Cloud-Native Architectures for High-Throughput Distributed Data Processing

Karthikeyan Selvarajan
Senior Staff Software Engineer, San Francisco, California, United States
karthik.selvarajan83@gmail.com

Abstract

Denser cloud-native designs are a robust answer to this increasing cry for high-throughput distributed data processing. The increasing volume, rate, and type of information require a processing system that can flexibly manage the resources being used without compromising performance, dependability, and cost efficiency. This paper outlines a distributed data processing system engineered using containerization, orchestration, and distributed computing technologies as cloud-native systems. The proposed solution includes Apache Spark to process data in parallel and Kubernetes to control the activities of containers so that allocation elasticity in resources and proper workload management in the distributed environment can be ensured. It is characterized by a set of architecture, which is composed of automatic scaling, workload scheduling, fault tolerance, service isolation, and load resource monitoring in order to foster large-scale data-intensive applications. With the implementation of micro services and containers, scaling microservices can be implemented on a scale, and can have the processing unit that has a scaling and management ability based on the requirement of the workload. The paper has demonstrated that the ideas of cloud-native enhance throughput, scalability, and flexibility and resource usage than the traditional distributed processing environment. The proposed architecture is able to scale to real-time analytics, huge-scale batch processing, machine learning pipelines, as well as data-intensive applications in the enterprise. The results reveal that, together with distributed processing frameworks, using cloud-native technologies creates a flexible and scalable platform to support efficient high.

Keywords: Cloud-Native Architecture, Distributed Computing, Apache Spark, Kubernetes, Data Processing, Scalability

DOI: 10.21590/ijhit.04.04.10

1. Introduction

The growth of digital services, interrelated technologies, cloud computing services and technologies that require massive amounts of data has led to larger quantities and demands of data generated in organizations more than ever. Internet of Things (IoT) use, social media analytics, financial services, healthcare, and scientific, and even artificial intelligence and recommendation systems in real-time continue to generate large datasets, which need an effective method of processing. Classical centralized data-processing systems are usually not scalable to such workloads, because of their limited computing capabilities, scalability, hardware-dependence and increasing infrastructure expenses. The distributed data-processing systems hence are becoming decisive in offering high throughput, shorter processing time as well as efficient utilization of the computational resources [1].

Cloud computing has drastically revolutionized the creation and implementation of distributed applications by offering a new on-payment access to computing, storage and networking infrastructure. However, the traditional cloud-based applications may still be faced with some problems related to resource management, scalability of applications, and complexity of implementation and efficient utilization of the heterogeneous infrastructure [2]. Cloud-native architecture delves into problems by purposely designing applications such that they can work in dynamic clouds. It focuses on containerization, microservices, automation, elastic scale, fault tolerance, and continuous deployment along with abstraction of infrastructure. These principles enable dynamically scaled applications to scale their operation dynamically to changing workload without depending on the presence of a particular physical infrastructure [3].

Distributed computing frameworks break down huge workloads into smaller tasks, and put these tasks to run in parallel hence cutting down on the processing time. Apache Spark is one of the distributed processing frameworks popular due to process ability in memory, parallel processing, fault tolerance framework, and capability to process batches, streams, machine learning and big data processing graphs. Spark is scaled to allow distribution of the calculation work in the clusters to ensure that, it can be applied in the high-processing throughput tasks. However, in order to efficiently operate and maintain Spark workloads in dynamic clouds, they require the presence of a good orchestrator.

Kubernetes is a container orchestrator platform that can manage the discontinuous application in clusters of physical and virtual machine cluster. It automates containers deployment, scheduling, scaling, service discovery, resource allocation, as well as recovery. It is possible to integrate Kubernetes with distributed computing systems, like Apache Spark to deploy computational workloads as containerized services and dynamically place resources per workload demands. The uniqueness is an important move in the evolution of scalable platforms in the cloud as far as data-processing is concerned.

Despite these benefits, it is difficult to guarantee high throughput on a constant basis of distributed performance while using a cloud-native application. Even the discrepancies between the data-processing loads can be horrifying: the amount of workload input, the intricacy of the requirements of the workload computation, the memory needs, and time. The resources can be apportioned in a fixed way with underutilization of the resources in times of low workloads, and poor performance in times of colossal workloads. Further, there are also problematic issues (in distributed environment) in terms of scheduling of tasks, network communication, locality of data, and contention among resources, fault recovery and performance monitoring. Therefore, a good architecture should be in a position to be able to dynamically coordinate the computational resources whilst ensuring reliable and efficient data processing.

Distributed processing and automated management of resources can be the cornerstones of implementation of such issues into a cloud-native structure. Containerization provides the isolation and portability of applications and Kubernetes provides a dynamic scaling and planning of the processing workloads. Parallel computing will be offered by Apache Spark and elastic resources will be offered by the cloud computing, which can be scaled on demand. Continuous analysis of resource consumption, processing latency, and throughput and system health can also be backed by monitoring and observability mechanisms. The combination of these technologies can offer an environment in which data-processing applications can dynamically respond to changes in workload.

