THE UNIVERSITY OF CHICAGO

Towards Continually Learning Application Performance Models

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
MASTER

DEPARTMENT OF COMPUTER SCIENCE

BY

Ray A. O. Sinurat

CHICAGO, ILLINOIS

2024

Copyright © 2024 by ***Ray A. O. Sinurat***

All Rights Reserved

Intentionally left blank.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I thank Haryadi Gunawi and Sandeep Madireddy for their insights, feedbacks, and supports in this work. Lastly, I express my gratitude to my family and friends for their love and support.

Abstract

Machine learning-based performance models are increasingly being used to build critical job scheduling and application optimization decisions. Traditionally, these models assume that data distribution does not change as more samples are collected over time. However, owing to the complexity and heterogeneity of production HPC systems, they are susceptible to hardware degradation, replacement, and/or software patches, which can lead to drift in the data distribution that can adversely affect the performance models. To this end, we develop continually learning performance models that account for the distribution drift, alleviate catastrophic forgetting, and improve generalizability. Our best model was able to retain accuracy, regardless of having to learn the new distribution of data inflicted by system changes, while demonstrating a $2\times$ improvement in the prediction accuracy of the whole data sequence in comparison to the baseline approach.

Chapter 1

Introduction

The complexity of leadership-class high-performance computing (HPC) systems is increasing rapidly due to the need to handle diverse workloads and applications. In particular, storage systems and I/O architectures are integrating different heterogeneous storage technologies to maximize the price-performance trade-off. This complexity entails the need for sophisticated empirical/machine learning models to accurately predict the application performance. The accuracy of the performance model is crucial due to its use in making decisions about job scheduling, application optimization, and capacity planning of facilities.

It is commonly assumed that the data used to learn the performance models do not undergo any distribution shift. Under this stationarity assumption, the observation of more data on the application performance in the HPC system should increase the predictive performance of these machine learning models. However, as reported in recent works [27], factors such as hardware degradation [16], replacement, anomalies [14] or software upgrades [20] can affect the state of the system and, in turn, lead to a change/drift in the underlying distribution. Performance models trained on the past data will be degraded due to this data distribution shift. Such distribution shifts have been handled either by updating the model *ad hoc* without drift detection [12] using a sliding window of data or by ignoring those data before the drift occurred. More recently, Madireddy et al. [24] proposed a moment-matching transform to correct for data drift post-detection.

A more general approach to train/adapt the performance model in the presence of this drift is using continual learning [31] algorithms, which acknowledge the presence of data distribution shifts and are designed to prevent catastrophic forgetting of the models on the data observed before the shift when training on data observed post-drift, thus generalizing across the distribution shifts. The ability to learn continuously from an incoming data stream without catastrophic forgetting is critical for designing intelligent systems.

Prior research in continuous learning has focused mainly on *virtual concept drift* [18] (or label drift) where there exists drift in the distribution of targets ($P(y)$) without affecting the functional relationship between the inputs and outputs ($P(y|x)$) of the model. For example, this scenario is encountered in training classification models, which should learn a new class without forgetting the previous ones when presented sequentially, one after the other. However, for performance modeling, we are interested in the *real concept drift* scenario in which learning occurs in a sequence of tasks where the input distribution ($P(x)$) remains the same but the functional relationship ($P(y|x)$) changes across tasks due to the change in the state of the system. This scenario has been less explored with few studies pertaining to image segmentation [8] and improvement in label precision [5].

To this end, we make the following contributions to this work:

1. We formulate performance modeling in the presence of data drifts as a *real concept drift* continuous learning scenario.

2. We adapted several *virtual concept drift* continuous learning approaches to performance modeling and compared to naive fine-tuned learning that ignores catastrophic forgetting.

3. Our results show a **2-fold** improvement in the accuracy of the continual learning models compared to their naive fine-tuned learning counterparts after learning all tasks on the real performance data from the production clusters.

Chapter 2

Dataset

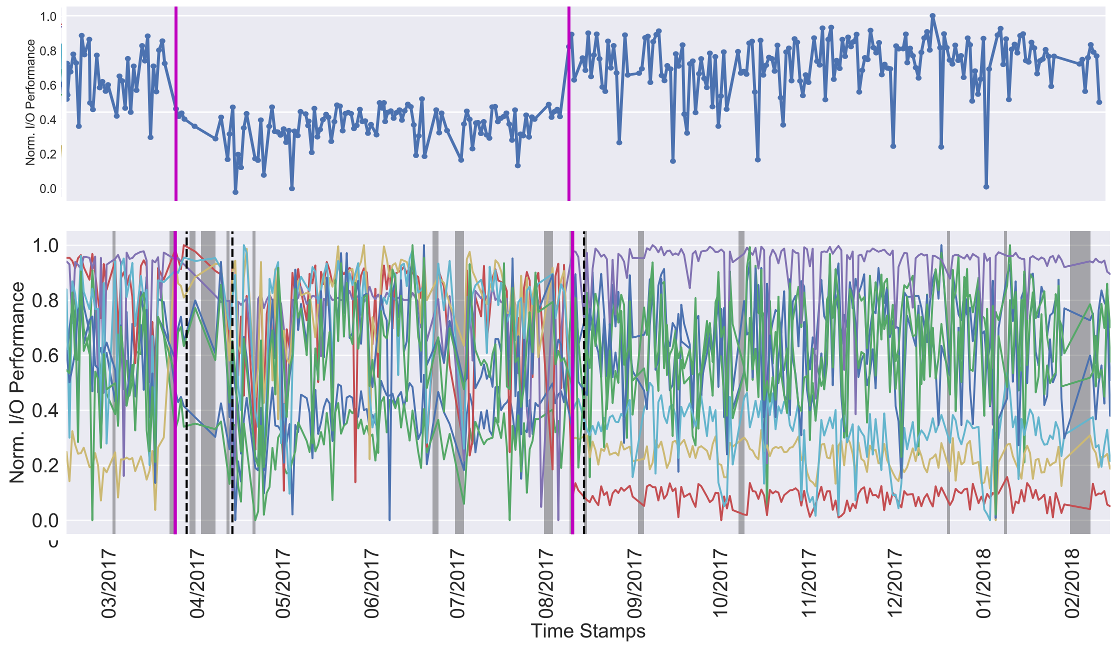


Figure 2.1: **I/O performance as a function of time.** *I/O performance as a function of time for all eight applications (bottom) and one of the applications (top); Magenta lines show the temporal location of the software upgrade.*

We collected our data from Cori, a production supercomputing system at the National Energy Research Scientific Computing Center that is used to run high-performance applications (HPC) tasks. Cori is a Cray XC40 system that comprises of 12,000 compute nodes and Lustre file systems [3] with a 30PB capacity and peak performance of 700 GB/sec. The data used in our experiments consist of eight production applications with diverse application workloads representative of the science application typically run. Numerous applications employ distinct write and read variations, with one typically being more intensive than the other, to measure the I/O performance of the related tasks. The mentioned applications are as follows:

1. **IOR Posix and IOR MPI-IO** [2]: These tools perform read and write operations to assess the I/O performance of parallel file systems. In total, 1.5 TB data were processed for IOR benchmark.

