Review of Knowledge Management in Optical Networks, Lambda Architecture using Database Technologies in Cloud Settings

Abdul Joseph Fofanah[1], Saidu Koroma[1], Issa Fofana[2]

[1]Milton Margai College of Education and Technology now "Milton Margai Technical University",
Department of Mathematics and Computer Science,
Goderich Road, Freetown, Sierra Leone
abduljoseph.fofanah@gmail.com

[2]School of Technology, Njala University,
Department of Physics and Computer Science
Issa_fofana@hotmail.com

DOI: 10.29322/IJSRP.11.08.2021.p11663
http://dx.doi.org/10.29322/IJSRP.11.08.2021.p11663

Abstract: In this paper, we present concepts, theories, and overview of knowledge management in an autonomous optical networks and Lamba Architecture in cloud related environment. This study presents some illustrative cases that has been used to illustrate the potential application of KM architecture and to evaluate the various policies for the knowledge sharing and integration algorithm. Here the knowledge is used at the optical transponder system level, sharing, and integration implemented at the node level and supervising and data analytics (SDA) controller level. The KM process has been evaluated by integrating on a metro-network situation in terms of model error convergence time and the data shared among agents. Indeed, the propagation and reinforcement actions illustrated by similar convergence time than data-based policies at various phases of the network learning process without compromising the convergence accuracy of the model prediction. The Lambda Architecture is the new model for Big Data and database research focus, that helps in data processing with a balance on throughput, latency, and fault-tolerance. To provide a complete solution and better accuracy, low latency, and high throughput, there exists no single tool. This introduced the idea to use a set of tools and methods to build a comprehensive Big Data approach. Although this paper does not provide a developed and working tool, however, provides an outline and the methods used by researchers to overcome some of the shortcomings of Lambda Architecture. The Lambda Architecture defines a set of layers to fit in a set of tools and methods rightly for constructing a comprehensive Big Data scheme: *Speed Layer*, *Serving Layer*, *Batch Layer*. Each layer satisfies a set of features and builds upon the functionality delivered by the layers beneath it. The *Batch Layer* is the place where the master dataset is warehoused, which is an unchangeable and add-only set of raw data. Also, the batch layer computes before the results using a distributed processing system like Hadoop, Apache Spark that can manage large amounts of data. The *Speed Layer* encapsulates new data coming in real-time and processes it. The *Serving Layer* comprises a parallel processing query steam engine, that takes results from both Batch and Speed Layers and responds to questions and requests in real-time with low latency. Stack Overflow is a Question-and-Answer forum with an enormous user community, millions of posts with rapid growth over the years. This paper demonstrates the Lambda Architecture by constructing a data pipeline, to add a new "Recommended Questions" section in the *Stack Overflow* user profile and update the questions suggested in real-time. Additionally, various indicators such as trending tags, user performance numbers such as $UpVotes, DownVotes$ are shown in user dashboard by querying through batch processing layer. Finally, this paper provides a seamless search of the various methods or techniques used to help solve complex databases which are provided by *Stack Overflow* platform infrastructure.

Index Terms- Knowledge management, networks, lambda architecture, database technologies

I. Introduction

1.1 *Knowledge Management*

The advancement and implementation of autonomous control loops encapsulate knowledge utilization and decision-making mechanism in enormous, related works in knowledge management (KM). The predictive autonomous transmission agent (ATA) based on artificial neural networks (ANN) that forecasts the right forward error correction (FEC) algorithm design for limited operation as a final real-time supervising of state of polarization (SOP) traces and in relation to pre-forward error correction and bit error rate [1]. Other authors in [2] conducted a study in exploration of various machine learning (ML) techniques based on decision tree and support vector machine (SVM) for fault management [3], ideally for soft-failure detection, identification, and localization taking the merit of optical spectrum analyzers (OSA) to supervise the optical spectrum. Due to its nature as a distributed system, knowledge utilization is placed at the device or machine level and decision-making process that is located close to the centralized software-defined networking (SDN) controller. According to [4], the concept of autonomous networking is disaggregated cases through use cases for the provisioning and self-tuning supervision of optical spectrum. The control loop involves a collection of supervised data from one device and tuning the design or configuration of the next one, therefore knowledge utilization and decision-making process need to be located in some centralized element.

For some considerable reasons knowledge utilization and decision making are required not only at the supervised level, but also at local subsystems level, variety of use cases and situation of autonomous networking for the control plane that should be designed to support such variety. Supervising and

data analytics (SDA) system and current operators use cases searching at automated optical network operation to enable the benefit of adding SDA in their system was presented by [5]. The effect of centralization to alternative hierarchical ones that enables it to implement control loops at various level, with numerous overviewed of the SDA architectures. Furthermore, [6] presents more concrete descriptions in regard to SDA architectures and their integration with some elements in the control and data plans.

In order to visualize the significance of artificial intelligence and machine learning enough real data (big data) and to produce accurate ML models' predictions is rarely available owing glut of factors, such as the current legal and regulatory context that bound the readiness of real network performance matrix, including the challenges to obtain training datasets controlled by specific institutions (commercial technologies) and utilize them in existing and predictive scenarios. In this regard, the author in [7] propounded a learning life cycle model to enhance ML deployment in real operator networks. Specifically, they included an ML training phase to be carried out after detecting "model imprecision" or predictive errors, and consequently, provide the ML models implemented and installed in the network. The development of such a model can be made quicker in regard to the model being utilized by various agents, and subsequently, share the model's imprecision among them known as "collective self-learning". The study shows that collective self-learning outperforms individualized approaches. Nevertheless, due to the size of the training data probably be large to arrive at high accuracy and performance (robustness), collective self-learning increases data to be saved and exchanged among agents of the model.

1.2 *Lambda Architecture and Database Technologies*

Computing arbitrary functions on an arbitrary dataset in real-time is not a simple problem. There is no single tool that provides a complete solution. Instead, we have to use a variety of tools and techniques to build a complete Big Data system. Lambda Architecture is a data-processing architecture designed to handle massive quantities of data by taking advantage of both batch-processing and stream-processing methods. This approach to architecture attempts to balance latency, throughput, and fault-tolerance by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data. There has been enormous effort put by researchers in handling Big Data. The introduction of the Lambda Architecture model for stream processing and its related processing engines has enabled LA a hot issue for processing large data over the cloud systems [8]. Basically, Lambda Architectures are designed to handle vast data in aggregation with batch and stream processing techniques. Batch processing helps to reduce latency, to improve data transfer, to provide fault tolerance, and as well providing a comprehensive and accurate view of the data, on the other hand, stream processing provides capabilities to deal with real-time data. This accounts for the fact that Las are directly related to the rapid growth of Big Data real-time analytics. However, this concept of LA will be explained further to provide a clear understanding of Lambda Architecture.

The rise of Lambda Architecture is correlated with the growth of Big Data, real-time analytics, and the drive to mitigate the latencies of MapReduce. Because the Lambda Architecture is based on functions of all data, it generalizes to all applications including financial management systems, social analytics, scientific applications, social networking, etc. it achieves scalability by adding new machines in every layer, which is called horizontal scaling. The main idea of Lambda Architecture is to build Big Data systems as a series of three main layers as shown in Figure 1.

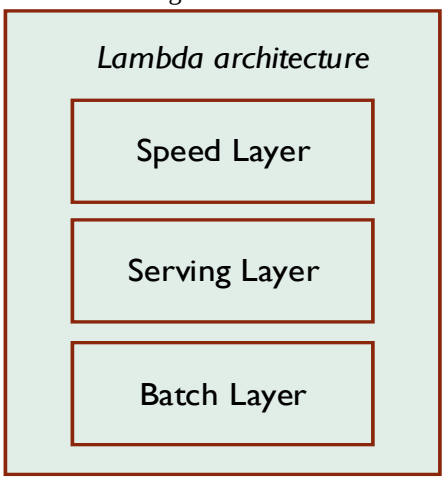


Figure 1: The three layers of Lambda Architecture

Each layer in Lambda Architecture satisfies a subset of the properties and builds upon the functionality provided by the layers beneath it. Each layer requires a lot of work to design, implement, and deploy, but the high-level ideas of the whole system are easy to understand. Starting everything from a "query = function (all data)" equation, you cloud ideally run the functions on the fly to get the results. However, this would take a huge amount of resources to do and would be unreasonably expensive. This is similar to have to read a petabyte every time you wanted to answer the query of someone's current location. The most obvious alternative approach is to precompute the query function, which is called the batch view. Instead of computing the query on the fly, we read the results from the precomputed view. The precomputed view is indexed so that it can be accessed with random reads:

$$Batch\ view = function\ (all\ data)$$
$$Query = function\ (batch\ view)$$

This system works as follows:

- Run a function on all the data to get the batch view
- When we want to know the value for a query, run a function on that batch view
- The batch view makes it possible to get the values you need from it very quickly, without having to scan everything in it.

The ***Batch Layer*** part of the Lambda architecture is responsible for creating the batch view. The master data set is stored in the batch layer and creates precompute views on the master data set. The two tasks of the batch layer are: to store an immutable growing master dataset and compute arbitrary functions on the data set to create batch views. This type of computation is done by batch processing systems like Hadoop, Apache Spark.

The ***Serving Layer*** loads the batch views that are produced by the batch layer as a result of its functions. The Serving Layer is a specialized distributed database that loads batch views and provides random access to the batch view through queries. The database should be able to update batch views coming from the batch layer periodically and allow random reads. Random writes are not important for the serving layer databases, reducing the most complex part of the database.

Thus, these databases are simply making them easy to operate, configure, and robust. The ElephantDB is an example of a database that can be used in the serving layer.

The ***Speed Layer*** is responsible for handling low dormancy updates, which arises due to high latency batch view creation in the batch layer. The serving layer updates when the batch layer completes computing a batch view, which means it misses processing the data that arrived during batch view creation. The Speed layer processes this missing data and ensures that new data is represented in query functions as quickly as needed based on the application need.