The proposed research works will focus on exploring a high-throughput and distributed data processing cloud-native platform which will be based on the termination of Apache Spark and Kubernetes. The proposed architecture is workload-oriented to execute, resource utilization is more efficient, automatic scaling, fault tolerance and scalable deployment. It aims at providing the generalized base of managing the large bulk data in distributed cloud set-ups, yet maintains the high throughput and operation rates. The architecture has the capacity to handle a variety of workloads such as large-scale batch analytics, real-time data processing, machine learning pipelines and enterprise data applications.

Such an approach is different, as the computing capability of distributed processing can now be a reality due to the adaptability and automation of cloud-native infrastructure. Organizations can ensure maximum performance under provisioning without offering unfair infrastructure provisioning by delivering the resources on-demand by workloads. Cloud-native architectures thus, represent an enlightening path in the making of the scalable, resilience and efficient architecture that can sufficiently accommodate the escalating needs of the present state-of-art data-processing programs.

2. Related Work

The changes in data placement have facilitated the realization of high-throughput distributed data processing along with stream processing and containerization as well as cloud infrastructure, and intense scientific workloads on data. Efficient storage and processing scheme to massive workloads became important with early researches which then now turn to concentrate on scaled up, disseminated and cloud architecture research.

The positioning and storage optimization of data in the distributed environment are one of the key research problems. In [1], the paper talked about optimization approach of hot and cold levels of top-K workloads. The issue addressed in the work was how to determine the appropriate levels of storage of the information that is frequently or not frequently used. High-throughput systems care about such placement since it can enable their efficient locality of data and minimizes the access latency wastage of any given resource and contributes to the efficient use of that resource. The solution however is mainly focused on where the storage tier is located and is not fit to offer a single cloud-native solution including distributed processing, container orchestration and monitoring, and scale elasticity.

There have been extensive stresses on the stream processing, applications of the real-life, which resonate with the existing streaming processing data actions are continuously growing. A comparison between Apache Spark streaming, Kafka and harmonica performance is presented in the paper [2]. The article reviewed the various organizations of the enterprise and scientific computing and emphasized the roles of throughput, latency and system architecture of the streaming workloads. Apache Spark can be used with distributed parallel processing and Kafka is a trustworthy and streaming message and events topology. This fact displays that the result of the introduction of the complementary distributed technologies is pleasant, and the literature primarily dwells on the performance benchmarking, as opposed to the extrapolation of a cloud-native architecture, considering the performance of dynamically balancing the resources to non-homogeneous workloads.

In [3], another stream-processing system was introduced that was called harmonica to support scaling of processing scientific data. The system was created to handle huge streams of scientific data, and scale up to data-intensive programs. Its architecture shows the significance of data streaming and effective movement of data in working with the constantly

generated data. Nonetheless, it is more oriented on the scientific workloads and does not address bigger enterprise demands, such as Kubernetes-based orchestration, microservice isolation, automated scaling, and single governance.

The other aspect that has been brought to the limelight is containerization that has now emerged as a significant point of origin when it comes to movable and repeatable data-processing systems. In [4], a biomathematical workflow with containers was presented, which demonstrated that container solutions could add value in terms of deployment and execution and reproducibility of computational pipelines. Containerization helps to minimize the issue of dependencies and allows apps to be cross-computer inter-operable. However, containerization is not able to guarantee effective resource scheduling and flexibility in execution. Resources allocation dynamically are also needed to facilitate sound high-throughput processing as well as giving them the capacity to allocate resources dynamically as per the need of workload through orchestration mechanisms.

Another key research direction is resource-aware scheduling. The paper [5] investigated resource and message size sensitive scheduling of the edge processing of streams, especially real-time microscopy purposes. The experiment revealed that the nature of workloads, availability of resources, and the size of the message might be very influential in determining the performance of distributed stream-processing. This kind of scheduling method could possibly be used in the minimization of the delays in processing as well as the designer scheduling. But it addresses mainly edge-based stream-processing applications and cloud-native data-platforms involve orchestrating when containers and clusters are scheduled, storage and distributed processing engines.

The scientific and data-intensive applications that are distributed in a large scale cloud infrastructure have also enabled large scale cloud infrastructure. In [6], academic clouds infrastructure at national level was described which illustrates how researcher can be offered with cloud resources with the aid of scalable computing infrastructure. Good with these infrastructures, they may be elastic in resources availability and enhance accessibility of the distributed computing resources. However, the infrastructure level provisioning fails to satisfy the needs of workload-friendly scheduling, distributed computations, service coordination, and auto-scaling on the application-level.

