applied sciences

MDPI

Article

A Unified Vendor-Agnostic Solution for Big Data Stream Processing in a Multi-Cloud Environment

Thalita Vergilio *, Ah-Lian Kor and Duncan Mullier

School of Built Environment, Engineering and Computing, Leeds Beckett University, Leeds LS6 3QS, UK; a.kor@leedsbeckett.ac.uk (A.-L.K.); d.mullier@leedsbeckett.ac.uk (D.M.)
* Correspondence: t.vergilio@leedsbeckett.ac.uk

Abstract: The field of cloud computing has witnessed tremendous progress, with commercial cloud providers offering powerful distributed infrastructures to small and medium enterprises (SMEs) through their revolutionary pay-as-you-go model. Simultaneously, the rise of containers has empowered virtualisation, providing orchestration technologies for the deployment and management of large-scale distributed systems across different geolocations and providers. Big data is another research area which has developed at an extraordinary pace as industries endeavour to discover innovative and effective ways of processing large volumes of structured, semi-structured, and unstructured data. This research aims to integrate the latest advances within the fields of cloud computing, virtualisation, and big data for a systematic approach to stream processing. The novel contributions of this research are: (1) MC-BDP, a reference architecture for big data stream processing in a containerised, multi-cloud environment; (2) a case study conducted with the Estates and Sustainability departments at Leeds Beckett University to evaluate an MC-BDP prototype within the context of energy efficiency for smart buildings. The study found that MC-BDP is scalable and fault-tolerant across cloud environments, key attributes for SMEs managing resources under budgetary constraints. Additionally, our experiments on technology agnosticism and container co-location provide new insights into resource utilisation, cost implications, and optimal deployment strategies in cloud-based big data streaming, offering valuable guidelines for practitioners in the field.

Keywords: multi-cloud; big data; containers; reference architecture; stream; vendor lock-in

Citation: Vergilio, T.; Kor, A.-L.; Mullier, D. A Unified Vendor-Agnostic Solution for Big Data Stream Processing in a Multi-Cloud Environment. *Appl. Sci.* **2023**, *13*, 12635. https://doi.org/10.3390/app132312635

Academic Editors: Ce Li and Bei Guan

Received: 16 October 2023
Revised: 13 November 2023
Accepted: 21 November 2023
Published: 23 November 2023

1. Introduction

1.1. Research Context

The advent of integrated yet pervasive systems such as the Internet of Things (IoT), the Internet of Everything (IoE), Fog/Edge computing, and Cloud computing has brought big data to the forefront of technology research. Smartphones, tablets, GPS trackers, sensors, and video surveillance devices produce vast amounts of data in varying formats in real time. Consequently, this poses challenges not only in terms of storage, but also in terms of timely processing and analysis for intelligence. Big data-related challenges are related to high volume, velocity, veracity, and variety. Historically, systems which focused on the volume aspect of big data appeared first. Those are known as batch architectures, as static finite data is stored in files and processed in batches. The most prominent and most highly utilised of such systems is Hadoop, an open-source distributed processing system based on the MapReduce algorithm developed by Google [1]. Hadoop is still currently popular but has evolved to include a complete ecosystem of open-source applications for batch data processing. However, the main critique of Hadoop and other batch systems is that they overlook two important aspects of big data: velocity and the fact that the data source is potentially infinite [2]. Batch systems were not designed to process streaming data and produce results in real time or close-to-real time. Following the first generation of big data processing systems, which focused on the processing of batch data, stream systems

were developed to process a potentially infinite source of data arriving at high velocities in close-to-real time. The focus of these systems was different; therefore, a new architectural design and approach to data processing was necessary. The concept of a window was introduced, which involved breaking up the data streams into manageable finite quanta for processing. This was facilitated by applying two functions: (1) assign a timestamp to arriving data; (2) based on the timestamp, assign data to a time window for processing. Thus, time windows are abstracted groups (with a finite number of records) used for processing [3]. In order to address late and out-of-order streaming data, watermarks were introduced to represent the time by which all records for a given window are expected to have arrived [4]. They are used, in conjunction with a defined tolerance, to signal that a window is ready for processing.

The Lambda architecture challenged the two main big data paradigms, namely, the batch and stream models, to propose a model where both technology stacks worked synergistically. Both batch and stream stacks were to be separately maintained, and data processing would be independently undertaken, with the results subsequently merged to obtain a combined view of the data. Although this architecture has been critiqued due to the need to create and maintain complex processing code in two separate places, thus incurring additional operational managing costs for two different technology stacks [5], it is still deployed in real-world applications (e.g., Facebook [6] and Twitter [7]). Though our proposed reference architecture does not completely address the Lambda architecture overhead-related problem, it exploits the use of a super-framework to enable the same processing code to be run on batch or stream infrastructures, thus resolving the code duplication issue. Additionally, it advocates multi-tenancy so that both batch and stream frameworks share the same cluster. Currently, there are existing hybrid architectures where batch-based architectures have been adapted to process streaming data, as well as vice-versa, where stream-based ones have been adapted to process data from batch files.

Although, currently, a plethora of big data architectures is available, there is a discernible need for the development of reference architectures, defined in this study as template architectures for a given domain. A reference architecture is based on best practices and serves as a guide for the design and implementation of concrete architectures. Examples of numerous published domain-specific reference architectures for big data built to address this limitation are: BDWFMSs designed for the domain of data-intensive scientific workflows [8], ref. [9]'s healthcare-related reference architecture, and ref. [10]'s national security-related reference architecture. Approaches to develop an SME big data strategy tend to prioritise simplicity and convenience by focusing on big data as a managed service, whilst overlooking the ominous risk of vendor lock-in. As an example, ref. [11] highlighted the complexity of big data implementations as a significant barrier to SME adoption. They have proposed a big data analytics as a service (BDAaaS) model for the SME market. A similar cloud consumption model, data as a service (DaaS) [12], envisions big data cloud analytics as a crucial strategy to give SMEs a competitive advantage in the market. Both these models, however, are dedicated forms of SaaS which could incur a high risk of vendor lock-in.

Ref. [13]'s research findings reveal that a significant barrier to the adoption of big data solutions by British manufacturing SMEs is the high cost of switching solutions. They suggest that these companies need not only appropriate tools, but also appropriate guidance when implementing an architecture for big data [13]. Big data challenges and potential within the SME market have been reviewed [14], while investigation of the opportunities for innovation created through big data has been conducted [15]. However, to date, a high-level vendor-agnostic blueprint to cater to these requirements does not exist. This is an identified gap which our research aims to address, namely, the creation of a reference architecture for big data stream processing targeted at SMEs and therefore specifically aimed at maximising economies of scale via commercial clouds [16,17]. For illustration, a study carried out by [18] using quantitative and qualitative methods on a target population of over a hundred UK-based companies from different sectors showed

that 71% of respondents saw the risk of vendor lock-in as a possible deterrent to cloud migration. Although their research also indicated a general lack of maturity in the sector and lack of familiarity with the concept of vendor lock-in, it revealed that UK decision-makers were essentially cautious about trusting a single commercial provider with their business-critical data [18]. This study's proposal of a reference architecture for big data using a multi-cloud setup is therefore motivated by its focus on smaller implementers and supported by relevant research in the area.

Currently, cloud computing business solutions are being made available to SMEs on a pay-as-you-go model. The cloud has brought with it economies of scale, making it more cost-effective for smaller companies to commission technology as services instead of purchasing hardware and maintaining their in-house IT staff. Such entry cost reduction has enabled more widespread adoption of computing resources to automate processes, satisfying significantly increased data processing needs. Undeniably, more data means greater complexity, but also the potential for more accurate predictions and increased turnover. The profitability of big data initiatives is corroborated by an independent research study of 559 companies from various sectors conducted by the Business Application Research Centre [19]. The findings reveal that the participating companies reported an average eight percent increase in revenue and ten percent decrease in costs as a result of their use of big data [19]. However, most of them own in-house-hosted technologies and are still sceptical about transferring their businesses to the cloud [19]. Given the elevated cost of big data implementations [20,21], one could speculate that the cloud's economies of scale could potentially raise the reported profits.

The following three main service models describe how cloud resources are commissioned: IaaS (infrastructure as a service), PaaS (platform as a service), and SaaS (software as a service). The more specialised a model is, the greater the economies of scale since the provider is responsible for commissioning and maintaining more resources at different levels. However, as a service model becomes more specialised, it also becomes less flexible and reliant on specific, if not proprietary, technology. However, a cloud consumer is understood to be vendor locked-in if they perceive better contract opportunities elsewhere in the market but is hindered from breaking an existing agreement with a provider. Vendor lock-in is a major cause of concern for potential cloud adopters [20,22–24], and its mitigation is one of the motivating factors for the current research. Recently, significant container-based virtualisation research has been conducted [25–28]. As containers provide decoupling between applications and the platforms to which they are deployed, they offer a level of mitigation against vendor lock-in. Furthermore, containers can be deployed on a PaaS (instead of IaaS) model because they represent a higher virtualisation level compared to virtual machines [25,29]. When container and cloud computing technologies are employed together, they afford greater economies of scale and facilitate greater collaboration amongst developers. As an example, images representing applications which are developed as containerised services have been shared in public repositories and libraries [30]. Such images can be deployed on any platform hosted on any infrastructure (cloud or non-cloud-based). Finally, the use of container and container orchestration technology also enables multi-cloud setups, which greatly mitigates the risk of vendor lock-in, giving cloud consumers the ability to compare and combine individual offers from providers to create the most appropriate and preferred configuration [21,28]. Should changes in prices or service level agreements render it advantageous to do so, resources may be easily and seamlessly transferred between providers.

1.2. Aim and Objectives

The aim of this research is to propose a unified vendor-agnostic stream processing solution for big data analytics in a multi-cloud environment. It is supported by the following research objectives: RO1—examine several existing architectures for big data processing and systematically evaluate the architectures of three well-known real-world companies (i.e., Facebook, Twitter, and Netflix) with the purpose of understanding non-functional

requirements for real-world big data systems. The selection criteria for these companies, the methodology, and the results of this evaluation are explained in a separate publication [31]; RO2—based on the findings in RO1, propose a new reference architecture based on industry's best practices and focused on vendor lock-in mitigation; RO3—develop a prototype based on a case study to enable the empirical evaluation of the proposed reference architecture; RO4: identify and implement a set of relevant performance metrics for the prototype in RO3; and RO5—design and execute distinct sets of experiments to evaluate the proposed reference architecture in terms of the non-functional requirements implemented.

1.3. Rationale

The risk of vendor lock-in is perceived as one of the greatest obstacles to cloud adoption by companies [18,20,24,32]. Consequently, as this study adopts a cloud consumer's perspective, the mitigation of vendor lock-in remains an important motivation throughout its development. The use of open-source technology is associated with a reduction in vendor lock-in risk [33] while simultaneously promoting collaboration and reuse amongst the community [34]. Another way of mitigating vendor lock-in-related risks is by hosting the cloud infrastructure in multiple clouds. From a business perspective, implementers are less vulnerable to unilateral changes in price or SLAs by single providers when adopting a multi-cloud setup. Furthermore, they can achieve more flexibility by combining offers from different providers, or by changing providers for certain services, but not others, if a more advantageous offer is made available [35]. Motivated to achieve the greatest level of flexibility and interoperability of components across clouds while, at the same time, taking advantage of the clouds' economies of scale, our study proposes a reference architecture based on the use of containers on a PaaS model. As containerised applications include all the environment configuration they need to run and can be deployed to any platform equipped with a container engine, they are fully portable across clouds, thus greatly reducing the risk of vendor lock-in and allowing implementers to shift away from the traditional SaaS model usually recommended for SMEs. This study therefore recommends that big data processing frameworks and other components of the proposed reference architecture be deployed as containerised services. From a technical perspective, a strong motivation of this study is to propose a reference architecture that has a discernible value for SMEs and is therefore reliable and fault-tolerant, since high availability and business continuity have been identified as important requirements for these companies [12,36]. Within a single commercial cloud, fault tolerance for containerised systems can be provided at three different levels: data centre, zone, and region. These are employed by cloud providers not only to improve fault tolerance, but also to facilitate managing the cloud infrastructure in a divide-and-conquer fashion. AWS has introduced the availability zones concept, which consists of isolated data centres hosted in the same region. Since no two availability zones share the same data centre, the resilience of the infrastructure is increased, even for single-region deployments [37]. An infrastructure that spans across multiple availability zones has been recommended by Netflix following the after-effects of AWS outages in the past [38]. However, zone failures involving multiple availability zones have occurred in the past [39]. Likewise, a DNS disruption reported in 2016 affected Azure customers in all regions, effectively rendering the entire cloud unavailable [40]. Our research aims to explore the most resilient design for a cloud reference architecture and has thus identified an infrastructure distributed not only across multiple regions, but also across multiple clouds as a pattern to aim for. The relationship between different levels of fault tolerance and the cost and volume of data transfer from a cloud consumer's perspective is also investigated in this study.

Finally, as the current study focuses on the utilisation of commercial clouds for the deployment of containerised distributed applications, another area of research which is yet to fully mature is that of resource consumption estimation for large-scale containerised systems. Currently, most cluster orchestration technologies allocate resources based on a user's initial request [28,41]. Since users tend to overestimate the resources required by their

applications, this approach leads to under-utilisation and, in the context of cloud computing, unnecessary costs [41]. Oversubscription is a possible solution, but a more systematic approach is called for in the literature [28]. Motivated by this gap in knowledge, the current study explores the relationship between the windowing function used in big data stream processing and the resource consumption of the cluster, with the aim of proposing a new approach to cluster size estimation for the domain of big data stream processing.

1.4. Novel Contribution

The novelty of this study lies in its comprehensive and innovative approach to big data stream processing for SMEs, organizations, and departments characterised by devolved management, tight budgetary constraints, and lack of in-house technical expertise (SMEODs). Our proposed reference architecture, MC-BDP, specifically addresses these challenges by:

1. Promoting an infrastructure hosted on commercial clouds leveraging the cost-effective pay-as-you-go model. This model is particularly beneficial for SMEODs as it allows for financial flexibility and scalability.
2. Shifting from a traditional SaaS model towards a standardised form of PaaS. This transition is crucial for SMEODs as it offers a more controlled and customisable computing environment suitable for specific big data needs.
3. Reducing the risk of vendor lock-in by recommending the use of portable, interoperable, and vendor-agnostic components. This strategy is vital for ensuring flexibility and independence in a multi-cloud environment.
4. Providing a domain-specific reference architecture that offers guidance and simplifies the implementation process. This aspect is particularly valuable for SMEODs that may lack the technical expertise or resources to navigate the complexities of big data systems.

This research further extends its novelty through a critical evaluation of big data architectures from leading companies such as Facebook, Twitter, and Netflix. This analysis aims to extract practical insights and non-functional requirements applicable to real-world big data systems. Additionally, the study introduces an innovative formula for adjusting CPU and data transfer requirements in a distributed computing cluster for big data streaming. This formula, based on the windowing function used for data processing, is a novel contribution that aids in optimizing resource utilization and cost-efficiency in big data stream processing.

This study's unique contribution, in essence, is the development, implementation, and evaluation of the MC-BDP architecture, coupled with insightful analyses and practical tools for enhancing big data stream processing in SMEODs, making it a pioneering study in the field.

The remainder of this paper is organised as follows; Section 2 reviews the related literature, situating the current work in the wider context of research in the fields of big data, cloud computing, and virtualisation, and discussing related work; Section 3 describes the MC-BDP reference architecture; Section 4 discusses the results of MC-BDP's evaluation; and Section 5 considers the impact of our research within its field and presents recommendations for future work.

2. Related Literature Review

Research in big data has flourished in recent years as the volume, velocity, and variety of data generated by systems and devices connected to the internet escalates at an unprecedented rate. This is apparent in the field of virtualisation, following the popularisation of Docker containers and the advent of sophisticated orchestration technologies enabling complex distributed architectures to be managed as a single cluster. Additionally, the popularisation of cloud computing and its associated pay-as-you-go model triggered a technological revolution by lowering the entry barrier for small and medium companies to commission complex and sophisticated systems. At the intersection of such rich and

exciting research fields, the current research project emerged. Reference architectures can be understood as an abstraction over concrete architectures aimed at a specific domain, or for a specific purpose, or both, based on which concrete architectures are derived. It is generally understood that every concrete system has an architecture [42]. The relationship between the system and the architecture can thus be abstracted as a one-to-one type, whereas a reference architecture holds a one-to-many relationship with specific implementations and concrete architectures. The objective of this study in proposing a reference architecture for big data stream processing in a multi-cloud environment is to provide a purely abstract template based on which specific architectures can be derived. MC-BDP is both domain-specific and purpose-specific, and although contributions were found that addressed one or even both aspects in the literature, an exact match was not encountered. This section takes a closer look at recent work directly related to the products of this research, summarised in Table 1, and demonstrates the significance of this study's contributions to the wider scientific community.

Table 1. Summary of related work and comparison with MC-BDP reference architecture.

Contribution	Domain	Primary Quantitative Evaluation	Data Processing Focus	Infrastructure	Virtualisation Level	Primary Qualitative Evaluation
[8]	Scientific Workflows	Y	Batch	Cloud	Hypervisor	N
[43]	Generic	N	Batch/Stream	Cloud/Bare-Metal	Hypervisor	N
[44]	Large Corporations	N	Batch	Cloud/Bare-Metal	N/A *	N
[9]	Healthcare	N	Batch/Stream	N/A *	N/A *	N
[45]	Generic	Y	Batch/Stream	Cloud	Hypervisor	N
[46]	Generic	N	Batch	N/A *	N/A *	Y †
[10]	National Security	Y	Batch/Stream	Bare-Metal	N/A *	N
[47]	Large Corporations	N	Batch/Stream	Cloud/Bare-Metal	Any	N
[48]	IoT	Y	Stream	Cloud/Bare-Metal	N/A *	N
[49]	Generic	N	N/A *	Cloud	Container	N
[50]	Edge Computing	Y	Stream	Bare-Metal	Container	N
[51]	Security	N	N/A *	Cloud	Any	N
[52]	Generic	Y	Stream	Cloud	Any	N
MC-BDP	SMEOD	Y	Stream	Cloud/Bare-Metal	Container	Y

* N/A means not applicable or not addressed by the author(s). † Qualitative data was analysed quantitatively.

The overview presented in Table 1 depicts the landscape of existing research in the domain of big data processing, revealing significant variations in the focus areas, infrastructure used, and virtualization levels across different studies. A critical observation derived from this table is the distinct nature of MC-BDP in its approach to big data stream processing for SMEODs, particularly in its use of both cloud and bare-metal infrastructures coupled with container-based virtualisation. This contrasts with most existing works, which predominantly focus on either cloud or bare-metal environments, and often employ hypervisor-based virtualisation. Additionally, while some studies address specific domains such as IoT, healthcare, or national security, MC-BDP is unique in its focus on SMEODs, a sector which faces unique challenges such as budget constraints and lack of in-house technical expertise. The qualitative evaluation in MC-BDP further sets it apart, as it provides an empirical assessment of its practical applicability, something often lacking in other studies. This gap in the literature, along with the unique combination of domain-specificity, purpose-specificity, and a balance of cloud and bare-metal approaches in MC-BDP, underscores the novelty and significance of this research in addressing the needs of SMEODs in the realm of big data stream processing.

2.1. Reference Architectures

A reference architecture for big data workflow management systems (BDWFMSs) was proposed by [8] for processing data-intensive scientific workflows in the cloud. Although

developed for processing big data using a distributed cloud-based infrastructure, BDWFMSs differs fundamentally from MC-BDP. Firstly, they were developed for the domain of scientific workflows, whereas MC-BDP is targeted at SMEODs entering the realm of big data stream analytics. Consequently, BDWFMSs do not specifically promote a multi-cloud environment, nor are they motivated by the desire to mitigate the risk of vendor lock-in. Virtualisation is provided at virtual machine level for BDWFMSs while MC-BDP uses container-level virtualisation. Thus, BDWFMSs are more significantly affected by the lack of standardisation in virtual machine models and specifications offered by each provider than MC-BDP. When it comes to the experimental evaluation of BDWFMSs, their design shares some similarities with MC-BDP's, such as the introduction of a computationally intensive calculation into some of the processing functions to observe how the system performs at higher loads. Both the BDWFMS and MC-BDP used real historical data for their experimental setups: the former looked for patterns in driving data for fifty New York drivers over the course of an hour for its first case study and analysed astronomical images for its second case study [8], while the latter calculated the energy efficiency of a data centre based on energy consumption records collected over the course of a year as part of its case study. An important difference between the experimental setups utilised to evaluate both reference architectures was that MC-BDP used a simulator to emit the data in real-time, whereas the BDWFMS analysed static data in its case studies. This difference can be explained by MC-BDP's focus on big data stream processing, whereas the BDWFMS's main concern is the processing of batch data.

The big data reference architecture was developed by the National Institute of Standards and Technology (NIST) [43]. NBDRA is composed of five different roles (system orchestrator, data provider, big data application provider, big data framework provider, and data consumer) and two fabrics (security and management), each implemented independently based on functional requirements. One significant difference between NBDRA and MC-BDP is that the former has a wider scope, applying to big data in general, while the latter applies specifically to the domain of big data streams for the SMEOD market. As a result, the components or roles and fabrics identified by NBDRA are broader and more abstract. As an example, the big data application provider role defined by NBDRA is responsible for processing the data according to domain-specific business logic. In MC-BDP's proposal, a different emphasis was given to this role, represented as the processing code uploaded to the big data framework. One of the main concerns of this study was to investigate the overhead of running technologically agnostic code capable of processing both batch and stream data. Given that development time is expensive, particularly in the context of SMEODs at which MC-BDP is targeted, it was one of the objectives of this study to measure the impact of technology agnosticism on performance. Finally, the data provider and data consumer roles defined by NBDRA were not differentiated in MC-BDP's proposal. Since MC-BDP was designed for stream processing, it used the conceptual model of a directed acyclic graphics, which is common in stream architectures [53,54]. Thus, the output of a process becomes the input of another, and each node in the process is capable of being both a provider and a consumer of data. The biggest difference between NBDRA and MC-BDP is perhaps that MC-BDP focuses on commercial clouds to provide the infrastructure required for big data processing, whereas NBDRA is neutral in this respect. This difference is explained by MC-BDP's focus on implementers whose presence in the big data arena is facilitated, to a great extent, by cloud computing.

The reference architecture proposed by [44] takes an approach similar to our architecture, starting with a thorough review of architectures and technologies for big data, followed by the proposal of a reference architecture and subsequent evaluation. However, their proposal focused on software only, whereas MC-BDP includes both hardware and software. In particular, MC-BDP offers an in-depth look at the advantages of leveraging container-based virtualisation technology to promote vendor lock-in mitigation and facilitate ingress into the big data market through cloud computing. Their proposed reference architecture looked mainly at frameworks for processing batch data. NoSQL databases

and the Hadoop ecosystem are described as relevant big data technologies. However, stream-specific components of the Hadoop ecosystem such as Flink, Spark, and Storm are not listed their research. MC-BDP, on the other hand, focuses on stream processing. Finally, the evaluation of Maier's reference architecture was performed by analysing the real-world big data implementations of Facebook, LinkedIn, and Oracle and retrofitting them to the reference architecture proposed [44]. While this study did perform a similar investigation of real-world big data implementations [31], its purpose was not to evaluate the MC-BDP proposal, but to gather requirements for it. The evaluation of MC-BDP was performed through a concrete case-study, with both quantitative and qualitative data collected and thoroughly analysed as part of the rigorous scientific process described in Section 4.

The generic architecture proposed by [9] has a number of elements in common with MC-BDP, such as its reliance on existing open-source big data frameworks and its focus on stream data processing for providing real-time intelligence. A fundamental difference is that, even though its architecture is focused on stream processing and real-time analytics, it contains a batch layer, implemented as a Hadoop cluster. This makes it an instance of the Lambda architecture, as opposed to MC-BDP, which is purely concerned with big data stream processing. Additionally, MC-BDP provides a domain-specific case study evaluation using a mixed-methods approach to provide not only the quantitative measurements needed to rigorously assess the contribution from a positivist standpoint, but also a highly valued qualitative appraisal of its impact and potential applications within the target domain.

A reference architecture for big data in the cloud focused on cost and the pay-as-you-go model is offered by [45]. This study presents three implementation possibilities for its reference architecture using stream processing technology offered by major commercial cloud providers, namely, AWS, Google Cloud, and Azure. While some similarities can be drawn between this proposal and MC-BDP, such as a focus on big data stream processing on infrastructure commissioned from commercial cloud providers, there are some fundamental differences, such as the preoccupation with vendor lock-in mitigation. Ref. [45]'s implementation proposals are neither portable nor interoperable with components on different clouds. They state that there was a requirement for each service proposed to be compatible with other services within the same provider (e.g., Amazon Kinesis Streams to provide data to Kinesis applications and Google Cloud Pup/Sub for Google Cloud Dataflow). There was, however, no requirement for the services selected to be compatible with services from other clouds, thus reducing the risk of vendor lock-in. Since vendor lock-in mitigation is one of the motivations of this study, the use of managed services is avoided.

The reference architecture for big data proposed by [10] is similar to MC-BDP in that it was designed for a specific domain. While MC-BDP focuses on the domain of big data streams for implementers looking to take advantage of the economies of scale brought about by cloud computing, their study's contribution was developed for the domain of national security. Technology agnosticism is a main concern of the reference architecture proposed by the study [10], one which it shares with the current study. The evaluation of the reference architecture was performed using a simple prototype implementation to analyse a data flow of Twitter sentiment data using stream processing and merge it with a news dataflow obtained through the processing of batch data to produce intelligence. Differently from MC-BDP's prototype evaluation, [10] was based on a typical Lambda architecture implementation where stream and batch data were processed separately before the results were merged. There was no controlled variation in the velocity of data ingress as performed for the MC-BDP evaluation, since the authors' objective was to demonstrate the feasibility of the reference architecture, not measure its performance. Case-specific implementations of such reference architecture within the domain of national security was called for as future work to enable a more thorough evaluation of the proposed reference architecture, similar to what was suggested by participants of the MC-BDP case study following its prototype evaluation within the domain of smart buildings.

A security reference architecture (SRA) for big data based on the UML notation was proposed by [51]. Their contribution builds upon other industry-standard reference architectures, particularly the NIST Big Data Interoperability Framework [43], by adding a security dimension defined using UML diagrams. Although their proposition differs significantly from MC-BDP, it is included in this review because they highlight the importance of incorporating security aspects into the early design stages of big data implementation [51]. This is the approach supported by this study and integrated into MC-BDP's security layer presented in Section 4.

Ref. [47] derived their reference architecture from studying publications related to four real-world big data implementations: Facebook, Twitter, Netflix, and LinkedIn. This is a similar approach to the study conducted early in this research where the Facebook, Twitter, and Netflix architectures for big data were analysed to gather non-functional requirements to inform the design of the MC-BDP reference architecture [31]. The use of an inductive process to arrive at a generalisation common to the observed cases and applicable to future cases is a known and widely used empirical methodology with its roots in [55]'s philosophy of science. Within this review, a similar inductive process to derive a reference architecture from known big data implementations from real-world companies was carried out by [56] and [44]. One criticism which can be made of this approach is that the systems were not evaluated in situ, nor were the source code and primary data available to the researchers. Indeed, it would have been advantageous to conduct a case study with the companies and to access their data, systems, and participants directly. However, as these are very large global companies, a case study such as the one proposed was not feasible. The evaluation of [47]'s reference architecture for big data systems was performed by retrofitting it to the real-world implementations at Facebook, Twitter, Netflix, and LinkedIn. This approach provides insufficient validation for the proposed model since it was derived from these very same architectures by induction, so the generalisation should logically retrofit the particular observations. The need for further validation through a real-world use case was acknowledged by the authors as a limitation and is addressed by the current research.