The Speed Layer is similar to the batch layer in that it computes views based on the data it receives. The only difference being the speed layer just considers recent input data for processing, whereas the batch layer takes all the data. As it receives the data, it updates the real-time views and does perform an incremental computation. We can generalize the speed layer computation into:

$$realtimeview = function(realtimeview, new\ data)$$

The Lambda Architecture in total can be generalized into following:

$$batch\ view = function(all\ data)$$
$$realtimeview = function(realtimeview, new\ data)$$
$$query = function(batchview.realtimeview)$$

Furthermore, because of these skills have laid foundations for more effective innovative research issues in Big Data and Internet of Things (IoT) projects, with an ongoing development of massive and varied data volumes, along with the use of data exhaustive applications. In an attempt to evaluate the work done by these researchers, the scholar presents a need to find out an effective means for data management in an efficient and cost-effective ways. Predicting a growth of $75 billion for small and medium-sized businesses using Clouds for data management applications, SAP industries argue lower costs, less installation needs, and ease of management of less IT resources as an attractive business model [9]. However, this scientific and technological innovations, comes with increasing challenges such as network availability, security and reliability as biggest concerns for businesses and other institutions across the world.

II. PART II

This part captures the concepts and techniques in knowledge management systems with essential ideas related to autonomous optical networks (architecture) and Lambda architecture in database technologies. Key concepts discussed in this part includes knowledge integration, model ensemble, model merge, and training data amalgamation.

2.1 Knowledge Management Process Explained

In relation to knowledge management in optical networks, Figure 1 describes the main architecture proposed by [10] to enhance KM, where two software agents in charge of networking machines are represented. Thus, the agents collect supervised data/supervising data from the underlying machines or device such as optical transponder (indicated in Figure 2a) that are inspired by an ML-based application with the idea to predict some output based on some ML predictive models in regard to some device or set of entities (example optical connection). The concluding the output can be utilized by decision-maker modules or segments (B) to tune configuration or design parameters in the machines (C) in their analysis the description of the typical control loop (A-B-C), focuses entirely on “knowledge utilization”.

The first assumption is the output produced by the ML-based application on the matrix data is saved (D) and the output could contrast with real data unrushed from the devices after some time. For some reasons of possibility, we could conceive an algorithm that would be possible, subsequently conceive an algorithm that would supervise the accuracy of the existing ML models and perceive events for which the models return imprecision output (E). Consequently, detecting such imprecisions would overt the possibility to increase our training data with new labeled data (where X is the input data and Y is the predicted response, in our case) and employ the ML training to produce more accurate ML models that can be automatically utilized by the ML-based representation (F). Finally, this loop (D-E-F) involves “knowledge acquiring” and it is based on self-learning [11]. The normalized data received from other agents can be utilized to complement the local training data and subsequently increases the learning speed since the probability of rare events to be understood increases as are more viewers.

Secondly, envisioning now that the knowledge acquiring process is accomplished individually per every various machine or entity, as the measures data could be specific for such device and the subsequent ML models. Hence, knowledge acquiring from one machine cannot be shared among various machines or devices. Nevertheless, by the assumption that the measured data can be utilized unchanged by other devices/entities or there exists a function that regularizes data can be utilized to train the ML models for other devices or entities. Next, the new “Knowledge” in the form of labeled data can be jointly shared with other agents immediately it is discovered or acquired (G), subsequently enabling the “collective learning” [11].

However, the labeled data requires an interface conversation of large volumes (big-data) unless the accuracy of the ML predictive models cannot achieve its high performance (values) due to sharing knowledge knowledge-based. Note that one single labeled data point involves a tuple of big data (values) and that complete training data can entail a large number of data points. The rationale to reduce the amount of data being swapped is to create specific models for the purpose of knowledge acquiring. These selected models can be uniquely accurate in a particular region of the features space where the new knowledge acquiring is achieved. The schematic illustrations in Figure 1b indicate that the components related to KM are the agent receiving the new knowledge; this is because the distinction between the agent receiving the new knowledge and the one identifying it is evaluated for illustrative reasons, thus, there is no restraint about being the same agent. According to [11], the proposed architecture encapsulates the following as parameters to KM in the context of optical networks (architecture, techniques, and use cases) as illustrated in Figure 3:

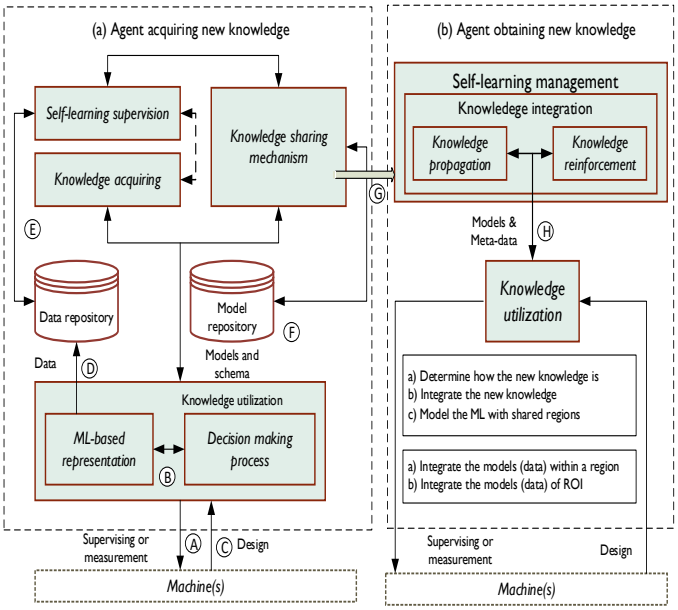


Figure 2: The KM process: acquiring new knowledge (a) and new knowledge obtained (b)

1. The architecture for KM involves accumulated information about the agent representation in relation to; knowledge acquiring, and knowledge integration (called self-learning), knowledge sharing, and knowledge utilization modules are comprehensive. Additionally, the knowledge manager module synchronizes KM functions.
2. The "*application manager*" that utilizes knowledge for the autonomous control of the machine or device is processed from the underlying physical device(s) through the data that is collected. The design of the devices is based on a set of algorithms for various problems was considered and generalization and generate outputs to a decision-maker module in large of discovering the optimal configuration or design for the predicted situations. However, any problem involves a set of combining various methods such as ML, statistical or mathematical models to generate its productivities (outputs). Primarily, the ideal situation of the application manager in the device control loop is to feed the different challenges of the expected inputs and to adequately synchronize the decision-maker and predict it localize performance or robustness.
3. The "*data repository*" with its operational tasks and the application manager exploration the preprocessed and labeled data including the real measurements and predictive models to be stored in the data repository. The data is analyzed by the knowledge acquiring module and entails in two folds: (i) to populate its internal training data and perform ML training to enhance new models and next stored in the model repository and (ii) to identify the flaws in the existing ML training to enhance new models. The main source of knowledge acquirement comes from real data from the function of the underlying machine(s) is referred to as the knowledge acquiring loop".
4. The negative feedback of knowledge utilization can be attributed to the task of knowledge acquiring and subsequently lead to various ML modes being stored in the repository. Hence, knowledge expansion and strengthening the focused on mining the number of models utilized for function while safeguarding its general accuracy.
5. With the adaptation of scheduling policy, for instance, every time a new ML model is developed and available with periodicity, the knowledge manager synchronizes the ML models of every problem in the knowledge utilization module to enable the algorithms to be utilized for purpose of functionality.
6. The knowledge utilization module enables a proactive part to facilitate knowledge acquiring, as the algorithm can discover that some given measured data identify into an unidentified region of the pace of the feature of their problems. In this case, the application manager notifies the knowledge manager by requesting the knowledge expansion module to demand other agents about labeled data around the measured one so as to produce a specific ML model for that unidentified region.

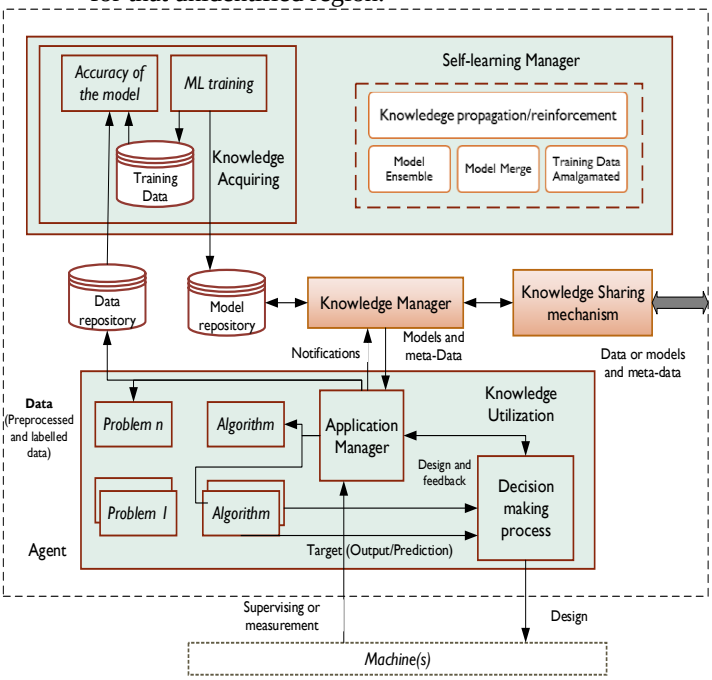


Figure 3: The architecture of an optical network KM

2.2 Knowledge Integration

Knowledge integration involves an understanding of how the new information (idea) corresponds to the existing knowledge-based and how this existing information must be modified to reflect the expert's view of the domain. As described in Figure 3, the three elementary techniques for assimilating knowledge were evaluated for the utilization of knowledge expansion and strengthening.

By assumption, the agents focus on one single problem and that they are ready to execute all types of modeling scenarios including self-supervised learning. The typology of the problems can be considered for both classification and regression ML-based applications and ideally (in our case), SVM is suitable for classification and ANN for regression analysis.

The generality loss function (f) model connects to set the input data (X) and provide a predictive model (target response Y). after transforming supervision of data into characteristics in which the response can be either a numerical value for regression purposes or classification, the input data can be supervised. The trained model of the algorithm and method that constitute the model and the required parameters, for institute ANN and all the parameters and requirements. The predictor and provisioning the context requirements to use a properly design model and to couple the meta-data. The range of each feature in the training data is characterized by the input features space region before applying a prediction to check if the input data is within the ranges observed during the training phase. The dichotomy of knowledge integration is presented in three main folds: model ensemble, model merge, and training data reinforcement:

2.2.1 Model ensemble

This technique involves a considerable number of ML models (not only single ML models) for a problem that is connected to various feasible scenarios that can be evaluated. In this situation, some models will produce precise predictions, whereas others will produce imprecise predictions. For instance, for a new situation, it is appended into the set of models utilized by the problem (Figure 3a). According to the output, an algorithm generates one single output from the predictions assemble by a set or subset of individual models in the ensemble to utilize the new model. Furthermore, the algorithm can discover how to integrate or select individual predictions or forecasts.