Much big-data literature has dealt with the increasing sizes and heterogeneity of data. The studies carried in [7] had found that the quantity and complexity of the information developed in science areas were soaring high. These characteristics impose colossal demands of scalable storage, distributed computation and effective data-management processes. Undoubtedly, [8] was a comprehensive discussion of the big-data issues and methods of analysis that involved concerns about data size, type of data, data speed, complexity of data processing and scalability of data analysis. These articles represent a drive of the premier rank distributed data-processing frameworks in which, a cloud-native deployment (Sys) is not explicitly overlaid on top of technologies, e.g., Spark and Kubernetes.

Live-time raid of scientific information has had impacts in developing streaming oriented systems as well. A data-streaming of real time saves can help in making scientific decisions based on workflow in real time as the experiments in [9] illustrated a data-streaming metabolomic experiment strategy. The article exemplifies the importance of utilizing information as close to the source as you can and respond effectively to inflow. Nonetheless, it lacks the generalization of enterprise or cloud-native workloads in more general form since it must be implemented in its domain.

To ensure security and reliability, standardized protocols have also been explored to deliver on safety and effectiveness in data streaming. The htsget protocol which is proposed in [10] offers the capability to obtain to streaming genomic information safely and instead can obtain more exact and finer information which is necessary without sending massive large-sized datasets was only needed to transfer a few bits of information. This method shows how effective and limited access to data can be effective when management of large data sets is required. However, the research concentrates more on data-access mechanisms as opposed to the distributed execution; resource coordination, and capacity of high-throughput processing platforms.

On balance, the current literature shows that there has been a significant advancement in terms of data placement [1], distributed stream processing [2], [3], containerized workflow [4], resource-aware scheduling [5], cloud infrastructure [6], big-data management [7], [8], real-time data acquisition [9], and secure data streaming [10]. However these strategies are generally more focused on a particular aspect of the problem, and not a collection of that into a coherent cloud-native design. A combined distributed processing, containerization, orchestration, victimized resource allocation, workload, fault tolerance and monitoring architecture are needed. The proposed work fills this gap since it studies the topic of cloud-native architecture which deploys the widespread processing technologies particularly Apache Spark and Kubernetes to acquire both scaled and high-throughput information processing.

3. Cloud-Native Elastic Execution Fabric: Architecture and Dataflow

It is proposed that the architecture is a throughput, execution cloud based architecture that makes use of distributed data processing that incorporate distributed computation, containerization, orchestration, continuous monitors and elastic resource management. It must be designed in a way that it can handle both the batch and the streaming loads and can be scaled in addition to fault tolerance, portable yet using small resources. The general structure is based on the layered approach where data ingestion, storage, processing, orchestration, and monitoring are loosely coupled but made to interact based on shared control mechanisms.

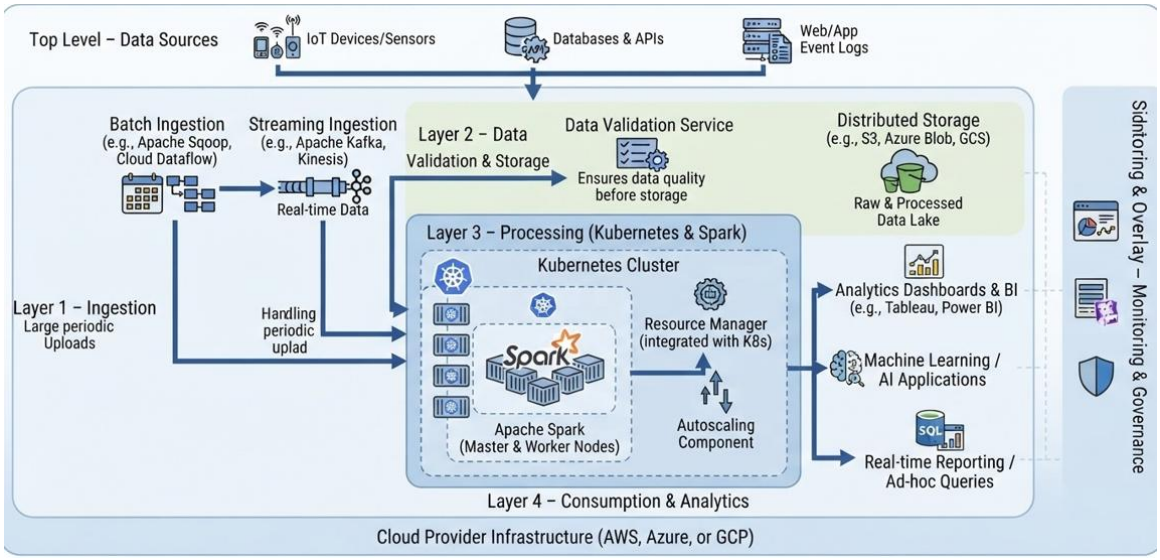


Figure 1 – Cloud-Native Architecture for High-Throughput Distributed Data Processing

3.1. Architectural Overview

The top-level condenses six big layers: Data Sources and Ingestion, Distributed Storage, Processing and Analytics, Container and Orchestration, Resource Management, and Monitoring and Governance. The physical separation between these layers enables the scaling of each of the parts by itself as per workload requirements. It is also able to deliver applications as containerized services and distributed processing jobs and not executed within a monolithic processing environment.

