APRIL: An Application-Aware, Predictive and Intelligent Load Balancing Solution for Data-Intensive Science

Deepak Nadig, Byrav Ramamurthy, Brian Bockelman, David Swanson
Department of Computer Science and Engineering,
University of Nebraska-Lincoln, Lincoln, NE, 68588 USA
Email: {deepaknadig, byrav, bbockelm, dswanson}@cse.unl.edu.

***Abstract*—In this paper, we propose an application-aware intelligent load balancing system for high-throughput, distributed computing, and data-intensive science workflows. We leverage emerging deep learning techniques for time-series modeling to develop an application-aware predictive analytics system for accurately forecasting GridFTP connection loads. Our solution integrates with a major U.S. CMS Tier-2 site; we use a real dataset representing *670 million* GridFTP transfer connections measured over 18 months to drive our predictive analytics solution. First, we perform extensive analysis on this dataset and use the connection loads as an example to study the temporal dependencies between various user-roles and workflow memberships. We use the analysis to motivate the design of a gated recurrent unit (GRU) based deep recurrent neural network (RNN) for modeling long-term temporal dependencies and predicting connection loads. We develop a novel application-aware, predictive and intelligent load balancer, APRIL, that effectively integrates application metadata and load forecast information to maximize server utilization. We conduct extensive experiments to evaluate the performance of our deep RNN predictive analytics system and compare it with other approaches such as ARIMA and multi-layer perceptron (MLP) predictors. The results show that our forecasting model, depending on the user-role, performs between 5.88%–92.6% better than the alternatives. We also demonstrate the effectiveness of APRIL by comparing it with the load balancing capabilities of an existing production Linux Virtual Server (LVS) cluster. Our approach improves server utilization, on an average, between 0.5 to 11 times, when compared with its LVS counterpart.**

I. Introduction

Recently, software defined networking (SDN) [1] and big data technologies [2] have received significant interest from both academia and industry. While big data, characterized by “5Vs” (volume, variety, velocity, value, and veracity), can have profound impacts on network design, such aspects have traditionally been addressed separately from the SDN paradigm. Some SDN features including control/data plane separation, programmability/reconfigurability, and logical centralization can positively benefit big data tasks such as data acquisition [3], delivery [4], [5], [6] and storage [7].

An ever-increasing need for big data in science has led to the rapid adoption of flexible (and programmable) high-speed network infrastructure. Such infrastructures typically rely on 100Gbps links to support large-scale data movement. As an example, the high-energy physics community through the Large Hadron Collider (LHC) project, has experimental data transfers reaching tens of petabytes every year. Example data-intensive science workflows include the Compact Muon Solenoid (CMS) [8] and Laser Interferometer Gravitational-Wave Observatory (LIGO) [9]. The most popular tools for big data movement include GridFTP [10] and XROOTD [11]. Since scientific research is highly data-driven, they place an undue burden on campus networks for data delivery, storage, and processing. Flexible and scalable end-to-end network architectures are necessary to ensure that data transfer applications use the network efficiently. Numerous scientific big data architectures have been developed (e.g. [12], [13], [14]) to avoid performance hot-spots associated with traditional networks.

Numerous research efforts (e.g., [15], [16], [17]) have focused on SDN-based efficient network resource allocation algorithms and techniques for cloud and data center networks. However, most of these techniques target the optimization of network resources allocation based on factors such as traffic demand/loads, quality of service (QoS) requirements and usage patterns. Such key factors are generally highly volatile and time-varying in nature. Limited work has been done to model data transfers, or to predict the key factors that affect network resource allocation. Load balancing forms a critical component of big data network architectures as they directly influence application response times and maximize throughput via optimized traffic delivery to the application servers. Large-volume data transfers associated with big data provides many opportunities for understanding usage patterns and gain insights into network resource requirements. Rather than viewing big data systems as placing an undue burden on campus networks, we can exploit the insights gained in better understanding user/traffic demands. This results in optimized resource allocation to better serve the needs of campus network users.

In this paper, we propose a novel intelligent load-balancing technique for improving server utilization using application-aware SDN and deep learning approaches. We also propose a deep learning approach for modeling large data transfers in the campus network. Deep learning is a representational learning technique that can automatically discover data representations using a multi-layer network [18]. The representation is then used to infer large dataset information without the need for complex analysis. In this work, we demonstrate how deep learning based predictors can be utilized to make accurate predictions and forecast future network connections by relying on application-layer metadata. We implement a gated recurrent unit (GRU) [19] based deep learning model for GridFTP connection time-series predictions. Our model incorporates an application-aware SDN system to classify traffic and to facilitate application-layer metadata exchange with the network-layer. To the best of our knowledge, this is the first effort to leverage application-aware SDN and deep learning techniques for modeling/predicting big data science data transfers for load balancing applications. We also show the effectiveness and superiority of our approach by evaluating it with a real-world dataset from a major U.S. CMS Tier-2 site.

A. Contributions and Organization

The specific contributions of this paper are as follows:

1) *GRU-based Deep Learning Predictive Analytics*: We develop a gated recurrent unit (GRU) based deep learning model for GridFTP connection time-series prediction. The model employs an application-aware SDN approach to obtain accurate and reliable traffic classification information to forecast future connections.
2) *Novel Application-aware Load Balancing*: We propose a novel application-aware, predictive and intelligent load balancing algorithm (APRIL). APRIL combines application-layer metadata with deep learning predictive analytics resulting in an intelligent load balancer.
3) *Real-world large-scale dataset*: We demonstrate our model's effectiveness through extensive evaluations using a real dataset from a U.S. CMS Tier-2 site. We present detailed data analysis to discover and identify long-term temporal dependencies in the dataset. We also compare our deep learning predictive model with other approaches such as Autoregressive integrated moving average (ARIMA) and multi-layer perceptron predictors.
4) *Scalability and Improvements over LVS*: We also demonstrate the scalability of our solution by deploying our model on a project testbed network that has been set up to integrate with a U.S. CMS Tier-2 site. We compare the benefits and superiority of our solution with an existing production Linux Virtual Server (LVS) cluster.

The paper is structured as follows: Section II provides a brief overview of application-aware SDN in the context of load-balancing, the role of predictive analytics, and describe related works; Section III presents the exploratory analysis of our dataset; In Section IV, we detail our experimental network testbed and our experimental setup; Section V presents our deep learning models and approaches to predicting GridFTP connection transfers. We describe our Gated Recurrent Unit (GRU) based prediction model for time-series load forecasting across multiple GridFTP servers. In Section VI, we present our intelligent application-aware load-balancing solution (APRIL) for managing distributed high-throughput data transfers in the campus network. Lastly, in Section VII, we conclude our work and discuss the future work.

