COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Par

Khaled ARSALANE

Scalable Data Stream Processing in Heterogeneous Environments

Thèse présentée et soutenue à Rennes, le 15 Décembre 2025
Unité de recherche : Institut de Recherche en Informatique et Systèmes Aléatoires - UMR6074

Rapporteurs avant soutenance :

Fabrice HUET — Professeur des Universités, Univ. Nice Sophia Antipolis
Christian PEREZ — Directeur de recherche, Inria

Composition du Jury :

Président :	Martin QUINSON	Professeur des Universités, ENS Rennes
Examinateur·trice·s :	Fabrice HUET	Professeur des Universités, Univ. Nice Sophia Antipolis
	Christian PEREZ	Directeur de recherche, Inria
	Françoise SAILHAN	Professeure, IMT Atlantique
Dir. de thèse :	Guillaume PIERRE	Professeur des Universités, Université de Rennes

Acknowledgement

Ces trois années m'ont semblé à la fois infini et, hélas, être passées bien trop vite !

Le parcours de thèse a été une aventure au cours de laquelle j'ai beaucoup appris : des connaissances scientifiques, des compétences techniques, mais aussi des leçons sur les autres et surtout sur moi-même. Ces enseignements, que je garderai avec moi toute ma vie, sont le fruit de nombreuses rencontres que je tiens à remercier ici.

Je voudrais tout d'abord remercier Guillaume Pierre d'avoir rendu possible le début de cette aventure. J'ai eu la chance d'avoir un directeur de thèse exceptionnel qui a fait preuve d'une grande patience et m'a guidé sur les chemins de la science. En m'acceptant comme doctorant, il m'a fait l'honneur de rejoindre l'équipe Myriads (aujourd'hui Magellan), ce qui m'a permis d'être en contact étroit avec des chercheurs de haut niveau qui m'ont profondément inspiré tout au long de mon parcours. Je tiens tout particulièrement à remercier Shadi Ibrahim pour son aide lors de mes premiers travaux et pour ses encouragements constants. Enfin, je remercie toute l'équipe pour les pauses café, les déjeuners et les autres moments d'échange durant lesquels j'ai appris beaucoup de choses qui m'ont été utiles dans ma recherche. Il me manquera de faire partie de cette équipe.

Le travail de thèse est une confrontation permanente avec l'inconnu et l'échec, ce qui peut peser lourd sur le moral et freiner l'épanouissement personnel. Heureusement, dans les moments les plus difficiles, j'ai toujours pu compter sur le soutien de ma famille. Je remercie profondément mes frères, Zakaria et Yassin, pour leur présence et leur soutien tout au long de mes études. Je tiens à exprimer toute ma gratitude à mes parents, Nora et Brahim, dont les prières, les paroles réconfortantes et l'encouragement constant m'ont soutenu et guidé dans les moments les plus difficiles. Je dédie cette réussite à tous.

Je tiens également à remercier tous mes amis, en Italie, en France et aux quatre coins du monde. Vous avez toujours cru en moi, m'avez soutenu et motivé, même lorsque je m'isolais le plus. Merci pour les sessions de jeux vidéo qui m'ont permis de me détendre, les week-ends escapade qui m'ont redonné de l'énergie, et les messages de soutien à distance. Grâce à vous, je n'ai jamais vraiment été seul. Votre amitié est un trésor que je porte avec moi.

Enfin, je tiens à remercier chaleureusement ma compagne, Sophie, qui m'a soutenu

sans faille durant la majeure partie de mon parcours académique. Ton amour, ta patience et tes encouragements ont été mon pilier. Sans toi, je n'y serais pas arrivé. Cette réussite est aussi la tienne.

Résumé

Les systèmes distribués sont le fondement de l'infrastructure informatique moderne. Ils coordonnent plusieurs nœuds de calcul pour améliorer la capacité de traitement, la fiabilité et la disponibilité. Traditionnellement centralisés dans des centres de données optimisés pour le traitement haute performance, ces systèmes ont évolué vers des architectures géo-distribuées pour répondre aux besoins des organisations opérant à l'échelle mondiale.

Le cloud computing a révolutionné l'accès à ces infrastructures géo-distribuées en proposant un modèle contractuel dans lequel des fournisseurs spécialisés, tels qu'Amazon Web Services, Microsoft Azure et Google Cloud Platform, exploitent de vastes réseaux de centres de données interconnectés. Le paradigme « Tout-comme-un-Service » (XaaS) permet aux organisations de bénéficier de ressources distribuées sans avoir à faire d'investissements considérables ni à disposer d'une expertise technique spécialisée pour les déployer de manière autonome.

L'explosion des applications de l'Internet des objets (IoT) génère aujourd'hui d'importants volumes de données à la périphérie du réseau. Des réseaux de surveillance environnementale aux systèmes de transport intelligents, en passant par la gestion des infrastructures urbaines et les applications de santé connectée, ces domaines génèrent des flux continus de données qui nécessitent un traitement en quasi-temps réel. Cette réalité a donné naissance au continuum edge-cloud, qui étend les infrastructures cloud traditionnelles en intégrant des nœuds de calcul de capacités diverses à travers la hiérarchie du réseau. Cette approche permet de réduire la latence, de diminuer les coûts de bande passante, d'améliorer la confidentialité et d'accroître la résilience opérationnelle.

Cette évolution architecturale introduit une hétérogénéité à plusieurs niveaux.

L'hétérogénéité du réseau se manifeste par des variations de la bande passante, de la latence et de la fiabilité entre les différents segments du continuum, allant des interconnexions haute performance entre centres de données aux liaisons sans fil contraintes en périphérie. Parallèlement, l'hétérogénéité du calcul se traduit par une diversité des capacités de traitement, allant des serveurs multicœurs puissants aux microcontrôleurs à faible consommation d'énergie.

Dans ce contexte, le traitement des flux de données (Data Stream Processing, ou DSP)

émerge comme un paradigme permettant d'analyser les données en continu, à mesure qu'elles arrivent, contrairement au traitement par lots traditionnel qui opère sur des ensembles de données finis. Les systèmes DSP, comme Apache Flink, organisent le calcul sous forme de graphes acycliques dirigés d'opérateurs. On distingue les opérateurs sans état, qui peuvent être facilement parallélisés, des opérateurs avec état, qui maintiennent un contexte sur plusieurs événements, ce qui pose des défis importants dans un environnement hétérogène.

Lorsqu'ils opèrent dans le continuum edge-cloud hétérogène, les systèmes DSP rencontrent des obstacles techniques spécifiques. L'hétérogénéité du réseau crée des déséquilibres dans les flux de données, certains opérateurs manquant de données tandis que d'autres peuvent être submergés. L'hétérogénéité de calcul introduit une variabilité des performances qui complique le placement des opérateurs, l'équilibrage de la charge et les décisions de mise à l'échelle. L'interaction entre ces deux dimensions génère des considérations concurrentes : les contraintes réseau favorisent le traitement local, tandis que les contraintes de calcul incitent à le déporter vers des nœuds plus puissants.

Les approches actuelles d'optimisation DSP supposent souvent des capacités réseau et de calcul simplifiées. Les solutions cloud supposent généralement des ressources homogènes et une bande passante abondante, tandis que les approches edge sont optimisées pour des plateformes spécifiques, sans prendre en compte la diversité de l'hétérogénéité. Cette déconnexion révèle un écart dans la compréhension de l'impact réel de l'hétérogénéité sur les performances globales, en particulier pour les opérations avec état, qui constituent l'aspect le plus complexe du déploiement DSP dans un environnement hétérogène.

L'objectif de ce travail est d'explorer les défis fondamentaux auxquels sont confrontés les systèmes DSP lorsqu'ils opèrent à travers le continuum edge-cloud hétérogène. Nous examinons dans quelle mesure deux dimensions critiques d'hétérogénéité — les caractéristiques du réseau et les capacités de calcul — affectent les performances et la scalabilité des systèmes DSP dans des environnements géo-distribués, en nous concentrant sur les opérateurs de traitement avec état. Si la recherche existante a apporté des éclairages précieux sur l'optimisation DSP pour des environnements spécifiques, notamment les déploiements cloud homogènes, le comportement de systèmes comme Apache Flink lors du déploiement sur le continuum cloud-edge reste mal compris.

Première contribution : Impact de l'hétérogénéité réseau sur la scalabilité des systèmes DSP. Pour comprendre les limitations imposées par l'hétérogénéité du réseau,

cette première contribution examine l'impact des variations de bande passante et de latence entre les différents segments du continuum edge-cloud sur les performances des applications de traitement de flux. Elle révèle les mécanismes par lesquels l'hétérogénéité du réseau crée des goulots d'étranglement et des déséquilibres dans les pipelines de traitement, affectant particulièrement les opérateurs avec état et la fiabilité des signaux d'élasticité traditionnels, comme le backpressure. Nos résultats montrent que les approches traditionnelles d'optimisation DSP, conçues pour des environnements homogènes, ne peuvent pas être appliquées directement dans des contextes géo-distribués hétérogènes, car les stratégies de mise à l'échelle automatique présentent une précision de prédiction réduite pour les charges de travail avec état dans des conditions réseau variables.

Deuxième contribution : Impact de l'hétérogénéité de calcul sur les performances des systèmes DSP. Complémentaire à l'analyse réseau, cette deuxième contribution se concentre sur l'impact de l'hétérogénéité de calcul sur les performances des systèmes DSP. Nous démontrons tout d'abord que l'utilisation optimale des ressources d'une machine unique nécessite de déployer plusieurs TaskManagers (instance de traitement dans l'architecture Apache Flink) avec un seul slot et deux cœurs par slot. Cela permet d'atteindre un débit 3,5 fois supérieur à celui des configurations comportant un seul TaskManager, grâce à une meilleure isolation des ressources au niveau de la machine virtuelle Java (JVM) et à une réduction de la surcharge des opérations de collecte de mémoire (Garbage Collection). Cependant, lors de l'extension à des déploiements distribués hétérogènes mélangeant des ordinateurs aux ressources limitées et des serveurs puissants, nous révélons que les stratégies d'affectation de tâches indépendantes des ressources et de partitionnement de données basées sur le hachage de Flink créent des goulots d'étranglement majeurs. Cette analyse met en évidence le fait que l'hétérogénéité de calcul empêche l'exploitation efficace des serveurs haute performance lors de leur coordination avec des dispositifs à faible puissance, même avec des configurations optimales par machine.

Troisième contribution : Environnement d'évaluation hétérogène pour systèmes DSP. Face à l'absence de plateformes expérimentales réalistes, cette troisième contribution présente la conception et le développement d'un environnement d'évaluation spécialement conçu pour l'étude des systèmes DSP dans des environnements hétérogènes. Nous avons développé un banc d'essai qui reproduit fidèlement les conditions d'hétérogénéité

du réseau et du calcul rencontrées lors des déploiements réels du continuum edge-cloud. Cet environnement permet d'évaluer de manière systématique et reproductible les performances des systèmes DSP, facilitant ainsi la validation expérimentale des optimisations proposées et ouvrant la voie à de futures recherches dans ce domaine.

Ces contributions visent à améliorer la compréhension des dynamiques complexes des systèmes DSP en environnement hétérogène et à fournir les bases théoriques et pratiques nécessaires pour exploiter efficacement les avantages du traitement géo-distribué tout en maîtrisant les défis inhérents à l'hétérogénéité.

Abstract

Distributed systems constitute the backbone of modern computing infrastructure, coordinating multiple computing nodes to achieve greater processing capacity, reliability, and availability. Traditionally centralized in datacenters optimized for high-performance processing, these systems have evolved toward geo-distributed architectures to meet the needs of organizations operating at a global scale.

Cloud computing has revolutionized access to these geo-distributed infrastructures by proposing a contractual model where specialized providers such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform operate vast networks of interconnected datacenters. This "Anything-as-a-Service" (XaaS) paradigm enables organizations to benefit from distributed resources without the considerable investments and specialized technical expertise required for autonomous deployment.

The explosion of Internet of Things (IoT) applications generates unprecedented volumes of data at the network periphery. From environmental monitoring networks to intelligent transportation systems, including urban infrastructure management and connected health applications, these domains produce continuous data streams that require near real-time processing. This reality has given birth to the edge-cloud continuum, which extends traditional cloud infrastructures by integrating computing nodes of diverse capacities across the network hierarchy. This approach enables reduced latency, decreased bandwidth costs, improved privacy, and increased operational resilience.

This architectural evolution introduces inherent heterogeneity at multiple levels. Network heterogeneity manifests through variations in bandwidth, latency, and reliability between continuum segments, from high-performance datacenter interconnections to constrained wireless links at the edge. Simultaneously, compute heterogeneity translates into a diversity of processing capabilities, extending from powerful multicore servers to energy-constrained microcontrollers.

In this context, Data Stream Processing (DSP) emerges as an essential paradigm, enabling continuous analysis of data as it arrives, unlike traditional batch processing that operates on finite datasets. DSP systems such as Apache Flink organize computation as directed acyclic graphs of operators, distinguishing stateless operators, easily paralleliz-

able, from stateful operators, which maintain context across multiple events and pose significant challenges in heterogeneous environments.

When DSP systems operate in the heterogeneous edge-cloud continuum, they encounter specific technical obstacles. Network heterogeneity creates data flow imbalances, with some operators lacking data while others may be overwhelmed. Compute heterogeneity introduces performance variability that complicates operator placement, load balancing, and scaling decisions. The interaction between these two dimensions generates competing considerations: network constraints favour local processing while compute constraints encourage offloading to more powerful nodes.

Current DSP optimization approaches often make simplifying assumptions about network and compute capabilities. Cloud solutions generally assume homogeneous resources and abundant bandwidth, while edge approaches optimize for specific platforms without addressing the complete diversity of heterogeneity. This disconnect reveals a gap in understanding the real impact of heterogeneity on overall performance, particularly for stateful operations that constitute the most complex aspect of DSP deployment in heterogeneous environments.

The fundamental challenge addressed in this work concerns the operation of DSP systems across the heterogeneous edge-cloud continuum. We examine how two critical dimensions of heterogeneity — network characteristics and computational capabilities — affect the performance and scalability of DSP systems in geo-distributed environments, focusing on stateful processing operations. While existing research has provided valuable insights into DSP optimization for specific environments, particularly homogeneous cloud deployments, the behaviour of systems such as Apache Flink when deployed across the cloud-edge continuum remains insufficiently understood.

First Contribution: Impact of Network Heterogeneity on DSP System Scalability. To understand the limitations imposed by network heterogeneity, this first contribution examines the impact of bandwidth, latency, and reliability variations between different segments of the edge-cloud continuum on stream processing application performance. This study reveals the mechanisms by which network heterogeneity creates bottlenecks and imbalances in processing pipelines, particularly affecting stateful operators and the reliability of traditional elasticity signals such as backpressure. Our results show that traditional DSP optimization approaches, designed for homogeneous environments, are not directly applicable in heterogeneous geo-distributed contexts, with auto-scaling

strategies exhibiting reduced prediction accuracy for stateful workloads in variable network conditions.

Second Contribution: Impact of Compute Heterogeneity on DSP System Performance. Complementary to the network analysis, this second contribution focuses on the impact of compute heterogeneity on DSP system performance. We first demonstrate that optimal resource utilization on a single machine requires multiple single-slot TaskManagers (processing instance within Apache Flink's architecture) with dual-core task slots, achieving up to 3.5× better throughput compared to traditional single TaskManager configurations through improved JVM-level resource isolation and reduced garbage collection overhead. However, when extending to heterogeneous distributed deployments mixing resource-constrained edge devices with powerful servers, we reveal that Flink's resource-agnostic task assignment and hash-based data partitioning strategies create fundamental performance bottlenecks. This analysis highlights how architectural heterogeneity prevents effective exploitation of high-performance servers when coordinating with resource-constrained devices, even with optimal per-machine configurations.

Third Contribution: Evaluation Environment for Heterogeneous DSP Systems. Faced with the absence of realistic experimental platforms, this third contribution presents ScaleHub, an infrastructure-as-code testbed orchestration framework specifically designed for studying DSP systems in heterogeneous environments. ScaleHub automatically provisions ephemeral Kubernetes clusters across heterogeneous hardware platforms, from edge devices to cloud servers, enabling systematic evaluation of DSP performance under realistic edge-cloud continuum conditions. The framework integrates heterogeneous network condition emulation and secure mesh networking between geographically dispersed nodes, providing an accessible platform for reproducible experimentation and validation of DSP optimizations in heterogeneous deployments.

These contributions aim to improve understanding of the complex dynamics of DSP systems in heterogeneous environments and provide insights to effectively exploit the advantages of geo-distributed processing while managing the inherent challenges of heterogeneity.

Table of Contents

Résumé 5

Abstract 9

1 Introduction 17
- 1.1 Contributions . . . 21
- 1.2 Publications . . . 22
- 1.3 Thesis Structure . . . 22

2 Background 23
- 2.1 Cloud Computing . . . 23
  - 2.1.1 Cloud Architecture . . . 23
  - 2.1.2 Limitations and Challenges . . . 25
- 2.2 Edge-Cloud Continuum . . . 27
  - 2.2.1 Architecture . . . 27
  - 2.2.2 A heterogeneous infrastructure . . . 29
- 2.3 Data Stream Processing . . . 30
  - 2.3.1 Stream Processing Evolution . . . 31
  - 2.3.2 Apache Flink Architecture . . . 32
  - 2.3.3 Scalability . . . 36
  - 2.3.4 Elasticity . . . 37
- 2.4 Resource Management with Kubernetes . . . 38
  - 2.4.1 Kubernetes Architecture . . . 39
  - 2.4.2 K3s for Edge Computing . . . 40

3 State of the Art 43
- 3.1 DSP Scalability . . . 43
  - 3.1.1 Stateful vs. Stateless Operators . . . 43
  - 3.1.2 Heterogeneous Environments . . . 44
- 3.2 DSP Elasticity . . . 45

3.2.1 Metrics and Objectives . . . 45
3.2.2 Adaptation Strategies . . . 46
3.3 DSP Autoscaling in Heterogeneous Environments . . . 46
3.3.1 Addressing Compute Heterogeneity . . . 46
3.3.2 Addressing Network Heterogeneity . . . 47
3.4 DSP Systems for Heterogeneous Environments . . . 49
3.5 Evaluating DSP in the Edge-Cloud Continuum . . . 50
3.5.1 Simulation Tools . . . 50
3.5.2 Emulation Tools . . . 51
3.5.3 Physical Testbeds . . . 51
3.6 DSP Benchmarks . . . 52
3.6.1 Applications . . . 53
3.6.2 Frameworks . . . 53

4 Data Stream Processing Effectiveness in Heterogeneous Networking Environments 55
4.1 Introduction . . . 55
4.2 Evaluation methodology . . . 56
4.2.1 Experimental setup . . . 56
4.2.2 Experimental Procedure . . . 60
4.3 Evaluation . . . 62
4.3.1 Effects of skewed data distribution on system busyness . . . 63
4.3.2 Flink's scaling behaviour in different latency setups . . . 64
4.3.3 Auto-scaling in heterogeneous network conditions . . . 68
4.4 Discussion . . . 69
4.5 Conclusion . . . 70

5 Data Stream Processing Effectiveness in Heterogeneous Computing Environments 71
5.1 Introduction . . . 71
5.2 Experimental Evaluation Procedure . . . 73
5.2.1 System Design . . . 73
5.2.2 Benchmark Application . . . 74
5.2.3 Workload Injection . . . 75
5.2.4 Experimental Procedure . . . 75

5.3 **RQ1:** Resource configuration . . . 75
5.3.1 CPU and Memory Resources Per Task Manager . . . 75
5.3.2 Throughput Scaling Across Node Types . . . 77
5.3.3 Single-Slot vs Multi-Slot TaskManagers . . . 78
5.4 **RQ2:** Homogeneous vs Heterogeneous computing . . . 83
5.5 Conclusion . . . 84

6 ScaleHub: A testbed on the edge **87**
6.1 Introduction . . . 87
6.2 Design . . . 88
6.2.1 Challenges . . . 88
6.2.2 ScaleHub Architecture . . . 89
6.2.3 Chaos Engineering Integration . . . 92
6.3 Experiment Orchestration and Monitoring . . . 92
6.3.1 Experiment-Monitor Architecture . . . 93
6.3.2 Experiment Lifecycle Management . . . 93
6.3.3 Scaling Operations . . . 94
6.3.4 Data Collection and Analysis . . . 95
6.4 Conclusion . . . 95

7 Conclusion **97**
7.1 Summary . . . 97
7.2 Future directions . . . 98
7.2.1 Reliable Backpressure Mechanisms in Heterogeneous Networks . . . 99
7.2.2 Resource-Aware Stream Processing Frameworks . . . 99
7.2.3 Cross-Platform DSP Evaluation Framework . . . 100

Bibliography **103**

Chapter 1

Introduction

Distributed systems have served as the backbone of modern computing infrastructure for decades with the goal of providing the coordination of multiple computing nodes to achieve greater processing capacity, reliability, and availability [39]. These systems traditionally manifest as datacenters, where computing resources are centralized in facilities optimized for high-throughput processing and dynamic scalability. This centralized model has proven effective for many workloads, offering economies of scale and simplified resource management through homogeneous hardware interconnected with high-bandwidth networks [200].

As organizations expand their digital footprints on a global scale, the necessity for a geographically distributed infrastructure becomes increasingly clear. This demand has led to the emergence of geo-distributed datacenters spanning multiple regions, often separated by significant distances [192, 223]. However, building and maintaining such an infrastructure requires significant capital investment and specialized expertise that most organizations cannot justify or achieve independently. This economic reality gave rise to a transformative business model: cloud computing [27, 43, 229].

Cloud computing fundamentally represents a contractual abstraction over geo-distributed infrastructure, where specialized providers such as Amazon Web Services [8], Microsoft Azure [150], Google Cloud Platform [88], and OVH Cloud [162] operate vast networks of interconnected datacenters and offer their capabilities as services to other organizations. This paradigm shift embodies the "Anything-as-a-Service" model [184], where computational resources, storage, networking, or even complete application platforms are delivered through Service Level Agreements (SLAs) that guarantee specific performance, availability, and reliability metrics [44]. Cloud providers essentially democratize access to geo-distributed systems by handling the complexities of infrastructure management, allowing organizations to focus on their core business logic rather than the underlying distributed system operations.

The cloud model delivers several critical advantages that individual user organizations

would struggle to achieve independently. Geographic distribution provides improved disaster resilience through redundancy across regions [63] and enables regulatory compliance through data sovereignty—ensuring data remains within specific jurisdictions as required by regulations such as GDPR [31, 78]. For end-users, this geographic distribution also translates to enhanced experience through reduced latency, as cloud providers can process requests at datacenters closest to users' locations. Furthermore, the economic efficiency of shared infrastructure allows cloud providers to offer sophisticated capabilities—from machine learning services to global content delivery networks—at costs far below what individual organizations could achieve through private infrastructure [90, 231].

The explosive growth of IoT applications has led to unprecedented volumes of data being generated at the network periphery, spanning a wide range of domains [4, 32]. For instance, Environmental monitoring networks deploy weather stations, air quality monitors, and water quality sensors to enable timely detection of pollution events, climate anomalies, and natural hazards [204]. Urban infrastructure management leverages connected streetlights, waste management systems, and public utility networks to optimize resource allocation and maintenance scheduling [46]. Transportation systems process vehicle telemetry, traffic camera feeds, and passenger movement data for traffic flow optimization and predictive maintenance [227]. Healthcare applications rely on patient monitoring devices and physiological sensors to provide continuous observation and prompt alerts to medical personnel [118]. Industrial automation employs equipment sensors, production line monitoring, and quality control systems to maintain optimal manufacturing parameters and detect failures before they cause downtime [191].

To address the challenges these applications face when relying solely on centralized cloud processing, the edge-cloud continuum has emerged as a paradigm that extends traditional cloud infrastructures [152, 224]. Rather than depending exclusively on datacenter resources, this approach integrates computing nodes of various sizes and capabilities throughout the network hierarchy [87, 112, 176]. By positioning computing resources closer to data sources, the edge-cloud continuum delivers significant benefits: reduced latency for time-sensitive operations, decreased bandwidth costs through local data filtering and aggregation, enhanced privacy through localized processing of sensitive information, and improved operational resilience when cloud connectivity becomes unreliable [37, 221].

The proliferation of IoT devices and the architectural evolution towards edge computing results in a landscape with a variety of network technologies that are predominantly wireless, including 5G/LTE, Wi-Fi, and LPWAN [34]. These technologies coexist alongside

a diverse array of hardware platforms, ranging from mini computers such as NUC, Raspberry Pi to edge-dedicated cluster boards such as Turing Pi [167, 195]. This architectural evolution gives rise to inherent heterogeneity at multiple levels. Network heterogeneity manifests as variations in bandwidth, latency, and reliability characteristics across different segments of the continuum—from high-performance datacenter interconnects to constrained wireless links at the edge. Simultaneously, heterogeneity in computation is characterized by a wide range of processing capabilities, extending from powerful multi-core servers in datacenters to energy-constrained microcontrollers in edge devices [111]. Together, these dimensions of heterogeneity create a complex landscape that challenges traditional approaches to distributed application design and deployment.

Despite the increasing complexity and heterogeneity of modern distributed environments, their core goal remains unchanged: processing data to extract value. Historically, batch processing dominated this landscape, operating on finite datasets to produce analytical results on demand. Systems such as Hadoop MapReduce championed this approach, processing large volumes of data through discrete jobs with defined start and end times [73, 76, 188]. As data velocity increases and the need for immediate analysis grows, stream processing emerges as a natural progression from traditional batch processing [48, 172]. Rather than waiting for complete datasets, stream processing systems operate continuously on data as they arrive, enabling time-sensitive applications that respond to information moments after generation.

Data Stream Processing (DSP) systems have evolved significantly to meet these demands, progressively decomposing their architectures into finer-grained components that can be easily deployed and scaled across distributed environments. This evolution has produced several generations of systems, from early frameworks such as Storm [21] to more sophisticated platforms such as Spark Streaming [20] and Samza [19]. Among these, Apache Flink has emerged as a leading platform [15], offering a comprehensive feature set including exactly-once processing guarantees, sophisticated state management, and native support for event-time processing—capabilities particularly valuable in distributed environments where consistency and timing are challenging to maintain.

At its core, stream processing in systems such as Flink organizes computation as a directed acyclic graph (DAG) of operators. Each operator is applying specific transformations on data streams such as filtering irrelevant data, aggregating values over time windows, joining multiple streams, and triggering external actions based on detected patterns. These operators fall into two categories: stateless operators, which process each data

item independently, and stateful operators, which maintain context information across multiple events. While stateless operators can scale relatively easily through simple parallelization, stateful operators present significant challenges—particularly in heterogeneous environments—as their internal state must be consistent when scaling or relocating processing [47, 185].

When DSP systems operate across heterogeneous environments, they encounter specific technical obstacles. Consider a stream processing pipeline spanning from edge devices through fog nodes to cloud infrastructure. Network heterogeneity introduces variable data transfer rates between processing stages—while cloud-to-cloud transfers might sustain hundreds of megabits per second, edge-to-cloud connections might support only a fraction of that capacity. These differences create imbalances where downstream operators either starve for input or become overwhelmed with backlogged data.

Similarly, compute heterogeneity introduces performance variability between operator instances that affect operator execution. The same filtering or aggregation logic that executes in milliseconds on cloud servers might require seconds on resource-constrained edge devices. This inconsistency complicates fundamental aspects of DSP systems: operator placement, load balancing, and scaling decisions all become multidimensional optimization problems that must account for the underlying heterogeneity.

The interplay between network and compute heterogeneity further compounds these challenges. Network constraints might suggest processing data close to its source, while compute constraints might favour offloading processing to more powerful nodes. Resolving these competing considerations requires nuanced approaches that consider both dimensions simultaneously. Moreover, as heterogeneity patterns change—through network congestion, hardware upgrades, or deployment changes—static optimization approaches quickly become suboptimal.

Current approaches to DSP optimization often make simplifying assumptions about network and compute capabilities [51]. On the one hand, Cloud-based solutions typically presume abundant network bandwidth and homogeneous compute resources. On the other hand, Edge-focused approaches often optimize for specific hardware platforms without addressing the full spectrum of heterogeneity [138, 157]. This disconnect reveals a gap in understanding how different forms and degrees of heterogeneity affect overall system performance, especially during rescaling operations.

This thesis addresses the fundamental challenges that arise when Data Stream Processing systems operate across the heterogeneous edge-cloud continuum. Specifically, we

examine how the two critical dimensions of heterogeneity — network characteristics and computational capabilities — affect the performance and scalability of DSP systems when deployed across geo-distributed environments. Our focus centers on stateful stream processing operations, which represent the most challenging aspect of DSP deployment in heterogeneous settings due to their requirement for consistent state management across diverse computing nodes.

While existing research in this domain has produced valuable insights into DSP optimization for specific environments, particularly homogeneous cloud deployments, the reality of modern distributed applications involves operation across the edge-to-cloud continuum with its inherent heterogeneity. The confluence of IoT applications leveraging edge-cloud architectures and the evolution of DSP systems raises critical questions about performance characteristics in these heterogeneous, geo-distributed environments. While systems such as Flink demonstrate excellent performance in homogeneous datacenters, their behaviour when deployed across the cloud-to-edge continuum remains insufficiently understood, particularly regarding the complex interactions between network constraints, compute diversity, and stateful processing requirements [157]. By understanding these dynamics, we can enable future approaches that effectively harness the benefits of geo-distributed processing while managing the inherent challenges of heterogeneity.