A weighted average of the individual responses adequately to some meta-data parameters that serve as weights can be done precisely to strategies as a prototype by the combination of individual predictive models. According to [12], the availability of supervising data enabling the dynamic evaluation of the individual predictions allows the implementation of adaptive voting procedures that can approach predictions to equal measurements. Since model ensemble is an alternative for knowledge propagation that needs low computational effort and that can be adapted to any ML method and combining various types of ML predictive models. A mathematical illustration for both classification and regression application is provided in equation [1] and equation [2]: Y* is the class of the most common response by considering the weights of the models in the classification model. Therefore, Y* can be computed as:

$$Y^*(X) = \arg \underset{c \in C}{max} \left\{ \sum_{t, \in E^*(X)} w_i * (y_i == C) \right\} \quad [1]$$

In regard to regression model, the weighted average of the individual responses can be utilized as:

$$Y^*(X) = \frac{1}{W} \sum_{f_i \in E^*(X)} w_i y_i \quad [2]$$

Where w is the sum of the individual weights. The individual weight can be classified into precise and imprecise by dynamically updating the individual wights w_i utilized for amalgamation of the imprecision models and deduced with $\rho \epsilon [0,1], as$:

$$w_i = \min(\tau * w_i, w_{min}), \forall f_i \in E^*_{impr}(X), \quad [3]$$

Whereas precise models can be succeeded by increasing its weight according to parameter $\tau \geq 1$ as:

$$w_i = \min(\tau * w_i, w_{max}), \forall f_i \in E^*_{pre}(X), \quad [4]$$

The directions and cross-relationship of ρ and τ enable the design and configuration of various strategies from long-term to short-term memory design leading to quick adaptation towards good models' predictions.

2.2.2 Model Merge

The technique involves merging individual ML model to obtain one single model for utilization of the knowledge that helps to streamline its function (see Figure 4b). To integrate the individual knowledge, the amalgamation of model parameters is a significant technique. As the new model is made available, the joint parameter model can be updated by the merging process. When the model is partially updated without affecting the performance (robustness) and accuracy of the non-modified part, the technique can provide essential benefits for such specific cases. The trained models $f_i and\ f_j$ are linear SVMs, the decision hyperplanes coefficients of each model to efficiently divides the feature space region into two various response classes and can be obtained from the support vectors V_i and V_j[16]. The combined model of the features of the individual models is given by:

$$B^* = \left[\frac{B_i^K * g(s_i) + B_j^K * g(s_j)}{g(s_i) + g(s_j)}, \forall K = 0 \ldots . m \right] \quad [5]$$

Where B_{ij} is the vector of linear coefficients (every feature and the intercept), $S_i and\ S_j$ the training data size of the model? By solving equation [5] and assuming $g(s_i) = g(s_j)$, the combined decision hyperplane is represented in Figure 5c, where the original margins and support vectors are presented. The combined decision hyperplane remains within the margin hyperplanes of the individual models along the conforming feature spaces and consequently lead to the efficiency of the model. In order to validate the model, we verify that combined decision hyperplane and original margins is disjoints at the intersect in the regions. Thus, the model cannot be rousted with adequate goodness of its assurance.

2.2.3 Training data amalgamation

Here, to obtain a synthetic training dataset from which a new ML model is trained, the individual ML models in the given regions need a technique to be generated the response. Reducing the available amount of data being exchanged from data reintegration from ML models is required to be enabled. The problem and the methods for modeling can be considered as the integrated data generation procedure for data generation to guarantee the persistence of the features of the observed data. It could be noted that some of the shared models and or part of the integrated data need to be kept for future training sequences.

These alternative models can be applied to both regression and classification problems. Specifically, classification with the utilization of SVMs needs to be guaranteed that integrate samples that are not generated inside the space defined by the margin hyperplanes. The integration support vectors with data reintegration needed to be restricted to generate samples on the margin hyperplanes. The study conducted by [10] illustrates two linear SVMs that cannot be merged due to the intersection of the combined decision hyperplane with one of the margins.

Upon applying the reintegration technique, several integration prototypes on the margins of every model are initially created to develop a transparent marker that would train new SVM. Their research concluded that the SVM training algorithm finds the best SVM configuration, with an emphasis on the most proper kernel. This can be automated by training with various kernels and returning the most accurate model. A polynomial kernel has been selected for the amalgamated model to enhance different kinds of classes of which the prototype of the integrated data generated becomes the support vectors of the amalgamated model.

However, in the regression model approach, the integration of the data points is performed by generating random samples that fit the statistical features of the input region of the features space of every novel model. This technique corresponds to the model proposed by Montecarlo [13], of which the conforming models are utilized to create the response to label the prototype. Consequently, and following the relevance of the data sample has been generated for the overall model, the amalgamated model is trained as purported using the ANN for the regression model approach and other corresponding techniques of the ML algorithms. However, every technique adopted in this paper for knowledge integration has both advantages and disadvantages, and thus enables the technique to fit better in some developed use cases than on other models which are beyond the scope of this paper under review.

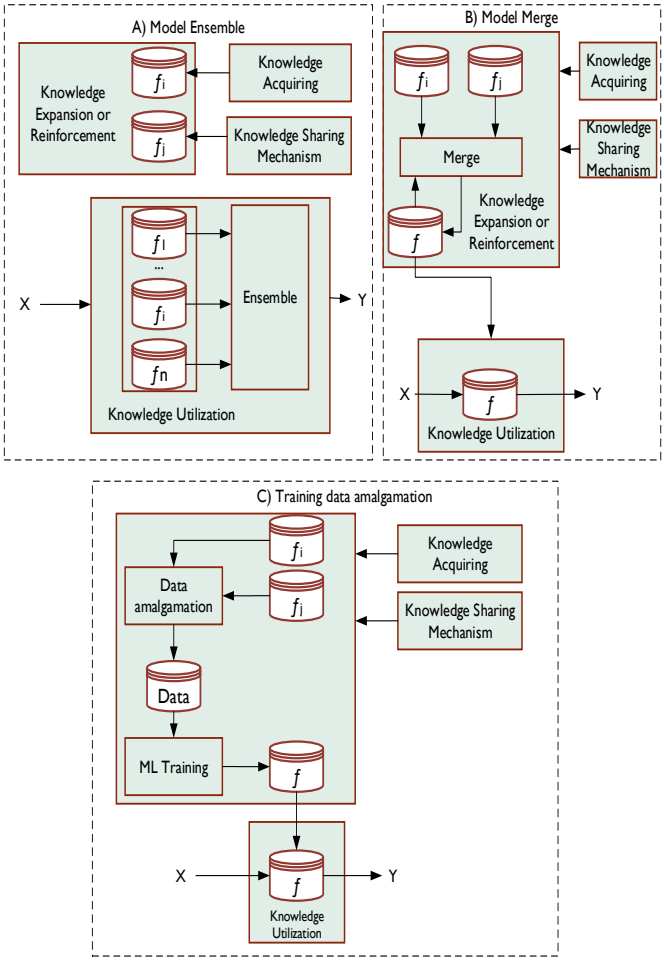


Figure 4: Knowledge integration alternatives: (A) model enable, (B) Model Merge, and (C) training data amalgamation

III. PART III

This part encapsulates the concepts and technologies of database technologies with particular emphasis on the Lambda Architecture. Detailed description of Lambda Architecture, related technologies in databases such as heterogenous infrastructure, hybrid infrastructure, and hybrid engines. Additionally, this part of the analysis includes current data processing solutions, description of Lambda Architecture, Lambda Architecture process model, and self-management of Lambda-Connections.

3.1 Lambda Architecture Explained

The Lambda Architecture model is the initial point of the currently proposing fourth (4th) Generation of Data processing Engines which include several characteristics regarding the design and implementation of processing engines for massive data, thus, fault-tolerance, robustness, low latency of reading and updating, generalization, scalability extensibility, ad-hoc queries, and minimal maintenance [14]. However, to make this happens these features, the LA foresees a Big Data system to be constructed in several layers. According to [15] they presented the SMART platform, which is a segmental framework for Big Data analysis. SMART considers a large variety of data sources, such as distributed datasets and social networks, where there is a clear need for standardization. The Dispatcher module (DM) in the SMART platform is an instrumentation system that needs several guidelines for handling data and responsibilities.

In view of this, numerous vendors are distributing services for data processing such as Amazon Web Services (AWS), Rackspace Hosting, and Google Cloud to name a few, presenting a collection of tools for gearing online data collection, cloud-hosted databases, and MapReduce processing such as using Hadoop, Hive or Spark. By offering users virtual machines to host, compute and manage their data, users can use advantages such as elasticity, multi-tenancy, and the pay-as-you-go cost model. In cloud services, where computing is delivered as a utility on a pay-as-you-go basis. Usually, business institutions are used to finance huge amounts of investment and time in the purchase and maintenance of computational resources. The advent of Cloud Computing is rapidly changing this ownership-based method to a subscription-oriented method by providing access to scalable infrastructure and services on-demand. Users can store, access, and share any amount of information in Cloud. That is, small or medium enterprises/administrations do not have to worry about acquiring, configuring, managing, and sustaining their own computing infrastructure. They can focus on improving their core capabilities by taking advantage of a number of Cloud computing benefits such as on-demand computing resources, faster and cheaper software development capabilities at low cost. Moreover, Cloud computing also offers a big amount of computing power to organizations which require the processing of a tremendous amount of data generated almost every day. For instance, financial companies have to maintain every day the dynamic information about their hundreds of clients, and genomics research has to manage huge bulks of gene sequencing data. In view of the numerous benefits cloud services provide, there have been serious challenges between the cloud service providers and the users/customers on issues related to the reliability of customer's data and virtualization technology involve during live migrations with their data.

According to [16], the U.S. physicists prepared an analysis of data from the CMS and ATLAS detectors at the Large Hadron Collider (LHC) at CERN and faced unprecedented challenges: Massive, globally distributed datasets growing over 200-petabyte level by 2025, Petaflops of distributed computing, Collaborative data analysis by global communities of thousands of scientists. The Particle Physics Data Grid (PPDG), together with the NSF-funded iVDGL and GriPhyN projects, is moving to the development of next-generation integrated Grid systems to meet these challenges and to fully exploit the LHC's potential for physics discoveries. Today, all these high-energy physics PPDG experiments' grid systems are limited by their treatment of the network as an external, passive, and largely unmanaged resource. Moreover, to date, no advanced network linking the U.S. HEP Laboratories and key universities involved in Grid and network development have been available to research and prototype solutions to these limitations [16].