II. Related Work

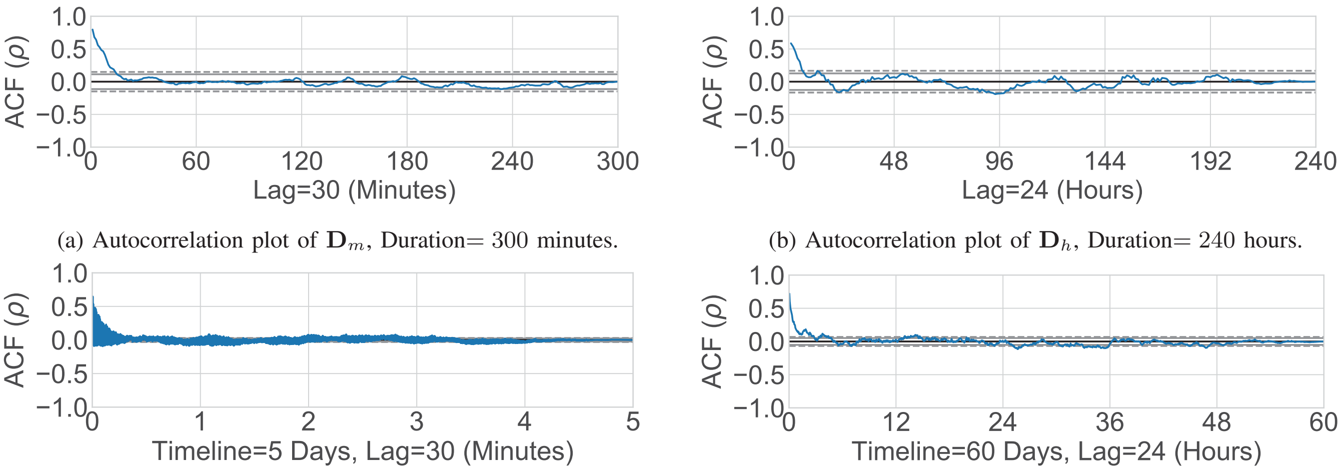
Numerous research efforts have focused on developing SDN load balancing mechanisms. The work in [20] presents a comprehensive survey on SDN load balancers. However, most of these are based on traffic routing mechanisms, or improve factors such as latency, synchronization, QoS, etc., or use heuristic optimizers to improve performance. Other works such as [21], [22] focus on LVS performance improvements. Although load balancers support specific transport-layer protocols, limited work has been done to develop true application-aware load balancing systems. Recent efforts such as [23], [24], [25] leverage machine/deep learning techniques for traffic classification and/or predictions. Integrated SDN and deep learning techniques have also be employed in VNF placement in NFV networks [26] and SDN security among others. Different from the above, our work focuses on leveraging deep learning techniques to accurately model and predict the data-intensive science traffic load by exploiting an application-aware SDN solution. Further, we develop a novel intelligent load balancer that combines both application-layer metadata and future forecast knowledge to improve server utilization.

III. Data Analysis and Modeling

Data analysis and modeling is an essential step in providing us valuable insights about the temporal dependencies in the dataset. This information is critical to choosing an appropriate data and prediction model for improved forecast accuracy. In this section, we present our dataset used in the modeling and prediction. We also perform exploratory data analysis on the dataset to motivate our design choices.

A. Dataset

The dataset consists of GridFTP transfer connection data obtained from a major U.S. CMS Tier-2 site that performs frequent high-volume (low- and high-priority) transfers to Fermilab and holds over 3 petabytes of data. The site uses both the GridFTP protocol and XROOTd for bulk batch transfer jobs and interactive jobs, respectively. The data was obtained using an application-aware approach similar to the ones in [27], [28]. The dataset includes GridFTP connection information collected from a single U.S. CMS Tier-2 site over 18 months. The dataset represents over *670 million* GridFTP connections from both CMS and LIGO workflows. The dataset contains connection information classified by four CMS user roles (as defined in the CMS computing model [8]) and a single LIGO user role. A pool of twelve (12) GridFTP servers are employed by the site to serve both campus network users and external researchers. The four CMS user roles include: i) *US CMS Pool* representing analysis transfers associated with users' jobs, ii) *CMSProd* similar to (i), but representing production workflows, iii) *CMS PhEDEx* representing the CMS production data movement, iv) *LCG Admin* representing site availability monitoring transfers. The LIGO user-role represents LIGO transfers that are opportunistic and share networking resources with CMS users. Other users include site administrators and computing center staff.



(a) Autocorrelation plot of $\mathbf{D}_m$, Duration= 300 minutes.

(b) Autocorrelation plot of $\mathbf{D}_h$, Duration= 240 hours.

(c) Autocorrelation plot of $\mathbf{D}_m$, Duration= 5 days.

(d) Autocorrelation plot of $\mathbf{D}_h$, Duration= 60 days.

Fig. 1: Temporal autocorrelation properties of the datasets.

The GridFTP protocol uses encrypted control and data channels for data movement between end-points. Application-layer metadata is crucial for classifying the GridFTP connection information. Therefore, we rely on the GridFTP XIO plugin [29] to securely interface with the GridFTP servers and facilitate application metadata exchange with the SDN. The obtained connection information is pre-classified based on its workflow (or experiment) membership and also based on the user role within the workflow. User roles not belonging to either workflow are classified as "*Other*" users. The default dataset measurement granularity is in the order microseconds. However, as this granularity does not permit meaningful analysis, we use data aggregation to obtain aggregate statistics with granularities of one minute and one hour respectively. A *nonuple* (i.e., 9-tuple) is used to identify each connection uniquely and includes connection strings, user/workflow membership, files transferred, transfer direction and a status field. We denote the GridFTP connection dataset by $\mathbf{G} = \{g_{(a_1,\cdots,a_9),n,t}\}, \forall t, \forall n \subset N$. We denote the downlink and uplink connections by $\mathbf{D} = \{d_{(a_1,\cdots,a_9),t}\}, \forall t, \forall n \subset N$ and $\mathbf{U} = \{u_{(a_1,\cdots,a_9),t}\}, \forall t, \forall n \subset N$, respectively, for N observations. We normalize both datasets $\mathbf{D}$ and $\mathbf{U}$ within a range of $[0, 1]$. To achieve this, we use *Min-Max* scaling to compute the normalized values $\hat{\mathbf{G}}$ as:

$$\hat{\mathbf{G}} = \frac{\mathbf{G} - \mathbf{G}_{min}}{\mathbf{G}_{max} - \mathbf{G}_{min}} \tag{1}$$

where, $\mathbf{G}_{max}$ and $\mathbf{G}_{min}$ represent the maximum and minimum values of the set $\mathbf{G}$, respectively. Also, we note that $\mathbf{D} \subset \mathbf{G}$ and $\mathbf{U} \subset \mathbf{G}$. Lastly, we denote the datasets with different measurement granularities by $\mathbf{G}_m$ (one minute) and $\mathbf{G}_h$ (one hour), respectively.