1.1 Contributions

This thesis presents a comprehensive study on the challenges in Data Stream Processing (DSP) systems in heterogeneous environments. The contributions of this thesis are threefold:

— **Impact of Network Heterogeneity:** We analyse how network heterogeneity affects the scalability of DSP systems, providing insights into the design and optimization of such systems in diverse network conditions.

— **Impact of Compute Heterogeneity:** We investigate the performance implications of compute heterogeneity on DSP systems, offering strategies to enhance their efficiency and adaptability in varied computational landscapes.

— **A Testbed for Heterogeneous DSP Systems:** We introduce a testbed that allows the evaluation of DSP systems in heterogeneous environments, addressing the need for a controlled and repeatable environment to study the impact of network and compute heterogeneity.

Through these contributions, we aim to advance the understanding and capabilities of DSP systems in the context of modern compute continuum and Big Data.

1.2 Publications

This thesis is based on the following publications:

1. Arsalane, Khaled and Pierre, Guillaume and Ibrahim, Shadi **Toward Stream Processing Elasticity in Realistic Geo-Distributed Environments.** In Proceedings of IEEE IC2E (2024).
2. Arsalane, Khaled and Pierre, Guillaume **Data Stream Processing Effectiveness in Heterogeneous Computing Environments.** In Proceedings of ACM SAC (2026).

1.3 Thesis Structure

The thesis is structured as follows: Chapter 2 provides the necessary background on Data Stream Processing (DSP) systems, cloud computing, and the challenges posed by network and compute heterogeneity. Chapter 3 reviews related work in the field of DSP optimization, focusing on existing approaches to address the challenges of heterogeneity in distributed systems. Chapter 4 presents the first contribution, focusing on the impact of network heterogeneity on DSP systems. Chapter 5 discusses the second contribution, which examines the effects of compute heterogeneity on DSP systems. Chapter 6 details the testbed architecture designed for evaluating DSP systems in heterogeneous environments. Finally, Chapter 7 concludes the thesis, summarizing the findings and discussing future work.

Chapter 2

Background

In this chapter, we provide the necessary background to understand the context of scalable data stream processing in heterogeneous geo-distributed environments. First, we introduce the cloud computing model and its extension to the edge to form the edge-cloud continuum. We then discuss the evolution of data stream processing systems, focusing on Apache Flink as a representative platform for distributed stream processing. Finally, we present the challenges associated with heterogeneous infrastructures and resource management in edge-cloud environments, setting the stage for the contributions presented in later chapters.

2.1 Cloud Computing

Cloud computing has transformed the way organizations deploy and manage computational resources. This paradigm represents a shift from traditional on-premises infrastructure to a model where computing resources are provided as services over the Internet. The evolution of cloud computing has been driven by the need for scalable, flexible, and cost-effective computing solutions that can adapt to varying workloads and business requirements [80].

2.1.1 Cloud Architecture

Cloud computing infrastructure is characterized by large-scale datacenters containing thousands of servers, storage systems, and networking equipment. As presented in Figure 2.1, cloud architectures implement a hierarchical multi-layer design with physical infrastructure forming the foundation, virtualization layers providing resource abstraction, and service delivery platforms positioned at the application level. The physical layer comprises servers, storage arrays, and networking equipment organized in rack-mounted

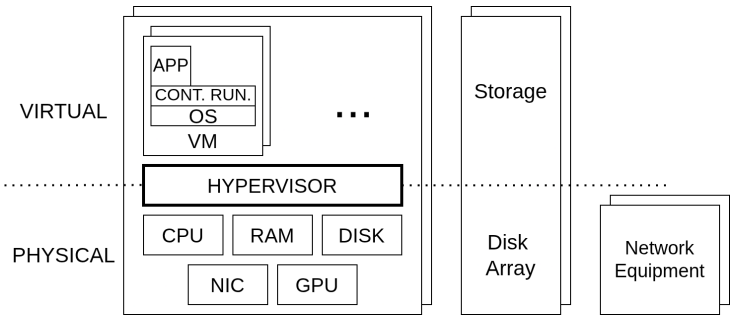


Figure 2.1 – Cloud computing architecture showing the layered approach from physical infrastructure through virtualization to service delivery.

configurations within datacenter facilities. These components are interconnected via high-bandwidth networks engineered for low-latency communication and fault tolerance [144].

Virtualization technologies constitute the fundamental abstraction layer of cloud infrastructure, facilitating the transformation of physical resources into logical units that support independent provisioning, management, and scaling operations. Hypervisor technologies including VMware ESXi, KVM, and Xen instantiate virtual machines, virtual storage systems, and software-defined networks that share underlying hardware resources while maintaining process isolation [38]. Container technologies such as Docker implement lightweight virtualization through operating system kernel sharing while preserving application process isolation [159].

As presented in Figure 2.2, major cloud service providers including Amazon Web Services (AWS), Oracle, Microsoft Azure, Google Cloud Platform (GCP), and OVH Cloud have deployed extensive global networks of interconnected datacenters across Europe and worldwide [8, 88, 150, 160, 162]. These infrastructures consist of large-scale facilities strategically positioned in regions with optimal conditions for large-scale computing operations, including access to robust network connectivity, renewable energy sources, and favorable thermal environments [173]. Beyond fundamental computing and storage capabilities, these providers deliver services including machine learning platforms, database management systems, content delivery networks, and domain-specific solutions. Through the establishment of geographically distributed availability zones, cloud providers deliver computing capabilities at costs that individual organizations would find challenging to achieve through on-premise infrastructures [90].

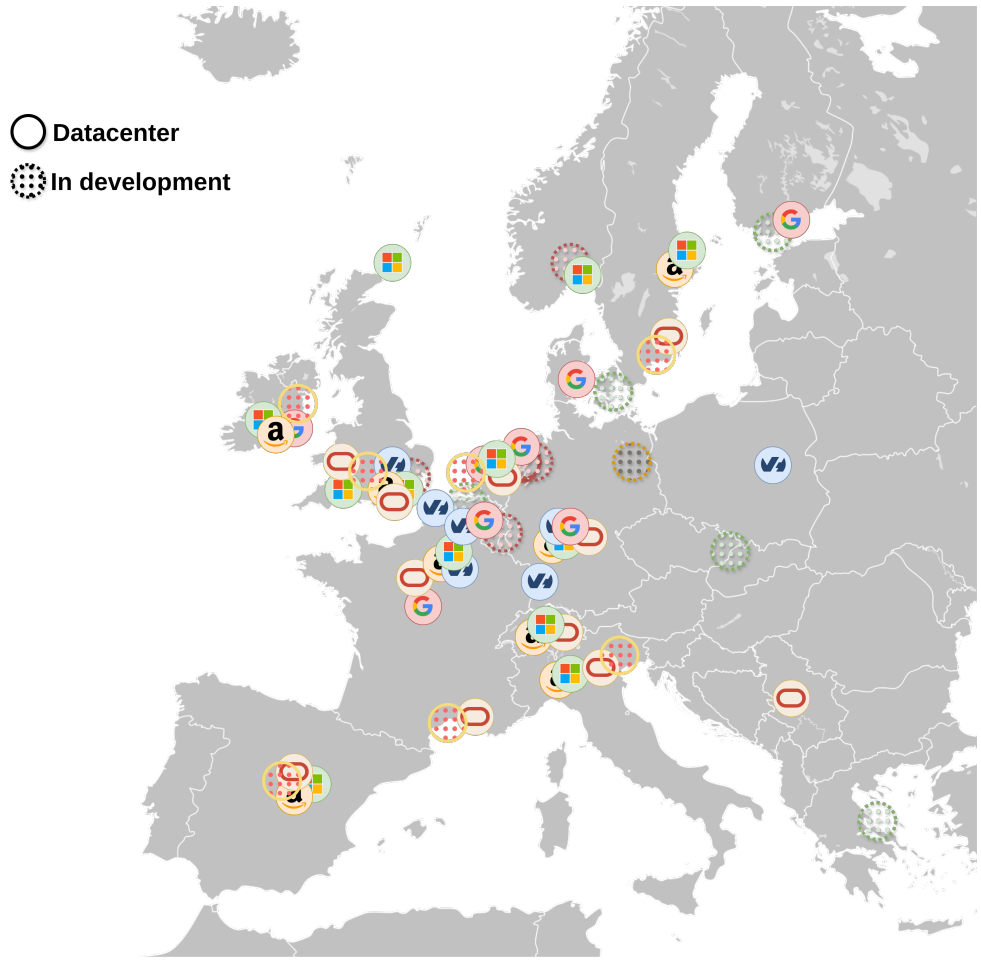


Figure 2.2 – Distribution of cloud datacenters in Europe, showing the geographical spread of major cloud providers.

2.1.2 Limitations and Challenges

Cloud datacenter distribution follows strategic patterns that prioritize regions with robust network infrastructure and access to cost-effective energy sources [173]. While this approach is suitable for most large-scale computing operations, it introduces significant limitations that become increasingly problematic with the proliferation of Internet of Things (IoT) devices and applications. Modern IoT use cases cannot rely exclusively on massive datacenters when data generation occurs at geographically distributed locations, often far from traditional cloud facilities. These applications generate significant data volumes requiring real-time processing, analysis, and response capabilities [4]. The high-velocity nature of these data, combined with the need to extract actionable insights and

trigger timely responses, requires prompt processing and analysis capabilities. Critical applications such as autonomous vehicle systems and remote environmental monitoring stations are examples of scenarios where excessive processing delays can compromise system effectiveness or safety [118, 140, 204, 219, 227].

The geographic separation between data sources and cloud facilities introduces significant challenges for direct IoT device connectivity to cloud datacenters. As IoT deployments scale, fundamental limitations of the traditional cloud model become clear across three primary dimensions: bandwidth consumption, latency constraints, and privacy considerations [187].

Bandwidth limitations emerge as many sensors and IoT devices generate continuous data streams that must be transmitted to remote cloud facilities. This massive data movement creates significant network traffic, possibly leading to congestion and increased operational costs. The challenge intensifies in environments with limited or expensive connectivity, where bandwidth resources represent a significant constraint [42].

Latency constraints arise from the physical distance between data sources and cloud processing facilities as well as physical network topology. Time-critical applications including industrial control systems, autonomous vehicles, and interactive media applications require processing responses that may be compromised by propagation delays inherent in remote cloud processing. Even with optimized network paths, fundamental physical limitations impose minimum latency bounds based on geographic distance.

Privacy and regulatory compliance requirements present additional deployment challenges for cloud-based systems. Data sovereignty regulations in many jurisdictions mandate that specific data types remain within defined geographic boundaries. Healthcare, financial, and personal data are often subject to stringent regulations governing storage location and processing jurisdiction [78]. These regulatory requirements often conflict with the globally distributed, location-agnostic characteristics of traditional cloud services [31].

These fundamental limitations have led to the evolution of cloud computing beyond traditional datacenter models, driving the development of distributed approaches that extend cloud to the network edge. This evolutionary process has resulted in the emergence of the edge-cloud continuum, which addresses these challenges by positioning additional computational resources close to data sources and end-users [36].

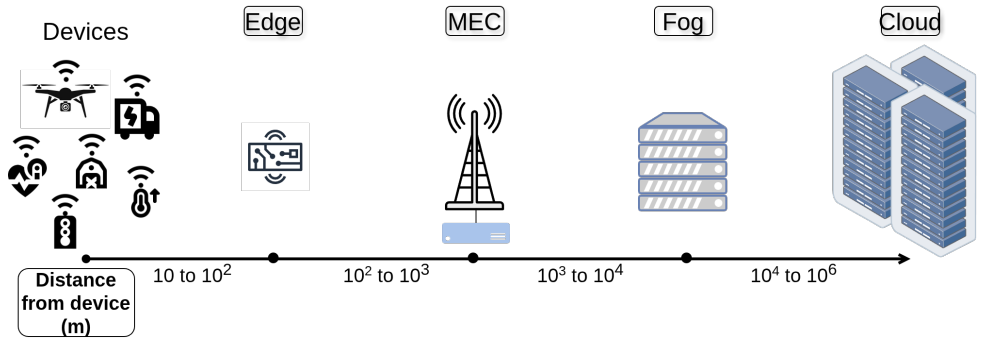


Figure 2.3 – Comparison of different edge-cloud architectural approaches including edge computing, fog computing, and multi-access edge computing, showing their positions in the network hierarchy.

2.2 Edge-Cloud Continuum

The edge-cloud continuum represents an architectural evolution that extends traditional cloud infrastructure to include computing resources positioned closer to data sources and end users [123, 187]. This paradigm has emerged in response to the limitations of cloud processing for certain applications, particularly those requiring low latency, bandwidth efficiency, or operation in environments with unreliable connectivity to central cloud resources [5, 124].

2.2.1 Architecture

The edge-cloud continuum consists of several architectural approaches, each positioning computing resources at different points in the network hierarchy to address specific requirements and constraints. Figure 2.3 illustrates the key architectural approaches within the edge-cloud continuum, including edge computing, fog computing, and multi-access edge computing (MEC).

Edge computing represents the most distributed approach, placing computing resources directly at or near data sources and end-users. Edge nodes may be embedded in devices such as industrial controllers, connected vehicles, and smart cameras. These resources typically have limited capabilities compared to cloud infrastructure but offer minimal latency for local processing. Edge computing architectures often implement local decision-making with limited or periodic synchronization to central systems [176].

Multi-access-Edge Computing (MEC) positions computing resources within the radio access network, which is particularly valuable for mobile applications requiring low latency.

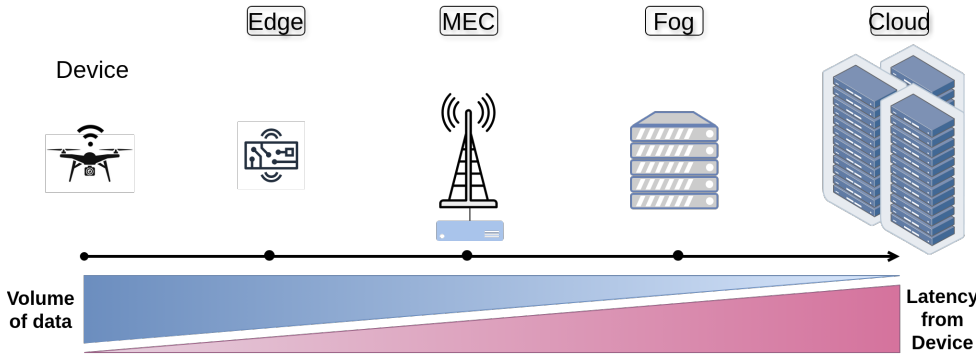


Figure 2.4 – Volume of data generated, processed, and transmitted at different layers of the edge-cloud continuum.

MEC nodes are typically deployed at cellular base stations or network aggregation points, providing cloud-like capabilities with significantly reduced latency for mobile users. This architecture is particularly relevant for applications such as augmented reality, connected vehicles, and mobile gaming [141].

Fog computing extends cloud capabilities to the edge of the network, creating a multi-tiered architecture where processing can occur at different levels based on application requirements. Fog nodes are typically positioned at intermediate points in the network hierarchy, such as in cellular base stations, local servers, or network gateways. This approach provides a balance between the latency advantages of edge computing and the processing power of cloud resources. Fog architectures often implement hierarchical data processing, with initial filtering and analysis at the edge, intermediate aggregation in fog nodes, and complex analytics in the cloud [221].

Figure 2.4 illustrates the benefits of the edge-cloud continuum in terms of data volume transmissions. The traditional cloud layer, while still essential for complex analytics and long-term storage, only receives the relevant fraction of the raw data generated at the edge. The layers in closer proximity to the data source (edge and fog) have the capacity to manage a significant proportion of data processing, thereby reducing the volume of data that must be transmitted. By handling significant data processing closer to the source, the architecture reduces exposure to increased network latency when transmitting data to the cloud [125].

2.2.2 A heterogeneous infrastructure

The integration of edge, fog, and cloud computing creates a distributed environment characterized by significant heterogeneity across multiple dimensions [126, 176]. Heterogeneity in the continuum manifests primarily through variations in compute resources and network characteristics, creating an environment that applications must be able to adapt to. This heterogeneity introduces both opportunities and challenges for applications deployed across the continuum.

Compute Resources: Compute heterogeneity spans variations in processing capabilities, memory capacity, and storage characteristics. Processing power ranges from multi-core, high-frequency CPUs and GPUs in datacenters to energy-constrained processors in edge devices. This variation affects not only raw computational throughput but also the types of operations that can be efficiently executed on different nodes [7].

Memory capacity follows a similar pattern, ranging from hundreds of gigabytes in cloud servers to megabytes in constrained-edge devices, impacting the ability to maintain state and cache data. Storage systems exhibit heterogeneity in capacity, throughput, and access patterns, from high-performance SSDs and distributed file systems in datacenters to limited local storage in edge devices [167].

Network Characteristics: Network heterogeneity represents another critical dimension in edge-cloud environments. Bandwidth availability ranges from high-capacity datacenter interconnects (often exceeding 100 Gbps) to constrained wireless links at the edge (potentially limited to kilobits per second) [133, 214]. Latency patterns differ markedly, from sub-millisecond communication within datacenters to potential seconds for edge-to-cloud communication, particularly in mobile or satellite networks [7, 34].

Network reliability also varies across the continuum. Datacenter networks typically offer high reliability with redundant paths and controlled environments [113]. In contrast, edge networks—particularly those involving wireless technologies—may experience intermittent connectivity, packet loss, and variable performance due to environmental factors or mobility [137, 216].

Energy Constraints and Physical Positioning: Edge nodes deployed in remote or specialized environments face challenges related to energy consumption and physical positioning. Nodes located in natural environments and remote industrial facilities often

operate with limited power budgets, sometimes relying on battery power or renewable energy sources with variable output. These constraints require energy-aware computation strategies that balance processing requirements against available power [106, 136].

The physical positioning of edge nodes introduces additional challenges related to environmental conditions, maintenance access, and physical security. Limited physical access for maintenance requires robust remote management capabilities and fault-tolerant designs that can continue functioning despite component failures [121, 167].

Data Processing Implications: The heterogeneity of the edge-cloud infrastructure has strong implications for data processing applications. Traditional approaches designed for homogeneous datacenter environments often make assumptions about resource availability and network characteristics that do not hold in heterogeneous settings.

As data volumes increase and processing requirements grow more complex, applications must adapt to leverage the full spectrum of available resources while accounting for their diverse capabilities [32, 222]. This adaptation requires approaches to resource allocation, workload distribution, and data movement optimization. Particularly for data-intensive applications, the heterogeneous infrastructure creates both constraints and opportunities. While resource limitations at the edge may restrict complex processing, the proximity to data sources enables filtering and preprocessing that can reduce bandwidth requirements and improve overall system efficiency.

These considerations are especially relevant for continuous data processing applications that operate on streams rather than static datasets. Such applications must consider the tradeoffs between processing location, data movement costs, and computational capabilities across the heterogeneous continuum.

2.3 Data Stream Processing

Data Stream Processing (DSP) systems process data as they arrive, enabling time-sensitive applications to extract value from information moments after generation. These systems have evolved since the 1990's to meet the demands of continuous, high-velocity data analysis.

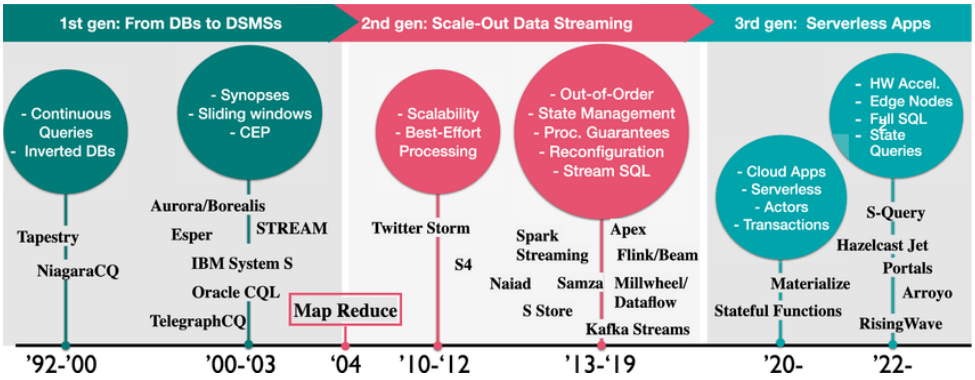


Figure 2.5 – Evolution of stream processing systems from batch processing to modern streaming architectures, highlighting key capabilities of each generation. Adapted from [48, 81].

2.3.1 Stream Processing Evolution

Stream processing emerged as a natural progression from traditional batch processing, driven by the need to analyse data in real time rather than in discrete jobs with defined start and end times. Early stream processing systems often adapted batch processing concepts to handle continuous data flows, with limited guarantees regarding consistency and fault tolerance [172].

As illustrated in Figure 2.5, the evolution of stream processing systems over the past three decades has been characterized by three distinct waves of technological development. The first generation, originating from database research in the early 2000s, established fundamental concepts through prototypical systems that focused on streaming queries and window operations but operated primarily on scale-up architectures with ordered event streams. The following generation, influenced by the rise of MapReduce and cloud computing, shifted towards distributed, data-parallel processing engines with shared-nothing architectures, introducing systems that handled massive out-of-order streams through fault-tolerant, scale-out processing [73]. Finally, the current generation represents a convergence towards unified processing models that support both streaming and batch computations, with stream processors becoming foundational components for event-driven architectures and large-scale continuous analytics.

This evolution reflects changing requirements and understanding of stream processing challenges. Modern systems must handle not only high data volumes but also complex event time semantics, stateful operations, and exactly-once processing guarantees—all while maintaining low latency and high throughput and tolerating device failures.

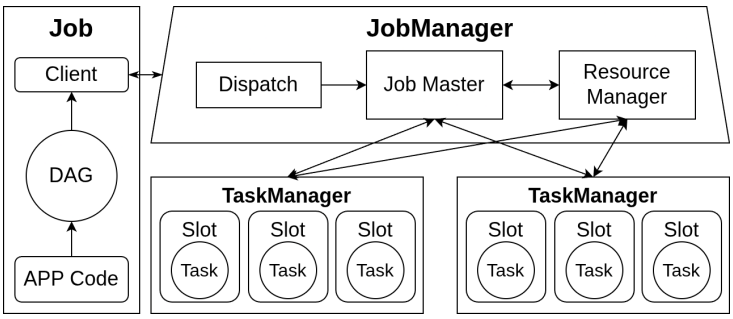


Figure 2.6 – Apache Flink architecture showing the relationship between JobManager, TaskManagers, and the execution of dataflow graphs across distributed resources.

2.3.2 Apache Flink Architecture

Apache Flink has emerged as a leading stream processing platform, arriving at the end of the second generation of scale-out data streaming and offering a comprehensive feature set particularly valuable in distributed environments. Its architecture is designed specifically for continuous processing of unbounded data streams with strong consistency guarantees [107]. Flink is implemented in Java and runs on the Java Virtual Machine (JVM), which influences its memory management and resource allocation strategies. We use Flink as a reference Stream Processing system for the contributions presented in Chapters 4 and 5.

Components: TaskManager and JobManager

Figure 2.6 illustrates the architecture of Apache Flink, which is designed to support distributed stream processing across clusters of machines. Flink's architecture consists of two main types of processes: JobManagers and TaskManagers. The JobManager serves as the central coordinator for distributed execution. It receives Flink applications, transforms them into executable dataflow graphs, and coordinates their execution across available TaskManagers. The JobManager is responsible for scheduling tasks, coordinating checkpoints for fault tolerance, and handling recovery from failures [47, 110].

TaskManagers are the worker processes that execute the actual data processing operations. Each TaskManager provides a set of processing slots, which are the basic unit of resource allocation in Flink. A TaskManager may execute one or more subtasks in parallel threads, with each subtask processing a partition of the data stream. TaskManagers communicate with each other to exchange data during stream shuffles and with the JobManager for coordination and status reporting [10].

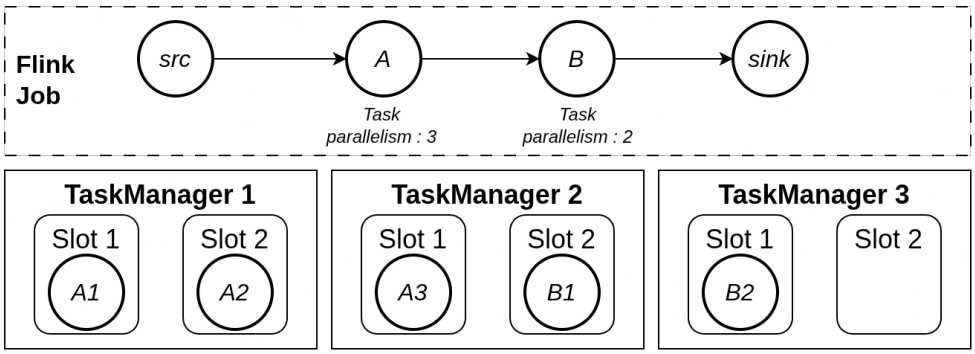


Figure 2.7 – Subtask allocation in multislot TaskManagers in Apache Flink, showing how multiple subtasks can run concurrently within a single TaskManager instance.

This separation of coordination and execution responsibilities enables Flink to scale horizontally by adding more TaskManagers as processing requirements increase. It also provides a clear failure domain, allowing the system to recover from worker failures without losing processing state [21].

Task Slots and Resource Management

Figure 2.7 illustrates the slot-based resource management model used by Apache Flink. Task slots represent the basic unit of resource allocation in Flink. Each slot can execute one single-threaded subtask at a time, and the number of slots per TaskManager determines its parallelism capacity. This slot-based resource model allows fine-grained control over how processing resources are allocated to different parts of a streaming application [13,14].

Flink's resource management integrates with cluster managers such as YARN, Kubernetes, and Mesos. These integrations allow Flink to request and release resources dynamically based on application requirements. When deployed on Kubernetes, Flink can leverage the platform's resource management capabilities to ensure appropriate allocation of CPU, memory, and other resources to TaskManagers [12]. Figure 2.8 illustrates how TaskManagers can be allocated resources in a Kubernetes cluster, with each TaskManager instance receiving a defined amount of CPU and memory resources. These resources are then divided between the available task slots of each TaskManager, allowing multiple subtasks to run concurrently within a single TaskManager instance.

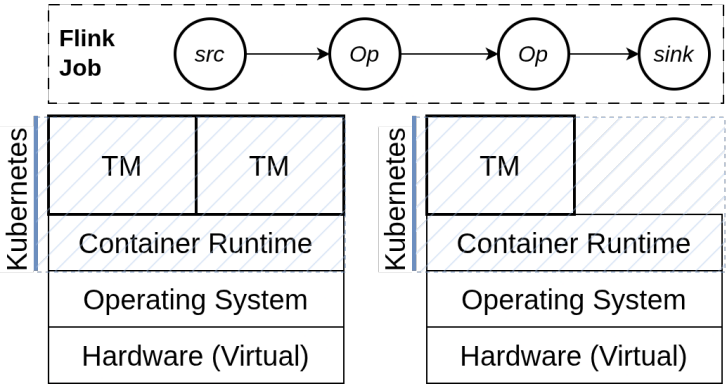


Figure 2.8 – TaskMager resource allocation in Kubenetes.

Operators: Stateless vs Stateful

Flink applications are composed of operators arranged in a directed acyclic graph (DAG), with each operator performing specific transformations on data streams. These operators fall into two categories: stateless and stateful, each with distinct characteristics and challenges [16].

Stateless operators process each data item independently, without maintaining context across multiple events. Examples include filtering operations which select items based on predefined criteria and map operations which transform individual records. These operators can scale relatively easily through simple parallelization, as each instance can process any subset of the data stream without coordination with other instances [19].

Stateful operators, conversely, maintain context information across multiple events. Examples include windowed aggregations which compute metrics over time intervals and join operations which combine data from multiple streams based on keys [18]. These operators present significant challenges in distributed environments, as their internal state must be consistently maintained when scaling or relocating processing.

Flink addresses these challenges through its state backend architecture, which provides mechanisms for state persistence, consistent state snapshots, and state restoration during recovery [11,47]. For applications with large state requirements, Flink supports RocksDB as an embedded state backend that stores state off-heap in local disk storage rather than in JVM memory [174]. RocksDB enables Flink applications to maintain state larger than available heap memory while providing predictable latency characteristics unaffected by garbage collection, making it particularly suitable for long-running windows and large keyed state operations.