2. **HACC Posix** [15]: This operates 8 TB data in read and write modes and is utilized for simulating the formation of gravitational structures in collisionless fluids in an expanding universe.

3. **DBSCAN MPI-IO** [29]: Processes 3 TB volume of data in read mode and is applied for clustering extensive datasets.

4. **VPIC MPI-IO** [4]: Functions in write mode and serves as a versatile, particle-in-cell simulation tool for kinetic plasma modeling with total amount of data reaching 4 TB.

To collect the I/O performance of these application, we used TOKIO (Total Knowledge of I/O; [21]) framework for I/O performance profiling. As part of TOKIO, the application performance statistics is provided by Darshan [7] logs, I/O traffic in Lustre file systems is obtained using LMT [13] (Lustre Monitoring Tools), and scheduling information using Slurm [32].

All of these applications were run daily for one year. These benchmarks also executed contemporaneously on Cori; hence, the temporal location of the system changes should be consistent across the applications. Specifically, two major software upgrades were performed during the course of this study. Both upgrades affected the I/O performance of some applications. Therefore, it can be assumed that the data are divided into three different windows by the two software upgrades as seen in Figure 2.1 that show the normalized I/O performance for a single application

Table 2.1: **Feature variables.** *Input features extracted from the I/O monitoring data.*

Metric	Description	Units	Tool
Perc_OST_Full	Average % OST fullness	–	LMT
Ave_OSS_CPU	Average OSS CPU load	–	LMT
Ave_MDS_CPU	Average MDS CPU Load	–	LMT
Num_Conc_Jobs	Number of concurrent jobs	–	Slurm
FS_Read_Vol	Total read volume across FS	GB	LMT
FS_Read_Vol	Total write volume across FS	GB	LMT
Num_Mkdir_Op	Number of mkdir operations	–	LMT
Num_Rename_Op	Number of rename operations	–	LMT
Num_Rmdir_Op	Number of rmdir operations	–	LMT
Num_Unlink_Op	Number of unlink operations	–	LMT

at the top and all eight applications at the bottom. The magenta line indicates the time that the software updates were performed, and the top figure shows the shift in the I/O performance distribution. We consider metrics that capture filesystem traffic (using LMT) and system load (using Slurm) as features used by the performance model to predict I/O performance. We also do not preprocess these features to choose a subset of these metrics that are not collinear. A summary of the adopted features is shown in Table 2.1.

Chapter 3

Continually Learning Performance Models

3.1 Definition

The goal of continual learning (CL) is to improve the model over time while retaining prior knowledge or experiences. CL came to fruition as a solution when the model forgets prior knowledge due to a dynamic data stream that changes over time [18]. This change is typically in the data distribution which leads to data drift/non-stationarity and ignoring it can lead to catastrophic forgetting [11]. Formally, continual learning on a sequence of tasks ($T^t, t = 1, 2, .., N \; \forall T \in \mathscr{T}$) consisting of ordered pairs of input data points and their corresponding targets $\{\mathscr{X}^t, \mathscr{Y}^t\}$ aims to maximize the performance of a system across all tasks $\mathscr{T}$, when trained sequentially. There are different kinds of data drifts that introduce non-stationarity in learning and motivate the need for continual learning approaches. Some well-known drifts that affect supervised learning include:

1. **Real concept drift** occurs when the distribution of inputs remain same across tasks, i.e, $P_t(x) = P_{t+1}(x), \forall x \in \mathscr{X}$ but the functional relation changes, i.e., $P_t(y|x) \neq P_{t+1}(y|x), \forall (x,y) \in (\mathscr{X}, \mathscr{Y})$.

2. **Virtual concept drift** happens when the distribution shifts only happen inside target variables ($P_t(y) \neq P_{t+1}(y)$) without altering the relationship towards input variables.

3. **Domain drift** exists if the drift occurs in the input variables ($P_t(x) \neq P_{t+1}(x)$) without affecting the relationship towards target variables.

We observed that the I/O performance data undergoes *real concept drift*, which is much harder and less studied in the context of continual learning. Most works have considered *virtual concept drift* in the classification scenario where data from disjoint classes form new tasks. Although I/O performance modeling is a regression problem, we formulate it as a classification problem by dividing the continuous output range into an equally spaced interval for simplicity, in addition to the potential to adapt the high-performing algorithms from the literature as well as software that have been primarily targeting the classification scenarios.

3.2 Formulation of Learning with Real Concept Drift

To this end, we adopt the classification-based continual learning settings to learn in the presence of real concept drift. Specifically, we incorporated the class-incremental setting [26] from *virtual concept drift* works but modified the learning so that each task's output spans the entire space as opposed to each of them accumulating from disjoint classes in the previous case. In addition to that, we keep the *task id* in training and inference consistent with the traditional class-incremental learning, where the *task id* is provided at training, but not inference. In this work, the *task id* is an integer that provides essential information when the software upgrade occurs. We also note that as we described in Chapter 2, we used the aggregation of performance statistics provided by Darshan [7] as our target variable (y) which results in a single floating number that should be restricted to regression problems. In this work, we converted regression problems into classification problems by splitting the y, which infinitely spans from $[0.0, 1.0]$, into 10 regimes equally.

To train our model, we need to provide features (x), targets (y) at training time, and additional *task id* (t) while splitting the data. We achieve this by providing *task id* (t) while loading the dataset and treating each dataset with a different *task id* as a separate dataset. This separation of the dataset is required to split the data correctly based on its performance shifts after upgrading software.

3.3 Continual Learning Strategies

In this work, we adopt six different methodologies that are popular in continual learning as benchmarks. These approaches include:

1. **Elastic Weight Consolidation (EWC)** [17], which is a regularization-based approach that measures the importance of the parameters for the current task and penalizes future updates. The regularization term of EWC consists of a quadratic penalty term for each previously learned task, whereby each task's term penalizes the parameters for how different they are compared to their value directly after finishing the training on that task. The strength of each parameter's penalty depends for every task on how important that parameter was estimated to be for that task, with higher penalties for more important parameters.