B. Exploratory Analysis

The objective of our exploratory data analysis is to discover and identify data dependencies in $\mathbf{G}$, $\mathbf{D}$ and $\mathbf{U}$ in the temporal domain. Initially, we analyze the dataset for the presence of systematic patterns combined with random error. By identifying and removing non-stationary processes within the dataset, we can obtain a dataset with independent identically distributed (i.i.d.) components that are amenable to modeling using linear regression on exogenous variables. We also examine temporal autocorrelation and data dependency between different user-roles for both datasets, i.e., $\mathbf{D}$ and $\mathbf{U}$, respectively. We make the following important observations:

Observation 1: The dataset $\mathbf{G}$ *exhibits non-stationarity.*

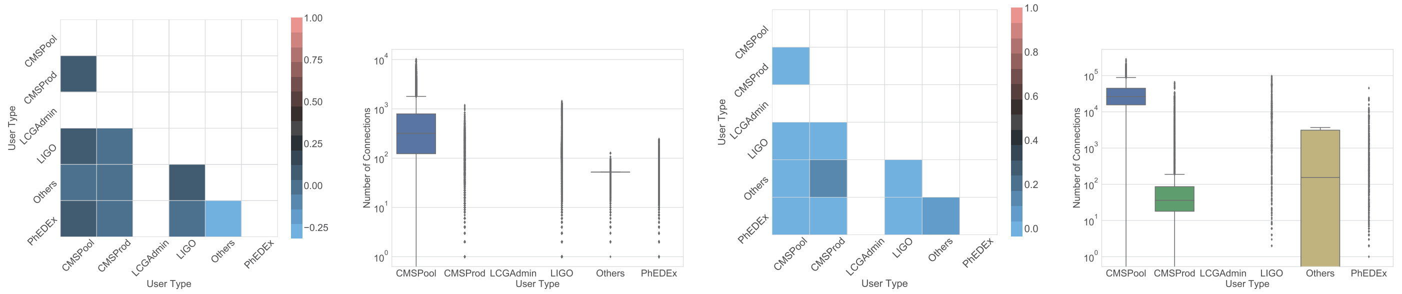
We denote the k^{th} observation as $G_k = g_{(a_1,\cdots,a_9),k,t}$. The dataset $\mathbf{G}$ is strictly stationary if:

$$(G_1, G_2, \cdots, G_n)' \overset{\mathrm{d}}{=} (G_{1+h}, G_{2+h}, \cdots, G_{n+h})', \forall n \geq 1 \tag{2}$$

where, $\overset{\mathrm{d}}{=}$ denotes that the two random vectors share the same joint distribution function. The Augumented Dickey-Fuller (ADF) test is a widely used tool to test stationarity [30]. The autocorrelation plots of the two downlink datasets with measurement granularities of one minute and one hour are shown in the Figure 1. Figures 1a and 1b represent the short-run dataset (measured over 300 minutes and 240 hours, respectively), while Figures 1c and 1d represent the corresponding long-run datasets (measured over 5 and 60 days, respectively). The presence of significant autocorrelation in the lags for time $t > 1$ is an indicator of non-stationarity. This is also verified by running the ADF test on the above datasets, resulting in the test accepting the null hypothesis, indicating a non-stationary process. Thus, (first-order) differencing is required to stationarize the datasets.

Observation 2: The dataset $\mathbf{G}$ *has non-zero temporal autocorrelation properties.*

In the absence of significant autocorrelation, the data points in Figure 1 fall within the confidence interval bands represented by the dashed lines. The sample autocorrelation function (ACF) is commonly used to identify and discover data dependency between the observations. We define the sample ACF



(a) Correlation Heatmap – $\mathbf{D}_m$.

(b) $\mathbf{D}_m$ User-roles.

(c) Correlation Heatmap – $\mathbf{D}_h$.

(d) $\mathbf{D}_h$ User-roles.

Fig. 2: Short- and long-run dataset properties and cross-correlation.

for the dataset $\mathbf{G}$ by:

$$\rho(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)} = \frac{\sum_{t=1}^{n-|h|}(G_{t+|h|} - \bar{G})(G_t - \bar{G})}{\sum_{t=1}^{T}(G_t - \bar{G})^2}, -n < h < n \tag{3}$$

where, $\bar{G}$ represents the sample mean of $G_1, G_2, \cdots, G_n$, and n is the total number of observations in the sample. The autocorrelation plots shown in Figure 1 exhibits high values for ρ for lags $h > 1$. This indicates the presence of systematic patterns mixed with random errors.

Observation 3: The dataset $\mathbf{G}$ *exhibits low cross-correlation in the temporal domain, between different user-roles.*
We use the correlation matrix to assess the strength of the relationship between two user-roles with either the same, or, with different workflow memberships. Figure 2 shows the correlation matrix heatmaps for the short-run datasets (both 1m and 1h aggregates) along with the corresponding box-whiskers plots. Only the lower triangular correlation matrix is presented in the correlation heatmaps for brevity. As shown in the Figures 2a and 2c, we observe low to moderate cross correlation between the users' transfer connections for both datasets. For the short-run dataset with one-minute aggregates $\mathbf{D}_m$, we see low- to moderate positive correlation between different users' transfers and low negative correlation in one instance. Similarly, we see low positive correlation between the users' transfers for the short-run dataset with hourly aggregates $\mathbf{D}_h$. Both correlation matrices represent the users' connection transfer relationships, and provide some insight into the number of parameters required to estimate them. The box plots for datasets $\mathbf{D}_m$ and $\mathbf{D}_h$ are shown in Figures 2b and 2d, respectively. From the box plot, we observe that the *CMSPool* users exhibit large variations across both $\mathbf{D}_m$ and $\mathbf{D}_h$ datasets. *CMSProd* and *Other* show large hourly variations, while *LIGO* and *PhEDEx* exhibit little variation across datasets but have significant outliers. The box plots are useful in helping us understand the distribution characteristics of the datasets and in outlier detection.

IV. Experimental Testbed

In this section, we present our experimental setup, an application-aware architecture to integrate with the GridFTP server pool, our data management framework, the testbed network topology, and how it interfaces with the Linux Virtual Server (LVS) [31] load balancing cluster. Our experimental network topology is shown in Figure 3. It consists of five components namely: (i) the GridFTP server pool, (ii) the LVS load balancing cluster and a LVS redirector, both of which are transparent to end-user applications, (iii) the application-aware SDN infrastructure, (iv) the Elastic stack cluster for data management, and (v) the SDN data plane infrastructure and 100Gbps connectivity to the wide area network (WAN).