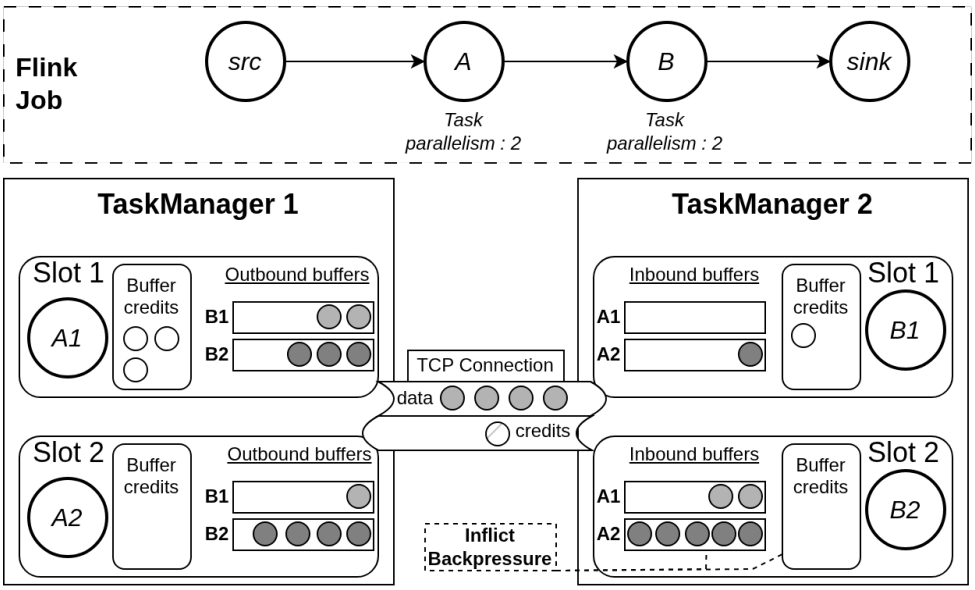


Figure 2.9 – Backpressure mechanism in Apache Flink, showing how downstream operator overload propagates upstream through credit-based flow control to regulate data flow rates.

The distinction between stateless and stateful operators is particularly important in heterogeneous environments, where varying compute and network capabilities affect how efficiently different types of operations can be executed and how state can be maintained across distributed nodes [185].

Overload Management

When data flows through a DAG of stream processing operators, there can be situations where downstream operators cannot keep pace with upstream data production. In Apache Flink, the backpressure mechanism regulates the data flow and prevents out-of-memory conditions by slowing down the upstream operators when needed [54].

Figure 2.9 illustrates Flink's backpressure mechanism in action. When a downstream operator becomes overwhelmed due to high data rates or complex processing requirements, it cannot consume data at the rate it arrives. This creates a bottleneck that propagates backwards through the operator chain. As shown in the figure, Flink's network stack implements backpressure through its credit-based flow control mechanism, where downstream tasks control the rate at which upstream tasks send data by managing network buffer availability [131, 132].

The slot-based resource model facilitates effective backpressure handling by providing

clear boundaries for resource consumption within each TaskManager. When backpressure occurs, the affected slots naturally throttle their data consumption, allowing the system to self-regulate without external intervention. This approach maintains system stability and prevents cascading failures that could result from overfull input data buffers [17].

Flink provides metrics and monitoring capabilities that allow operators to identify when backpressure occurs and trace its propagation through the processing topology.

2.3.3 Scalability

While backpressure mechanisms assist in the management of temporary overload scenarios by regulating data flow, scalability offers a more comprehensive solution by augmenting the system's processing capacity.

Scalability refers to the ability of a stream processing system to handle increasing workloads by adding resources while maintaining performance characteristics. It represents the system's capacity to accommodate growth in data volume, processing complexity, or user demands through resource expansion [59].

This capability includes both horizontal scaling, which involves adding more processing nodes, and vertical scaling, which involves increasing the capacity of existing nodes. In the context of stream processing systems, scalability is achieved through mechanisms such as data partitioning, parallel operator execution, and distributed state management.

Data Partitioning and Routing

Data partitioning is a fundamental technique for achieving scalability in stream processing systems. By dividing the input stream into multiple disjoint partitions, processing can be distributed across multiple nodes, enabling horizontal scaling.

Figure 2.10 illustrates common data partitioning strategies used in Flink. Forward partitioning maintains the same partitioning scheme across consecutive operators, ensuring that data processed by a specific instance of an upstream operator are sent to the corresponding instance of the downstream operator. Broadcast partitioning sends a copy of the data to all instances of the downstream operator, useful for operations that require access to the entire input stream, such as lookups or reference data enrichment. Key-based partitioning assigns data items to partitions based on a key, ensuring that all items with the same key are processed by the same operator instance. Random partition-

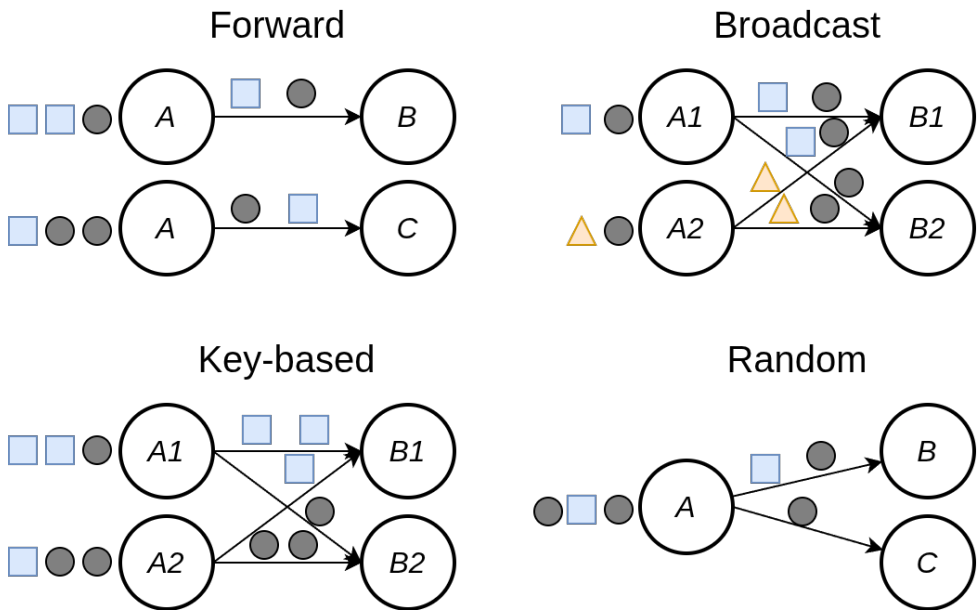


Figure 2.10 – Data partitioning strategies in Flink, showing forward, broadcast, key-based and random message exchange approaches between tasks.

ing distributes data items randomly across available partitions, which can be effective for stateless operations where the relationship between items is not important.

In heterogeneous environments, partitioning strategies must account for varying processing capabilities across nodes. Adaptive partitioning approaches can adjust partition assignments based on observed processing rates, assigning larger or smaller portions of the data stream to nodes based on their demonstrated capacity [51].

2.3.4 Elasticity

Elasticity extends the concept of scalability by adding the dimension of dynamic resource adjustment in response to changing workloads. While scalability refers to the ability to handle larger workloads by adding resources, elasticity encompasses the ability to both scale out and scale in resources automatically as demand fluctuates. Flink implements elasticity through its checkpoint mechanism, which creates consistent snapshots of application state that can be used to restart the application with a different parallelism level [47].

Elastic stream processing systems continuously monitor workload characteristics such as data arrival rates, processing latencies, and resource utilization. Based on these metrics,

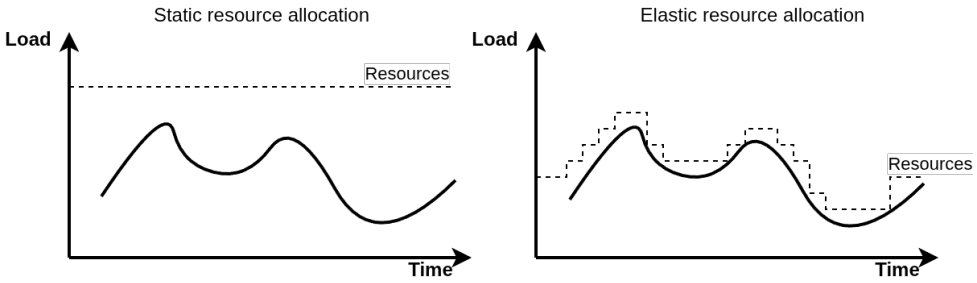


Figure 2.11 – Elasticity in stream processing systems, showing dynamic resource adjustment in response to changing workload characteristics over time.

they make decisions about when to add or remove processing resources. As illustrated in Figure 2.11, elastic systems demonstrate the ability to dynamically adjust resources in response to fluctuating workload patterns, scaling both horizontally and vertically to maintain performance goals.

The timing of elastic decisions is critical—scaling too aggressively can lead to unnecessary resource churn and state migration overhead, while delayed responses can result in performance degradation or resource waste. Implementing elasticity in stateful stream processing comes with an increased cost. When scaling out, the system must redistribute state across a larger number of processing nodes while maintaining consistency. When scaling in, state from nodes being removed must be transferred to remaining nodes. These state migrations must occur without disrupting ongoing processing or compromising exactly-once semantics [85, 185].

Decisions about which nodes to add or remove should consider not only the quantity of resources but also their quality—including processing power, memory capacity, and network connectivity. Decisions about when to scale out or in must also consider the reconfiguration overhead, which can include stopping and restarting applications, transferring state, and ensuring that the new configuration is compatible with the existing workload [205].

These challenges become even more complex in heterogeneous environments, where resource characteristics vary significantly across the edge-cloud continuum.

2.4 Resource Management with Kubernetes

Managing heterogeneous resources across the cloud continuum requires sophisticated orchestration platforms that can abstract underlying diversity while efficiently allocating

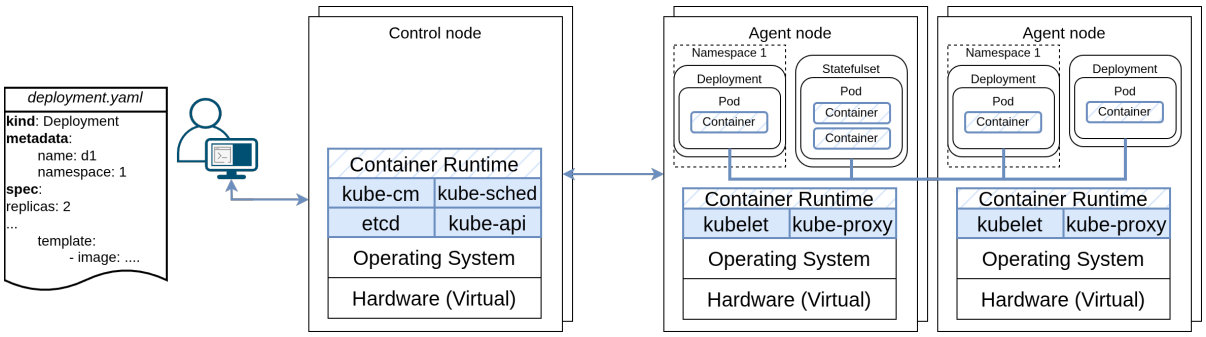


Figure 2.12 – Kubernetes Architecture.

resources to applications [104]. Kubernetes has emerged as a leading solution for this challenge, with extensions such as K3s adapting its capabilities to resource-limited environments.

2.4.1 Kubernetes Architecture

Kubernetes is an open-source platform designed to automate the deployment, scaling, and management of containerized applications [201].

Figure 2.12 presents an overview of Kubernetes' architecture. A user defines application requirements through configuration files such as deployment.yaml, which specify parameters such as namespace, replica count, and resource requirements. This declarative configuration interacts with the control plane, which consists of one or more control nodes for high-availability scenarios.

Each control node runs on either bare-metal hardware or virtual machines, with an operating system layer hosting the container runtime and four essential components: **etcd** for distributed storage, **kube-apiserver** as the cluster's entry point, **kube-controller-manager** for maintaining desired state, and **kube-scheduler** for workload placement decisions. The control plane connects to multiple agent nodes that execute the actual workloads.

Agent nodes follow a similar layered architecture with bare-metal or virtual hardware, an operating system, and a container runtime. They run two primary Kubernetes components: **kubelet** for pod lifecycle management and **kube-proxy** for network services. These agent nodes host application workloads such as StatefulSets and Deployments, demonstrating how namespaces can span multiple nodes and how pod replicas are distributed across the cluster infrastructure. Kubernetes creates a network overlay that seamlessly

connects pods across different nodes, abstracting the underlying network complexity and enabling direct pod-to-pod communication regardless of their physical location.

The fundamental unit of deployment in Kubernetes is the pod, which represents one or more containers that are deployed together on the same host. Pods are managed by higher-level abstractions such as Deployments, which maintain a specified number of pod replicas, and StatefulSets, which provide guarantees about data persistency and unique network identity. These abstractions enable declarative management of applications, where users specify the desired state and Kubernetes continuously works to maintain that state [183].

Kubernetes provides several mechanisms for resource management. Resource requests and limits allow specifying the minimum resources a container needs and the maximum it can consume. Quality of Service (QoS) classes determine how pods are treated under resource pressure. The scheduler uses these specifications, along with node labels and taints, to make placement decisions that optimize resource utilization while respecting application requirements [178].

2.4.2 K3s for Edge Computing

K3s is a lightweight Kubernetes distribution designed specifically for resource-constrained environments such as edge devices and IoT gateways. It maintains API compatibility with standard Kubernetes while reducing resource requirements through several optimizations. These include replacing etcd with SQLite as the default data store, bundling required components into a single binary, and removing unnecessary features and legacy components [207].

The reduced footprint of K3s makes it suitable for deployment on devices with limited CPU, memory, and storage capabilities. This enables extending Kubernetes-based orchestration to the edge of the network, allowing consistent application deployment and management practices across the entire edge-cloud continuum [40, 120].

For edge-cloud scenarios, K3s can be configured in various topologies. Edge nodes can operate independently when disconnected from the central infrastructure, with local control planes making decisions based on local conditions. Alternatively, they can function as worker nodes connected to a central Kubernetes cluster, with workload placement decisions made centrally but executed locally. These flexible deployment options allow adapting to different network conditions and organizational requirements [148, 194].

The platform supports multi-architecture images, automatically selecting the appropriate image variant for each node's architecture. Additionally, K3s can be configured to

use local image registries or air-gapped deployments, enabling operation in disconnected environments where direct access to external registries is not available [114].

K3s supports both ARM and x86 architectures, accommodating the diverse hardware found in edge environments [129]. This cross-platform compatibility, combined with its lightweight design, makes K3s particularly suitable for deploying applications across heterogeneous device fleets that may include single-board computers, embedded systems, and traditional servers.

In Chapter 6, we present our approach that leverages K3s for Kubernetes cluster deployment on heterogeneous edge hardware.

Chapter 3

State of the Art

In this chapter, we review the state of the art in Data Stream Processing (DSP) systems, focusing on their scalability and performance in heterogeneous environments. We explore the mechanisms that enable DSP scalability, the challenges posed by network and compute heterogeneity, and the methodologies for evaluating DSP systems in such contexts. Additionally, we discuss the tools and frameworks available for experimenting with heterogeneous DSP systems, including emulation and simulation techniques.

3.1 DSP Scalability

As workloads and data volumes increase, the scalability of DSP systems becomes an important factor for maintaining data processing performance. Scalability refers to the ability of a system to handle an arbitrary load by harnessing sufficient resources without significant degradation in performance [175]. To achieve scalability, multiple steps are typically involved, including data partitioning and operator replication. Data partitioning involves dividing data streams into smaller, manageable chunks that can be processed in parallel across multiple nodes [119, 142, 203]. Operator replication involves creating multiple instances of processing operators to handle the increased load, allowing for parallel processing of data streams [47, 175]. The scalability challenge is further compounded by the reconfiguration time required during scaling operations, as prolonged delays in adapting the system can lead to increased downtime and unprocessed data [56, 205].

3.1.1 Stateful vs. Stateless Operators

The scalability challenges differ significantly between stateful and stateless operators in DSP systems. Stateless operators can be easily replicated and scaled horizontally since they do not maintain any internal state between processing events. In contrast, stateful operators, such as window aggregations, joins, and pattern detection operators, maintain

a persistent state that must be carefully managed during scaling operations [95,96]. State partitioning, migration, and consistency become critical concerns when scaling stateful operators across multiple nodes [47]. This fundamental difference makes the scalability of stateful operators considerably more complex and often requires specialized techniques for state management and redistribution.

For systems to maintain high throughput and low latency, it is important that they can automatically adapt to workload changes by scaling resources up or down as needed and by deciding the best time to do so.

3.1.2 Heterogeneous Environments

Heterogeneous environments, characterized by the coexistence of diverse computing resources across edge, fog, and cloud layers, present unique challenges and opportunities for the scalability of data stream processing (DSP) systems. As discussed in Section 2.2, the Edge-Cloud Continuum is characterized by heterogeneous network and compute resources. Compute heterogeneity refers to the differences in processing capabilities, memory capacity, and storage characteristics across different nodes in a distributed system. Network heterogeneity refers to the variations in network performance characteristics such as bandwidth, latency, and reliability, across different nodes in a distributed system. In the context of DSP systems, this heterogeneity can have a considerable impact on the performance and predictability of data processing tasks.

Recent research has explored the deployment and evaluation of DSP platforms across heterogeneous infrastructures. A comprehensive comparative analysis of big data processing platforms (Apache Spark, Flink, OpenWhisk) deployed across edge, fog, and cloud layers demonstrates that deploying big data frameworks in resource-constrained environments is viable and often beneficial for latency-sensitive applications [189]. The study shows that fog computing often achieves the best trade-off between performance and resource constraints, while edge deployments are possible for lightweight tasks.

A framework built on top of Apache Storm aims to optimize network resource allocation by considering network and compute heterogeneity when generating DAG workflows in distributed environments [217].

3.2 DSP Elasticity

Elasticity, or automatic scaling of processing resources, is achieved by continuously monitoring the workload and efficiently adjusting the number of processing nodes to match the demand [102]. Stream Processing Engines (SPEs) expose live metrics on how the system ingests, processes, and outputs data. These metrics can then be used by a scaling algorithm to trigger scaling actions such as adding or removing processing nodes, based on the desired Quality of Service (QoS) objective [51, 209].

3.2.1 Metrics and Objectives

Effective elasticity in DSP systems relies on accurate monitoring of key performance metrics that indicate when scaling actions are needed or when goals are met. These metrics can be broadly categorized into system-level metrics, which reflect the overall health and performance of the DSP system, and application-specific metrics, which are tailored to the specific requirements of the data processing tasks being executed. System-level metrics provide insights into the overall performance of the DSP system, including resource utilization, throughput, and latency. Application-specific metrics, on the other hand, focus on the performance of individual data processing tasks, such as the progress of window joins or the completion of aggregations. Commonly used system-level metrics include throughput, which measures the rate at which data are processed, and latency, which captures the end-to-end delay from data ingestion to output [71, 100, 116, 158, 211]. Maximum Sustainable Throughput (MST) has also been widely adopted as a key metric, representing the highest processing rate a system can maintain while keeping stable performance [24, 108, 109, 117, 133, 177]. Additionally, the trans-precision approach proposed by [165] can further increase MST and reduce resource usage by dynamically adapting processing precision to workload variation. Another commonly used metric to derive rescaling decisions is backpressure, which indicates when downstream operators cannot keep up with the incoming data rate, signalling potential bottlenecks [25, 58, 62, 98, 145, 165]. CPU utilization and memory consumption are also widely monitored to assess resource saturation and trigger horizontal or vertical scaling decisions [52, 103, 185, 212]. Energy-aware approaches have also gained traction, where energy consumption is monitored to optimize resource usage and maximize performance [70, 72]. Recent work has also explored application-specific metrics, such as watermark progression in window-based operations, to enable more precise scaling decisions for stateful operators [79].

3.2.2 Adaptation Strategies

Various strategies have been developed to achieve elasticity in DSP systems, each with distinct approaches to triggering and executing scaling decisions. Threshold-based strategies represent the most straightforward approach, where scaling actions are triggered when monitored metrics exceed predefined thresholds [85, 94, 103]. These strategies are straightforward to implement, but they can suffer from oscillating behaviour and delayed responses to workload changes. Model-based approaches address these limitations by using predictive models to expect resource needs and proactively scale the system [25, 71]. These models often incorporate historical workload patterns and system performance characteristics to make more informed scaling decisions. Adaptive strategies dynamically adjust their parameters based on observed system behaviour, learning from past scaling decisions to improve future performance [98, 128, 179]. Finally, learning-based approaches use machine learning techniques, such as reinforcement learning, to optimize scaling policies over time [26, 67, 181].

3.3 DSP Autoscaling in Heterogeneous Environments

As discussed in Section 3.3, the heterogeneity of distributed DSP systems significantly complicates the achievement of effective elastic scaling [50, 69, 154]. The dynamic nature of workloads, combined with the diverse characteristics of computing and networking resources, requires sophisticated autoscaling strategies that can adapt to resource constraints and performance variations across the Edge-Cloud Continuum.

3.3.1 Addressing Compute Heterogeneity

A number of approaches have been proposed to address the issue of compute heterogeneity through horizontal scaling strategies [230]. Horizontal scaling is defined as the process of adjusting an operator's level of parallelism [52, 85] A recent approach presents a hierarchical horizontal operator scaling strategy for cloud infrastructures composed of heterogeneous computing systems that have variable costs [181]. This method optimizes resource usage costs at the application layer while leveraging reinforcement learning at the operator layer to handle per-operator scaling decisions based on resource usage, performance requirements, and adaptation overhead.

Es-Stream considers three types of resources for scaling stateful operators: CPU, memory, and I/O utilization [215]. The system monitors these resources to ensure compute nodes do not exceed their maximum capacity during operator instance deployment.

To address load imbalances in heterogeneous environments, a dynamic data partitioning strategy has been proposed which uses a scoring system to evaluate nodes based on both static hardware characteristics (e.g. CPU and memory capacity) and dynamic resource utilization [99]. This approach enables real-time adjustment of data distribution ratios to mitigate performance bottlenecks across heterogeneous compute resources.

One approach defines heterogeneity as the composition of a system with multiple types of processors, such as CPUs and GPUs. WindFlow leverages accelerators by performing record computations through a C++ library that enables GPU offloading of general partitioned-stateful operators [146, 147].

Vertical scaling increases the underlying compute resources attributed to an operator by adjusting CPU cores, memory allocation, or other hardware resources without changing the parallelism level. Q-Flink is a controller designed for Apache Flink that maintains Quality of Service (QoS) during traffic bursts by monitoring and preserving compute resources allocated to operators in shared environments [105]. MEAD addresses workload burstiness by using Markovian Arrival Processes to predict performance under varying load conditions [180]. Emma provides dynamic adaptation of CPU and memory resources to workload variations by resizing resources allocated to containers [75].

Recent research has explored hybrid approaches that combine both horizontal and vertical scaling to achieve more effective elasticity in heterogeneous DSP environments. Elasticutor uses both vertical and horizontal scaling, where CPU cores are dynamically added to a subset of keys before using distant nodes [212]. Justin is an elastic auto-scaler for stateful operators in Apache Flink that combines both scaling dimensions [185]. It dynamically monitors CPU usage and the performance of storage operations, subsequently reconfiguring tasks by adjusting their parallelism and/or their specific memory configuration.

3.3.2 Addressing Network Heterogeneity

Edge computing environments are characterized by heterogeneous network performance due to factors such as distance, congestion, and interference [168]. This network heterogeneity poses additional challenges for DSP elasticity, as variations in bandwidth,

latency, and reliability across different network links can significantly impact system performance and scaling decisions.

Many works focus on addressing bandwidth optimization and latency minimization strategies in heterogeneous geo-distributed DSP systems [6, 55, 158, 182, 213, 226]. Bandwidth maximization approaches aim to optimize data transmission rates across heterogeneous network links to achieve higher Maximum Sustainable Throughput (MST), while latency optimization strategies focus on reducing end-to-end processing delays in distributed environments.

Klink presents an optimization for Flink's scheduler which monitors window queries at run-time and analyses their watermark progression [79]. It reduces end-to-end processing latency by dynamically releasing windows and firing tuples to meet the QoS goal. LC-Stream addresses high latency in data processing within geo-distributed environments through an elastic scheduling strategy [197].

A commonly used metric for achieving bandwidth optimization and latency reduction in network-heterogeneous environments is backpressure monitoring. Many DSP auto-scaling works use backpressure as the main signal for triggering rescaling actions to adapt to network variations.

AdCom pre-aggregates tuples in mini-batch intervals to achieve sustainable throughput [98]. A PID controller monitors the workload and adjusts the interval based on the level of network buffer fill to reduce backpressure. SpinStreams is a static optimization tool for DSP systems that analyses and suggests a new pipeline configuration based on potential bottlenecks that can cause backpressure [145]. TransScale combines scaling and approximate data processing to respond to workload fluctuations at run-time [165]. Finally, GOVERNOR exploits a backpressure controller which takes into consideration the checkpointing costs to adapt the input sizes [58]. Although these works provide interesting solutions for managing elasticity, their effectiveness can be significantly impacted by network heterogeneity, which introduces unpredictability in performance and scaling decisions, as presented in Chapter 4.

Recent research proposes a strategy that considers both network and compute characteristics, aiming to reduce average latency and communication costs while minimizing computing usage overhead [196]. However, the number of works addressing both heterogeneity dimensions remains limited, highlighting an important research gap for elastic adaptation of DSP systems in the Edge-Cloud Continuum.

3.4 DSP Systems for Heterogeneous Environments

Beyond autoscaling approaches, recent research has focused on developing Stream Processing Engines (SPEs) that natively handle environment heterogeneity through specialized mechanisms. These systems address the unique challenges of network heterogeneity, compute resource diversity, and dynamic conditions across distributed computing environments. Some systems are specifically designed for edge computing, while others focus on exploiting heterogeneous computing capabilities or handling network variations. Unlike traditional cloud-based SPEs that assume homogeneous and reliable conditions, these systems incorporate heterogeneity-aware mechanisms directly into their core architecture.

EdgEWisE incorporates engine-level, congestion-aware operation scheduling and employs a fixed-size worker pool to improve performance over traditional systems such as Apache Storm in edge environments [84]. Building upon EdgEWisE's concepts, AgileDART achieves superior latency and scalability through a dynamic dataflow abstraction via DHT-based peer-to-peer overlays for autonomous operator management and a bandit-based path planning model to adapt data routing to heterogeneous and unreliable-edge networks [61]. Both systems have been evaluated on resource-constrained devices representative of the edge, such as Raspberry Pi nodes, demonstrating their suitability for deployment on single-board computers with limited computational resources.

NebulaStream represents a comprehensive Data Management System specifically designed for the Internet of Things [149]. It integrates Governor as its operator placement algorithm and incorporates EcoJoin, a novel approach for energy-efficient join processing optimized for IoT devices [57].

Frontier addresses network heterogeneity through backpressure stream routing, which dynamically adjusts transmission rates between operator replicas across distributed IoT nodes based on real-time network conditions [158].

A distinct form of heterogeneity emerges when considering processing units within a single system, particularly the combination of CPUs and GPUs (Graphics Processing Units). While this represents a different dimension of heterogeneity compared to the continuum-based approaches discussed earlier, GPU-accelerated stream processing has become increasingly relevant for emerging IoT use cases, especially Artificial Intelligence at the Edge [190].

SABER introduces a hybrid stream processing model that enables automatic work-

System	Edge-specific	Compute Het.	Network Het.
AgileDART	✓	–	✓
NebulaStream	✓	✓	–
Frontier	✓	–	✓
FineStream	–	✓	–

Table 3.1 – Stream Processing Systems Addressing Environment Heterogeneity

load distribution between CPUs and GPUs based on dynamic performance characteristics [127].

FineStream extends the hybrid CPU-GPU approach by addressing the critical challenge of efficient memory management and data movement between heterogeneous processors [228]. The system introduces a novel memory pool design that pre-allocates GPU memory to eliminate dynamic allocation overhead during runtime, which is particularly important for latency-sensitive-edge applications.

Table 3.1 presents a summary of the capabilities of the aforementioned systems. While each system addresses specific aspects of heterogeneity, none provides a comprehensive solution that can handle both compute and network heterogeneity simultaneously in the edge-cloud continuum. This represents a significant limitation since real-world edge deployments typically encounter both forms of heterogeneity simultaneously, revealing an important gap in current research.

3.5 Evaluating DSP in the Edge-Cloud Continuum

The experimental evaluation of Data Stream Processing (DSP) systems in heterogeneous environments is of key importance to determine their performance and scalability under varying network and compute conditions. This section provides a review of tools that facilitate the establishment of a geo-distributed environment, including simulation and emulation tools as well as physical testbeds.

3.5.1 Simulation Tools

Simulation tools provide mathematical models to represent system behaviour without requiring physical resources, making them cost-effective for large-scale experiments. FogNetSim++ extends CloudSim to simulate fog computing environments with mobility support and energy consumption models [170]. CloudSim and its variants such as

iFogSim and EdgeCloudSim offer discrete-event simulation capabilities for cloud, fog, and edge computing scenarios, enabling researchers to model complex hierarchical topologies and evaluate resource allocation strategies [45, 82, 97]. Although simulation tools excel in scalability testing and parameter exploration, they rely on simplified models that may not capture the full complexity of real-world network dynamics and hardware heterogeneity. In contrast, emulation provides more realistic behaviour through actual software execution.

3.5.2 Emulation Tools

Emulation frameworks provision virtual infrastructures through virtualization techniques. EmuFog provides a test environment for Fog computing [143]. It allows the user to define Fog topologies, and it deploys them by leveraging container technologies. Fogbed is an emulation system that leverages Mininet and Docker to provide virtual fog nodes [65]. It offers various features such as wireless network emulation and a native interface with external cloud datacenters.

Chaos Engineering techniques complement emulation by introducing controlled failures to evaluate system resilience [33]. Tools such as Chaos Mesh enable the emulation of network disruptions, resource contention, and other failure scenarios in distributed systems, providing insights into DSP behaviour under adverse conditions [202]. In Chapter 4, we use chaos engineering to emulate heterogeneous network conditions.

