HPCGCN: A Predictive Framework on High Performance Computing Cluster Log Data Using Graph Convolutional Networks

1st Avishek Bose
Department of Computer Science
Kansas State University
Manhattan, Kansas, USA
abose@ksu.edu

2nd Huichen Yang
Department of Computer Science
Kansas State University
Manhattan, Kansas, USA
huichen@ksu.edu

3rd William H. Hsu
Department of Computer Science
Kansas State University
Manhattan, Kansas, USA
bhsu@ksu.edu

4th Daniel Andresen
Department of Computer Science
Kansas State University
Manhattan, Kansas, USA
dan@ksu.edu

2021 IEEE International Conference on Big Data (Big Data) | 978-1-6654-3902-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/BigData52589.2021.9671370

***Abstract*—This paper presents a novel use case of Graph Convolutional Network (GCN) learning representations for predictive data mining, specifically from user/task data in the domain of high-performance computing (HPC). It outlines an approach based on a coalesced data set: logs from the Slurm workload manager, joined with user experience survey data from computational cluster users. We introduce a new method of constructing a heterogeneous unweighted HPC graph consisting of multiple typed nodes after revealing the manifold relations between the nodes. The GCN structure used here supports two tasks: i) determining whether a job will complete or fail and ii) predicting memory and CPU requirements by training the GCN semi-supervised classification model and regression models on the generated graph. The graph is partitioned into partitions using graph clustering. We conducted classification and regression experiments using the proposed framework on our HPC log dataset and evaluated predictions by our trained models against baselines using test_score, F1-score, precision, recall for classification, and R1 score for regression, showing that our framework achieves significant improvements.**

***Index Terms*—HPC, Slurm, GCN, Beocat, ReqCPUS, ReqMem, CPU Usage, Memory Usage**

I. Introduction

Prior work on analytics of high-performance computing (HPC) scheduling systems has shown that requested allocation of resources such as memory, CPU cores, CPU time, and number of clusters, as well as self-reported user experience with HPC systems, are relevant to predicting the successful completion or termination of any job submitted to an HPC cluster [11]. The sufficiency of requested resources and competency of users' experience level of using HPC systems determine whether the job will succeed. Again, the ability to predict resource requirements for a job to successfully execute in the cluster is necessary before a user submits a job [12], [14]. In HPC systems, such open-source software packages as the Sun Grid Engine (SGE) [2] and Slurm [3] allow users and managers of computing clusters to monitor the status of real-time jobs and efficiently allocate requested system resources. However, we are still in immense need to automate the process of HPC resource allocation for submitted jobs because none of the existing cluster resource management tools provide this feature. On the other hand, several previous studies [4]–[8] show that effective use of machine learning techniques strengthens decisions support based on estimating computational resource needs to ensure a job is successfully completed by training learning models on historical log data. Although user experience has a part to play in using HPC systems efficiently by offering an additive gain along with training with the respective HPC datasets, user experience data is not easily obtainable. Therefore, training a classification model and a regression model with user experience data must account for the scarcity of representative data. However, supervised classifier and regression models need a large amount of annotated/labeled data for training prediction models to provide high accuracy; prediction models cannot be confirmed as well-performing using fewer user data for predicting a job status and its resource needs. For this reason, we sought to use a semi-supervised approach using GCN to predict job status and estimate vital computational resources.

In this study, we introduce a graphical data model of the HPC ecosystem with the historical log data of *Beocat*, which is the primary HPC platform at Kansas State University.

Beocat maintains a queue for submitted jobs; for each submission, the *Beocat* system requires user inputs of estimated running time and amount of memory for each job. User experience with the HPC ecosystem may positively affect prediction accuracy for the status of each submitted job as well as estimates of necessary computational resources. Collecting user experience data, a recently added optional feature in *Beocat*, helps answer some additional questions related to user experience: i) how much experience does the user have with an HPC system; ii) what rating do users give themselves; iii) what courses or formal training do users have with HPC. After obtaining the necessary information, the system prompts users to schedule jobs based on the requested resources and system resource availability.

We have found a major limitation of the current job management system of *Beocat*, similar to that exhibited by many other HPC systems, is that user-specific information regarding their level of experience and training is not collected. These properties are thus unavailable as user modeling features to be associated with resource allocation data at job submission time. In addition, estimating the resources needed is an error-prone task in HPC systems, one where even many trained and experienced users have difficulty in confirming actual resource needs. Moreover, underestimating necessary computational resources makes a failure of a job more probable, which in turn, can further waste resources by pushing back other jobs in the queue that require access to the occupied resources.