A. Application-aware SDN and GridFTP Integration

Application-awareness is achieved using the Globus eXtensible I/O (XIO) [29] extensible I/O library. We develop a Globus XIO SDN Callout to interface with the SDN infrastructure. The XIO Callout module integrates GridFTP servers with the SDN via an SDN application similar to SNAG [27]. It also uses a Hadoop Distributed File System (HDFS) plugin to interact with GridFTP servers' distributed storage/processing infrastructure. The XIO Callout module facilitates the exchange of application-layer metadata with the SDN infrastructure.

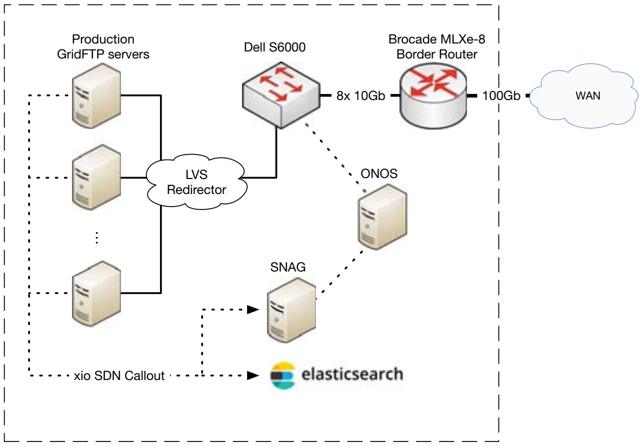


Fig. 3: Experimental Testbed.

B. Network Testbed Topology

Our network testbed setup is an exemplary implementation of an SDN that can handle frequent, high-volume, low- and high-priority data transfers from a major U.S. CMS Tier-2 site to Fermilab. The U.S. CMS Tier-2 site holds over 3PB of data, and uses both GridFTP and XROOTD protocols for bulk batch transfer jobs and interactive jobs, respectively. Our testbed network architecture effectively combines several important components including: (i) Intelligent flow control, flow forwarding and management using the ONOS SDN controller, (ii) An application-aware SDN application to facilitate secure exchange of application-layer metadata with the network-layer, (iii) A GridFTP Callout module that serves as an interface between the GridFTP servers, its HDFS storage backends and the SDN infrastructure. The XIO callout communicates with SDN controller using a secure representational state transfer (REST) application programming interface (API), and (iv) A Brocade MLXe border router at the campus network edge with 100Gbps WAN connectivity to Internet2. A Dell S6000 40 GbE switch to serve as the CMS cluster network core hosting 12 production GridFTP servers and an Edge-Core AS4600-54T SDN-capable switch for testing purposes.

C. *Data Management System*

Our current dataset includes information of over *670 million* GridFTP transfer connections from both CMS and LIGO workflows. This dataset is consistently growing as it is updated with new real-time connection information, while also being expanded to other workflows. To manage this large dataset, we employ a 11-node Elastic stack [32] cluster with the following configuration: (i) Master Nodes: 3×Dell SC1435, 16GB RAM, 250GB HDDs, (ii)Hot-Data/Ingest Nodes: 3×Sun SunFire X2200, 32GB RAM, 240GB SSDs, (iii) Warm Data Nodes: 5×Sun SunFire X2200, 32GB RAM, 2TB HDDs, and (iv) 1Gb Ethernet interconnects between all nodes.

This Elastic cluster is responsible for storing all application-aware information exchanged between the GridFTP server pool and the SDN infrastructure. A *syslog* style file on each of the 12 GridFTP servers feeds a *filebeat* agent (a lightweight data shipper for the Elastic stack), which in turn feeds the *logstash*, a server-side data ingestion pipeline on the Elastic cluster.

V. Modeling Loads and Predictive Analytics

A. *Overview*

We propose the use of recurrent neural networks (RNNs) for modeling and predicting time-series data. RNNs are a class of generalized feed-forward neural networks that exhibit dynamic temporal behavior and can, therefore, be used for time sequence modeling. The RNN can maintain and use internal states (memory) to process input sequences. However, standard RNNs suffer from well-known problems of vanishing/exploding gradients and therefore, using RNNs to model long-term dependencies is difficult [33]. Many solutions have been proposed including long short-term memory (LSTM) [34] and gated recurrent units (GRU) [19] to capture and model long-term temporal dependencies [35]. Both LSTM and GRU use “forget” gates that enable a model to both learn to forget previous states (i.e., dropping memory), and to update current states (i.e., adding new memory).

We specifically use a deep GRU network to make GridFTP connection predictions. A GRU cell is shown in Figure 4. The GRU cell is a simplified version of LSTM cell, but is known exhibit similar performance [36].

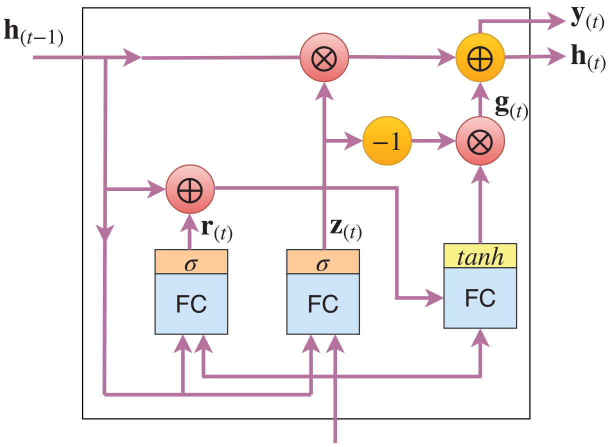


Fig. 4: Gated Recurrent Unit (GRU) Cell.

Unlike LSTM, both state vectors in GRU are merged into a single vector $\mathbf{h}_{(t)}$. The GRU cell state computations are summarized below:

$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z) \tag{4}$$

$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r) \tag{5}$$

$$\mathbf{g}_{(t)} = tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g) \tag{6}$$

$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)} \tag{7}$$

The terms $\mathbf{W}$ and $\mathbf{b}$ denote the weight matrix and the bias terms, respectively. Two types of activation functions are used by the fully-connected (FC) units namely: (i) $\sigma(\cdot)$, which is the sigmoid activation function, and (ii) $tanh(\cdot) = 2\sigma \cdot 2(x) - 1$, the hyperbolic tangent function. The terms $\oplus$ and $\otimes$ denote the sum and dot products, respectively. The update gate $\mathbf{z}_{(t)}$ helps the model control the amount of historical information passed to the next state. A reset gate $\mathbf{r}_{(t)}$ controls the amount of past information to forget. We note that a single gate controller is used to control both the update gate and the input gate. Whenever a new memory must be stored, its (storage) location is erased first. Lastly, we also note the absence of an output gate, and a full state vector is output at every time-step.

