Pattern Matching to improve Accuracy and Efficiency of File Lifecycles Forecasting

Adrian KHELILI
Eviden Atos R&D Data Management
Li-PaRAD, UPSaclay-UVSQ, France
adrian.khelili@eviden.com

Sophie ROBERT HAYEK
Eviden Atos R&D Data Management
Echirolles, France
sophie.robert@eviden.com

Soraya ZERTAL
Li-PaRAD, UPSaclay-UVSQ
Guyancourt, France
soraya.zertal@uvsq.fr

***Abstract*—Time series prediction is a crucial task with applications in various domains. In this research, we introduce a transfer learning approach that goes beyond the historical time series of individual instances to enhance prediction performance. Traditional time series prediction methods often focus solely on historical data from a single time series, limiting their ability to capture already known patterns and behaviors. Our proposed method addresses this limitation by constructing a curated database of time series representative of the behavior to predict. We apply our suggested method to the prediction of the I/O (Input/Output) behavior of scientific applications such as NEMO, NAMD, and LQCD. Our non-redundant database contains diverse time series, allowing us to extract valuable insights and patterns that might not be discernible from individual series. By leveraging similarities and patterns across those time series, our approach improves prediction accuracy and robustness. The results we obtained are highly promising, demonstrating that in term of time serie ranking prediction accuracy our proposed inter-timeseries method achieves from 72.9% to 83.8% accuracy in most difficult scenarios to 97% for the others. To compare our solution to existing methods, we show an improvement of 10.1% accuracy compared to ARIMA.**

***Index Terms*—DataBase, File ranking, Time Series, Neirest Neighbor, Pattern matching.**

I. Introduction

In the last fifty years, time series analysis has gained increasing importance across various domains, ranging from the analysis of financial temporal data [23] to the healthcare domain [15]. A variety of methods for time series analysis have been developed over the years, including both parametric and non-parametric approaches, each with its own strengths and limitations. A possible approach is leveraging known time series behaviors to infer the behavior of similar time series. The insight that resemblance between time series provides valuable information has broader implications across different domains of application and is essential for uncovering hidden patterns and trends in time series behavior data.

A particular application of time series analysis using time series matching is the improvement of the understanding of the behavior of High Performance Computing (HPC) systems [21], [24]: by predicting application's future behavior, systems can become more adaptable and thus more efficient. As distinct types of behavioral I/O patterns can be observed by studying the File Lifecycles (FLC) depending on the file type [12], exploiting the resemblance between time series can provide a wealth of information and predict application's future behavior: for example, input files show high read activity at the beginning, providing initial data for computation, checkpointing files exhibit phased activity for fault tolerance, while output files display high activity towards the end for writing computation results.

In this paper, we suggest to use time series forecasting using pattern matching, by comparing the currently observed time series to a previously collected database of observed patterns. The database is carefully selected to avoid any redundancy and minimize overhead, so that the forecasting time remains acceptable for systems running in production such as HPC systems.

The main contributions of our paper are:

- The introduction of a new custom distance metric for file-based time series forecasting.
- The introduction of a new method for time series forecasting using a non redundant database.
- The validation of the proposed forecasting algorithm in the context of HPC file lifecycles prediction on a set of 3 real applications.

This paper is structured as follows. Section II introduces works that are similar to ours and highlights the novelty of our suggestion while section III motivates our work. Section IV presents the proposed methods for time series forecasting using non redundant databases construction and the forecast algorithm. Section V describes the experiments conducted to validate our approach on file-based time series. Section VI exposes and discusses the obtained results and section VII concludes the paper giving some insight into future works.

II. Related works

In the field of applied time series analysis, forecasting future events such as file behavior is a challenging task that requires a balance between high accuracy and time efficiency. Parametric methods as ARIMA [22] and SARIMA [7] assume linearity and stationarity of data, which restricts their effectiveness in handling cases involving non-linearity or non-stationarity. Exponential smoothing models [9] assume that the future values of the time series depend on past observations with exponentially decreasing weights. While these models work well for time series with stable trends and seasonality, they may not