Therefore, given these issues, neither an HPC management system nor any current machine learning model can effectively use the heterogeneous data to predict job status and estimate resource needs. This provides the motivation for research into applying a Graph Convolutional Network (GCN) to construct a classification model that can predict the status of submitted jobs and create two regression models to estimate the computational resources required for a job. Many studies show that GCN outperforms CNN, RNN, and other traditional machine learning approaches, and GCN can also provide personalized recommendations for users in estimating resources.

Traditional machine learning classifier and regression algorithms show promise with a grid-based dataset where data do not consider any relations between the data rows. These algorithms do not, however, produce highly accurate results when trained on graph-based data such as an HPC historical log. The *Beocat* HPC dataset can be considered as a graph dataset because it contains user experience and aggregated demographic information that supports the presence of implicit relationships between experience metrics, resource allocation, and historical user profile information [10]. Other than having independent data points, GCN exploits following latent relationships i) among users with similar levels of expertise with an HPC system; ii) among users from the same project or department; iii) among jobs from the same user. As for being a semi-supervised algorithm, GCN also addresses the limitations of inadequate data on user experience, which we mentioned earlier. Our proposed model contributes to the research in the field in the following ways:

- We introduce a new method for constructing a heterogeneous graph from a historical HPC log dataset.
- To the best of our knowledge, we are the first to train a GCN model on HPC data with user experience information to predict job status and estimate computational resources.
- We show GCN on HPC achieves significant results in the classification task with high precision, recall, F1-score, and high R1 score for regression tasks.

We organize the rest of this paper with the following contents i) provide references and discuss earlier research in this domain, ii) data model generation and proposed approach for applying GCN, iii) experiments, and evaluations of performance, iv) result discussion, and finally, v) recommendations for future research and a conclusion.

II. Background and Literature Study

The unique structure of the Deep Learning Networks (DNN), and the steep rise in DNN usage in countless applications distinguish Machine Learning (ML) research in two timelines: i) conventional ML and ii) DNN approaches. In this section, we discuss the basics of conventional ML and DNN and how they have been used in the HPC domain.

A. *Conventional ML approaches in HPC*

We use conventional ML to refer to various supervised approaches such as Linear Regression, Ridge Regression, and Lasso Regression for regression analysis and Logistic Regression, Gaussian Naive Bayes, and Random Forest for classification analysis. These algorithms need training data and target variables to train models where the models compute errors on their predictions using various loss functions based on the difference between predicted values and actual target values. Test data fed to a regression model predicts a numeric value while a classification model predicts a target class for test data. Research works on conventional approaches had difficulties extracting sufficient discriminative deep features to efficiently perform the learning task. As a result, the predictive output may not be satisfactory. Moreover, all earlier research [8], [9] applied to the SGE log dataset instead of the Slurm dataset; Slurm is a state-of-the-art HPC resource management tool.

B. *DNN approaches in HPC domain*

The advent of DNNs has revolutionized many predictive tasks in both academics and real life. CNNs are a type of DNN consisting of artificial neurons organized in layers and responsible for propagating information layer by layer, ultimately reaching the output layer. There are three types of layers: i) input, ii) hidden, and iii) output. There are weight parameters between these layers that can be learned by the gradient descent of the loss calculated during backpropagation. Although DNN has countless usability, only a few DNN approaches have been adapted to the HPC predictive analytics domain [13]. However, DNNs are also limited in extracting implicit relationships between data points because DNN, by nature, considers data points independent of each other. That means, the data points can also be considered as independent and identically distributed (IID). When HPC historical log data joins with user experience data, we attain the leverage to establish the implicit relationships between data points. Therefore, we sought a DNN architecture that considers not only data features but also implicit relationships. This intuition inspired us to adopt GCN that considers data-node features with node interactions obtained from a graph constructed using the HPC log data.

III. Data model generation and proposed framework

In this section, we discuss how we proposed to apply a GCN model to *Beocat* log data to predict the class of a submitted job and to estimate computational resources necessary to successfully complete a job. To articulate our research, we divided this section into two subsections following the two main steps of implementing GCN: i) graph construction and ii) model training.

