tracks awareness of every workflow run and its stages. Tamper-evident data structures also augment advertising responsibility and ensure reliable source tracing across the network.

International Journal of Science, Research and Technology (IJSRAT)
|Bimonthly | Peer-Reviewed | Scholarly Journal|
|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|
DOI: 10.15662/IJSRAT.2020.0306002



Equation 3) SLA and burn-in → measurable formulas

3.1 SLA as a probabilistic latency guarantee

A practical SLA form is:

$$\Pr(L \le L_{\text{SLA}}) \ge p$$

Example: "95% of events must be delivered within 2 seconds".

3.2 Burn-in and steady-state estimation

Let observed latencies during burn-in be $L_1, L_2, \ldots, L_n$.

Step-by-step

8. Running mean latency:

$$\bar{L}_n = \frac{1}{n}\sum_{i=1}^{n} L_i$$

2. Running throughput over time window T:

$$\bar{X} = \frac{N_T}{T}$$

4.1. Data Quality Assurance Across Distributed Data Pipelines

Data quality plays an indispensable role in the reliability and usability of real-time integrated healthcare data. If patients and external clinical systems depend on up-to-date electronic health record (EHR) information, then any data delivered through pipelines must not only be timely but also valid, trustworthy, and actionable. Whether deployed in a centralized or distributed manner, any real-time data engineering pipeline should therefore include mechanisms for continuous monitoring, quality validation, and cleansing along the data-processing paths. This section outlines a set of quality assurance activities spanning multiple nodes, examining how they can safeguard against propagation of faults and errors.

Real-time healthcare data pipelines typically draw on multiple sources to provide a consolidated view of real-time patient conditions. A record-consistent view of the data thus requires successful checking and data-handling measures at all sources. To detect possible problems at database or service levels, any upstream data node should have an appropriate validation module. For databases, these checks can determine the success of recent health-care operations—surgical or nonsurgical—involving a patient. For external clinical systems, at minimum, that such a

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

patient's principal data elements (e.g. identifiers, main conditions, allergies) remain unchanged. Such validation activities may be trivial for low-profile systems but can incur a substantial operational load for high-profile systems.

4.2. Provenance, Lineage, and Auditing

Within heavily regulated sectors, self-documenting systems must provide reliable accounts of the origins and transformations of their data over time. Provenance details for data in real-time streams must therefore be captured and preserved, along with lineage information for the relationships that link various stream data sources. Auditing requirements can also necessitate reliably tamper-evident records. To meet these requirements, across-the-board tracing of the trajectory of all data as it passes through the data streams is essential. All probe nodes of the data streams should therefore collect and record provenance information for their corresponding input and output streams. Provenance and lineage data can be retained in different ways based on the implementation of various downstream processes.

Provenance and lineage information should be aggregated and stored in a suitable back-end for further analysis, thereby creating a tamper-evident trail of every event in the stream at a higher level of detail. In addition to this stored data, other techniques should be adopted to ensure compliance with the auditing needs. The alerts generated from the detected violations of the safety requirement should be treated as provenance data of the corresponding stream processing systems. A clinical alert and recommendation system participating in a safety process therefore creates provenance data that link the detected violation with the recommendations provided to the clinician.

V. SCALABILITY, RELIABILITY, AND PERFORMANCE CONSIDERATIONS

Real-time integration of health data requires careful consideration of scalability, reliability, and performance. Scalability determines whether a system can continue to run efficiently as the workload increases. An unreliable system that fails to deliver results is of limited use. It is important to measure both reliability and service-level agreements under load to understand their limits and determine appropriate burn-in periods. Real-time data processing requirements are generally satisfied with cloud infrastructure; however, rapid scaling can be difficult, especially for workloads with unpredictable bursts.