2.2. Architectures, Technology Stacks, and Concrete Implementations

Ref. [48] proposed an architecture for big data streams which shares some common aspects with MC-BDP, such as a focus on stream processing, reliance on open-source technologies, and evaluation using real-world data emitted by a purpose-built simulator. Among the principal differences is their aim to produce an architecture focused on the IoT domain, while MC-BDP is a reference architecture aimed at the SMEOD market. Their evaluation methodology for the proposed architecture is comparable to that used for MC-BDP's empirical evaluation. A simulator was used to emit smart parking streaming data in real time at different velocities, and the acquisition and processing times were independently measured to understand the overhead introduced by their proposed architecture. The velocities used varied between one and one hundred messages per second, while in MC-BDP's experiments they varied between two and two thousand messages per second, making the latter a more comprehensive test of the prototype under a heavy load. The focus of the experiments and the metrics observed were fundamentally different. They were interested in the speed of data processing while our research monitored performance metrics such as CPU, memory, and network utilisation to understand the overhead introduced by different features of the proposed reference architecture such as its multi-cloud proposal, as well as scalability, fault tolerance, technology agnosticism, the use of windowing to process streaming data, and container co-location. Ref. [49] proposed a specific technology stack for the utilisation of commercial cloud computing resources with minimal risk of vendor lock-in. Although not aimed at big data, their proposal shares some common aspects with MC-BDP, such as the use of container-level virtualisation, an orchestrator, and promotion of multi-cloud deployment as mitigation against the risk of vendor lock-in. In order to verify the viability of their proposed technology stack, a prototype was built whereby a simple browser game was deployed to three commercial providers as a containerised service. Nev-

ertheless, plans on how the prototype was going to be evaluated, as well as a full account of the research's methodology, were not published. The architecture proposed by [50] for the domain of edge cloud computing is an example of a Lambda architecture implementation based on Hadoop for batch processing and Spark for stream processing. Although aimed at very different domains, their research is relevant to the current study as it corroborates the empirical methodology followed in MC-BDP's evaluation. They evaluated the performance of their containerised architecture based on Docker and swarms by measuring CPU and memory utilisation at the container and node levels; disregarding disk I/O, since it is not relevant to stream processing; and monitoring the total processing time. The differences, however, are more significant. Ref. [50]'s architecture was developed for the domain of edge computing, where computation is performed by smaller edge devices before being transferred to the cloud. MC-BDP, on the other hand, was developed for SMEODs and aims to transfer as much of the computation as possible to the cloud to take advantage of its economies of scale. Although [50]'s choice of metrics and experimental setup were similar to MC-BDP's and provided validation for the decisions taken earlier in this study, their experimental work differed considerably in terms of the infrastructure, container allocation, container co-location, velocities, and windowing approach. MC-BDP's evaluation was more extensive, making it an important contribution to the field.

A recent contribution by [52] proposed a real-time scheduling algorithm for distributed big data stream processing in a multi-region cloud infrastructure. Using task duplication, the algorithm created by the authors schedules parallel tasks across different configured regions within the same cloud, aiming to maximise resource utilisation while minimising both costs and the completion time. The purpose of their work is rather different from that of the current study: while the former's main preoccupation was with achieving an efficient method of running parallel co-dependent workflows by scheduling (and sometimes duplicating) their component tasks, our research delegates the scheduling implementation to the orchestrator and focuses, instead, on the underlying architecture. Nevertheless, both studies are concerned with big data stream processing in a heterogeneous cloud environment where price variation is particularly significant to implementers.

3. MC-BDP Reference Architecture

3.1. Methods

A gap was initially identified in the literature, reflecting a shortage of systematic academic studies where container technology was applied to the domains of big data processing and cloud computing [21]. Additionally, reference architectures with a focus on cloud consumers based on models other than SaaS are under-studied. To address this gap, our research aims to leverage the latest advances in containers and container orchestration technology for a new PaaS-based multi-cloud reference architecture targeted at big data stream processing.

As part of the requirements engineering phase, we conducted a critical review of published work by three major big data corporations: Facebook, Twitter, and Netflix [31,57]. Ten non-functional requirements were identified and discussed in the context of these companies' architectures: batch data, stream data, late and out-of-order data, processing guarantees, integration and extensibility, distribution and scalability, cloud support and elasticity, fault tolerance, flow control, and flexibility and technology agnosticism. Based on these requirements and industry's best practices gathered from real-world implementations, a new MC-BDP reference architecture for big data stream processing in a multi-cloud environment was designed and developed. Subsequently, this architecture was implemented as a prototype in an energy efficiency case study involving the Estates and Sustainability departments at Leeds Beckett University. This case study used real data obtained by a previous data centre energy efficiency study [58] to calculate its PUE (power usage effectiveness). Since the mechanisms in place to emit consumption data and calculate the PUE have not evolved since the original study took place, the data collected remained relevant. The case study targeted by our research has been selected because it has characteristics which

are appropriate for the testing of our prototype [59]. A needs analysis interview revealed that the Estates and Sustainability departments at Leeds Beckett University managed data which had inherent big data characteristics: volume (log files with operational data that spans several years), velocity (data from meters that is sampled and streamed in real-time), and variety (device log exports, for example, contain unstructured data, as the format is specific to each manufacturer). The real-time sampling rate used for the data centre energy efficiency study was, however, limited by the technology available at the time. For the purpose of this research, the data granularity has been enhanced to meet the higher volume and velocity requirements together with MC-BDP's inherent strategy for dealing with late data [60]. Due to operational and availability constraints, a simulation exercise was carried out for the evaluation. Instead of using the real EGX300 server located at one of Leeds Beckett University's data centres, a simulator was written to emulate the behaviour of this server. The simulator used real energy consumption data obtained from the previous power usage effectiveness study [58] and readings were transmitted at desired frequencies using an interpolation algorithm. While using a simulator carries lower risks and is time and resource-efficient, it has the disadvantage of not providing real-time insight into live data. The main purpose of this case study was to conduct a technical evaluation of the proposed reference architecture. Three out of the ten non-functional requirements were selected: scalability, fault tolerance, and technology agnosticism. This selection is supported by current research in the field, as scalability and fault tolerance have been identified as key issues in a systematic literature review of big data stream analytics [61]. Correspondingly, a major factor of concern, the vendor lock-in risk, was mitigated through technology agnosticism. The seven remaining functional requirements were not evaluated due to the time and scope limitations of this project.

3.2. Experimental Procedure

The experimental procedure common to all simulations is summarised in Figure 1. It starts with booting all participating machines, which are connected to the internet via SSH secured by public key authentication. The decision to use a public key instead of password authentication was informed by security considerations at the design stage of the prototype, as recommended by the Cloud Computing Adoption Framework (CCAF) proposed by [62]. Public key authentication is more secure than password authentication given that, in the event of the server being compromised, password authentication confirms a valid username/password combination to a potential attacker [63].

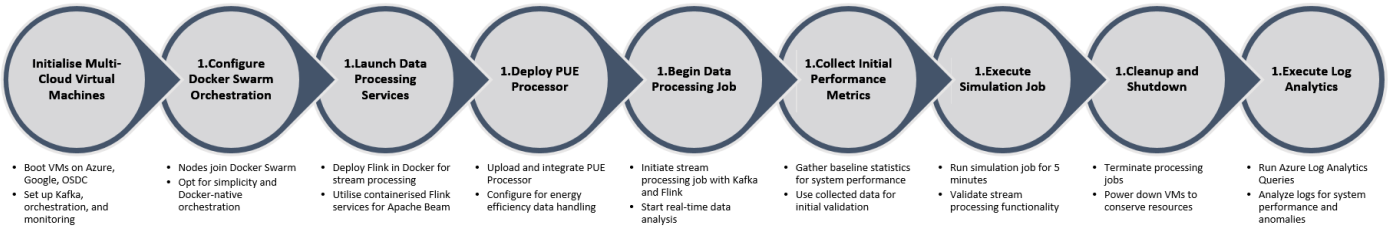


Figure 1. Experimental procedure for all simulations.

The next step is to run a command in each machine to register with the swarm manager as a worker. As part of this step, the machine's IP address is sent to the swarm manager via SSH. Since the Azure machines have static IP addresses, this step must be completed only once for all experiments. The Google Cloud machines are configured with ephemeral IP addresses, as static IP addresses are not available for purchase. For this reason, they must be registered with the swarm manager after each reboot.

The Flink container services were launched from the Azure swarm manager virtual machine via the Docker swarm interface, accessible through the SSH console. The Flink job manager service used the multi-cloud network created using the Weave plugin and was constrained to run on a dedicated machine. This constraint was introduced to isolate the

job manager service from the task manager service running on worker nodes, thus allowing for more granular monitoring and gathering of performance statistics. The image used to create the job manager service was obtained from the Docker Hub repository and was minimally modified. The Flink task manager service, distributed across different containers, also used the network created using the Weave plugin. A constraint was added to run this service on any machine but the one dedicated to run the job manager service. Once the task manager service was up and running, it was scaled to the desired level of parallelism using the Docker swarm interface.

Once the data processing job is running, initial statistics are gathered using the Weave Cloud service [64]. CPU and memory utilisation metrics are verified to check for discrepancies such as readings which are abnormally high or low. Any reading above 2% or below 0.1% for CPU utilisation was considered abnormal, and the setup process was restarted with a fresh reboot of the machine where the abnormality was observed. Similarly, any reading above 600 MB or below 200 MB for memory utilisation was considered abnormal and warranted a reboot of the machine and a restart of the setup process.

With the data processing job running and ready to receive data, the energy consumption simulator job is started using the graphical user interface. It runs for five minutes, transmitting data at the desired frequency. Once the energy consumption simulator finishes transmitting data, it displays the start and end transmission times. These are noted and used to set the times that the Azure Log Analytics queries are run 'from and to'. The setup is then cleared to prepare it for the next experiment: the PUE processor job is stopped via the Flink job manager interface, the Flink job manager and task manager services are stopped using the Docker swarm interface, the nodes are removed from the swarm, and the virtual machines are switched off. The final step in the process is to run the Azure Log Analytics queries using the Azure Portal. The 'from and to' times are set to the start and end transmission times. The queries executed are shown in Figure A1.

For the container co-location experiments, because the number of containers per node was no longer fixed at one, in order to compare the performance of clusters with different co-location distributions, the container measurements were aggregated, as per Figure A2, to generate node-level values. For metrics which constitute a sum (node network receive bytes and node network send bytes), the aggregation was calculated as the sum of the container-level measurements. For metrics which constitute an average (average node CPU utilisation and average node memory utilisation), the aggregation was calculated as the sum of the average utilisation of each container running on a node. Finally, for metrics which constitute a maximum, the aggregation of measurements at the container level to derive metrics at node level (CPU utilisation maximum and memory utilisation maximum) is inherently more complex.

3.3. MC-BDP Reference Architecture for Big Data Stream Processing

MC-BDP is an evolution of the PaaS-BDP architectural pattern originally proposed by the authors. While PaaS-BDP introduced a framework-agnostic programming model for batch and stream processing and enabled different frameworks to share a pool of resources [65], MC-BDP focuses on stream processing and expands the previous model by explicitly prescribing a multi-tenant environment where nodes are deployed to multiple clouds [57]. Therefore, the rationale for proposing a multi-cloud model is to mitigate the risk of vendor lock-in, perceived as a major obstacle to the adoption of cloud computing.

3.4. MC-BDP Architectural Layers

The MC-BDP reference architecture is depicted in Figure 2. The model comprises six horizontal and three vertical layers. At the lowest level is the persistence layer, used in different ways and to varying degrees by components in the node, container, service, and messaging layers. The next layer represents nodes or machines which can be physical or virtual. It is followed by the containers layer, which provides an additional layer of virtualisation, isolation, and abstraction on top of each node. Networking is represented

next since, in a distributed system used for big data processing, all processing units must be capable of communicating in order to enable parallel data processing. Networked containers working as part of a single distributed system must be centrally managed and coordinated, which is why orchestration is shown as the next layer in the diagram. Finally, a services layer contains all the application services deployed in the infrastructure described, including those responsible for processing big data as a stream and those used for monitoring and analytics. Security, monitoring, and messaging are represented as vertical layers, as they permeate every other layer in the diagram.

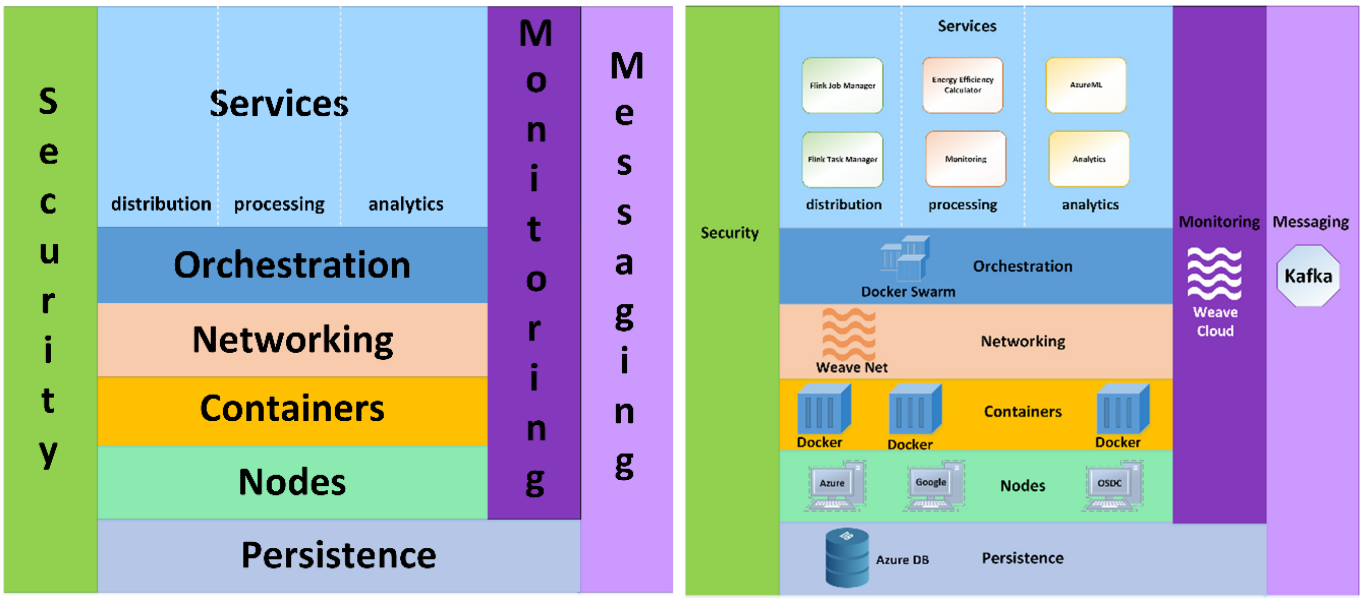


Figure 2. MC-BDP architectural layers (source: [57]).

3.4.1. The Persistence Layer

The persistence layer consists of file stores, disk space, and relational and non-relational databases. In the context of MC-BDP, it is used differently by components in other layers, namely, the node, container, service, and messaging layers. Depending on how monitoring is implemented, it can also use some form of persistence although, if a cloud service is used, its representation falls outside the scope of the diagram represented in Figure 2.

Typically, bare metal nodes hosted on-premises contain a hard disk drive (HDD) which uses magnetic recording technology to store information [66]. Likewise, virtual machines commissioned from commercial cloud providers include a configurable amount of disk space which can be increased or decreased as needed. Although this research focuses on the stream aspect of big data and therefore on processes which are not disk-intensive or storage-focused, it is aware that some degree of disk usage does happen at the node level. Likewise, containers running on physical or virtual machines are configured with an amount of disk space which can be fixed or flexible up to the maximum amount available on the host.

Services deployed to containers could be configured to use external storage such as databases or file stores hosted on bare metal or in the cloud. In MC-BDP's context of stream data processing, for instance, external storage is used primarily in checkpointing, a common approach used by large scale distributed architectures to provide fault tolerance [31]. The Flink framework, used in the prototype implementation of MC-BDP, recommends that states be stored in an external file system or database for extra resilience [67]. The cloud simplifies the provision of these storage resources, which can be commissioned as services and easily configured to provide an adequate level of resilience. MC-BDP's messaging layer also involves an element of persistence, albeit for a short (configurable) amount of

time. One of the reasons for this requirement is to allow consumers of the data to handle backpressure. Thus, should data processing slow down, the messaging layer can continue to receive new data while old data remains stored in the brokers waiting for its consumers to catch up.

Finally, for general persistence requirements such as the introduction of a sink to store data after processing, MC-BDP recommends a physical or cloud-based database or distributed file system such as Hadoop [68] or Aurora [69]. The advantages of using a cloud-based service are high scalability and fault tolerance, security out-of-the-box, and a potentially lower initial investment and maintenance cost, since the service is fully managed by the cloud provider [69–71].

3.4.2. The Node Layer

The node layer on PaaS-BDP, a predecessor of MC-BDP developed in the early stages of this research, was composed of physical or virtual machines to which containers were deployed [60]. MC-BDP was developed in an effort to reduce entry barriers for SMEODs to big data analytics. Taking into account that some of the biggest entry barriers perceived by such companies are the initial investment required and the risk of vendor lock-in [32], MC-BDP recommends an infrastructure based on commercial clouds and, where possible, it recommends that multiple clouds be utilised simultaneously. It is worth noting that MC-BDP is based on a minimal platform-as-a-service (PaaS) model where virtual machines with an operating system and a container runtime are provided. Any other dependencies needed for containerised services to run are specified in their respective container images and imported from public or private repositories at runtime, thus decoupling these services from the environment where they run and allowing for a very lightweight PaaS model [60].

One of the advantages of a lightweight PaaS model is that virtual machines have the potential to be cheaper, since no middleware or runtime technologies other than the container runtime need to be provided, maintained, or kept up to date. Additionally, there is a greater chance that different cloud providers' offerings will be equivalent, since fewer components are involved, thus enabling consumers to compare them in terms of price, SLAs, or other non-technical criteria. Since lack of standardisation amongst providers has been identified as one of the challenges in the field of cloud computing [72], research developments such as MC-BDP must be careful to account for it in their design. The proposal of a lightweight PaaS systems thus conforms with the state of current research in the field.

3.4.3. The Container Layer

The container layer consists of containers running services based on container images. A container image can be understood as the blueprint of a service. It contains information about the dependencies needed to build the service such as runtime and middleware, as well as the addresses from where these can be fetched. Additionally, images contain instructions on how to build and deploy the service as a container, including the setting of environment variables and the mapping of ports. Images can be stored in public or private repositories, from which they can be easily retrieved and shared among developers. Since images contain all the necessary configuration that a service needs to run, nothing needs to be installed on the platform other than the container runtime, thus allowing consumers to compare offerings by different cloud providers more easily and to migrate to a different provider if needed.

The Open Container Initiative (OCI) was launched by the Linux Foundation with the purpose of standardising container image formats and container runtime [73]. It has received widespread support in the community, including large donations of code by Docker [74], and is therefore proposed as MC-BDP's containerisation technology compliancy requirement. Due to Docker's status as a paradigm in the industry today, this technology is suggested as a good fit for default implementations, since it has a strong community of adopters as well as an abundance of documentation. It is worth highlighting,

however, that other container technologies such as Linux Containers (LXC), OpenVZ, and Linux-VServer exist, and could be used in place of Docker provided that they were supported by the orchestration technology selected. One clear advantage of using Docker is its popularity, as reported by Sysdig in their 2019 Container Usage Report [75]. According to this report, based on over two million real-world containers monitored by their technology, alternative runtimes such as Mesos or Linux Containers suffered a significant drop in usage and are nearly non-existent in recent commercial deployments. Docker, on the other hand, had an adoption of 79%, while a newcomer, CRI-O, amassed an impressive 4% of the market. The Sysdig report foresees an increase in the popularity of new container technologies as consequence of the standardisation promoted by the Open Container Initiative (OCI) [75]. This research is aware of this predicted trend and therefore adopts a flexible stance with regards to the container technology adopted.

3.4.4. The Networking Layer

The networking layer represents a network which allows containers deployed to different nodes at different locations to communicate seamlessly. As MC-BDP is a container-based reference architecture, it departs from the traditional approach of networking machines to that of networking containers. Since containers are standalone abstractions decoupled from the environment where they run, it is a sensible design decision to network at the container, rather than the node level.

There are different ways in which networking can be implemented at the container level, and MC-BDP is agnostic in this regard. Some examples are:

- Pipework, a legacy networking tool based on Linux's cgroups and namespaces [76].
- Flannel, a virtual network that provides a subnet to each host to use with container runtimes [77].
- Open vSwitch, which creates either an underlay or overlay network to connect containers running on multiple hosts [78].
- Weave Net, a Docker plugin that creates a virtual network to connect Docker containers across multiple hosts and enable their automatic discovery [79].

Although some container orchestrators claim to offer out-of-the-box networking [80], when building a prototype for MC-BDP using Docker swarms, it was observed that only containers deployed to machines in the same cloud could communicate with each other. In order to achieve inter-cloud networking, the Weave Net plugin had to be installed [81]. The Weave Net plugin is built around the concept of an overlay network, defined as an abstract network built on top of an existing physical network. The plugin deploys a router container and creates a network bridge on each host. The router uses TCP and UDP connections to transfer packets to other routers in the network, connected via the network bridges. Packets destined for containers located within the same host are handled by the kernel directly without going through the network [82]. In the context of the proposed reference architecture, this feature is particularly desirable since commercial cloud consumers are charged for the number of packets transferred over the network. The overlay network provided by the Weave Net plugin is thus a suitable technology for the implementation of container-to-container networking in the context of the reference architecture proposed. Nevertheless, MC-BDP remains agnostic in terms of the technology or approach selected for network containers hosted in different clouds.

3.4.5. The Orchestration Layer

MC-BDP's orchestration layer is responsible for launching, stopping, and managing containers in a cluster. It is therefore in charge of managing services deployed to containers, registering additional nodes from different clouds (or removing them), scaling the number of containers that a service runs on, controlling which containers run on which nodes, and monitoring the overall state of the cluster. A number of technologies exist for providing container cluster orchestration, the most popular of which is Kubernetes, developed by Google and later open-sourced to the wider community [83]. As Kubernetes is the most

established orchestrator technology at present, and Docker swarms are becoming increasingly popular due to their simplicity and ease of installation, most container networking technologies are compatible with them (Flannel, Open vSwitch, Weave Net).

Ref. [28] conducted a thorough survey of container orchestration systems whereby ten popular technologies were evaluated and classified according to three main categories (application model, scheduling, and cluster infrastructure and management), each with a number of sub-categories. MC-BDP is agnostic in terms of the orchestration technology used; provided that it is compatible with the container and container networking approach selected, orchestration could be implemented with any of the technologies reviewed in the aforementioned study.

3.4.6. The Service Layer

The service layer comprises the applications deployed to the container cluster, which range from smaller-scale deployments such as front-end user interfaces to larger-scale frameworks for big data processing distributed across hundreds of containers. MC-BDP relies on established big data frameworks to undertake the parallel processing of streaming data. They are responsible for parallelizing the computations, scheduling work between the processing units, and ensuring fault tolerance at the data processing level. As with previous components, MC-BDP is agnostic with regards to the big data framework used for data processing, provided that it can be deployed as a set of containerized services.

The service layer for a container-based reference architecture such as MC-BDP can be relatively complex, since advances in technology have made it possible to deploy monitoring and messaging services, as well as databases and file stores, as containerized services. This challenges the traditional stratified layering approach. Given the advantages of deploying parts of the architecture as containerised services, namely, greater portability and interoperability and a reduction in the risk of vendor lock-in, this research recommends that containers be the preferred deployment strategy of implementers. Thus, instead of a traditional layered approach containing a very busy services layer with components represented indistinguishably, this research proposes a service-optimised layer diagram where vertical swim lanes separate service components by function, as depicted in Figure 2.

3.4.7. The Security Layer

The security layer is orthogonal to all other layers, since it is implemented in multiple contexts. Encryption, for example, can be configured independently at the framework, orchestration, networking, messaging, and persistence levels. In the prototype implementation of MC-BDP, access to cloud-based virtual machines is achieved via SSH connections authenticated by public keys. Not only is public key authentication the recommended and most secure authentication method, but cryptographic algorithms are used to secure both the client and server ends of the connection, and all data transmitted is encrypted [84]. Due to the complexity inherent to the security aspect of large-scale cloud-based systems, [62] recommend addressing it through a systematic and comprehensive framework which is multi-layer and multi-purpose. This research used the Cloud Computing Adoption Framework (CCAF) proposed by these authors [62] in its prototype implementation of the MC-BDP reference architecture. However, it remains agnostic with regards to which security framework is ultimately implemented in specific cases.

3.4.8. The Monitoring Layer

The monitoring layer consists of services aimed at providing metrics related to the performance of specific components. Since diverse aspects of a system can be, and usually are, monitored, this layer is also orthogonal to the others, and would likewise benefit from a systematic approach such as a multi-layer and multi-purpose framework. Separate advances have been made to provide a monitoring framework for cloud systems [63] and for big data systems [85]. However, this study believes an integrated approach similar

to [62]'s CCAF is more suitable to the domain of big data in the cloud, since monitoring incorporates diverse aspects pertaining to different layers.

As an example, MC-BDP's prototype implementation utilised, at the application level, the Flink metrics functionality of the Flink framework [86]. At the orchestration layer, there were two monitoring perspectives employed: swarm monitoring and Prometheus monitoring. Swarm monitoring provided a terminal-based interface through which the state of the cluster could be monitored. Information, such as which nodes were part of the overlay network and which joined the cluster and which containers were running which services, was available by running simple command queries [87]. Prometheus monitoring, implemented as part of a Weave Cloud installation, provided similar information, but displayed it in a more user-friendly way through a graphical user interface provided by Weave Cloud [64].

At the networking layer, monitoring for the MC-BDP prototype was provided by the Weave command line interface, which showed the nodes connected to the overlay network and their status [81]. At the container level, information such as the status of each container, the services they were running, and CPU, memory, and network usage were monitored by three different technologies: Prometheus, Weave Cloud, and Azure Container Monitoring. Essentially, Weave Cloud provided an easy-to-use and aesthetically pleasing user interface for Prometheus, on which it is based. Prometheus queries were also run directly to obtain more fine-grained data. Finally, the Azure Container Monitoring service was used to obtain and analyse container performance metrics.

At the virtual machine level, Weave Cloud was used in combination with Azure Log Analytics. At the persistence level, Azure storage analytics was used for monitoring the Azure database commissioned as a service. Finally, at the messaging level, the Confluence Control Centre was used to monitor the health of the Kafka server [88].

3.4.9. The Messaging Layer

The messaging layer is used primarily to facilitate the transmission of data from one system to another. One example of such usage in the context of streaming architectures is use as a sink or output for real-time data from IOT devices and as a source or input for big data processing frameworks. Likewise, the messaging layer could be configured as a sink for the result of the data processing performed by the big data framework and as a source for subsequent processing by the same or a different framework.

As MC-BDP is a reference architecture for big data stream processing, it is recommended that the messaging technology be distributed and scalable in order to avoid bottlenecks. It should also be fault-tolerant so that processing can continue with minimal disruption in the event of a node, container, or even region failure. If savings derived from economies of scale are to be maximised, it is recommended that the messaging layer be implemented as a set of containerised services. Finally, in order to facilitate the handling of backpressure without loss of data by the big data framework, it is advisable that the messaging technology be capable of a buffering or persisting state for a short (configurable) amount of time. Technologies such as Google Cloud's Pub/Sub [89], Amazon Kinesis [90], or Apache Kafka [91] are suitable choices to fulfil these requirements. The latter was selected for the prototype implementation of MC-BDP since the others are cloud services and carry an inherent risk of vendor lock-in.

3.5. MC-BDP Prototype Implementation

This section summarises the case study based on which a prototype was created to evaluate the MC-BDP reference architecture. It also describes the results of MC-BDP's quantitative evaluation. A needs analysis exercise conducted at the start of our research revealed the existence of previous research conducted in 2012 with the Sustainability Centre at Leeds Beckett University to assess the energy efficiency of a data centre by calculating its power usage effectiveness (PUE) [58], and our current study leveraged more advanced technology with an aim of providing close-to-real-time data processing.

Figure 3 summarises the original PUE calculation process: energy consumption readings from IT equipment and from the rooms in the data centre building were sent to an EGX300 integrated gateway server. The data was then extracted into Excel reports as an ad hoc process. Notably, the readings were emitted every 30 min, and the log extraction into Excel took 43 min to run [58]. Moreover, the reporting was produced retrospectively on a pull basis. A simulator was developed to emulate the behaviour of the EGX300 integrated gateway server at the Woodhouse Data Centre at Leeds Beckett University.

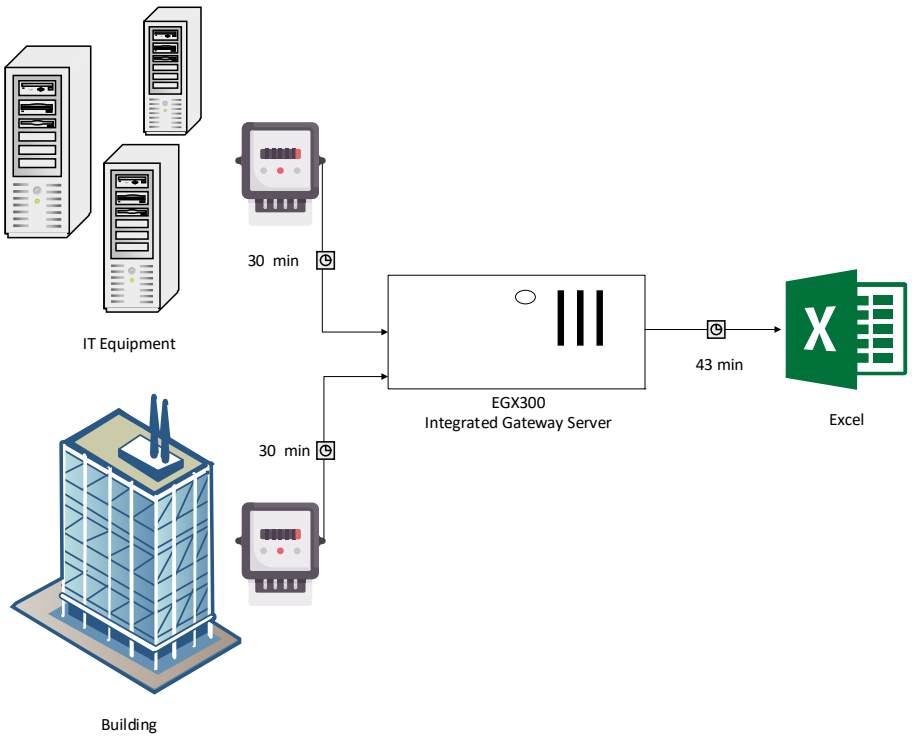


Figure 3. Original PUE calculation process at Woodhouse Data Centre.