While these tools offer means to quickly spin up heterogeneous environments, they usually cannot fully represent certain traits of geo-distributed heterogeneous systems, such as hardware heterogeneity. To overcome these limitations, physical testbeds can be employed to provide realistic evaluation environments with actual hardware diversity and network conditions.

3.5.3 Physical Testbeds

Physical testbeds provide the most realistic evaluation environment for DSP systems by offering actual hardware heterogeneity and network conditions. Local clusters composed of smaller nodes such as Raspberry Pis offer an accessible starting point for researchers, enabling controlled experiments with limited resources while still capturing heterogeneous characteristics [35, 53, 206, 210, 218]. However, to evaluate DSP systems at scale and across truly geo-distributed environments, easy access to larger research testbeds

become essential. Research groups around the globe have developed experimental testbeds to support computer science research. These testbeds rely on HPC nodes, which are often distributed across several university campuses and research laboratories [53]. Grid'5000, located in France, is a large-scale testbed for parallel and distributed computing experiments [29,30,41]. Grid'5000 offers heterogeneous hardware, including GPUs, SSD/NVMe storage, and high-speed interconnects, as well as bare-metal provisioning. These features allow users to deploy fully customized software stacks and isolate their experiments at the network level. The US-based CloudLab and Chameleon testbeds offer cloud computing research platforms, providing researchers with access to a range of configurable resources for large-scale experiments [77, 122]. They are built on mainstream technologies such as OpenStack, facilitating broad accessibility and supporting diverse experimental requirements for the research community. The FIT IoT-LAB project consists of nearly 3000 IoT devices, including wireless sensors and mobile robots, distributed across six sites in France [2]. These devices enable various experimental scenarios, such as tracking power consumption and analysing radio signals indoors. The upcoming Slices project aims to group major testbeds across the European continent, including Grid'5000 and FIT IoT-LAB [193]. This will provide a large-scale platform enabling experimental research in several domains, including parallel and distributed heterogeneous computing.

While large-scale testbeds provide extensive resources and realistic geo-distributed conditions, they may not always be accessible to all researchers due to resource allocation constraints or availability limitations. This motivates the development of hybrid approaches that combine the accessibility of local clusters with the scale of research testbeds. Chapter 6 describes ScaleHub, a tool that provisions heterogeneous testbeds by seamlessly combining local Raspberry Pi clusters with Grid'5000 resources, enabling the creation of realistic edge-cloud setups that span from resource-constrained edge devices to powerful cloud infrastructure. The tool integrates Chaos Mesh to emulate variable network conditions across heterogeneous testbeds, enabling realistic evaluation of DSP systems under dynamic network behaviours.

3.6 DSP Benchmarks

Benchmarking DSP systems requires both standardized applications and specialized frameworks to evaluate performance, scalability, and resource utilization across different platforms and configurations.

3.6.1 Applications

Several benchmark applications are commonly used to evaluate DSP system performance [225]. The research community has established standard datasets that provide realistic streaming scenarios, including DEBS Challenge datasets and the NYC Taxi dataset for geospatial, temporal, and continuous query analytics [1, 166]. These datasets offer the advantage of representing actual data patterns and distributions found in production systems.

Building upon these datasets, complete benchmark applications have been developed that combine specific data sources with standardized processing logic. The Yahoo Streaming Benchmark is widely adopted across different DSP frameworks including Flink, Storm, and Spark, providing standardized workloads for performance comparison [220]. Auction-based benchmarks such as Nexmark combine synthetic data generation with complex event processing scenarios, enabling controlled evaluation of join operations and windowing functions.

While these datasets and applications are popular in DSP benchmarking, they are not specifically designed for edge-cloud environments. In Chapters 4 and 5, we conduct micro-benchmarks on single-operator pipelines to assess their performance scalability, drawing inspiration from these established benchmarks [163].

3.6.2 Frameworks

Several benchmarking frameworks have been developed to evaluate DSP systems in different contexts and environments. Theodolite is a comprehensive benchmark tool designed to evaluate the scalability of DSP systems, supporting multiple DSP frameworks and applications while providing metrics for assessing performance under various workloads and Service Level Objectives (SLOs) [101]. It explores the scalability characteristics of both stateful and stateless applications, offering insights into how different configurations affect performance in cloud-native environments. StreamBed focuses on offline slot capacity planning for Apache Flink, using micro-benchmarks to determine appropriate job configurations before deployment, thereby optimizing resource allocation [177]. PDSP-Bench provides a benchmarking suite specifically designed for parallel distributed stream processing systems, enabling comprehensive performance evaluation across different computational paradigms [3]. NAMB offers a prototype application generator that creates DSP applications from high-level YAML descriptions, enabling cross-platform de-

ployment on Storm, Flink, and Heron while supporting synthetic workload generation for quick benchmarking scenarios [164].

While these frameworks excel in their respective domains, they primarily target homogeneous cloud environments and lack specific support for edge-cloud heterogeneous deployments. Our Python-based orchestration solution fills this gap by automating node provisioning across multiple testbed types including Grid'5000, IoT-LAB, and custom clusters using Infrastructure as Code principles, while deploying DSP systems via Kubernetes for comprehensive benchmarking in truly heterogeneous environments.

Chapter 4

Data Stream Processing Effectiveness in Heterogeneous Networking Environments

4.1 Introduction

In the era of rapidly growing usage of real-time data, the emergence of Data Stream Processing (DSP) has revolutionized the management, analysis, and timely utilization of large volumes of continuous data [32, 115]. DSP systems excel at handling high-velocity data streams and enabling applications to promptly extract actionable insights. However, their deployment at a large scale poses significant challenges, especially as the number of IoT devices producing input data increases. The projected amount of data generated by IoT devices is expected to reach 163 zettabytes (10^{21}bytes) by 2025 [172].

As the number of data sources increases, so does the usage of long-distance networks to transport the raw data to cloud datacenters where they may be processed [153]. Geo-distributed environments emerge as a promising paradigm to address the limitations of such datacenter-centric approaches [4, 224]. Such environments extend centralized cloud platforms with additional resources close to the data sources to reduce data transfer latency and to enable faster decision-making processes. Incorporating DSP into geo-distributed environments offers significant advantages, allowing the DSP system to leverage the proximity to data sources for timely analytics [153].

An important challenge when processing data produced by IoT devices is that these data streams are often non-stationary [198]. It is therefore necessary to dynamically scale the processing capacity up and down according to the time-varying demand. Several strategies for managing the elasticity of DSP systems have been proposed in the literature [51]. They differ based on the deployment environment of the DSP, the monitored data type, the targeted quality-of-service objective, and the used optimization method.

DSP systems such as Apache Spark and Apache Flink were initially designed to operate in cluster-like environments based on a homogeneous set of powerful servers connected with high-capacity network links. However, these hypotheses are not necessarily met when the DSP system is deployed using multiple geo-distributed servers located close to the sources of input data. This chapter aims to identify the strengths and weaknesses of current DSP systems and their auto-scalers when they execute stateless as well as stateful operators in geo-distributed environments characterized by high and heterogeneous latency.

We focus our study on the Apache Flink DSP system [15] coupled with the Gesscale auto-scaler [25]. We evaluate the system throughput of stateless and stateful workflows when varying the number of servers, and when the network latency between servers is large and fluctuating. We show that:

— Stateful applications suffer from unstable throughput when scaled on nodes with heterogeneous network conditions;

— Backpressure is not a reliable signal for scaling decisions in heterogeneous network conditions;

— Skewed data distribution causes performance to be unpredictable during rescale operations.

Based on these observations, we propose a research agenda to improve the way DSP systems and their auto-scalers work in heterogeneous geo-distributed environments.

The remainder of the chapter is organized as follows: Section 4.2 details the experimental system. Section 4.3 presents our evaluations. Section 4.4 discusses the strengths and weaknesses of current DSP systems. Finally, and Section 4.5 concludes.
Parts of this chapter were published in IC2E 2024 [28].

4.2 Evaluation methodology

4.2.1 Experimental setup

Platform

We conducted all experiments in the Paravance cluster of the Grid'5000 experimental testbed [30]. Each machine is equipped with an Intel Xeon E5-2630 v3 processor (x86_64, 2.40 GHz, 2 CPUs/node, 8 cores/CPU), 128 GB of RAM, and a 10 Gbps network card.

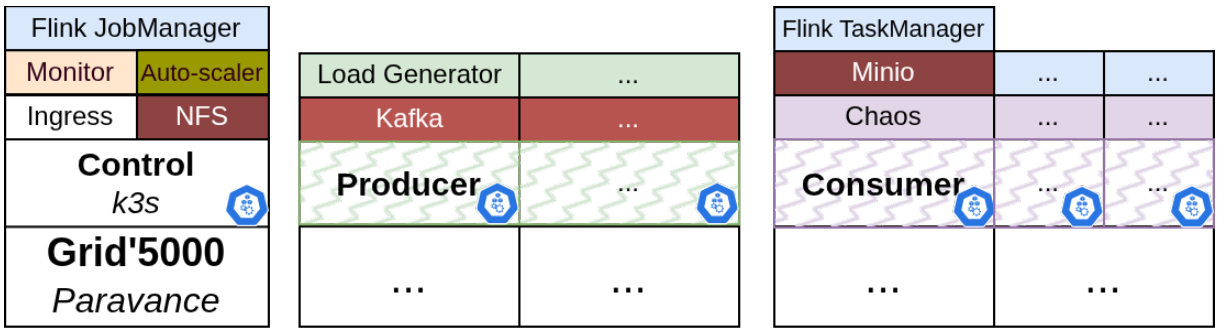


Figure 4.1 – Experimental platform.

We chose to use a single cluster as it allows us to precisely control the internode network latencies.

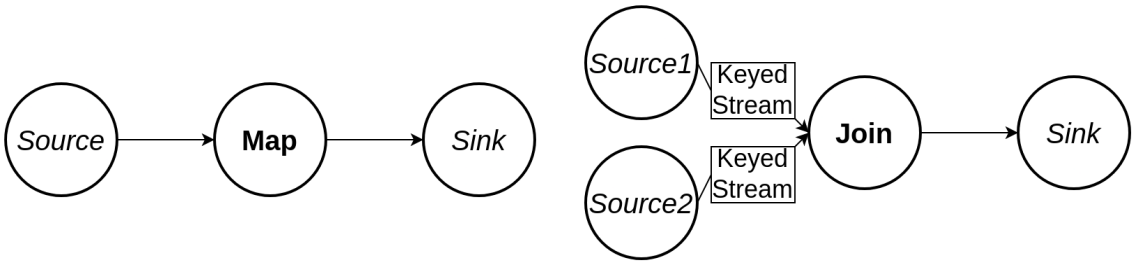
As shown in Figure 4.1, we deploy Kubernetes (K3S v1.28.3+k3s2) [114] on the 15 nodes of the cluster. Kubernetes manages the life cycle and the orchestration of the DSP system. We limit resources to 1 CPU core and 4 GB of memory to all applications deployed on top of the k3s nodes. The Kubernetes cluster has 5 nodes labelled as producers and 10 nodes labelled as consumers. Producers host the load generators and the Kafka broker replicas, whereas consumers host the DSP system instances and its distributed storage system.

To emulate network degradation between nodes, we add chaos-mesh (v2.6.2) [202] in the consumer nodes alongside the DSP system. It helps to create a controlled environment with different network profiles, such as bandwidth limitation and artificial network delays. Additionally, one node serves as a control node within the Kubernetes cluster, hosting the monitoring stack with an ingress endpoint for dashboard access, the NFS driver for storage backend connectivity, and the auto-scaler for DSP system management.

Stream processing system

We use the Apache Flink (v1.17.2) DSP system [15]. Flink is a powerful open-source stream processing framework which provides low-latency and high-throughput processing of data streams with exactly-once semantics. It is designed to execute as a distributed system, making it ideal for running in geo-distributed environments.

Flink's run-time consists of two types of processes: one JobManager, and one or more TaskManagers. The TaskManagers are responsible for executing the tasks assigned to them by the JobManager. The JobManager is responsible for scheduling tasks, coordi-



(a) Stateless job with Map.

(b) Stateful job with Window Join.

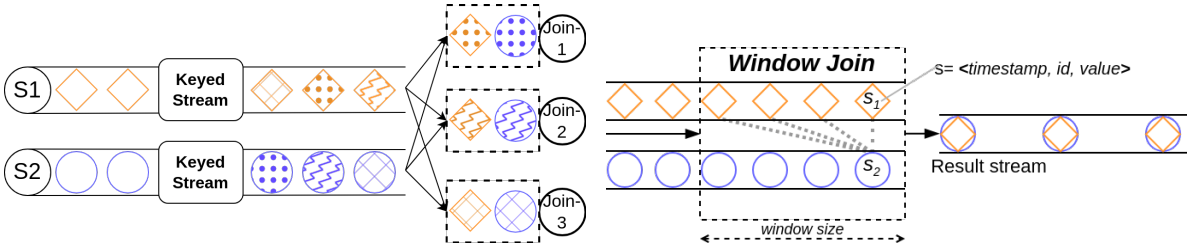
Figure 4.2 – Tested jobs.

nating checkpoints, and managing the overall execution of the job. A job in Flink is a program that describes a pipeline, composed of operators, in which data flows and where each operator applies transformations to data flows. In Kubernetes, JobManager and TaskManager run in separate containers, making scaling of TaskManagers a trivial task. A single task slot per TaskManager is set to force the allocation of a single operator's replica per TaskManager. This allows us to apply targeted network degradation to a subset of TaskManagers while observing how heterogeneous network conditions affect part of the stream processing pipeline.

To handle failure recovery, Flink leverages its internal checkpointing mechanism [47]. It allows Flink to recover state and positions in the stream by storing consistent snapshots of all the state in timers and stateful operators, including connectors, windows, and any user-defined state. Similar to checkpoints, a savepoint is a consistent image of the execution state of a streaming job. A savepoint allows a job to be gracefully stopped and reconfigured (e.g. re-configuring the parallelism level of a job with a stateful operator) while minimizing data loss. We set up Flink to use MinIo [139, 151] as its distributed storage system for checkpoints and savepoint.

Workload

We implement two pipelines to evaluate the performance of two types of operator: a stateless one and a stateful one. To emulate compute work, each operator applies a custom function consisting of a Fibonacci function evaluation. Figure 4.2(a) presents the pipeline with a *Map* acting as a stateless operator. It processes incoming data injected at the *Source*, performs computations by applying a custom *Map* function, and outputs the results to a *Sink* operator.



(a) Keyed streams mechanism. (b) Window mechanism.

Figure 4.3 – Window join mechanisms.

Figure 4.2(b) shows the pipeline with a *Window Join* acting as a stateful operator. Input data are produced in two *Source* operators at equal rate and with the same range of keys. We then transform each data stream to KeyedStream by using the record IDs. KeyedStreams [22] allow Flink to partition streams between tasks, thus helping to route records of a key subspace to independent task instances without incurring a synchronization overhead, as shown in Figure 4.3(a). Figure 4.3(b) illustrates the *Window Join* operation. Records are collected in a time-sized tumbling event window of 4 seconds. Each record of the first stream is compared to all other records of the other stream within this window. If the join condition used for the comparison holds true, a custom *Join* function is applied between the records. Given records s_1 from stream S_1 and s_2 from S_2, for the implementation of the join condition, we focus on two possible join criteria:

1. $s_1.\text{id} = s_2.\text{id}$ (ID-ID join)
2. $s_1.\text{id} = s_2.\text{value}$ (ID-value join)

In the first scenario, the join condition is applied on the same IDs used for the two keyed streams. This creates a compute-intensive pipeline with a high occurrence of joins, due to both streams being symmetrical. The second scenario involves a join condition applied between the ID of the first stream's record and the value of the second stream's record. We generate values randomly in a key subspace smaller than the IDs to force the join condition on keys different from those used for the keyed stream. We can thus observe skew in key distribution and the impact on performance caused by non-deterministic triggers of the join function.

Auto-scaler

Gesscale [25] is a DSP auto-scaler that adapts resources by using a performance model [24]. We deploy it in the master node of the Kubernetes cluster. It estimates

the internode network delays of consumer nodes using the Vivaldi network coordinates algorithm [68, 74, 134, 135] implemented by the Serf module [186] of Consul (v1.17) [64].

Based on the backpressure metrics on *Source* nodes and current data throughput, the auto-scaler predicts the system's performance and decides to scale the number of TaskManagers on the best available nodes. With the adequate number of TaskManagers, it adjusts the parallelism of every operator of the Flink application. Gesscale always tries to select nodes that have the lowest internode delay. Particularly, from all the available nodes, it looks at the maximum internode delay that a node would add to the system and it tries to select that with the minimal one, as it will be the weakest link in the graph, in terms of processing latency.

4.2.2 Experimental Procedure

Workload injection

The load generators inject data streams into the Kafka broker, which provides data for consumption to the stream processing application. Each stream consists of a sequence of $< id, timestamp, value >$ data records produced by multiple identical emulated sensors, each of which is identified by a unique identifier between 0 and $x - 1$. Each emulated sensor generates one message per second. In the *Map* use case, a single stream emulating $x = 100,000$ sensors injects data into the pipeline. In contrast, in the *Window Join* use cases, two streams emulating $x = 50,000$ sensors each injects data into each *Source* of the pipeline. This results in a constant number of comparisons being made within the *Window Join* between triplets of the two streams. Injecting a static workload which significantly exceeds the pipeline's processing capacity allows us to observe the evolution of throughput under conditions of both stable and degraded nodes following a scale-up operation.

Operator scaling

Figure 4.4 presents the typical deployment of a stream processing application. The experiment starts with the deployment of the job with a parallelism value of 1. This results in the entire pipeline being deployed on a single TaskManager. As the application scales out, Flink's scheduler co-locates the first instance of the operator with the *Source* operator, and all other instances on subsequent available TaskManagers. The auto-scaler gives the application a warm-up interval of 180 seconds, then it monitors the job metrics for an interval of 60 seconds at the end of which it computes a prediction matrix.

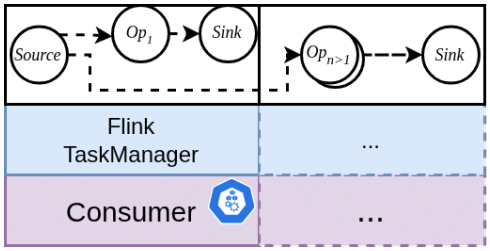


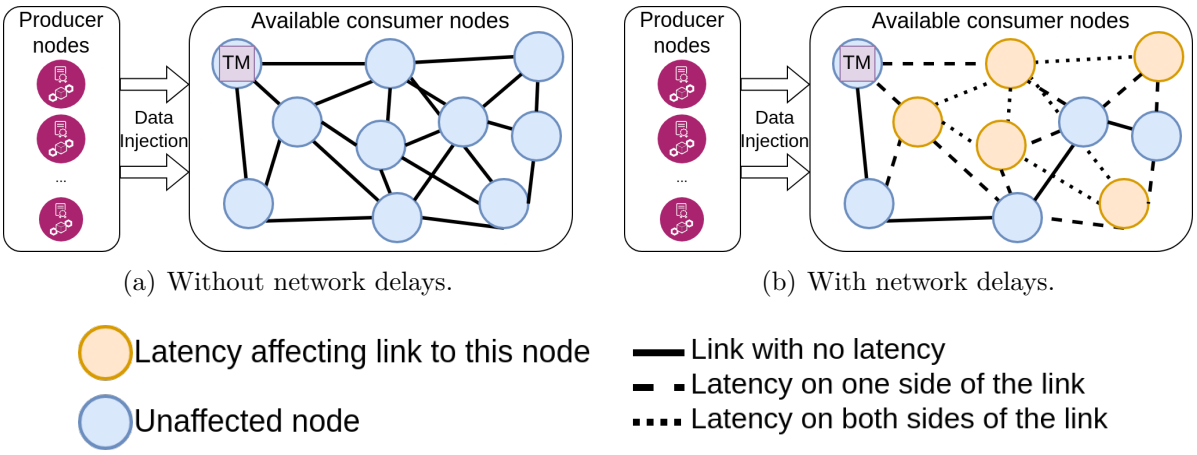
Figure 4.4 – Pipeline deployment.

This matrix is computed using metrics representing: the ingestion throughput of the operator (numRecordsInPerSecond); the backpressure generated on upstream operators (backPressuredTimeMsPerSecond); and the estimated internode network delays. Based on the prediction matrix, the auto-scaler adds TaskManager instances, if necessary, on available Consumer nodes and rescales the pipeline with a new parallelism level for the bottleneck operator. The reconfiguration of an operator is triggered according to a backpressure threshold. If, during a one-second interval, an instance is backpressured for at least 500 milliseconds, the auto-scaler executes then a scale-out decision, according to the prediction matrix. The first run of the model produces a matrix with modest accuracy, as the system has not gathered enough information on the job. After three rescaling actions, the prediction matrix starts to converge to a more precise forecast. The experiment concludes when the auto-scaler is no longer able to adjust parallelism due to shortage of resources or when the backpressure drops below the defined threshold of 500 ms, as this indicates that upstream operators are no more throttled down.

Wide-area network conditions emulation

Figure 4.5(a) shows a deployment with no artificial network delays. Since the testbed server nodes are co-located in a single rack, they result in an average internode latency lower than 1 ms. We run each job in this environment to observe the minimum number of instances required to process the imposed workload. Figure 4.5(b) depicts a network configuration in which 50% of consumer nodes are subject to network delays. This implies that the inbound connection of each affected node experiences an artificial delay. For example, two nodes affected with 25 ms injected delay will experience overall 50 ms round-trip latency on the link connecting them.

In the initial phase of the experiment, the auto-scaler selects nodes that are not impacted by network delays. This allows us to observe how scaling affects throughput when



(a) Without network delays.

(b) With network delays.

Figure 4.5 – Deployed consumer nodes.

network conditions are not a factor. Once all unaffected nodes have been consumed, the application starts using nodes affected by network delays, which impacts the performance of the job and the prediction model. We explore two scenarios with delay injection: (i) fixed 25 ms network delays; and (ii) dynamic network delays with values fluctuating between 15 ms and 35 ms. The first scenario illustrates the impact on the job's performance when it is rescaled on nodes with heterogeneous network conditions. The second scenario highlights the impact of dynamic heterogeneity in network conditions.

4.3 Evaluation

We now evaluate stateless and stateful operators using three jobs: a stateless job implementing the *Map* operator, and two stateful jobs implementing the *Window Join* operator, one with join condition demanding equality between $id_1 - id_2$ and the other between $id_1 - value_2$ of triplets s_1 and s_2 from streams S_1 and S_2. Each job runs with either no artificial latency, fixed 25 ms latency, or variable 25±10 ms latency with jitter.

We conduct five runs of each experiment to ensure the reliability of the results. The results display the minimum, maximum, and average throughput ingested by the operator per level of parallelism. We also include the model's predictions for each level of parallelism, along with the error percentage compared to the actual measured throughput.

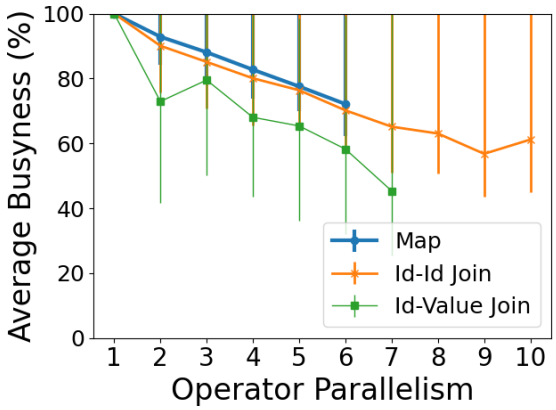
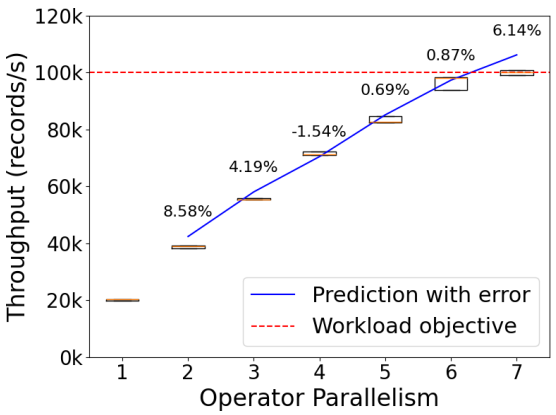


Figure 4.6 – Average busyness of individual TaskManagers when scaling streaming jobs with no artificial latency.

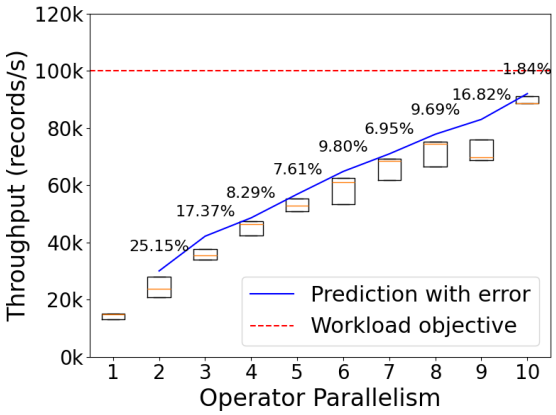
4.3.1 Effects of skewed data distribution on system busyness

Figure 4.6 presents the average busyness of each operator instance under our standardized workloads. The experiment, conducted without artificial latency, illustrates the minimum and maximum values of busyness experienced by any instance of the operator. It can be observed that all operators have at least one instance that is always 100% busy. In the *Map* and *Id-Id Join* use cases, this always occurs for the first instance of the operator that is co-located by Flink in the same TaskManager as the *Source* connectors, as illustrated in Figure 4.4. This TaskManager therefore handles an extra task to ingest and distribute input data to downstream operators, which explains the additional busyness of this TaskManager compared to the others. However, the distribution of keys among all instances of the operator remains balanced. This is evidenced by the linear decrease in busyness for increasing levels of parallelism.

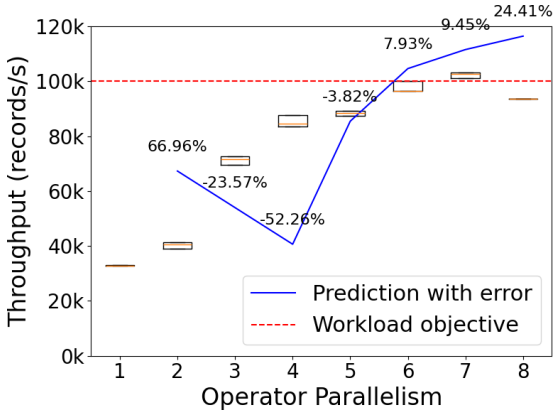
In contrast, the *Id-Value Join* operator experiences large variations in busyness, which can be attributed to the skewed distribution of matching id-value pairs. Unlike previous operators, the instance that is always busy changes at every re-scale operation, while at the same time, other instances experience low levels of busyness. This observation clearly highlights the effects of data skewness on stateful applications.



(a) Map.



(b) Id-Id Window Join.



(c) Id-Value Window Join.

Figure 4.7 – Throughput measurements – No latency scenario.

4.3.2 Flink's scaling behaviour in different latency setups

With no artificial latency

Figure 4.7 shows the throughput of the stateless and stateful operators in homogeneous environments with no artificial latency. We see that *Map* and *Id-Id Window Join* scale almost linearly with the number of instances. For the *Map* operator, we observe that each additional instance increases throughput by 90% of that of the first instance. In addition, the application manages to sustain the full workload (indicated in the figure by the red line) when it reaches 7 instances.

Stateful operators, however, behave differently as state management operations such as checkpointing cause an overhead that reduces throughput. Figure 4.7(b) shows the behaviour of the *Id-Id Window Join* operator. The throughput again grows monotonously

with the number of operator instances, but the throughput increase per added instance is only 70% of the throughput of a single operator. As a result, the *Id-Id Window Join* operator requires more instances to achieve the target throughput. Note that the experiment ends at 10 instances—the maximum number of consumer nodes in our settings.