Over the past several years, there has been a great deal of research effort and funding put into the deployment of optical advanced research networks, such as National Lambda Rail, CAnet4, Netherlight, UKLight, and most recently, the DOE UltraScience Net. These networks potentially have the capacity and capabilities to meet the data movement requirements of the particle physics collaborations [16]. Currently, the attention of research efforts in the advanced network area has been principal to provision, dynamically configure and control, and monitor the wide-area optical network infrastructure itself.

Accurate identification of IP flows and proper management of lambda connections are important tasks to achieve the desired move. Two approaches are currently used for that [17]: conventional management and GMPLS signaling. The former is characterized by a centralized management entity such as a human manager or an automated management process, that oversees establishing lambda-connections and deciding which IP flows should be moved to the optical level. distinctively, the latter is characterized by the fact that optical switches synchronize the creation of lambda-connections between themselves after being activated for that. The decision on which IP flows will be moved to the optical level however is taken by a centralized entity or by the entities exchanging data flows.

Both approaches, however, have some limitations. Both approaches require human interaction to detect flows and manage lambda-connections. This interaction may be slow and error prone. Currently, when a lambda-connection is requested within one single domain (intra-domain), several steps are taken (e.g., phone calls and emails exchanging) between requesters and network domain administrators in order to establish the lambda-connection. Evidently, it may take hours or more before the desired lambda- connection can be used. When the requests for a connection spans multiple domains (inter-domain), the lambda-connection provisioning may take even much longer. In addition to that, a troubleshooting process may be needed to solve connection problems, which may delay the connection setup still more. According to [18], their solution to overcome these shortcomings "consists of extending the GMPLS approach by automatically detecting IP flows eligible for lambda-connections. With this extension, multi-service optical switches automatically detect IP flows and establish/release lambda-connections for them". This can be considered self-management behaviour. In this context, the main goal of this review paper is to propose an architecture for the self-management of lambda-connections in hybrid networks.

3.2 Related Technologies

This section explains the main technologies of the Lambda Architecture and provide an overview of the technologies used to overcome massive storage media. This portion of the paper also elaborate on heterogenous infrastructure, hybrid infrastructure, hybrid engines and current data processing solutions. Lambda architecture is a data processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream-processing methods. This approach to architecture attempts to balance latency, throughput, and fault-tolerance by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data.

The two view outputs may be joined before presentation. The rise of lambda architecture is correlated with the growth of Big Data, real-time analytics, and the drive to mitigate the latencies of map-reduce. Lambda architecture depends on a data model with an append-only, immutable data source that serves as a system of record. It is intended for ingesting and processing timestamped events that are appended to existing events rather than overwriting them. State is determined from the natural time-based ordering of the data.

A. Heterogeneous Infrastructure

SMART is a platform that provides an efficient architecture for Big Data analysis applications for small and medium-sized institutions or organizations [19]. Its implementation deliberates on heterogeneous data sources and aims at data analysis situations in geo-distributed settings, considers cost, fault tolerance, network overhead, input/output throughput, as well as the minimization of data transfers between computational resources. However, these variables are not sufficient to work with volatile environments, specifically in-stream procession. The scholar of this paper thinks to overcome these environmental limitations, it is necessary to incorporate information from physical components, thus includes CPU speed, memory, and storage. Consequently, the inclusive capacity impacts the overall performance. In addition, regarding the unpredictability, replication must be encapsulated as a fault control mechanism.

Equally, Pham et al. [20] recommended a generic, extensible, scalable, fine-grained, and re-configurable multi-cloud framework. It is based on an insubstantial kernel and provides a hierarchical Domain Specific Language (DSL). The DSL allows for a fine-grained level of administration. However, the proposed solution does not control the workload at the nodes and possible faults, as the Deployment Manager is just an interface to set the devices and the Virtual Machines (VMs). The prevalent demands of users are increasing so rapidly that greater servers and disks, more powerful chips are needed to process them fast enough within the required period. Cloud computing with progressively interacting between front-end client devices and backend data centers will cause an enormous escalation of energy usage and data security, privacy, and the reliability of virtual environments during live migration on data. Improving the reliability and security of data centers while meeting service provision guarantees is the big challenge and requirement that Cloud computing faces.

B. Hybrid Infrastructure

Another model used for the execution of MapReduce on hybrid cloud and desktop grid computation environments is HybridMR, developed by Tang, He, and Fedak [21]. These researchers proposed two main innovative solutions to enable large amounts of data-intensive computation: Firstly, the hybrid distributed file system is known as HybridDES. The main feature of this technique is reliability distributed storage that alleviates the volatility of desktop PCs that is fault tolerance and file replication mechanism; and secondly, a Node priority-based fair scheduling (NPBFS) algorithm has been developed to achieve both data storage balance and job assignment balance by assigning each node a priority through quantifying CPU speed, memory size, and input and out capacity. The NPBFS approach is very motivating because it uses some miscellaneous environment parameters to schedule the tasks. Although regarding stream processing, its application is not possible due to the high flow latency. HybridMR just uses the flow rate to deploy the tasks, however, the rate does not consider particular task information. This is a very huge task to undertake especially when processing power is high in terms of market demand and the use of virtualization technology in data migrations to some other nodes. However, we thought that creating such innovation can still be feasible in minimizing the potential challenges of data storage and retrieval in the Lambda Architecture model.

C. Hybrid Engines

Hybrid Engines are one of the most complex techniques in Lambda Architecture for the computation of data storage and servers across various networks in the cloud computing environment. Originally Apache Flink was developed by Alexandrov et al. [22], enables massively parallel in-situ data analytics, using a programming model based on second-order functions. Currently, Apache Flink is the state-of-the-art processing engine according to Ewen et al. [14]. The scheduling policies are designed to work in commodities environments (i.e., clusters and cloud). Commodities strategies do not work well with dynamic environments, as argued by Peng et al. [23] and Eskandari, Huang, and Eyers [24]. It is essential for heterogeneous computing resources to acknowledge the surrounding environment. However, because the Hadoop system provides a high degree of parallelism, scalability for Big Data applications. HDFS and the MapReduce framework provide a high degree of fault tolerance through the replication of data and processes. It demonstrates the power of procedural programs once it its executed in massive parallel. Only certain types of analytic jobs can be effectively decomposed into MapReduce jobs hence the effect of the Hybrid Engine proposed in this paper clear provides a method of handling complex and massive data storage challenges and synchronization.

In major innovative research on Summingbird by Boykin et al. [25] integrates batch and online analysis with the aid of a hybrid processing model, where access can be provided efficiently and seamlessly for aggregations across of long-time spans while maintaining up-to-date values with minimal latency. However, there are shortcomings of this approach, the fact that Hummingbird does not provide access to the Message Queue writing in Hadoop, it only has knowledge of that has been recorded. The scheduling policies are abstracted, and the Hadoop and Storm systems handle the management. In many seamless decompositions, the scholar of this paper argues that the pipeline showed a processing step that can be used for optimizing bandwidth usage by our MapReduce job. Called the Combiner, this pass runs after the Mapper and before the Reducer. Usage of the Combiner is optional. If this pass is suitable for our job, instances of the Combiner class are run on every node that has run map tasks. The combiner will receive as input all data emitted by the Mapper instances on a given node. The output from the Combiner is then sent to the Reducers, instead of the output from the Mappers. The Combiner is a mini-reduce process that operates only on data generated by one machine. Figure 4 shows a clear view of the Combiner step inserted into the MapReduce data flow to help synchronized the Hybrid processing model.

3.3 Current Data Processing Solutions

According Mian, Martin, and Vazquez-Poletti [26] presented a cost-effective model for virtual machine provisioning to implement dynamic data analytic capabilities, at the same time trying to comply with all service level agreement (SLA) restraints. The paper highlighted how an optimized infrastructure would be more reliant on the provider setting up experiments and would not be defined SLAs. Dobre and Xhafa [27] presented a context-aware framework, specifically designed for handling multiple devices, mapping between components, and storing or handling requests from numerous users. As a means to support smart data processing through contexts, the authors however did not discuss how the data is moved through multiple abstraction layers to aid with speed and cost of delivery.

The most interesting and serious attention focused by the scholar of this paper, having read the availability of data storage security, privacy, virtual environment technology, and its impact on data security challenges poses a serious question about the measures of the service level agreement (SLA) whether it tends to be protected by cloud service providers. Such as where is my data hosted? Are my data not be compromised by the cloud service providers if the government changes its jurisdictions? Are the SLAs protecting both end-users and the cloud service providers? What is the relationship between the end-users and the software developers and the cloud service providers? Are there more risks in storing personal information in data centers that belong to a single entity rather than in multiple data centers? In the cloud computing environment, there appeared to be limited concerns about protecting the ownership of intellectual property when the big validity in cloud computing was to provide an omnipresent system that was distributed without reflections of authority or security, regulation, and controls. In industrialized nations that eventually provide cloud services such as Microsoft, Amazon, Yahoo, and Google to name but a few, the SLAs between them and the end-users are basically based on the jurisdictions or regulations of those countries. However, sometimes cloud service providers contradict the issues of the SLAs during the provision of cloud services to their customers. This to me, is a big challenge. For instance, with copyrights protection there appeared to be little deliberations beyond what the end-user cloud contributes to their artefacts before everything gets uploaded into the diverse cloud environments

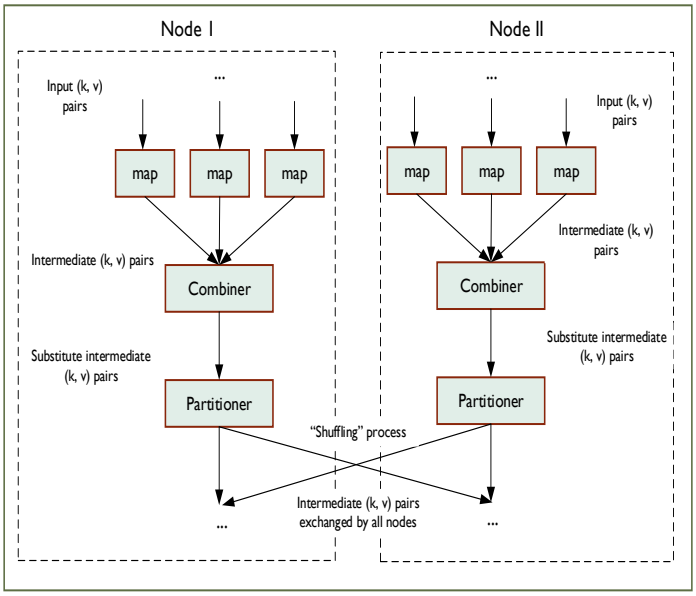


Figure 4: Combiner step inserted into the MapReduce

.