979-8-3503-0821-1/23/$31.00 ©2023 IEEE

be appropriate for complex and irregular data patterns. Non-parametric methods, such as kernel density [10] estimation, Fourier analysis, and wavelet analysis, offer valuable alternatives in time series analysis by avoiding strong assumptions about the data distribution. Kernel density estimation gives the probability density function of the data, providing insights into the underlying data distribution. However, it can be computationally intensive and requires careful bandwidth selection, which can impact its efficiency and accuracy, especially for large datasets. Fourier analysis [3] decomposes the time series into a sum of sinusoidal components, revealing underlying periodicities, but its limitation lies in the lack of precise time localization, hindering its ability to capture rapidly changing or localized patterns. Wavelet analysis [16] decomposes the time series into different frequency components, enabling analysis of localized patterns at various scales. Nevertheless, the choice of an appropriate wavelet can be challenging and may impact the effectiveness of the analysis. Despite these limitations, non-parametric methods remain valuable tools for time series analysis, particularly when the data distribution and characteristics are complex and cannot be easily captured by parametric approaches. Among non-parametric methods, k-nearest neighbor regression [20] is another powerful approach that captures patterns by comparing subsequences with their nearest neighbors. This is known as pattern recognition and involves several algorithms. Techniques like SAX (Symbolic Aggregate approXimation) [14] and PAA (Piecewise Aggregate Approximation) are popular for compressing time series data and reducing dimensionality [19] and Dynamic Time Warping (DTW) is a crucial algorithm for calculating the distance between time series, taking temporal shifts into account [11]. Additionally, to the best of our knowledge, the existing literature provides clustering methods for time series databases [13], while other approaches involve forecasting using LSTM networks [2]. However, these LSTM-based methods often suffer from significant time complexity. The database construction process is still poorly, if at all, described in the literature. When it comes to the prediction of I/O behavior using time series methods, Dong et al. use a time series model to estimate file system server load in [8] while Boito et al. propose a pattern matching approach for server-side access pattern detection for the HPC I/O stack in [4]. Both Lalitha et al. and Tran et al. use ARIMA models to capture both temporal and seasonal trends in time series data, making it a popular choice for predicting I/O performance in various settings in [22] and [6] respectively. Indeed, to our knowledge, there is no existing method in the literature that explicitly leverages file similarity for the purpose of file behavior prediction. By the present work, we aim to fill this gap by exploring the potential benefits of incorporating file-based time series similarity into the predictive modeling process of I/O behavior.

III. Motivation

In this research work, our motivation stems from the need to tackle the challenging task of predicting time series behavior by using knowledge of previous known time series that relies on FLC's.

Both ARIMA and KNN have their advantages and drawbacks as studied by Smith et al. [1] and cannot be used straight forward. We recognize that ARIMA, as a parametric model, is well-suited to predict time series representing intra-file variations. This can be used to analyze the temporal patterns within individual files. However, our target inter-file predictions and aim to leverage the collective knowledge gained from a diverse database of time series representing various file behaviors. By adopting non-parametric approaches like K-Nearest Neighbors (KNN), we can effectively compare new files with existing profiles in the database, identifying similarities with previously encountered profiles. Additionally, K-NN has the distinctive advantage of allowing a free choice of a distance metric. This distance determines how much two time series are similar or close to each others. Depending on the specific characteristics of the time serie and the objective of the analysis, different distance measures can be employed such as euclidean distance, Dynamic Time Warping distance (DTW) or custom distance offering the flexibility for user to define a distance metrics that align with the specific requirements of their context. Through this inter-file time series prediction, we can gain valuable insights into the likely behavior of new files based on their similarity to known patterns.

This allows us to leverage knowledge transfer between files, contributing to a more comprehensive understanding of file behavior.

We use both inter-timeseries predictions with K-Nearest Neighbors (KNN) and intra-timeseries predictions with ARIMA. By comparing the results obtained from these two separate methodologies, we aim to determine which approach is better suited for predicting file behavior in the online context and facilitates effective knowledge transfer between files. This analysis will provide valuable insights for improved decision-making and optimization of online systems.

IV. Methods

In this section, we present our proposed time series modeling methodology and describe our forecasting algorithm in two distinct phases. The first step consists in the creation of two databases: one for READ operations and another for WRITE operations, with the least possible redundancy of FLC profiles. A profile is a unique record of POSIX access operations (such as open, read, write, close) performed on an individual file over time. It captures the file's distinct behavioral pattern, which can vary depending on the file type (e.g., input, checkpoint, output). By analyzing these patterns, we can forecast how each file will be accessed and utilized in the future, taking into account their specific characteristics and usage scenarios.