B. *Temporal Prediction Model*

Taking the observations in Section III-B into consideration, we develop a GRU-based deep learning network to model and predict the temporal GridFTP connection information classified by user-role. The deep learning network will forecast per-role connections for both CMS and LIGO users. The GRU memory cell updates the current hidden state $\mathbf{h}_{(t)}$ by combining the input and the past state as described in Equations 4–7. To predict the future value $G_{k,t+1} = g_{(a_1,\cdots,a_9),k,t+1}$, we rely on past T observations, i.e. $\sum_{\tau=t-T}^{t} G_{k,\tau}$. A 3-layer deep RNN with 64, 32 and 16 GRU cells is used for forecasting per user-role future connection values. We use a step-size of seven (7) at the input layer. A dropout layer is added to final hidden layer with a probability of 50% to avoid overfitting. We also note that Adam optimizer [37] was used with a training batch size of 16, with a linear activation function at the output layer.

C. Performance Evaluation

We compared our deep RNN model with two other time series analysis and prediction methods. First, we compare the prediction capabilities of our model with an Autoregressive integrated moving average (ARIMA) [38] multi-step predictor. ARIMA is a widely used method for time-series analysis and forecasting [38]. An ARIMA model is selected by minimizing the model's Akaike Information Criterion (AIC). The ARIMA model has three parameters: the AR model order p, the MA model order q, and the differencing component d. The model parameters (p,d,q) search-space is upper-bounded by (10,2,10). Next, we compare our model with a deep multi-layer perceptron (MLP) predictor consisting of three dense hidden layer with 64, 32, and 16 fully connected units, a dropout layer added to the final hidden layer with a probability of 50%, hyperbolic tangent activation functions at the hidden layer and linear activation at the output layer.

We compare the performance of our model with the ARIMA and the MLP predictors using three widely used performance metrics namely: (i) Mean Absolute Error (MAE), (ii) Mean Squared Error (MSE), and (iii) the coefficient of determination (r^2 score). We have also presented the root mean squared error (RMSE) for convenience. The dataset used in making the predictions $\mathbf{G}_m$ (1-minute aggregate granularity), was measured over 60 days and contained over 512,000 GridFTP transfer connection records from six user-roles described in Section III-A. This dataset was partitioned into a training set and a validation set, categorized by user-roles. Next, we present the prediction results of our deep RNN model and compare it with ARIMA and MLP forecasting models.

D. Prediction Results and Discussion

The prediction results for the $\mathbf{D}_m$ dataset (downlink connections 1-minute aggregate granularity) is shown in Figure 5. Figures 5a, 5b, 5c and 5d show the actual vs. predicted connection values for US CMS Pool, CMS Prod, CMS PhEDEx and LIGO users, respectively. From the results, we see that predicted results show a good fit with actual observations. The prediction models' performance categorized by user-role is presented in Figure 6. Specifically, we present the MAE, MSE, RMSW and r^2 scores in the Figures 6a, 6b, 6c, and 6d, respectively. Our proposed model, depending on the user-role, shows an improved error performance between 22.03%–65.96%, 23.8%–92.6%, and 13.37%–72.87% regarding MAE, MSE and RMSE, respectively over the ARIMA model. Our model also shows r^2 score improvements between 21.8%–217.14% over the ARIMA model. Further, our model, in comparison to the MLP model, shows an improved error performance between 3.28%–62.8%, 5.88%–85%, and 2.93%–62.36% regarding MAE, MSE and RMSE, respectively. It also shows r^2 score improvements between 8.06%–105.64% over the MLP model. The above results show the effectiveness of our GRU-based deep RNN model in making accurate GridFTP connection load predictions. Also, importantly, the superiority of our design ensures that it takes long-term temporal dependencies into account. Such a system is vital in providing timely intelligence to load balancing systems. In the next section, we demonstrate how accurate per user-role predictions can be effectively used to develop intelligent load balancing schemes for the LVS [31] cluster.

VI. Application-aware Load Balancing

Data-intensive science applications, with users interacting with massive amounts of data, place dynamically varying demands on the network infrastructure. However, conventional campus network and supercomputing center architectures, without a global view of the network, rely on load balancers that are not precise. With the emergence of SDN, significant research has gone into developing accurate load balancing methods with better performance than their conventional alternatives [20]. However, limited work has been done in developing efficient load balancers capable of handling massive amounts of data transfers intelligently from high-throughput distributed computing workflows.

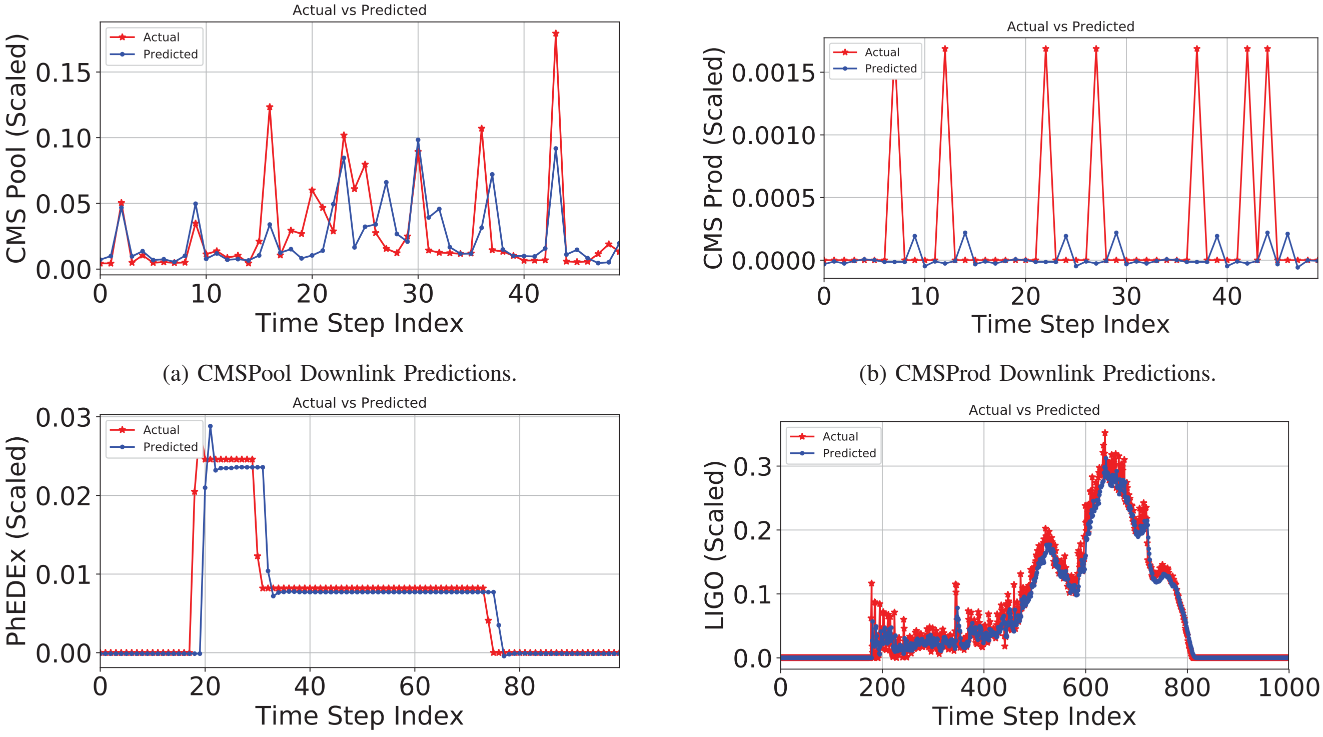
The Linux Virtual Server (LVS) [31] is a high-availability, highly-scalable load balancing solution built on a cluster of real servers. The LVS architecture is fully transparent to end-users/applications and behaves as a single high-performance virtual server. LVS is a widely used open source load balancing solution in many supercomputing centers. LVS implements several load balancing schedulers including (weighted) round-robin, (weighted) least-connections, source/destination hashing, and locality-based least-connection schedulers. While these schedulers perform adequately, they do not provide fine-grained controls for intelligently balancing connection loads based on application behavior.