A. Graph data construction and methodology

We constructed a heterogeneous HPC data graph comprising various types of data nodes with different types of relationships between nodes. We used the Networkx library [19] to construct the graph using the obtained relationships that are considered as edges between nodes. The following two subsections provide the details of the constructed HPC graph.

1) Dataset Pre-processing: Beocat records accounting data for all jobs submitted and executed on the Beocat HPC system. We obtained the current dataset from the Beocat Slurm log data; data collection is described in Subsection IV A, Dataset. The dataset has 112 attributes that roughly record most details for all submitted jobs, such as whether a submitted job has been executed successfully, resources required by a user during submission, and resources allocated to the job itself by the system. However, the raw dataset must be cleaned because it has redundant attributes and some missing attribute values. The dataset is cleaned using the following strategies:

- Remove duplicate attributes: Remove the attributes of allocated CPUs and the number of nodes because the attribute AllocTres already includes these two attributes
- Remove the attributes that have only two values or only have value of NaN
- Remove the jobs if job states are neither failed nor completed; some submitted jobs time out, are canceled or fall under other states
- Parsing and Type Casting non-numerical attribute values to numeric values: To obtain representations of node features for graph nodes, we need numeric values on certain attributes that are typecast after being transformed from non-numeric values (i.e., *MaxVMSize*, *MaxRSS*, *AveVMSize*, *AveRSS*, *AssocID*, *ReqCPUS*, and *AvePages*).
- Scaling attribute value: Attribute values in different scales are transformed into a single scale e.g., all the *ReqMem* entries are transformed into a single scale.
- Extracting information from attributes: The three consecutive data rows in the dataset with a same jobID are produced after joining with LDAP data (described in the following subsection) represent an instance of a single job submitted into *Beocat*. To obtain a single value for each attribute of a submitted job, we used different group aggregation functions such as *first*, *max*,and *last*.

2) User Data Integration: User information (e.g., department, project title) are also important attributes that can help identify the relationship between user behavior and submitted jobs, but this information is not included in the primary raw *Beocat* dataset. Information about users can be collected using public services such as the Lightweight Directory Access Protocol (LDAP) [1] command on the HPC system. Other than user information, we also consider user proficiency in using the HPC system as an applicable attribute indicating user experience in submitting jobs. For this reason, we provided three survey questions for active users during job submission time and collected user feedback as additional user-related attributes. The survey questions are presented in Table I:

TABLE I
SURVEY QUESTIONS AND OPTIONS

Questions	Options
q1: How much experience do you have with high performance computing (HPC)?	a. 0-6 month b. 6-12 month c. 1-2 years d. 2-5 years e. more than 5 years
q2: Rate your own proficiency using HPC	a. Novice b. Fairly okay c. Average d. Very good e. Proficient
q3: How many courses or formal training have you had in HPC?	a. None b. Single one-day courses c. Week-long training d. Multiple courses

3) Node creation: We extracted all the distinct entities from: *JobID*, *UID*, and *GID*; all the entities from these attributes in the dataset are defined as distinct nodes. This step made the graph heterogeneous in having different types of nodes. To make a graph compatible with GCN learning, we mapped distinct entities derived from these attributes to unique numeric values. In other words, each node is represented by a unique integer. To create nodes for the HPC graph, we scan the edge relationships between these nodes using Networkx [19], which automatically constructs the graph for training.

4) Edge formulation: We extracted and explored the relationships between different nodes where the nodes were mapped to distinct entities from the attributes and connected by undirected edges. Node connections were assigned for each data row according to the following relationships: *UID* → *JobID*, *GID* → *UID*. We also applied binning operations to the values of each of the following attributes: *ReqMem* and *ReqCPUS*, *q1*, *q2*, and *q3*; these operations were an attempt to connect distinct *JobID* nodes to each other. The interval ranges of all the bining operations were determined by HPC domain experts. Table III and Table IV show the five attributes where the binning operation was applied. The *Limit* column defines the range of each bin, *Container* column defines the binning group where *JobID* nodes have undirected edges. Please note that each GCN model shows a decrease in predictive performance after a certain average value of node degree is observed. In other words, a graph topology in which existing nodes having too many incoming and outgoing edges can converge toward a similar vector representation. As a result, the lack of discriminative feature representation of nodes causes performance to decline in the predictive model. To address this problem, we created a dummy node initialized with a random vector representation and connected all job nodes in a container to that node by edges to keep the number of average node degrees limited in the graph.