A workload can affect response times in different ways in each processing model. Batch processing performance degrades as the volume of data increases, and it becomes possible to introduce additional resources to parallelize processing. For microservices, a high volume of data from a single source or for a single query can lead to longer processing times and, consequently, slower response times. For event-driven architectures, the only limitation on responsiveness in processing lower-priority events such as clinical decision support rules is the latency in executing queries on the data source. Thus, event-driven architectures lend themselves to better throughput for processing queries from multiple event sources. High throughput is possible for alert-triggering rules in event-driven architectures that naturally connect with the data sources; responsiveness of the rule evaluation is only due to the latency of updates to the sources.

Latency, throughput, and service-level agreements have different meanings depending on the application. During development, testing, and initial deployment, burn-in periods allow systems to be tested and adjusted in a more lenient manner before full service-level agreements are applied. Burn-in periods also help assess the averaging value of latency, throughput, and other measurements, which may all have high variance early in a system's execution. Latency measures the amount of time between the occurrence of an event and the delivery of a result for that event. It is often referred to as the end-to-end response time of the system. Throughput measures the number of events processed by the system or delivered to end clients, such as the number of alerts sent per unit time. Service-level agreements for throughput generally specify a minimum value to ensure the system can handle the expected load, rather than a maximum value.

5.1. Distributed Processing Models and Fault Tolerance

Distributed Data Engineering Frameworks for Real-Time Healthcare Record Integration: A variety of streaming workloads and their relative characteristics support decisions on processing models, degree of parallelism, and choice of fault-tolerance techniques in the context of healthcare data integration. At the fine-grained level, distributed streaming processing is governed by system-level limitations rather than component behavior, while at coarser granularity trade-offs between latency and throughput, transient load variations, and availability requirements become more important.

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

Streaming-integration workloads often cover two or more of the aforementioned purposes but are usually dominated by one. When data streams into a system, for example an EHR, testing for faults happens quickly at the outer boundary. In contrast, when data leaves the system—when a streaming rule engine evaluates rules and generates alerts or external systems read from event streams—the most important question can be one of throughput. What volume of alerts can be generated? How many external systems can read alerts without impacting operations? How many drops of the most volatile alerts can be tolerated? These service-level agreement parameters can then guide burn-in periods and capacity planning.