There are two important modes of processing that are supported by the architecture. Large volumes of historical data are processed using batch pipelines in the former mode. Second mode is where continually generated data is processed in pipelines in a streaming fashion. The storage and orchestration infrastructure, monitoring and resource-management infrastructure of the two modes are the same and therefore minimizes duplication and maximizes uniformity of operations.

3.2. Data Sources and Ingestion Layer

These are the heterogeneous data in the first layer which form structured, semi-structured data and unstructured data. These can be enterprise databases, application logs, IoT and sensor, the transaction system, APIs, scientific applications, and event streams. The architecture offers set data ingestion and streaming data ingestion streams.

Big data ingestion We consider ingestions that involve large volumes of historical data into the distributed storage layer as big different types of ingestion and ingestions that transfer events continuously into near-real-time processing's as streaming ingestions. Producers can be decoupled and consumers can be decoupled through a powerful event buffering message-broker system like Apache Kafka. This decoupling is what helps to absorb other temporary adjustments in the rate at which data-generation occurs in the downstream processing elements without causing an immediate overloading.

Basic validation and schema checking, filtering and metacreation are also done by the ingestion layer. Such functions make sure that invalid or malformed data does not needlessly eat up distributed processing resources.

3.3. Distributed Storage Layer

The distributed storage layer provides a persist and scaled storage of the incoming and processed data. A cloud compatible distributed file system or object-storage platform can store large volumes of data, in order to provide high availability and parallel access.

The storage architecture opens up the opportunities of subdivision of the data logically into raw, refined and curate space. The raw zone retains data flowing into it with a minimum transformation. The refined zone holds data that is verified and standardized and the curated zone holds the refined data, optimized to be used in analytics and have downstream use.

This tiered approach to storage decouples data ingestion and the execution of analytical processing and enables various processing tasks to obtain suitable versions of the data. Others are replication and distributed storage mechanisms that imply that the problems are not sensitive to failure of single node. Moreover, storage and computing are physically dispersed, but can be processed in a manner that can be scaled separately of storage resources.

3.4. Distributed Processing Layer

The engine of the distributed processing of the proposed architecture is based on Apache Spark. Spark is chosen as it allows to run parallel on a variety of compute nodes and has APIs to run batch, streaming, SQL based analytics and machine-learning workloads.

A Spark application is partitioned into tasks which can run in parallel using worker resources. The Spark driver speeds up the running of the applications, the distributed computation is performed by executors and intermediate data is stored. With the help of this model, big data sets will be segregated and ran at the same time.