5) Node feature formulation: We selected several attributes from the *Beocat* dataset to be considered features of job nodes. We also added some derived features by averaging certain current attributes (*CPUTimeRAW*, *MaxVMSize*, *TimelimitRaw*, *ReqMem*,and *MaxRSS*) for each user. To obtain average values, we grouped across jobs by *UID* and projected the newly generated columns as *aCPUTimeRAW*, *aMaxVMSize*, *aTimelimitRaw*, *aReqMem*, and *aMaxRSS*. Then, we re-joined the existing attributes with the newly derived attributes by *UID* that results in each row for each job. That means values of the derived attributes were mapped to their respective *UID*s where those values were replicated across all the jobs submitted by that *UID*. Table II shows the distinct attributes based on their categories, whether numeric, categorical, or aggregated. Ultimately, we had numeric entries in all 32 different attributes that served as feature representations for submitted jobs. For predicting memory usage, we excluded MaxRSS, and for predicting CPU usage, we excluded *CPUTimeRAW* following the convention of regression analysis for considering them as target variables. Now, we could map each job node to a valid feature representation, but we had other types of nodes as well in the HPC graph. To address this limitation, for all other nodes except job nodes, we initialized a node by generating a random vector of the same length such as a job node feature representation. To train the GCN model efficiently, we normalized all attribute columns using min-max normalization.

TABLE II
CONSIDERED NUMERIC ATTRIBUTES FOR NODE FEATURES

Numeric Attributes
TimelimitRaw, *ReqMem*, *NCPUS*, *NNodes* *AveVMSize*, *AveRSS*, *MaxVMSize* *ReqCPUS*, *q5*, *q6*, *q7*, *CPUTimeRAW*, *AvePages*
Categorical Attributes
department_ComputerScience *department_ChemicalEngineering*, *department_Chemistry*, *department_Mechanical&NuclearEngineering*, *department_Physics*, *department_PlantPathology*, *department_ComputerScience*, *department_Agronomy*, *department_InstituteforEnvironmentalResearch*, *role_Faculty*, *role_GraduateStudent*, *role_PostDoctoralResearcher*, *role_ResearchAssociate*, *department_VeterinaryDiagnosticLaboratory*
Aggregate Attributes
aMaxVMSize, *aTimelimitRaw*, *aReqMem*, *aMaxRSS*, *role_UndergraduateStudent*

6) Sample structure of a Graph Convolutional Network: Figure 1 presents a prototype of a heterogeneous graph structure based on the user-job information extracted from the given **Beocat** HPC dataset. The label for *userID* nodes is U, for *JobID* nodes is J, and for *GID* nodes is G. Each is shown in a different color as well. Distinct nodes connected through edges are derived from explicit and implicit relationships. Implicit relationships such as jobIDs in the same container, as defined in Table III, are connected to a dummy node assigned for each container. For example, α and β were assigned to two *reqCPUS* containers, γ and δ were assigned to two *reqMem* containers, and χ, ψ, and ω were assigned to three different containers for three different questions *q1*, *q2*, and *q3* respectively. We provided target labels *Failed* and *Completed* for JobID nodes in the *State* target variable for classification analysis. For two regression analyses, one target variable we had is *CPUTimeRAW* and another one is *MaxRSS*. Figure 2 magnifies a *JobID* from the HPC graph to illustrate its structure.