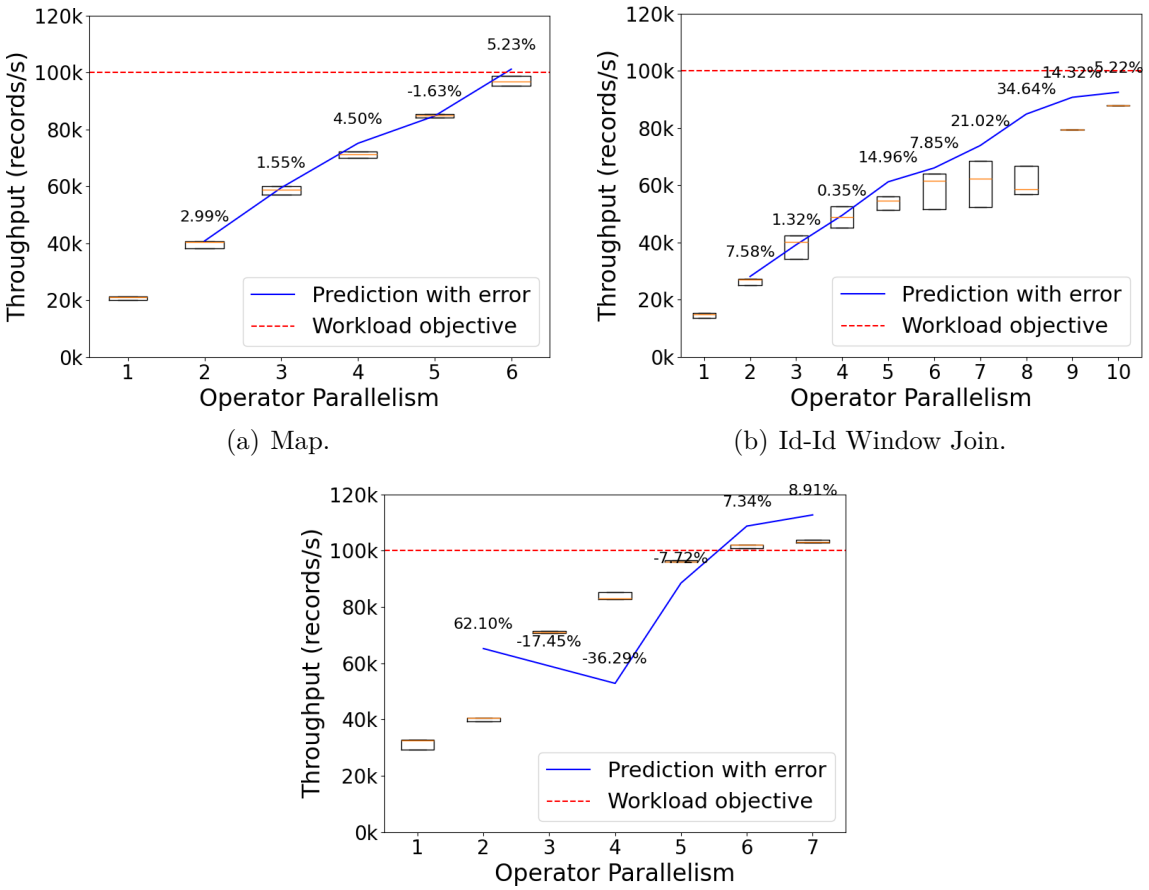
Finally, the *Id-Value Window Join* operator does reach its target throughput using only 7 instances, but the increase in throughout is not consistent as some of the new instances bring more extra throughput than others. This is due to the nature of the join condition, which causes skewness in the distribution of data. Data skewness causes an imbalance in the load across instances, which affects the overall throughput (as discussed in Section 4.3.1). We observe that this join condition does not generate a deterministic number of triplets for each comparison. Consequently, the join condition triggers the custom function less often, resulting in less computational work compared to the *Id-Id Window Join* use case. This allows the operator to ingest more incoming data without producing enough records.

We also see that Gesscale predicts the throughput of the *Map* and *Id-Id Window Join* operators with high accuracy. However, it fails to predict the throughput of *Id-Value Window Join* operator. We observe significant errors such as -52% because the model does not consider data partitioning and the impact of data skew. These results complement previous work [25], which only shows an expected near-linear scalability for *Map* and *Id-Id Window Join* under homogeneous network conditions. This suggests that performance models should take data skewness into account in their predictions.

With fixed latency

Figure 4.8 shows the data processing throughput when scaling DSP operators in heterogeneous environments where 50% of the consumer nodes are affected by artificial network latency (i.e., by adding 25 ms ingress network latency). Note that the auto-scaler selects new nodes based on their latency to previous nodes, so the impact of artificial network latency is only observed when scaling to 6 instances and above.

We make two observations. First, although the *Map* operator scales normally despite the network heterogeneity, the overall performance of the *Id-Id Window Join* operator degrades in heterogeneous environments when scaling beyond 5 instances. Most runs (4 out of 5) of the *Id-Id Window Join* operator show a peak throughput at parallelism level 7, with an average ingestion of 62k records/s. The throughput eventually drops to an average of 58k records/s at parallelism level 8. At this point, experiments stop scaling beyond 8



(a) Map.

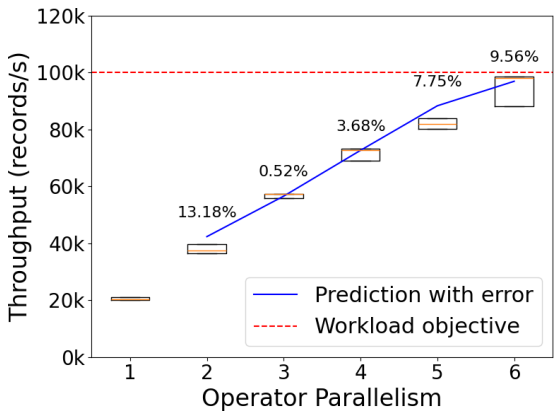
(b) Id-Id Window Join.

(c) Id-Value Window Join.

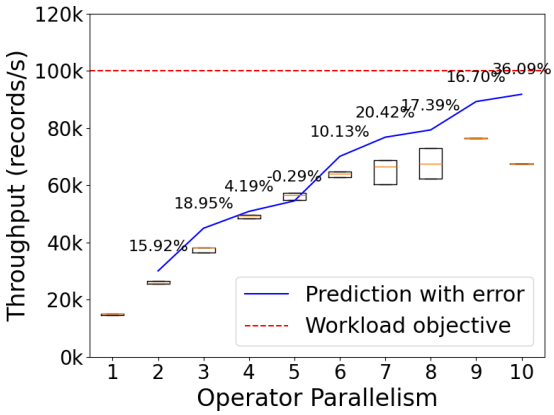
Figure 4.8 – Throughput measurements – Fixed 25 ms latency scenario.

instances, even though the target throughput is not reached, because the backpressure level drops below the defined threshold of 500 ms. Only one run of the experiment achieves parallelism 10, with a throughput of 88k records/s. The *Id-Value Window Join* operator exhibits a comparable behaviour to that observed in homogeneous environments. This operator, however, demonstrates its ability to ingest the full workload at parallelism 7. This is due to the join condition not being triggered often enough to saturate the operator with the ingested data.

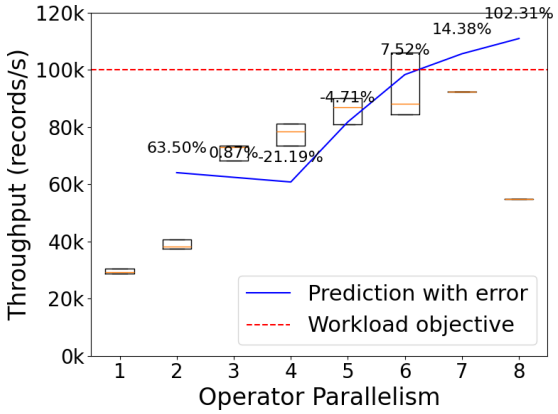
Second, Gesscale's predictions for the *Map* operator are reasonably accurate. In contrast, *Id-Id Window Join* shows poor predictions in the presence of latency. As heterogeneity appears from parallelism 6 and beyond, we observe prediction errors reaching 34% for the *Id-Id Window Join* operator. Finally, *Id-Value Window Join* exhibits very poor predictions, with errors up to 62%. Similar to the situation with no artificial latency,



(a) Map.



(b) Id-Id Window Join.



(c) Id-Value Window Join.

Figure 4.9 – Throughput measurements – Variable 25±10 ms latency with jitter scenario.

prediction errors of +62% and -36% render the auto-scaler highly unreliable although the DSP system reaches its target throughput between parallelism levels 6 and 7.

With fluctuating latency

Figure 4.9 shows the results when scaling DSP operators in dynamic environments, where 50% of consumer nodes are affected by latency jitter (i.e., with continuously changing network ingress delays between 15 ms and 35 ms). Once again, we observe that the Gesscale model correctly accounts for network delays when scaling stateless operators. The full workload is ingested at parallelism level 6, with low prediction errors. However, the behaviour differs for stateful operators. The performance of the *Id-Id Window Join* and *Id-Value Window Join* operators begins to decrease when using more than 6 instances. For instance, the throughput of the *Id-Id Window Join* operator drops by ∼13%, from

77k records/s to 67k records/s, when the number of instances is scaled from 9 to 10. The model predictions diverge from the measured throughput with errors up to 34%. This can be explained by the heterogeneous network conditions affecting half of the nodes. This degradation is more pronounced for the *Id-Value Window Join* operator. This use-case shows instability in throughput and prediction errors similar to the fixed latency experiment, which can be attributed to the nature of the application.

We also observe that scaling DSP operators usually helps to increase the data processing throughput. However, the expected increase in throughput is different depending on whether the operator to be scaled is stateless or stateful. In fact, DSP applications composed of stateful operators have an increased overhead due to state management and other factors. In some cases, the nature of the stateful operation itself causes unexpected variations in throughput. Furthermore, heterogeneous network conditions can significantly impact the throughput of stateful operators, potentially leading to performance degradation. The current prediction model accurately accounts for the impact of heterogeneous network conditions on stateless operators, enabling precise forecasts. However, the model struggles to accurately predict the throughput of stateful operators, resulting in significant prediction errors.

4.3.3 Auto-scaling in heterogeneous network conditions

We saw that the auto-scaler sometimes cannot rescale even though it does not reach the target throughput because the backpressure signal drops below the defined threshold. This results in the application not scaling as intended.

Figure 4.10 presents the effect of network delays on backpressure for the *Id-Id Window Join* operator. Here, 70% of consumer nodes are subject to fixed artificial network delays. We explore two cases: the first one injects 25 ms inter-node delay, whereas the second injects 50 ms. Increasing the number of instances of an operator can alleviate backpressure on upstream operators, provided that network conditions are not the limiting factor. The case with 25 ms injected delay shows that, with 3 parallel instances of the operator not affected by heterogeneity, the throughput scales linearly with small decrease in backpressure. However, when a pair of nodes with degraded network conditions is added, the throughput levels off. Simultaneously, backpressure experiences a sudden drop below the threshold level, which affects the auto-scaler's decision-making. This trend is even more pronounced in the case with 50 ms injected delay, where the backpressure drops significantly while the throughput also deteriorates, ultimately affecting the application

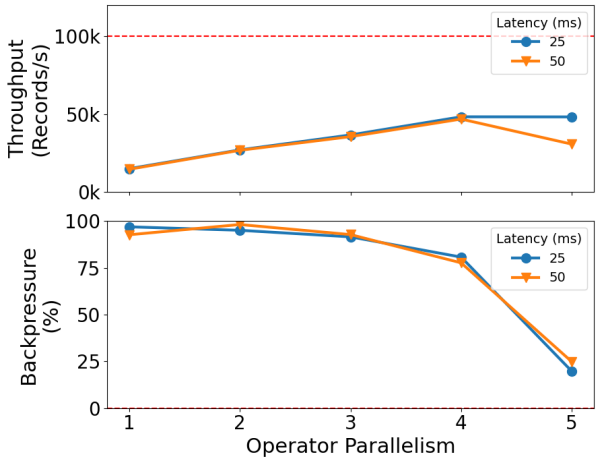


Figure 4.10 – Throughput and backpressure measurements – Id-Id Window Join with fixed network delays affecting 70% of nodes.

performance. In this case, the throughput bottleneck is due to network delays rather than limited computation capabilities. Backpressure is produced only locally within each TaskManager with limited computation capability, which explains the reason why the auto-scaler does not receive the correct trigger.

Although we conducted these experiments using Gesscale, we expect other auto-scalers which rely on backpressure to suffer from the same problem [58, 98, 145, 165].

4.4 Discussion

Our evaluations highlight strengths and weaknesses when using Apache Flink and Gesscale in realistic geo-distributed environments. We derive research directions for future work.

Stateless vs. stateful operators: Flink and Gesscale handle stateless operators with predictable and scalable performance, even in heterogeneous networking conditions. However, managing stateful operators is more challenging, and applications with a skewed data distribution exhibit significant performance instability during rescaling. Fluctuating network performance also degrades the performance of stateful operators as well as the quality of rescaling decisions.

Heterogeneous networking infrastructure: DSP systems were designed for homogeneous environments such as data centres. When the data sources are geo-distributed such as in IoT use-cases, it becomes useful to deploy the DSP in a cloud-continuum in-

frastructure. In our experiments, we could see that even in scenarios with just a couple of nodes suffering from degraded network performance, slow TaskManagers become the overall performance bottleneck, and force the unaffected nodes to slow down as well. This is a concern both in terms of absolute performance of the DSP system and for its auto-scaler.

Backpressure as rescaling signal: Backpressure is crucial for DSP systems to ensure exactly-once semantics without data loss. Many works also use it to identify potential bottlenecks and to trigger rescaling operations. However, in geo-distributed environments, this signal may be distorted, causing incorrect resource adaptation decisions. In such environments, additional signals should be considered for the adaptation of the stream processing pipeline.

4.5 Conclusion

In this chapter, we experimentally studied the performance and predictability of Apache Flink and Gesscale when handling stateless and stateful data processing tasks in realistic geo-distributed environments. We identified domains where these systems perform sufficiently well, in particular concerning stateless operators in heterogeneous networking environments. Stateful operators constitute a greater challenge in terms of absolute performance (especially for tasks which exhibit data skew), predictability, and rescaling trigger. These topics have attracted limited attention so far in the state of the art and constitute promising avenues for future research.

Chapter 5

Data Stream Processing Effectiveness in Heterogeneous Computing Environments

5.1 Introduction

The computing resources which constitute the edge-to-cloud continuum are inherently heterogeneous [176]. Besides the obvious resource disparity between cloud servers and edge devices [7, 161], computational requirements and energy constraints within the edge itself also often commend the usage of edge resources with heterogeneous computational capabilities [83, 121, 214] and network connectivity properties [28, 86].

Efficiently exploiting these heterogeneous resources is difficult for applications which may span multiple nodes in multiple locations. To exploit the available resources to the best of their ability, such applications need to balance their workload such that each compute node is given tasks to execute that have proportional complexity to their capabilities [49].

A popular family of edge-cloud applications which needs to tackle the heterogeneity challenge is Data Stream Processing (DSP) [115]. DSP frameworks enable their users to define and execute continuous computation over unbounded data streams such as those produced by IoT devices. These systems allow developers to specify processing pipelines that transform, aggregate, and analyze data as they are ingested, enabling real-time insights and actions. Deploying DSP systems in the edge-cloud continuum is very popular as it allows one to process data close to their sources, thereby reducing processing latency and the pressure on long-distance networks [25, 181].

However, modern DSP systems such as Apache Flink were designed for exploiting high-performance clusters with homogeneous computing resources [15]. Therefore, using them in the context of heterogeneous edge-cloud nodes places them out of their comfort

zone and raises fundamental questions regarding the way to execute them efficiently using heterogeneous resources spanning at least one order of magnitude in their processing capabilities [81].

In this chapter we address the following questions: (1) is it possible for Apache Flink to effectively exploit the capabilities of individual servers with very heterogeneous performance? ; (2) how should different types of machines be configured to deliver the best possible DSP throughput?; and (3) how do DSP applications perform when their operators are deployed on a heterogeneous mix of machines?

We conduct extensive experimental evaluations based on four very different types of machines: a homogeneous set of high-end bare-metal servers, two different types of virtual machines, and a set of Raspberry Pi 4B single-board computers. To preserve the generality of our findings we measure the throughput of a stateful "join" operator using Apache Flink under synthetic workloads and a wide variety of system configurations.

Our work identifies three main findings:

1. Simply declaring the available number of CPU cores and memory capacity in the default Apache Flink configuration is not enough to obtain the maximum throughput.

2. In many cases, deploying multiple single-slot TaskManagers consistently outperforms a single large multi-slot TaskManager by up to 350%.

3. When deploying an operator on a heterogeneous set of machines, the throughput of each operator instance aligns with that of the slowest instance. This means for example that the performance of a system configuration with two fast machines will significantly *drop* if we add a third machine with lower level of performance.

These findings provide both immediately actionable insights for practitioners deploying Flink in diverse environments and a longer-term research roadmap for the design of a future DSP system that can effectively exploit arbitrary sets of heterogeneous resources.

The remainder of this chapter is organized as follows: Section 5.2 details the experimental system and procedure. Then Section 5.3 evaluates how to obtain the best performance out of a single machine, and Section 5.4 presents the challenges of scaling out using heterogeneous types of nodes. Finally, Section 5.5 concludes the chapter.

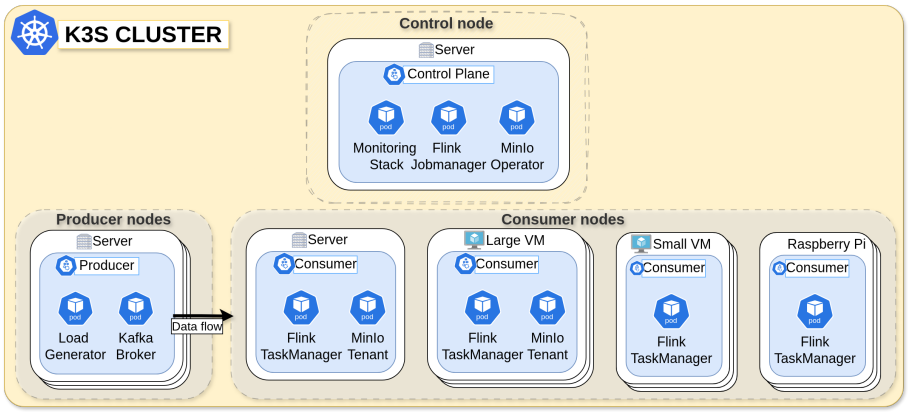


Figure 5.1 – Kubernetes cluster deployment across geographically distributed sites.

Table 5.1 – Heterogeneous Data Consumer Node Configurations.

Name	Node Type	# nodes	CPU	# cores/node	RAM	Network
BM	Bare metal server	1	Intel Xeon Gold 5320 2.2 GHz	52	384 GB	25 Gbps
VM-L	Virtual machine	4	Intel Xeon Gold 5320 2.2 GHz	8	32 GB	25 Gbps
VM-S	Virtual machine	10	Intel Xeon Gold 5320 2.2 GHz	2	8 GB	25 Gbps
RPi	Raspberry Pi 4 Model B	10	Broadcom BCM2711 1.8 GHz	4	4 GB	1 Gbps

5.2 Experimental Evaluation Procedure

5.2.1 System Design

Figure 5.1 shows the experimental system architecture used for our evaluations. Our heterogeneous testbed spans two geographically distributed sites connected via a mesh network overlay. The first site consists of seven high-performance "Paradoxe" servers from the Grid'5000 experimental HPC testbed [30, 91]. The second site hosts ten Raspberry Pi 4B (RPi) single-board computers that form a custom edge cluster [171]. To establish connectivity between the HPC testbed and the Raspberry Pi cluster, we deploy Tailscale to provide a mesh network overlay across both sites [199]. The measured latency between the two sites averages 20 ms, with a maximum of 25 ms.

Within this distributed infrastructure, we assign specific roles to different node types in our Kubernetes cluster. Producer nodes host data-generating applications including Kafka brokers and load generators. Consumer nodes host data-processing applications, specifically Flink TaskManagers and distributed storage components. From the Grid'5000 servers, we configure one server as a control node hosting the Kubernetes control plane, and three servers as producer nodes. Of the remaining three Grid'5000 servers, one operates as a bare-metal consumer node (labeled "BM"), while the other two host virtual

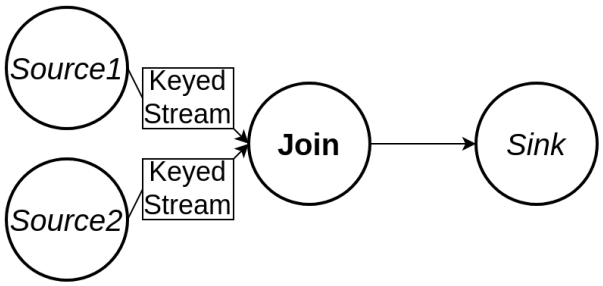


Figure 5.2 – Data workflow of the benchmark application.

machines that serve as consumer nodes. The first VM host runs four large virtual machines ("VM-L": 8 cores, 32 GB RAM each), and the second hosts ten small virtual machines ("VM-S": 2 cores, 8 GB RAM each). All ten Raspberry Pi 4B nodes operate as consumer nodes (labeled "RPi"). Table 5.1 summarizes the resource specifications for each of the four consumer node types where Flink TaskManagers are deployed.

We orchestrate this heterogeneous platform using the lightweight k3s (v1.32.2+k3s1) distribution [114]. The control node hosts instances of the Flink (v1.20) JobManager, monitoring stack, and MinIO controller. Producer nodes run replicated Kafka brokers and load generators. All consumer nodes (bare-metal, VM-L, VM-S, and RPi) host Flink TaskManagers. Bare-metal and VM-L nodes additionally host MinIO object storage tenants where Flink can save its periodic checkpoints. We configure Flink to generate checkpoints at 4-second periodicity. Savepoints used during Flink reconfigurations are stored there as well.

5.2.2 Benchmark Application

We exercise Flink using the stream processing application depicted in Figure 5.2. This application consists of two source operators which generate separate streams of events, a stateful join operator which processes and matches events from both sources, and a sink operator which simply discards the results. Each source operator reads events from a dedicated Kafka topic. Events are then partitioned by id using the KeyedStream routing strategy: this ensures that events with the same id are processed by the same parallel instance of the join operator. The join operator maintains a tumbling window of all events received during the previous 1 second, and emits a record when it can match two input records with the same ID.

5.2.3 Workload Injection

We generate a synthetic workload using two instances of the Theodolite load generator [101]. Each instance emulates a set of 175,000 IoT sensors that generate one record per second each and write these records in dedicated Kafka topics. Each record is a tuple in the form of $< sensor_id, timestamp, value >$.

5.2.4 Experimental Procedure

Each experiment generates $2 \times 175,000 = 350,000$ records/second, and measures the actual ingestion rate of the evaluated Flink operator. This workload is designed to saturate most Flink setups, allowing us to measure the Maximum Sustainable Throughput (MST) of various configurations.

We execute each evaluation during a 300-second period before issuing a Flink reconfiguration to test a different setup. We repeat the full evaluation process five times, and report the average MST values across these five runs. The figures present error bars indicating variability across runs, though the maximum observed standard deviation remains below 2,000 records per second at any parallelism level—representing less than 0.5% variation relative to the measurement scale of up to 400,000 records per second.

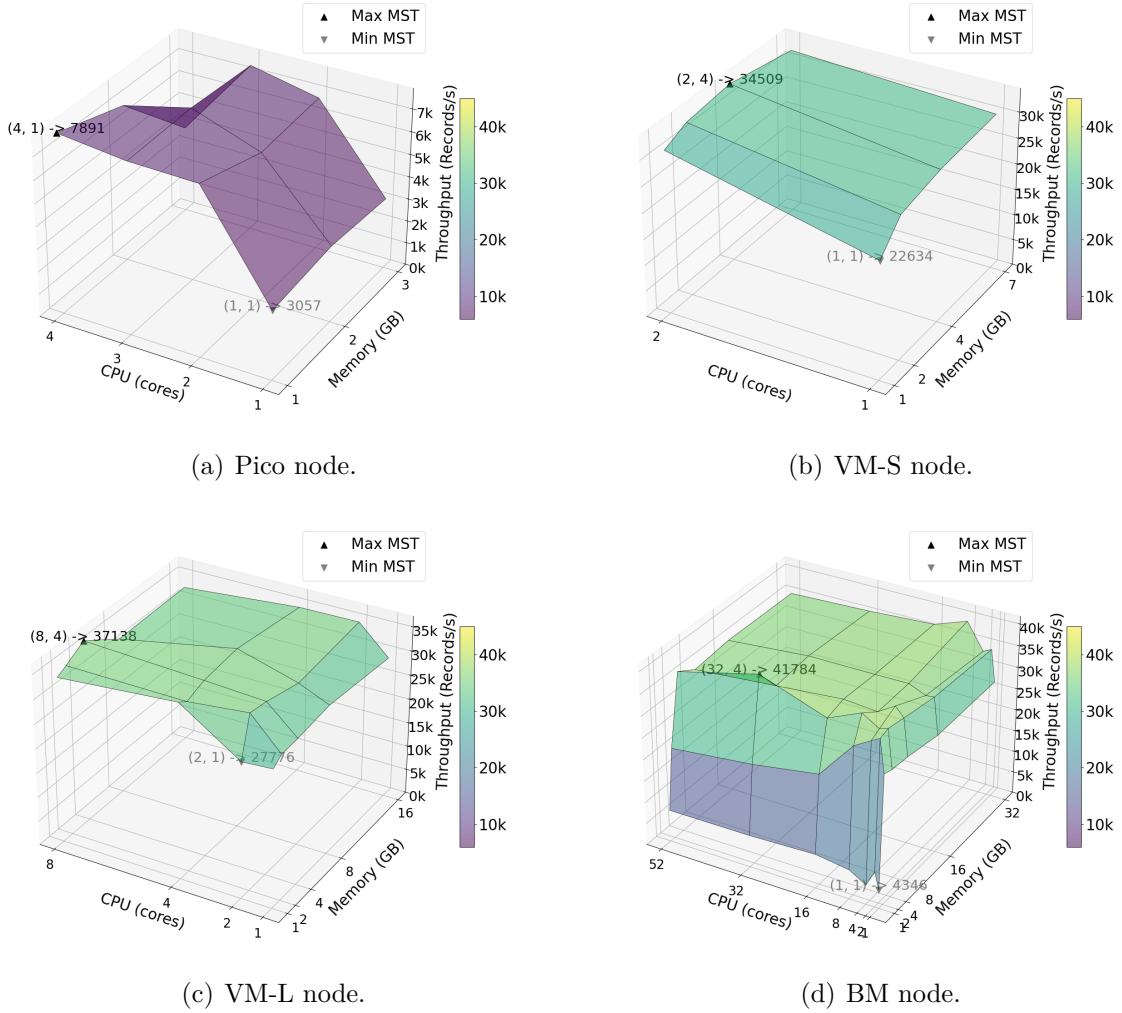
5.3 RQ1: Resource configuration

The first research question addressed in this chapter relates to the optimal Flink configuration on each of the four types of machines. Arguably, obtaining the best possible throughput out of a high-end datacenter server, a virtual machine, and a single-board computer may not require the same resource allocation parameters.

5.3.1 CPU and Memory Resources Per Task Manager

The first question we address in this section is to identify the relation between the CPU and memory resources being assigned to a single slot and the throughput of the operator which executes in this slot.

For each machine type we deploy a single Flink slot executing the join operator. We then systematically explore multiple combinations of CPU and memory resources ranging from 1 to 52 CPU cores and 1 GB to 32 GB of memory, adjusting the upper limits based



(a) Pico node.

(b) VM-S node.

(c) VM-L node.

(d) BM node.

Figure 5.3 – Impact of CPU and memory allocation on the throughput of a single slot on different node types.

on each node type's available resources. Figure 5.3 presents the throughput achieved by single-slot Task Managers under different resource allocations.

We first observe that the amount of allocated memory has limited impact on performance. As soon as the TM has access to sufficient memory to store a full data window, adding more memory provides little performance benefits. Note that this observation may be dependent on the nature of the observed operator; other operator functions may have different behavior [185].

The second observation is that an increased number of CPU cores delivers improved throughput, but with strongly diminishing returns. Task Managers configured with 1 core

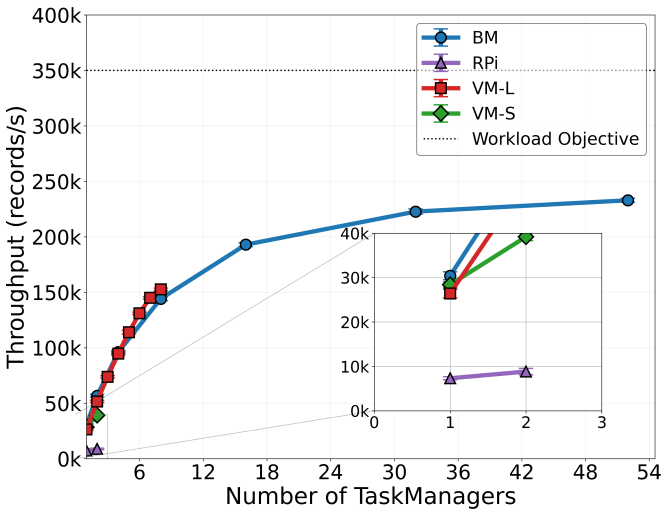


Figure 5.4 – Filling different node types with increasing number of TMs and one TS per TM.

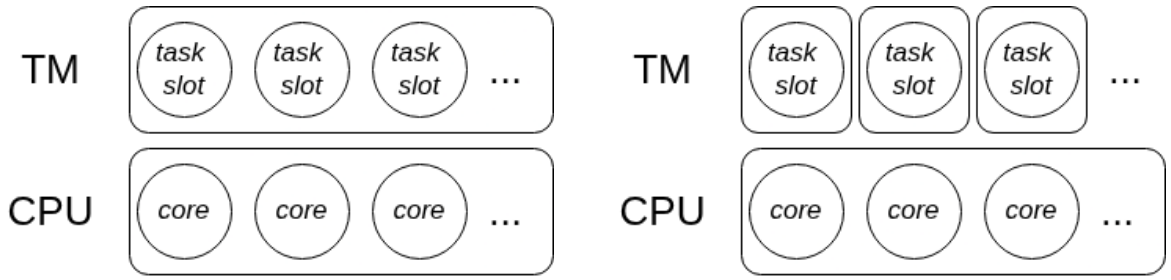
and 4 GB RAM consistently deliver the highest throughput per core for BM, VM-L, and VM-S.