2. **Synaptic Intelligence (SI)** [33] is another regularization-based approach that consists of only one quadratic term that penalizes changes to important parameters which are identified by tracking each synapse's credit assignment during the task. The importance parameter is measured by computing the per parameter contribution to the change of loss for the current task and thus strongly contributing parameters are heavily penalized in subsequent tasks.

3. **Learning without Forgetting (LwF)** [19] is a distillation-based approach towards continual learning, wherein previous model outputs are used as soft labels for previous tasks. For this, each input to be replayed is labeled with a "soft target", which is a vector containing a probability for each active class.

4. **Averaged Gradient Episodic Memory (A-GEM)** [9] uses episodic memory as an optimization constraint to avoid catastrophic forgetting. The sample handling of A-GEM avoids solving a quadratic optimization problem for handling samples in the buffer and, instead

uses the mean gradient of such samples from the buffer. The sample is selected, if they point in the same direction in which the current gradient is applied, otherwise an orthogonal projection to the averaged gradient is performed.

5. **Gradient-based Sample Selection (GSS) Greedy** [6] is a sample selection strategy for a setup without task boundaries or at least knowledge about these. Each previously seen sample is regarded as an individual constraint, to which every following sample must be compatible. Sample selection in this context is identical to a constraint reduction problem which is solved by a greedy strategy. This strategy selects n random samples from the buffer and calculates the cosine-similarity between the gradient of the current sample and the gradients of the selected samples Samples of the buffer are only replaced if the similarity falls below a defined threshold, that is, the sample with maximal cosine-similarity is replaced.

6. **Greedy Sampler and Dumb learner (GDumb)** [30] greedily stores samples balanced over the observed classes and at test time learns a new model from scratch with the rehearsal memory. This approach consists of two components, namely the gradient balancing sampler and learner. The sampler greedily creates a new bucket for that class and starts removing samples from the old ones, in particular, from the one with a maximum number of samples.

For the baseline, a simple fine-tuning method without any optimization to prevent catastrophic forgetting is used. We refer to this method as **Baseline** in this work. In **Baseline**, a model is just incrementally fine-tuned depending on the new (drifted) data. **Baseline** is provided natively inside Avalanche, with the name **Naive**, to give the lowest baseline of a model that suffers from catastrophic forgetting and mimics typical (stationary) supervised learning.

Chapter 4

Experimental Setup

The following section defines the experiment setup and establishes the improvement induced by the use of performance modeling as proposed. All experiments were carried out inside Chameleon [1] *bare metal* machines with a single Intel (R) Xeon (R) CPU @ 2.00GHZ CPUs consisting of 8 cores with hyper-threading enabled, 12 GiB of memory, and 200 GB of disk space. We also use RTX-6000 GPUs with 24 GB of memory for the deep learning.

4.1 Metrics

To evaluate the performance of the models, we use the metrics defined as follows:

1. **Average Accuracy** [25] is used to determine the accuracy of the trained model to predict current and prior data after training. The formula is shown as follows:

$$\text{Average Accuracy}(A_i) = \frac{1}{i}\sum_{j=1}^{i} a_{i,j} \tag{4.1}$$

where i is current task and $a_{i,j}$ represents the accuracy assessed on the test set reserved for task j after training the network on tasks from 1 to i.

2. **Remembering** [10] is used to measure how well model remembers the knowledge of previous tasks. The formula is shown as follows:

$$\text{Remembering} = 1 - min\left(0, \left|\frac{\sum_{i=2}^{T}\sum_{j=1}^{i-1}(a_{i,j} - a_{j,j})}{\frac{T\times(T-1)}{2}} 1\right|\right) \tag{4.2}$$

where T is the number of tasks and $a_{i,j}$ is as described (4.1).

4.2 Models Implementation

Table 4.1: **Hyperparameter configurations.** *The configurations used in our experiments.*

	Parameters	Value
General	Epochs	60
	Training Batch Size	4
	Test Batch Size	4
	Learning Rate	0.001
	Optimizer	Adam
	Hidden Layers	3
	Hidden Size	400
Synaptic Intelligence (SI)	Lambda	1.0
	Eps	1×10^{-7}
EWC	Mode	Separate
	Lambda	0.5
LwF	Alpha	1.0
	Temperature	2.0
AGEM	Patterns Per Exp	2
GSS Greedy	Mem Size	200
GDumb	Mem Size	200

We use the API provided by Avalanche [22], a continual learning framework based on PyTorch [28] which provides us with essential continual learning methodologies that are used in this work.

We used the class-incremental setting [26], as specified in Section 3.2, which provides *task id* at the training time but omits the *task id* while performing inferences. Furthermore, Table 4.1 shows the hyperparameters of neural network models, training and evaluation configurations, and continual learning approaches parameter that are used through the experiments in this work. For memory-based approaches such as GSS Greedy [6] and GDumb [30], we use the same memory size of 200 samples to ensure a fair comparison and avoid overfitting to the memory buffer.

Chapter 5

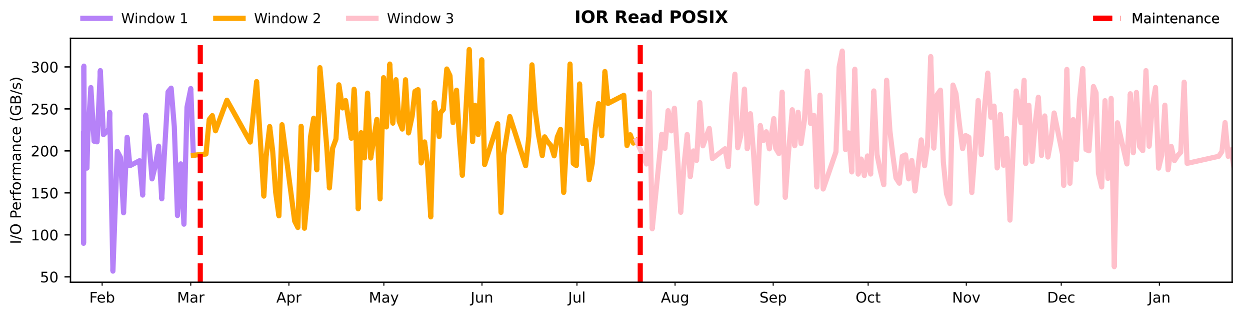
Results and Discussion

This section is dedicated to showcasing the outcomes of our experimental research on the I/O performance modeling problem in Cori Cluster as specified in the Chapter 2. These results are aimed at answering the research questions as stated below:

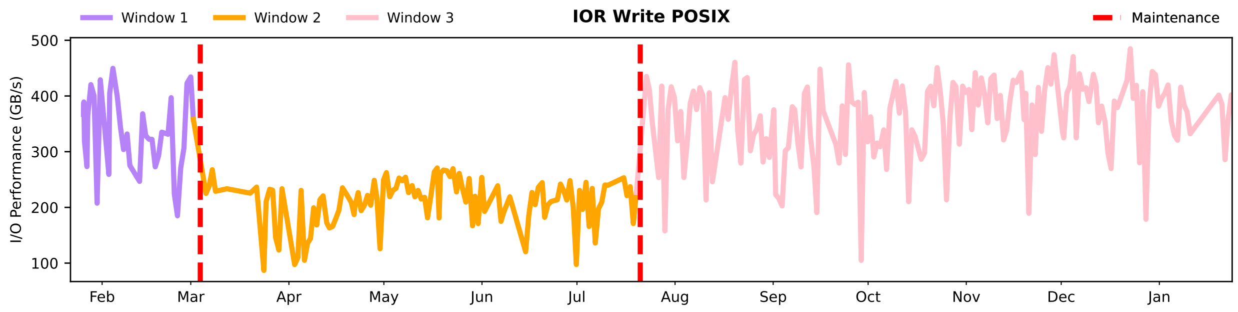
- **RQ1:** How does the performance of the CL methodologies compare to the baseline model?
- **RQ2:** How do the different CL methodologies perform on our I/O profiling data?

5.1 IOR Benchmark

5.1.1 POSIX API



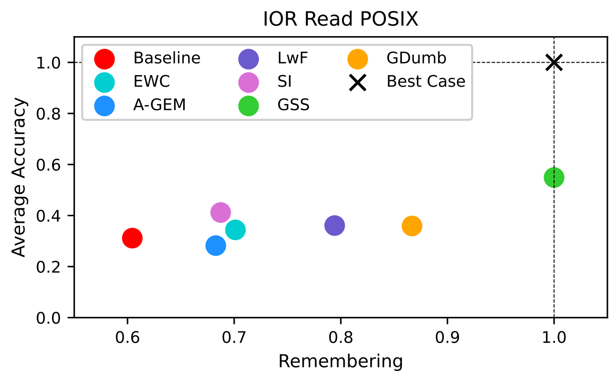
(a) IOR Read Posix Data



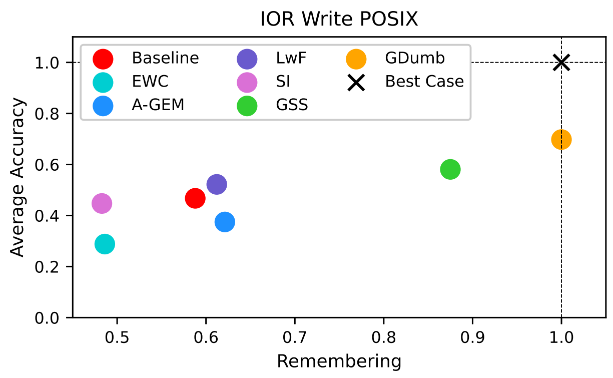
(b) IOR Write Posix Data

Figure 5.1: **IOR Posix Data.** *IOR Read Posix (a) and IOR Write Posix (b) data. Red line shows the temporal location of the software upgrade.*

In this section, the outcomes of the IOR benchmark utilizing the POSIX API are detailed. Referencing Fig. 5.1, it is observed that the IOR Read Posix (5.1a) lacks a distinct trend in performance decline. Conversely, the IOR Write Posix (5.1b) exhibits a marked performance deterioration subsequent to the software update. As indicated in Figure 5.2, the GSS Greedy method demonstrates superior performance in IOR Read Posix, achieving an average accuracy of 0.54, surpassing the baseline model's lower average accuracy of approximately 0.31. In the context of IOR Write Posix, GDumb exceeds the baseline model and other CL strategies, achieving an average accuracy of 0.69, in contrast to the baseline model's average accuracy of around 0.46. These findings suggest that the CL methodologies effectively detect performance degradation following the software upgrade in scenarios where IOR Read Posix exhibits no obvious performance decline and where there is a noticeable performance decline in IOR Write Posix.



(a) IOR Read Posix Result

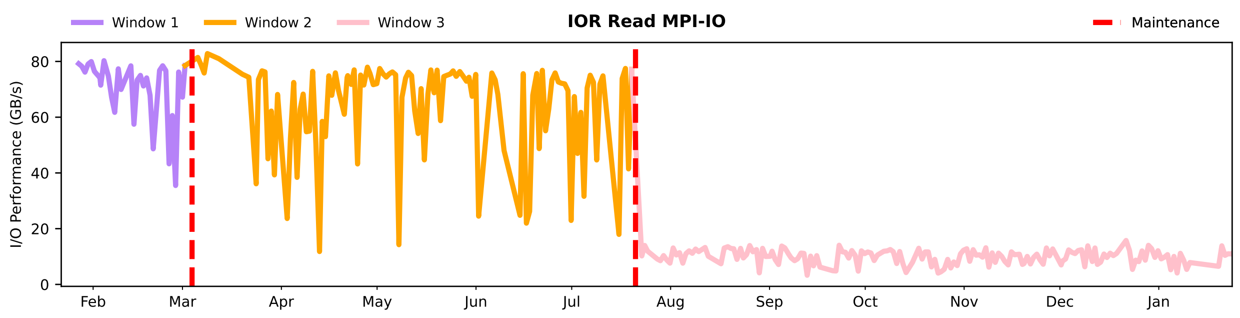


(b) IOR Write Posix Result

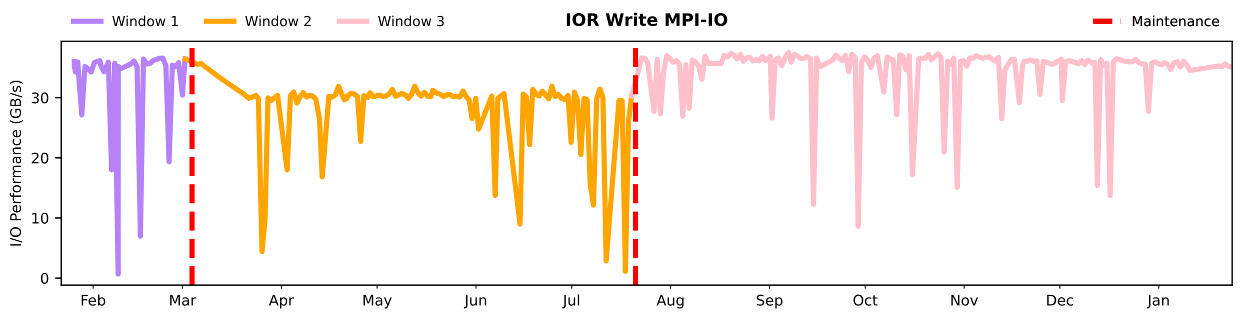
Figure 5.2: **IOR Posix Result.** *Average Accuracy vs Remembering of CL methodologies on IOR Read Posix (a) and IOR Write Posix (b). Best case is at the top right corner.*