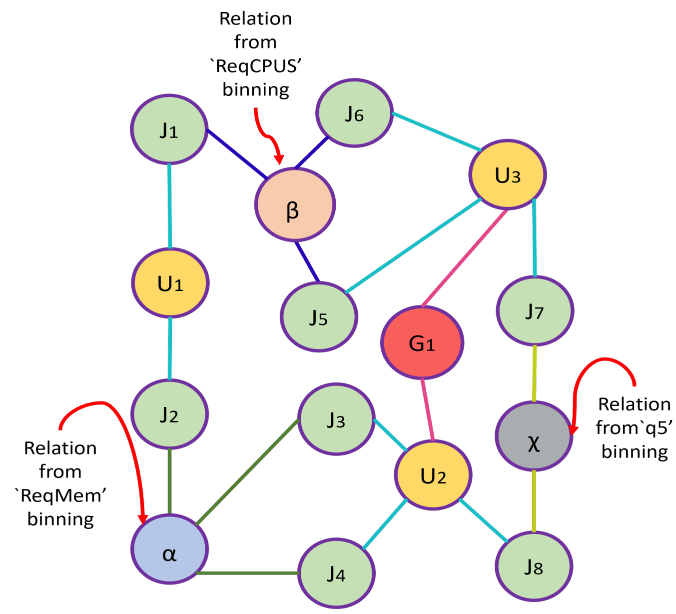

Fig. 1. Prototype of the proposed HPC graph

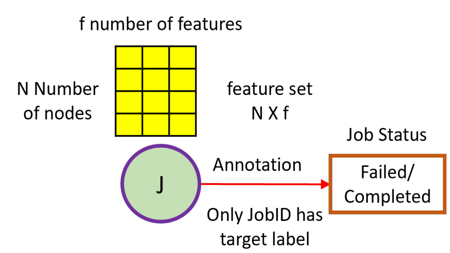

Fig. 2. A sample JobID node with features

B. Training GCN on the graph

The constructed heterogeneous graph was then used to train each of three different GCN training models: one job status classification model and two regression models. In our implementation, we used the Cluster GCN architecture [15] from pyTorchGeometric open-source graph learning library [16] that partitions the single large graph into smaller sub-graphs using *metis* [17] graph-portioning algorithm. The graph partitioning algorithm applied here uses the same process as data batches are rendered in a CNN. The GCN model [18] was developed as a multi-class classification model, but in this work, we employ it to predict job status from any of the two classes (Failed and Completed). The classification model discriminates each node feature at the end of a certain number of epochs to effectively classify job nodes on behalf of predicting job status where we initialized node features with the feature matrix of all transformed numeric attributes. On the other hand, two different target variables (*CPUTimeRAW* and *MaxRSS*) correspond to two different regression models, are used to predict target values where each model also learns discriminative features but in fewer epochs compared to the classification model. The reason for this is that regression analysis performs better with a simpler learning setup.

TABLE III
BINNING RANGE FOR REQCPUS AND REQMEM

ReqCPUS		*ReqMem*	
Limit	Container	Limit	Container
256 < *ReqCPUS* ≤ 512	A	256 < *ReqMem* ≤ 512	C
512 < *ReqCPUS*	B	512 < *ReqMem*	D

TABLE IV
BINNING RANGE FOR THE USER EXPERIENCE QUESTIONS

Questions		
Question Types	Limit	Container
q1	*q1*==5	E
q2	*q2*==5	F
q3	*q3*==4	G

IV. EXPERIMENTS AND PERFORMANCE EVALUATION

A. Dataset

The raw dataset spans four years of job submission history from February 2018 to February 2021; the dataset has 26.6 million instances and 112 data rows. In our study, for classification analysis, we selected a sample dataset of 200K data rows where we found 56837 distinct nodes and 93866 edges. In addition, the heterogeneous graph consists of 56727 JobID nodes, 55 GID nodes, 55 UID nodes. The average degree of the nodes is 3.30. Splitting the dataset as 80%:20% generates 45470 nodes for training and 11367 nodes for testing the model. For regression analysis, we selected 100K data rows as a sample dataset from the original raw dataset.

B. Experiments

Because we have three different GCN models, we used three different experimental configurations for training them. Model architectures and hyperparameter setup for classification and regression analyses are discussed for each model individually.

1) Classification Analysis: For classification, we used three consecutive fully connected hidden layers; the sizes of each hidden layer were [(number of all nodes, 64), (64, 64), (64, 64), (64, 2)]. We applied Negative Log-Likelihood as a loss function and log_softmax as an activation function in the last layer of the classification network. Furthermore, we ran the model for 400 epochs with the following parameter settings; learning rate = 0.01, and dropout = 0.5.

2) Regression Analysis: For both regression analyses, we used one fully connected hidden layer with the following shape of the layer [(number of all nodes, 64), (64, 1)]. We used Mean Squared Error and the Linear activation function to run our models for 100 epochs where the learning rate = 0.1, and dropout = 0.5. We used a simpler model in both regression analyses than the classification model; simpler regression models perform better than complex networks. We trained the two regression models only on those data rows in which job status is successfully complete.

C. Performance Evaluation

In Table V, we display the classification result compared against two baseline classification algorithms i) Naive Bayes, and ii) Logistic Regression. As these algorithms are meant for binary classification, we employ these to predict whether a submitted job would be successful or not. We found that our proposed GCN framework achieves performance gain over the baselines in terms of prediction accuracy, precision, recall, and F1-score by significant margins.