Interestingly, RPi nodes demonstrate significantly lower throughput than other node types, despite having more CPU cores than VM-S nodes. This performance gap is directly attributable to differences in CPU architecture and memory subsystem characteristics, highlighting that raw core count alone is insufficient to predict performance for data stream processing workloads. This type of node achieves the best throughput per core when configured with 2 cores and at least 1 GB RAM.

5.3.2 Throughput Scaling Across Node Types

Since the most efficient TS configurations require small numbers of cores and amount of memory, making the best possible use of a full machine requires multiple TS per node. However, simply increasing the number of TS per node does not necessarily deliver linear throughput gains.

Figure 5.4 shows how the aggregate throughput scales when filling a single machine with increasing number of TMs, using one TS per TM. For each type of node, we deploy up to the maximum number of single-slot TM configured with 1 core and 4 GB of RAM each: the BM node can host up to 52 single-slot TMs whereas VM-L nodes can host up to 8, and VM-S up to 4. The RPi nodes can accommodate up to 2 TM with a dual-core



(a) A single TaskManager with multiple slots. (b) Multiple TaskManagers with a single slot each.

Figure 5.5 – Two opposite TaskManager and Task Slot configuration strategies on underlying machine cores.

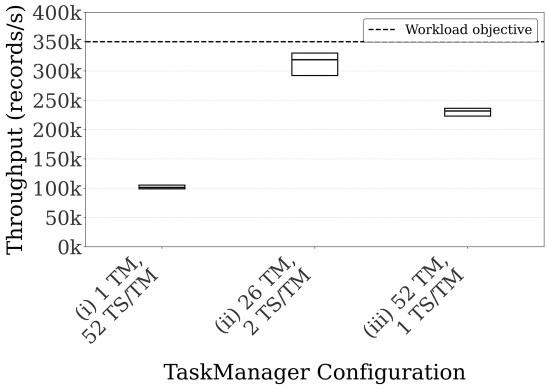
TS each, considering that dual-core TS delivered more throughput per core on this type of machine.

The results show linear throughput scaling on RPi and both types of virtual machines, with throughput increasing proportionally to the number of TMs. However, the BM node exhibits a notable deviation from this pattern. As the number of TMs increases beyond a certain threshold, throughput reaches a plateau and eventually fails to increase, although more and more hardware resources are being used.

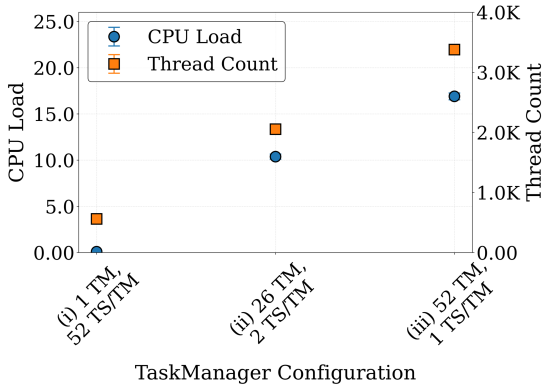
5.3.3 Single-Slot vs Multi-Slot TaskManagers

The previous section identified scaling limitations on the BM server which motivate further exploration into alternative ways to efficiently exploit its full hardware capabilities. Specifically, we want to examine whether dividing its core and memory resources differently between TMs and TS could provide better resource utilization and throughput performance.

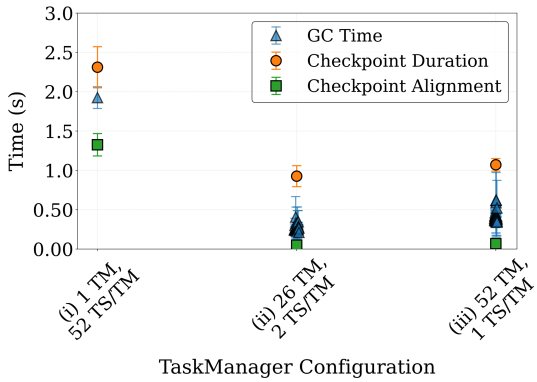
Figure 5.5 illustrates two opposite approaches for organizing TaskManagers and Task Slots on a multi-core machine: Figure 5.5(a) shows a single TaskManager with multiple slots that associate all task slots to the underlying CPU cores, whereas Figure 5.5(b) shows multiple single-slot TaskManagers that use the same CPU with each slot associated with one core. The results from these scenarios, along with intermediate configurations where TaskManagers are deployed with more than one slot and have more than one core per slot, are presented in Figures 5.6, 5.7, and 5.8. The results are presented in the form of box plots which illustrate the minimum, average, and maximum throughput achieved across five runs.



(a) Throughput.



(b) Aggregate CPU Load and Thread Count across JVMs.



(c) Garbage Collection Time per TaskManager & Checkpoint Duration and Alignment of the Job.

Figure 5.6 – Performance of single-core task slot configurations on a BM server (52 cores / 386 GB).

Single-core task slots: Figure 5.6(a) compares three options based on single-core Task Slots: a single TM with 52 task slots, a configuration with 26 TMs and 2 task slots each, and a configuration with 52 TMs with 1 task slot each. The results reveal significant differences. The single-TM option achieves only approximately 100k records/s, suggesting a performance bottleneck when too many slots compete for resources within a single JVM process. Deeper investigations reveal strong contention at the JVM level for shared resources, including the overhead of garbage collection and saturation of memory bandwidth across the 52 concurrent task slots.

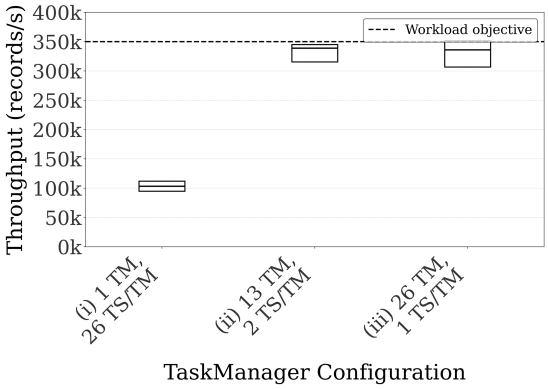
In contrast, the 26-TM configuration reaches 330k records/s, representing a 3.3x improvement. As shown in Figure 5.6(b), configurations based on multiple TaskManagers can exploit a significantly greater number of threads compared to single-TaskManager se-

tups when aggregating threads across all JVMs, as each TaskManager runs its own JVM instance. This dramatic difference highlights the importance of distributing workloads across multiple JVM processes to improve resource isolation and reduce intra-process contention. The superior performance of the 26-TM configuration can be attributed to better workload distribution, requiring approximately 3× less CPU time per thread compared to single TaskManager setups, as the computational load is distributed across multiple JVM instances. Conversely, single TaskManager configurations exhibit a lower overall CPU load percentage despite delivering significantly lower throughput, indicating inefficient resource utilization due to internal bottlenecks rather than hardware saturation.

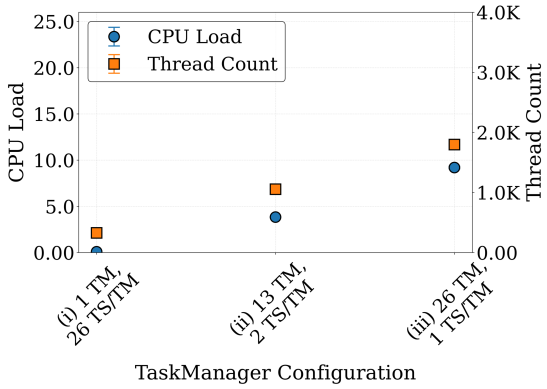
Finally, the 52-TM configuration performs at 220k records/s, significantly worse than the 26-TM configuration, indicating that excessive process creation overhead and inter-process communication costs begin to outweigh the benefits of resource isolation.

Dual-core task slots: Figure 5.7(a) presents three configurations based on dual-core Task Slots: a single TM with 26 task slots, a configuration with 13 TMs and two TS each, and a configuration with 26 TMs and 1 TS each. The single-TM configuration exhibits similar performance limitations to its counterpart in Figure 5.6(a), confirming the presence of bottlenecks related to memory bandwidth and garbage collection when concentrating too many slots within a single JVM process. The 13-TM and 26-TM configurations achieve 342k and 350k records/s respectively, demonstrating that allocating two cores per slot can enhance performance when properly distributed across TMs. As illustrated in Figures 5.7(b) and 5.7(c), the marginal difference between these configurations suggests that the dual-core allocation reaches an optimal balance between computational resources per task and memory channel utilization, with the 26-TM configuration providing slightly better performance due to improved memory access patterns and reduced garbage collection frequency per JVM instance.

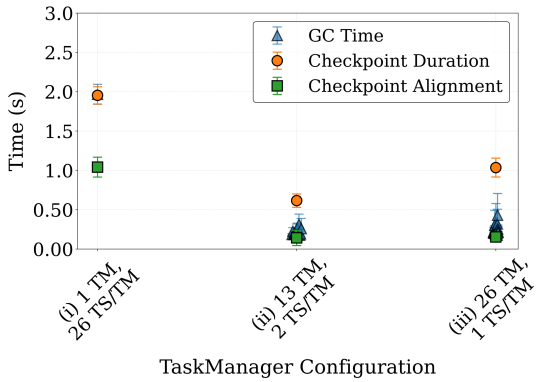
Quad-core task slots: Finally, Figure 5.8(a) examines two options based on quad-core Task Slots: a single TM with 13 TS, and 13 TMs with 1 TS each. We see that the single-TM option delivers poor performance whereas the 13-TM option achieves approximately 280k records/s. However, this level of performance still does not match that of dual-core TS options, indicating that beyond two cores per slot, the stateful join operator cannot effectively use additional CPU resources due to limitations in memory bandwidth per core and the inherently sequential nature of certain join operations. This plateau suggests that



(a) Throughput.



(b) Aggregate CPU Load and Thread Count across JVMs.



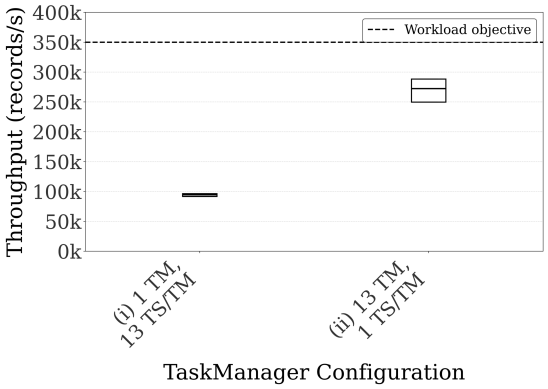
(c) Garbage Collection Time per TaskManager & Checkpoint Duration and Alignment of the Job.

Figure 5.7 – Performance of dual-core task slot configurations on a BM server (52 cores / 386 GB).

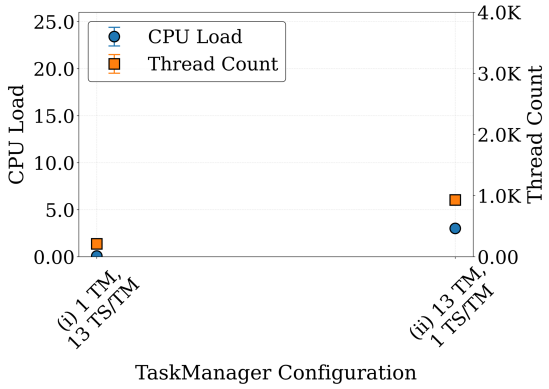
the memory subsystem becomes the limiting factor rather than CPU processing capacity, as evidenced by the declining performance despite increased core allocation.

As shown across Figures 5.6(c), 5.7(c), and 5.8(c), an interesting observation is that single TaskManager setups consistently exhibit checkpoint durations and alignment times that are approximately 3x longer compared to multiple TaskManager configurations. This extended checkpoint overhead directly correlates with the observed 3× longer garbage collection times in single TaskManager setups, creating cascading effects on both performance and fault-tolerance properties.

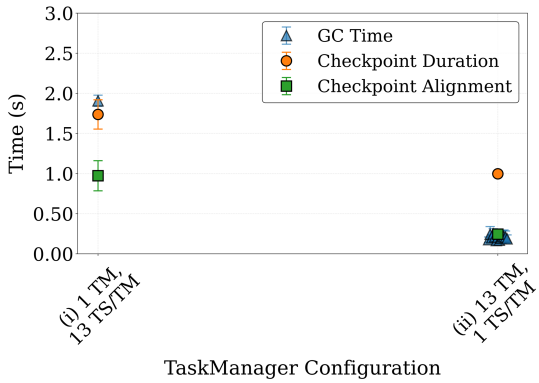
These results challenge the intuitive assumption that it is best to create a small number of large TMs, to declare the available resources to Flink, and to let Flink manage the resources within the TM. The observed performance patterns rather suggest that optimal



(a) Throughput.



(b) Aggregate CPU Load and Thread Count across JVMs.



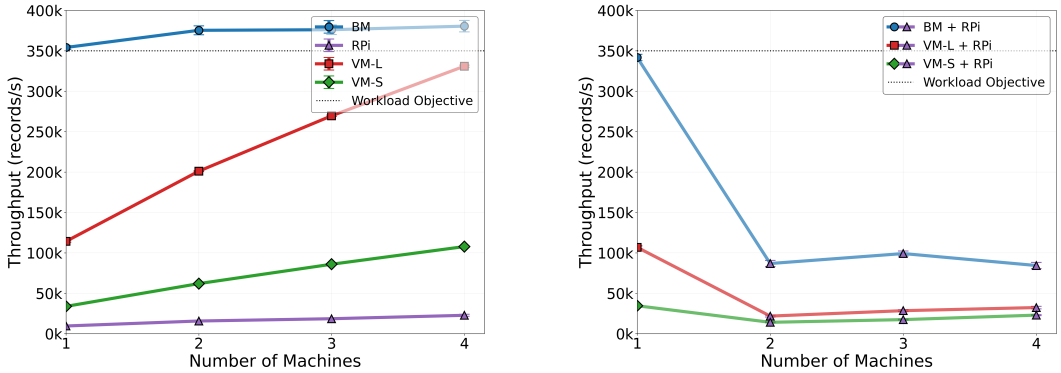
(c) Garbage Collection Time per TaskManager & Checkpoint Duration and Alignment of the Job.

Figure 5.8 – Performance of quad-core task slot configurations on a BM server (52 cores / 386 GB).

configurations balance JVM-level resource management with distributed processing benefits, as evidenced by the superior thread distribution, reduced CPU time per thread, and improved checkpoint performance achieved through multiple TaskManager deployments. These results lead to two key findings on how to use a powerful machine such as our BM node:

Finding 1: Optimal memory allocation per TaskManager: Our evaluation of CPU and memory configurations reveals that throughput-per-core is maximized with a 4 GB memory allocation for each TM. While allocating additional memory beyond 4GB shows minimal improvement, providing less than 4 GB severely impacts performance due to increased garbage collection overhead.

Finding 2: TaskManager and Task Slot configuration: Deploying multiple



(a) Homogeneous scaling.

(b) Heterogeneous scaling.

Figure 5.9 – Scaling performance across several machines running at capacity.

single-slot TaskManagers consistently outperforms single large multi-slot TaskManagers by up to 3.5× on bare-metal servers. This improvement results from multiple complementary factors: better thread distribution across JVM instances (achieving approximately 3× less CPU time per thread), reduced garbage collection overhead (3× shorter GC times), and improved checkpoint performance (3× faster checkpoint duration and alignment times). The optimal configuration uses 26 TaskManagers with dual-core task slots, achieving 350k records/s compared to 100k records/s with a single 52-slot TaskManager.

Based on these findings, we retain the following configurations for using each of our four node types at full capacity:

— RPi nodes use 2 TMs with 1 TS and 2 cores each;

— VM-S nodes use 2 TMs with 1 TS and 2 cores each;

— VM-L nodes use 8 TMs with 1 TS and 1 core each;

— BM nodes use 26 TMs with 1 TS and 2 cores each.

5.4 RQ2: Homogeneous vs Heterogeneous computing

We now examine Flink's scaling behaviour when extending processing across multiple machines. Specifically, we compare performance scaling of the stateful operator when using multiple machines of the same type vs. heterogeneous environments based on different machine types.

Figure 5.9 presents throughput measurements for both scenarios. In Figure 5.9(a), we observe the scaling behaviour when adding nodes of the same type. The bare-metal

nodes start at 350k records/second and achieve a maximum throughput of approximately 380k records per second. The surplus of 30,000 records can be attributed to the system processing buffered records placed on hold during the reconfiguration phase.

All other machine types (VM-L, VM-S, and RPi) demonstrate linear throughput scaling when increasing the number of machines, albeit at different absolute performance levels. However, the BM+RPi experiment, a single bare-metal node, achieves 342k records per second. However, when we add a single Raspberry Pi, the system throughput dramatically drops to just 87k records per second—a 74.6% reduction. Adding a second Pi slightly improves performance to 99k records per second, but adding a third Pi causes throughput to drop again to 84k records per second. Similar patterns of severe performance degradation appear in other heterogeneous combinations.

This counter-intuitive behaviour is attributable to Flink's utilization of a hash-based data routing strategy, which distributes workload uniformly across all TMs without taking into account their respective processing capabilities. In heterogeneous environments, this means that slower nodes receive the same workload as faster ones, creating bottlenecks that severely degrade the overall system performance. Although it is understandable that adding a slow node can add little extra throughput to a fast one, it turns out that the presence of a single slow machine actively undermines the processing capability of the entire system.

Finding 3: Hash-based routing creates performance bottlenecks in heterogeneous deployments. Flink's default hash-based routing creates severe performance limitations in heterogeneous environments. By distributing data uniformly without considering node capabilities, slower nodes become bottlenecks that dramatically limit overall system throughput. As demonstrated in Figure 5.9(b), adding Raspberry Pi nodes to a more powerful machine does not just fail to improve performance—it causes throughput reduction of up to 75%. This indicates that Flink deployments in heterogeneous environments require different routing strategies that account for the individual processing capabilities of each node.

5.5 Conclusion

This chapter presents a comprehensive evaluation of Apache Flink's performance when deployed on heterogeneous computing nodes. Our experiments revealed that Flink's default configuration leads to significant performance limitations in heterogeneous envi-

ronments, primarily due to its resource-agnostic task assignment and data partitioning strategies.

We found that the best performance in heterogeneous deployments is achieved with single-slot TaskManagers configured with at least 2 cores and 4 GB of memory. However, even with this optimal configuration, smaller nodes can become bottlenecks that limit the performance of the entire system. The default hash-based routing strategy in Flink creates performance limitations in heterogeneous environments, as it does not consider the processing capacity of the destination nodes.

Future work should focus on developing resource-aware stream processing systems that can automatically adapt to heterogeneous environments. This includes implementing adaptive data partitioning strategies and resource-aware task assignment.

Chapter 6

ScaleHub: A testbed on the edge

Edge-cloud environments present unique challenges for evaluating Data Stream Processing (DSP) systems due to their heterogeneous nature spanning from resource-constrained edge devices to powerful cloud infrastructure. Existing benchmarking frameworks and testbed platforms primarily target homogeneous cloud environments, leaving researchers without adequate tools for realistic edge-cloud experimentation.

This chapter presents ScaleHub, an orchestration tool that addresses these limitations by enabling hybrid testbed deployments. ScaleHub combines local-edge device clusters with large-scale research infrastructure, creating realistic experimental environments for heterogeneous DSP system evaluation. The tool leverages Infrastructure-as-Code principles to provision ephemeral Kubernetes clusters using K3s across diverse hardware platforms while integrating chaos engineering capabilities for comprehensive resilience testing.

ScaleHub's hybrid approach bridges the gap between accessible local experimentation and large-scale research infrastructure such as Grid'5000, democratizing access to realistic edge-cloud testing environments without requiring exclusive access to major research facilities.

6.1 Introduction

Evaluating Data Stream Processing (DSP) systems across edge-cloud environments is challenging because it requires heterogeneous infrastructure that spans from resource-constrained edge devices to powerful cloud servers, with varying network conditions and computational capabilities (see Section 2.2.2).

For infrastructure provisioning, researchers typically rely on large-scale testbeds such as Grid'5000, Chameleon, and FIT IoT-Lab, but getting access can be challenging due to resource constraints and availability [2, 41, 122]. Moreover, while FIT IoT-Lab provides IoT devices, these are low-power microcontrollers unsuitable for hosting orchestration tools such as Kubernetes and the complex DSP applications that run on them.

This creates a practical problem: local clusters built with affordable edge devices (such as Raspberry Pi boards) are accessible and controllable but too small for realistic edge-cloud scenarios. Large-scale testbeds provide the necessary scale but are often unavailable and lack the edge device characteristics needed for realistic edge-cloud experimentation.

Some recent works have addressed these infrastructure challenges for DSP experimentation. GESBED provisions Raspberry Pi clusters for DSP experiments, while Greenflow automates deployments on Grid'5000 [23, 130]. These approaches demonstrate the value of infrastructure-as-code for DSP research but target single infrastructure types.

For benchmarking, frameworks such as Theodolite, StreamBed, and PDSP-Bench provide workload generation and performance metrics but assume the underlying infrastructure is already provisioned and homogeneous [3, 101, 177]. Researchers end up having to choose between accessibility and realism.

ScaleHub uses infrastructure-as-code such as GESBED and Greenflow, but aims to combine clusters of different nature. This Python-based tool automatically deploys ephemeral Kubernetes clusters using K3s across heterogeneous hardware platforms, from edge devices to cloud nodes, so that benchmarking frameworks can be run in computationally heterogeneous environments. Additionally, ScaleHub provides heterogeneous network conditions through Chaos Mesh to emulate realistic edge-cloud scenarios.

Rather than requiring exclusive access to large-scale testbeds, ScaleHub lets researchers combine whatever resources they have available—their own edge device clusters plus allocated time on research infrastructure.

6.2 Design

This section presents the design of ScaleHub, a platform for orchestrating distributed stream processing experiments across heterogeneous edge-cloud infrastructures. We first identify the key challenges that arise when conducting such experiments, then describe our architectural approach to address these challenges.

6.2.1 Challenges

Conducting distributed stream processing experiments across heterogeneous edge-cloud infrastructures presents several key challenges that must be addressed to enable reproducible and reliable research.

Heterogeneous Infrastructure Orchestration. Edge-cloud deployments span a variety of hardware platforms, ranging from resource-constrained ARM-based devices to powerful x86 bare-metal servers. Careful resource management and workload placement strategies are required to deploy and manage distributed systems across this heterogeneous landscape. However, traditional Kubernetes deployments assume homogeneous environments, rendering them unsuitable for edge-cloud scenarios where nodes exhibit vastly different computational capabilities.

Network Connectivity Across Testbeds. Establishing reliable network connectivity between nodes from different testbeds is challenging. Although Grid'5000 provides users with VPN access, this mechanism is not intended for internode communication during experiments [93]. Although Grid'5000 offers global accessibility via IPv6 and reconfigurable firewalls, local clusters frequently operate within corporate networks that restrict external connectivity [92]. This heterogeneity in network policies hinders the integration of local infrastructure with research testbeds.

Reproducible Experimental Environments. Reproducible deployments are required for edge-cloud research, covering not only infrastructure provisioning, but also application deployment, monitoring setup, and network configuration. Researchers must be able to consistently provision, configure, use, and destroy experimental environments across multiple campaigns. Coordinating these operations across multiple testbed platforms while maintaining reproducibility remains a significant challenge.

6.2.2 ScaleHub Architecture

ScaleHub addresses these challenges through a layered architecture that separates concerns between infrastructure provisioning, cluster orchestration, and experiment execution. The system builds on Enoslib library to provide unified access to heterogeneous testbeds while implementing Infrastructure-as-Code principles for reproducible deployments [60].

Figure 6.1 presents the three-layer architecture managed through Ansible playbooks [9]. The **Infrastructure Layer** handles node provisioning and establishes a unified network overlay across heterogeneous testbeds. The **Orchestration Layer** deploys lightweight Kubernetes clusters optimized for resource-constrained environments. The **Application Layer** manages DSP applications, monitoring infrastructure, and experiment orchestration.

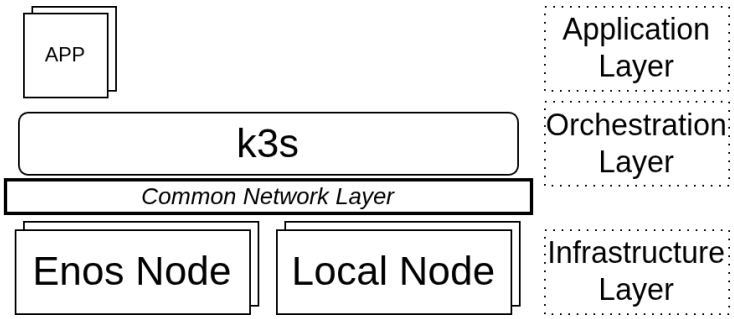


Figure 6.1 – ScaleHub architecture showing the three-layer design: infrastructure provisioning through Enoslib, cluster orchestration with K3s, and experiment management through the containerized experiment-monitor module.

Infrastructure Provisioning

The infrastructure layer enables the integration of different testbeds by providing two key mechanisms: unified provisioning and network overlay establishment.

Enoslib offers unified provisioning across major scientific testbeds, including Grid'5000, FIT IoT-Lab, and Chameleon. Users specify the platforms they wish to use in configuration files, detailing the number of nodes required, the necessary resources, and the reservation parameters. This abstraction allows researchers to use the same configurations on different testbed platforms without having to learn platform-specific APIs.

ScaleHub can be extended to incorporate local clusters that are not supported by Enoslib, thereby incorporating them into the same configuration framework.

To overcome network connectivity issues, ScaleHub uses Tailscale VPN to establish a common network layer [199]. This approach overcomes the limitations of the user-oriented VPN of Grid'5000 and the connectivity restrictions of corporate networks hosting local clusters. Tailscale establishes secure mesh networking between geographically dispersed nodes, achieving an average stable network delay of around 20 ms between nodes and enabling seamless communication regardless of the underlying network policies.

Cluster Orchestration

The orchestration layer addresses the challenges posed by heterogeneous infrastructure by deploying K3s, a lightweight Kubernetes distribution that is optimized for environments with limited resources. K3s provides full Kubernetes API compatibility whilst utilizing a significantly reduced quantity of resources in comparison to a full Kubernetes

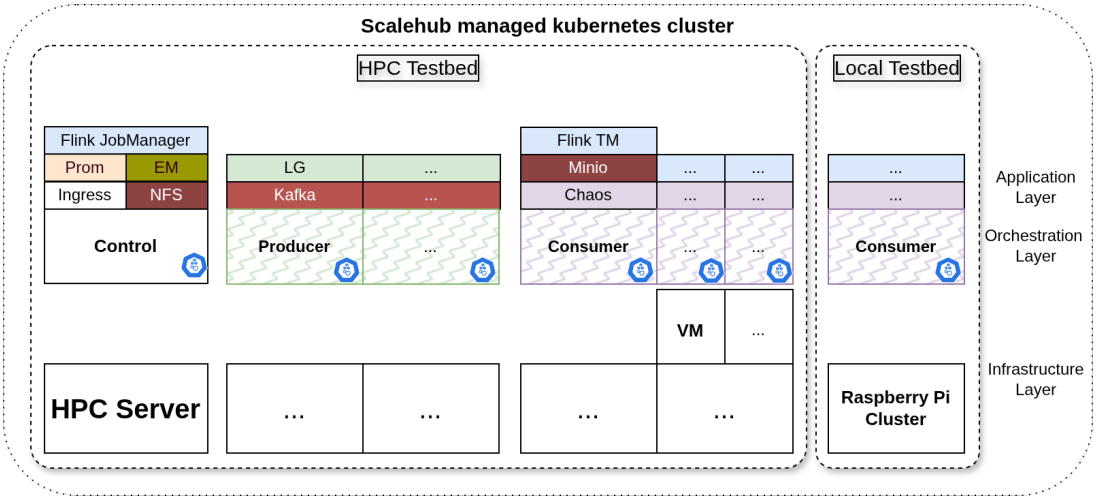


Figure 6.2 – Cluster state after application deployment, showing per-role workload distribution across heterogeneous infrastructure.

deployment. This makes it especially well-suited to edge devices with limited computational capabilities.

ScaleHub uses a role-based node isolation strategy to optimize workload placement across heterogeneous infrastructure. Producer nodes host data generation workloads, including Kafka brokers, load generators and supporting services. Consumer nodes run the DSP systems being evaluated, while being isolated from the overhead of data generation. Control nodes oversee cluster operations, infrastructure monitoring and experiment orchestration.

Node labelling, Kubernetes selectors, and affinity rules enforce this separation. Multiple TaskManager templates with different resource configurations ensure optimal placement — lightweight configurations are used for edge devices and more powerful ones are used for bare-metal servers.

Application Deployment

The application layer enables the complete experimental stack required for DSP evaluation to be deployed using Infrastructure-as-Code. Figure 6.2 illustrates the per-role workload distribution after application deployment.

Each component is implemented as a dedicated Ansible role, enabling researchers with specific requirements to easily extend the system. The default stack uses Apache Flink as the DSP system under test, but researchers can substitute this with alternative frameworks

by adding the relevant roles. The supporting infrastructure includes MinIO for distributed object storage; the full Kafka ecosystem for message streaming; Prometheus and Grafana for monitoring; and load generators for benchmarking.