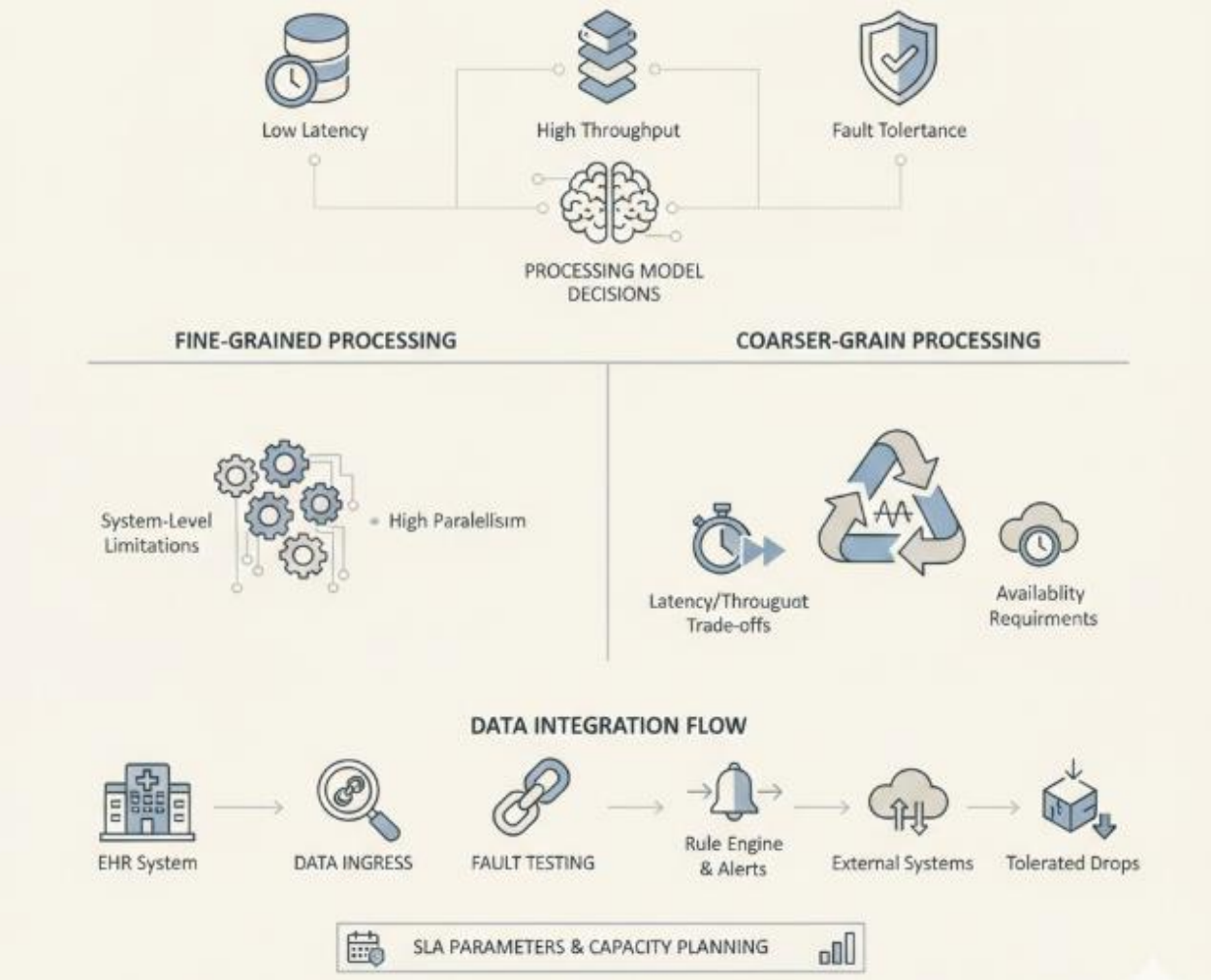


Fig 3: Optimizing Distributed Stream Processing for EHR Integration: Throughput-Latency Trade-offs and Fault-Tolerant Frameworks in Real-Time Healthcare Informatics

5.2. Latency, Throughput, and SLA Management

Measured latency and throughput are important performance indicators for time-sensitive applications. Latency quantifies the completion time from event origination to destination arrival, while throughput records the rate of completed transactions over a specific interval. Stream processing solutions deployed to meet strict time constraints often undergo a burn-in period, representing the time between deployment and steady-state operation. During this phase, sensor data is ingested, rule bases activated, and downstream consumers trained to meaningfully interpret the data flow. The duration of burn-in varies with the processing network characteristic time, which approaches the input data rate in low-latency designs.

Service-level agreements formalize quality-level commitments between service consumers and providers. Timing-sensitive services require dedicated SLA specifications. Latency responses to business service requests may also be specified in SLA terms for oversight, monitoring, and capacity evaluation. Failure to meet SLAs in live or burn-in

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

phases typically indicates an insufficient deployment with inadequate resources. Burn-in periods inform SLA management by providing a steady-state latency estimate, while rate determination enables capacity planning.