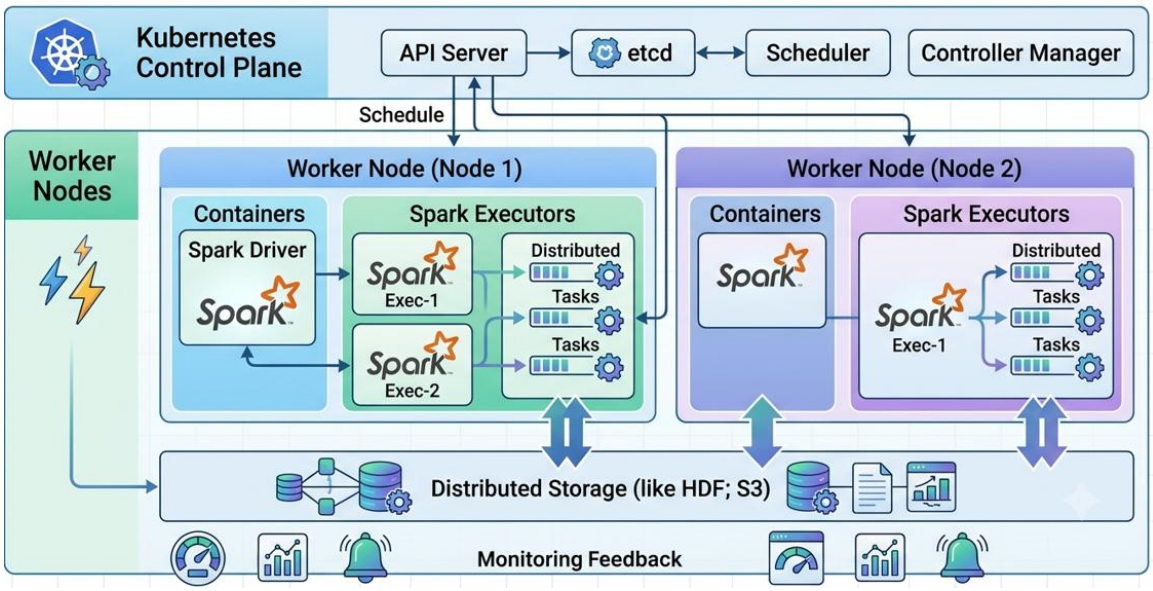


Figure 2 – Spark–Kubernetes Execution Model

Spark executes tasks and conversion of distributed data in case of bulk workloads. In the case of streaming workloads, Spark Structured Streaming is able to stream incoming event massages. Both historical analytics and real-time data pipelines can thus be supported using the same processing environment.

Certain optimizations of workloads are also implementable at the processing layer which include; workload partitioning, caching, parallel processing, tool able resource allocation. The processes dominate in large variance of work between periods of times.

3.5. Containerization and Orchestration Layer

The realization of applications portability and isolation of services is based on containerization. Packages are data-processing dependencies, which can be standardized and include applications and supporting services, and management parts. This will minimize the variations in the environment and make it easier to implement in the development, testing and production environment.

Kubernetes considers itself to be the orchestration layer and this coordinates the containerized workloads within the cluster. It also reserves containers to free nodes, keeps them healthy, restores components and also enables horizontal scaling. Service discovery, workload isolation and controlled deployment plus configuration management are also offered by Kubernetes.

Spark together with Kubernetes integration allows Spark driver and executor loads to be deployed on a dynamically managed container environment. Whoever requires an additional processing power, new instances of executors can be generated. In case of the demand reduction, the unneeded resources would be freed. This offers an elastic implementation framework overtop of provisioning clusters in a stateless manner.

3.6. Resource Management and Elastic Scaling

The resource allocation which is demand driven is one of the major characteristics of the architecture. Workloads can have a wide range of different computational complexity, memory, input size, and execution time differences. The allocation of the resources subject to static allocation can thus lead to either underutilization or shortage of the resources.

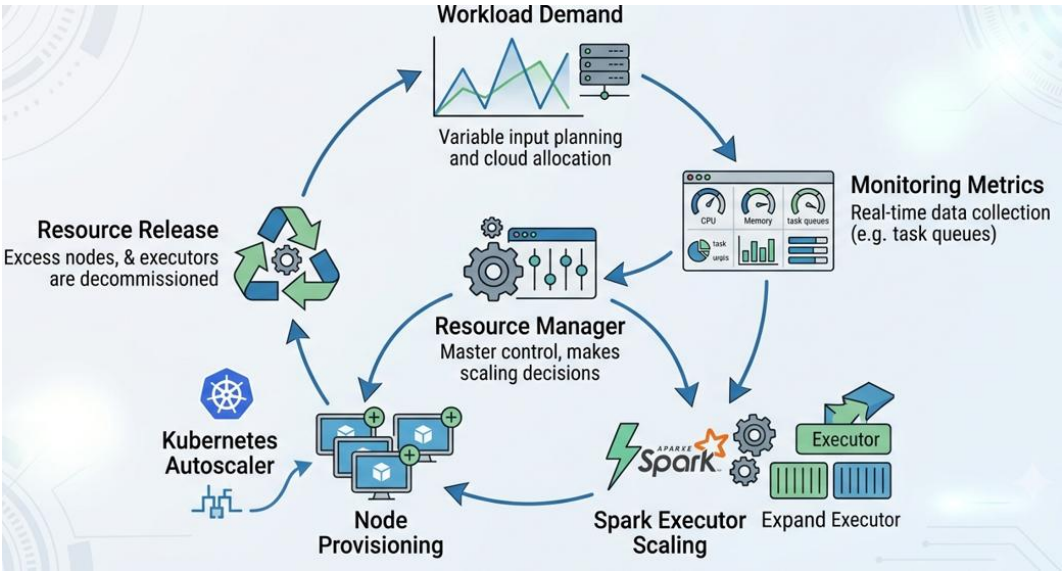


Figure 3 – Elastic Resource Allocation and Auto scaling Loop

The proposed architecture keeps track of the current measurement of the nature of workload and the resource availability of the cluster. Scheduling systems in kubernetes will be able to assign workloads onto the appropriate node and autoscaling systems can be able to add or remove a potential compute resource, based on demand. The resource management can also be utilized at Spark level to define how application is executed in per application basis by which executors are used.

The resources can be allocated based on CPU utilization, memory, queue, and processing latency, throughput and tasks waiting on these resources. Throughput can be maintained by scaling multiple compute resources when there are too many workloads. They can also unleash resources that may decrease the needless infrastructure use when the demand is low.