The components involved in the prototype implementation of MC-BDP and their interfaces are shown in Figure 4. The energy consumption simulator provides a simple web-based interface through which a user can launch a simulation (e.g., transmit energy consumption records every 5 s for 10 min). It interfaces with the energy consumption producer component which receives historical readings from a database hosted in the Azure cloud, uses the data interpolator to generate realistic readings based on the historical readings, and emits the results to a Kafka server in the frequency requested. The PUE processor component is a distributed big data stream processing pipeline which runs on the Flink framework. It consumes the energy readings from Kafka, calculates the PUE, then exposes the results which, in the example depicted in the diagram, are posted to an Azure Event Hub. Different configurations were used in the development, including logging the results to a file to verify accuracy and posting them back to a different Kafka topic. Microsoft Power BI integrates easily with the Azure Event Hub and was used to create a simple dashboard to display the PUE calculations in real time. The formatted PUE results from the Azure Event Hub were also stored using Azure Blob Storage as proof of concept. The framework and orchestrator components shown on the bottom right-hand side of the diagram show how the processing job is launched and the parallelism is set through the job manager interface. The work is then distributed across several parallel task managers. The number of job managers and task managers available is controlled by the orchestrator, since these components run as containerised services, and the interface to scale these up or down is also exposed.

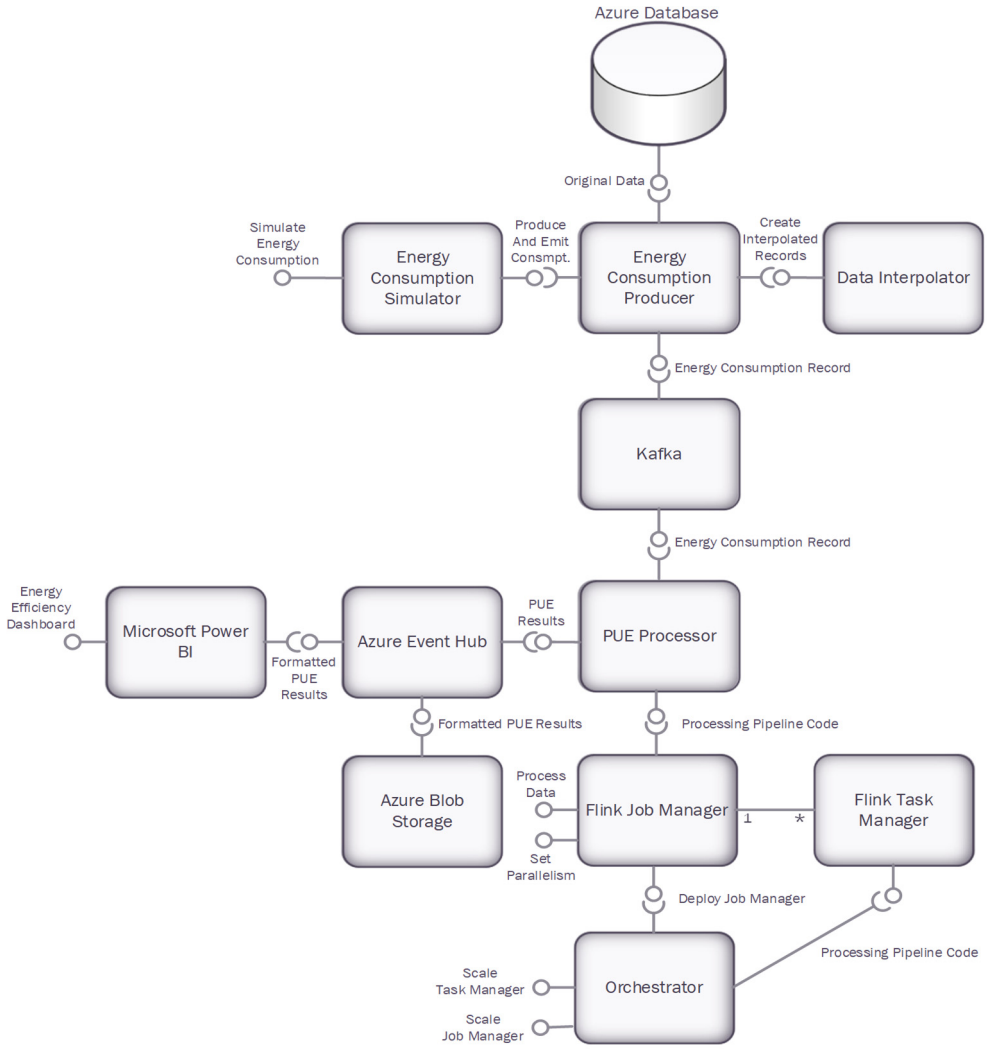


Figure 4. MC-BDP prototype implementation for the data centre energy efficiency case study.

The EGX300 integrated gateway server simulator is used to transmit data for the prototype implementation. The simulator comprises three modules: a data interpolator, a transmitter to transmit readings at desired frequencies, and a front-end user interface. The data interpolator component is a simple Maven project with a single class and a single static method. The static method takes an original consumption record for a given duration as a parameter and outputs simulated consumption records for a shorter duration contained in the initial duration. Instead of using a simple average to calculate how much energy was consumed for smaller durations contained within the original duration, the consumption values generated vary randomly, up to a maximum variation percentage passed in as a parameter. The energy consumption producer component was developed to gather energy consumption records from the original readings database, generate interpolated records using the data interpolator, and emit generated readings to the Kafka server in the desired time interval. Finally, the energy consumption simulator is a very simple out-of-the-box MVC application designed to gather runtime values from the user and pass them to the energy consumption producer. It captures user input for how many records to retrieve from the database, the maximum number of records to query per request, the duration that each original record corresponds to, the desired duration, the desired variation, and the credentials to access the database. The full implementation code, unit tests, and commit history for all components are available in public repositories [92–94]. The main technologies used in the prototype implementation of MC-BDP, the alternatives considered, and the rationale for selection are summarised in Table 2.

Table 2. Summary of technologies used in MC-BDP prototype implementation.

Technology	Purpose	Alternatives Considered	Rationale
Virtual Machines from Azure, Google, and OSDC	Run Kafka server, orchestration, data processing, and monitoring containers	Virtual machines from different providers, private cloud, bare metal, CaaS	The experiments were designed to evaluate a multi-cloud setup using commercial providers. Grants were received from the providers selected.
Linux Operating System	Run containers on a PaaS model	Windows, macOS, ChromeOS	Native integration with Docker, open-source, and free of costs.
Docker Containers	Run the big data framework's data processing containers	Linux Containers (LXC), OpenVZ, or Linux-VServer	Most popular container technology, open-source.
Docker Swarms	Container orchestration	Kubernetes, Mesos	Although Kubernetes is more feature-rich and more widely used, Docker swarms are simpler to configure and native to Docker.
Weave Net	Container networking across clouds	Pipework, Flannel, Open vSwitch	Simple to install, support for multi-cloud networking.
Kafka	Stream data source.	Pub/Sub, Kinesis, RabbitMQ	Open-source, persistent, scalable, suitable for high data throughput.
Flink	Stream processing	Spark, Dataflow	Flink was, at the time of implementation, the most capable open-source runner for Apache Beam [95]. It was also available as a set of containerised services for Docker.

In evaluating the MC-BDP reference architecture, it is evident that its assumptions and design are deeply rooted in the practicalities of current cloud-based data processing. The model's reliance on advanced technology for near real-time data processing is particularly relevant. This approach reflects the ongoing evolution in cloud and big data technologies, making it a realistic and forward-thinking choice. Furthermore, the use of a simulator to emulate the behavior of the EGX300 integrated gateway server illustrates a commitment to accuracy and feasibility in a controlled environment, which is crucial for credible and replicable research.

The shift towards real-time data processing and reporting, using sophisticated tools such as Kafka and Azure Event Hub, aligns perfectly with current industry trends. This not only enhances the model's applicability in real-world scenarios but also showcases an understanding of the dynamic nature of data processing in modern times. The integration of various components, such as a data interpolator and a PUE processor, in the Flink framework, mirrors the complexity often encountered in the industry, thus adding another layer of realism to the study.

The scalability aspect of the model, facilitated through cloud integration and the ability to adjust computational resources, is also a testament to its practicality and relevance. This aspect is particularly important given the fluctuating demands in big data processing. In summary, the MC-BDP model has its foundations firmly planted in the realities of contemporary big data challenges and cloud computing solutions.

4. MC-BDP Prototype Evaluation

The ever-increasing volume, velocity, and variety of big data means that architectures designed to process it need an infrastructure capable of expanding accordingly. Moreover, since big data processing is generally performed by distributed nodes, there must be provisions in place to ensure that the system continues to operate normally should one of the nodes fail. Both these qualities are most adequately provisioned by cloud computing,

which offers scalability and fault tolerance at potentially lower costs due to its economies of scale. However, an innovative approach to utilising commercial clouds from the perspective of an implementer wishing to mitigate the risk of vendor lock-in is needed, and MC-BDP was created to fulfil this demand. The experiments in this section measure container CPU and memory utilisation, as well as network utilisation in clusters of different sizes, to understand the impact of scaling up or down on resource consumption. Likewise, the same metrics are used to understand how different configurations of containers deployed to different clouds may have an impact on resource consumption in a scenario of fault tolerance, e.g., when one of the nodes becomes unavailable mid-processing.

Being able to write the data processing code once and run it in different platforms and frameworks has also been identified as a desirable requirement for companies who may not have the technical expertise in-house to maintain complex bespoke programs. Moreover, preventing duplication and promoting reuse is an accepted software development principle which implementers should strive to observe in order to promote efficiency and enhance the maintainability of the system in the long run [96]. Technology agnosticism is enabled, in this prototype implementation, by using the Beam framework. The experiments in this section aim to compare CPU, memory, and network utilisation in clusters running equivalent Beam and Flink processing code to determine whether one, which is technologically agnostic, has a significant advantage over the other, which is not.

Finally, an understanding of how different windowing rates and container co-location configurations affect resource utilisation was needed so implementers could make more informed business and architectural decisions, particularly in the cloud, where the financial impact of such decisions is more readily felt. CPU, memory, and network utilisation were measured in configurations using different windowing rates to determine the effect of the latter on resource consumption. Likewise, the same metrics were monitored in experiments using different container co-locations to understand whether co-locating containers in the same virtual machine (or the same cloud) had a measurable effect on resource consumption.

This section describes the results of the quantitative evaluation of the MC-BDP reference architecture for big data stream processing in a multi-cloud environment. The evaluated dimensions are the scalability, fault tolerance, technology agnosticism, windowing rate versus resource utilisation, and container co-location. An understanding of the impact of these dimensions on resource consumption is needed in order to validate the applicability of MC-BDP to a commercial cloud context where resources are charged on a pay-as-you-go model. A total of 110 experiments have been conducted for this research. The details of the experimental design have been tabulated in Tables A2–A4.

4.1. Experimental Results

The experimental results for the evaluation of the MC-BDP reference architecture are described in this section. Three categories of experiments were run, where each metric was monitored for different velocities, windowing rates, and cluster sizes. The velocity for all experiments was measured as two records per time unit and represents the speed of emission for: (1) total energy consumption of the data centre and (2) energy consumption by the IT equipment in the data centre. The windowing rate represents the period (how often a window of processing starts) divided by the duration (how long each window lasts). For example, the scalability, fault tolerance, and technology agnosticism experiments used a sliding window of five seconds, starting every second, for data processing. Non-parametric Mann–Whitney U tests (two independent samples) were employed for hypothesis testing due to the very small sample sizes [97]. The significance level chosen for all tests was $\alpha = 0.05$. The null hypothesis (Ho) was that the means of the two samples were not significantly different, while the alternative hypothesis, Ha, stated otherwise.

4.1.1. Average Container CPU Utilisation

Based on the summary results for average CPU utilisation presented in Table 3, the following conclusions were drawn: (i) average container CPU utilisation by velocity for

scalability—means of three or six workers (single and multi) are not significantly different from each other. The same conclusion was drawn for all the possible combinations of three workers (single and multiple) compared to six workers (single and multiple); (ii) average container CPU utilisation by velocity for fault tolerance—the same conclusions for all combinations in (i); (iii) average container CPU utilisation by velocity for technology agnosticism—the means for three or six workers (Beam, Flink) x (single, multiple) were found not to be significantly different from each other. The same conclusion was drawn for three workers [(Beam, Flink) and (single, multiple)] x six workers [(Beam, Flink) and (single, multiple)]. In terms of the number of workers (three, six, ten), the means were also not significantly different from each other for the average container CPU utilisation by windowing rate.

Table 3. Experimental results for the evaluation of MC-BDP reference architecture in terms of average CPU utilisation.

Average CPU Utilisation Hypothesis Testing Results, Average Container CPU Utilisation By Velocity	Scalability Metrics (for Workers)								
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)			Mann–Whitney U Test Results
	2 records/time	Single %		Multi %		Single %		Multi %	Three Workers and Three Workers (Single and Multi)
	1 min	3.52		3.44		2.51		3.61	*p*-value = 0.8857, accept Ho
	1 sec	37.94		40.21		18.22		23.17	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	42.25		36.39		25.46		26.24	*p*-value = 0.1905, accept Ho (S,S) *p*-value = 0.2857, accept Ho (S,M) *p*-value = 0.1905, accept Ho (M,S) *p*-value = 0.2857, accept Ho (M,M)
	10 ms	41.58		45.46		23.02		29.27	
	1 ms	NA		NA		36.26		33.02	Six Workers and Six Workers (Single and Multi)
	Average	31.32		31.38		17.30		20.57	*p*-value = 0.5476, accept Ho
	Fault Tolerance Metrics								
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)			Mann–Whitney U Test Results
	2 records/time	Single %		Multi %		Single %		Multi %	Three Workers and Three Workers (Single and Multi)
	1 min	10.08		3.58		4.97		5.02	*p*-value = 0.3429, accept Ho
	1 sec	7.11		5.92		6.25		8.30	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	11.09		14.83		8.44		7.17	*p*-value = 0.3429, accept Ho (S,S) *p*-value = 0.4857, accept Ho (S,M) *p*-value = 0.8857, accept Ho (M,S) *p*-value = 0.8857, accept Ho (M,M)
	10 ms	22.68		8.83		16.55		16.90	
	1 ms	NA		NA		NA		NA	Six Workers and Six Workers (Single and Multi)
	Average	12.74		8.29		9.05		9.35	*p*-value = 0.8857, accept Ho
	Technology Agnosticism Metrics (for Beam and Flink SDK)								
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)			Mann–Whitney U Test Results
	2 records/time	Beam		Flink		Beam		Flink	Beam / Flink
		S%	M%	S%	M%	S%	M%	S%	M%	Three Workers and Three Workers (Single and Multi)
	1 min	3.4	2.9	3.4	1.9	1.5	2.6	2.1	2.6	*p*-value = 0.6905, accept Ho / *p*-value = 0.4633, accept Ho
	1 sec	33.8	27.9	36.5	34.3	19.7	17.3	20.0	16.6	Three Workers and Six Workers ([S,S] x [M,M])

Table 3. *Cont.*

Average CPU Utilisation Hypothesis Testing Results, Average Container CPU Utilisation By Velocity	100 ms	44.7	36.5	38.5	36.1	23.2	23.2	18.7	20.2	*p*-value = 0.1425, accept Ho (S,S) *p*-value = 0.1508, accept Ho (S,M) *p*-value = 0.1425, accept Ho (M,S) *p*-value = 0.4206, accept Ho (M,M)	*p*-value = 0.09524, accept Ho (S,S) *p*-value = 0.09524, accept Ho (S,M) *p*-value = 0.1508, accept Ho (M,S) *p*-value =0.1508, accept Ho (M,M)
	10 ms	46.4	35.9	37.4	36.8	23.2	29.0	18.6	20.8		
	1 ms	64.8	66.7	35.2	36.5	35.6	37.2	17.7	18.4	Six Workers and Six Workers (Single and Multi)	
	Average	36.82	33.98	30.20	29.12	20.64	21.86	15.42	15.72	*p*-value = 0.8325, accept Ho	*p*-value = 0.8413, accept Ho

	Windowing Rate versus Resource Utilisation Metrics				
Average of Container CPU Utilisation by Windowing Rate (Period/Duration)	Windowing Rate (Period/Duration)	Three Workers (n = 3) (%)	Six Workers (n = 6) (%)	Ten Workers (n = 10) (%)	Mann–Whitney U Test Results
	0.1	80	46	29	Three Workers and Six Workers
	0.2	39	19	16	*p*-value = 0.4206, accept Ho
	1.0	16	9	6	Three Workers and Ten Workers
	1.5	10	6	6	*p*-value = 0.1719, accept Ho
	2.0	8	7	5	Six Workers and Ten Workers
	Average	30.60	10.25	12.40	*p*-value = 0.3976, accept Ho

	Container Co-Location Metrics		
Average node CPU utilisation by node Cluster	Cluster (n = 1) (%)	Cluster (n = 2) (%)	Cluster (n = 4) (%)
	13.39	15.96	14.47

Note: NA means data is not available.

Results (in the form of graphs and regression models) for the average CPU utilisation by velocity, windowing rate, and cluster are tabulated in Table A5. The velocity is measured in two records per time, where the time units have been converted to seconds (0.001, 0.01, 0.1, 1.0, 60.0). Due to the inconsistent time interval, a linear trend is not appropriate. Thus, a more appropriate model would either be an exponential (when the initial decrease is gradual, followed by a sharp decrease), logarithmic (when the initial decrease is sharp, followed by a gradual decrease), power, or polynomial regression model. Notably, the type of regression model was chosen based on the highest R-squared value. Graphical displays for all the regression models cannot be included in this paper due to space constraints.

4.1.2. Maximum Container CPU Utilisation

Based on the summary results for maximum CPU utilisation presented in Table 4, the following conclusions were drawn: (i) maximum container CPU utilisation by velocity for scalability—means of three or six workers (single and multiple) are not significantly different from each other. The same conclusion was drawn for all the possible combinations of three workers (single and multiple) compared to six workers (single and multiple); (ii) maximum container CPU utilisation by velocity for fault tolerance—the same conclusions for all combinations in (i); (iii) maximum container CPU utilisation by velocity for technology agnosticism—all means for three or six workers (Beam, Flink) x (single, multiple) were compared and found not to be significantly different from each other. The same conclusion was drawn for three workers [(Beam, Flink) x (single, multiple)] and six workers [(Beam, Flink) x (single, multiple)]. In terms of the number of workers (three, six, ten), the means were also not significantly different from each other for the maximum container CPU utilisation by windowing rate.

Table 4. Experimental results for the evaluation of MC-BDP reference architecture in terms of maximum CPU utilisation.

	Scalability Metrics									
Maximum Container CPU Utilisation By Velocity	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results
		Single %		Multi %		Single %		Multi %		Three Workers and Three Workers (Single and Multi)
	1 min	25		41		39		36		*p*-value = 0.7715, accept Ho
	1 sec	70		70		71		56		Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	86		67		84		60		*p*-value = 0.9048, accept Ho (S,S) *p*-value = 1.0952, accept Ho (S,M) *p*-value = 1.0952, accept Ho (M,S)
	10 ms	90		89		64		88		*p*-value = 0.9048, accept Ho (M,M)
	1 ms	NA		NA		88		92		Six Workers and Six Workers (Single and Multi)
	Average	67.75		66.75		64.50		60.00		*p*-value = 0.9166, accept Ho
	Fault Tolerance Metrics									
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results
		Single %		Multi %		Single %		Multi %		Three Workers and Three Workers (Single and Multi)
	1 min	24		7		17		15		*p*-value = 0.08143, accept Ho
	1 sec	17		13		13		28		Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	25		24		22		20		*p*-value = 0.3836, accept Ho (S,S) *p*-value = 0.8857, accept Ho (S,M) *p*-value = 0.3836, accept Ho (M,S)
	10 ms	58		14		47		54		*p*-value = 0.1143, accept Ho (M,M)
	1 ms	NA		NA		NA		NA		Six Workers and Six Workers (Single and Multi)
	Average	31.00		14.50		24.75		29.25		*p*-value = 0.6857, accept Ho
	Technology Agnosticism Metrics (for Beam and Flink SDK)									
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results
		Beam		Flink		Beam		Flink		Beam / Flink
		S%	M%	S%	M%	S%	M%	S%	M%	Three Workers and Three Workers (Single and Multi)
	1 min	27	27	27	20	27	38	19	18	*p*-value = 0.8335, accept Ho / *p*-value = 0.5476, accept Ho
	1 sec	70	53	44	41	59	44	29	25	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	79	79	62	47	70	49	30	27	*p*-value = 0.4606, accept Ho (S,S) *p*-value = 0.5476, accept Ho (S,M) *p*-value = 0.7533, accept Ho (M,S) *p*-value = 0.5476, accept Ho (M,M) / *p*-value = 0.09369, accept Ho (S,S) *p*-value = 0.04653, accept Ho (S,M) *p*-value = 0.09369, accept Ho (M,S) *p*-value =0.03615, accept Ho (M,M)
	10 ms	90	75	54	55	66	71	29	38	
	1 ms	92	95	50	46	92	91	31	28	Six Workers and Six Workers (Single and Multi)
	Average	71.60	65.80	47.40	41.80	62.80	58.60	27.60	27.20	*p*-value = 0.8413, accept Ho / *p*-value = 0.09524, accept Ho

Table 4. *Cont.*

Maximum Container CPU Utilisation by Windowing Rate (Period/Duration)	Windowing Rate versus Resource Utilisation Metrics				
	Windowing Rate (Period/Duration)	Three Workers (n = 3) (%)	Six Workers (n = 6) (%)	Ten Workers (n = 10) (%)	Mann–Whitney U Test Results
	0.1	93	92	86	Three Workers and Six Workers
	0.2	85	50	72	*p*-value = 1.0000, accept Ho
	1.0	41	40	44	Three Workers and Ten Workers
	1.5	32	45	42	*p*-value = 0.6905, accept Ho
	2.0	24	26	40	Six Workers and Ten Workers
	Average	55.00	50.60	56.80	*p*-value = 0.9166, accept Ho
Maximum node CPU utilisation by node Cluster	Container Co-Location Metrics				
		Cluster (n = 1) (%)	Cluster (n = 2) (%)	Cluster (n = 4) (%)	
		43.00	83.00	64.00	
Maximum node CPU utilisation by Windowing Rate and node Cluster	Windowing Rate (Period/Duration)	Cluster (n = 1) (%)	Cluster (n = 2) (%)	Cluster (n = 4) (%)	Mann–Whitney U Test Results
	0.1	43.00	83.00	63.50	Three Workers and Six Workers
	0.2	26.17	69.00	40.00	*p*-value = 0.2000, accept Ho
	1.0	11.58	30.00	17.00	Three Workers and Ten Workers
	1.5	11.93	13.00	20.00	*p*-value = 0.4857, accept Ho
	Average	23.17	48.75	35.13	Six Workers and Ten Workers
					p-value = 0.6857, accept Ho

Note: NA means data is not available.

Results (in the form of graphs and regression models) for the maximum CPU utilisation by velocity, windowing rate, and cluster are tabulated in Table A6.

4.1.3. Average Container Memory Utilisation

Based on the summary results for average memory utilisation presented in Table 5, the following conclusions were drawn: (i) average container memory utilisation by velocity for scalability—the means of three or six workers (single and multiple) are not significantly different from each other. The same conclusion was drawn for all the possible combinations of three workers (single and multiple) x six workers (single and multiple); (ii) average memory utilisation by velocity for fault tolerance—the same conclusions for all combinations in (i); (iii) average memory utilisation by velocity for technology agnosticism—all means for three or six workers (Beam, Blink) x (single, multiple) were found not to be significantly different from each other. The same conclusion was drawn for three workers

[(Beam, Flink) x (single, multiple)] and six workers [(Beam, Flink) x (single, multiple)]. In terms of number of workers (three, six, ten), the means were also not significantly different from each other for the average container memory utilisation by windowing rate.

Table 5. Experimental results for the evaluation of MC-BDP reference architecture in terms of average memory utilisation.

Average Container Memory Utilisation (MB) By Velocity	Scalability Metrics									
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results
		Single		Multi		Single		Multi		Three Workers and Three Workers (Single and Multi)
	1 min	824		865		790		812		*p*-value = 0.2000, accept Ho
	1 sec	955		976		933		1015		Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	957		995		941		992		*p*-value = 0.4217, accept Ho (S,S) *p*-value = 0.1905, accept Ho (S,M) *p*-value = 0.1905, accept Ho (M,S) *p*-value = 0.5556, accept Ho (M,M)
	10 ms	968		1002		942		998		
	1 ms	NA		NA		964		1058		Six Workers and Six Workers (Single and Multi)
	Average	926.00		959.50		901.50		954.25		*p*-value = 0.09524, accept Ho
	Fault Tolerance Metrics									
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results
		Single		Multi		Single		Multi		Three Workers and Three Workers (Single and Multi)
	1 min	837		758		882		684		*p*-value = 0.6857, accept Ho
	1 sec	839		842		717		758		Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	833		829		879		776		*p*-value = 0.2000, accept Ho (S,S) *p*-value = 0.05714, accept Ho (S,M) *p*-value = 0.3429, accept Ho (M,S) *p*-value =0.0294, accept Ho (M,M)
	10 ms	699		849		897		904		
	1 ms	NA		NA		NA		NA		Six Workers and Six Workers (Single and Multi)
	Average	802.00		819.50		843.75		780.50		*p*-value = 0.5614, accept Ho
	Technology Agnosticism Metrics (Azure)									
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results
		Beam		Flink		Beam		Flink		Beam / Flink
		S	M	S	M	S	M	S	M	Three Workers and Three Workers (Single and Multi)
	1 min	869	861	914	911	783	764	880	876	*p*-value = 1.0000, accept Ho / *p*-value = 1.0000, accept Ho
	1 sec	965	971	894	927	941	949	929	903	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	957	978	926	909	931	913	926	912	*p*-value = 0.09524, accept Ho (S,S) / *p*-value = 0.8335, accept Ho (S,S) *p*-value = 0.09524, accept Ho (S,M) / *p*-value = 0.4206, accept Ho (S,M) *p*-value = 0.09524, accept Ho (M,S) / *p*-value = 0.9163, accept Ho (M,S) *p*-value = 0.1508, accept Ho (M,M) / *p*-value = 0.402, accept Ho (M,M)
	10 ms	959	955	886	910	948	899	914	891	
	1 ms	1000	987	924	911	946	956	909	916	Six Workers and Six Workers (Single and Multi)
	Average	950.0	950.4	908.8	913.6	909.8	896.2	911.6	899.6	*p*-value = 1.0000, accept Ho / *p*-value = 0.3095, accept Ho

Table 5. *Cont.*

	Windowing Rate versus Resource Utilisation Metrics				
	Windowing Rate (Period/Duration)	Three Workers (n = 3) (MB)	Six Workers (n = 6) (MB)	Ten Workers (n = 10) (MB)	Mann–Whitney U Test Results
Average Container Memory Utilisation by Windowing Rate (Period/Duration)	0.1	1008	986	980	Three Workers and Six Workers
	0.2	974	1002	988	*p*-value = 0.5476, accept Ho
	1.0	947	962	903	Three Workers and Ten Workers
	1.5	968	943	892	*p*-value = 0.4206, accept Ho
	2.0	972	910	815	Six Workers and Ten Workers
	Average	973.8	960.6	915.6	*p*-value = 0.3095, accept Ho
	Container Co-Location Metrics				
Average memory utilisation by node Cluster		Cluster (n = 1) (MB)	Cluster (n = 2) (MB)	Cluster (n = 4) (MB)	
		39301	39727	41799	

Note: NA means data is not available.

Results (in the form of graphs and regression models) for the average memory utilisation by velocity, windowing rate, and cluster are tabulated in Table A7.