VI. USE CASES AND DEMONSTRATIONS IN HEALTHCARE SCENARIOS

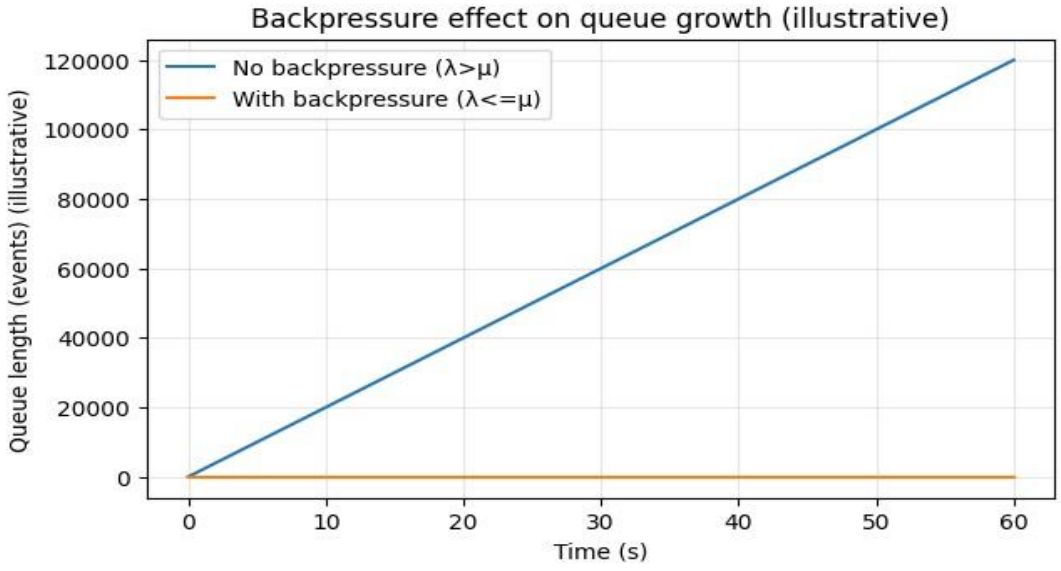
Representative instances and demonstrations illustrate specific data flows within the system and the use of real-time data for decision-support tasks. They respond to two critical questions: What data flows into and out of the system in real time? What decisions are supported in real time, and how are they carried out?

The first question concerns real-time integration, focusing on patient record ingestion and synchronization across clinical systems. First, a patient record is inserted into the system; for example, when new records are created in an external electronic health record (EHR) system. The integration framework subsequently handles updates to the record, including both controlled attribute changes, such as changes to the patient's active medication list, and potentially conflicting updates from multiple systems. The second question focuses on the application of real-time data to clinical decision support and alerting. A set of business rules determines whether rules apply to a record. Rules are evaluated as data arrive in the system, and alerts are delivered independently of any specific, system-initiated action. For critical clinical safety monitoring functions, a burn-in period ensures a stabilizing signature for the relevant data.

6.1. Real-Time Patient Record Ingestion and Update Propagation

Sooner or later, any organization making extensive use of electronic health records (EHRs) will need to deal with nontrivial issues arising from the fact that these records have been created and are being maintained by other organizations: In a large health network, it may be feasible for an organization to connect directly to those EHRs for consultation and possibly also for updates, but outside of such a network, continuity of care may require more than mere reference. Many other systems, such as hospital information systems (HISs) and emergency services dispatch centers, maintain similar records for relatively short time periods. These system are frequently consulted, but seldom for changes. Nevertheless, a centrally controlling SOA can place these records into a common time series with lower priority, and others real-time clinical decision support (CDS) mechanisms can serve as time-critical independent monitors. During times of surge, alarms can be expanded.

Patient records from sources other than the primary health-care provider organization can be ingested in real time and merged into a central integration hub while preserving the original data sources in their native format. Although the process is preferably nonintrusive, it can occasionally require updates to records stored in other systems using standard FHIR resources. However, multiple systems may have independently generated overlapping but contradictory clinical data about the same patient during the same timeframe: To mitigate this, a set of conflict-resolution and reconciliation rules can be applied in order of precedence across the systems involved, while outof-band mechanisms can identify problems in the medical records to relevant personnel.



International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

Equation 4) Event-time → ordering/watermark equations

A common mathematical way to express correctness under out-of-order arrivals:

9. Each event has:

- event-time t_e (when it happened),
- processing-time t_p (when the system sees it).

3. Out-of-orderness (lateness) for an event:

$$\ell = t_p - t_e$$

4. If you allow lateness up to δ, you accept events if:

$$\ell \leq \delta$$

6.2. Clinical Decision Support and Alerting in Real Time

In scenarios requiring clinical decision support and alerting, real-time event evaluation detects patients meeting specific conditions and generates appropriate notifications. Example use cases include the identification of patients at risk for venous thromboembolism, renal failure, or other complications, and the alerting of relevant healthcare providers. Rules may employ additional external information in their conditions, such as the coverage status and presence of genotyping results for a patient starting a course of therapy with a positively associated genotype.