3.7. Monitoring, Fault Tolerance, and Governance

The infrastructure, application-level behavior is monitored in a special monitoring layer. Some of the CPU and memory usage, executor usage, processing latency, throughput, failed tasks, container restarts, and network usage and storage performance are included in the metrics.

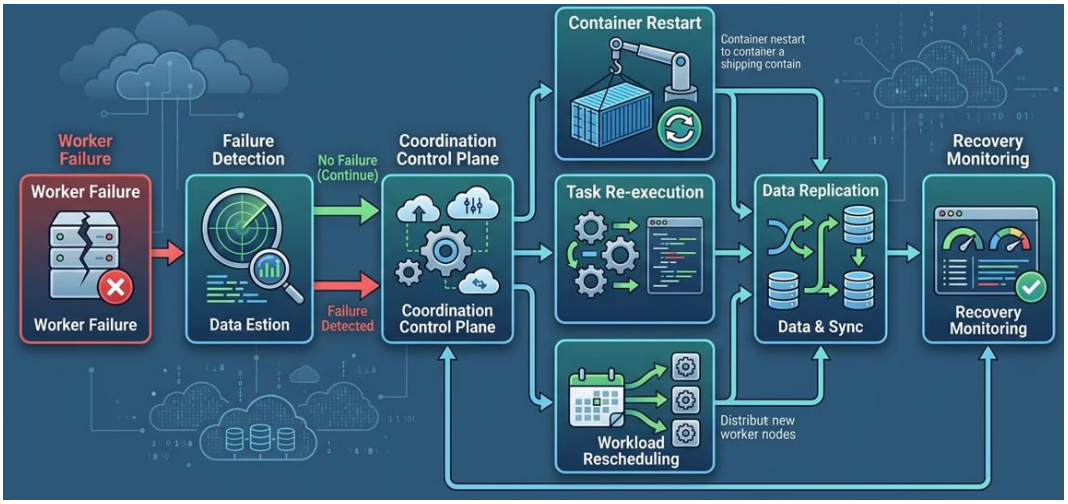


Figure 4 – Fault-Tolerant Cloud-Native Processing Model

The ongoing monitoring allows detecting the decrease in the performance and blocking of resources in time. Fault tolerance is provided in many different ways: Spark tasks restarts, data replication, Kubernetes container restart policies, and restarting failed workloads are only some of these ways. In this way a failure of a single worker or even a container will not lead to a general failure of an application.

The governance component offers access policy control to centralized access, configuration and metadata and logging and auditing. Security can be implemented at the clusters, container and storage levels as well as application level. This is especially true in an enterprise type of environment where two or three or more teams and/or workloads share similar infrastructure.

3.8. End-to-End Dataflow

This entire process of data flows in the starting of a data generation out of heterogeneous sources. The batch and streaming data are entered in the ingestion layer in which they are validated and pre-processed. The distributed storage layer stores the validated data according to their processing level.

The requests are sent to the distributed execution environment that receives them to be processed. Kubernetes requests the necessary containers and Spark distributes calculation over executors that are online. The resource-management layer will be able to track the workload requirements as it continues to execute, and will also re-allocate, where feasible, resources as it is needed. The resulting process is then coded into a preferred refined storage or fed into the next analysis software.

In this process services are continuously monitored by collection of operational and performance measures. The potential bottlenecked processes could be monitored using feedback and scaling or recovery measures taken. In so doing, the architecture creates a feedback process between the demands of the workload, distributed execution, resource allocation and monitoring of the system.