5.1.2 MPI-IO API

Within this section, the outcomes of the IOR benchmark employing the MPI-IO API are elaborated. The IOR Read benchmark (5.3a) demonstrates a pronounced pattern of performance degradation following the second software update. Conversely, the IOR Write benchmark (5.3b) exhibits slight performance decline after the initial software update but reverts to normal performance post the second upgrade. This phenomenon is widely known as reoccuring concepts [23]. Once more, GSS Greedy emerges as the most effective methodology for IOR Read MPI-IO, delivering an average accuracy of 0.82, markedly surpassing the baseline model's average accuracy of 0.31. In the case of IOR Write MPI-IO, GDumb proves to be the superior methodology, achieving an average accuracy of 0.75 and remembering rate of 0.84, albeit not outperforming GSS Greedy, which has an average accuracy of 0.67 and remembering rate of 0.91. Notably, A-GEM seems to excel above all other CL methodologies in retaining prior knowledge in IOR Read MPI-IO. However, this is attributed to the observation that A-GEM's accuracy remains constant at 0.3 throughout the experiment, not



(a) IOR Read MPI-IO Data



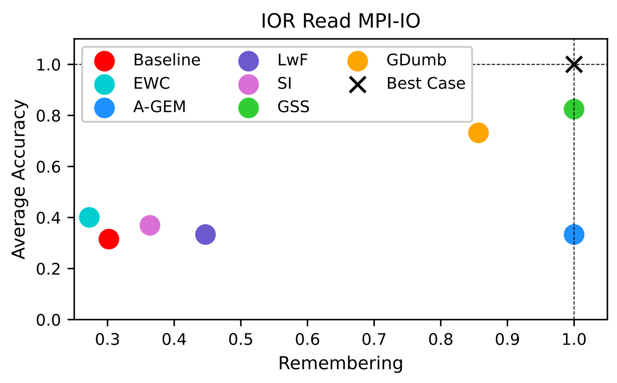
(b) IOR Write MPI-IO Data

Figure 5.3: **IOR MPI-IO Data.** *IOR Read MPI-IO (a) and IOR Write MPI-IO (b) data. Red line shows the temporal location of the software upgrade.*

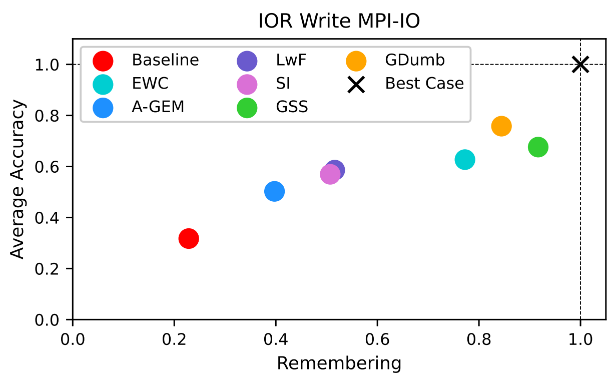
showing improvement. The reasons behind this phenomenon warrant further investigation and are designated as a subject for future research. Nevertheless, the results from other CL methodologies are not as promising as those from GSS Greedy and GDumb.

5.2 HACC Posix Benchmark

This section focuses on the outcomes related to HACC Posix. Specifically, the HACC Write Posix (5.5) demonstrates a distinct performance drift subsequent to the second software update, while the HACC Read Posix (5.5b) does not exhibit any notable performance drift. As depicted in Figure 5.6, both GSS Greedy and GDumb maintain their performance within the 0.5-0.6 range for both HACC Read Posix and HACC Write Posix, outperforming the baseline model, which is susceptible to drift in both instances. The baseline model exhibits an average accuracy of approximately 0.5 in

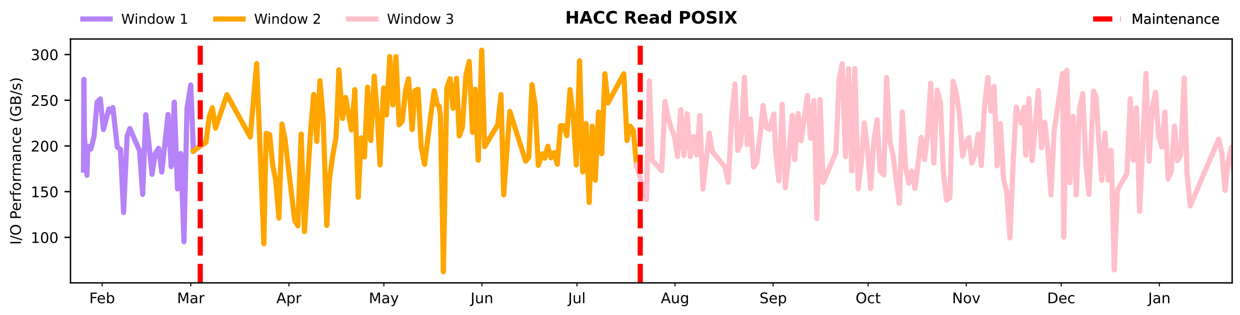


(a) IOR Read MPI-IO Result

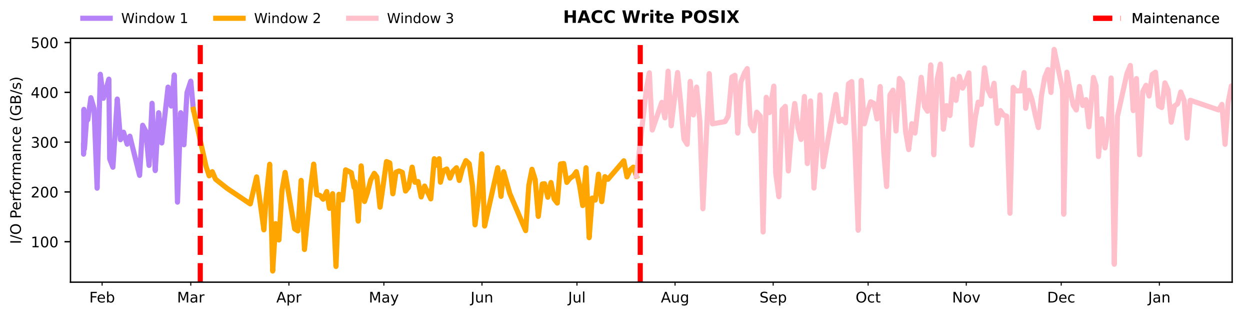


(b) IOR Write MPI-IO Result

Figure 5.4: **IOR MPI-IO Result.** *Average Accuracy vs Remembering of CL methodologies on IOR Read MPI-IO (a) and IOR Write MPI-IO (b). Best case is at the top right corner.*