A containerized experiment-monitoring module acts as an experiment orchestrator, overseeing scaling operations and collecting performance data. This allows the scalability of DSP systems to be assessed across heterogeneous environments by coordinating experiments that span nodes with different computational capabilities and network conditions.

6.2.3 Chaos Engineering Integration

Chaos engineering involves proactively testing system resilience by introducing controlled failures, as opposed to reactively addressing issues that occur naturally [66]. Netflix is an example of an organization that runs chaos engineering experiments using their Simian Army tool [155, 156].

Chaos Mesh, a CNCF incubating project, provides a declarative approach to simulating failures in Kubernetes environments through custom resources [202]. The platform can simulate pod failures, network disruptions, I/O anomalies, kernel faults, and resource contention scenarios. This approach allows integration of failure testing into development workflows and continuous integration pipelines.

In ScaleHub, Chaos Mesh is deployed as part of the K3s cluster infrastructure, enabling the execution of controlled network condition experiments on DSP applications across heterogeneous nodes. This integration supports the simulation of constrained edge network behaviours that may not be easily replicable in traditional testing environments. Behaviours such as increased latency, variable latency, and bandwidth limitations can be introduced between specific node roles (e.g. consumers) or across the entire cluster. This provides valuable insights into how applications behave under diverse failure conditions and how heterogeneous environments affect the performance and scalability of distributed applications, as seen in Chapter 4.

6.3 Experiment Orchestration and Monitoring

This section presents the experiment orchestration and monitoring capabilities of ScaleHub, focusing on the automated management of DSP experiment lifecycles and the collection of performance metrics during scaling operations.

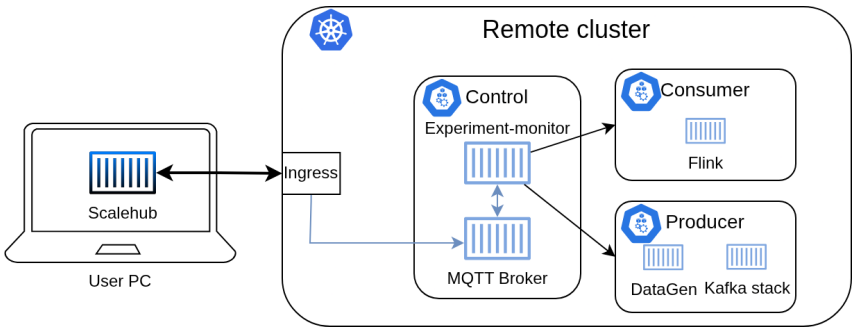


Figure 6.3 – Experiment-monitor module architecture showing MQTT-based command interface and component lifecycle management.

6.3.1 Experiment-Monitor Architecture

The experiment-monitor module serves as the central orchestrator for DSP experiments within ScaleHub. This containerized Python application runs inside the K3s cluster and manages the complete lifecycle of experimental campaigns, from DSP application deployment to result collection.

Figure 6.3 shows the experiment-monitor architecture. The module connects to an MQTT broker to receive commands from the ScaleHub orchestrator running outside the cluster. This design enables a "start-and-forget" approach for experiments, which is particularly useful for long-running or overnight experiments where users do not need to monitor the terminal continuously during experiment runtime.

One of the core mechanisms implemented in the experiment-monitor is a Finite State Machine (FSM). The experiment-monitor is a multithreaded application that handles MQTT connections on one side (listening for START/STOP commands with serialized experiment configurations) and runs the FSM with the received configuration on the other side.

6.3.2 Experiment Lifecycle Management

Figure 6.4 illustrates the FSM that governs experiment execution consisting of four distinct states: IDLE, STARTING, RUNNING, and FINISHING.

The system processes one experiment at a time, with new experiments only accepted when the FSM resides in the IDLE state. During the STARTING phase, the system validates all prerequisites for experiment execution, including cluster resource availability and application requirements. If validation fails, the system cleans up the provisioned

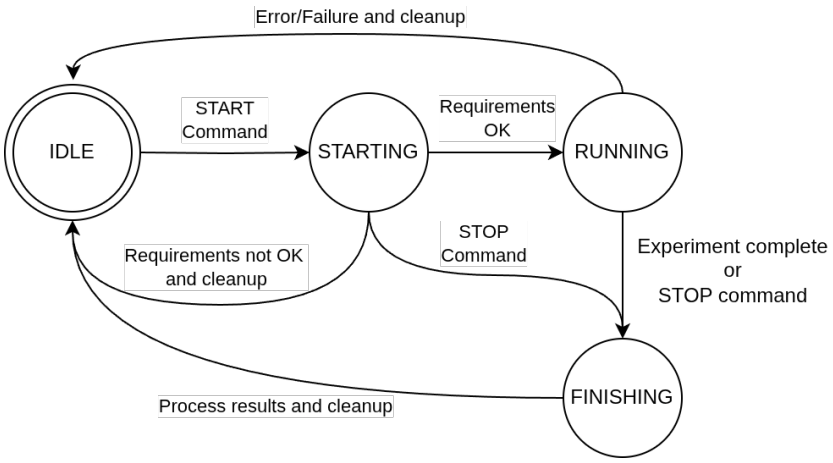


Figure 6.4 – Finite State Machine governing experiment lifecycle in the experiment-monitor module.

resources and transitions back to IDLE; otherwise, it proceeds to the RUNNING state. The RUNNING phase orchestrates the actual experimental campaign according to the received configuration specification. Should errors occur during execution, the system cleans up and then transitions to IDLE. Upon successful experiment completion or reception of a STOP command, the FSM transitions to FINISHING, where metrics are extracted from the VictoriaMetrics database and processed to generate comprehensive analysis plots.

For Flink-based experiments, the module interfaces with the JobManager to submit DSP applications and continuously monitor their execution status throughout the experimental campaign. The system orchestrates job lifecycle management, including graceful shutdowns during scaling operations and savepoint-based state recovery for stateful stream processing applications. This ensures seamless state preservation across dynamic parallelism adjustments while maintaining data consistency guarantees.

6.3.3 Scaling Operations

Scaling scenarios follow configurable patterns defined in experiment specifications. ScaleHub implements static time-based autoscaling to assess scalability performance under different compute and network variability scenarios. The system supports three distinct scaling patterns for adding or removing operator replicas: linear scaling (± 1 replica increments up to the desired parallelism level), exponential scaling (± 1, ± 2, ± 4, ± 8 replica increments), and block scaling ($\pm$n replicas applied simultaneously). Additionally, adap-

tive autoscalers such as TransScale can be integrated to handle dynamic scaling decisions based on runtime conditions rather than predetermined time-based schedules [165].

When a scaling operation is triggered, ScaleHub orchestrates a complex rescaling workflow to maintain application state consistency. First, the system creates a savepoint in MinIO object storage to preserve the current application state. The running Flink job is then gracefully stopped using the Flink CLI, ensuring all in-flight data are properly processed and checkpointed. Subsequently, the Flink StatefulSet is scaled via Kubernetes API calls to adjust the number of TaskManager replicas according to the target parallelism level. Once the new TaskManager instances are available, the job is restarted from the previously created savepoint with the updated parallelism configuration. The experiment-monitor continuously verifies that the job reaches a running state before proceeding with the experimental campaign. This process repeats iteratively until the desired scaling trajectory defined in the experiment specification is achieved.

6.3.4 Data Collection and Analysis

During an experiment, Prometheus collects metrics into a VictoriaMetrics database [169,208]. These metrics capture throughput and parallelism levels during experiment execution. When the experiment finishes, metrics are exported from VictoriaMetrics and processed using pandas and matplotlib to produce comprehensive experiment plots.

The module exports results in multiple formats: JSON for programmatic analysis, CSV for statistical tools, and PNG plots for immediate visualization. Grafana dashboards provide real-time monitoring during experiments, while generated reports enable detailed offline analysis and comparison between experimental campaigns [89].

6.4 Conclusion

ScaleHub represents a step forward in addressing the challenges of evaluating Data Stream Processing (DSP) systems in heterogeneous edge-cloud environments. By leveraging Infrastructure-as-Code principles and integrating tools such as Enoslib, Ansible, and K3s, ScaleHub enables the seamless provisioning of hybrid testbeds that combine local clusters with large-scale research infrastructure. This approach bridges the gap between accessibility and scalability, providing researchers with a repeatable and realistic environment for DSP system experimentation.

The inclusion of Chaos Mesh further enhances ScaleHub's capabilities by enabling controlled heterogeneous network injection across nodes. This allows for the evaluation of DSP systems under dynamic network latency scenarios, offering insights into their behaviour and reliability in edge-cloud scenarios.

ScaleHub's modular design and support for distributed storage solutions such as MinIO, along with its role-based workload isolation strategy, make it a versatile tool for deploying and benchmarking DSP applications.

While ScaleHub addresses many limitations of existing testbeds, future work could focus on expanding its support for additional testbed platforms, enhancing its automation capabilities, and integrating advanced monitoring tools for real-time performance analysis.

ScaleHub source code is available at: https://github.com/ark-sys/scalehub

Chapter 7

Conclusion

7.1 Summary

The proliferation of Internet of Things (IoT) devices and edge computing applications has created an urgent need for distributed data stream processing (DSP) systems that can operate effectively across heterogeneous computing environments. Traditional DSP frameworks such as Apache Flink are primarily designed for homogeneous cloud deployments, where nodes have homogeneous computational capabilities and predictable network conditions. However, the shift toward edge-cloud continuum deployments introduces significant challenges: heterogeneous hardware configurations, variable network latencies, and limited accessibility to realistic experimental environments for researchers.

This thesis addresses the fundamental challenge of evaluating DSP system performance in heterogeneous edge-cloud environments. We identified three critical sub-problems that hinder effective DSP deployment at the edge: (1) the impact of heterogeneous network conditions on stateful operator scalability and the reliability of traditional elasticity signals; (2) performance limitations imposed by heterogeneous computing resources and suboptimal resource allocation strategies; and (3) the lack of accessible, realistic testbed environments that combine local experimentation capabilities with large-scale research infrastructure.

First contribution: We conducted a comprehensive experimental evaluation of Apache Flink's behaviour in geo-distributed environments with heterogeneous network conditions. Our analysis revealed that network latency significantly disrupts the performance of stateful operators and compromises the reliability of backpressure as a scaling signal. We demonstrated that traditional auto-scaling approaches such as Gesscale, while effective for stateless operators, exhibit reduced prediction accuracy for stateful workloads in variable network environments. This finding challenges the common assumption that backpressure-based scaling decisions remain valid across diverse deployment scenarios.

Second contribution: We systematically evaluated DSP performance across het-

erogeneous computing nodes, ranging from resource-constrained Raspberry Pi devices to high-performance servers. Our experiments first revealed that the optimal way to maximize resource utilization on a single machine is through multiple single-slot TaskManagers with dual-core task slots, achieving up to 3.5× better throughput compared to traditional single TaskManager configurations with many slots. This improvement stems from better JVM-level resource isolation, reduced garbage collection overhead (3× shorter GC times), and improved thread distribution across processes. However, when combining heterogeneous machines in a distributed deployment, we demonstrated that Flink's resource-agnostic task assignment and hash-based data partitioning strategies create significant bottlenecks. Mixing small edge devices with powerful servers fundamentally limits system-wide performance, as the hash-based message routing fails to account for differences in computational capabilities and network connectivity between nodes. Even with optimal per-machine TaskManager configurations, architectural heterogeneity prevents the system from effectively exploiting the full capacity of high-performance servers when they must coordinate with resource-constrained devices.

Third contribution: We designed and implemented ScaleHub, a Python-based testbed orchestration framework that enables infrastructure-as-code provisioning across heterogeneous hardware platforms for DSP experimentation. ScaleHub automatically deploys ephemeral Kubernetes clusters using K3s, combining different types of nodes from edge devices to cloud servers, allowing benchmarking frameworks to run in computationally heterogeneous environments. The framework provides heterogeneous network condition emulation through Chaos Mesh and uses Tailscale VPN to establish secure mesh networking between geographically dispersed nodes. Our experimental validation demonstrates that current DSP systems require fundamental adaptations to operate effectively in heterogeneous edge environments, and that an accessible heterogeneous testbed infrastructure is essential for advancing research in this domain.

7.2 Future directions

The contributions presented in this thesis address several challenges in heterogeneous geo-distributed stream processing, but they also reveal opportunities for further research. Based on our experimental findings and analysis, we identify three key areas that warrant continued investigation to advance the state of data stream processing in edge-cloud environments.

7.2.1 Reliable Backpressure Mechanisms in Heterogeneous Networks

As introduced in Section 2.3.2, backpressure mechanisms in stream processing systems regulate data flows to prevent overload conditions by propagating signals upstream when downstream operators cannot keep pace. However, our findings in Chapter 4 demonstrate that backpressure becomes an unreliable signal for scaling decisions in heterogeneous network environments where variable latency can create false bottleneck indications.

The core issue is that current backpressure mechanisms were designed for homogeneous datacenter environments with predictable, low-latency network characteristics. In edge-cloud environments, network-induced delays can trigger backpressure signals even when computational resources are underutilized, leading to incorrect scaling decisions and suboptimal resource allocation.

Future work should develop network-aware backpressure mechanisms that can distinguish between computational bottlenecks and network-induced delays. This requires extending current monitoring approaches to capture network-specific metrics such as round-trip times, packet loss rates, and bandwidth utilization, and incorporating these metrics into backpressure detection algorithms.

Additionally, adaptive buffering strategies could help mitigate the impact of network variability on backpressure propagation. Rather than using fixed buffer sizes, systems could dynamically adjust buffer allocation based on observed network characteristics, providing larger buffers for high-latency connections while maintaining low latency for local processing.

Integrating network status signals into backpressure-based autoscaling decisions could improve scaling accuracy. By weighting backpressure signals against metrics such as inter-node latency and bandwidth utilization, autoscaling controllers could distinguish between congestion caused by insufficient computational resources and congestion induced by network conditions, leading to more appropriate scaling actions.

7.2.2 Resource-Aware Stream Processing Frameworks

Our evaluation in Chapter 5 revealed that current DSP systems such as Apache Flink use resource-agnostic approaches to task assignment and data partitioning, as discussed in Section 2.3.3. These approaches distribute workload uniformly across available nodes without considering their computational capabilities, leading to significant performance

limitations in heterogeneous environments where nodes with vastly different processing capacities coexist.

The fundamental limitation lies in the assumption that all nodes have equivalent processing capacity, which does not hold in edge-cloud environments. Our experiments demonstrated that this resource-agnostic approach can result in resource-constrained nodes becoming bottlenecks that limit the performance of the entire system, even when more capable nodes remain underutilized.

Future research should focus on developing truly resource-aware stream processing frameworks that automatically adapt to heterogeneous environments. This includes implementing capacity-aware data partitioning strategies where the key space or data distribution is proportional to the computational capacity of each node, ensuring that all machines contribute optimally to processing workload.

Such frameworks would require sophisticated resource discovery and monitoring mechanisms that can continuously assess the processing capabilities of different nodes, including CPU performance, memory capacity, and network connectivity. This information would then inform dynamic workload distribution algorithms that can adapt as cluster composition changes or as node performance varies due to environmental factors.

Additionally, resource-aware frameworks should support heterogeneity-conscious operator placement, where different types of operations are assigned to nodes based on their suitability for specific computational patterns. For example, memory-intensive operations could be preferentially placed on nodes with larger memory capacity, while compute-intensive operations could be assigned to nodes with more powerful processors.

The development of such frameworks would represent a significant advancement toward making stream processing systems truly suitable for the heterogeneous, dynamic environments characteristic of modern edge-cloud deployments.

7.2.3 Cross-Platform DSP Evaluation Framework

While ScaleHub enables heterogeneous testbed provisioning, comparing DSP system behavior across different frameworks remains difficult. Existing benchmarking tools such as Theodolite focus on scalability evaluation within individual systems, while NAMB provides cross-platform application generation, but neither addresses the complete workflow needed for systematic comparison in heterogeneous environments.

A cross-platform evaluation framework could extend ScaleHub with templated applications written for major DSP systems (Flink, Storm, Spark Streaming) that researchers

can adapt to their needs. Standard benchmarks such as the DEBS challenges and Yahoo Streaming Benchmark could be provided in multiple framework implementations, enabling direct performance comparison under identical conditions. The framework would orchestrate deployment across different DSP systems while maintaining consistent heterogeneous network and compute conditions through the existing Chaos Mesh integration.

A critical missing component is a configurable workload generator that allows fine-grained control over data characteristics including volume, velocity, and variety. This generator would need to produce data in formats compatible with different application templates, enabling researchers to evaluate how various DSP systems handle changing workload patterns in heterogeneous deployments. Combined with ScaleHub's heterogeneous testbed provisioning, this would provide an end-to-end platform for reproducible cross-system evaluation.

Bibliography

[1] ACM DEBS Grand Challenge. ACM DEBS Grand Challenge. https://debs.org/grand-challenges/.

[2] Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., and Watteyne, T. FIT IoT-LAB: A large scale open experimental IoT testbed. In *Proc. IEEE WF-IoT* (2015).

[3] Agnihotri, P., Koldehofe, B., Heinrich, R., Binnig, C., and Luthra, M. PDSP-Bench: A benchmarking system for parallel and distributed stream processing. In *Proc. TPCTC* (2024).

[4] Ahmed, A., Arkian, H., Battulga, D., Fahs, A. J., Farhadi, M., Giouroukis, D., Gougeon, A., Guttierrez, F. O., Pierre, G., Souza Jr, P. R., Ayalew Tamiru, M., and Wu, L. Fog computing applications: Taxonomy and requirements. *CoRR abs/1907.11621* (2019). http://arxiv.org/abs/1907.11621.

[5] Al-Dulaimy, A., Jansen, M., Johansson, B., Trivedi, A., Iosup, A., Ashjaei, M., Galletta, A., Kimovski, D., Prodan, R., Tserpes, K., Kousiouris, G., Giannakos, C., Brandic, I., Ali, N., Bondi, A. B., and Papadopoulos, A. V. The computing continuum: From IoT to the cloud. *Internet of Things 27* (2024), 101272.

[6] Aljoby, W., Wang, X., Fu, T. Z., and Ma, R. T. On SDN-enabled online and dynamic bandwidth allocation for stream analytics. *IEEE Journal on Selected Areas in Communications 37*, 8 (2019).

[7] Amarasinghe, G., De Assuncao, M. D., Harwood, A., and Karunasekera, S. A data stream processing optimisation framework for edge computing applications. In *Proc. IEEE ISORC* (2018).

[8] Amazon Web Services. Amazon Web Services (AWS) – cloud computing services. https://aws.amazon.com.

[9] Ansible Authors. Ansible: Automation for everyone. https://www.ansible.com/.

[10] Apache Software Foundation. Apache Flink® – Flink architecture. https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/concepts/flink-architecture.

[11] Apache Software Foundation. Apache Flink® – Flink keyed stream. https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/dev/datastream/fault-tolerance/state/#keyed-datastream.

[12] Apache Software Foundation. Apache Flink® – Flink Kubernetes. https://nightlies.apache.org/flink/flink-docs-master/docs/deployment/resource-providers/standalone/kubernetes/.

[13] Apache Software Foundation. Apache Flink® – Flink task lifecycle. https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/internals/task_lifecycle.

[14] Apache Software Foundation. Apache Flink® – slot sharing groups. https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/deployment/finegrained_resource.

[15] Apache Software Foundation. Apache Flink® – stateful computations over data streams. https://flink.apache.org.

[16] Apache Software Foundation. Apache Flink® – a deep dive into rescalable state in Apache Flink. https://flink.apache.org/2017/07/04/a-deep-dive-into-rescalable-state-in-apache-flink.

[17] Apache Software Foundation. Apache Flink® – checkpointing under backpressure. https://nightlies.apache.org/flink/flink-docs-release-1.18/docs/ops/state/checkpointing_under_backpressure.

[18] Apache Software Foundation. Apache Flink® – peeking into Apache Flink's engine room. https://flink.apache.org/2015/03/13/peeking-into-apache-flinks-engine-room.

[19] Apache Software Foundation. Apache Samza®˙https://samza.apache.org.

[20] Apache Software Foundation. Apache Spark® – unified engine for large-scale data analytics. https://spark.apache.org.

[21] Apache Software Foundation. Apache Storm®˙https://storm.apache.org.

[22] Apache Software Foundation. Stateful stream processing. https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/concepts/stateful-stream-processing.

[23] Arkian, H. *Resource management for data stream processing in geo-distributed environments*. PhD thesis, Université de Rennes 1, 2021.

[24] Arkian, H., Pierre, G., Tordsson, J., and Elmroth, E. An experiment-driven performance model of stream processing operators in fog computing environments. In *Proc. ACM SAC* (2020).

[25] Arkian, H., Pierre, G., Tordsson, J., and Elmroth, E. Model-based stream processing auto-scaling in geo-distributed environments. In *Proc. ICCCN* (2021).

[26] Armah, E., and Bannning, L. A. Towards a proactive autoscaling framework for data stream processing at the edge using GRU and transfer learning. *arXiv preprint arXiv:2507.14597* (2025).

[27] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., and Stoica, I. Above the clouds: A Berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, University of California, 2009.

[28] Arsalane, K., Pierre, G., and Ibrahim, S. Toward stream processing elasticity in realistic geo-distributed environments. In *Proc. IEEE IC2E* (2024).

[29] Badia, S., Carpen-Amarie, A., Lèbre, A., and Nussbaum, L. Enabling large-scale testing of IaaS cloud platforms on the Grid'5000 testbed. In *Proc. ACM WTSC* (2013).

[30] Balouek, D., Amarie, A. C., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lèbre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Pérez, C., Quesnel, F., Rohr, C., and Sarzyniec, L. Adding virtualization capabilities to the Grid'5000 testbed. In *Proc. IEEE CLOUDS* (2012).

[31] Barati, M., and Rana, O. Tracking GDPR compliance in cloud-based service delivery. *IEEE Transactions on Services Computing 15*, 3 (2020).

[32] Bartley, K. Big data statistics: How much data is there in the world?, 2024. https://rivery.io/blog/big-data-statistics-how-much-data-is-there-in-the-world.

[33] Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., and Rosenthal, C. Chaos engineering. *IEEE Software 33*, 3 (2016).

[34] Battulga, D., Farhadi, M., Tamiru, M. A., Wu, L., and Pierre, G. LivingFog: Leveraging fog computing and LoRaWAN technologies for smart marina management (experience paper). In *Proc. IEEE ICIN* (2022).

[35] Battulga, D., Miorandi, D., and Tedeschi, C. FogGuru: a fog computing platform based on Apache Flink. In *Proc. IEEE ICIN* (2020).

[36] Beckman, P., Dongarra, J., Ferrier, N., Fox, G., Moore, T., Reed, D., and Beck, M. Harnessing the computing continuum for programming our world. *Fog Computing: Theory and Practice* (2020).

[37] Bermbach, D., Pallas, F., Pérez, D. G., Plebani, P., Anderson, M., Kat, R., and Tai, S. A research perspective on fog computing. In *Proc. ICSoC* (2017).

[38] Bhardwaj, A., and Krishna, C. R. Virtualization in cloud computing: Moving from hypervisor to containerization—a survey. *Arabian Journal for Science and Engineering 46*, 9 (2021).

[39] Birman, K. P. *Reliable distributed systems: technologies, web services, and applications*. Springer, 2005.

[40] Böhm, S., and Wirtz, G. Profiling lightweight container platforms: MicroK8s and k3s in comparison to kubernetes. In *Proc. ZEUS* (2021).

[41] Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.-G., and Touche, I. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *The International Journal of High Performance Computing Applications 20*, 4 (2006).

[42] Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. Fog computing: A platform for Internet of Things and analytics. In *Big data and Internet of Things: A roadmap for smart environments*. Springer, 2014.

[43] Buyya, R., Broberg, J., and Goscinski, A. *Cloud Computing: Principles and Paradigms*. Wiley Series on Parallel and Distributed Computing. Wiley, 2010.

[44] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandić, I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems 25*, 6 (2009).

[45] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience 41*, 1 (2011).

[46] Camero, A., and Alba, E. Smart city and information technology: A review. *Cities 93* (2019).

[47] Carbone, P., Ewen, S., Fóra, G., Haridi, S., Richter, S., and Tzoumas, K. State management in Apache Flink®: Consistent stateful distributed stream processing. *Proc. VLDB Endow. 10*, 12 (2017).

[48] Carbone, P., Fragkoulis, M., Kalavri, V., and Katsifodimos, A. Beyond analytics: The evolution of stream processing systems. In *Proc. ACM SIGMOD* (2020).

[49] Cardellini, V., Dazzi, P., Mencagli, G., Nardelli, M., and Torquati, M. Scalable compute continuum. *Future Generation Computer Systems 166* (2025).

[50] Cardellini, V., Grassi, V., Lo Presti, F., and Nardelli, M. Optimal operator replication and placement for distributed stream processing systems. *ACM SIGMETRICS Performance Evaluation Review 44*, 4 (2017).

[51] Cardellini, V., Lo Presti, F., Nardelli, M., and Russo Russo, G. Runtime adaptation of data stream processing systems: The state of the art. *ACM Computing Surveys 54*, 115 (2022).

[52] Cardellini, V., Nardelli, M., and Luzi, D. Elastic stateful stream processing in Storm. In *Proc. IEEE HPCS* (2016).

[53] Casino, F., Lopez-Iturri, P., and Patsakis, C. Cloud continuum testbeds and next-generation ICTs: Trends, challenges, and perspectives. *Computer Science Review 56* (2025).

[54] Celebi, U. How Apache Flink™ handles backpressure. Ververica blog. https://www.ververica.com/blog/how-flink-handles-backpressure.

[55] Chao, M., and Stoleru, R. R-MStorm: A resilient mobile stream processing system for dynamic edge networks. In *Proc. IEEE ICFC* (2020).

[56] Chardet, M., Coullon, H., and Pérez, C. Predictable efficiency for reconfiguration of service-oriented systems with Concerto. In *Proc. IEEE CCGRID* (2020).

[57] Chaudhary, A., Zeuch, S., and Markl, V. Governor: Operator placement for a unified fog-cloud environment. In *Proc. EDBT* (2020).

[58] Chen, X., Vigfusson, Y., Blough, D. M., Zheng, F., Wu, K.-L., and Hu, L. GOVERNOR: Smoother stream processing through smarter backpressure. In *Proc. ACM ICAC* (2017).

[59] Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Cetintemel, U., Xing, Y., and Zdonik, S. B. Scalable distributed stream processing. In *Proc. CIDR* (2003), vol. 3.

[60] Cherrueau, R.-A., Delavergne, M., Van Kempen, A., Lebre, A., Pertin, D., Balderrama, J. R., Simonet, A., and Simonin, M. EnOSlib: A library for experiment-driven research in distributed computing. *IEEE Transactions on Parallel and Distributed Systems 33*, 6 (2021), 1464–1477.

[61] Ching, C.-W., Chen, X., Kim, C., Wang, T., Chen, D., Da Silva, D., and Hu, L. AgileDART: An agile and scalable edge stream processing engine. *IEEE Transactions on Mobile Computing 24*, 5 (2025).

[62] Collins, R. L., and Carloni, L. P. Flexible filters: load balancing through backpressure for stream programs. In *Proc. ACM EMSOFT* (2009).

[63] Colman-Meixner, C., Develder, C., Tornatore, M., and Mukherjee, B. A survey on resiliency techniques in cloud computing infrastructures and applications. *IEEE Communications Surveys & Tutorials 18*, 3 (2016).

[64] Consul Authors. Consul. https://www.consul.io.

[65] Coutinho, A., Greve, F., Prazeres, C., and Cardoso, J. Fogbed: A rapid-prototyping emulation environment for fog computing. In *Proc. IEEE ICC* (2018).

[66] Craveiro, D., and Barreiros, J. Chaos engineering tool analysis. In *Proc. IEEE CISTI* (2023).

[67] da Silva Veith, A., de Assunçao, M. D., and Lefevre, L. Monte-carlo tree search and reinforcement learning for reconfiguring data stream processing on edge computing. In *Proc. IEEE SBAC-PAD* (2019).

[68] Dabek, F., Cox, R., Kaashoek, F., and Morris, R. Vivaldi: A decentralized network coordinate system. *ACM SIGCOMM Computer Communication Review 34*, 4 (2004).

[69] de Assuncao, M. D., da Silva Veith, A., and Buyya, R. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications 103* (2018).

[70] De Matteis, T., and Mencagli, G. Keep calm and react with foresight: Strategies for low-latency and energy-efficient elastic data stream processing. *ACM Sigplan Notices 51*, 8 (2016), 1–12.

[71] De Matteis, T., and Mencagli, G. Elastic scaling for distributed latency-sensitive data stream operators. In *Proc. IEEE PDP* (2017).

[72] De Matteis, T., and Mencagli, G. Proactive elasticity and energy awareness in data stream processing. *Journal of Systems and Software 127* (2017).

[73] Dean, J., and Ghemawat, S. MapReduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008).

[74] Donnet, B., Gueye, B., and Kaafar, M. A. A survey on network coordinates systems, design, and security. *IEEE Communications Surveys and Tutorials 12*, 4 (2010).

[75] Dou, R., Wang, X., and Ma, R. T. Emma: Elastic multi-resource management for realtime stream processing. In *Proc. IEEE INFOCOM* (2024).

[76] Doulkeridis, C., and Nørvåg, K. A survey of large-scale analytical query processing in MapReduce. *The VLDB journal 23*, 3 (2014).