Events are evaluated at an interval defined by the system administrator, which balances the needs of timely delivery with the overhead of processing the rules. Each rule's disposition—when to warn, what message to send to whom—is also defined externally, typically in a database table populated by a non-technical user. For instance, conditions generating such messages could be discerned semantically in a public database like RXNORM, facilitating straightforward and safe clinical safety monitoring.

Alerts are complementary to the real-time stream; they serve to convey knowledge relevant for clinical safety. The rules are defined and managed outside the safety monitoring system itself, enabling the creation of additional alerts without direct technical involvement in the stream processing pipeline. Nevertheless, they may be bolstered within the pipeline by also providing decision support for interaction with directed therapy beyond traditional coverage queries.

VII. CONCLUSION

Real-time integration of health data from heterogeneous sources and the support of up-to-date clinical decision support and alerting systems are essential steps toward a learning health system. These processes should not be limited to a dedicated set of systems; rather, any data-producing information system should be able to contribute. Therefore, the integration operations should be performed in an automated way, possibly with little human supervision, simplifying the work for healthcare professionals. The demarcation between distributed systems, data pipelines, and data quality may be blurry in health data engineering: these domains interact with each other, and advances made in any one area positively impact the others. The opportunistic approach assumes that integration occurs when the timing and context are suitable.

Given the breadth of real-time data engineering for healthcare and the impact of individual aspects on the entire system, ranging from stability to timely alerting, it is not realistic to expect a single, monolithic solution. Multiple solutions exist, each focused on specific aspects of the problem domain, such as microservice-based data pipelines, data quality in streaming environments, or support for safe data ingestion. These solutions can interoperate if properly designed; in particular, those addressing different aspects of real-time health data integration are good candidates for combining efforts. Despite addressing a specific area with various solutions, the proposed approaches remain relevant to the broader domain of distributed data engineering for health.

International Journal of Science, Research and Technology (IJSRAT)