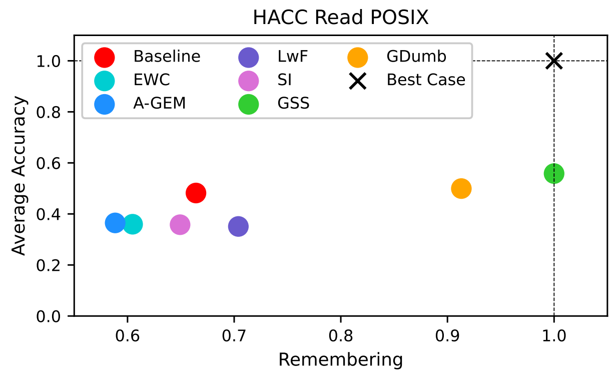
(a) HACC Read Posix Data



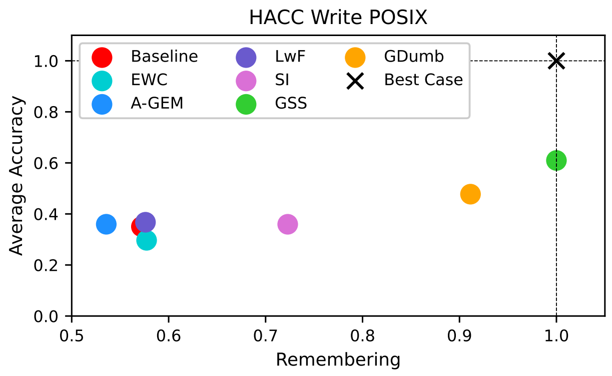
(b) HACC Write Posix Data

Figure 5.5: **HACC Posix Data.** *HACC Read Posix (a) and HACC Write Posix (b) data. Red line shows the temporal location of the software upgrade.*

both cases, with a retention rate of 0.65 for HACC Read Posix and approximately 0.57 for HACC Write Posix. Nevertheless, it's observed that other CL methodologies also do not exhibit learning in these drift scenarios.



(a) HACC Read Posix Result



(b) HACC Write Posix Result

Figure 5.6: **HACC Posix Result.** *Average Accuracy vs Remembering of CL methodologies on HACC Read Posix (a) and HACC Write Posix (b). Best case is at the top right corner.*

5.3 DBSCAN MPI-IO Benchmark

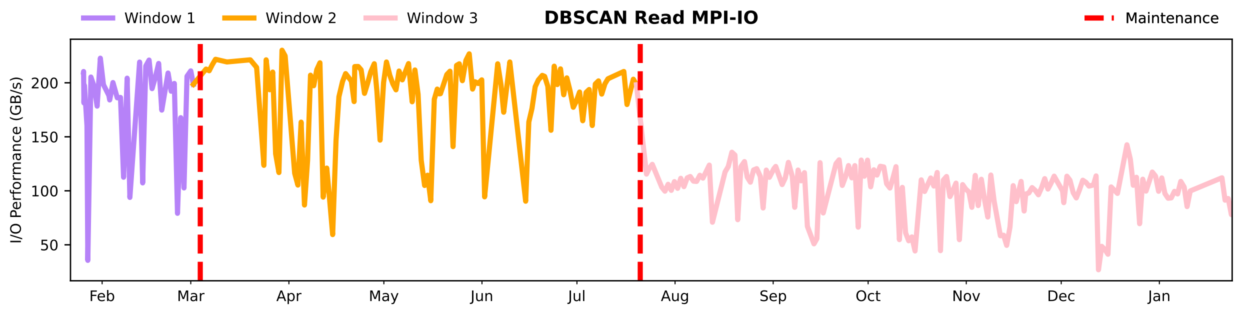


Figure 5.7: **DBSCAN MPI-IO Data.** *Red line shows the temporal location of the software upgrade.*

In this benchmark, a significant I/O performance dropped was noted after the second software upgrade, as illustrated in Figure 5.7. Figure 5.8 reveals that GSS Greedy and GDumb are closely matched in terms of average accuracy and retention rate, with both methodologies demonstrating equivalent memory of past information. Nevertheless, GSS Greedy exhibits higher average accuracy than GDumb, with values of 0.64 and 0.60, respectively. It is also noted that the baseline model, along with other CL methodologies, struggle to adapt in this scenario of drift, achieving only about 0.3 in average accuracy and a remembering rate ranging between 0.4 and 0.5.

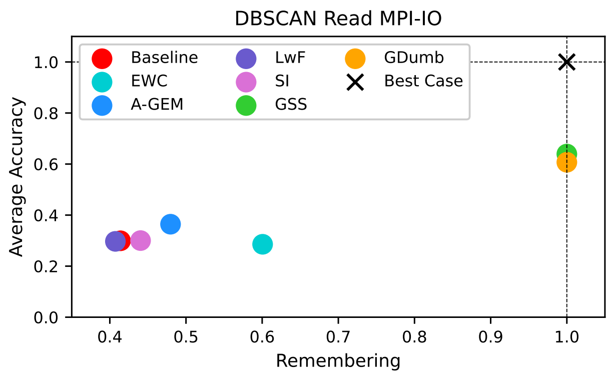


Figure 5.8: **DBSCAN MPI-IO Result.** *Average Accuracy vs Remembering of CL methodologies on DBSCAN Read MPI-IO. Best case is at the top right corner.*