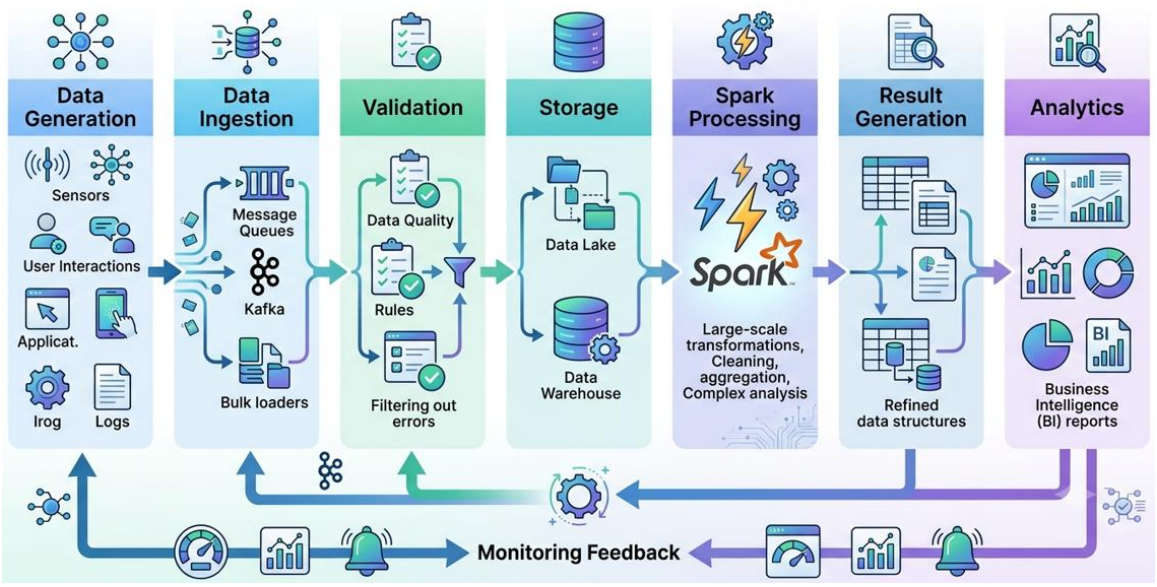


Figure 5 – End-to-End Data Processing Workflow

Overall, the suggested architecture has a mixture of scalability of distributed processing and portability/elasticity of cloud-native infrastructure. Apache Spark, Kubernetes, distributed storage, streaming ingestion, auto- scaling, fault tolerance and monitoring offer a wide range of tools to support the high-throughput workload on data-processing. This architecture can therefore be said to be capable of providing heterogeneous workloads and facilitating better resource usage, resiliency of the operations and scalability.

4. Adaptive Performance Validation and Cloud-Native Scalability Assessment

The suggested framework is discussed in regards to its potential in terms of being able to work with huge workloads and be scaled with high reliability and efficiency levels. Since the dynamically evolving workload nature requires the cloud-native data-processing environments, processing and operational elasticity are measured. A distributed environment of containerization is used to execute representative batch and streaming workloads to test the framework.

4.1. Evaluation Objectives

The key aim is to determine roughly whether the proposed architecture can handle a large data-processing throughput together with how well the computing resources that are dynamically assigned can be utilized. It is evaluated on five major aspects such as throughput, processing latency, scalability, resource utilization and fault tolerance. All these measures will result in the highly detailed assessment compared to that, when the execution time alone is used, and will enable the comparison of the framework with the workload of different levels.

4.2. Experimental Workload and Environment

The analysis takes into consideration non-homogenous datasets of batch and continuously generated and heavy workloads. Illustrated below are the variations of these two in volumes of input to be used and events at different arrival rates in a batch and streaming experiment

respectively: Workloads are loaded into an Apache Spark processing atmosphere with Kubernetes containers.

Experimental context is comprised of a number of worker nodes which are suppliers of the CPU, memory and network resource. Kubernetes controls schedule and application lifecycle of applicants that include containers, and Spark allocates the calculation of jobs, between active executors. This is analyzed with different resource settings in such a way that the effect of elastic resource assignment on the performance of the application is observed.

4.3. Performance Metrics

The most important value of high-throughput ability is the number that is used to measure the amount of information processed in a certain time, which is throughput. An increase in throughput shows that the architecture is able to handle increased workloads without a corresponding increase in the execution time.

Latency is used as a measure of the time interval which is required to acquire and produce some batch/ streaming event. Low latency is sought after in applications which are real-time, and where processing time directly relates to responsiveness.

The use of resource is measured by the CPU and memory usage in worker nodes. In the case of an efficient utilization of the resources, it would mean that the orchestration layer will be capable of providing the resources depending on the workload needs without an unutilized excess capacity.

The scalability test is to examine the processes as regards the size of the work and the processing ability. Theoretically, computing resources must augment corresponding to a similar processing level. Both the scale-out and scale-up behavior are taken into consideration.

4.4. Elasticity and Fault-Tolerance Evaluation

The rate of elasticity is also experimented with the measure of differentiating with the workloads that involve sudden increase and fall of workloads. The reactivity of the Kubernetes based autoscaling and Spark executor allocation to the demand in expanding or contracting resources is observed.

Introducing failures of workers or containers into the process is also used to test fault tolerance. The recovery time, re-execution behavior of the task as well as the overall impact on throughput is reported. The effectiveness of the proposed framework illustrates the robustness of the framework since the proposed framework experienced a successful recovery without necessarily shutting down the applications.

4.5. Expected Evaluation Outcome

Hopefully, the new framework will experience a better throughput, shorter processing latency, better resource utilization and higher elasticity to an increase of the workload. The distributed execution ability of Spark and the Kubernetes orchestration ability will largely likely yield a greater resource dimension of elasticity, compared to fixed clusters. The test thus determines the way the proposed cloud-native can deliver the scalable and robust environment of high throughput distributed data processing.

5. Conclusion

This study describes a cloud-native architecture on which a high-throughput widely distributed data processing system will be built using distributed computing, containers, orchestration, elastic resource management, and continuous monitoring. The proposed architecture applies Apache Spark to process the data simultaneously with Kubernetes to create and schedule containers and dynamically adjust the workloads. This architecture supports full load and streaming loads as well as in stores and segregates the functions of data ingestion, distributed storage, processing, resource administration, and monitoring in the layer architectural layers of scale.