Again, since virtualization is a new technology to end-users and even computer science, little do end-users know that during virtualization most of the storage environments are put on hold hence creating security threats on end user's data without providing clarity of what really happens with the owner data to another virtual environment used by a third party. In businesses such as photographic studios and others, this is a major challenge. Their business is images and yet these images clouds be taken without permission and reused in other jurisdictions because of the reach of the cloud. Hence this seemed very sensitive to the researcher to develop a tool and techniques that will begin to resolve these unique challenges.

With the development of state-of-the-art technologies' researchers such as the M3 project proposed a disk communication layer between Reducers and Mappers to allow dynamic multi-streaming of applications and rate-based load balancing in order to achieve an efficient data processing solution (Sim et al. 2018). Similarly, project Chameleon to augment the query for fast delivery used a specific context-based indexing approach Wingerath et al [28]. Projects like Yahoo's Pig Yan et al. [29], Microsoft's SCOPE Chaiken et al. [30], and Google's initiatives (Hoover 2008) respectively in the context of the scholar's research are gearing to integrate declarative query constructs from the database community into MapReduce-like software to allow bigger data independence, code reusability, and instinctive query optimization. These projects approached the problem as a distributed model, however further work needs to explore hybrid solutions which consider resources, data models, varied queries in accordance with network traffic or cost. In their quest to solve some of these challenges also possess another serious challenge of performance and reliability of data storage security in a cloud computing environment.

3.4 Description of Lambda Architecture

In response to emerging technologies like the Lambda Architecture amalgamates online and batch processing transactions within a single framework. However, these patterns are more suitable for applications that have sufficient time delay in data collection and availability through control panels, demanding data validity for online processing as it arrives. Marz and Warren [31] presented a paper that allows the pattern for batch processing for grown-up datasets to find interactive patterns as per user needs.

Figure 5a shows the straightforward architecture of how the lambda architecture works for both batch and speed layers. It caters to three layers firstly, Batch processing for precomputing large amounts of datasets secondly, Speed or real-time computing to minimize latency by doing real-time calculations as the data arrives and thirdly a layer to respond to queries, interfacing to query and provide the results of the calculations.

Lambda architecture allows users to optimize their costs of data meting out by understanding which parts of the data need online or batch processing. According to Marz and Warren [31] the architecture also partitions datasets to allow various kinds of calculation scripts to be executed on them. However, a few critiques of the architecture have argued that the multiple set of projects that need to be maintained under the data branch to allow multiple data executions, requires more skills from the developers set up the jobs to execute and produce results. The batch layers the portion of the Lambda architecture that implements the batch "view = function (all data)" equation. The batch layer stores the master copy of the dataset and precomputes batch views on that master dataset. On the other hand, the speed layer only looks at recent data, whereas the batch layer looks at all the data at once. In the author of this paper view, in order to achieve the smallest latencies possible, the speed layer does not look at all the new data at once. Instead, it updates the real-time views as it receives new data instead of re-computing the views from scratch as the batch layer does.

Regardless of this effort, the architecture is well suitable for big data processing problems with several kinds of analysis required to understand and study the online data arriving through devices with sensors. The online stream can be used to sense data irregularities authenticating whether it is correct before processing it further. Tested data can then be archived into databases, which can have batch scripts performed once a day or a month to study data behavior over a period of time. End-Users can diminish the costs of executing these scripts on larger data sets by modularizing the problem down in manageable steps dropping down costs and tailoring the data analysis routines to suit their needs. This architecture can be adapted for collecting and analyzing online sensor data to find efficient solutions to process large datasets. An example of Lambda Architecture implemented on Amazon web services is shown in Figure 5b.

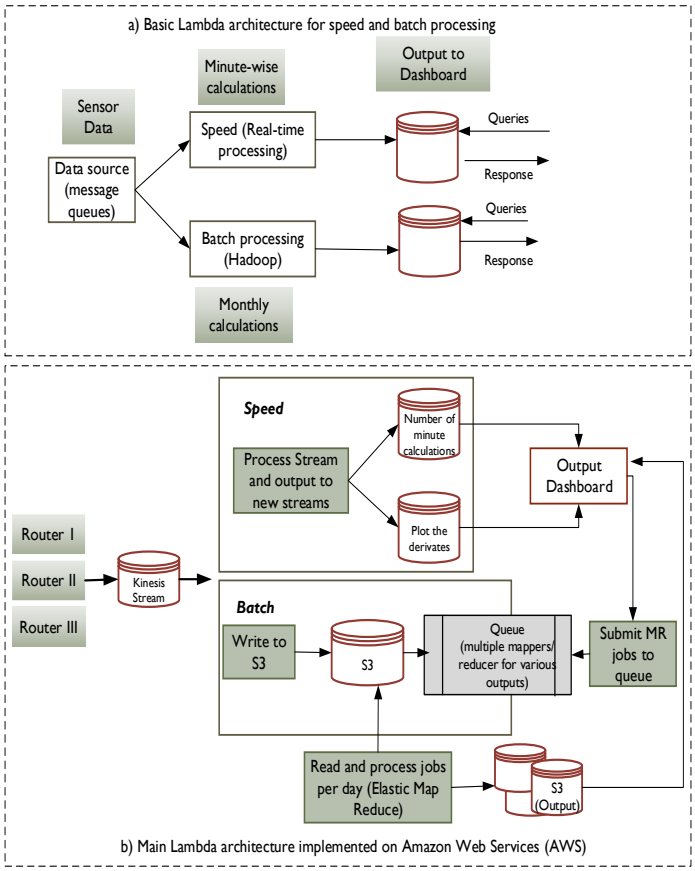


Figure 5: Basic Lambda Architecture for speed and batch processing (a) and main Lambda Architecture implemented on AWS

3.5 Lambda Architecture Process Model

Public clouds provide several services which could be employed for online and batch processing. Table 1 presents a comparison of Microsoft azure and Amazon AWS services offering similar capabilities. For the purpose of this paper and the limitation of carrying out this task, Amazon EC2 is chosen as a starting point for accessing multiple services. A comparison of the services presented in Table 1 shows that the online processing needs stream and batch processing which was easier to be performed in Amazon cloud rather than Azure services. The availability of services and cost plans for first time users of the Amazon infrastructure were also suitable for the project objectives.

Conventional computation systems are inadequate when facing Big Data for scalability, consistency, fault tolerance and low latency. However, advancement of computing has produced several architectures and computation models which are the efforts for processing Big Data. Apart from cloud computing, distributed computing and Lambda Architectures described in this paper, there are many other Big Data technologies that have been making an impact for handling Big Data such as open-source ecosystems and mobile data analysis [32]

Table 1: Comparisons of Cloud Services

Example services	**Microsoft Azure**	**Amazon web services Subhead**
Available Region	Azure Region	**AWS Global Infrastructure**
Compute Services	Virtual Machines (VMs)	**Elastic Compute Cloud (EC2)**
Storage Options	Azure Storage (Blobs, Tables, Queues, Files	**Amazon Simple Storage (S3)**
Database Options	Azure SQL Database	**Amazon Relational Database, Service (RDS) Amazon Redshift**
NoSQL Database Options	Azure DocumentDB Azure Managed Cache (Redis Cache)	**Amazon DynamoDB Amazon Elastic Cache**
Data Orchestration	Azure Data Factory	**AWS Data Pipeline**
Administration and Security	Azure Directory	Active **AWS Directory Service AWS, Identity and Access, Management (IAM)**
Analytics	Azure Analytics	Stream **Amazon Kinesis**
Other Services & Integrations	**Azure Learning None**	**Machine None AWS, Lambda, AWS Config**

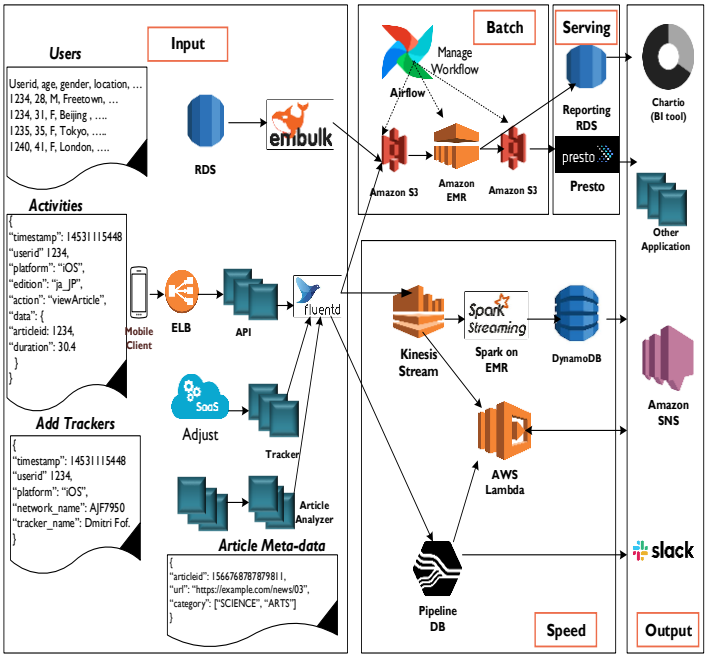


Figure 6: An illustration of batch and speed layer architecture in cloud settings

Among all of these new and vibrant technologies, Apache Hadoop and Lambda Architecture are the one technology that has been the darling of Big Data talk. Hadoop is an open-source platform for the storage and processing of diverse data

types based on distributed architecture and Lambda Architecture that enables data-driven enterprises to rapidly drive the complete value from all their data. Figure 5 shows the data service view in Lambda Architecture. The speed layer uses databases that support random reads and random writes, so they are orders of magnitude more complex than the databases in the serving layer, both in terms of implementation and operation. However, once data makes it through the batch layer into the serving layer, the corresponding results in the real-time views are no longer needed. This means we can discard pieces of the real-time view as they are no longer needed. This is a wonderful result because the speed layer is far more complex than the batch and serving layers. This property of the Lambda Architecture is called complexity isolation, meaning that complexity is pushed into layers whose results are only temporary. If anything, ever goes wrong, you can discard the state for the entire speed layer, and everything will be back to normal within a few hours (see Figure 6).