The use of application-layer metadata benefits load balancing systems by allowing them to make intelligent decisions based on application behavior. However, such application metadata exchange is often *limited or nonexistent*. In the following, we propose an intelligent load balancer that exploits both application-awareness and predictive analytics knowledge to provide fine-grained load balancing controls.

A. Application-aware Predictive Intelligent Load Balancer (APRIL)

We propose APRIL, an application-aware, predictive, intelligent load balancer. APRIL intelligently combines application-layer metadata with deep learning predictive analytics to create customized load balancing policies. Our proposed approach is highly adaptable to both end-user/application requirements and behavior while providing fine-grained controls to the site administrator to prioritize or isolate desired flows.

We demonstrate an approach that exploits application-awareness and per user-role forecast information to maximize GridFTP server utilization. The proposed approach, APRIL is described in Algorithm 1. First, we define per-server maximum



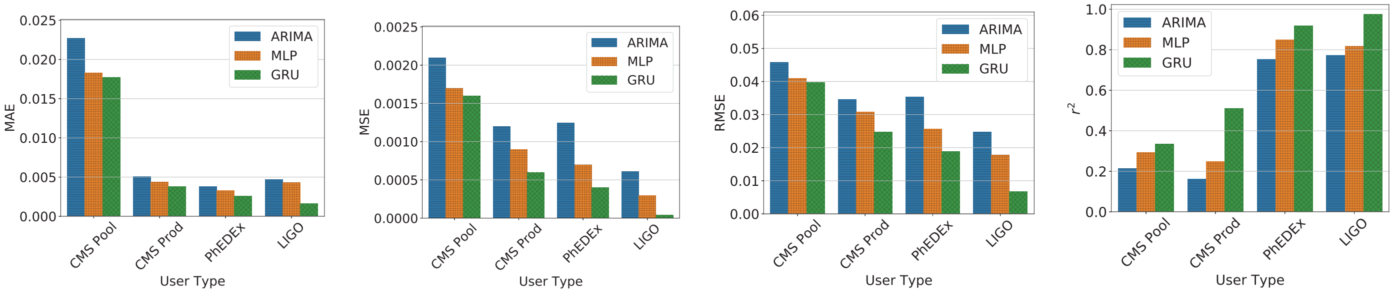
(a) CMSPool Downlink Predictions.

(b) CMSProd Downlink Predictions.

(c) CMS PhEDEx Downlink Predictions.

(d) LIGO Downlink Predictions.

Fig. 5: Predicted values vs. Actual $\mathbf{D}_m$ measurements by user-role.



(a) Mean Absolute Errors.

(b) Mean Squared Errors.

(c) Root Mean Squared Errors.

(d) Coefficient of Determination.

Fig. 6: Prediction model performance categorized by user-role.

capacity and weights. The weights are used to decide the preferred order of load distribution among the servers upper-bounded by their capacity. The weights are (re)adjusted based on the per user-role forecast information periodically to ensure that the utilization is maximized. We formally define the problem as:

$$Min. \left(\frac{C_n \cdot S_k}{W_k} \right), \forall n \in N, \forall k \in K \quad (8)$$

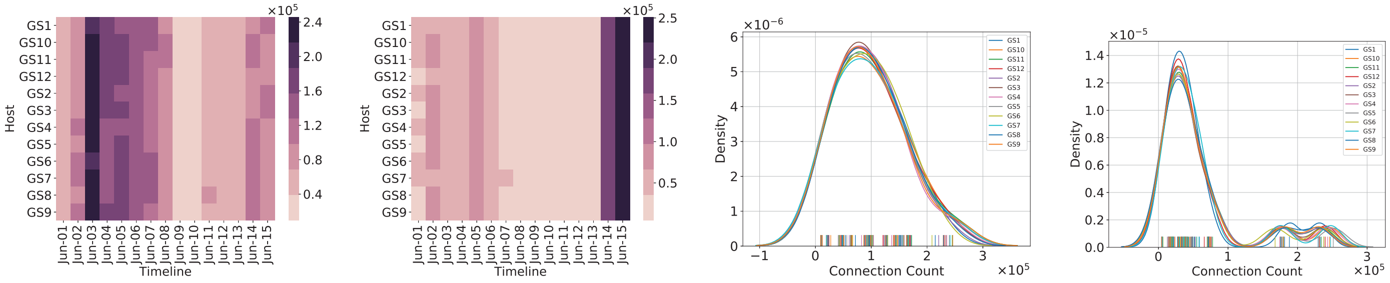
where, C_n represents the number of connections across K servers S_k, each weighted by W_k. The weight updates, W_k for each server S_k is defined as:

$$W_k = \alpha_\kappa \kappa_{S_k} \times W_{k-1} \times \frac{|C_{act}|}{|C_{pred}|}, \forall k \in K, \forall \alpha_\kappa \in (0,1] \quad (9)$$

where, κ_{S_k} is the current server capacity; α_κ is the capacity threshold. W_k and W_{k-1} are the current and previous weights, respectively. C_{act} and C_{pred} represent the total current and predicted connections, respectively. Each servers' weights are adjusted based on the predictions for that observation period. By using application-layer metadata and per user-role forecast information, we can maximize the server utilization by assigning the appropriate weights for each server. The weights also ensure that an appropriate number of connections live on each server without exceeding the capacity (*viz.* controlled by α_κ).