4.1.4. Maximum Container Memory Utilisation

Based on the summary results for maximum memory utilisation presented in Table 6, the following conclusions were drawn: (i) maximum container memory utilisation by velocity for scalability—means of three or six workers (single and multiple) were not significantly different from each other. The same conclusion was drawn for all the possible combinations of three workers (single and multiple) x six workers (single and multiple); (ii) maximum container memory utilisation by velocity for fault tolerance—the same conclusions for all combinations in (i); (iii) maximum container memory utilisation by velocity for technology agnosticism—all means for three or six workers (Beam, Flink) x (single, multiple) are not significantly different from each other. The same conclusion was drawn for three workers [(Beam, Flink) x (single, multiple)], and six workers [(Beam, Flink) x (single, multiple)], except for three workers (Beam, multiple) and six workers (Beam, multiple). Maximum memory utilisation was therefore slightly higher in multi-cloud clusters than it was in single-cloud clusters in the Beam SDK setups The reason for this discrepancy has not yet been established, and it may be related to the use of the Beam SDK and not to the infrastructure being single or multiple, since the same was not observed when using the Flink SDK. It is believed that repeated runs of the experiment over time would provide more relevant data to better understand the relationship between the cloud provider selected, the SDK selected, and the maximum memory utilisation metrics gathered. The means of different numbers of workers (three, six, ten) are not significantly different from each other for the maximum container memory utilisation by windowing rate.

Table 6. Experimental results for the evaluation of MC-BDP reference architecture in terms of maximum memory utilisation.

Maximum Container Memory Utilisation (MB) By Velocity	Scalability Metrics									
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results
		Single		Multi		Single		Multi		Three Workers and Three Workers (Single and Multi)
	1 min	947		987		946		934		*p*-value = 0.02857, accept Ho
	1 sec	959		983		939		1015		Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	969		1004		947		1025		*p*-value = 0.4606, accept Ho (S,S) *p*-value = 0.1905, accept Ho (S,M) *p*-value = 0.06349, accept Ho (M,S) *p*-value = 0.2957, accept Ho (M,M)
	10 ms	977		1022		967		1037		
	1 ms	NA		NA		990		1083		Six Workers and Six Workers (Single and Multi)
	Average	963.00		999.00		949.75		1002.75		*p*-value = 0.1508, accept Ho
	Fault Tolerance Metrics									
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results
		Single		Multi		Single		Multi		Three Workers and Three Workers (Single and Multi)
	1 min	906		831		764		815		*p*-value = 0.1143, accept Ho
	1 sec	897		879		762		847		Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	928		911		764		872		*p*-value = 0.0294, accept Ho (S,S) *p*-value = 0.3429, accept Ho (S,M) *p*-value = 0.0294, accept Ho (M,S) *p*-value = 0.6857, accept Ho (M,M)
	10 ms	1003		876		825		1021		
	1 ms	NA		NA		NA		NA		Six Workers and Six Workers (Single and Multi)
	Average	933.50		874.25		778.75		888.75		*p*-value = 0.05907, accept Ho
	Technology Agnosticism Metrics (Azure)									
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results
		Beam		Flink		Beam		Flink		Beam: *p*-value = 0.5476, accept Ho Flink: *p*-value = 0.7533, accept Ho
		S	M	S	M	S	M	S	M	Three Workers and Three Workers (Single and Multi)
	1 min	974	1012	934	923	961	953	935	909	
	1 sec	1006	1014	909	956	993	968	966	914	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	995	1001	941	939	973	969	961	935	Beam: *p*-value = 0.2222, accept Ho (S,S) *p*-value = 0.05556, accept Ho (S,M) *p*-value = 0.05556, accept Ho (M,S) *p*-value = 0.01587, reject Ho (M,M) Flink: *p*-value = 0.3457, accept Ho (S,S) *p*-value = 0.7533, accept Ho (S,M) *p*-value = 0.3457, accept Ho (M,S) *p*-value = 0.7533, accept Ho (M,M)
	10 ms	992	982	900	934	1000	973	930	946	
	1 ms	1079	1064	962	935	991	999	941	944	Six Workers and Six Workers (Single and Multi)
	Average	1009	1014	929	941	983	972	946	929	Beam: *p*-value = 0.3457, accept Ho Flink: *p*-value = 0.3457, accept Ho

Table 6. *Cont.*

Maximum Container Memory Utilisation by Windowing Rate (Period/Duration)	Windowing Rate versus Resource Utilisation Metrics				
	Windowing Rate (Period/Duration)	Three Workers (n = 3) (MB)	Six Workers (n = 6) (MB)	Ten Workers (n = 10) (MB)	Mann–Whitney U Test Results
	0.1	10,176	1011	991	Three Workers and Six Workers
	0.2	994	1003	1011	*p*-value = 0.1412, accept Ho
	1.0	999	1010	1005	Three Workers and Ten Workers
	1.5	986	1003	983	*p*-value = 0.5476, accept Ho
	2.0	998	1011	952	Six Workers and Ten Workers
	Average	2830	1007	988	*p*-value = 0.2031, accept Ho
Maximum node Memory utilisation by Windowing Rate and node Cluster	Windowing Rate (Period/Duration)	Cluster (n = 1) (MB)	Cluster (n = 2) (MB)	Cluster (n = 4) (MB)	Mann–Whitney U Test Results
	0.1	1050	2161	4474	Three Workers and Six Workers
	0.2	1085	2186	4517	*p*-value = 0.01193, reject Ho
	1.0	1043	2174	4475	Three Workers and Ten Workers
	1.5	1039	2171	4475	*p*-value = 0.01193, reject Ho
	2.0	1078	2161	4441	Six Workers and Ten Workers
	Average	1059	2170	4476	*p*-value = 0.01167, reject Ho
Maximum node Memory utilisation by node Cluster	Container Co-Location Metrics				
		Cluster (n = 1) (MB)	Cluster (n = 2) (MB)	Cluster (n = 4) (MB)	
		8091	8537	8824	

Note: NA means data is not available.

The means for the maximum node memory utilisation by windowing rate and node cluster sizes (one, two, four) were significantly different from each other, denoting a clear difference between the three co-locations observed. The nodes with two co-located containers had a maximum reading roughly twice as high as the nodes with no co-locater containers. Similarly, the nodes with four co-located containers had a maximum reading roughly twice as high as the nodes with two co-located containers. This was to be expected, however, since nodes with no co-located containers were in a cluster of eight, nodes with two co-located containers were in a cluster of four, and nodes with no co-located containers were in a cluster of two. In order to compare memory utilisation and co-location, it was necessary to take the entire cluster into consideration and aggregate the results. The cluster memory utilisation (maximum) formula, depicted in Figure A2, was utilised in order to obtain maximum memory utilisation results for the cluster. The last three rows of Table 6 summarise the maximum memory utilisation findings from the examination of the entire cluster.

Results (in the form of graphs and regression models) for the maximum memory utilisation by velocity, windowing rate, and cluster are tabulated in Table A8.

4.1.5. Container Network Utilisation (GB Sent by Workers)

Based on the summary results presented in Table 7 for total number of GB sent over the network, the following conclusions were drawn: (i) total number of GB sent over the network by velocity for scalability—means of three or six workers (single and multiple) are not significantly different from each other. The same conclusion was drawn for all the possible combinations of three workers (single and multiple) x six workers (single and multiple); (ii) total number of GB sent over the network by velocity for fault tolerance—the same conclusions for all combinations in (i); (iii) total number of GB sent over the network by velocity for technology agnosticism—all means for three or six workers (Beam, Flink) x (single, multiple) are not significantly different from each other. The same conclusion is drawn for three workers [(Beam, Flink) x (single, multiple)] and six workers [(Beam, Flink) x (single, multiple)]. The means of number of workers (three, six, ten) as well as the number of containers per node (one, two, four) were also not significantly different from each other for the total number of GB sent over the network by windowing rate.

Table 7. Network utilisation—total number of gigabytes sent over the network.

	Scalability Metrics					
Total Number of GB Sent over the Network by Velocity	Velocity	Three Workers (n = 3)		Six Workers (n = 6)		Mann–Whitney U Test Results
		Single	Multi	Single	Multi	Three Workers and Three Workers (Single and Multi)
	1 min	0.04	0.03	0.06	0.05	p-value = 0.7702, accept Ho
	1 sec	0.06	0.04	0.09	0.06	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	0.06	0.10	0.12	0.10	p-value = 0.3832, accept Ho (S,S) p-value = 0.3832, accept Ho (S,M) p-value = 0.5556, accept Ho (M,S) p-value = 0.3532, accept Ho (M,M)
	10 ms	0.48	0.47	0.21	0.49	
	1 ms	NA	NA	0.36	3.30	Six Workers and Six Workers (Single and Multi)
	Average	0.16	0.16	0.12	0.12	p-value = 0.9166, accept Ho
	Fault Tolerance Metrics					
	Velocity	Three Workers (n = 3)		Six Workers (n = 6)		Mann–Whitney U Test Results
		Single	Multi	Single	Multi	Three Workers and Three Workers (Single and Multi)
	1 min	0.84	0.34	4.50	0.80	p-value = 0.8845, accept Ho
	1 sec	0.76	1.44	2.62	0.95	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	0.84	0.52	1.48	1.10	p-value = 0.0294, accept Ho (S,S) p-value = 0.1102, accept Ho (S,M) p-value = 0.02857, accept Ho (M,S) p-value = 0.2000, accept Ho (M,M)
	10 ms	0.33	0.63	1.70	1.88	
	1 ms	NA	NA	NA	NA	Six Workers and Six Workers (Single and Multi)
	Average	0.69	0.73	2.58	1.18	p-value = 0.1143, accept Ho

Table 7. *Cont.*

Total Number of GB Sent over the Network by Velocity	Technology Agnosticism Metrics (for Beam and Flink SDK)										
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results	
		Beam		Flink		Beam		Flink		Beam	Flink
		S	M	S	M	S	M	S	M	Three Workers and Three Workers (Single and Multi)	
	1 min	0.14	0.02	0.04	0.01	0.07	0.05	0.05	0.03	*p*-value = 0.2948, accept Ho	*p*-value = 0.5219, accept Ho
	1 sec	0.11	0.02	0.04	0.01	0.08	0.06	0.05	0.03	Three Workers and Six Workers ([S,S] x [M,M])	
	100 ms	0.08	0.06	0.03	0.03	0.14	0.10	0.05	0.04	*p*-value = 1.0000, accept Ho (S,S) *p*-value = 0.4206, accept Ho(S,M) *p*-value = 0.2948, accept Ho(M,S) *p*-value = 0.7526, accept Ho(M,M)	*p*-value = 0.8668, accept Ho (S,S) *p*-value = 0.9131, accept Ho (S,M) *p*-value = 0.1599, accept Ho (M,S) *p*-value = 0.4564, accept Ho (M,M)
	10 ms	0.37	0.35	0.04	0.05	0.48	0.27	0.08	0.05		
	1 ms	3.52	3.25	0.17	0.17	6.26	3.02	0.28	0.24	Six Workers and Six Workers (Single and Multi)	
	Average	0.84	0.74	0.06	0.05	1.41	0.70	0.10	0.08	*p*-value = 0.5476, accept Ho	*p*-value = 0.1599, accept Ho

Total Number of GB Sent over the Network by Windowing Rate (Period/Duration)	Windowing Rate versus Resource Utilisation Metrics				
	Windowing Rate (Period/Duration)	Three Workers (n = 3)	Six Workers (n = 6)	Ten Workers (n = 10)	Mann–Whitney U Test Results
	0.1	0.49	0.63	1.02	Three Workers and Six Workers
	0.2	0.34	0.44	0.56	*p*-value = 0.6004, accept Ho
	1.0	0.13	0.14	0.23	Three Workers and Ten Workers
	1.5	0.09	0.11	0.19	*p*-value = 0.3095, accept Ho
	2.0	0.10	0.10	0.12	Six Workers and Ten Workers
	Average	0.23	0.28	0.42	*p*-value = 0.4206, accept Ho

Total Number of GB Sent over the Network by Number of Containers per Node (by Windowing Rate and Node Cluster)	Container Colocation Metrics				
	Windowing Rate (Period/Duration)	Cluster (n = 1) (GB)	Cluster (n = 2) (GB)	Cluster (n = 4) (GB)	Mann–Whitney U Test Results
	0.1	0.70	0.80	0.30	Three Workers and Six Workers
	0.2	0.30	0.40	0.30	*p*-value = 0.7526, accept Ho
	1.0	0.00	0.00	0.00	Three Workers and Ten Workers
	1.5	0.20	0.10	0.10	*p*-value = 0.3398, accept Ho
	2.0	0.42	0.10	0.11	Six Workers and Ten Workers
	Average	0.32	0.28	0.16	*p*-value = 0.9153, accept Ho

Note: NA means data is not available.

Results (in the form of graphs and regression models) for the number of gigabytes sent over the network by velocity, windowing rate, and cluster are tabulated in Table A9.

4.1.6. Container Network Utilisation (GB Received by Workers)

Based on the summary results presented in Table 8 for the total number of GB received over the network, the following conclusions were drawn: (i) total number of GB received over the network by velocity for scalability—means of three or six workers (single and multiple) are significantly different from each other for all combinations except for three workers (single, multiple). The same conclusion is drawn for almost all possible combinations of three workers (single and multiple) x six workers (single and multiple), except for three workers (single) x six workers (multiple); (ii) total number of GB received over the network by velocity for fault tolerance –there is no significant difference in the means for all combinations, except for six workers (single x multiple); (iii) total number of GB received over the network by velocity for technology agnosticism—all means for three or six workers (Beam) x (single, multiple) are not significantly different from each other. There is a significant difference for three workers (Flink) (single x multiple) and (Flink) [three workers (single) x six workers (single, multiple)], while the rest of the combinations show no significant difference. The means of different numbers of workers (three, six, ten) are significantly different from each other, while the number of containers per node (one, two, four) are not significantly different from each other for the total number of GB received over the network by windowing rate.

Table 8. Network utilisation (gigabytes received over the network)—graphs and regression models.

	Scalability Metrics					
Total Number of GB Received over the Network by Velocity	Velocity	Three Workers (n = 3)		Six Workers (n = 6)		Mann–Whitney U Test Results
		Single	Multi	Single	Multi	Three Workers and Three Workers (Single and Multi)
	1 min	3.91	3.53	7.94	5.17	*p*-value = 0.4857, accept Ho
	1 sec	4.80	4.17	7.57	5.18	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	3.32	3.58	7.38	4.84	*p*-value = 0.01587, reject Ho (S,S) *p*-value = 0.02684, accept Ho (S,M) *p*-value = 0.01587, reject Ho (M,S)
	10 ms	4.84	3.36	5.72	5.73	*p*-value = 0.01587, reject Ho (M,M)
	1 ms	NA	NA	9.64	12.49	Six Workers and Six Workers (Single and Multi)
	Average	4.22	3.66	7.15	5.98	*p*-value = 0.2222, reject Ho
	Fault Tolerance Metrics					
	Velocity	Three Workers (n = 3)		Six Workers (n = 6)		Mann–Whitney U Test Results
		Single	Multi	Single	Multi	Three Workers and Three Workers (Single and Multi)
	1 min	1.91	2.02	5.52	5.58	*p*-value = 0.8857, accept Ho
	1 sec	1.66	1.94	6.34	4.71	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	2.95	1.90	4.64	4.75	*p*-value = 0.02857, accept Ho (S,S) *p*-value = 0.02857, accept Ho (S,M) *p*-value = 0.02857, accept Ho (M,S)
	10 ms	1.46	1.62	5.27	5.64	*p*-value = 0.02857, accept Ho (M,M)
	1 ms	NA	NA	NA	NA	Six Workers and Six Workers (Single and Multi)
	Average					*p*-value = 0.2222, reject Ho

Table 8. *Cont.*

	Technology Agnosticism Metrics (for Beam and Flink SDK)										
Total Number of GB Received over the Network by Velocity	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results	
		Beam		Flink		Beam		Flink		Beam	Flink
		S	M	S	M	S	M	S	M	Three Workers and Three Workers (Single and Multi)	
	1 min	4.84	2.28	1.83	1.03	8.75	7.10	4.19	3.45	*p*-value = 0.09269, accept Ho	*p*-value = 0.007937, reject Ho
	1 sec	4.84	2.28	2.25	1.09	9.16	7.23	4.73	3.27	Three Workers and Six Workers ([S,S] x [M,M])	
	100 ms	4.61	2.31	2.32	1.16	9.47	6.51	4.57	2.92	*p*-value = 0.05933, accept Ho (S,S) *p*-value = 0.09369, accept Ho (S,M) *p*-value = 0.03615, accept Ho (M,S) *p*-value = 0.5296, accept Ho (M,M)	*p*-value = 0.007937, reject Ho (S,S) *p*-value = 0.01587, reject Ho (S,M) *p*-value = 0.007937, accept Ho (M,S) *p*-value = 0.01587, accept Ho (M,M)
	10 ms	5.14	4.19	2.04	1.92	10.06	7.96	4.66	3.79		
	1 ms	10.03	9.26	3.14	3.07	20.79	16.21	7.76	5.57	Six Workers and Six Workers (Single and Multi)	
	Average	5.89	4.06	2.32	1.65	11.65	9.00	5.18	3.80	*p*-value = 0.09524, accept Ho	*p*-value = 0.09524, accept Ho

	Windowing Rate versus Resource Utilisation Metrics				
Total Number of GB Received over the Network by Windowing Rate (Period/Duration)	Windowing Rate (Period/Duration)	Three Workers (n = 3)	Six Workers (n = 6)	Ten Workers (n = 10)	Mann–Whitney U Test Results
	0.1	4.46	7.40	13.49	Three Workers and Six Workers
	0.2	4.45	7.81	13.70	*p*-value = 0.007937, reject Ho
	1.0	4.39	7.57	13.69	Three Workers and Ten Workers
	1.5	4.23	7.56	12.91	*p*-value = 0.007937, reject Ho
	2.0	4.12	7.42	12.87	Six Workers and Ten Workers
	Average	4.33	7.55	13.33	*p*-value = 0.007937, reject Ho

	Container Co-Location Metrics				
Total Number of GB Received over the Network by Number of Containers per Node (by Windowing Rate and Node Cluster)	Windowing Rate (Period/Duration)	Cluster (n = 1) (GB)	Cluster (n = 2) (GB)	Cluster (n = 4) (GB)	Mann–Whitney U Test Results
	0.1	9.88	9.96	8.64	Three Workers and Six Workers
	0.2	8.51	8.90	8.42	*p*-value = 0.4206, accept Ho
	1.0	9.11	8.97	7.68	Three Workers and Ten Workers
	1.5	9.10	8.70	9.21	*p*-value = 0.1508, accept Ho
	2.0	9.15	7.91	8.18	Six Workers and Ten Workers
	Average	9.15	8.89	8.43	*p*-value = 0.3095, accept Ho

Note: NA means data is not available.

Increased network utilisation as clusters increased in size was an expected effect, since more workers means the greater parallelisation of work, which optimises performance but increases container-to-container communication [53]. This is particularly relevant in the context of cloud computing, where data transfers are charged on a metered basis. As expected, co-location did not significantly affect network utilisation, since containers were configured to use the internet to communicate with each other. This means that, even if containers were co-located in the same host, they used the internet to communicate with each other. It is worth noting, however, that there are other orchestration technologies, such as Kubernetes, which allow a greater degree of resource sharing between containers deployed to the same node. Kubernetes uses the concept of a pod of group-related containers running on the same machine, so containers which are part of the same pod can access each other's ports and transfer data internally without leaving the host [98]. It would be interesting to implement an alternative prototype in the future using a different orchestrator such as Kubernetes in order to gain an understanding of other ways in which the co-location of containers into the same machine could translate into reduced cloud network utilisation.

Results (in the form of graphs and regression models) for the number of gigabytes received over the network by velocity, windowing rate, and cluster are tabulated in Table A10.

4.1.7. Container Network Utilisation (GB Sent and Received by the Job Manager)

Based on the summary results presented in Table 9 for total number of GB received and sent over the network by the job manager during the technology agnosticism experiments, the following conclusions were drawn: (i) total number of GB received by the job manager over the network by velocity for technology agnosticism—all means for three or six workers (Beam, Flink) x (single, multiple) are not significantly different from each other except for six workers (multiple) x six workers (multiple). However, means are significantly different for all possible combinations [Beam, Flink] [(three workers (single, multiple) x six workers (single, multiple)], except for [Beam] [(three workers (multiple) x six workers (multiple)]; (ii) total number of GB sent by the job manager over the network by velocity for technology agnosticism—all means for three or six workers (Beam, Flink) x (single, multiple) are not significantly different from each other. However, means are significantly different for all possible combinations [Beam, Flink] [(three workers (single, multiple) x six workers (single, multiple)]. This corroborates the premise that the greater the number of workers in the cluster, the greater the parallelisation of work, which leads to increased communication between containers across the network.

Results (in the form of graphs and regression models) for the number of gigabytes received and sent by the job manager during the technology agnosticism experiments by velocity are tabulated in Table A11.

Based on the summary results presented in Table 10 for the total number of GB received and sent by the job manager over the network during the windowing rate versus resource utilisation experiments, the following conclusions were drawn: (i) total number of GB received by the job manager over the network by velocity for windowing rate—all means for (three, six, ten) workers are significantly different from each other except for six workers x ten workers; (ii) total number of GB sent by the job manager over the network by velocity for windowing rate—all means for (three, six, ten) workers are significantly different from each other. This is consistent with the results observed in previous experiments and explained by the fact that parallelisation of work is higher in larger clusters, leading to increased communication between containers across the network. While this model is a good fit for distributed systems hosted entirely on-premises, it could result in additional costs when commercial clouds are used.

Table 9. Network utilisation for technology agnosticism– total number of gigabytes received and sent by the job manager.

	Technology Agnosticism Metrics (for Beam and Flink SDK)										
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results	
		Beam		Flink		Beam		Flink		Beam	Flink
		S	M	S	M	S	M	S	M	Three Workers and Three Workers (Single and Multi)	
Total Number of GB Received by the Job Manager over the Network	1 min	5.87	4.59	2.46	2.25	8.55	10.25	4.60	5.04	*p*-value = 0.1425, accept Ho	*p*-value = 1.0000, accept Ho
	1 sec	4.88	4.65	2.31	2.25	9.42	10.35	5.02	4.38	Three Workers and Six Workers ([S,S] x [M,M])	
	100 ms	5.14	4.74	2.47	2.51	10.27	11.18	4.99	4.98	*p*-value = 0.01193, reject Ho (S,S) *p*-value = 0.007937, reject Ho (S,M) *p*-value = 0.001193, reject Ho (M,S)	*p*-value = 0.007937, reject Ho (S,S) *p*-value = 0.01193, reject Ho (S,M) *p*-value = 0.01193, reject Ho (M,S)
	10 ms	5.17	4.71	2.30	2.53	10.27	10.33	5.52	4.98	*p*-value = 0.4020, accept Ho (M,M)	*p*-value = 0.01167, reject Ho (M,M)
	1 ms	6.98	7.31	3.80	3.38	15.78	15.66	7.68	7.76	Six Workers and Six Workers (Single and Multi)	
	Average	5.61	5.20	2.67	2.58	10.86	11.55	5.56	5.43	*p*-value = 0.01167, reject Ho	*p*-value = 0.6572, accept Ho
	Technology Agnosticism Metrics (for Beam and Flink SDK)										
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results	
		Beam		Flink		Beam		Flink		Beam	Flink
		S	M	S	M	S	M	S	M	Three Workers and Three Workers (Single and Multi)	
Total Number of GB Sent by the Job Manager over the Network	1 min	16.13	13.53	6.60	6.52	49.13	58.96	25.92	28.27	*p*-value = 0.1425, accept Ho	*p*-value = 1.0000, accept Ho
	1 sec	13.62	13.56	6.50	6.52	54.26	58.98	28.24	23.59	Three Workers and Six Workers ([S,S] x [M,M])	
	100 ms	14.83	13.59	7.12	7.13	59.12	63.85	28.29	28.23	*p*-value = 0.007937, reject Ho (S,S) *p*-value = 0.007937, reject Ho (S,M) *p*-value = 0.01193, reject Ho (M,S)	*p*-value = 0.007937, reject Ho (S,S) *p*-value = 0.007937, reject Ho (S,M) *p*-value = 0.01193, reject Ho (M,S)
	10 ms	14.84	13.59	6.54	7.09	59.11	58.94	30.70	28.24	*p*-value = 0.01193, reject Ho (M,M)	*p*-value = 0.01193, reject Ho (M,M)
	1 ms	19.77	21.02	10.72	9.52	88.56	88.42	42.57	42.43	Six Workers and Six Workers (Single and Multi)	
	Average	15.84	15.06	7.50	7.36	62.04	65.83	31.14	30.15	*p*-value = 0.8413, accept Ho	*p*-value = 0.4633, accept Ho

Table 10. Network utilisation for windowing rate versus resource utilisation—total number of gigabytes received and sent by the job manager over the network.

	Windowing Rate versus Resource Utilisation Metrics				
	Windowing Rate (Period/Duration)	Three Workers (n = 3)	Six Workers (n = 6)	Ten Workers (n = 10)	Mann–Whitney U Test Results
Total Number of GB Received by the Job Manager over the Network	0.1	5.21	9.76	8.29	Three Workers and Six Workers
	0.2	4.77	10.51	18.02	*p*-value = 0.01587, reject Ho
	1.0	2.42	4.89	18.17	Three Workers and Ten Workers
	1.5	4.79	9.73	7.01	*p*-value = 0.007937, reject Ho
	2.0	4.87	8.94	16.56	Six Workers and Ten Workers
	Average	4.41	8.77	13.61	*p*-value = 0.4206, accept Ho
	Windowing Rate versus Resource Utilisation Metrics				
	Windowing Rate (Period/Duration)	Three Workers (n = 3)	Six Workers (n = 6)	Ten Workers (n = 10))	Mann–Whitney U Test Results
Total Number of GB Sent by the Job Manager over the Network	0.1	14.90	54.93	76.20	Three Workers and Six Workers
	0.2	13.75	59.49	165.26	*p*-value = 0.007937, reject Ho
	1.0	6.91	27.45	165.13	Three Workers and Ten Workers
	1.5	13.76	54.94	63.68	*p*-value = 0.007937, reject Ho
	2.0	13.78	50.35	139.77	Six Workers and Ten Workers
	Average	12.62	49.43	122.01	*p*-value = 0.07937, reject Ho

Results (in the form of graphs and regression models) for the number of gigabytes received and sent by the job manager during the windowing rate versus resource utilisation experiments by windowing rate are tabulated in Table A12.

Based on the summary results presented in Table 11 for the total number of GB received and sent by the job manager over the network during the container co-location experiments, the following conclusions were drawn: (i) total number of GB received by the job manager over the network by number of containers per node—all means for (one, two, four) clusters are not significantly different from each other; (ii) total number of GB received by the job manager over the network by number of containers per node—all means for (one, two, four) workers are not significantly different from each other, except for (three workers x six workers). This is an important finding when considered alongside the network utilisation readings for workers. It establishes that, generally, when a cluster increases in size, the effect on manager-to-worker communication is minimal. Worker-to-worker communication, on the other hand, increases significantly since there is greater parallelisation.

Table 11. Network utilisation for container co-location—total number of gigabytes received and sent by the job manager over the network.

	Container Co-Location Metrics				
Total Number of GB Received by the Job Manager over the Network by Number of Containers per Node	Windowing Rate (Period/Duration)	Cluster (n = 1)	Cluster (n = 2)	Cluster (n = 4)	Mann–Whitney U Test Results
	0.1	1.67	1.36	1.64	Three Workers and Six Workers
	0.2	1.64	1.37	1.51	p-value = 0.02733, accept Ho
	1.0	1.53	1.50	1.36	Three Workers and Ten Workers
	1.5	1.65	1.36	1.64	p-value = 0.05701, accept Ho
	2.0	1.84	1.64	1.50	Six Workers and Ten Workers
	Average	12.62	49.43	122.01	p-value = 0.3337, accept Ho
	Container Co-Location Metrics				
Total Number of GB Sent by the Job Manager over the Network by Number of Containers per Node	Windowing Rate (Period/Duration)	Cluster (n = 1)	Cluster (n = 2)	Cluster (n = 4)	Mann–Whitney U Test Results
	0.1	12.20	10.17	12.21	Three Workers and Six Workers
	0.2	12.22	10.16	11.18	p-value = 0.01167, reject Ho
	1.0	11.20	11.18	10.17	Three Workers and Ten Workers
	1.5	12.20	10.17	12.19	p-value = 0.09269, accept Ho
	2.0	12.28	10.18	10.17	Six Workers and Ten Workers
	Average	1.67	1.45	1.53	p-value = 0.2343, accept Ho

Results (in the form of graphs and regression models) for the number of gigabytes received and sent by the job manager during the container co-location experiments by windowing rate are tabulated in Table A13.

4.1.8. Total Records Processed and Data Loss

Based on the summary results presented in Table 12 for the total number of records processed during the fault tolerance experiments, the following conclusions were drawn: (i) total records processed by fault tolerance—all means for (three, six) workers x (single, multiple) are not significantly different from each other; (ii) data loss by fault tolerance—all means for (three, six) workers x (single, multiple) are not significantly different from each other.

Table 12. Experimental results for records processed and data loss during the evaluation of MC-BDP reference architecture for fault tolerance.