5.4 VPIC MPI-IO Benchmark

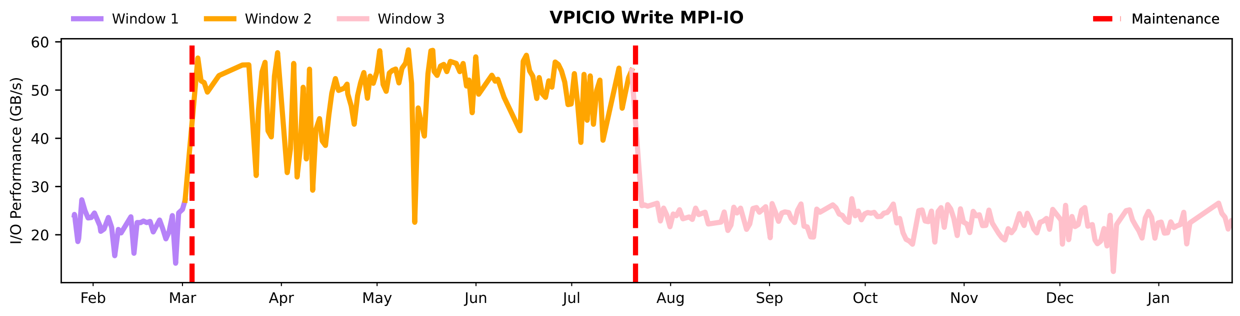


Figure 5.9: **VPIC MPI-IO Data.** *Red line shows the temporal location of the software upgrade.*

In the VPIC MPI-IO benchmark, a significant improvement in performance was observed following the first software upgrade, as indicated in Figure 5.9. However, after the second software upgrade, the performance reverted to its original levels, exemplifying a case of recurring concept drift [23]. According to Figure 5.10, GSS Greedy and GDumb emerge as the leading methodologies for this benchmark, achieving average accuracies of 0.69 and 0.71 respectively, with both maintaining a remembering rate of around 0.9. In contrast, the baseline model and other CL methodologies again fail to adapt effectively, achieving only between 0.2 to 0.5 in average accuracy and a retention rate in the range of 0.3 to 0.6.

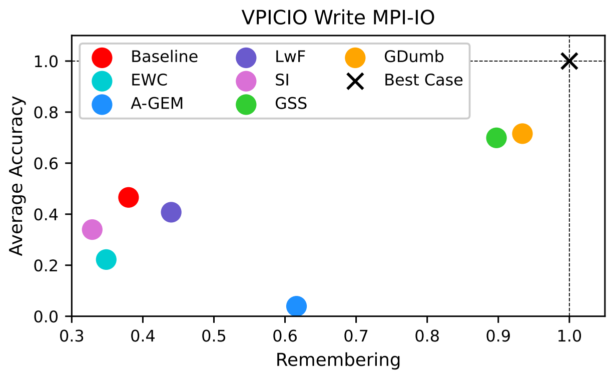


Figure 5.10: **VPIC MPI-IO Result.** *Average Accuracy vs Remembering of CL methodologies on VPIC Write MPI-IO. Best case is at the top right corner.*

5.5 Discussion

Having gathered and analyzed all the results, we are now equipped to respond to the questions presented in 5. This step is essential for fulfilling the objectives of our study and provides clarity on the research inquiries initially proposed.

5.5.1 RQ1: How does the performance of the CL methodologies compare to the baseline model?

It has been observed that two out of three memory-based Continual Learning (CL) algorithms, notably GSS Greedy and GDumb, exhibit superior performance across all benchmark applications used in this study. A-GEM, on the other hand, performed poorly on every benchmark results. GSS and GDumb surpass the baseline model and other CL methodologies significantly, demonstrating at least a two-fold improvement in remembering rate and an average accuracy that is about 1.5 times better. The success of memory-based CL algorithms can be attributed to their ability to recall past knowledge through techniques involving sampling and data storage in memory. These algorithms are better in adapting to recurring drift also due to its ability to retain the knowledge. In contrast, methodologies like Elastic Weight Consolidation (EWC), Learning without Forgetting (LwF) and Synaptic Intelligence (SI), which primarily rely on regularization to mitigate catastrophic forgetting, do not perform as well. These methods continue to struggle with catastrophic forgetting, leading to poorer predictions despite strategies aimed at updating the model to prevent memory loss. Further investigation is required to ascertain the reasons behind the inadequate performance of these methodologies in our experiments. A potential factor could be the lack for proper tuning of their hyperparameters. Addressing these issues and optimizing the performance of these methodologies remain areas for future research.

5.5.2 RQ2: How do the different CL methodologies perform on our I/O profiling data?

In the context of retaining past knowledge, GSS Greedy and GDumb have proven to be the most effective, particularly in situations where data drift recurs. Such scenarios are common in computer systems following multiple maintenance activities, which may involve hardware replacement or software upgrades. The advantage of these Continual Learning (CL) methodologies lies in their ability to capitalize on recurring patterns, as they are adept at remembering past knowledge and updating the model in response to these changes. However, their performance is not as robust when faced with non-recurring data drifts or data that contains significant noise. In these instances, CL methodologies still face challenges due to the increased variability in the data. To improve predictions in such scenarios, further exploration and processing of the data are essential. This approach would enable a more accurate adaptation to the variations in the data, enhancing the overall efficacy of the CL models.

Chapter 6

Future Work

In this work, we focus on modeling applications performance inside HPC production systems which usually suffer from catastrophic forgetting due to data distribution drifts as a result of system changes over time. We highlight that the distribution drift for performance modeling follows *real concept drift* and present a continual learning-based approach to performance modeling that is able to retain prior knowledge while maintaining the ability to learn new data. We incorporate real concept drift within the Avalanche framework to evaluate on various continual learning benchmarks. We propose a novel approach towards modeling continuous concept drift as a class incremental learning problem, and observe that the best performing GDumb model provides a 2$\times$ improvement in accuracy over the naive fine-tuning approach.

In this study, we have not yet undertaken an extensive optimization of hyperparameters for both the baseline model and the various continual learning approaches employed. It is our hypothesis that the model's performance could be significantly enhanced through meticulous hyperparameter tuning. This process involves adjusting parameters such as learning rate, batch size, network architecture, and continual learning algorithms specific parameter, such as memory size, which are crucial for optimizing the model's efficiency and accuracy. Our future work will focus on this aspect to fully realize the potential of our models in continual learning scenarios.

As of now, the task identification process, as outlined in Section 3.2, relies on manual collection by server operators. This manual approach, while effective, introduces potential delays and inaccuracies in data acquisition. To enhance efficiency and accuracy, the integration of an automated system based on the concept drift-aware modeling technique [24] is needed. This technique is capable of autonomously detecting instances of concept drift in near real-time. The integration of this automated system with our existing predictive models offers significant promise. It will enable our models to continuously and adaptively learn from data within production environments, thereby optimizing performance and reducing the reliance on manual processes. This integration is anticipated to substantially improve the responsiveness and precision of our system in adapting to changing data patterns in real-world scenarios.