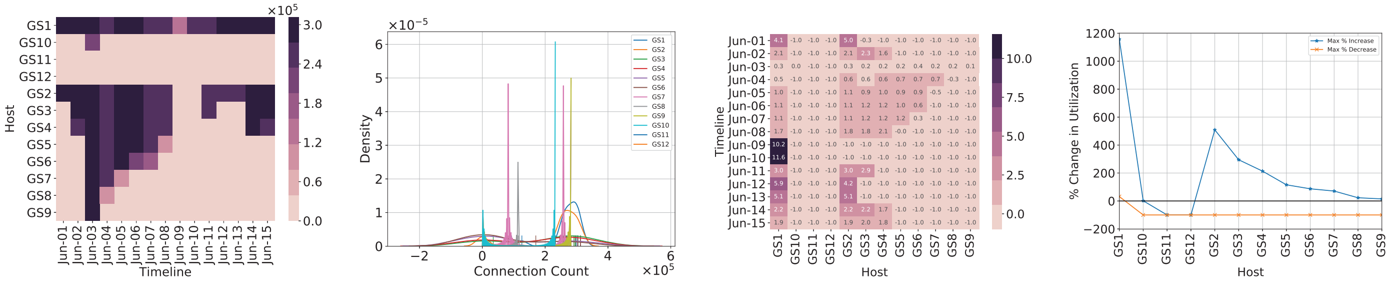
B. Results and Discussion

First, we present our experiences with the LVS weighted least-connection (WLC) scheduling, which is the primary scheduler used in our production U.S. CMS Tier-2 site. Figure 7 shows the LVS WLC scheduling heatmaps for the 12 GridFTP servers (labeled *GS1–GS12*) in the production network. The Figures 7a and 7b show the downlink and uplink connection distribution, respectively, when LVS WLC



(a) Load - Downlinks. (b) Load - Uplinks. (c) KDE - Downlinks. (d) KDE - Uplinks.

Fig. 7: LVS weighted least-connection scheduling load distribution (15 Days).



(a) Load - Downlinks. (b) KDE - Downlinks. (c) Daily % change/server. (d) Change in utilization/server.

Fig. 8: APRIL scheduling load distribution (15 Days).

Algorithm 1 APRIL($\mathbf{G}, S_k, \alpha_k$)

```
Require: Connection dataset (G), Servers $S_k$, Capacity threshold $\alpha_k$.
Output: Load distribution.
$\kappa_S = \alpha_k \cdot \kappa_{S_k}, \forall \alpha_{S_k} \in (0, 1]$
$W_k = 1, \forall s \in S_k$
for $C_{act} \in \mathbf{G}$ do
    Compute $C_{pred,t+1} = \text{RNN}(C_{act}, \tau), \forall \tau \in (t - \tau, t)$
    while $C_{act} \neq \phi$ do
        Find $s \in S_k$ with the smallest $\kappa_S$
        for $s \in S_k$ do
            if $\kappa_S \leq \alpha_k \cdot \kappa_{S_k}$ then
                $s := \{C_{act,t}, W_k\}$
                $W_k := \alpha_\kappa \kappa_{S_k} \times W_{k-1} \times \frac{|C_{act}|}{|C_{pred}|}, \forall k \in K$
                break
            else
                Find $s \mid \kappa_S \leq \alpha_k \cdot \kappa_{S_k}$
            end if
        end for
        $W_{k-1} := W_k$
        $C_{act} := \{C_{act} \cup C_{pred}\}$
    end while
end for
```

is used. The corresponding kernel density estimates (KDE) shown in Figures 7c and 7c indicate that load is almost equally distributed across all servers. The distribution performance of our proposed method, APRIL, is shown in Figure 8. From the heatmap shown in Figure 8a, we see that APRIL is better at redistributing loads with an objective of maximizing server utilization. This is also confirmed by the KDE in Figure 8b, which shows the difference in probability density for servers with increased utilization. Lastly, we show the resulting daily percentage change effected by APRIL in each server when compared to LVS WLC, in Figure 8c. We observe that our approach simultaneously maximizes utilization (up to 11 times increase) in some servers while reducing utilization significantly in others (a minimum of 0.54 times decrease). The per-server (maximum and minimum) percentage change averaged over 15 days is also presented in Figure 8d.

Although we have mainly presented the results by comparing our approach with LVS WLC, we note that other LVS scheduling algorithms were also evaluated during our experiments. Specifically, we configured LVS to use three additional schedulers on the production network namely: round-robin (both pure and weighted), source hashing, and destination hashing. Other than WLC, these other schedulers exhibited unstable behavior for opportunistic transfers such as LIGO workflows. This resulted in frequent dropped connections in the production network and server loading problems, and therefore we had to revert to WLC for stable network operation.

VII. Conclusions

We proposed an application-aware intelligent load balancing system (APRIL) for high-throughput data-intensive science workflows such as CMS and LIGO. Our proposed solutions used a real dataset representing *670 million* GridFTP transfer connections from a major U.S. CMS Tier-2 site. We presented an extensive analysis of this dataset to identify long-term temporal dependencies between different user-roles and workflow memberships. Using the insights from the data analysis, we leveraged deep learning techniques for time-series modeling to develop an application-aware predictive analytics system using gated recurrent units (GRU) based recurrent neural network (RNN). Our deep RNN predictive analytics system accurately forecasts GridFTP connection loads and performs between 5.88%–92.6% better than ARIMA or multi-layer perceptron (MLP) models. We then developed a novel application-aware, predictive and intelligent load balancer, APRIL, that effectively integrates application metadata and load forecast information to maximize server utilization. Through extensive experiments, we demonstrated the effectiveness of APRIL by comparing it with an existing production Linux Virtual Server (LVS) cluster. We show that our approach improves server utilization, on an average, between 0.5–11 times over its LVS counterpart. Our future work will focus on developing load balancing schemes that will consider a broader range of application metadata parameters.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Numbers OAC-1541442 and CNS-1817105. This work was completed using the Holland Computing Center of the University of Nebraska, which receives support from the Nebraska Research Initiative. The authors would like to thank Garhan Attebury, Holland Computing Center at UNL for his valuable support.