	Fault Tolerance Metrics					
	Velocity	Three Workers (n = 3)		Six Workers (n = 6)		Mann–Whitney U Test Results
		Single	Multi	Single	Multi	Three Workers and Three Workers (Single and Multi)
Total Records Processed	1 min	50	50	50	50	*p*-value = 1.0000, accept Ho
	1 sec	2987	2998	2990	2986	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	29,478	30,030	29,902	29,876	*p*-value = 0.7715, accept Ho (S,S) *p*-value = 1.0000, accept Ho (S,M) *p*-value = 1.0000, accept Ho (M,S) *p*-value = 1.0000, accept Ho (M,M)
	10 ms	295,765	294,179	300,673	297,689	
	1 ms	NA	NA	NA	NA	Six Workers and Six Workers (Single and Multi)
	Average	82,070	81,814	83,403	82,650	*p*-value = 0.7715, accept Ho
	Fault Tolerance Metrics					
	Velocity	Three Workers (n = 3)		Six Workers (n = 6)		Mann–Whitney U Test Results
		Single %	Multi %	Single %	Multi %	Three Workers and Three Workers (Single and Multi)
Data Loss %	1 min	0.00	0.00	0.00	0.00	*p*-value = 0.6573, accept Ho
	1 sec	0.44	0.07	0.33	0.47	Three Workers and Six Workers ([S,S] x [M,M])
	100 ms	1.77	0.00	0.33	0.42	*p*-value = 0.1804, accept Ho (S,S) *p*-value = 0.5614, accept Ho (S,M) *p*-value = 0.877, accept Ho (M,S) *p*-value = 0.6573, accept Ho (M,M)
	10 ms	1.43	1.98	0.00	0.78	
	1 ms	NA	NA	NA	NA	Six Workers and Six Workers (Single and Multi)
	Average	0.91	0.51	0.17	0.42	*p*-value = 0.1804, accept Ho

Note: NA means data is not available.

4.1.9. Data Ingested

Based on the summary results presented in Table 13 for the total data ingested during the technology agnosticism experiments, the following conclusions were drawn: data ingestion by technology agnosticism—means are not significantly different for all combinations for Beam and Flink [(three workers, six workers) x (single, multiple), (three workers (single, multiple) x six workers (single, multiple)].

Table 13. Experimental results for data ingestion during the evaluation of MC-BDP reference architecture for technology agnosticism.

	Technology Agnosticism Metrics (for Beam and Flink SDK)									
	Velocity	Three Workers (n = 3)				Six Workers (n = 6)				Mann–Whitney U Test Results
		Beam		Flink		Beam		Flink		Three Workers and Three Workers (Single and Multi)
Data Ingestion (KB) by Velocity		S	M	S	M	S	M	S	M	*p*-value = 1.0000, accept Ho / *p*-value = 0.9166, accept Ho
	1 min	172	74	55	39	259	205	158	126	Three Workers and Six Workers ([S,S] x [M,M])

Table 13. *Cont.*

Data Ingestion (KB) by Velocity	1 sec	2373	2328	98	90	4824	4824	242	230	*p*-value = 0.6905, accept Ho (S,S) *p*-value = 0.6905, accept Ho (S,M) *p*-value = 0.8413, accept Ho (M,S) *p*-value = 0.8413, accept Ho (M,M)	*p*-value = 0.5476, accept Ho (S,S) *p*-value = 0.5476, accept Ho (S,M) *p*-value = 0.5476, accept Ho (M,S) *p*-value = 0.5476, accept Ho (M,M)
	100 ms	21,900	219,300	510	543	44,580	44,580	1086	1074		
	10 ms	215,700	215700	4680	4680	432,600	432,600	9480	9420		
	1 ms	2,154,000	2,154,000	46,500	34,500	4,308,000	4,308,000	93,000	93,000	Six Workers and Six Workers (Single and Multi)	
	Average	478,829	518,280	10,368	7970	958,052	958,041	20,793	20,770	*p*-value = 1.000, accept Ho	*p*-value = 0.583, accept Ho

Based on the summary results presented in Table 14 for the data ingested during the windowing rate versus resource utilisation and container co-location experiments, the following conclusions were drawn: (i) total KB ingested by windowing rate—means for all (three, six, ten) workers are not significantly different from each other; (ii) total number of records ingested by windowing rate—means for all (three, six, ten) workers are not significantly different from each other; (iii) total KB ingested by windowing rate and number of containers per node—means for all (one, two, four) clusters are not significantly different from each other; (iv) total number or records ingested by windowing rate and number of containers per node—means for all (one, two, four) clusters are not significantly different from each other.

Table 14. Experimental results for data ingestion during the evaluation of MC-BDP reference architecture for the windowing rate versus resource utilisation and container co-location.

	Windowing Rate versus Resource Utilisation Metrics				
	Windowing Rate (Period/Duration)	Three Workers (n = 3)	Six Workers (n = 6)	Ten Workers (n = 10)	Mann–Whitney U Test Results
Total KB Ingested by Windowing Rate (Period/Duration)	0.1	143,700	143,800	144,000	Three Workers and Six Workers
	0.2	71,900	71,990	72,280	*p*-value = 0.6905, accept Ho
	1.0	14,500	14,650	14,814	Three Workers and Ten Workers
	1.5	9690	9892	10,052	*p*-value = 0.6905, accept Ho
	2.0	7410	7515	7638	Six Workers and Ten Workers
	Average	49,440	49,569	49,756	*p*-value = 0.6905, accept Ho

Table 14. *Cont.*

	Windowing Rate versus Resource Utilisation Metrics				
Total Number of Records Ingested by Windowing Rate (Period/Duration)	Windowing Rate (Period/Duration)	Three Workers (n = 3)	Six Workers (n = 6)	Ten Workers (n = 10)	Mann–Whitney U Test Results
	0.1	600,000	600,000	600,000	Three Workers and Six Workers
	0.2	300,000	300,000	300,000	*p*-value = 0.9161, accept Ho
	1.0	60,000	60,000	60,000	Three Workers and Ten Workers
	1.5	39,945	40,106	39,920	*p*-value = 0.9161, accept Ho
	2.0	30,361	30,197	29,637	Six Workers and Ten Workers
	Average	206,061	206,060	205,911	*p*-value = 0.9161, accept Ho
	Container Co-Location Metrics				
Total KB ingested by Number of Containers per Node Windowing Rate (Period/Duration)	Windowing Rate (Period/Duration)	Cluster (n = 1)	Cluster (n = 2)	Cluster (n = 4)	Mann–Whitney U Test Results
	0.1	144,100	144,000	143,900	Three Workers and Six Workers
	0.2	72,150	72,090	72,140	*p*-value = 0.8413, accept Ho
	1.0	14,746	14,724	14,711	Three Workers and Ten Workers
	1.5	10,044	9935	9998	*p*-value = 0.8413, accept Ho
	2.0	7548	7569	7593	Six Workers and Ten Workers
	Average	49,717	49,663	49,668	*p*-value = 1.0000, accept Ho
	Container Co-Location Metrics				
Total Number of Records Ingested by Number of Containers per Node by Windowing Rate (Period/Duration)	Windowing Rate (Period/Duration)	Cluster (n = 1)	Cluster (n = 2)	Cluster (n = 4)	Mann–Whitney U Test Results
	0.1	600,000	600,000	600,000	Three Workers and Six Workers
	0.2	300,000	300,000	300,000	*p*-value = 0.9161, accept Ho
	1.0	60,000	60,000	60,000	Three Workers and Ten Workers
	1.5	39,827	39,869	40,073	*p*-value = 0.9161, accept Ho
	2.0	28,881	29,898	30,037	Six Workers and Ten Workers
	Average	205,741	205,953	206,022	*p*-value = 0.9161, accept Ho

5. Conclusions and Future Work

This study set out to investigate a unified vendor-agnostic solution to big data stream processing in a multi-cloud environment. As key beneficiaries of commercial cloud com-

puting, small and medium enterprises, organisations, or departments characterised by devolved administration, tight budgetary constraints, and a lack of specialised technical skills in-house (SMEODs) were selected as the target domain for MC-BDP, a new reference architecture for big data stream processing aimed at maximising the cloud's economies of scale while at the same reducing the risk of vendor lock-in.

The majority of developments targeted at facilitating the adoption of big data analytics through cloud computing have focused on reducing the complexity of implementing and managing large distributed systems. Consequently, solutions are predominantly SaaS-based and associated with another major cause of apprehension: the risk of vendor lock-in. A number of authoritative studies have been conducted on possible ways of mitigating the risk of vendor lock-in [18,32,49]. However, no solutions were found which minimised the risk of vendor lock-in while simultaneously allowing implementers to maximise the cloud's economies of scale. Although the combination of containers, cloud computing, and big data streams is not new, the field lacked a domain-specific approach developed from a cloud consumer's perspective to enable implementers to ingress into big data stream analytics using a cloud model less restrictive than SaaS. As a response to the breadth, extensiveness, and complexity of big data research, recent developments have tended to focus on specific domains such as the biomedical sciences [9], the IoT [48], edge computing [50], national security [10], and scientific simulations [8]. The current study is one such effort, developed to fill the gap for a systematic approach to big data stream processing targeted at a domain whose main preoccupation is with minimising the risk of vendor lock-in while at the same time maximising the cloud's economies of scale.

Motivated by a desire to facilitate the adoption of big data stream analytics, this study has proposed a new systematic and unified approach to big data streams. The MC-BDP reference architecture was created to:

1. Leverage the cloud's economies of scale by promoting an infrastructure hosted on commercial clouds;
2. Move away from the traditional SaaS approach towards a standardised form of PaaS;
3. Mitigate the risk of vendor lock-in by prescribing the use of portable, interoperable, and vendor-agnostic components deployed to multiple clouds; and
4. Alleviate concerns around complexity and skill shortages in the domain by providing a domain-specific reference architecture to guide implementers.

This study challenges the accepted belief according to which the most appropriate cloud consumption model for SMEs is SaaS [11,12,99–101]. While acknowledging the concerns around technical complexity and skill shortages which have led previous authors to recommend the SaaS cloud model to SMEs, particularly within the realm of big data, this study places equal importance on the perceived risk of vendor lock-in explored extensively in recent research [18,32,102–104]. The MC-BDP reference architecture, designed specifically for the SMEOD domain, provides a good level of scaffolding to enable implementers to navigate the complexities of big data technology and cloud computing without having to employ a dedicated and highly specialised technical team.

The simplification and systematisation provided by the MC-BDP reference architecture enabled this study to break with tradition [11,12,99–101] and recommend a cloud consumption model more desirable than SaaS from a vendor lock-in perspective. This is a significant contribution to the field of big data streams which, up until now, lacked a systematic and thoroughly researched approach to big data based on a consumption model other than SaaS. It is not expected that research on specialised SaaS provisions for SMEs will cease given that concerns around the risk of vendor lock-in are not universally or evenly shared amongst companies—some could find that this is a risk worth taking for the benefit of simplicity. Therefore, works such as [45]'s reference architecture for big data designed from a cost perspective or [56]'s reference architecture based on real-world big data implementations, discussed in Section 2, remain valid and indeed useful within the field. What this study has demonstrated, however, is that rather than searching for a universal architecture for big data to suit all SMEODs, research should be directed towards

finding approaches which are flexible and abstract enough to account for the immense variety in requirements and priorities that characterise the domain.

The 110 experiments performed under controlled conditions demonstrated that MC-BDP's prototype was satisfactorily scalable and fault-tolerant across clouds, with no considerable difference observed between single and multi-cloud equivalent setups. Container co-location was also found not to significantly affect performance. In terms of technology agnosticism, the overhead introduced by the Beam framework in terms of CPU, memory, and data transfer was considerable. The conclusion reached was that there is indeed a performance cost to technology agnosticism as implemented in MC-BDP's prototype due to its use of a super-framework to provide portability of the code across various stream and batch technologies. Nevertheless, this may be a price some implementers would be willing to pay, so studies such as this where the overhead introduced by a super-framework was rigorously identified and measured are of utmost value to implementers who are able to take more informed decisions based on case-specific requirements for portability and interoperability of the data processing code. The relationship between the windowing function used for stream data processing and resource utilisation was also more thoroughly understood following MC-BDP's experimental evaluation. A simple formula representing the effects of the windowing function selected on CPU and network utilisation was proposed, It is important to note that, although exiting work can be found in the literature which links the windowing function to resource utilisation [105], the approach taken is one which assumes the infrastructure to be static and adjusts the windowing function to keep resource utilisation within a desirable range. Taking advantage of one of the cloud's most advantageous characteristics, its elasticity, the current study recommends a model where the windowing rate remains constant while the infrastructure commissioned expands and shrinks to keep performance metrics within a desirable range.

The quantitative findings of this research demonstrated that the proposed reference architecture is adequately scalable across different clouds. Given the cloud's pay-as-you-go model whereby consumers are only charged for the resources they utilise, being able to scale up or down is particularly important for SMEODs since budgetary constraints demand that resources be allocated sensibly and waste minimised. Fault tolerance across clouds was also demonstrated empirically, with no significant overhead observed in multi-cloud setups when compared to single-cloud ones. The relevance of this requirement to the domain of SMEs has been identified in the literature as the need for high availability of their production systems [12]. Since the aim of this study was to investigate big data stream processing from a vendor-agnostic perspective, being able to use any stream (or even batch) framework to run the processing code was identified as a desirable quality and integrated into the reference architecture proposed. The significance of these results to researchers and implementers is in being able to take more informed architectural decisions, particularly in the cloud, where the estimated overhead can be translated into projected costs.

Of similar relevance are the findings of a direct relationship between the windowing rate and resource utilisation in a distributed big data stream system. Since it was demonstrated that the CPU and network utilisation can be predicted using a simple formula based on the windowing rate and constants derived from empirical observation, implementers now have a more accurate way of projecting the cluster size and the cost of data transfers. It is believed that the aforementioned relationship has a wider application within the field of big data stream processing using commercial clouds. The container co-location findings confirm that co-location does not significantly affect the performance of CPU-intensive big data stream systems. In the context of cloud computing, this corroboration enables implementers to experiment with a greater number of deployment configurations, from multiple virtual machines with a few containers running in them to a smaller number of more powerful machines hosting a greater number of containers. Factors such as the desired level of fault tolerance (container, node, region, or provider) or projected data transfer costs may play an important part in the decision-making, particularly considering that container-to-container data transfer costs can be reduced or eliminated through co-location.

The cost-centric approach to task distribution offered by [106], aimed at minimising data transfer costs across clouds, is an important development in this direction. Performance monitoring at the container and node levels, combined with a weighted list of desired SLAs and the use of a flexible learning algorithm to make predictions, is this study's answer to the same question and is elaborated as a proposal for future work.

5.1. Recommendations for Future Research

The success of this research project has left some interesting avenues open for exploration. The case study conducted with the Estates and Sustainability departments at Leeds Beckett University concluded with a strong desire by the participants to extend the investigation by considering an important aspect of big data which had previously been left outside of scope: its variety. The complexity and multidimensionality of building data has, in fact, been identified in recent research as one of the biggest challenges to big data technology adoption within the sector [107]. A new research project aimed at investigating the appropriateness of the MC-BDP reference architecture for use-cases where the variety of the data is paramount was created as the case study concluded.

Tasks for future works investigating aspects of MC-BDP include:

- Investigating how MC-BDP handles batch processing. Comparing [108]'s approach of using a stream engine to process both batches and streams with an alternative approach based on the Lambda architecture.
- Following a trend observed in [109]'s research on container co-location, observing how MC-BDP performs with jobs of different characteristics. Memory-intensive, network-intensive, and disk I/O-intensive jobs could be created and compared to the CPU-intensive job utilised in the experiments.
- Integrating recent research on scheduling algorithms such as RTSATD, which uses task duplication to optimise the performance of big data stream processing across different cloud regions [52], or [110]'s algorithm developed to minimise data transfers over the network into MC-BDP's orchestration layer.
- Adding a cost perspective to MC-BDP's evaluation by integrating fine-grained billing information obtained from cloud providers. This is in line with [45]'s and [106]'s research. It is believed that understanding non-functional requirements from a cost as well as performance perspective would be advantageous to budget-constrained organisations.
- Extending MC-BDP's evaluation to include other domain-specific industry case studies.
- Conducting additional case studies with other types of SMEODs to validate the proposition that MC-BDP is beneficial to them.
- Strengthening the statistical significance of the quantitative results obtained by widening the scale of the experiments, using more than two commercial cloud providers and a greater number of virtual machines and containers and extending the experiments in duration and volume of incoming data.

As other relevant research is added to the fields of big data, cloud computing, and virtualisation, new hypotheses and unexplored questions shall become apparent, impelling further investigation and solidifying this study's position as a mature contribution to the field.

5.2. Limitations of the Study

As is the case with all empirical investigations, there were limitations to the experiments conducted to evaluate the MC-BDP reference architecture, in light of which the results obtained must be understood.

5.2.1. Internal Validity

The main limitations to the internal validity of MC-BDP's experimental evaluation were the measuring frequency, the checkpointing frequency, and the monitoring metrics utilised. Usage statistics were gathered from the host and sent to the Azure Monitoring

service. These measurements were subsequently queried using the Azure Log Analytics service. A collection sample interval of ten seconds was used. The rationale for keeping this configuration was that it constituted the greatest granularity available to describe a time interval of five minutes (thirty readings). At the time the experiments were designed and conducted, the minimal collection sample interval allowed for Linux performance counters was ten seconds [111]. It is believed that, should this technical limitation be lifted in the future, it will be worth experimenting with higher sample frequencies, provided that they do not significantly interfere with the performance of the host machine. Alternatively, further studies could be designed using other monitoring solutions such as Prometheus [112] to better understand the effect of different data collection strategies and technologies on the metrics observed.

The checkpointing frequency used by the fault tolerance experiments is another configuration value which was kept constant in the run of the experiments conducted. Checkpointing was configured to honour 'exactly-once' processing guarantees, take one snapshot every five hundred milliseconds, and to use the internal memory of the machine where the job manager was running for storage. The timeout for completion of the snapshot operation was set to one minute. There was no limit to the number of execution retries, but the retry delay was set to five hundred milliseconds. The rationale for choosing those values was to achieve as close as possible a result to exactly-once processing while, at the same time, ensuring that the overhead introduced by taking the snapshots did not significantly interfere with data processing. This had to be true even for the scenario where a node was suddenly lost during the highest volumes/velocities of data ingress and processing. The percentage of data loss was monitored during the fault tolerance experiments. It was also used to fine-tune the checkpointing configuration in test runs of the experiment. Once an acceptable level of data loss was achieved (less than 2%), the configuration was saved and treated as a constant. This study believes that the checkpointing frequency is an experimental setting which could be explored further. A thorough understanding of the relationship between checkpointing frequency, resource utilisation, and percentage of data loss could be particularly useful to architects and implementers of big data stream systems, as could the inclusion of their associated costs when utilising commercial cloud providers.

In terms of the monitoring metrics utilised, one of the limitations of using the Azure Container Monitoring Solution is that it only gathers data at the container level. If node and cluster-level data is needed for further comparisons, either a different solution must be used or the container level data must be aggregated, leaving it susceptible to approximation errors. The adjusted queries detailed in Figure A2 were designed with the help of the Microsoft Log Analytics Team to enable this research to compare the resource utilisation of containers deployed to virtual machines of different specifications. The solution encountered, which aggregates the container utilisation measurements by virtual machine using different functions, has its limitations. The approach is coarse-grained, particularly with regards to calculating maximum utilisations at the node level by employing a technique not dissimilar to windowing in order to assimilate out-of-order, late, and missing data.

Suggestions for improvement involve implementing node and cluster-level monitoring using technologies other than the Azure Container Monitoring Solution. Azure, for instance, offers an extension which provides metrics for any Linux virtual machine [111]. Alternatively, by leveraging open-source monitoring technology such as Prometheus, services such as Weave Cloud are capable of offering all three levels of monitoring [64]. It is important to note, however, that changing the level of monitoring would make it difficult to focus on the containers performing the data processing to the exclusion of all other resource-consuming processes running on the machine, including monitoring processes. Since the co-location of containers is treated as a variable in this set of experiments, the effect that supplementary processes would have on resource consumption is expected to increase as the rate of co-location decreases, simply because there are more virtual machines running.

5.2.2. External Validity

The main limitations to the external validity of MC-BDP's evaluation were the choice of technology, the data transfer configuration, and the complexity of cloud pricing models. The need to limit the experimental part of this research in terms of both time and scope meant that only one prototype was developed and evaluated. However, since MC-BDP is modular and technologically agnostic by design, a number of alternative technologies could have been used instead of the ones selected. Better yet, controlled experiments comparing the performance of equivalent technology would have given architects and potential implementers a good understanding of the advantages and disadvantages of using the different options. Worthwhile substitutions in the prototype implementation would have been the big data framework, the container technology, the orchestrator, or even the platform/operating system.

In terms of data transfer configuration, the current experiments were set up with no resource sharing between containers deployed to the same node, so container co-location did not translate into reduced network utilisation. This study believes, however, that improvements could have been made if resources deployed to the same node had been able to transfer data without using the network. Additionally, it would have been beneficial to measure the reduction in network resource consumption when the container running the job manager service is co-located in the same machine as some of the containers running the task manager service. Although there are disadvantages to this approach, such as increased coupling between the services, it would have been advantageous to understand exactly how much could be gained in terms of reduced network consumption. A more complete picture would indeed be achieved if these savings could be measured not only in terms of a reduction in the number of gigabytes sent and received over the network, but also in terms of the cost incurred. Ref. [106]'s optimisation framework aimed at minimising data transfer costs across clouds is an important step towards a cost-centric approach to task distribution, as opposed to the traditional computational-centric algorithms.

The final external limitation is related to the complexities of cloud resource pricing. Since cloud resources are charged on a pay-as-you-go basis, one could argue that estimation models offering cost predictions are more useful to implementers than those which focus on initial capital expenditure. However, the great complexity and considerable variety in pricing models adopted by different cloud providers mean it is difficult to find a cost prediction model with generic application across providers.

5.2.3. Construct Validity

In terms of construct validity, the main limitations of this study are to do with the cost, and consequently the scope, of its experiments. Since one of the motivations of this study was to adopt the perspective of a cloud consumer, the experiments were conducted using commercial cloud providers and thus relied on generous, although limited, grants awarded by them. Instead of two commercial providers, a larger number could have been utilised. Likewise, larger clusters of up to hundreds of nodes could have been configured to generate a larger spread of performance data and thus provide a more in-depth understanding of the attributes evaluated.

All the simulations involved in the experiments had a fixed duration of five minutes. In an ideal scenario, the run of each experiment would have been increased substantially, ideally by several hours. However, considering that one-hundred and ten experiments were conducted in total, each involving a large amount of data being streamed and processed, and bearing in mind that each scenario also involved a number of test runs to ensure that the desired experimental conditions were being rigorously observed, extending the chosen duration was not feasible under this study's grant limitations.

5.3. Conclusions

MC-BDP was evaluated as part of a case study involving the Estates and Sustainability departments at Leeds Beckett university using a mixed-methods approach which combined

post-positivist and interpretivist elements. In its post-positivist evaluation, presented in this paper, MC-BDP was demonstrated to be scalable and fault-tolerant across clouds. Given the cloud's pay-as-you-go model, whereby consumers are only charged for the resources they utilise, being able to scale up or down is particularly important for SMEODs since budgetary constraints demand that resources be allocated sensibly and waste minimised. The relevance of fault tolerance to the domain of SMEs has been identified in the literature as the need for high availability of their production systems [12].

The results of the technology agnosticism experiments revealed that the use of a super-framework such as Beam to provide portability and interoperability of the data processing code did have a cost in terms of performance, measured as, on average, 30% more CPU, 3% more RAM, and 105% more network usage. The significance of these results to researchers and implementers is in being able to take more informed architectural decisions, particularly in the cloud, where the estimated overhead can be translated into projected costs. This study believes that a concern over costs can be generalised to SMEODs in general, and that the overhead measurements obtained empirically are therefore of direct relevance to the field.

Of similar relevance are the findings of a direct relationship between the windowing rate and resource utilisation in a distributed big data stream system. Since it was demonstrated that the CPU and network utilisation can be predicted using a simple formula based on the windowing rate and constants derived from empirical observation, implementers now have a more accurate way of projecting the cluster size and the cost of data transfers. It is believed that the relationship derived experimentally is of wider application within the field of big data stream processing using commercial clouds.

The container co-location findings confirm that co-location does not significantly affect the performance of CPU-intensive big data stream systems. In the context of cloud computing, this corroboration enables implementers to experiment with a greater number of deployment configurations, from multiple virtual machines with a few containers running in them to a smaller number of more powerful machines hosting a greater number of containers. Factors such as the desired level of fault tolerance (container, node, region, or provider) or projected data transfer costs may play an important part in the decision-making, particularly considering that container-to-container data transfer costs can be reduced or eliminated through co-location.

The practical implications of the MC-BDP study conducted at Leeds Beckett University are far-reaching, particularly for SMEODs. These enterprises, which often operate with limited resources, can greatly benefit from the scalability and fault tolerance aspects demonstrated by MC-BDP. The ability to efficiently scale resources in response to demand, a crucial feature in cloud environments, aligns well with the budgetary constraints common to SMEODs. Additionally, the study's insights into the cost-performance trade-offs inherent in using super-frameworks such as Beam provide these organisations with valuable guidance for making cost-effective architectural decisions. This is particularly relevant in the context of cloud computing, where resource utilization directly impacts operational costs. Moreover, the findings regarding container co-location and its minimal impact on performance enable more flexible and cost-efficient deployment strategies, which are crucial for SMEODs striving for high availability and reduced data transfer costs.

From a theoretical perspective, the MC-BDP study offers new insights into cloud computing and data processing. By establishing a relationship between windowing rates and resource consumption in stream architectures, the current study enhances our understanding of resource allocation and cost forecasting, particularly in cloud environments. Additionally, the exploration of container co-location in CPU-intensive settings reveals a minimal performance impact, broadening our theoretical knowledge in distributed big data stream processing. These insights not only aid SMEODs but also enrich the broader field of cloud computing, laying groundwork for future research and innovation.

Author Contributions: T.V. built the prototype, performed the experiments, collected the data, and wrote the manuscript. A.-L.K. produced the evaluation graphs, tables, and quantitative analysis. D.M. reviewed the draft manuscript and made editing suggestions. All authors have read and agreed to the published version of the manuscript.

Funding: This work made use of the Open Science Data Cloud (OSDC), which is an Open Commons Consortium (OCC)-sponsored project. Cloud computing resources were provided by Google Cloud and Microsoft Azure for Research awards. Container and cloud native technologies were provided by Weaveworks.

Institutional Review Board Statement: This research project received ethics approval from the Local Research Ethics Committee of the School of Built Environment, Engineering and Computing at Leeds Beckett University. Informed consent was obtained from all individual participants included in the study.

Informed Consent Statement: Participants of the energy efficiency case study signed informed consent regarding publishing their data.

Data Availability Statement: The data used to support the findings of this study is available from the corresponding author upon request. The data is not publicly available due to privacy.

Acknowledgments: We would like to thank the OSDC, Google Cloud, Microsoft Azure for Research, and Weaveworks for their support of this research.

Conflicts of Interest: The authors declare that they have no competing interest.

Appendix A

Appendix A.1 Supplementary Tables

Table A1. MC-BDP reference architecture layers.

Layer	Description
Horizontal Layers	
Persistence Layer	The persistence layer consists of file stores, disk space, and relational and non-relational databases. This layer is used differently by components in other layers, namely, the node, container, service, and messaging layers.
Node Layer	The node layer is composed of virtual machines to which containers are deployed. In order to maximise economies of scale, MC-BDP recommends an infrastructure based on commercial clouds and, where possible, it recommends that multiple clouds be utilised simultaneously. This is a known strategy for mitigating the risk of vendor lock-in, as identified in the literature [18,22,103].
Container Layer	The container layer consists of containers running services based on container images. Since images contain all the necessary configuration that a service needs to run, nothing needs to be installed on the platform other than the container runtime, thus allowing consumers to compare offerings by different cloud providers more easily and to migrate to a different provider if needed.
Networking Layer	The networking layer represents a network which allows containers deployed to different nodes at different locations to communicate seamlessly. MC-BDP departs from the traditional approach of networking machines to that of networking containers.
Orchestration Layer	MC-BDP's orchestration layer is responsible for launching, stopping, and managing containers in a cluster. It is therefore responsible for managing services deployed to containers, registering additional nodes from different clouds (or removing them), scaling the number of containers that a service runs on, controlling which containers run on which nodes, and monitoring the overall state of the cluster.
Service Layer	The service layer comprises the applications deployed to the container cluster, which range from smaller-scale deployments such as front-end user interfaces to larger-scale frameworks for big data processing distributed across hundreds of containers.