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

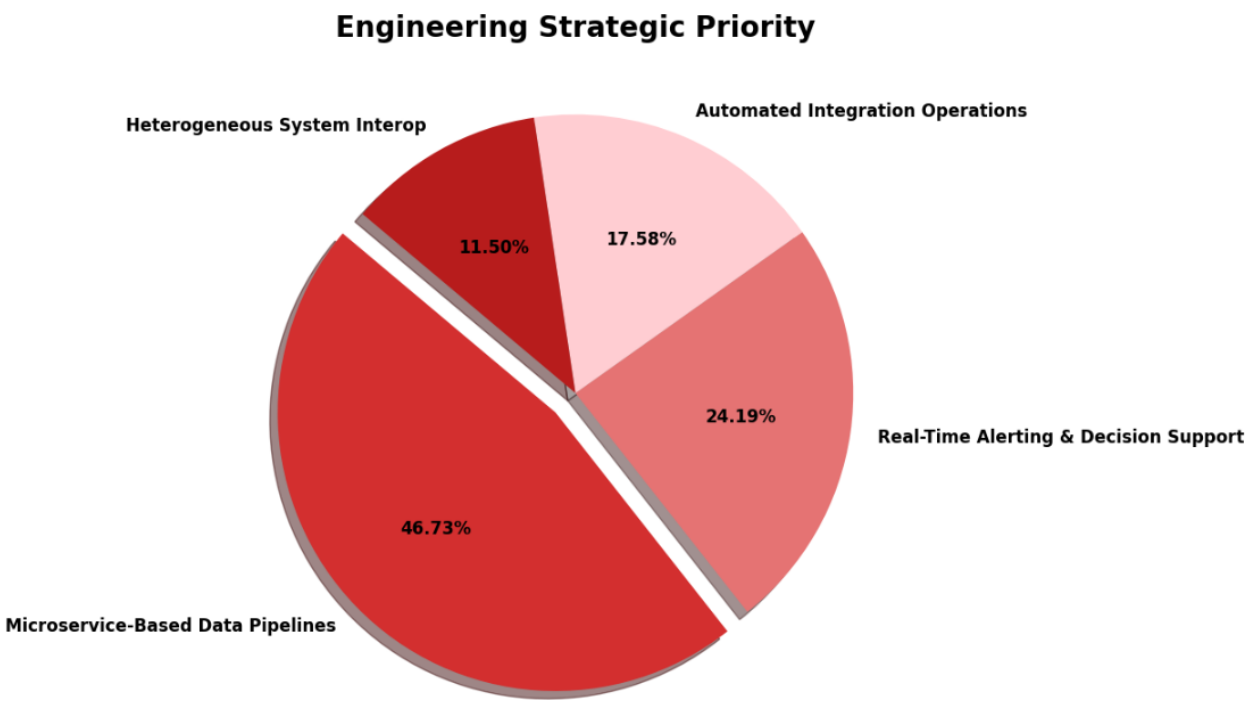


Fig 4: Engineering Strategic Priority

7.1. Summary and Future Directions in Healthcare Data Engineering

Real-time distributed data engineering is increasingly used to assist with timely clinical decision-making, automate responses to anticipated conditions, and enable proactive patient monitoring. A comprehensive and integrated information fabric built using these principles can support many current and future use cases, as well as support requirements for data quality, governance, security, scalability, and reliability. Timeliness alone, however, cannot guarantee the correctness and usefulness of the data. Real-time data makes inference more attractive, but rule results still must be verified to prevent false positives. The real-time fuse-and-forward example indicates that while an end-to-end pipeline is feasible, latency remains an issue. Future enhancements may include automated verification of alert conditions, agent-based monitoring systems, the continued expansion of external data sources, clinical lab result prediction, and deduplication of response actions to prevent overreaction.

Beyond individual uses, real-time data engineering within healthcare organizations is likely to gain momentum as proof-of-concept deployments transition to production settings. The combination of low-latency integration driven by clinical practices, the increasing popularity of non-EHR systems in hospitals and research, acceptance of web-based frameworks for governance in life sciences, and advances in data engineering may come together to establish real-time data streams as a fundamental component of biomedical data ecosystems. Nevertheless, gaps in pipeline operation remain to be addressed, such as data quality assurance, forward privacy preservation, capture of provenance and lineage details, and the provision of tunable service-level agreements, supported by burn-in periods during initialization. Real-time data engineering, combined with high-quality and governed record integration of all data feeds, can facilitate a new wave of smart healthcare applications in the public health domain, such as disease outbreak detection, infectious disease propagation modeling, disaster preparedness, and intelligent response.

REFERENCES

[1] Akidau, T., Chernyak, S., & Lax, R. (2018). Streaming systems: The what, where, when, and how of large-scale data processing. O'Reilly Media.

[2] Amistapuram, K. Energy-Efficient System Design for High-Volume Insurance Applications in Cloud-Native Environments. International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering (IJIREEICE), DOI, 10.

[3] Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A distributed messaging system for log processing. Proceedings of the NetDB Workshop.

|Bimonthly | Peer-Reviewed | Scholarly Journal|
|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|
DOI: 10.15662/IJSRAT.2020.0306002