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo *et al.*, "Software-Defined Networking: A Comprehensive Survey," *Proc. of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[2] L. Cui, F. R. Yu, and Q. Yan, "When big data meets software-defined networking: SDN for big data and big data for SDN," *IEEE Network*, vol. 30, no. 1, pp. 58–65, January 2016.

[3] I. Monga, E. Pouyoul, and C. Guok, "Software-Defined Networking for Big-Data Science - Architectural Models from Campus to the WAN," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Nov 2012, pp. 1629–1635.

[4] Y. Han, S. s. Seo, J. Li *et al.*, "Software defined networking-based traffic engineering for data center networks," in *16th Asia-Pacific Network Operations and Management Symposium*, Sept 2014, pp. 1–6.

[5] S. Jain, M. Khandelwal, A. Katkar *et al.*, "Applying big data technologies to manage QoS in an SDN," in *2016 12th Conference on Network and Service Management (CNSM)*, Oct 2016, pp. 302–306.

[6] G. Wang, T. E. Ng, and A. Shaikh, "Programming Your Network at Run-time for Big Data Applications," in *Hot Topics in Software Defined Networks*, ser. HotSDN '12. ACM, 2012, pp. 103–108.

[7] P. Qin, B. Dai, B. Huang *et al.*, "Bandwidth-Aware Scheduling With SDN in Hadoop: A New Trend for Big Data," *IEEE Systems Journal*, vol. 11, no. 4, pp. 2337–2344, Dec 2017.

[8] D. Bonacorsi, "The CMS Computing Model," *Nuclear Physics B - Proceedings Supplements*, vol. 172, pp. 53 – 56, 2007.

[9] B. P. Abbott, R. Abbott, R. Adhikari *et al.*, "LIGO: the Laser Interferometer Gravitational-Wave Observatory," *Reports on Progress in Physics*, vol. 72, no. 7, p. 076901, 2009.

[10] W. Allcock, J. Bresnahan, R. Kettimuthu *et al.*, "The Globus Striped GridFTP Framework and Server," in *Supercomputing, 2005. Proc of the ACM/IEEE SC 2005 Conference*, Nov 2005, pp. 54–54.

[11] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky, "XROOTD-A Highly scalable architecture for data access," *WSEAS Transactions on Computers*, vol. 1, no. 4.3, 2005.

[12] I. Monga, E. Pouyoul, and C. Guok, "Software-defined networking for big-data science - architectural models from campus to the wan," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Nov 2012, pp. 1629–1635.

[13] E. Dart, L. Rotman, B. Tierney *et al.*, "The Science DMZ: A network design pattern for data-intensive science," *Scientific Programming*, vol. 22, no. 2, pp. 173–185, 2014.

[14] D. Nadig Anantha and B. Ramamurthy, "ScienceSDS: A Novel Software Defined Security Framework for Large-scale Data-intensive Science," in *ACM Security in Software Defined Networks & Network Function Virtualization*, ser. SDN-NFVSec '17. ACM, 2017, pp. 13–18.

[15] D. Tuncer, M. Charalambides, S. Clayman *et al.*, "Adaptive Resource Management and Control in Software Defined Networks," *IEEE Trans. on Network and Service Management*, vol. 12, no. 1, pp. 18–33, March 2015.

[16] W. Jeong, G. Yang, S. M. Kim *et al.*, "Efficient big link allocation scheme in virtualized software-defined networking," in *2017 13th Conf. on Network and Service Management (CNSM)*, Nov 2017, pp. 1–7.

[17] T. Zinner, M. Jarschel, A. Blenk *et al.*, "Dynamic application-aware resource management using Software-Defined Networking: Implementation prospects and challenges," in *IEEE NOMS*, May 2014, pp. 1–6.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[19] K. Cho, B. Van Merrienboer, C. Gulcehre *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing*. Assn. for Computational Linguistics, 2014, pp. 1724–1734.

[20] A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh *et al.*, "Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature," *IEEE Access*, vol. 6, pp. 14 159–14 178, 2018.

[21] M. Zhang and H. Yu, "A New Load Balancing Scheduling Algorithm Based on Linux Virtual Server," in *2013 Intl. Conf. on Computer Sciences and Applications*, Dec 2013, pp. 737–740.

[22] K. Wu, X. Wang, H. Chen *et al.*, "Improvement on LVS based IP network connection status synchronization," in *IEEE Conference on Software Engineering and Service Science*, Sept 2015, pp. 746–749.

[23] J. Wang, J. Tang, Z. Xu *et al.*, "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *IEEE INFOCOM 2017*, May 2017, pp. 1–9.

[24] P. Wang, S. Lin, and M. Luo, "A Framework for QoS-aware Traffic Classification Using Semi-supervised Machine Learning in SDNs," in *IEEE Conference on Services Computing*, June 2016, pp. 760–765.

[25] F. Tang, Z. M. Fadlullah, B. Mao *et al.*, "An Intelligent Traffic Load Prediction Based Adaptive Channel Assignment Algorithm in SDN-IoT: A Deep Learning Approach," *IEEE IoT Journal*, pp. 1–1, 2018.

[26] J. Xu, J. Wang, Q. Qi *et al.*, "IARA: An Intelligent Application-Aware VNF for Network Resource Allocation with Deep Learning," in *2018 15th IEEE SECON*, June 2018, pp. 1–3.

[27] D. N. Anantha, Z. Zhang, B. Ramamurthy *et al.*, "SNAG: SDN-managed Network Architecture for GridFTP Transfers," in *3rd Innovating the Network for Data-Intensive Science (INDIS) '16, SC16*, Nov 2016.

[28] D. N. Anantha, B. Ramamurthy, B. Bockelman *et al.*, "Differentiated network services for data-intensive science using application-aware SDN," in *2017 IEEE ANTS*, Dec 2017, pp. 1–6.

[29] W. Allcock, J. Bresnahan, K. Kettimuthu *et al.*, "The globus extensible input/output system (XIO): a protocol independent IO system for the grid," in *19th IEEE IPDPS*, April 2005, p. 8.

[30] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. springer, 2016.

[31] W. Zhang and W. Zhang, "Linux Virtual Server Clusters: Build highly-scalable and highly available network services at low cost," *Linux Magazine*, vol. 11, 2003.

[32] Elastic Stack. [Online]. Available: https://www.elastic.co/products

[33] S. Hochreiter, Y. Bengio, P. Frasconi *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[34] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," vol. 9, pp. 1735–80, 12 1997.

[35] A. Gers F., J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," Tech. Rep., 1999.

[36] K. Greff, R. K. Srivastava, J. Koutnik *et al.*, "LSTM: A Search Space Odyssey," *IEEE Trans.Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct 2017.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[38] G. E. Box, G. M. Jenkins, G. C. Reinsel *et al.*, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.