Table A1. *Cont.*

Layer	Description
Vertical Layers	
Security Layer	The security layer is orthogonal to all other layers since it is implemented in multiple contexts. Encryption, for example, can be configured independently at the framework, orchestration, networking, messaging, and persistence levels. Due to the complexity inherent to the security aspect of large-scale cloud-based systems, our study recommends addressing it through a systematic and comprehensive framework which is multi-layer and multi-purpose, such as the Cloud Computing Adoption Framework (CCAF) [62].
Monitoring Layer	The monitoring layer consists of services aimed at providing metrics related to the performance of specific components. Since diverse aspects of a system can and usually are monitored, this layer is also orthogonal to the others, and would likewise benefit from a systematic approach such as a multi-layer and multi-purpose framework.
Messaging Layer	The messaging layer is used primarily to facilitate the transmission of data from one system to another. One example of such usage in the context of streaming architectures is use as a sink or output for real-time data from IOT devices and as a source or input for big data processing frameworks. Likewise, the messaging layer could be configured as a sink for the result of the data processing performed by the big data framework and as a source for subsequent processing by the same or different framework.

Table A2. Experimental design for the evaluation of MC-BDP reference architecture.

Experimental Design	Environment Setup Using Multi-Cloud Clusters of Azure and Google Cloud Virtual Machines.			
Experimental Setup	Experiment	Parallelism	Azure VMs	Google VMs
	Scalability			
	Single-Cloud Three Workers	3	3	0
	Multi-Cloud Three Workers	3	2	1
	Single-Cloud Six Workers	6	6	0
	Multi-Cloud Six Workers	6	2	4
	Fault Tolerance			
	Single-Cloud Three Workers	2	3	0
	Multi-Cloud Three Workers	2	2	1
	Single-Cloud Six Workers	4	6	0
	Multi-Cloud Six Workers	4	4	2
	Technology Agnosticism			
	Single-Cloud Three Workers Beam	3	3	0
	Single-Cloud Three Workers Flink	3	3	0
	Multi-Cloud Three Workers Beam	3	2	1
	Multi-Cloud Three Workers Flink	3	2	1
	Single-Cloud Six Workers Beam	6	6	2
	Single-Cloud Six Workers Flink	6	6	2
	Multi-Cloud Six Workers Beam	6	4	0
	Multi-Cloud Six Workers Flink	6	4	0

Table A2. *Cont.*

Experimental Design	Environment Setup Using Multi-Cloud Clusters of Azure and Google Cloud Virtual Machines.			
Experimental Setup	Experiment	Parallelism	Azure VMs	Google VMs
	Windowing Rate vs. Resource Utilisation			
	Multi-Cloud Three Workers	3	2	1
	Multi-Cloud Six Workers	6	3	3
	Multi-Cloud Ten Workers	10	6	4
	Container Co-location			
	One Container per node	8	4	4
	Two Containers per node	8	2	2
	Four Containers per node	8	1	1

Table A3. Experiments conducted for the evaluation of MC-BDP reference architecture.

Experiments Conducted	Scalability					
	Single-Cloud Three Workers	Exp. 1 2 records/min	Exp. 2 2 records/s	Exp. 3 2 records/100 s	Exp. 4 2 records/10 ms	Exp. 5 2 records/ms
	Multi-Cloud Three Workers	Exp. 6 2 records/min	Exp. 7 2 records/s	Exp. 8 2 records/100 s	Exp. 9 2 records/100 s	Exp. 10 2 records/ms
	Single-Cloud Six Workers	Exp. 11 2 records/min	Exp. 12 2 records/s	Exp. 13 2 records/100 s	Exp. 14 2 records/100 s	Exp. 15 2 records/ms
	Multi-Cloud Six Workers	Exp. 16 2 records/min	Exp. 17 2 records/s	Exp. 18 2 records/100 s	Exp. 19 2 records/100 s	Exp. 20 2 records/ms
	Fault Tolerance					
	Single-Cloud Three Workers	Exp. 21 2 records/min	Exp. 22 2 records/s	Exp. 23 2 records/100 s	Exp. 24 2 records/10 ms	Exp. 25 2 records/ms
	Multi-Cloud Three Workers	Exp. 26 2 records/min	Exp. 27 2 records/s	Exp. 28 2 records/100 s	Exp. 29 2 records/10 ms	Exp. 30 2 records/ms
	Single-Cloud Six Workers	Exp. 31 2 records/min	Exp. 32 2 records/s	Exp. 33 2 records/100 s	Exp. 34 2 records/10 ms	Exp. 35 2 records/ms
	Multi-Cloud Six Workers	Exp. 36 2 records/min	Exp. 37 2 records/s	Exp. 38 2 records/100 s	Exp. 39 2 records/10 ms	Exp. 40 2 records/ms
	Technology Agnosticism					
	Single-Cloud Three Workers Beam	Exp. 41 2 records/min	Exp. 42 2 records/s	Exp. 43 2 records/100 s	Exp. 44 2 records/10 ms	Exp. 45 2 records/ms
	Single-Cloud Three Workers Flink	Exp. 46 2 records/min	Exp. 47 2 records/s	Exp. 48 2 records/100 s	Exp. 49 2 records/10 ms	Exp. 50 2 records/ms
	Multi-Cloud Three Workers Beam	Exp. 51 2 records/min	Exp. 52 2 records/s	Exp. 53 records/100 s	Exp. 54 2 records/10 ms	Exp. 55 2 records/ms
	Multi-Cloud Three Workers Flink	Exp. 56 2 records/min	Exp. 57 2 records/s	Exp. 58 records/100 s	Exp. 59 records/10 ms	Exp. 60 2 records/ms
	Single-Cloud Six Workers Beam	Exp. 61 2 records/min	Exp. 62 2 records/s	Exp. 63 2 records/100 s	Exp. 64 2 records/10 ms	Exp. 65 2 records/ms
	Single-Cloud Six Workers Flink	Exp. 66 2 records/min	Exp. 67 2 records/s	Exp. 68 2 records/100 s	Exp. 69 2 records/10 ms	Exp. 70 2 records/ms
	Multi-Cloud Six Workers Beam	Exp. 71 2 records/min	Exp. 72 2 records/s	Exp. 73 2 records/100 s	Exp. 74 2 records/10 ms	Exp. 75 2 records/ms
	Multi-Cloud Six Workers Flink	Exp. 76 2 records/min	Exp. 77 2 records/s	Exp. 78 2 records/100 s	Exp. 79 2 records/10 ms	Exp. 80 2 records/ms

Table A3. *Cont.*

Experiments Conducted	Windowing Rate versus Resource Utilisation					
	Multi-Cloud Three Workers	Exp. 81 R = 2.0 5 s start every 10 s	Exp. 82 R = 1.5 5 s start every 7.5 s	Exp. 83 R = 1.0 5 s start every 5 s	Exp. 84 R = 0.2 5 s start every 1 s	Exp. 85 R = 0.1 5 s start every 0.5 s
	Multi-Cloud Six Workers	Exp. 86 R = 2.0 5 s start every 10 s	Exp. 87 R = 1.5 5 s start every 7.5 s	Exp. 88 R = 1.0 5 s start every 5 s	Exp. 89 R = 0.2 5 s start every 1 s	Exp. 90 R = 0.1 5 s start every 0.5 s
	Multi-Cloud Ten Workers	Exp. 91 R = 2.0 5 s start every 10 s	Exp. 92 R = 1.5 5 s start every 7.5 s	Exp. 93 R = 1.0 5 s start every 5 s	Exp. 94 R = 0.2 5 s start every 1 s	Exp. 95 R = 0.1 5 s start every 0.5 s
	Container Co-location					
	One Container per node	Exp. 96 R = 2.0 5 s start every 10 s	Exp. 97 R = 1.5 5 s start every 7.5 s	Exp. 98 R = 1.0 5 s start every 5 s	Exp. 99 R = 0.2 5 s start every 1 s	Exp. 100 R = 0.1 5 s start every 0.5 s
	Two Containers per node	Exp. 101 R = 2.0 5 s start every 10 s	Exp. 102 R = 1.5 5 s start every 7.5 s	Exp. 103 R = 1.0 5 s start every 5 s	Exp. 104 R = 0.2 5 s start every 1 s	Exp. 105 R = 0.1 5 s start every 0.5 s
	Four Containers per node	Exp. 106 R = 2.0 5 s start every 10 s	Exp. 107 R = 1.5 5 s start every 7.5 s	Exp. 108 R = 1.0 5 s start every 5 s	Exp. 109 R = 0.2 5 s start every 1 s	Exp. 110 R = 0.1 5 s start every 0.5 s

Table A4. Performance metrics used in the evaluation of MC-BDP reference architecture.

	Container CPU Utilisation	Container Memory Utilisation	Container Network Utilisation	Additional Metric
Performance Metrics	Scalability Metrics			
	Average and maximum CPU utilisation by a container during experiment execution time.	Average and maximum memory utilisation by a container during experiment execution time (in MB).	Total number of bytes transmitted and received over a network by a container during experiment execution time.	
	Fault Tolerance Metrics			
	Average and maximum CPU utilisation by a container during experiment execution time.	Average and maximum memory utilisation by a container during experiment execution time (in MB).	Total number of bytes transmitted and received over a network by a container during experiment execution time.	Records Processed Percentage of total number of records processed compared to total number of transmitted records
	Technology Agnosticism Metrics			
	Average and maximum CPU utilisation by a container during experiment execution time.	Average and maximum memory utilisation by a container during experiment execution time (in MB).	Total number of bytes transmitted and received over a network by a container during experiment execution time.	
	Windowing Rate versus Resource Utilisation Metrics			
	Average and maximum CPU utilisation by a container during experiment execution time.	Average and maximum memory utilisation by a container during experiment execution time (in MB).	Total number of bytes transmitted and received over a network by a container during experiment execution time.	Data Volume Processed Total number of kilobytes and total number of records received by each worker for processing after the windowing function is applied.

Table A4. *Cont.*

	Container CPU Utilisation	Container Memory Utilisation	Container Network Utilisation	Additional Metric
	Container Co-Location Metrics			
Performance Metrics	Average and maximum CPU utilisation by data processing containers running a node during experiment execution time.	Average and maximum CPU utilisation by data processing containers running a node during experiment execution time (in MB).	Total number of bytes transmitted and received over a network by the data processing containers during experiment execution time.	Data Volume Processed Total number of kilobytes and total number of records received by each worker for processing after the windowing function is applied.

Table A5. Summary of regression models for average CPU utilisation.

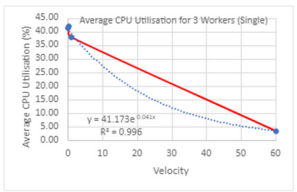
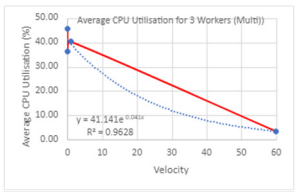
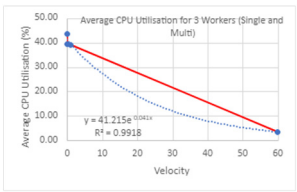
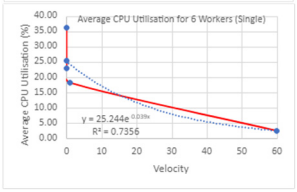
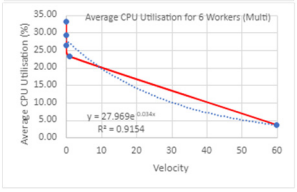
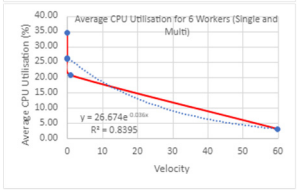
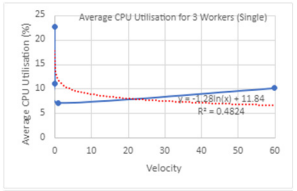
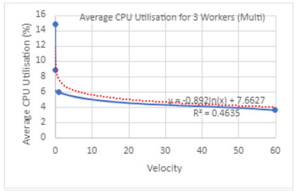
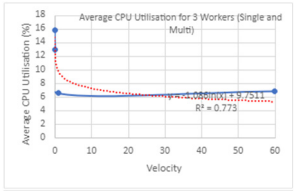
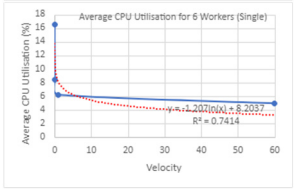
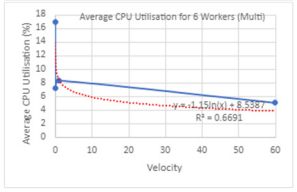
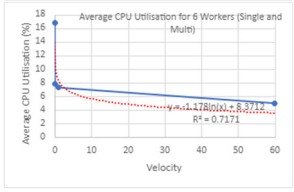
Average CPU Utilisation by Velocity		
Scalability Metrics		
		
		
Fault Tolerance Metrics		
		
		

Table A5. *Cont.*

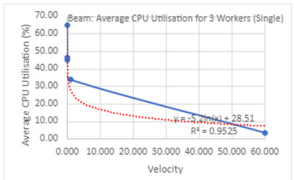
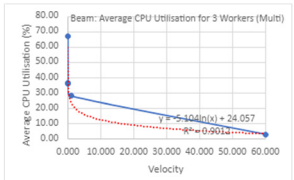
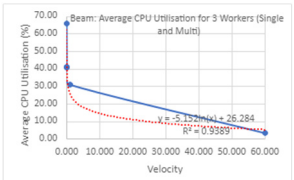
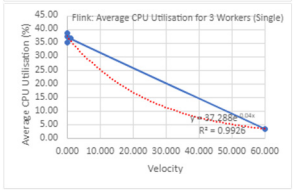
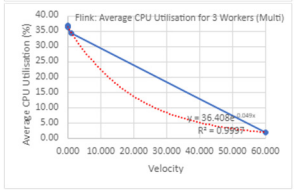
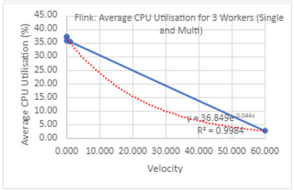
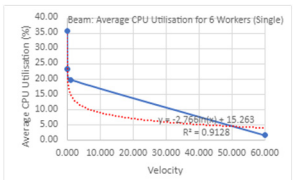
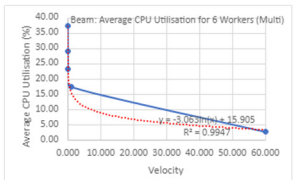
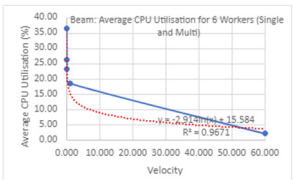
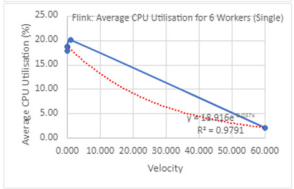
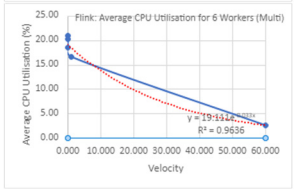
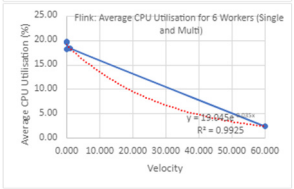
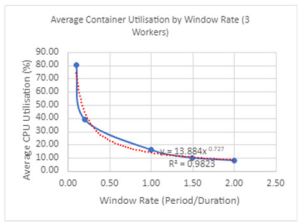
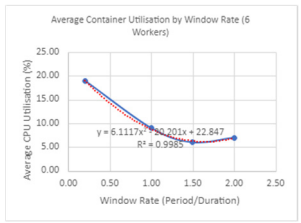
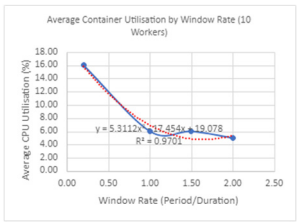
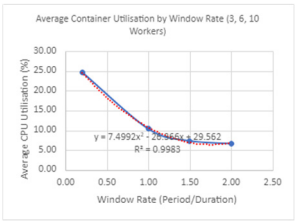
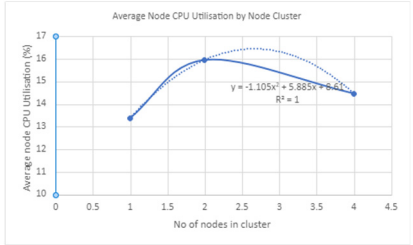
Average CPU Utilisation by Velocity
Technology Agnosticism Metrics (Beam and Flink) for Three Workers
     
Technology Agnosticism Metrics (Beam and Flink) for Six Workers
     
Windowing Rate versus Resource Utilisation Metrics
   
Container Co-location Metrics
Average Node CPU Utilisation by Node Cluster


Table A6. Summary of regression models for maximum CPU utilisation.

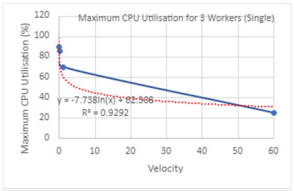
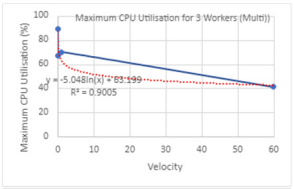
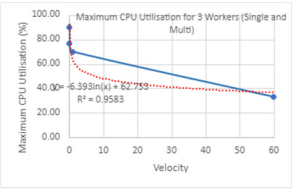
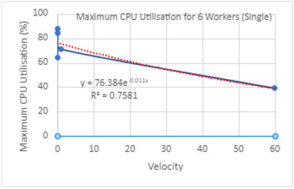
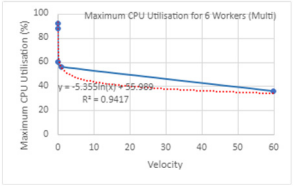
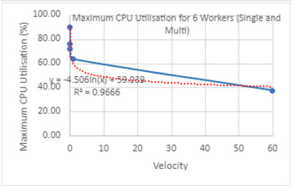
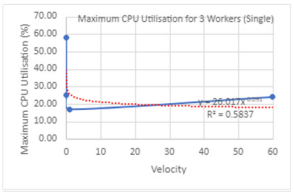
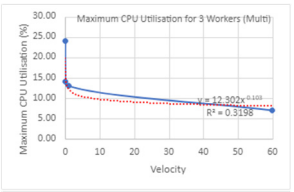
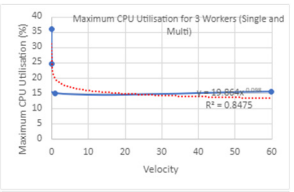
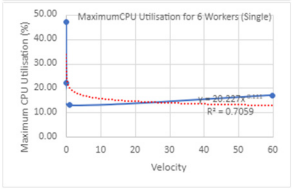
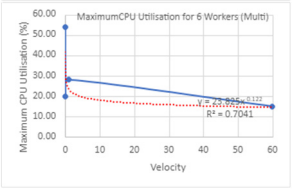
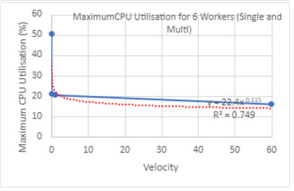
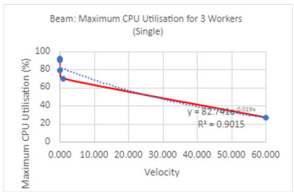
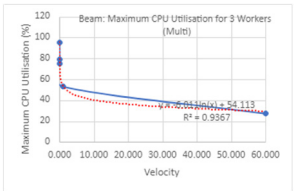
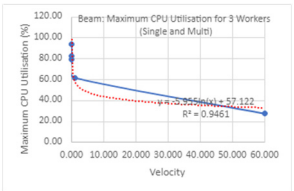
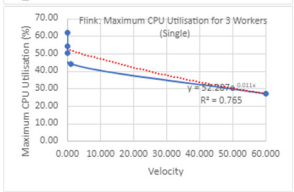
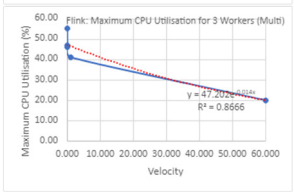
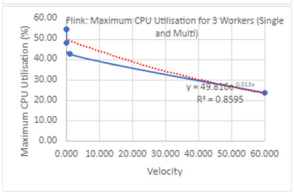
Maximum CPU Utilisation by Velocity		
Scalability Metrics		
		
		
Fault Tolerance Metrics		
		
		
Technology Agnosticism Metrics (Beam and Flink) for Three Workers		
		
		

Table A6. *Cont.*

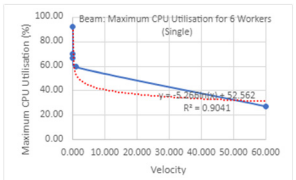
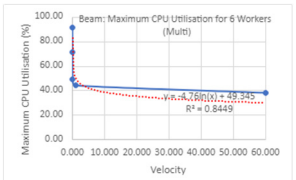
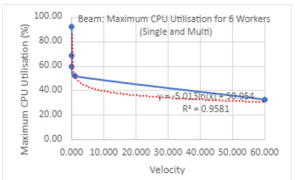
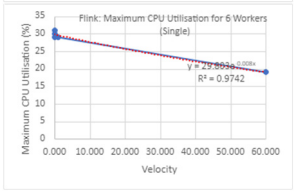
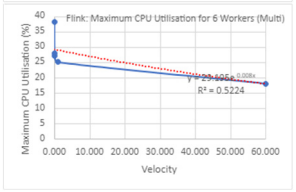
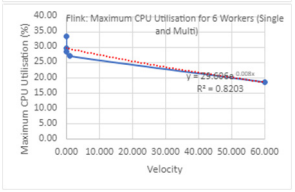
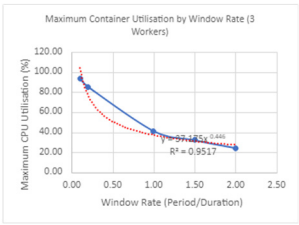
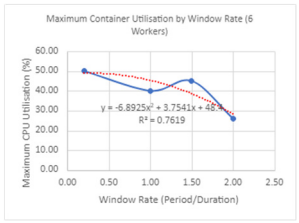
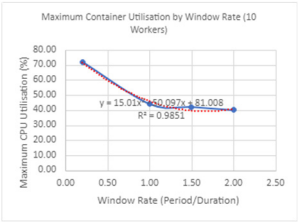
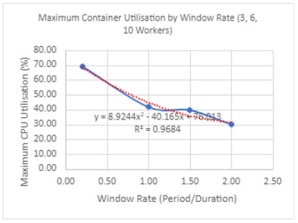
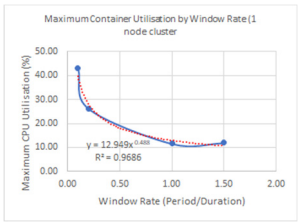
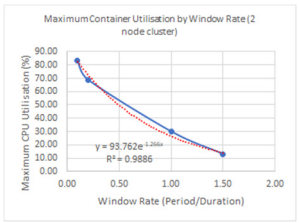
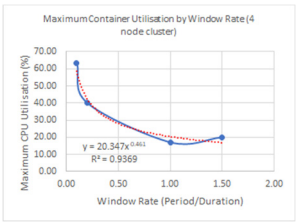
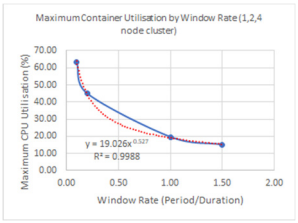
Maximum CPU Utilisation by Velocity		
Technology Agnosticism Metrics (Beam and Flink) for Six Workers		
		
		
Windowing Rate versus Resource Utilisation Metrics		
			
Container Co-location Metrics		
Maximum Node CPU Utilisation by Node Cluster		
			

Table A7. Average memory utilisation graphs and regression models.

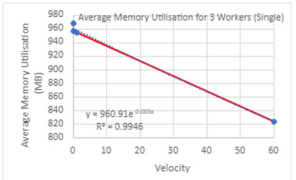
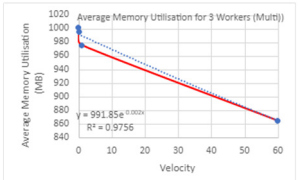
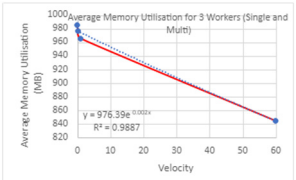
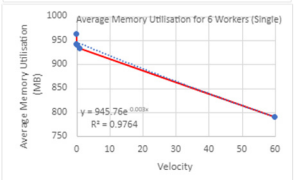
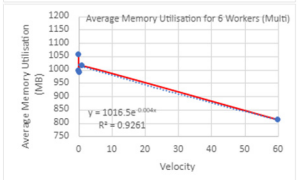
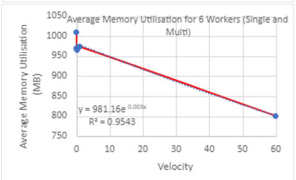
Average Memory Utilisation by Velocity		
Scalability Metrics		
		
		

Table A7. *Cont.*

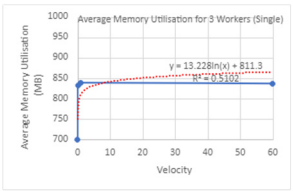
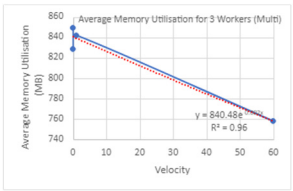
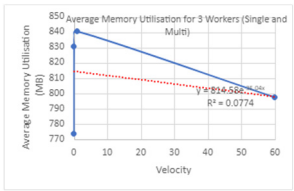
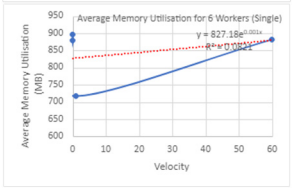
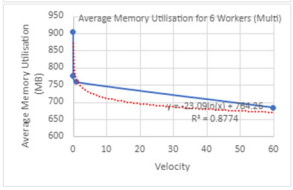
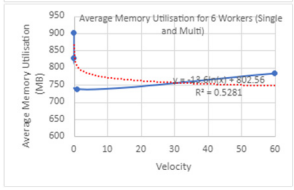
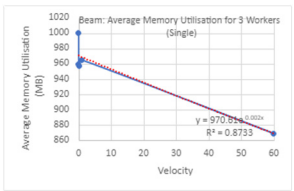
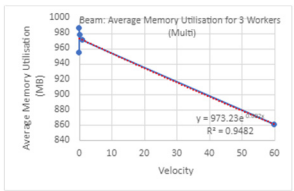
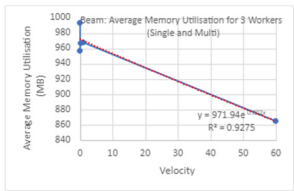
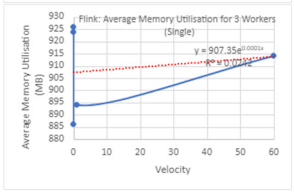
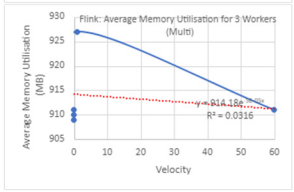
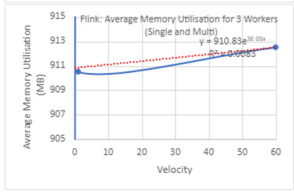
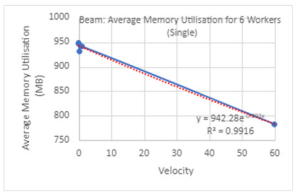
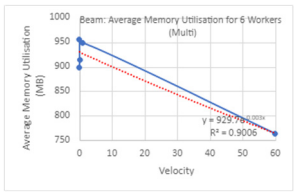
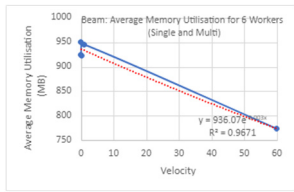
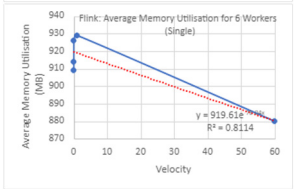
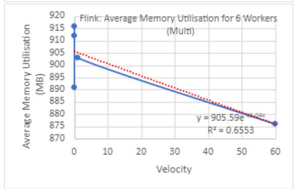
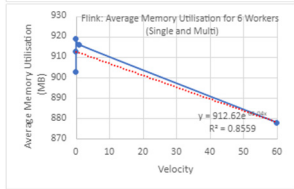
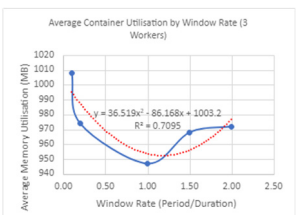
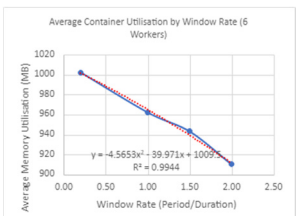
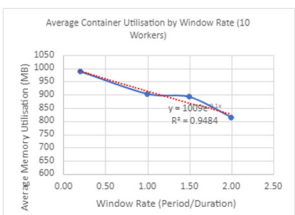
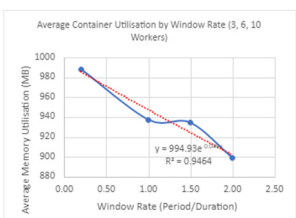
Average Memory Utilisation by Velocity		
Fault Tolerance Metrics		
		
		
Technology Agnosticism Metrics (Beam and Flink) for Three Workers		
		
		
Technology Agnosticism Metrics (Beam and Flink) for Six Workers		
		
		
Windowing Rate versus Resource Utilisation Metrics		
		
		

Table A7. *Cont.*

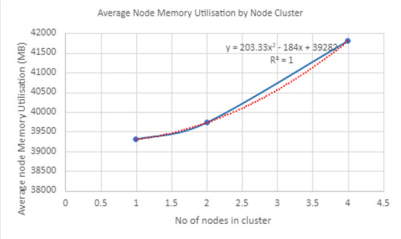
Average Memory Utilisation by Velocity
Container Co-location Metrics
Average Node Memory Utilisation by Node Cluster


Table A8. Summary of regression models for maximum memory utilisation.