TABLE V
CLASSIFICATION RESULT

Classification Model	Accuracy	Precision	Recall	F-1 Score
Proposed GCN	82%	0.814	0.975	0.88%
Logistic Regression	73.7%	0.37	0.50	0.42%
Naive Bayes	74%	0.37	0.51	0.43

Table VI projects the results of two regression models for CPU and memory using GCN respectively. We compare our regression performance against two baselines regression algorithms i) Lasso Regression, and ii) Ridge Regression using R1 score. We also observe here, the GCN regression model that predicts CPU usage outperforms the baselines. However, another GCN regression model for memory usage, attains performance equal to that of the Ridge Regression algorithm.

TABLE VI
REGRESSION RESULT FOR CPU USAGE

Regression Model	R-1 Score
Proposed Model	0.26
Linear Regression	0.2450
LassoLarsIC Regression	0.2457
ElasticNetCV Regression	0.0029
Ridge Regression	0.2543

Fig 3, Fig 4, and Fig 5 show loss curves for the GCN classification model on predicting job status, a regression model on CPU usage, and another regression model on memory usage, respectively.

TABLE VII
REGRESSION RESULT FOR MEMORY USAGE

Regression Model	R-1 Score
Proposed Model	0.14
Linear Regression	0.0410
LassoLarsIC Regression	0.0415
ElasticNetCV Regression	0.0107
Ridge Regression	0.1468

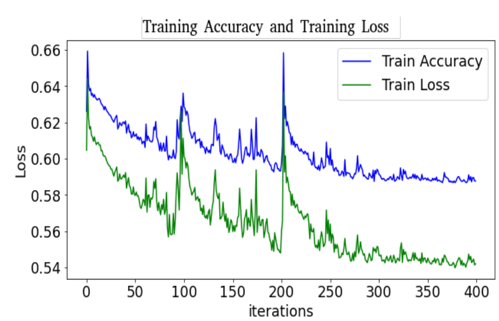

Fig. 3. Training Accuracy and loss curve for job status classification

V. RESULTS AND DISCUSSION

Based on the performance result of GCN models on HPC data, we can anticipate that job data rows are not fully independent episodes or IID. The reason behind this anticipation is that the implicit relations of jobs considered in

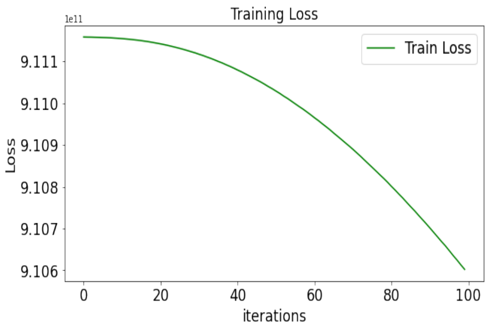

Fig. 4. Training loss curve for required CPU usage estimation

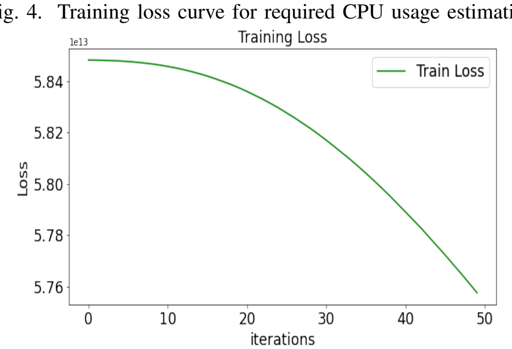

Fig. 5. Training loss curve for required memory usage estimation

GCN models contribute to deploying better prediction models. Again, for a big graph, after a certain number of epochs, the node representations in the graph tend to converge to similar distributions due to the message passing process. On the other hand, if an appropriate node relation setup policy is taken into account to construct a graph, GCN delivers sufficient predictive capabilities without even being initialized by any external node feature set.

VI. Conclusions and Future Work

In this paper, we focused on adapting GCNs in the HPC domain to exploit the potential of GCNs for predictive analytic tasks in HPC user modeling. We have found GCN shows significantly better performance than the baselines on standard performance matrices. However, GCN limitation in having high computational overhead for graph construction and network training is not considered in this analysis. We still face challenges taking advantage of the user data because the data collection was not verified and that process itself makes the process error-prone. Constructing an effective HPC graph is also a matter of prime concern in obtaining good predictions. We have, however, vindicated our suggestion of conceptualizing HPC data as a graph model through experimental analysis and we have much room to improve the model further. We plan to apply our system to a larger sample dataset after addressing the computational overhead and scalability limitations of GCNs with respect to the problem domain reported in this paper. We can also include other variables of relational importance. We also want to apply the HPC data to other types of GCNs to examine their performance gain.