The framework eases some of the inefficiencies of old-fashioned fixed-location distributed-processing environments. Kubernetes-based orchestration can be implemented to feature flexible deployment, workload segregation, fault tolerance, and real-time resource allocation, and Spark can be deployed to offer the distributed execution of large workloads of calculations. Such technologies can be combined to form a software environment that allows the flexible execution of changes in the number of work and processing requirements. The proposed architecture should contain monitoring and governance systems to enhance the optimization of resources, performance, and reliability of operation.

The evaluation framework considers throughput, latency, CPU and memory use, scalability, elasticity, and fault tolerance as key performance indicators. These gauges provide a holistic base for assessing the efficiency and dependability of cloud-native orchestration in enhancing the efficiency and stability of high-throughput distributed data-processing workloads. Typically, the suggested architecture ensures a scalable data-intensive application, which is necessary to be continuously processed, an elastic case, and distributed implementation, which is also very reliable.

The next step of the subsequent research will be the elaboration of the prototype of the proposed construction in its completion and the thorough experimental background on the foundation of real data bases and clouded system. Different workloads will also be considered, and the arrival rates will be considered to determine the relationship between resource allocation, processing throughput, and latency.

The creation of AI-based resource optimization is one of the opportunities. The prediction of workload demand can be provided by machine-learning models to emphasize the assignment of computing resources in advance before the performance decreases. It might also be smart in the scheduling as priority in workload, data proximity, and power usage and application-level service-level target could also be taken into account.

The adoption of multi-clouds and hybrid-clouds to enhance portability, as well as not being bound to a specific cloud provider, will also be discussed in future studies. The next study will focus on serverless and edge-cloud hybrid to support a streaming application with a latency problem. This framework will also incorporate security-minded orchestration, automatic detection of anomalies, and policy-based governance.

Finally, a significant amount of comparison against the other distributed-processing and orchestration configurations will also be conducted to determine how quantitative growth in throughput, scalability, cost effectiveness, and fault recovery has been achieved. These extensions have the potential to convert the proposed architecture to an adaptive, smart, and autonomously controlled platform of next-generation high-throughput data processing.

References

[1]. B. Blamey, F. Wrede, J. Karlsson, *et al*., "Adapting the secretary hiring problem for optimal hot-cold tier placement under top-K workloads," in *Proc. 19th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Comput. (CCGRID)*, Larnaca, Cyprus, 2019, pp. 576–583.

[2]. B. Blamey, A. Hellander, and S. Toor, "Apache Spark Streaming, Kafka and HarmonicIO: A performance benchmark and architecture comparison for enterprise and scientific computing," in *Benchmarking, Measuring, and Optimizing*, Denver, CO, USA, 2019, pp. 1–12, doi: 10.1007/978-3-030-49556-5_30.

[3]. P. Torruangwatthana, H. Wieslander, B. Blamey, *et al*., "HarmonicIO: Scalable data stream processing for scientific datasets," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, San Francisco, CA, USA, 2018, pp. 879–882.

[4]. J. A. Novella, P. Emami Khoonsari, S. Herman, *et al*., "Container-based bioinformatics with Pachyderm," *Bioinformatics*, vol. 35, no. 5, pp. 839–846, 2019.

[5]. B. Blamey, I.-M. Sintorn, A. Hellander, *et al*., "Resource- and message size-aware scheduling of stream processing at the edge with application to realtime microscopy," *arXiv preprint arXiv:1912.09088*, 2019.

[6]. S. Toor, M. Lindberg, I. Falman, *et al*., "SNIC Science Cloud (SSC): A national-scale cloud infrastructure for Swedish academia," in *Proc. IEEE 13th Int. Conf. e-Science*, Auckland, New Zealand, 2017, pp. 219–227.

[7]. Z. D. Stephens, S. Y. Lee, F. Faghri, *et al*., "Big data: Astronomical or genomical?," *PLoS Biology*, vol. 13, no. 7, Art. no. e1002195, 2015.

[8]. U. Sivarajah, M. M. Kamal, Z. Irani, *et al*., "Critical analysis of big data challenges and analytical methods," *Journal of Business Research*, vol. 70, pp. 263–286, 2017.

[9]. D. Rinehart, C. H. Johnson, T. Nguyen, *et al*., "Metabolomic data streaming for biology-dependent data acquisition," *Nature Biotechnology*, vol. 32, no. 6, pp. 524–527, 2014.

[10]. J. Kelleher, M. Lin, C. H. Albach, *et al*., "Htsget: A protocol for securely streaming genomic data," *Bioinformatics*, vol. 35, no. 1, pp. 119–121, 2019.