[4] Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005). The 8 requirements of real-time stream processing. ACM SIGMOD Record, 34(4), 42–47.
[5] Carbone, P., Katsifodimos, A., Ewen, S., et al. (2015). Apache Flink: Stream and batch processing in a single engine. IEEE Data Engineering Bulletin, 38(4), 28–38.
[6] Inala, R. Designing Scalable Technology Architectures for Customer Data in Group Insurance and Investment Platforms.
[7] Kleppmann, M. (2017). Designing data-intensive applications. O'Reilly Media.
[8] Newman, S. (2015). Building microservices. O'Reilly Media.
[9] Pahl, C. (2015). Containerization and the PaaS cloud. IEEE Cloud Computing, 2(3), 24–31.
[10] Varri, D. B. S. (2020). Automated Vulnerability Detection and Remediation Framework for Enterprise Databases. Available at SSRN 5774865.
[11] Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California.
[12] Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1), 107–113.
[13] Tanenbaum, A. S., & Van Steen, M. (2017). Distributed systems (2nd ed.). Pearson.
[14] Bass, L., Clements, P., & Kazman, R. (2013). Software architecture in practice (3rd ed.). Addison-Wesley.
[15] Gottimukkala, V. R. R. (2020). Energy-Efficient Design Patterns for Large-Scale Banking Applications Deployed on AWS Cloud. power, 9(12).
[16] Fowler, M. (2018). Patterns of enterprise application architecture. Addison-Wesley.
[17] Bender, D., & Sartipi, K. (2013). HL7 FHIR: An agile and RESTful approach to healthcare information exchange. Proceedings of CBMS, 326–331.
[18] Davuluri, P. N. (2020). Improving Data Quality and Lineage in Regulated Financial Data Platforms. Finance and Economics, 1(1), 1-14.
[19] Lehne, M., et al. (2019). The use of FHIR in digital health. Studies in Health Technology and Informatics, 267, 52–58.
[20] Rongali, S. K. (2020). Predictive Modeling and Machine Learning Frameworks for Early Disease Detection in Healthcare Data Systems. Current Research in Public Health, 1(1), 1-15.
[21] Jensen, P. B., Jensen, L. J., & Brunak, S. (2012). Mining electronic health records. Nature Reviews Genetics, 13(6), 395–405.
[22] Shickel, B., Tighe, P. J., Bihorac, A., & Rashidi, P. (2018). Deep EHR. IEEE Journal of Biomedical and Health Informatics, 22(5), 1589–1604.
[23] Singireddy, S., & Adusupalli, B. (2019). Cloud Security Challenges in Modernizing Insurance Operations with Multi-Tenant Architectures. International Journal of Engineering and Computer Science, 8, 12.
[24] Esteva, A., et al. (2019). A guide to deep learning in healthcare. Nature Medicine, 25(1), 24–29.
[25] Raghupathi, W., & Raghupathi, V. (2014). Big data analytics in healthcare. Health Information Science and Systems, 2, 3.
[26] Sittig, D. F., & Singh, H. (2016). A socio-technical approach to EHR-related safety hazards. Journal of the American Medical Informatics Association, 23(4), 641–647.
[27] Vadisetty, R., Polamarasetti, A., Guntupalli, R., Rongali, S. K., Raghunath, V., Jyothi, V. K., & Kudithipudi, K. (2020). Generative AI for Cloud Infrastructure Automation. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 1(3), 15-20.
[28] Wilkowska, W., & Ziefle, M. (2012). Privacy and data security in e-health. Health Policy and Technology, 1(3), 119–131.
[29] Abouelmehdi, K., Beni-Hessane, A., & Khaloufi, H. (2018). Big healthcare data security. Journal of Big Data, 5, 1.
[30] Inala, R. (2020). Building Foundational Data Products for Financial Services: A MDM-Based Approach to Customer, and Product Data Integration. Universal Journal of Finance and Economics, 1(1), 1-18.
[31] Berwick, D. M., Nolan, T. W., & Whittington, J. (2008). The triple aim. Health Affairs, 27(3), 759–769.
[32] Porter, M. E. (2010). What is value in health care? New England Journal of Medicine, 363(26), 2477–2481.
[33] Pope, G. C., et al. (2004). Risk adjustment of Medicare capitation payments using the CMS-HCC model. Health Care Financing Review, 25(4), 119–141.
[34] Rose, S. (2016). A machine learning framework for plan payment risk adjustment. Health Services Research, 51(6), 2358–2374.
[35] Braverman, A., et al. (2020). Deep learning for Medicare risk adjustment. Health Services Research, 55(1), 68–77.
[36] Adusupalli, B., Pandiri, L., & Singireddy, S. (2019). DevOps Enablement in Legacy Insurance Infrastructure for Agile Policy and Claims Deployment. risk, 7(12).

|Bimonthly | Peer-Reviewed | Scholarly Journal|

|ISSN 2454-3853|www.ijsrat.com|editor@ijsrat.com|Vol.3, Issue 6, November-December 2020|