3.6 Self-Management of Lambda-Connections

According to Tiago Fioreze et al. (2015), they proposed a self-management approach of Lambda-Connections in Hybrid networks. The proposed architecture is introduced by presenting first its functional part and then it is the physical part. IP and optical domains coordinate with one another in order to detect IP flows and manage lambda-connections across various networks. Both domains are assumed as already been configured by network managers. IP routers located at IP domain B are exchanging network information (e.g., bandwidth consumed per-flow and its duration) regarding the existence of an elephant flow transiting between IP domains A and C (step 1). Based on this exchanged information and the configuration performed by network managers they make decisions on if a flow is eligible or no longer eligible for a dedicated lambda-connection at the optical level. If the decision is in favor of creating a lambda connection (i.e., the elephant flow is eligible to be moved to the optical- level), the IP routers signal the optical switches in lambda domain A (step 2). Then, the optical switches coordinate among themselves in order to create a dedicated lambda- connection to the detected elephant IP flow (step 3). From that point on, the elephant flow is switched at the optical level and IP routing is accordingly changed (see Figure 7).

However, this approach has some shortcomings in that the GMPLS signaling approach offers some autonomy in the steps of establishing and releasing lambda connections. However, these steps must still explicitly be activated by the users or network managers of the optical network. In addition to that, these users and network managers must also provide the information about which IP flows will be moved to the optical level and hence this makes the process very cumbersome to address some of the hybrid lambda architecture.

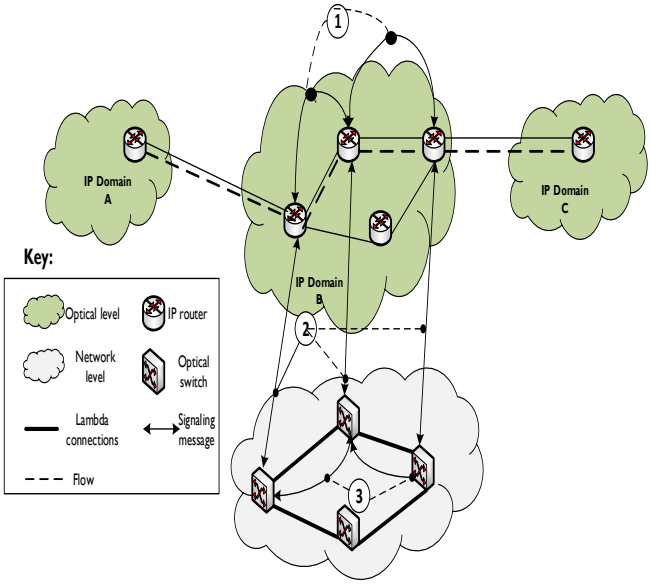


Figure 7: Self-management of Lambda-Connection in hybrid networks

IV. PART III: DISCUSSIONS AND FINDINGS

In part III, entails an analysis of both KM in optical networks and Lambda Architecture and it related technologies. The KM in optical networks key highlight includes data-based KM and model-based KM while analysis of database technologies includes an overview of SMART architecture, SMART layer architecture, Lambda architecture for batch and real-time processing, and evaluation of Lambda Architecture.

4.1 Knowledge Management in Optical Networks Architecture

a. Data-based Knowledge Management

The performance of KM can be best evaluated by data sharing mechanism. By assumptions, the imprecision are data shared when they are diagnosed or detected. If we consider two various policies for data sharing that includes imprecision and propagated data. Imprecision data points are shared particularly when we consider a small window of sample (for instance 40 prototype) to calculate the features for the imprecision in the regard to the purely distributed economic transmission. On the other hand, one prototype is required in the case of the purely centralized virtual network configuration or design (VNC/VND). This is adapted from the collective self-learning techniques presented in [33].

Secondly, the “propagated data” where the imprecision datapoints correspond with other data points that were not identified initially as imprecision and subsequently as necessarily to improve the ML models. To allow the computation of the evolution of the characteristics is distributed in the case of autonomic transmission for an extended window if another alternative is considered. In regard to VNT reconfiguration, the individual prototypes measured automatically before the imprecision are shared. The amount of additional data points that provides the best trade-off between that was analyzed. Specifically, the extended data sharing indicates a better convergence time, reaching the target error (60-70%) faster than without sharing knowledge [10].

b. Model-based Knowledge Management

Specifically. If we limit the meta-data to specify the minimum and maximum of each input feature the models' meta-data is needed to be specified their region of applicability. For instance, if selecting $\rho = 0.7, \tau = 1.7, W_{min} = 1, Max = 12$, the model merge and training data reintegrate that we can utilize the features of the ML model techniques for the purely distributed (SVM) and purely centralize (ANN) use cases, respectively to aid the performance of the model.

Evaluating the robustness of various policies and the influence of the main blocks entailed in knowledge integration, thus includes knowledge propagation and reinforcement by comparing the three basic policies: propagation, reinforcement, and propagation and reinforcement. In these three parameters, can be explained in three folds: firstly, every new shared model is added to a device models pool and utilized together with a method of ensemble model (propagation); secondly, single model by model merge or training data reintegration techniques dependent in the use case 'reinforcement'; thirdly, where both knowledge propagation and reinforcement is continuously updated to key moderated the size of the model's pool. Once the exceptional robustness of the model-based KM with the policy amalgamation knowledge propagation and reinforcement has been illustrated on its practical applicability and dependents mainly on the amount of data involved in knowledge sharing as contrast to data-based KM policies.

Congruently, the KM scheme has been discussed, analyses, comparing with other recent researchers for a truly autonomous optical network operations based on four main pillars: knowledge acquiring, knowledge sharing mechanism, knowledge integration, and knowledge utilization. Analysis shows that these pillars or parameters that enable optical networks to independently discover and propagate the knowledge that can be utilized to adjust its unique configuration and design to variable situations without human interference [10]. Ideally, a typical control loop implementation that enable knowledge sharing within various agents ignoring their distributed network nodes using a centralized controller (supervising and data analysis) with an overall architecture to enable KM was evaluated in the context of autonomous optical network. The analysis indicates that, knowledge sharing mechanism allows a collective self-sharing to reduce the model error convergence time.

The literature of this paper evaluates three techniques for knowledge integration: a model ensemble, model merge, and training data reintegration. The knowledge sharing mechanism involves data distribution storage. Data prototypes related to model imprecision and models representation (imprecision) are evaluated as the two options mechanisms for the distribution of data. Subsequently, propagation and reinforcement are the two main parameters to manage ML models in regard to the knowledge integration component and its architecture. Essentially, model ensemble enables an efficient and accurate utilization of ML model pools while model merge enables the amalgamated model and training data reintegration to reinforcement various models base on regenerating data from the utilized to train the new models.

4.2 Lambda Architecture and Database Technologies

4.2.1 SMART Architecture

This section of the paper presents an analysis of technology presented by da Sliva et al. [34] of the layered architecture that requires the deployment of the SMART-based environment. They provided four main modules of the layered architecture: *Global Collector, Global Dispatcher, Core Engine, and Global Aggregator* (see Figure 8). For the Global Dispatcher, the data was collected and serialized under standard TCP/IP protocol, which is decoupled from the lower layers in the message queue mechanism. However, since both Cloud/Multi-Cloud and Grid/Multi-Grid environments are placed in a FIFO queue so that it can be distributed to servers in agreement with the availability of their resources. The Optimization Layer studies the volume of input data and employs the *Decision Engine* to make decisions about scheduling tasks and data through distinct environments. A simulation process implements an execution time prediction that will be used by the *Decision Engine* to improve the accuracy of the scheduling mechanism. The main function of the *Global Collector* layer is to handles the management and coordination of the detecting modules. It is responsible for obtaining data from several sources and maintaining the data integrity mechanisms. The data integrity mechanisms filter possible noises in the hardware devices. The Core Engine support hybrid systems which improve computational performance by implementing Volunteer Computing (VC) in a hybrid infrastructure. SMART allows computational resources to be taken from Cloud/Multi-Cloud, and Multi- Grid environments, a *Client User API* provides an easy method for users to submit their application and indicate the data sources. The Client UI provides the security interface of a single-user identification through an encrypted key and in turn, the key which is kept by the users is employed in the *Encryption-Decryption Engine* to ensure the data is safe.

4.2.2 SMART Layer Architecture

Smart contracts characterize special algorithms for the automation of contracts including deal-making practices Sergey et al [35]. Exchange shares, money, selling realities, documents or any other proprietary are typical scenarios of smart contracts that enable people to evaluate the processes. Services such as brokers, notaries, agents, etc. that enable smart contract implementation are one of the major significant features of smart contracts with the use of intermediary services. Sergey presented three main properties of the SMART contracts namely: Autonomy, Decentralization, and Auto-sufficiency. Firstly, Autonomy implies that after a smart contract launches, the deal initiator does not have to participate anymore in the process. Additionally, smart contracts focus on the distribution of different networks nodes or points and are thus referred to as being decentralized and not necessarily focused on one central server. For instance, to allow a larger capacity of storage and computation power, and overall auto-sufficiency should enable smart contracts to collect money, distribute resources, issue, realize transactions, and spend funds. Again, smart contracts can be attributed to Blockchain technology as an ideal space for storage. Since a decentralized system does not need an intermediary to be presented at the signal deals or at the time of realization of the transactions. In the distributed registry or one-time written code, smart contracts are stored. With the emergence of computer network that supervises the blockchain, smart contract carries out their work. Consequently, if a code is written in a proper way, nobody will be able to change it. Lastly, the use of smart contracts has just started to be prevalent. The current boundaries of the development of such a type of deal signing due to some technical aspects can soon fade out thanks to such platforms as CREDITS. To write and transfer smart contracts in blockchain, the system uses a powerful encryption system

and has language-specific comprehensiveness as per Turing. Comparatively, a Bitcoin protocol with its much tighter functionality only authorizations economic operations, and it does not offer the possibility to store smart contracts or transfer assets.