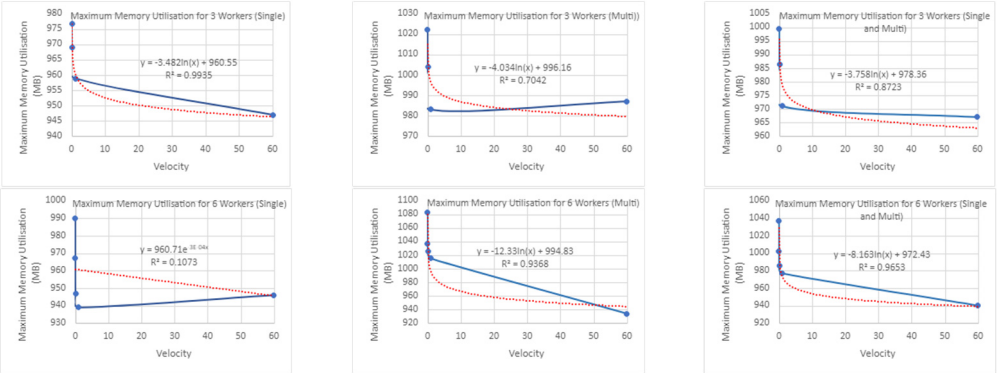
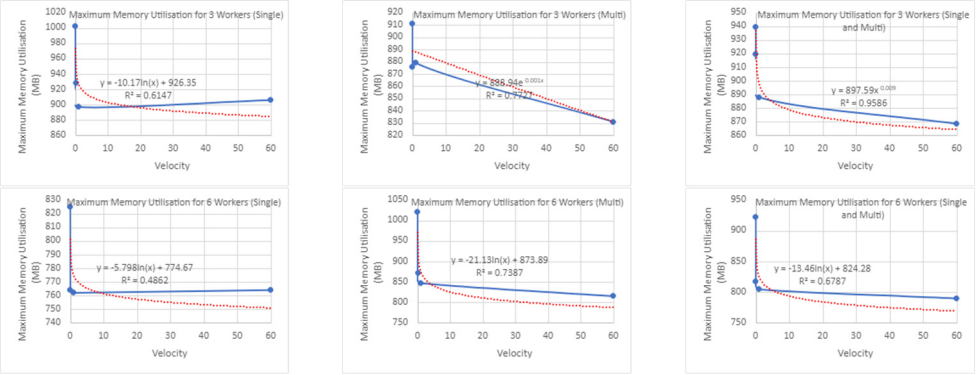
Maximum Memory Utilisation by Velocity
Scalability Metrics

Fault Tolerance Metrics


Table A8. *Cont.*

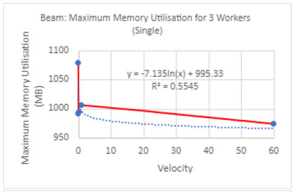
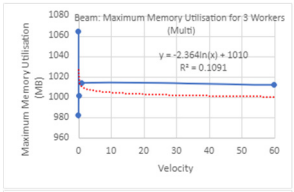
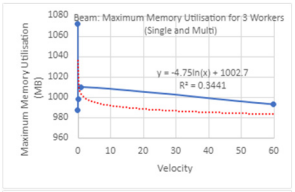
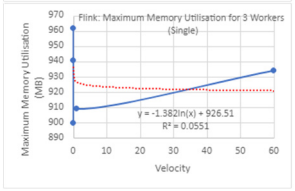
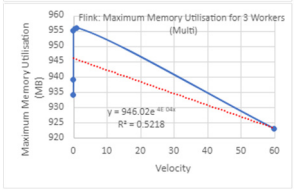
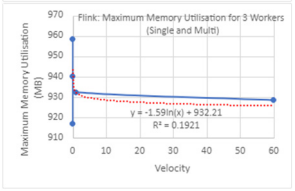
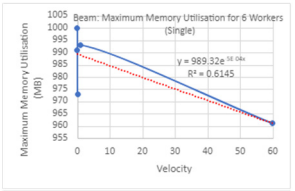
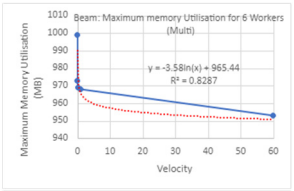
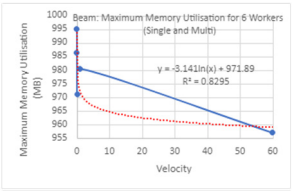
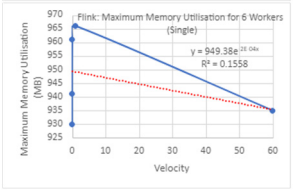
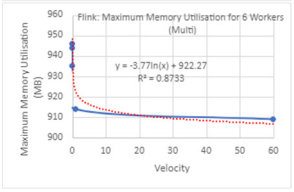
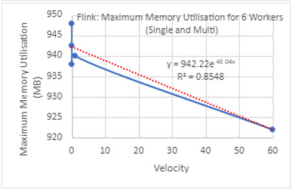
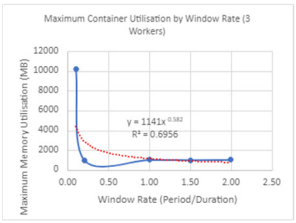
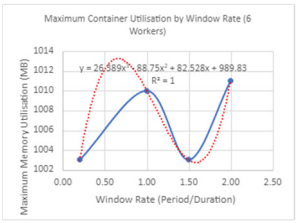
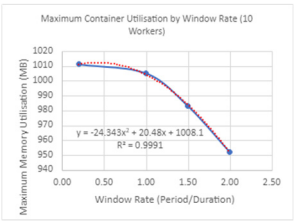
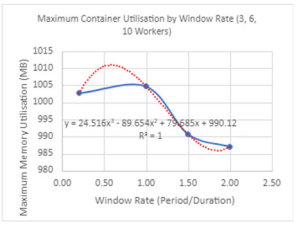
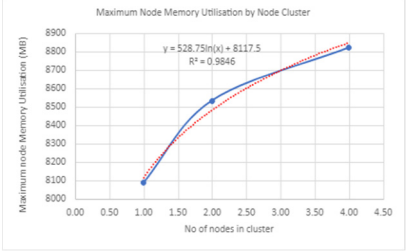
Maximum Memory Utilisation by Velocity		
Technology Agnosticism Metrics (Beam and Flink) for Three Workers		
		
		
Technology Agnosticism Metrics (Beam and Flink) for Six Workers		
		
		
Windowing Rate versus Resource Utilisation Metrics		
		
		
Container Co-location Metrics		
Maximum Node Memory Utilisation by Node Cluster		
		

Table A9. Network utilisation (gigabytes sent over the network)—graphs and regression models.

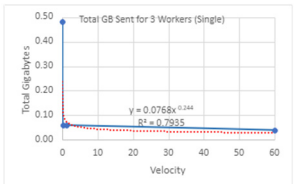
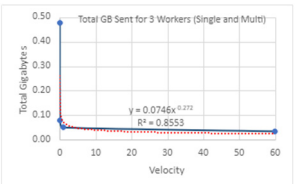
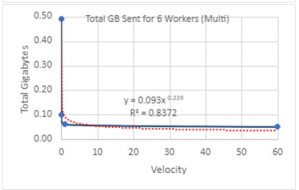
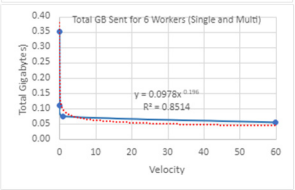
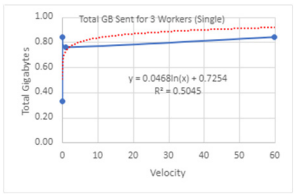
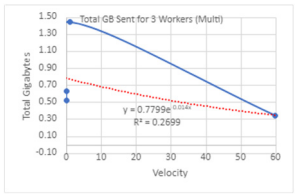
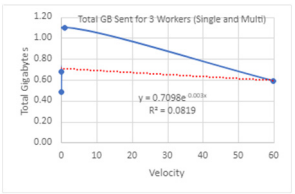
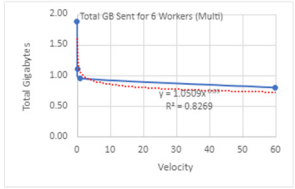
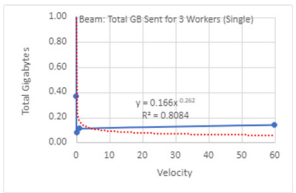
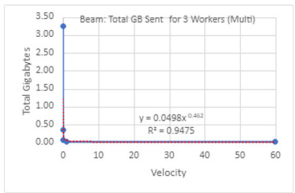
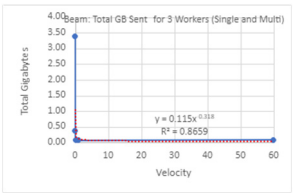
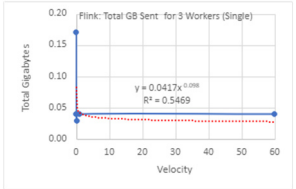
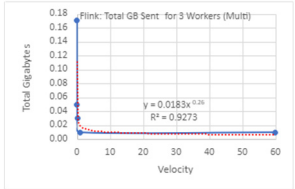
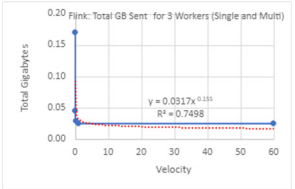
Network Utilisation: Total Number of Gigabytes Sent over the Network		
Scalability Metrics		
		
		
Fault Tolerance Metrics		
		
		
Technology Agnosticism Metrics (Beam and Flink) for Three Workers		
		
		

Table A9. *Cont.*

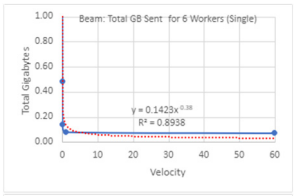
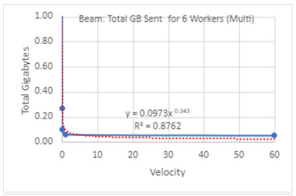
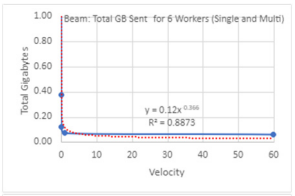
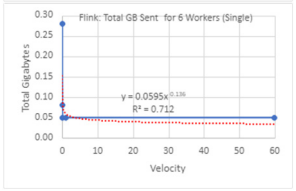
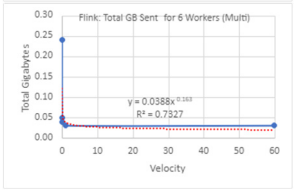
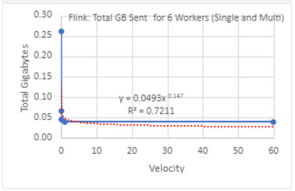
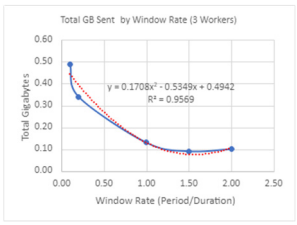
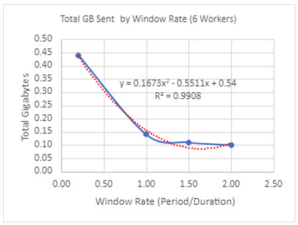
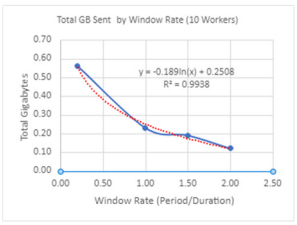
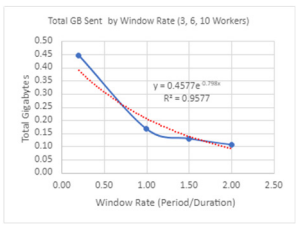
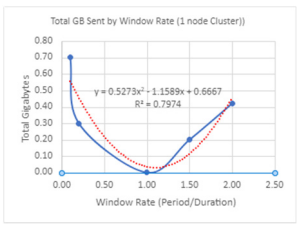
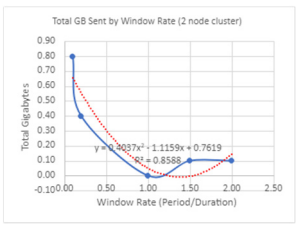
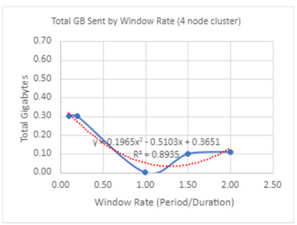
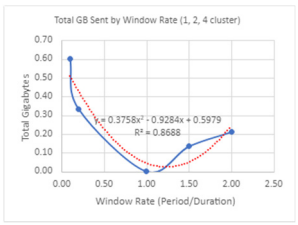
Network Utilisation: Total Number of Gigabytes Sent over the Network		
Technology Agnosticism Metrics (Beam and Flink) for Six Workers		
		
		
Windowing Rate versus Resource Utilisation Metrics		
		
		
Container Co-location Metrics		
Maximum Node Network Utilisation by Node Cluster		
		
		

Table A10. Network utilisation (gigabytes received over the network)—graphs and regression models.

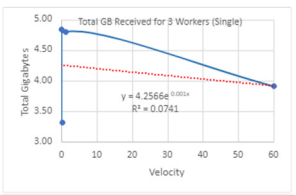
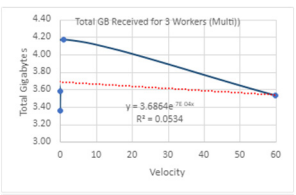
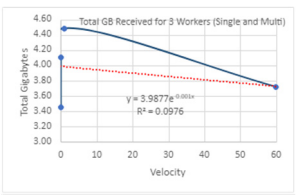
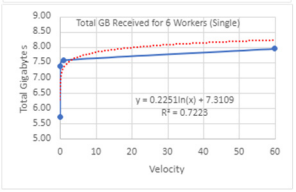
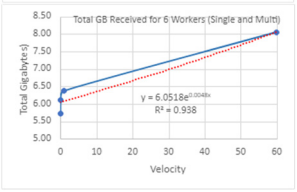
Network Utilisation: Total Number of Gigabytes Received over the Network		
Scalability Metrics		
		
		

Table A10. *Cont.*

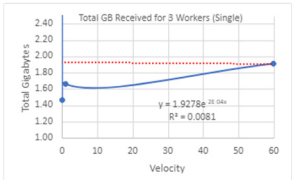
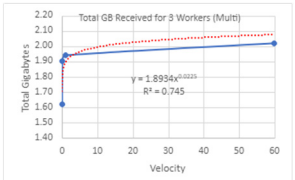
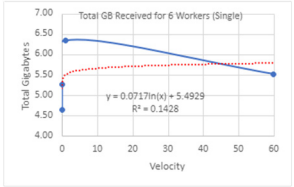
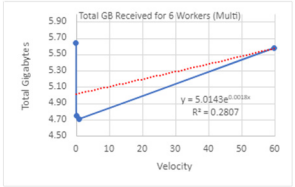
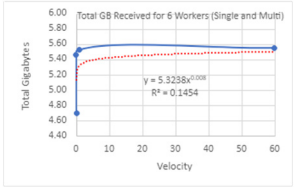
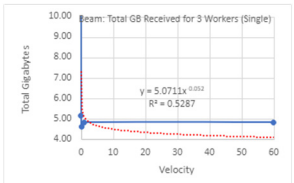
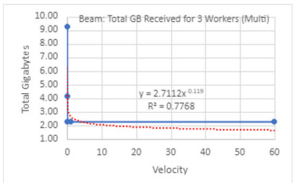
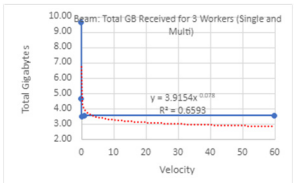
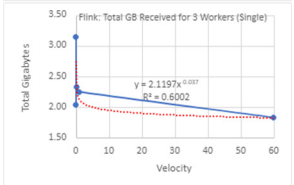
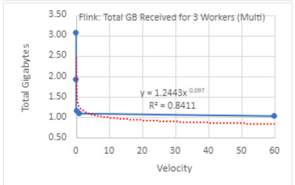
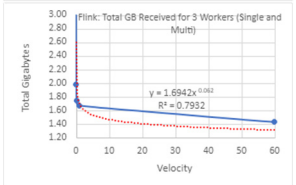
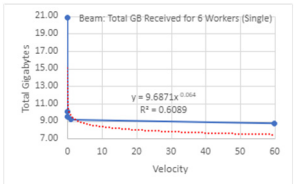
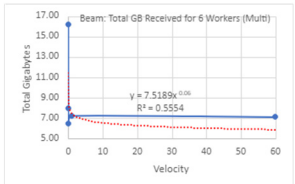
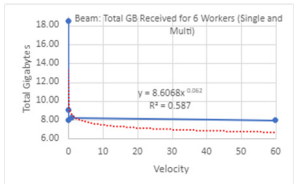
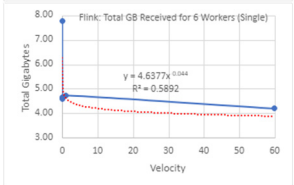
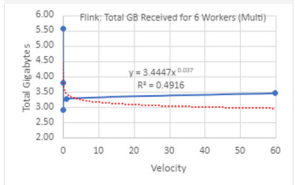
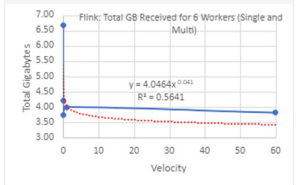
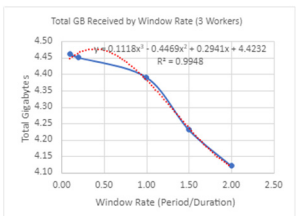
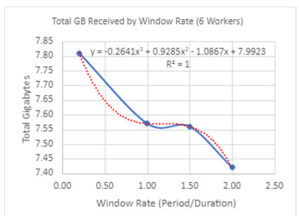
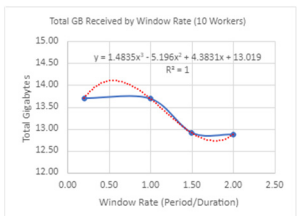
Network Utilisation: Total Number of Gigabytes Received over the Network		
Fault Tolerance Metrics		
		
		
Technology Agnosticism Metrics (Beam and Flink) for Three Workers		
		
		
Technology Agnosticism Metrics (Beam and Flink) for Six Workers		
		
		
Windowing Rate versus Resource Utilisation Metrics		
			

Table A10. *Cont.*

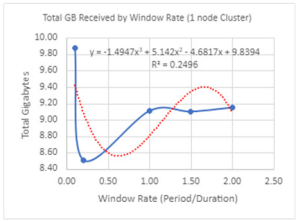
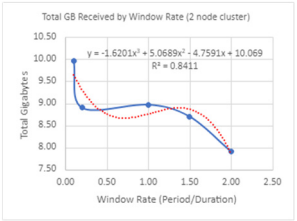
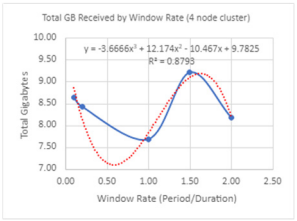
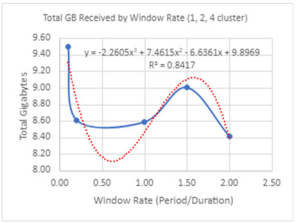
Network Utilisation: Total Number of Gigabytes Received over the Network			
Container Co-location Metrics			
Maximum Node Network Utilisation by Node Cluster			
			

Table A11. Network utilisation for technology agnosticism (gigabytes received and sent by the job manager)—graphs and regression models.

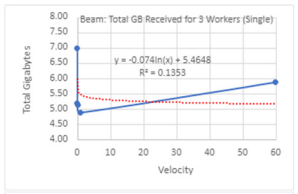
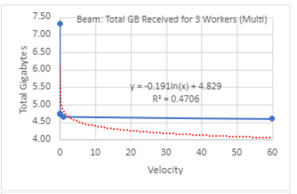
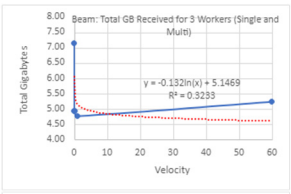
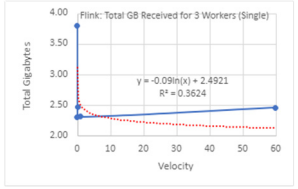
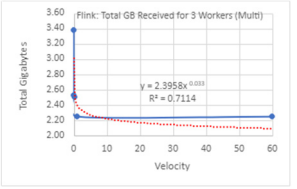
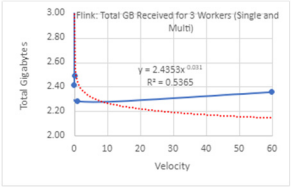
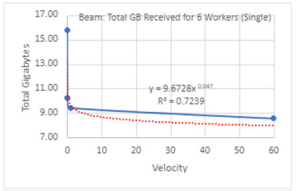
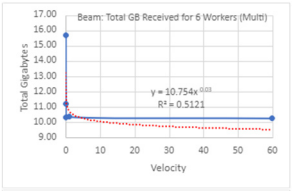
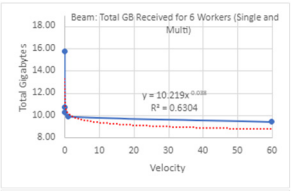
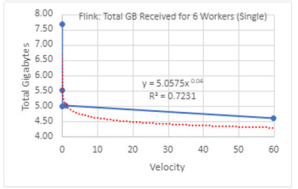
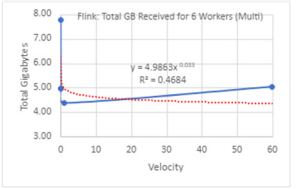
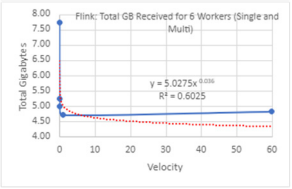
Total Gigabytes Received via the Network by the Job Manager		
Technology Agnosticism (Beam and Flink) for Three Workers		
		
		
Technology Agnosticism (Beam and Flink) for Six Workers		
		
		

Table A11. *Cont.*

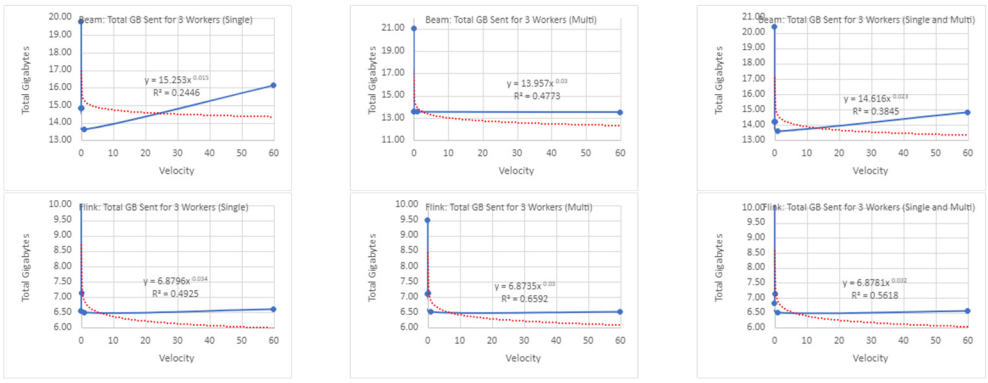
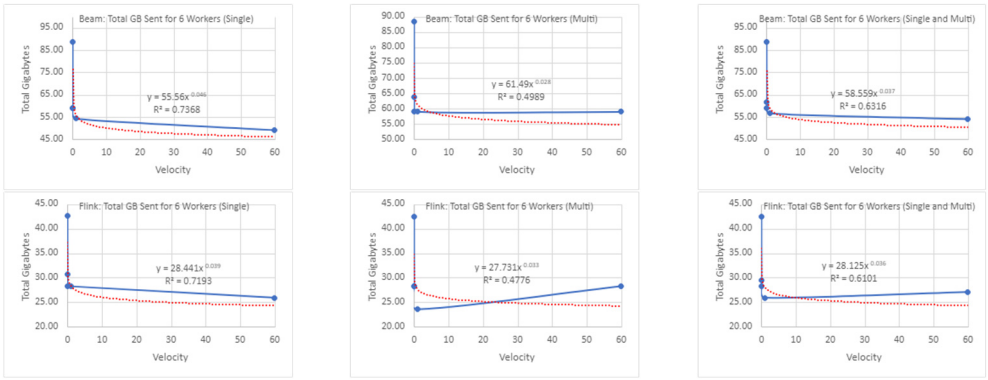
Total Gigabytes Received via the Network by the Job Manager
Total Gigabytes Sent via the Network by the Job Manager
Technology Agnosticism Metrics (Beam and Flink) for Three Workers

Technology Agnosticism Metrics (Beam and Flink) for Six Workers


Table A12. Network utilisation for windowing rate versus resource utilisation (gigabytes received and sent by the job manager)—graphs and regression models.

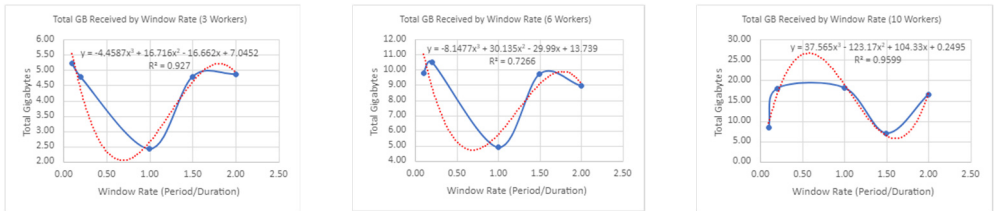
Total Gigabytes Received via the Network by the Job Manager
Windowing Rate versus Resource Utilisation

Total Gigabytes Sent via the Network by the Job Manager

Table A12. *Cont.*

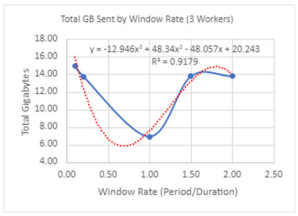
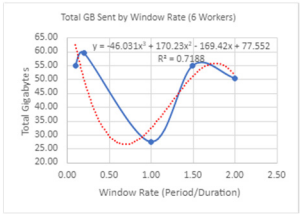
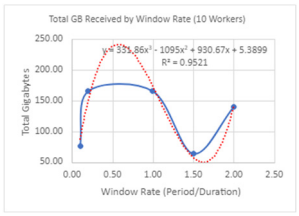
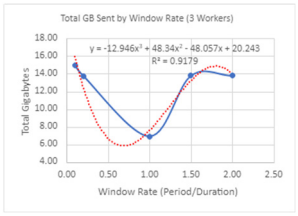
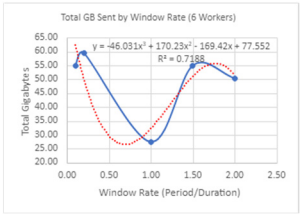
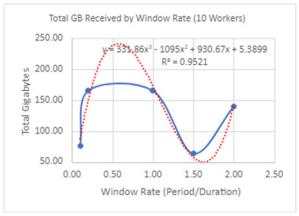
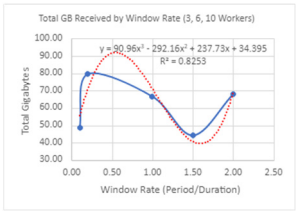
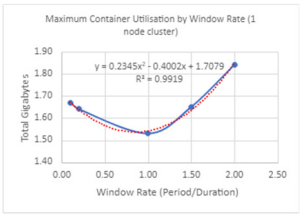
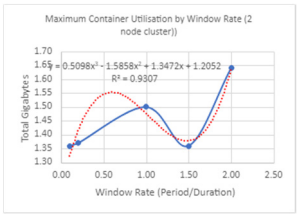
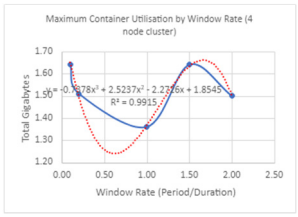
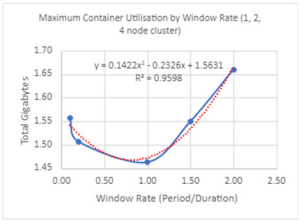
Total Gigabytes Received via the Network by the Job Manager
Windowing Rate versus Resource Utilisation
  

Table A13. Network utilisation for container co-location (gigabytes received and sent by the job manager)—graphs and regression models.