DOI: 10.15662/IJSRAT.2020.0306002

[38] Breunig, M. M., et al. (2000). LOF. Proceedings of SIGMOD, 93–104.
[39] Liu, F. T., et al. (2008). Isolation forest. Proceedings of ICDM, 413–422.
[40] Kingma, D. P., & Welling, M. (2014). Auto-encoding variational Bayes. ICLR.
[41] Goodfellow, I., et al. (2014). Generative adversarial nets. NeurIPS.
[42] Kipf, T. N., & Welling, M. (2017). Graph convolutional networks. ICLR.
[43] Hamilton, W., Ying, Z., & Leskovec, J. (2017). GraphSAGE. NeurIPS.
[44] Segireddy, A. R. (2020). Cloud Migration Strategies for High-Volume Financial Messaging Systems.
[45] Wu, Z., et al. (2020). A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems, 32(1), 4–24.
[46] Grover, A., & Leskovec, J. (2016). node2vec. Proceedings of KDD, 855–864.
[47] Perozzi, B., et al. (2014). DeepWalk. Proceedings of KDD, 701–710.
[48] Paszke, A., et al. (2019). PyTorch. Advances in Neural Information Processing Systems, 8024–8035.
[49] Pedregosa, F., et al. (2011). Scikit-learn. Journal of Machine Learning Research, 12, 2825–2830.
[50] Akhtar, N., & Mian, A. (2018). Adversarial attacks. IEEE Access, 6, 14410–14430.
[51] Finlayson, S. G., et al. (2019). Adversarial attacks on medical machine learning. Science, 363(6433), 1287–1289.
[52] Ribeiro, M. T., et al. (2016). Why should I trust you? Proceedings of KDD, 1135–1144.
[53] Lundberg, S. M., & Lee, S.-I. (2017). SHAP. Advances in Neural Information Processing Systems, 4765–4774.
[54] Rudin, C. (2019). Stop explaining black box machine learning models. Nature Machine Intelligence, 1, 206–215.
[55] Molnar, C. (2019). Interpretable machine learning. Lulu.
[56] Pandiri, L., Singireddy, S., & Adusupalli, B. (2020). Digital Transformation of Underwriting Processes through Automation and Data Integration. Global Research Development (GRD) ISSN, 2455-5703.
[57] Johnson, A. E. W., et al. (2020). MIMIC-IV. Scientific Data, 7, 412.
[58] Haux, R. (2010). Medical informatics: Past, present, future. International Journal of Medical Informatics, 79(9), 599–610.
[59] Topol, E. (2019). Deep medicine. Basic Books.
[60] Beam, A. L., & Kohane, I. S. (2018). Translating artificial intelligence into clinical care. New England Journal of Medicine, 379(23), 2165–2167.
[61] Obermeyer, Z., & Emanuel, E. J. (2016). Predicting the future. New England Journal of Medicine, 375(13), 1216–1219.
[62] Wiens, J., et al. (2019). Do no harm. Nature Medicine, 25(9), 1337–1340.
[63] World Health Organization. (2016). Global diffusion of eHealth. WHO.
[64] OECD. (2019). Health in the 21st century. OECD Publishing.
[65] European Commission. (2020). European data strategy.
[66] Kuo, A. M. H. (2011). Cloud computing in healthcare. Journal of Medical Internet Research, 13(3), e67.
[67] Buyya, R., et al. (2010). Cloud computing and emerging IT platforms. Wiley.
[68] Amor, D., et al. (2020). Data governance in healthcare. Health Informatics Journal, 26(2), 1167–1181.
[69] Mans, R., et al. (2008). Application of process mining in healthcare. Information Systems, 33(4–5), 389–406.
[70] Rojas, E., et al. (2016). Process mining in healthcare. Journal of Biomedical Informatics, 61, 224–236.
[71] van der Aalst, W. (2016). Process mining (2nd ed.). Springer.
[72] Dumas, M., et al. (2018). Fundamentals of business process management (2nd ed.). Springer.