We can use smart contracts in lots of everyday life situations, but its greatest potential is in the financial sector. With the help of a smart contract, the suspicion between parties and business partners is solved. For instance, if company A sells shares or any other product to company B and the parties do not trust each other, they will choose an intermediary who can help them in case of any challenging situation. If company A sends shares or the code that are encrypted through blockchain storage it is only after company B pays money, that they will get the shares. It is possible for you to write in the code as a transaction condition that the shares will be transferred to the party after the money is deposited into the account. So, first, the algorithm checks the balance and then the contract will be signed. SMART Layer architecture provides some merits in Lambda Architecture such as agent impartiality in signing contracts, mechanization in signing contracts, streamlined, security that data in the distributed registry cannot be lost by cyber-attacks, the accuracy that no errors can be made due to the nonappearance of hand-filled forms.

However, SMART Layers has some weaknesses such as the consumers are quite skeptical because it is a new technology and they do not understand it yet, creating changes. For instance, you may change your mind about renting an apartment building, but the data is previously registered, and it is technically hard to make modifications. This may bring errors into the system and make it less safe; one can keep and save data in smart contracts securely and it is a vacuum of any misrepresentations, only if the code is written perfectly and accurately.

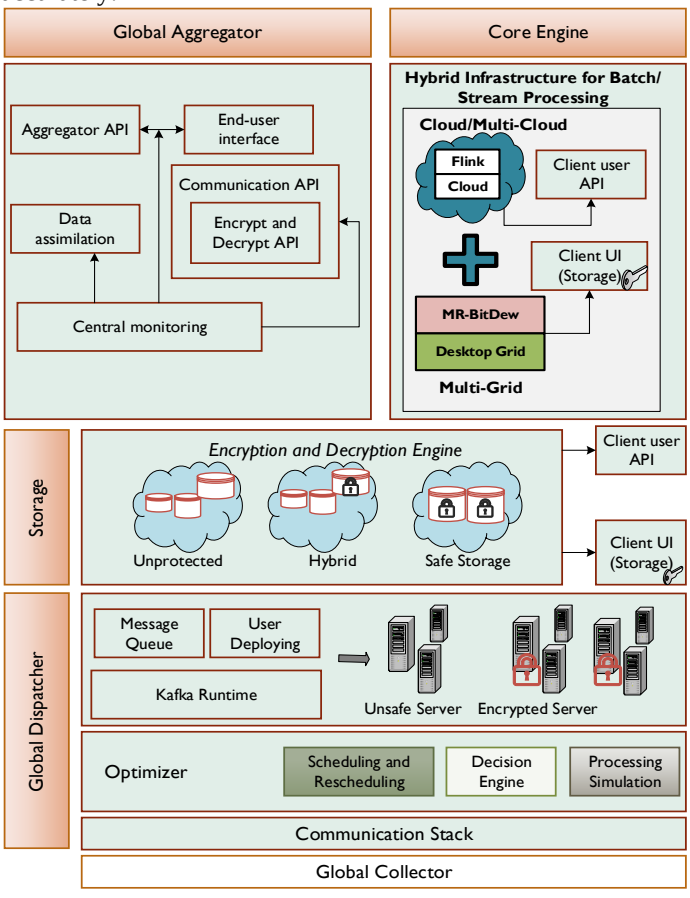


Figure 8: An illustration of SMART Architecture

Humans can be tired or make clerical mistakes and thus the whole system is threatened; The third-party agents do not dissolve but start performing a distinct role. The need for lawyers experienced in IT increases in the future because the programmers of smart contracts will need discussions for making new kinds of contracts. Thus, and inappropriately, the smart contract possibility in the financial sphere is not exposed to full working capacity. The main constraints are transaction time and cost. Also, it should be mentioned that for various industries there are not enough dedicated projects. One such project is Red Pulse, a Chinese start-up in financial consulting. This is a decentralized information platform for exchanging up-to-date info and knowledge about market trading with the RPX token bounty system.

4.2.3 Implementation of Lambda Architecture

According to Kiran M. et al. [35] Lambda architecture allows multiple data processing scripts which are tailored to specific data sets. “For online processing stream processing can be used to perform calculations as data arrives and batch processing scripts can be created to run on data stored from before. The above architecture was implemented on Amazon AWS utilizing multiple resources and producing various results in terms of cost and usage”. A number of lessons were learned while exploring the architecture on both batch and real time processing such as:

- For ***batch processing***: Performing local tests before deploying the scripts on EC2 helps to find code errors and manages in reducing the costs of failed clusters on EC2. Even if the cluster was executed for 2 minutes, Amazon cloud charges the machine as a full hour, which causes the steep increase in EMR costs Which machines are used in the cluster cause an impact on the cost and affects the time the service will take to execute the jobs?
- For ***real-time or online processing***: Firstly, the kinesis stream read data as a last-in-first-out that meant data needed to be saved locally to calculate the 5-minute data aggregations. Secondly, Amazon Cloud management allows roles and rights to be assigned to multiple members of a team. Multiple members may have rights for kinesis processing, but if the kinesis stream interacts with other services by moving data across would require the member to have rights to the other services as well. Software testing strategies such as try and catch exceptions need to be implemented in the code to prevent services to fail.

In terms of the processing time, the kinesis stream was able to process data in real-time while the EMR cluster used approximately 10 minutes to complete a job. Some of the services charge while they are active, and thus should only be dynamically started and stopped when being used in order to optimize costs.

4.2.4 Evaluation of Lambda Architecture

According to Kiran M. et al. [35], their results show that by depicting the usage of instances and the cost incurred by running batch jobs over the three-month period using elastic map-reduce. Using larges machine instances such as m3.xlarge machines produces more costs even if run a few times. These costs can be optimized by replacing larger machines with

smaller machines such as C1.Medium, to give similar results but at lower costs. Elastic map-reduce involves creating a cluster of machines to act as master-slaves to deploy and run the map-reduce jobs. Therefore, replacing these devices with smaller devices and more methods can help reduce the device cost and also execute the same processing. The runtime of C1.Medium cluster was approximately 10 minutes, which was slower (of 5 minutes) than the m3.large machine clusters. This explains the fact that the lambda architecture on Amazon AWS was able to present a proof-of-concept for data management and processing both as data reaches and if it is protected prior to the scripts. The tailored solutions allow users to perform cost-optimized processing. Both processes can produce data aggregations that are easier to plot and reduces the time for data processing through the data visualization interface.

As described briefly earlier, Lambda Architecture is a pattern with well-defined standards and is technology atheist. Looking at its various components/layers, any technology can be brought in to do the required job. With, the emergence of various cloud providers, you could even get ready-made components in the cloud (many are cloud-dependent) that actually implements the Lambda Architecture. In this paper, we are continuing ahead to create a Data Lake in which the lambda pattern just covers one layer, called Lambda Layer. Since there are so many preferences for technologies, the future of technologies is a bit dogmatic. When we make each technology, we would give the rationale for our choice, but keeping it as open as possible. We would also give our other technology preferences so that if needed, these technologies can indeed be exchanged by the reader if required.

The shortcomings of lambda architecture have focused on its inherent complexity and its limiting influence. The batch and streaming sides each require a different codebase that must be preserved and kept in synchronization, so that managed data produces the same result from both paths. Yet endeavoring to abstract the codebases into a single structure that puts many of the specialized tools in the batch and real-time ecosystems out of reach. In a technical discussion over the merits of employing a pure streaming approach, it was noted that using a flexible streaming framework such as Apache Samza could enhanced some of the same advantages as batch processing without the latency. Such a flowing structure could allow for collecting and processing arbitrarily large windows of data, accommodate blocking, and handle state.

Furthermore, due to its different layers, it is commonly considered to be complex. Keeping synchronization between these two layers incurs cost and effort, and this has to be thought through and handled. Secondly, due to these two discrete and fully dispersed layers (batch and speed), maintenance and support events are quite hard. Thirdly, there are a good number of technologies that have to be mastered to construct a Lambda-Architecture-driven Data Lake. Getting people who have expertise in these technologies can be troublesome for our recruitment division for instance. Implementing a Lambda Architecture with open sources technologies and then deploying it in the cloud can be troublesome. To avoid this, we could very well use cloud technologies to implement Lambda Architecture, but by doing so, the Enterprise-level Data Lake is one of the applications of the Lambda Architecture pattern.

V. CONCLUSIONS

This paper presents an overview of both knowledge management in optical networks, Lambda Architecture, and Big-Data analytics in cloud environments. The concept has been evaluated and presented in two folds:

Knowledge Management: In conclusion, the KM process has been reviewed with various propositions aiming at the implementation of autonomous optical network operation with reference to their four pillars (knowledge acquiring, knowledge sharing mechanism, knowledge integration, and knowledge utilization). These parameters enable optical networks to autonomously identify and integrate knowledge that can utilize to adapt their design to variable situations without human interference. Furthermore, two optional strategies comprise of distribution of data samples related to mode imprecision called data-based are evaluated. In regard to knowledge integration, three techniques were introduced (mode ensemble, model merge, and training data amalgamation). With these techniques' knowledge integration can be implemented in two main folds to supervised ML model called propagation and reinforcement learning.

Lambda Architecture: The Lambda Architecture provides a consistent approach to building a Big Data system that can perform real-time updates with low latency, high throughput, and fault-tolerant way. A single tool cannot meet all the requirements of an application, so there is a need to use technologies/tools as a conjunction in a layered way as specified in the Lambda Architecture to achieve an application that satisfies our requirements. Stack Overflow is a proper application to demonstrate the Lambda Architecture as it has a huge user community with large posts getting created every second. There is scope for a lot of improvement in obtaining results with better accuracy. We can use some machine learning techniques in identifying top tags for a user. Also, tags are provided by a user which is not validated, building a prediction model that predicts accurate tags to a question can improve question recommendation overall. We can also add Job Recommendations to the application based on user expertise. We can add events like MeetUps, Conference's information to the application based on user location and user expertise. All this information keeps the user more occupied whenever he visits the application and makes him an active user of the application. We can also increase the input events per second and batch dataset size to perform load testing on the application and scale the application accordingly.

REFERENCES

[1] Ruiz, M., Boitier, F., Layec, P. and Velasco, L., 2019, March. Self-learning approaches for real optical networks. In 2019 Optical Fiber Communications Conference and Exhibition (OFC) (pp. 1-3). IEEE.

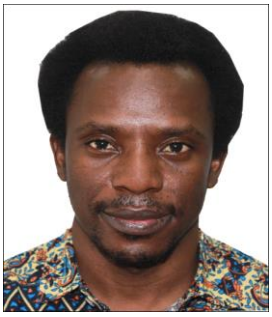
[2] Shahkarami, S., Musumeci, F., Cugini, F. and Tornatore, M., 2018, March. Machine-learning-based soft-failure detection and identification in optical networks. In 2018 Optical Fiber Communications Conference and Exposition (OFC) (pp. 1-3). IEEE.