Total Gigabytes Received by the Job Manager over the Network by Number of Containers per Node
Container Co-Location
Maximum Node Network Utilisation by Node Cluster
   
Total Number of GB Sent by the Job Manager over the Network by Number of Containers per Node
Container Co-Location
Maximum Node Network Utilisation by Node Cluster
   

Appendix A.2 Azure Log Analytics Queries

Container CPU Utilisation (Avg)	Container CPU Utilisation (Max)	Container Memory Utilisation (Average)	Container Memory Utilisation (Max)	Network Receive Bytes	Network Send Bytes
• Perf | where ObjectName == "Container" and CounterName == "% Processor Time" | where InstanceName has "taskmanager" | summarize AvgCPUPercent = avg(CounterValue) by Computer, InstanceName	• Perf | where ObjectName == "Container" and CounterName == "% Processor Time" | where InstanceName has "taskmanager" | summarize MaxCPUPercent = max(CounterValue) by Computer, InstanceName	• Perf | where ObjectName == "Container" and CounterName == "Memory Usage MB" and InstanceName has "taskmanager" | summarize AggregatedValue = avg(CounterValue) by Computer, InstanceName	• Perf | where ObjectName == "Container" and CounterName == "Memory Usage MB" and InstanceName has "taskmanager" | summarize AggregatedValue = max(CounterValue) by Computer, InstanceName	• search in (Perf) ObjectName == "Container" and CounterName == "Network Receive Bytes" | where InstanceName has "taskmanager" | summarize NetworkReceiveBytes = sum(CounterValue) by Computer, InstanceName	• search in (Perf) ObjectName == "Container" and CounterName == "Network Send Bytes" | where InstanceName has "taskmanager" | summarize NetworkSendBytes = sum(CounterValue) by Computer, InstanceName

Figure A1. Azure Log Analytics queries to extract metrics.

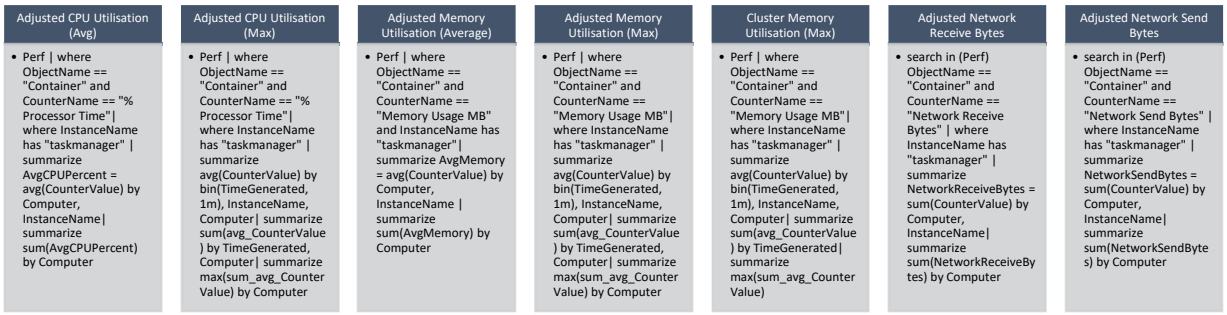


Figure A2. Azure Log Analytics queries to extract metrics—adjusted to aggregate by node.

References

1. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. *ACM. Commun.* **2008**, *51*, 107. [CrossRef]
2. Patel, K.; Sakaria, Y.; Bhadane, C. Real Time Data Processing Frameworks. *Int. J. Data Min. Knowl. Manag. Process* **2015**, *5*, 49–63. [CrossRef]
3. Li, J.; Maier, D.; Tufte, K.; Papadimos, V.; Tucker, P.A. Semantics and Evaluation Techniques for Window Aggregates in Data Streams. In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 14–16 June 2005; pp. 311–322. [CrossRef]
4. Akidau, T.; Balikov, A.; Bekiroğlu, K.; Chernyak, S.; Haberman, J.; Lax, R.; McVeety, S.; Mills, D.; Nordstrom, P.; Whittle, S. MillWheel: Fault-tolerant stream processing at internet scale. *Proc. VLDB Endow.* **2013**, *6*, 1033–1044. [CrossRef]
5. Kreps, J. Questioning the Lambda Architecture—O'Reilly Media. 2 July 2014. Available online: https://www.oreilly.com/ideas/questioning-the-lambda-architecture (accessed on 28 October 2016).
6. Chen, G.J.; Wiener, J.L.; Iyer, S.; Jaiswal, A.; Lei, R.; Simha, N.; Wang, W.; Wilfong, K.; Williamson, T.; Yilmaz, S. Realtime Data Processing at Facebook. In Proceedings of the 2016 International Conference on Management of Data, New York, NY, USA, 25 June 2016; pp. 1087–1098. [CrossRef]
7. Krishnan, S. Discovery and Consumption of Analytics Data at Twitter. Twitter Engineering Blog. 29 June 2016. Available online: https://blog.twitter.com/engineering/en_us/topics/insights/2016/discovery-and-consumption-of-analytics-data-at-twitter.html (accessed on 9 February 2018).
8. Kashlev, A.; Lu, S.; Mohan, A. Big Data Workflows: A Reference Architecture and The Dataview System. *Serv. Trans. Big Data* **2017**, *4*, 19. [CrossRef]
9. Ta, V.-D.; Liu, C.-M.; Nkabinde, G.W. Big data stream computing in healthcare real-time analytics. In Proceedings of the 2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), Chengdu, China, 5–7 July 2016; pp. 37–42. [CrossRef]
10. Klein, J.; Buglak, R.; Blockow, D.; Wuttke, T.; Cooper, B. A Reference Architecture for Big Data Systems in the National Security Domain. In Proceedings of the 2016 IEEE/ACM 2nd International Workshop on Big Data Software Engineering (BIGDSE), 16 May 2016; pp. 51–57.
11. Ardagna, C.A.; Ceravolo, P.; Damiani, E. Big data analytics as-a-service: Issues and challenges. In Proceedings of the 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 17–20 December 2016; pp. 3638–3644. [CrossRef]
12. Kalan, R.S.; Ünalir, M.O. Leveraging big data technology for small and medium-sized enterprises (SMEs). In Proceedings of the 2016 6th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 20–21 October 2016; pp. 1–6. [CrossRef]
13. Liu, Y.; Soroka, A.; Han, L.; Jian, J.; Tang, M. Cloud-based big data analytics for customer insight-driven design innovation in SMEs. *Int. J. Inf. Manag.* **2020**, *51*, 102034. [CrossRef]
14. Sen, D.; Ozturk, M.; Vayvay, O. An Overview of Big Data for Growth in SMEs. *Procedia Soc. Behav. Sci.* **2016**, *235*, 159–167. [CrossRef]
15. Vecchio, P.D.; Minin, A.D.; Petruzzelli, A.M.; Panniello, U.; Pirri, S. Big data for open innovation in SMEs and large corporations: Trends, opportunities, and challenges. *Creat. Innov. Manag.* **2018**, *27*, 6–22. [CrossRef]
16. Shetty, J.P.; Panda, R. An overview of cloud computing in SMEs. *J. Glob. Entrep. Res.* **2021**, *11*, 175–188. [CrossRef]
17. Sultan, N.A. Reaching for the "cloud": How SMEs can manage. *Int. J. Inf. Manag.* **2011**, *31*, 272–278. [CrossRef]
18. Opara-Martins, J.; Sahandi, R.; Tian, F. Critical analysis of vendor lock-in and its impact on cloud computing migration: A business perspective. *J. Cloud Comput.* **2016**, *5*, 4. [CrossRef]
19. Bange, C.; Grosser, T.; Janoschek, N. Big Data Use Cases 2015—Getting Real on Data Monetization. July 2015. Available online: http://barc-research.com/research/big-data-use-cases-2015/ (accessed on 15 February 2019).
20. Assis, M.R.M.; Bittencourt, L.F. A survey on cloud federation architectures: Identifying functional and non-functional properties. *J. Netw. Comput. Appl.* **2016**, *72*, 51–71. [CrossRef]
21. Naik, N. Docker container-based big data processing system in multiple clouds for everyone. In Proceedings of the 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, 11–13 October 2017; pp. 1–7. [CrossRef]

22. Satzger, B.; Hummer, W.; Inzinger, C.; Leitner, P.; Dustdar, S. Winds of Change: From Vendor Lock-In to the Meta Cloud. *IEEE Internet Comput.* **2013**, *17*, 69–73. [CrossRef]
23. Silva, G.C.; Rose, L.M.; Calinescu, R. Towards a Model-Driven Solution to the Vendor Lock-In Problem in Cloud Computing. In Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Bristol, UK, 2–5 December 2013; Volume 1, pp. 711–716. [CrossRef]
24. Toosi, A.N.; Calheiros, R.N.; Buyya, R. Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey. *ACM Comput. Surv.* **2014**, *47*, 1–47. [CrossRef]
25. Bernstein, D. Cloud Foundry Aims to Become the OpenStack of PaaS. *IEEE Cloud Comput.* **2014**, *1*, 57–60. [CrossRef]
26. Leung, A.; Spyker, A.; Bozarth, T. Titus: Introducing Containers to the Netflix Cloud. *Queue* **2017**, *15*, 53–77. [CrossRef]
27. Al-Dhuraibi, Y.; Paraiso, F.; Djarallah, N.; Merle, P. Elasticity in Cloud Computing: State of the Art and Research Challenges. *IEEE Trans. Serv. Comput.* **2018**, *11*, 430–447. [CrossRef]
28. Rodriguez, M.A.; Buyya, R. Container-based cluster orchestration systems: A taxonomy and future directions. *Softw. Pract. Exp.* **2019**, *49*, 698–719. [CrossRef]
29. Pahl, C. Containerization and the PaaS Cloud. *IEEE Cloud Comput.* **2015**, *2*, 24–31. [CrossRef]
30. Pahl, C.; Lee, B. Containers and Clusters for Edge Cloud Architectures—A Technology Review. In Proceedings of the 2015 3rd International Conference on Future Internet of Things and Cloud, Rome, Italy, 24–26 August 2015; pp. 379–386. [CrossRef]
31. Vergilio, T.; Ramachandran, M. Non-functional Requirements for Real World Big Data Systems—An Investigation of Big Data Architectures at Facebook, Twitter and Netflix'. In Proceedings of the 13th International Conference on Software Technologies, Porto, Portugal, 26–28 July 2018; pp. 833–840. [CrossRef]
32. Silva, G.C.; Rose, L.M.; Calinescu, R. A Systematic Review of Cloud Lock-In Solutions. In Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Bristol, UK, 2–5 December 2013; Volume 2, pp. 363–368. [CrossRef]
33. Jokonya, O. Investigating Open Source Software Benefits in Public Sector. In Proceedings of the 2015 48th Hawaii International Conference on System Sciences, Kauai, HI, USA, 5–8 January 2015; pp. 2242–2251. [CrossRef]
34. Palyart, M.; Murphy, G.C.; Masrani, V. A Study of Social Interactions in Open Source Component Use. *IEEE Trans. Softw. Eng.* **2018**, *44*, 1132–1145. [CrossRef]
35. Al-Hazmi, Y.; Campowsky, K.; Magedanz, T. A monitoring system for federated clouds. In Proceedings of the 2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET), Paris, France, 28–30 November 2012; pp. 68–74. [CrossRef]
36. Palos-Sanchez, P.R. Drivers and Barriers of the Cloud Computing in SMEs: The Position of the European Union. *Harv. Deusto Bus. Res.* **2017**, *6*, 116–132. [CrossRef]
37. Hui, K. AWS 101: Regions and Availability Zones. Rackspace Blog. 16 February 2017. Available online: https://blog.rackspace.com/aws-101-regions-availability-zones (accessed on 22 February 2019).
38. Scott, R. Mitigating an AWS Instance Failure with the Magic of Kubernetes. Medium. 1 March 2017. Available online: https://medium.com/spire-labs/mitigating-an-aws-instance-failure-with-the-magic-of-kubernetes-128a44d44c14 (accessed on 24 January 2018).
39. Brodkin, J.; EC, A. "Availability Zones" into Question. Network World. 21 April 2011. Available online: https://www.networkworld.com/article/2202805/cloud-computing/amazon-ec2-outage-calls{-}{-}availability-zones{-}{-}into-question.html (accessed on 22 February 2019).
40. Dayaratna, A. Microsoft Azure Recovers From Multi-Region Azure DNS Service Disruption. *Cloud Comput. Today* **2020**, 51–56. Available online: https://cloud-computing-today.com/2016/09/15/microsoft-azure-recovers-from-multi-region-azure-dns-service-disruption/ (accessed on 22 February 2019).
41. Rattihalli, G. Exploring Potential for Resource Request Right-Sizing via Estimation and Container Migration in Apache Mesos. In Proceedings of the 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, Switzerland, 17–20 December 2018; pp. 59–64. [CrossRef]
42. Bass, L.; Clements, P.; Kazman, R. *Software Architecture in Practice*, 3rd ed.; Addison-Wesley Professional: Upper Saddle River, NJ, USA, 2012.
43. Chang, W.L.; Boyd, D.; Levin, O.; NIST Big Data Public Working Group. *NIST Big Data Interoperability Framework*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2019. [CrossRef]
44. Maier, M. Towards a Big Data Reference Architecture. Master's Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2013. Available online: https://pure.tue.nl/ws/files/46951182/761622-1.pdf (accessed on 7 December 2020).
45. Heilig, L.; Voß, S. Managing Cloud-Based Big Data Platforms: A Reference Architecture and Cost Perspective. In *Big Data Management*; Márquez, F.P.G., Lev, B., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 29–45. [CrossRef]
46. Geerdink, B. A reference architecture for big data solutions introducing a model to perform predictive analytics using big data technology. In Proceedings of the 8th International Conference for Internet Technology and Secured Transactions (ICITST-2013), London, UK, 9–12 December 2013; pp. 71–76. [CrossRef]
47. Pääkkönen, P.; Pakkala, D. Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. *Big Data Res.* **2015**, *2*, 166–186. [CrossRef]

48. Belli, L.; Cirani, S.; Davoli, L.; Melegari, L.; Mónton, M.; Picone, M. An Open-Source Cloud Architecture for Big Stream IoT Applications. In *Interoperability and Open-Source Solutions for the Internet of Things, Proceedings of the International Workshop, FP7 OpenIoT Project, Held in Conjunction with SoftCOM 2014, Split, Croatia, 18 September 2014*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 73–88. [CrossRef]
49. Pellegrini, R.; Rottmann, P.; Strieder, G. Preventing vendor lock-ins via an interoperable multi-cloud deployment approach. In Proceedings of the 2017 12th International Conference for Internet Technology and Secured Transactions (ICITST), Cambridge, UK, 11–14 December 2017; pp. 382–387. [CrossRef]
50. Scolati, R.; Fronza, I.; El Ioini, N.; Samir, A.; Pahl, C. A Containerized Big Data Streaming Architecture for Edge Cloud Computing on Clustered Single-board Devices. In Proceedings of the 9th International Conference on Cloud Computing and Services Science, Heraklion, Greece, 2–4 May 2019; pp. 68–80. [CrossRef]
51. Moreno, J.; Serrano, M.A.; Fernández-Medina, E.; Fernández, E.B. Towards a Security Reference Architecture for Big Data. In Proceedings of the 20th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data Co-Located with 10th EDBT/ICDT Joint Conference (EDBT/ICDT 2018), Vienna, Austria, 26–29 March 2018; Volume 2062. Available online: http://ceur-ws.org/Vol-2062/paper04.pdf (accessed on 6 August 2020).
52. Chen, H.; Wen, J.; Pedrycz, W.; Wu, G. Big Data Processing Workflows Oriented Real-Time Scheduling Algorithm using Task-Duplication in Geo-Distributed Clouds. *IEEE Trans. Big Data* **2020**, *6*, 131–144. [CrossRef]
53. Verbitskiy, I.; Thamsen, L.; Kao, O. When to Use a Distributed Dataflow Engine: Evaluating the Performance of Apache Flink. In Proceedings of the 2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France, 18–21 July 2016; pp. 698–705. [CrossRef]
54. Verma, A.; Mansuri, A.H.; Jain, N. Big data management processing with Hadoop MapReduce and spark technology: A comparison. In Proceedings of the 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Indore, India, 18–19 March 2016; pp. 1–4. [CrossRef]
55. Sang, G.M.; Xu, L.; de Vrieze, P. A reference architecture for big data systems. In Proceedings of the 2016 10th International Conference on Software Knowledge, Information Management Applications (SKIMA), Chengdu, China, 15–17 December 2016; pp. 370–375. [CrossRef]
56. Kant, I. *Critique of Pure Reason*; Penguin Classics: London, UK, 1781.
57. Vergilio, T.; Ramachandran, M.; Mullier, D. Requirements Engineering for Large Scale Big Data Applications. In *Software Engineering in the Era of Cloud Computing*; Ramachandran, M., Mahmood, Z., Eds.; Springer: Berlin/Heidelberg, Germany, 2019.
58. Pattinson, C.; Kor, A.L.; Cross, R. *Measuring Data Centre Efficiency*; Leeds Beckett University: Leeds, UK, 2012.
59. Dubé, L.; Paré, G. Rigor in information systems positivist case research: Current practices, trends, and recommendations. *MIS Q.* **2003**, *27*, 597–635. [CrossRef]
60. Vergilio, T.; Ramachandran, M. PaaS-BDP—A Multi-Cloud Architectural Pattern for Big Data Processing on a Platform-as-a-Service Model. In Proceedings of the 3rd International Conference on Complexity, Future Information Systems and Risk, Funchal, Portugal, 20–21 March 2018; pp. 45–52. [CrossRef]
61. Kolajo, T.; Daramola, O.; Adebiyi, A. Big data stream analysis: A systematic literature review. *J. Big Data* **2019**, *6*, 47. [CrossRef]
62. Ramachandran, M.; Chang, V. Towards performance evaluation of cloud service providers for cloud data security. *Int. J. Inf. Manag.* **2016**, *36*, 618–625. [CrossRef]
63. Ylonen, T. SSH-Secure Login Connections over the Internet. In Proceedings of the 6th USENIX Security Symposium, Focusing on Applications of Cryptography, San Jose, CA, USA, 22–25 July 1996; Available online: https://ci.nii.ac.jp/naid/10019459981/ (accessed on 10 April 2019).
64. Weave Cloud: Kubernetes Automation for Developers. Weave Cloud. 2019. Available online: https://www.weave.works/product/cloud/ (accessed on 10 April 2019).
65. Hiroishi, Y.; Fukuda, K.; Tagawa, I.; Iwasaki, H.; Takenoiri, S.; Tanaka, H.; Mutoh, H.; Yoshikawa, N. Future Options for HDD Storage. *IEEE Trans. Magn.* **2009**, *45*, 3816–3822. [CrossRef]
66. 'State Backends'. Apache Flink 1.3 Documentation. 2017. Available online: https://ci.apache.org/projects/flink/flink-docs-release-1.3/ops/state_backends.html (accessed on 10 April 2019).
67. Myers, T.; Schonning, N.; King, J.; Stephenson, A.; Gries, W. Data redundancy in Azure Storage. Azure Storage Redundancy. 18 January 2019. Available online: https://docs.microsoft.com/en-us/azure/storage/common/storage-redundancy (accessed on 25 March 2019).
68. Verbitski, A.; Gupta, A.; Saha, D.; Brahmadesam, M.; Gupta, K.; Mittal, R.; Krishnamurthy, S.; Maurice, S.; Kharatishvili, T.; Bao, X. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In Proceedings of the 2017 ACM International Conference on Management of Data, Chicago, IL, USA, 14–19 May 2017; pp. 1041–1052. [CrossRef]
69. Paz, J.R.G. Introduction to Azure Cosmos DB. In *Microsoft Azure Cosmos DB Revealed: A Multi-Model Database Designed for the Cloud*; Paz, J.R.G., Ed.; Apress: Berkeley, CA, USA, 2018; pp. 1–23. [CrossRef]
70. 'Datastore–NoSQL Schemaless Database'. Datastore. 2019. Available online: https://cloud.google.com/datastore/ (accessed on 8 June 2019).

71. Yasrab, R.; Gu, N. Multi-cloud PaaS Architecture (MCPA): A Solution to Cloud Lock-in. In Proceedings of the 2016 3rd International Conference on Information Science and Control Engineering (ICISCE), Beijing, China, 8–10 July 2016; pp. 473–477. [CrossRef]
72. Ranjan, R. The Cloud Interoperability Challenge. *IEEE Cloud Comput.* **2014**, *1*, 20–24. [CrossRef]
73. Poulton, N. *Docker Deep Dive*; Independently Published: Chicago, IL, USA, 2017.
74. Walli, S. Demystifying the Open Container Initiative (OCI) Specifications. Docker Blog. 19 July 2017. Available online: https://www.docker.com/blog/demystifying-open-container-initiative-oci-specifications/ (accessed on 8 May 2020).
75. Carter, E. Sysdig 2019 Container Usage Report. Sysdig. 29 October 2019. Available online: https://sysdig.com/blog/sysdig-2019-container-usage-report/ (accessed on 8 May 2020).
76. Combe, T.; Martin, A.; Pietro, R.D. To Docker or Not to Docker: A Security Perspective. *IEEE Cloud Comput.* **2016**, *3*, 54–62. [CrossRef]
77. Petazzoni, J. GitHub—Software-Defined Networking Tools for LXC (LinuX Containers). 22 August 2017. Available online: https://github.com/jpetazzo/pipework (accessed on 27 March 2019).
78. Yakubovich, E.; Denham, T. 'Flannel'. CoreOS. March 2019. Available online: https://github.com/coreos/flannel (accessed on 27 March 2019).
79. Open Virtual Networking with Docker. Open vSwitch Documentation. 2016. Available online: http://docs.openvswitch.org/en/latest/howto/docker/ (accessed on 27 March 2019).
80. How the Weave Net Docker Network Plugins Work. Weaveworks. 2019. Available online: https://www.weave.works/docs/net/latest/install/plugin/plugin-how-it-works/ (accessed on 27 March 2019).
81. Dua, R.; Kohli, V.; Konduri, S.K. *Learning Docker Networking*; Packt Publishing: Birmingham, UK, 2016.
82. Vergilio, T. Multi-Cloud Big Data Processing with Flink, Docker Swarm and Weave Plugin. Weaveworks. 1 August 2018. Available online: https://www.weave.works/blog/multi-cloud-big-data-processing-with-flink-docker-swarm-and-weave-plugin (accessed on 15 February 2018).
83. Zismer, A. *Performance of Docker Overlay Networks*; University of Amsterdam: Amsterdam, The Netherlands, 2016.
84. Production-Grade Container Orchestration. Kubernetes. 2019. Available online: https://kubernetes.io/ (accessed on 27 March 2019).
85. Syed, H.J.; Gani, A.; Nasaruddin, F.H.; Naveed, A.; Ahmed, A.I.A.; Khan, M.K. CloudProcMon: A Non-Intrusive Cloud Monitoring Framework. *IEEE Access* **2018**, *6*, 44591–44606. [CrossRef]
86. Zacheilas, N.; Maroulis, S.; Priovolos, T.; Kalogeraki, V.; Gunopulos, D. Dione: A Framework for Automatic Profiling and Tuning Big Data Applications. In Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 16–19 April 2018; pp. 1637–1640. [CrossRef]
87. 'Metrics'. Apache Flink 1.11 Documentation 2020. Available online: https://ci.apache.org/projects/flink/flink-docs-stable/monitoring/metrics.html (accessed on 27 June 2020).
88. Luzzardi, A. Announcing Swarm 1.0: Production-Ready Clustering at Any Scale. Docker Blog. 3 November 2015. Available online: https://blog.docker.com/2015/11/swarm-1-0/ (accessed on 8 September 2017).
89. Confluent Control Center. Confluent Platform. 27 June 2020. Available online: https://docs.confluent.io/current/control-center/index.html (accessed on 27 June 2020).
90. What Is Cloud Pub/Sub? Cloud Pub/Sub Documentation. 15 March 2019. Available online: https://cloud.google.com/pubsub/docs/overview (accessed on 28 March 2019).
91. Amazon Kinesis Data Streams. 2020. Available online: https://aws.amazon.com/kinesis/data-streams/ (accessed on 10 July 2020).
92. Apache Kafka. The Apache Software Foundation. 2019. Available online: https://github.com/apache/kafka (accessed on 28 March 2019).
93. Vergilio, T. Data-Interpolator. 31 May 2018. Available online: https://bitbucket.org/vergil01/data-interpolator/src/master/ (accessed on 28 June 2020).
94. Vergilio, T. Energy-Consumption-Producer'. 21 September 2018. Available online: https://bitbucket.org/vergil01/energy-consumption-producer/src/master/ (accessed on 28 June 2020).
95. Vergilio, T. Energy-Consumption-Simulator'. 5 October 2018. Available online: https://bitbucket.org/vergil01/energy-consumption-simulator/src/master/ (accessed on 28 June 2020).
96. Apache Beam Capability Matrix. 2017. Available online: https://beam.apache.org/documentation/runners/capability-matrix/ (accessed on 9 August 2017).
97. Heitlager, I.; Kuipers, T.; Visser, J. A Practical Model for Measuring Maintainability. In Proceedings of the 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007), Lisbon, Portugal, 12–14 September 2007; pp. 30–39. [CrossRef]
98. Bridgmon, K.D.; Martin, W.E. *Quantitative and Statistical Research Methods: From Hypothesis to Results: 42*, 1st ed.; Jossey-Bass: San Francisco, CA, USA, 2012.
99. 'Pods—Kubernetes'. Kubernetes. 12 May 2019. Available online: https://kubernetes.io/docs/concepts/workloads/pods/pod/#motivation-for-pods (accessed on 8 June 2019).
100. Karabek, M.R.; Kleinert, J.; Pohl, A. Cloud Services for SMEs—Evolution or Revolution? *Bus. Innov.* **2011**, *2*, 26–33. [CrossRef]

101. Hamburg, I.; Marian, M. Learning as a Service—A Cloud-based Approach for SMEs. In Proceedings of the the SERVICE COMPUTATION 2012, The Fourth International Conferences on Advanced Service Computing, Nice, France, 22–27 July 2012; pp. 53–57. Available online: https://www.thinkmind.org/index.php?view=article&articleid=service_computation_2012_3_30_10065 (accessed on 31 July 2020).
102. Oyekola, O.; Xu, L. Selecting SaaS CRM Solution for SMEs. In Proceedings of the ICIST 2020: 10th International Conference on Information Systems and Technologies, Lecce, Italy, 4–5 June 2020; Available online: http://eprints.bournemouth.ac.uk/33047/ (accessed on 31 July 2020).
103. Martino, B.D. Applications Portability and Services Interoperability among Multiple Clouds. *IEEE Cloud Comput.* **2014**, *1*, 74–77. [CrossRef]
104. Finta, G. Mitigating the Effects of Vendor Lock-in in Edge Cloud Environments with Open-Source Technologies. October 2019. Available online: https://aaltodoc.aalto.fi:443/handle/123456789/40884 (accessed on 31 July 2020).
105. Cammert, M.; Kramer, J.; Seeger, B.; Vaupel, S. A Cost-Based Approach to Adaptive Resource Management in Data Stream Systems. *IEEE Trans. Knowl. Data Eng.* **2008**, *20*, 230–245. [CrossRef]
106. Li, P.; Guo, S.; Yu, S.; Zhuang, W. Cross-Cloud MapReduce for Big Data. *IEEE Trans. Cloud Comput.* **2020**, *8*, 375–386. [CrossRef]
107. Koševeva, N.; Ropaitė, G. Big data in building energy efficiency: Understanding of big data and main challenges. *Procedia Eng.* **2017**, *172*, 544–549. [CrossRef]
108. Akidau, T.; Bradshaw, R.; Chambers, C.; Chernyak, S.; Fern, R.J.; Lax, R.; Mcveety, S.; Mills, D.; Perry, F.; Schmidt, E.; et al. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proc. VLDB Endow.* **2015**, *8*, 1792–1803. [CrossRef]
109. Jha, D.N.; Garg, S.; Jayaraman, P.P.; Buyya, R.; Li, Z.; Ranjan, R. A Holistic Evaluation of Docker Containers for Interfering Microservices. In Proceedings of the 2018 IEEE International Conference on Services Computing (SCC), San Francisco, CA, USA, 2–7 July 2018; pp. 33–40. [CrossRef]
110. Zhao, D.; Mohamed, M.; Ludwig, H. Locality-Aware Scheduling for Containers in Cloud Computing. *IEEE Trans. Cloud Comput.* **2020**, *8*, 635–646. [CrossRef]
111. Goedtel, M.; Swathi, D.; Bradley, M.; Wren, B.; Wallace, G. Collect and Analyze Performance Counters in Azure Monitor. Azure Monitor Documentation. 28 November 2018. Available online: https://docs.microsoft.com/en-us/azure/azure-monitor/platform/data-sources-performance-counters (accessed on 17 May 2019).
112. Prometheus—Monitoring System & Time Series Database. Prometheus. 2019. Available online: https://prometheus.io/ (accessed on 17 May 2019).