In the upcoming phase, our goal is to implement the enhanced model within operational systems. For the model to accurately capture and adapt to the evolving behaviors of applications, it is essential to gather a more extensive set of data, particularly data that exhibits varying concept drifts. Additionally, the insights derived from our model have the potential to significantly assist related software tools, such as job schedulers and monitoring utilities. By supplying these tools with comprehensive information, our model aims to support and refine their decision-making processes. In essence, by embedding our model into the operational infrastructure and its associated software components, we strive to elevate the efficiency of managing and optimizing tasks across these systems.

Chapter 7

Conclusion

In this research, we have pioneered the application of continual learning to real-world system I/O performance data. Our study utilizes I/O performance data sourced from Cori, a production supercomputer located at the National Energy Research Scientific Computing Center (NERSC). This data serves as a basis for demonstrating the efficacy of continual learning in forecasting I/O performance. The findings reveal that, in the context of system changes, continual learning surpasses traditional fine-tuning methods by approximately 1.5 to 2 times in terms of average accuracy. Moreover, it demonstrates a two to threefold improvement in retaining previous knowledge. This highlights the significant potential of continual learning methodologies in adapting to and accurately predicting performance in dynamic, real-world computing environments.

References

[1] Chameleon. https://www.chameleoncloud.org.

[2] IOR Benchmark. https://github.com/hpc/ior.

[3] Lustre File System. http://lustre.org/.

[4] PIOK: Parallel I/O Kernels. https://github.com/hpc-io/PIOK.

[5] Mohamed Abdelsalam, Mojtaba Faramarzi, Shagun Sodhani, and Sarath Chandar. IIRC: Incremental Implicitly-Refined classification. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11033–11042, June 2021.

[6] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019.

[7] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Trans. Storage*, 7(3):1–26, October 2011.

[8] Fabio Cermelli, Massimiliano Mancini, Samuel Rota Bulò, Elisa Ricci, and Barbara Caputo. Modeling the background for incremental learning in semantic segmentation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9230–9239, June 2020.

[9] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[10] Natalia Díaz-Rodríguez, Vincenzo Lomonaco, David Filliat, and Davide Maltoni. Don't forget, there is more than forgetting: new metrics for Continual Learning. In *Continual Learning Workshop at NeurIPS 2018*, 2018.

[11] R M French. Catastrophic forgetting in connectionist networks. *Trends Cogn. Sci.*, 3(4): 128–135, April 1999.

[12] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):1–37, March 2014.

[13] Garlick and Morrone. Lustre monitoring tools. https://github.com/LLNL/lmt.

[14] Haryadi S Gunawi, Riza O Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Deepthi Srinivasan, Biswaranjan Panda, Andrew Baptist, Gary Grider, Parks M Fields, Kevin Harms, Robert B Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H Birali Runesha, Mingzhe Hao, and Huaicheng Li. Fail-Slow at scale: Evidence of hardware performance faults in large production systems. *ACM Trans. Storage*, 14(3):1–26, October 2018.

[15] Salman Habib, Vitali Morozov, Hal Finkel, Adrian Pope, Katrin Heitmann, Kalyan Kumaran, Tom Peterka, Joe Insley, David Daniel, Patricia Fasel, Nicholas Frontiere, and Zarija Lukic. The Universe at extreme scale: Multi-petaflop sky simulation on the BG/Q. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.

[16] Michael P Kasick, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Black-Box problem diagnosis in parallel file systems. In *FAST*, pages 43–56, 2010.

[17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114(13):3521–3526, 2017. ISSN 0027-8424. doi: 10.1073/pnas.1611835114.

[18] Timothée Lesort, Massimo Caccia, and Irina Rish. Understanding continual learning settings with data distribution drift analysis. April 2021.

[19] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[20] Glenn K Lockwood, Shane Snyder, Teng Wang, Suren Byna, Philip Carns, and Nicholas J Wright. A year in the life of a parallel file system. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 931–943, November 2018.

[21] Glenn K Lockwood, Nicholas J Wright, Shane Snyder, Philip Carns, George Brown, and Kevin Harms. TOKIO on ClusterStor: Connecting standard tools to enable holistic I/O performance analysis. Technical report, Lawrence Berkeley National Lab. (LBNL), Berkeley, CA (United States), January 2018.

[22] Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L. Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido van de Ven, Martin Mundt, Qi She, Keiland Cooper, Jeremy Forest, Eden Belouadah, Simone Calderara, German I. Parisi, Fabio Cuzzolin, Andreas Tolias, Simone Scardapane, Luca Antiga, Subutai Amhad, Adrian Popescu, Christopher Kanan, Joost van de Weijer, Tinne Tuytelaars, Davide

Bacciu, and Davide Maltoni. Avalanche: an end-to-end library for continual learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2nd Continual Learning in Computer Vision Workshop, 2021.

[23] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, Guangquan Zhang of reoccuring concepts, and etc. Learning under Concept Drift: A Review. In *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[24] Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert Latham, Glenn K Lockwood, Robert Ross, Shane Snyder, and Stefan M Wild. Adaptive learning for concept drift in application performance modeling. In *Proceedings of the 48th International Conference on Parallel Processing*, New York, NY, USA, August 2019. ACM.

[25] Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. Online continual learning in image classification: An empirical survey. *Neurocomputing*, 469: 28–51, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2021.10.021. URL https://www.sciencedirect.com/science/article/pii/S0925231221014995.

[26] M. Masana, X. Liu, B. Twardowski, M. Menta, A. D. Bagdanov, and J. van de Weijer. Class-incremental learning: Survey and performance evaluation on image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(05):5513–5533, may 2023. ISSN 1939-3539. doi: 10.1109/TPAMI.2022.3213473.

[27] Martin Mundt, Yong Won Hong, Iuliia Pliushch, and Visvanathan Ramesh. A wholistic view of continual learning with deep neural networks: Forgotten lessons and the bridge to active and open world learning. *arXiv preprint arXiv:2009.01797*, 2020.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, and Others. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.*, 32, 2019.

[29] Md. Mostofa Ali Patwary, Suren Byna, Nadathur Rajagopalan Satish, Narayanan Sundaram, Zarija Lukic, Vadim Roytershteyn, Michael J. Anderson, Yushu Yao, Prabhat, Pradeep Dubey BD-CATS: big data clustering at trillion particle scale. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2015.

[30] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *European Conference on Computer Vision*, pages 524–540. Springer, 2020.

[31] Sebastian Thrun and Lorien Pratt. *Learning to Learn*. Springer Science & Business Media, December 2012.

[32] Andy B Yoo, Morris A Jette, and Mark Grondona. SLURM: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer Berlin Heidelberg, 2003.

[33] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3987–3995. JMLR. org, 2017.