[77] Duplyakin, D., Ricci, R., Maricq, A., Wong, G., Duerig, J., Eide, E., Stoller, L., Hibler, M., Johnson, D., Webb, K., Akella, A., Wang, K., Ricart, G., Landweber, L., Elliott, C., Zink, M., Cecchet, E., Kar, S., and Mishra, P. The design and operation of CloudLab. In *Proc. USENIX ATC* (2019).

[78] European Data Protection Board. EU Cloud Code of Conduct – GDPR compliance for cloud service providers, 2024. https://www.edpb.europa.eu/system/files/2024-02/eucloudcoc.pdf.

[79] Farhat, O., Daudjee, K., and Querzoni, L. Klink: Progress-aware scheduling for streaming data systems. In *Proc. ACM SIGMOD* (2021).

[80] Foster, I., Zhao, Y., Raicu, I., and Lu, S. Cloud computing and grid computing 360-degree compared. In *Proc. IEEE Grid computing environments workshop* (2008).

[81] Fragkoulis, M., Carbone, P., Kalavri, V., and Katsifodimos, A. A survey on the evolution of stream processing systems. *The VLDB Journal 33*, 2 (2024).

[82] Freymann, R., Shi, J., Chen, J.-J., and Chen, K.-H. Renovation of EdgeCloudSim: an efficient discrete-event approach. In *Proc. IEEE FMEC* (2021).

[83] Fu, K., Zhang, W., Chen, Q., Zeng, D., Peng, X., Zheng, W., and Guo, M. QoS-aware and resource efficient microservice deployment in cloud-edge continuum. In *Proc. IEEE IPDPS* (2021).

[84] Fu, X., Ghaffar, T., Davis, J. C., and Lee, D. EdgeWise: A better stream processing engine for the edge. In *Proc. USENIX ATC* (2019).

[85] Gedik, B., Schneider, S., Hirzel, M., and Wu, K.-L. Elastic scaling for data stream processing. *IEEE Transactions on Parallel and Distributed Systems 25*, 6 (2013).

[86] Geldenhuys, M. K., Will, J., Pfister, B. J., Haug, M., Scharmann, A., and Thamsen, L. Dependable IoT data stream processing for monitoring and control of urban infrastructures. In *Proc. IEEE IC2E* (2021).

[87] Gkonis, P., Giannopoulos, A., Trakadas, P., Masip-Bruin, X., and D'Andria, F. A survey on IoT-edge-cloud continuum systems: Status, challenges, use cases, and open issues. *Future Internet 15*, 12 (2023).

[88] Google LLC. Google Cloud Platform – cloud computing services. https://cloud.google.com.

[89] Grafana Labs. Grafana: The open observability platform. https://grafana.com/.

[90] Greenberg, A., Hamilton, J., Maltz, D. A., and Patel, P. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review 39*, 1 (2008).

[91] Grid'5000. Grid'5000 paradoxe cluster. https://www.grid5000.fr/w/Rennes:Hardware#paradoxe.

[92] Grid'5000. Grid'5000 reconfigurable firewall. https://www.grid5000.fr/w/Reconfigurable_Firewall.

[93] Grid'5000. Grid'5000 VPN. https://www.grid5000.fr/w/VPN.

[94] Gulisano, V., Jimenez-Peris, R., Patino-Martinez, M., Soriente, C., and Valduriez, P. StreamCloud: An elastic and scalable data streaming system. *IEEE Transactions on Parallel and Distributed Systems 23*, 12 (2012).

[95] Gulisano, V., Nikolakopoulos, Y., Papatriantafilou, M., and Tsigas, P. ScaleJoin: A deterministic, disjoint-parallel and skew-resilient stream join. *IEEE Transactions on Big Data 7*, 2 (2016).

[96] Gulisano, V., Papadopoulos, A. V., Nikolakopoulos, Y., Papatriantafilou, M., and Tsigas, P. Performance modeling of stream joins. In *Proc. ACM DEBS* (2017).

[97] Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., and Buyya, R. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience 47*, 9 (2017).

[98] Gutierrez, F., Beedkar, K., Souza, A., and Markl, V. AdCom: Adaptive combiner for streaming aggregations. In *Proc. ACM EDBT* (2021).

[99] He, C., Huang, Y., Wang, C., and Wang, N. Dynamic data partitioning strategy based on heterogeneous Flink cluster. In *Proc. IEEE ICAIBD* (2022).

[100] Heinze, T., Jerzak, Z., Hackenbroich, G., and Fetzer, C. Latency-aware elastic scaling for distributed data stream processing systems. In *Proc. ACM DEBS* (2014).

[101] Henning, S., and Hasselbring, W. Theodolite: Scalability benchmarking of distributed stream processing engines in microservice architectures. *Big Data Research 25* (2021).

[102] Herbst, N. R., Kounev, S., and Reussner, R. Elasticity in cloud computing: What it is, and what it is not. In *Proc. USENIX ICAC* (2013).

[103] Hidalgo, N., Wladdimiro, D., and Rosas, E. Self-adaptive processing graph with operator fission for elastic stream processing. *Journal of Systems and Software 127* (2017).

[104] Hong, C.-H., and Varghese, B. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. *ACM computing surveys 52*, 5 (2019).

[105] HoseinyFarahabady, M. R., Jannesari, A., Taheri, J., Bao, W., Zomaya, A. Y., and Tari, Z. Q-flink: A QoS-aware controller for Apache Flink. In *Proc. IEEE CCGRID* (2020).

[106] Hua, W., Liu, P., and Huang, L. Energy-efficient resource allocation for heterogeneous edge–cloud computing. *IEEE Internet of Things Journal 11*, 2 (2023).

[107] Hueske, F., and Kalavri, V. *Stream processing with Apache Flink: fundamentals, implementation, and operation of streaming applications*. O'Reilly Media, 2019.

[108] Imai, S., Patterson, S., and Varela, C. A. Maximum sustainable throughput prediction for data stream processing over public clouds. In *Proc. IEEE CCGRID* (2017).

[109] Imai, S., Patterson, S., and Varela, C. A. Uncertainty-aware elastic virtual machine scheduling for stream processing systems. In *Proc. IEEE CCGRID* (2018).

[110] Jayasekara, S., Karunasekera, S., and Harwood, A. Optimizing checkpoint-based fault-tolerance in distributed stream processing systems: Theory to practice. *Software: Practice and Experience 52*, 1 (2022).

[111] Jennings, B., and Stadler, R. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management 23* (2015).

[112] Jonathan, A., Chandra, A., and Weissman, J. Awan: Locality-aware resource manager for geo-distributed data-intensive applications. In *Proc. IEEE IC2E* (2016).

[113] Jonathan, A., Uluyol, M., Chandra, A., and Weissman, J. Ensuring reliability in geo-distributed edge cloud. In *Proc. IEEE RWS* (2017).

[114] K3s authors. K3s. https://k3s.io.

[115] Kalavri, V. Why stream processing systems are now more relevant than ever. SIGOPS blog, 2020. https://www.sigops.org/2020/streams.

[116] Kalavri, V., Liagouris, J., Hoffmann, M., Dimitrova, D., Forshaw, M., and Roscoe, T. Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows. In *Proc. USENIX OSDI* (2018).

[117] Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., and Markl, V. Benchmarking distributed stream data processing systems. In *Proc. IEEE ICDE* (2018).

[118] Kashani, M. H., Madanipour, M., Nikravan, M., Asghari, P., and Mahdipour, E. A systematic review of IoT in healthcare: Applications, techniques, and trends. *Journal of Network and Computer Applications 192* (2021).

[119] Kastrinakis, D., and Petrakis, E. G. Video2Flink: real-time video partitioning in Apache Flink and the cloud. *Machine Vision and Applications 34*, 3 (2023), 42.

[120] Kayal, P. Kubernetes in fog computing: Feasibility demonstration, limitations and improvement scope : Invited paper. In *Proc. IEEE WF-IoT* (2020).

[121] Kazem, A., Pierre, G., and Longuevergne, L. Enhancing environmental observatories with fog computing. *Frontiers in Environmental Science 13* (2025).

[122] Keahey, K., Anderson, J., Zhen, Z., Riteau, P., Ruth, P., Stanzione, D., Cevik, M., Colleran, J., Gunawi, H. S., Hammock, C., Mambretti, J., Barnes, A., Halbah, F., Rocha, A., and Stubbs, J. Lessons learned from the Chameleon testbed. In *Proc. USENIX ATC* (2020).

[123] Khalyeyev, D., Bureš, T., and Hnětynka, P. Towards characterization of edge-cloud continuum. In *Proc. ECSA* (2022).

[124] Khan, L. U., Yaqoob, I., Tran, N. H., Kazmi, S. A., Dang, T. N., and Hong, C. S. Edge-computing-enabled smart cities: A comprehensive survey. *IEEE Internet of Things journal 7*, 10 (2020).

[125] Kimovski, D., Mathá, R., Hammer, J., Mehran, N., Hellwagner, H., and Prodan, R. Cloud, fog, or edge: Where to compute? *IEEE Internet Computing 25*, 4 (2021).

[126] Kimovski, D., Saurabh, N., Jansen, M., Aral, A., Al-Dulaimy, A., Bondi, A. B., Galletta, A., Papadopoulos, A. V., Iosup, A., and Prodan, R. Beyond Von Neumann in the computing continuum: Architectures, applications, and future directions. *IEEE Internet Computing 28*, 3 (2023).

[127] Koliousis, A., Weidlich, M., Castro Fernandez, R., Wolf, A. L., Costa, P., and Pietzuch, P. Saber: Window-based hybrid stream processing for heterogeneous architectures. In *Proc. ACM SIGMOD* (2016).

[128] Kombi, R. K., Lumineau, N., and Lamarre, P. A preventive auto-parallelization approach for elastic stream processing. In *Proc. IEEE ICDCS* (2017).

[129] Koziolek, H., and Eskandani, N. Lightweight Kubernetes distributions: A performance comparison of microk8s, k3s, k0s, and microshift. In *Proc. ACM ICPE* (2023).

[130] Kp, G., Pierre, G., and Rouvoy, R. Studying the energy consumption of stream processing engines in the cloud. In *Proc. IEEE IC2E* (2023).

[131] Kruber, N. Apache Flink® – a deep dive into Flink's network stack. https://flink.apache.org/2019/06/05/a-deep-dive-into-flinks-network-stack.

[132] Kruber, N., and Nowojski, P. Apache Flink® – Flink network stack vol. 2: Monitoring, metrics, and that backpressure thing. https://flink.apache.org/2019/07/23/flink-network-stack-vol.-2-monitoring-metrics-and-that-backpressure-thing/#network-metric.

[133] Lambert, T., Guyon, D., and Ibrahim, S. Rethinking operators placement of stream data application in the edge. In *Proc. ACM CIKM* (2020).

[134] Ledlie, J., Gardner, P., and Seltzer, M. I. Network coordinates in the wild. In *Proc. USENIX NSDI* (2007).

[135] Lee, S., Zhang, Z.-L., Sahu, S., and Saha, D. On suitability of euclidean embedding for host-based network coordinate systems. *IEEE/ACM Transactions on Networking 18*, 1 (2009).

[136] Li, Y., Orgerie, A.-C., Rodero, I., Amersho, B. L., Parashar, M., and Menaud, J.-M. End-to-end energy models for edge cloud-based IoT platforms: Application to data stream analysis in IoT. *Future Generation Computer Systems 87* (2018).

[137] Liu, H., Eldarrat, F., Alqahtani, H., Reznik, A., De Foy, X., and Zhang, Y. Mobile edge cloud system: Architectures, challenges, and approaches. *IEEE Systems Journal 12*, 3 (2017).

[138] Liu, X., and Buyya, R. Resource management and scheduling in distributed stream processing systems: a taxonomy, review, and future directions. *ACM Computing Surveys 53*, 3 (2020).

[139] Makris, A., Kontopoulos, I., Psomakelis, E., Xyalis, S. N., Theodoropoulos, T., and Tserpes, K. Performance analysis of storage systems in edge computing infrastructures. *Applied Sciences 12*, 17 (2022).

[140] Malek, Y. N., Kharbouch, A., El Khoukhi, H., Bakhouya, M., De Florio, V., El Ouadghiri, D., Latré, S., and Blondia, C. On the use of IoT

and big data technologies for real-time monitoring and data processing. *Procedia computer science 113* (2017).

[141] Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. A survey on mobile edge computing: The communication perspective. *IEEE communications surveys & tutorials 19*, 4 (2017).

[142] Marcu, O.-C., Costan, A., Antoniu, G., and Pérez-Hernández, M. S. Spark versus Flink: Understanding performance in big data analytics frameworks. In *Proc. IEEE CLUSTER* (2016).

[143] Mayer, R., Graser, L., Gupta, H., Saurez, E., and Ramachandran, U. EmuFog: Extensible and scalable emulation of large-scale fog computing infrastructures. In *Proc. IEEE FWC* (2017).

[144] Mell, P., and Grance, T. The NIST definition of cloud computing. *National Institute of Standards and Technology* (2011).

[145] Mencagli, G., Dazzi, P., and Tonci, N. SpinStreams: a static optimization tool for data stream processing applications. In *Proc. ACM Middleware* (2018).

[146] Mencagli, G., Torquati, M., Cardaci, A., Fais, A., Rinaldi, L., and Danelutto, M. WindFlow: High-speed continuous stream processing with parallel building blocks. *IEEE Transactions on Parallel and Distributed Systems 32*, 11 (2021).

[147] Mencagli, G., Torquati, M., Griebler, D., Fais, A., and Danelutto, M. General-purpose data stream processing on heterogeneous architectures with WindFlow. *Journal of parallel and distributed computing 184* (2024).

[148] Méndez, S. A use case guide for building edge systems using K3s, K3OS, and Open Source Cloud Native Technologies. *Edge Computing Systems with Kubernetes* (2022).

[149] Michalke, A., Lepping, A., Markl, V., Martinez, R., Schubert, N., Schwerdtfeger, L., Tekdogan, T., Zeuch, S., Ziehn, A., Falkensteiner, C., Krüger, K., Meyer, A., Röschl, T., and Wilkending, S. Nebulastream: An extensible, high-performance streaming engine for multi-modal edge applications. In *Proc. ACM SIGMOD* (2025).

[150] Microsoft Corporation. Microsoft Azure – cloud computing platform & services. https://azure.microsoft.com.

[151] MinIO Authors. Minio. https://min.io.

[152] Moreschini, S., Moreschini, S., Pecorelli, F., Li, X., Naz, S., Hästbacka, D., and Taibi, D. Cloud continuum: The definition. *IEEE Access 10* (2022).

[153] Muñoz, R., Vilalta, R., Yoshikane, N., Casellas, R., Martínez, R., Tsuritani, T., and Morita, I. Integration of IoT, transport SDN, and edge/cloud computing for dynamic distribution of IoT analytics and efficient use of network resources. *IEEE Journal of Lightwave Technology 36*, 7 (2018).

[154] Nardelli, M., Cardellini, V., Grassi, V., and Presti, F. L. Efficient operator placement for distributed data stream processing applications. *IEEE Transactions on Parallel and Distributed Systems 30*, 8 (2019).

[155] Netflix. Chaos monkey. https://github.com/Netflix/chaosmonkey/tree/master.

[156] Netflix Technology Blog. The Netflix Simian Army. https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116.

[157] Noghabi, S. A., Cox, L., Agarwal, S., and Ananthanarayanan, G. The emerging landscape of edge computing. *GetMobile: Mobile Computing and Communications 23*, 4 (2020).

[158] O'Keeffe, D., Salonidis, T., and Pietzuch, P. Frontier: Resilient edge processing for the Internet of Things. *Proceedings of the VLDB Endowment 11*, 10 (2018).

[159] Open Container Initiative. Open container initiative. https://opencontainers.org/faq/.

[160] Oracle. Oracle Cloud Infrastructure – cloud computing services. https://www.oracle.com/cloud/.

[161] Orive, A., Agirre, A., Truong, H.-L., Sarachaga, I., and Marcos, M. Quality of service aware orchestration for cloud–edge continuum applications. *Sensors 22*, 5 (2022).

[162] OVH SAS. OVHcloud – European cloud provider. https://www.ovhcloud.com.

[163] Pagliari, A., Huet, F., and Urvoy-Keller, G. Towards a high-level description for generating stream processing benchmark applications. In *Proc. IEEE Big Data* (2019).

[164] Pagliari, A., Huet, F., and Urvoy-Keller, G. NAMB: A quick and flexible stream processing application prototype generator. In *Proc. IEEE CCGRID* (2020).

[165] Pagliari, A., and Pierre, G. TransScale: Combined-approach elasticity for stream processing in fog environments. In *Proc. IEEE MobileCloud* (2023).

[166] Pandey, V., Kipf, A., Vorona, D., Mühlbauer, T., Neumann, T., and Kemper, A. High-performance geospatial analytics in hyperspace. In *Proc. ACM SIGMOD* (2016).

[167] Pierre, G. Fog computing, from smart city services to natural environment monitoring. In *Proc. SBRC* (2025).

[168] Pietzuch, P., Ledlie, J., Shneidman, J., Roussopoulos, M., Welsh, M., and Seltzer, M. Network-aware operator placement for stream-processing systems. In *Proc. IEEE ICDE* (2006).

[169] Prometheus Authors. Prometheus: Monitoring system & time series database. https://prometheus.io/.

[170] Qayyum, T., Malik, A. W., Khattak, M. A. K., Khalid, O., and Khan, S. U. FogNetSim++: A toolkit for modeling and simulation of distributed fog environment. *IEEE Access 6* (2018).

[171] Raspberry Pi Foundation. Raspberry Pi 4B. https://www.raspberrypi.com/products/raspberry-pi-4-model-b.

[172] RedHat Research. Real-time data stream processing, 2020. https://research.redhat.com/blog/publication/real-time-data-stream-processing.

[173] Rio, E. D. How the data center site selection process works at Dropbox. Dropbox.Tech. https://dropbox.tech/infrastructure/how-the-data-center-site-selection-process-works-at-dropbox.

[174] RocksDB Authors. Rocksdb. https://rocksdb.org.

[175] Röger, H., and Mayer, R. A comprehensive survey on parallelization and elasticity in stream processing. *ACM Computing Surveys 52*, 2 (2019).

[176] Rosendo, D., Costan, A., Valduriez, P., and Antoniu, G. Distributed intelligence on the edge-to-cloud continuum: A systematic literature review. *Journal of Parallel and Distributed Computing 166* (2022).

[177] Rosinosky, G., Schmitz, D., and Rivière, E. StreamBed: capacity planning for stream processing. In *Proc. ACM DEBS* (2024).

[178] Rossi, F., Cardellini, V., Presti, F. L., and Nardelli, M. Geo-distributed efficient deployment of containers with Kubernetes. *Computer Communications 159* (2020).

[179] Russo Russo, G. Self-adaptive data stream processing in geo-distributed computing environments. In *Proc. ACM DEBS* (2019).

[180] Russo Russo, G., Cardellini, V., Casale, G., and Presti, F. L. MEAD: Model-based vertical auto-scaling for data stream processing. In *Proc. IEEE CCGRID* (2021).

[181] Russo Russo, G., Cardellini, V., and Lo Presti, F. Hierarchical auto-scaling policies for data stream processing on heterogeneous resources. *ACM Transactions on Autonomous and Adaptive Systems 18*, 4 (2023).

[182] Saurez, E., Hong, K., Lillethun, D., Ramachandran, U., and Ottenwälder, B. Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In *Proc. ACM DEBS* (2016).

[183] Sayfan, G. *Mastering Kubernetes: Master the art of container management by using the power of Kubernetes*. Packt Publishing Ltd, 2018.

[184] Schaffer, H. E. X as a service, cloud computing, and the need for good judgment. *IT Professional Magazine 11*, 5 (2009).

[185] Schmitz, D., Rosinosky, G., and Rivière, E. Justin: Hybrid CPU/Memory elastic scaling for distributed stream processing. In *Proc. DAIS* (2025).

[186] Serf Authors. Serf. https://www.serf.io/docs/internals/coordinates.html.

[187] Shi, W., and Dustdar, S. The promise of edge computing. *Computer 49*, 5 (2016).

[188] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The Hadoop distributed file system. In *Proc. IEEE MSST* (2010).

[189] Shwe, T., and Aritsugi, M. Optimizing data processing: a comparative study of big data platforms in edge, fog, and cloud layers. *Applied Sciences 14*, 1 (2024), 452.

[190] Sipola, T., Alatalo, J., Kokkonen, T., and Rantonen, M. Artificial intelligence in the IoT era: A review of edge AI hardware and software. In *Proc. IEEE FRUCT* (2022).

[191] Sisinni, E., Saifullah, A., Han, S., Jennehag, U., and Gidlund, M. Industrial Internet of Things: Challenges, opportunities, and directions. *IEEE transactions on industrial informatics 14*, 11 (2018).

[192] Sitaraman, R. K., Kasbekar, M., Lichtenstein, W., and Jain, M. Overlay networks: An Akamai perspective. *Advanced Content Delivery, Streaming, and Cloud Services* (2014).

[193] Slices. Slices-RI – slices-research infrastructure. https://www.slices-ri.eu/.

[194] Sochovsky, J., Karagiannis, V., Edinger, J., Kaaser, D., and Schulte, S. A framework for dynamic dependency-based service placement in the cloud-edge continuum. In *Proc. IEEE ICDCSW* (2024).

[195] Suárez, D., Almeida, F., Blanco, V., and Toledo, P. Containerization for heterogeneous and hybrid parallelism. In *Proc. Euro-Par* (2025).

[196] Sun, D., Fan, Y., Guan, C., Rong, J., Gao, S., and Buyya, R. An elastic reconfiguration strategy for operators in distributed stream computing systems. *The Journal of Supercomputing 81*, 5 (2025).

[197] Sun, D., Wang, Y., Sui, J., Gao, S., Rong, J., and Buyya, R. Lc-Stream: An elastic scheduling strategy with latency constraints in geo-distributed stream computing environments. *Concurrency and Computation: Practice and Experience 36*, 14 (2024).

[198] Tadakamalla, U., and Menascé, D. A. Characterization of IoT workloads. In *Proc. IEEE EDGE* (2019).

[199] Tailscale. Tailscale. https://tailscale.com/blog/how-tailscale-works.

[200] Tanenbaum, A. S., and van Steen, M. *Distributed systems*. CreateSpace Independent Publishing Platform, 2017.

[201] The Kubernetes authors. Kubernetes. https://kubernetes.io.

[202] The Linux Foundation. Chaos Mesh: a powerful chaos engineering platform for Kubernetes. https://chaos-mesh.org.

[203] Toliopoulos, T., and Gounaris, A. Adaptive distributed partitioning in Apache Flink. In *Proc. IEEE ICDEW* (2020).

[204] Ullo, S. L., and Sinha, G. R. Advances in smart environment monitoring systems using IoT and sensors. *Sensors 20*, 11 (2020).

[205] van Dongen, G., and van Den Poel, D. A performance analysis of fault recovery in stream processing frameworks. *IEEE Access 9* (2021).

[206] van Kempen, A., Crivat, T., Trubert, B., Roy, D., and Pierre, G. MEC-ConPaaS: An experimental single-board based mobile edge cloud. In *Proc. IEEE MobileCloud* (2017).

[207] Vaño, R., Lacalle, I., Sowiński, P., S-Julián, R., and Palau, C. E. Cloud-native workload orchestration at the edge: A deployment review and future directions. *Sensors 23*, 4 (2023).

[208] VictoriaMetrics. VictoriaMetrics: Simple & reliable monitoring for everyone. https://victoriametrics.com/.

[209] Vogel, A., Griebler, D., Danelutto, M., and Fernandes, L. G. Self-adaptation on parallel stream processing: A systematic review. *Concurrency and Computation: Practice and Experience 34*, 6 (2022), e6759.

[210] Wan, Z., Zhang, Z., Yin, R., and Yu, G. KFIML: Kubernetes-based fog computing IoT platform for online machine learning. *IEEE Internet of Things Journal 9*, 19 (2022).

[211] Wang, K., Floratou, A., Agrawal, A., and Musgrave, D. Spur: Mitigating slow instances in large-scale streaming pipelines. In *Proc. ACM SIGMOD* (2020).

[212] Wang, L., Fu, T. Z., Ma, R. T., Winslett, M., and Zhang, Z. Elasticutor: Rapid elasticity for realtime stateful stream processing. In *Proc. ACM SIGMOD* (2019).

[213] Wang, Y., Tari, Z., Huang, X., and Zomaya, A. Y. A network-aware and partition-based resource management scheme for data stream processing. In *Proc. IEEE IPDPS* (2019).

[214] Wolfrath, J., and Chandra, A. Efficient transmission and reconstruction of dependent data streams via edge sampling. In *Proc. IEEE IC2E* (2022).

[215] Wu, M., Sun, D., Gao, S., Li, K., and Buyya, R. Elastic scaling of stateful operators over fluctuating data streams. *IEEE Transactions on Services Computing* (2024).

[216] Xu, J., Ota, K., and Dong, M. Big data on the fly: UAV-mounted mobile edge computing for disaster management. *IEEE Transactions on Network Science and Engineering 7*, 4 (2020).

[217] Xu, J., Palanisamy, B., Wang, Q., Ludwig, H., and Gopisetty, S. Amnis: Optimized stream processing for edge computing. *Journal of Parallel and Distributed Computing 160* (2022), 49–64.

[218] Xu, Q., and Zhang, J. piFogBed: a fog computing testbed based on Raspberry Pi. In *Proc. IEEE IPCCC* (2019).

[219] Xu, W., Zhou, H., Cheng, N., Lyu, F., Shi, W., Chen, J., and Shen, X. Internet of vehicles in big data era. *IEEE/CAA Journal of Automatica Sinica 5*, 1 (2017).

[220] Yahoo. Streaming benchmarks. https://github.com/yahoo/streaming-benchmarks.

[221] Yi, S., Hao, Z., Qin, Z., and Li, Q. Fog computing: Platform and applications. In *Proc. IEEE HotWeb* (2015).

[222] Yi, S., Li, C., and Li, Q. A survey of fog computing: concepts, applications and issues. In *Proc. ACM workshop on mobile big data* (2015).

[223] Yi, X., Liu, F., Liu, J., and Jin, H. Building a network highway for big data: architecture and challenges. *IEEE Network 28*, 4 (2014).

[224] Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., and Jue, J. P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of systems architecture 98* (2019).

[225] Yue, W., Boissier, M., and Rabl, T. A survey of stream processing system benchmarks. In *Proc. TPCTC* (2024).

[226] Zamani, A. R., Balouek-Thomert, D., Villalobos, J. J., Rodero, I., and Parashar, M. An edge-aware autonomic runtime for data streaming and in-transit processing. *Future Generation Computer Systems 110* (2020).

[227] Zantalis, F., Koulouras, G., Karabetsos, S., and Kandris, D. A review of machine learning and IoT in smart transportation. *Future Internet 11*, 4 (2019).

[228] Zhang, F., Yang, L., Zhang, S., He, B., Lu, W., and Du, X. FineStream: Fine-Grained Window-Based stream processing on CPU-GPU integrated architectures. In *Proc. USENIX ATC* (2020).

[229] Zhang, Q., Cheng, L., and Boutaba, R. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications 1* (2010).

[230] Zhang, S., Zhang, F., Wu, Y., He, B., and Johns, P. Hardware-conscious stream processing: A survey. *ACM SIGMOD Record 48*, 4 (2020).

[231] Zhao, J., Li, H., Wu, C., Li, Z., Zhang, Z., and Lau, F. C. Dynamic pricing and profit maximization for the cloud with geo-distributed data centers. In *Proc. IEEE INFOCOM* (2014).

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Scalabilité du traitement des flux de données dans des environnements hétérogènes

Mot clés : traitement de flux, systèmes distribués, hétérogénéité, edge-cloud continuum

Résumé : Le traitement de flux de données (DSP) dans le continuum edge-cloud hétérogène présente des défis majeurs. L'explosion des applications IoT génère des volumes de données sans précédent nécessitant un traitement en quasi-temps réel. Le continuum edge-cloud étend les infrastructures cloud traditionnelles en intégrant des nœuds de calcul de capacités diverses, introduisant une hétérogénéité réseau et de calcul. Les systèmes DSP comme Apache Flink rencontrent des obstacles spécifiques : déséquilibres de flux, variabilité des performances, et complexité du placement d'opérateurs. Nous analysons l'impact de l'hétérogénéité réseau et de calcul sur les performances DSP, et présentons un environnement d'évaluation pour systèmes DSP hétérogènes. Ces contributions visent à améliorer la compréhension des dynamiques complexes et fournir les bases pour exploiter efficacement le traitement géo-distribué.

Title: Scalable Data Stream Processing in Heterogeneous Environments

Keywords: stream processing, distributed systems, heterogeneity, edge-cloud continuum

Abstract: Data Stream Processing (DSP) in the heterogeneous edge-cloud continuum presents major challenges. The explosion of IoT applications generates unprecedented data volumes requiring near real-time processing. The edge-cloud continuum extends traditional cloud infrastructures by integrating computing nodes of diverse capacities, introducing network and compute heterogeneity. DSP systems such as Apache Flink encounter specific obstacles: flow imbalances, performance variability, and operator placement complexity. We analyze the impact of network and compute heterogeneity on DSP performance, and present an evaluation environment for heterogeneous DSP systems. These contributions aim to improve understanding of complex dynamics and provide foundations for effectively exploiting geo-distributed processing.