[3] Nouioua, M., Fournier-Viger, P., He, G., Nouioua, F. and Min, Z., 2021. A Survey of Machine Learning for Network Fault Management. Machine Learning and Data Mining for Emerging Trend in Cyber Dynamics: Theories and Applications, p.1.

[4] Sambo, N., Tomkos, I., Shaikh, A., Bigo, S., Suzuki, M. and Schmidtke, H.J., 2018. Guest editorial: Optical networks

supporting interoperability and white boxes. Journal of Lightwave Technology, 36(15), pp.3058-3061.
[5] Esmenats, P.R., Casellas, R., Gifre, L., Vela, A.P., Ruiz, M., Martínez, R. and Velasco, L., 2019, March. Autonomic NFV network services on top of disaggregated optical metro networks. In 2019 Optical Fiber Communications Conference and Exhibition (OFC) (pp. 1-3). IEEE.
[6] Velasco, L., Casellas, R., Llana, S., Gifre, L., Martínez, R., Vilalta, R., Muñoz, R. and Ruiz, M., 2019. A control and management architecture supporting autonomic NFV services. Photonic Network Communications, 37(1), pp.24-37.
[8] Gillet, A., Leclercq, É. and Cullot, N., 2021, June. Lambda+, the Renewal of the Lambda Architecture: Category Theory to the Rescue. In International Conference on Advanced Information Systems Engineering (pp. 381-396). Springer, Cham.
[9] Kiran, M., Murphy, P., Monga, I., Dugan, J. and Baveja, S.S., 2015, October. Lambda architecture for cost-effective batch and speed big data processing. In 2015 IEEE International Conference on Big Data (Big Data) (pp. 2785-2792). IEEE.
[10] Ruiz, M., Tabatabaeimehr, F. and Velasco, L., 2020. Knowledge management in optical networks: architecture, methods, and use cases. Journal of Optical Communications and Networking, 12(1), pp.A70-A81.
[11] Lai, V.S. and Guynes, J.L., 1994. A model of ISDN (integrated services digital network) adoption in US corporations. Information & Management, 26(2), pp.75-84.
[13] Kroese, D.P., Taimre, T. and Botev, Z.I., 2013. Handbook of monte carlo methods (Vol. 706). John Wiley & Sons.
[12] Wang, X., Kihara, D., Luo, J. and Qi, G.J., 2020. EnAET: A self-trained framework for semi-supervised and supervised learning with ensemble transformations. IEEE Transactions on Image Processing, 30, pp.1639-1647.
[14] Ewen, S., Schelter, S., Tzoumas, K., Warneke, D. and Markl, V., 2013, June. Iterative parallel data processing with stratosphere: an inside look. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (pp. 1053-1056).
[15] Anjos, J.C., Fedak, G. and Geyer, C.F., 2016. BIGhybrid: a simulator for MapReduce applications in hybrid distributed infrastructures validated with the Grid5000 experimental platform. Concurrency and Computation: Practice and Experience, 28(8), pp.2416-2439.
[16] Ball, A., 2015. Technical challenges of the Large Hadron Collider experiments (ATLAS and CMS). Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 373(2032), p.20140045.
[17] Jirsik, T. and Celeda, P., 2020, April. Cyber Situation Awareness via IP Flow Monitoring. In NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium (pp. 1-6). IEEE.
[18] Mendiola, A., Astorga, J., Jacob, E. and Higuero, M., 2016. A survey on the contributions of software-defined networking to traffic engineering. IEEE Communications Surveys & Tutorials, 19(2), pp.918-953.
[19] Galibus, T., dos Anjos, J.C., de Freitas, E.P., Geyer, C.F.R., Fedak, G., ́oteo de Sousa Jr, R.T., Costa, J.P.C., Pereira, R., Fergus, P., Zaleski, A. and Vissia, H., 2016. Security framework for distributed data processinge.
[20] Pham, L.M., El-Rheddane, A., Donsez, D. and De Palma, N., 2016. CIRUS: an elastic cloud-based framework for Ubilytics. *Annals of Telecommunications*, *71*(3-4), pp.133-140.
[21] Tang, B., He, H. and Fedak, G., 2015. HybridMR: a new approach for hybrid MapReduce combining desktop grid and cloud infrastructures. Concurrency and computation: Practice and experience, 27(16), pp.4140-4155.
[22] Alexandrov, A., Bergmann, R., Ewen, S., Freytag, J.C., Hueske, F., Heise, A., Kao, O., Leich, M., Leser, U., Markl, V. and Naumann, F., 2014. The stratosphere platform for big data analytics. The VLDB Journal, 23(6), pp.939-964.
[23] Peng, Boyang et al. (2015). "R-storm: Resource-aware scheduling in storm". In: Proceedings of the 16th Annual Middleware Conference. ACM, pp. 149–161.
[24] Eskandari, Leila, Zhiyi Huang, and David Eyers (2016). "P-Scheduler: adaptive hierarchical scheduling in apache storm". In: Proceedings of the Australasian Computer Science Week Multiconference. ACM, p. 26.
[25] Boykin, O., Ritchie, S., O'Connell, I. and Lin, J., 2014. Summingbird: A framework for integrating batch and online mapreduce computations. Proceedings of the VLDB Endowment, 7(13), pp.1441-1451.
[26] Mian, R., Martin, P. and Vazquez-Poletti, J.L., 2013. Provisioning data analytic workloads in a cloud. Future Generation Computer Systems, 29(6), pp.1452-1458.
[27] Dobre, C. and Xhafa, F., 2014. Intelligent services for big data science. Future generation computer systems, 37, pp.267-281.
[28] Wingerath, W., Ritter, N. and Gessert, F., 2019. Data Stream Management. In Real-Time & Stream Data Management (pp. 43-55). Springer, Cham.
[29] Yan, Z., Tracy, C., Veeraraghavan, M., Jin, T. and Liu, Z., 2016. A network management system for handling scientific data flows. Journal of Network and Systems Management, 24(1), pp.1-33.
[30] Chaiken, Ronnie et al. (2008). "SCOPE: easy and efficient parallel processing of massive data sets". In: Proceedings of the VLDB Endowment 1.2, pp. 1265–1276.
[31] Warren, J. and Marz, N., 2015. Big Data: Principles and best practices of scalable realtime data systems. Simon and Schuster.
[32] Vogelstein, J.T., Perlman, E., Falk, B., Baden, A., Roncal, W.G., Chandrashekhar, V., Collman, F., Seshamani, S., Patsolic, J.L., Lillaney, K. and Kazhdan, M., 2018. A community-developed open-source computational ecosystem for big neuro data. Nature methods, 15(11), pp.846-847.
[33] Velasco, L., Shariati, B., Boitier, F., Layec, P. and Ruiz, M., 2019. Learning life cycle to speed up autonomic optical transmission and networking adoption. Journal of Optical Communications and Networking, 11(5), pp.226-237.
[34] da Silva, V.A., 2016. Strategies for big data analytics through lambda architectures in volatile environments. IFAC-PapersOnLine, 49(30), pp.114-119.
[35] Kiran, M., Murphy, P., Monga, I., Dugan, J. and Baveja, S.S., 2015, October. Lambda architecture for cost-effective batch and speed big data processing. In 2015 IEEE International Conference on Big Data (Big Data) (pp. 2785-2792). IEEE.

Author's Profile



First Author: Abdul Joseph Fofanah is with the Department of Mathematics & Computer Science, Faculty of Environmental Sciences, Milton Margai College of Education and Technology. In 2008, he received an Associate Degree in Mathematics at the Milton Margai College of Education and Technology now "Milton Margai Technical University", in 2013 he attained a B. Sc (Hons.) in Computer Science, in 2018 he received a master's degree in Computer Science at Njala University, and in 2020 he attained a Master's in Software Engineering at Nankai University, China, and currently he is pursuing his Ph.D. program in Information Health Systems at the Atlantic International University. From 2009-date he has worked as a lecturer and researcher at MMCET now MMTU, Freetown, Sierra Leone. He has also worked for the International Organization for Migration as Data and GIS Officer and as a consultant. During the multiple outbreaks in West Africa, he attained the position of Data Scientist, where he was managing migration data across the region. As a result of his outstanding performance, he was sent to Kenya to manage mobility tracking mechanism during their flooding deserter including the COVID-19 pandemic in Nepal and Afghanistan

Second Author: Saidu Koroma was formerly the Head of Department of Mathematics & Computer Science in the Faculty of Environmental Sciences, Milton Margai College of Education and Technology (MMCET), Goderich Campus, Freetown, Sierra Leone. Currently, he is the Dean of Faculty of Environmental Sciences. He studied at Njala University College, University of Sierra Leone (NUC/USL) and attained a B.Sc. degree in Agriculture General. In 1996, he attained a Diploma in Data Processing at the Institute of Public Administration and Management, University of Sierra Leone (IPAM/USL). In 2003/2004 academic year, he pursued a Master of Science (M.Sc.) degree in Development Studies. In 2013, he attained Certificates in Computer Networking (C+, A+, CCNA and MCITP), Linux Administration and IT Security at UTL, Bangalore, India. In 2021, he also completed a Master of Science (M.Sc.) program in Information Systems Management at Njala University and is awaiting result. In 1998, he worked as a Teacher at the Harford Secondary School for Girls and in 1999, he was employed as Programme Manager at the Jesus Healing Ministries, a Local NGO, Committed to Community Development and Evangelism. He has also been involved in some research work for various organizations. He was however, employed at MMCET now Milton Margai Technical University (MMTU) as Lecturer in the year 2000, where he rose through the ranks.

Third Author: Issa Fofana (Ph.D.), Lecturer, Department of Physics and Computer Science, Njala University. He obtained B.Sc. (Ed) Physics and Mathematics – Njala University College, University of Sierra Leone in 1980, taught Physics and Mathematics in secondary schools from 1980 t0 1984. Attained M.Sc., and Ph.D. in Applied Physics, Technical University of Wroclaw, Poland in 1987 and 1992, respectively. Employed as a lecturer in the Department of Physics in 1992. Head of Department, Department of Physics 2004-2010, where he transformed the department and had it retitled Physics and Computer Science. Acted as Dean School of Technology, from 2010 – 2013.