Acknowledgments

We deeply appreciate the contributions by Mohammed Tanash and Brandon Dunn. We are also thankful to the HPC staff at Kansas State University including Adam Tygart, Kyle Hutson, and Jonathon Anderson for their help and technical support. This research was supported by NIH award P20GM113109, and Kansas State University.

References

[1] Howes, Tim, and Mark Smith. LDAP: Programming directory-enabled applications with lightweight directory access protocol. Sams Publishing, 1997.

[2] Gentzsch, Wolfgang. “Sun grid engine: Towards creating a compute power grid.” In Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 35-36. IEEE, 2001.

[3] Slurm: A Highly Scalable Workload Manager, https://github.com/SchedMD/slurm (2018).

[4] Li, Xiuqiao, Nan Qi, Yuanyuan He, and Bill McMillan. “Practical resource usage prediction method for large memory jobs in hpc clusters.” In Asian Conference on Supercomputing Frontiers, pp. 1-18. Springer, Cham, 2019.

[5] Ferretti, Marco, and Luigi Santangelo.“Cloud vs On-Premise HPC: a model for comprehensive cost assessment.” Parallel Computing: Technology Trends 36 (2020): 69.

[6] Gainaru, Ana, Brice Goglin, Valentin Honoré, and Guillaume Pallez. “Profiles of upcoming HPC Applications and their Impact on Reservation Strategies.” IEEE Transactions on Parallel and Distributed Systems 32, no. 5 (2020): 1178-1190.

[7] Witt, Carl Philipp. “Predictive Resource Management for Scientific Workflows.” (2020).

[8] Adedolapo, Okanlawon, Yang Huichen, Bose Avishek, Hsu William, Andresen Dan, and Tanash Mohammed. “Feature Selection for Learning to Predict Outcomes of Compute Cluster Jobs with Application to Decision Support.” In 2020 International Conference on Computational Science and Computational Intelligence (CSCI), pp. 1231-1236. IEEE, 2020.

[9] Andresen, Dan, William Hsu, Huichen Yang, and Adedolapo Okanlawon. “Machine learning for predictive analytics of compute cluster jobs.” arXiv preprint arXiv:1806.01116 (2018).

[10] Bobadilla Dias, Luis Enrique. “Graph mining for role extraction in predictive analytics of high-performance computing systems.” PhD diss., 2020.

[11] SANTANGELO, Luigi, and Marco FERRETTI. “HPC and Cloud Computing.”

[12] Kumar, Rakesh, Saurabh Jha, Ashraf Mahgoub, Rajesh Kalyanam, Stephen Harrell, Xiaohui Carol Song, Zbigniew Kalbarczyk, William Kramer, Ravishankar Iyer, and Saurabh Bagchi.“The mystery of the failing jobs: Insights from operational data from two university-wide computing systems.” In 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 158-171. IEEE, 2020.

[13] Han, Jingoo, M. Mustafa Rafique, Luna Xu, Ali R. Butt, Seung-Hwan Lim, and Sudharshan S. Vazhkudai. “Marble: A multi-gpu aware job scheduler for deep learning on hpc systems.” In 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), pp. 272-281. IEEE, 2020.

[14] Xu, Huanle, Yang Liu, and Wing Cheong Lau. “Optimal Job Scheduling With Resource Packing for Heterogeneous Servers.” IEEE/ACM Transactions on Networking (2021).

[15] Chiang, Wei-Lin, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. “Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks.” In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 257-266. 2019.

[16] Fey, Matthias, and Jan Eric Lenssen. “Fast graph representation learning with PyTorch Geometric.” arXiv preprint arXiv:1903.02428 (2019).

[17] Karypis, George, and Vipin Kumar. “METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices.” (1997).

[18] Kipf, Thomas N., and Max Welling. “Semi-supervised classification with graph convolutional networks.” arXiv preprint arXiv:1609.02907 (2016).

[19] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, “Exploring network structure, dynamics, and function using NetworkX”, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008