A. Definitions and notations

In the upcoming sections, we will use the following terms and notations:

- **Time Serie:** A time serie $\mathbf{T} = (\mathbf{t_1}, \mathbf{t_2}, ..., \mathbf{t_n})$ of length n is a sequence of n real-valued observations, where each t_i represents a value observed at a specific time point.
- **Subsequence:** Let T be a time serie of length $\mathbf{n}$. A subsequence $\mathbf{S_{i,l}}$ is a contigous time serie

of length $\mathbf{l}$ with $\mathbf{1 \leq i \leq n}$ and $\mathbf{1 \leq i + l \leq n}$. $\mathbf{i}$: $\mathbf{S_{i,l}} = (\mathbf{t_i}, \mathbf{t_{i+1}}, ..., \mathbf{t_{i+l-1}})$.

- **FLC**: a temporal series of FLCevents $(o_t)_{1 \leq t \leq T}$ performed on a given file, with o_t an operation in the POSIX set {READ, WRITE, OPEN, CLOSE}.
- **FLC_Events**: FLC_Events are all POSIX operations such as OPEN, READ, WRITE, and CLOSE made on the same file.
- **NN**: Neirest Neighbor.

B. Time series pre-processing

We model files as time series, representing the FLC (File Lifecycles) where each observation in the FLC represents an FLC_Events.

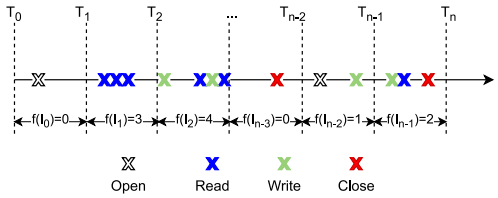

Fig. 1: FLC Time serie representation

As a pre-precessing phase, we perform a binning on FLC's that involves dividing the time series into intervals $\{T_1, T2, .., T_n\}$. This fixed-size interval constitutes an hyper-parameter denoted as w. This binning divides the time series into a set of bins denoted $I = \{I_1, I_2, ..., I_n\}$, where I_i represents the ith interval of the time serie. The total number of bins, n, is determined by the ratio between the total duration of the application and the bin size. Function f counts the number of requests within interval i and $f(I_i)$ represents the file access frequency at each interval i as represented in figure 1, calculated for READ and/or for WRITE operations.

C. Distance metrics

For nearest neighbor search, a distance metric is required to measure the similarity between data points. This metric plays a crucial role in determining the nearest neighbors for a given query point. However, in our case of working with timeseries with associated metadata, using only a standard distance metric solely based on numerical values would result in the loss of important information related to file names. To address this issue, we propose a similarity metric as a weighted sum of the normalized Euclidean distance and the Levensthein distance. The Euclidean Distance is defined as:

$$\mathrm{ED}_{\mathrm{norm}}(X, Y) = \sqrt{\frac{\sum_{i=1}^{N}(X_i - Y_i)^2}{N}}$$

With X and Y the normalized FLC's being compared and X_i, Y_i the observations at the instant i and N the size of the vectors. The normalized Euclidean distance measures the numerical similarity between FLC's.
The Levenshtein distance [25] captures the dissimilarity between file names by calculating the minimum number of insertions, deletions, and substitutions required to transform one file name into another defined recursively as :

$$\mathrm{lev}_{A,B}(i,j) = \begin{cases} \max(i,j) \text{if } \min(i,j) = 0 \\ \text{else } \min \begin{cases} \mathrm{lev}_{A,B}(i-1,j) + 1 \\ \mathrm{lev}_{A,B}(i,j-1) + 1 \\ \mathrm{lev}_{A,B}(i-1,j-1) + \mathbb{I}(A_i \neq B_j) \end{cases} \end{cases}$$

With A and B the strings to compare and i, j the index in the string A and B respectively. And $I()$ an indicator function that is equal to 1 if $A_i \neq B_j$ and 0 else.
By combining these two distances, we aim to create a trade-off metric that considers both the behavior of the FLC's and the similarity of the file names. The distance between two FLC instances, flc1 and flc2, can be defined as:

$$CombDist(file_1, file_2) = \lambda_1 \cdot ED(flc_1, flc_2) + \lambda_2 \cdot Lev(fnm_1, fnm_2)$$

With λ_1 and λ_2 the weighting factors that determine the relative importance of each distance measure in the overall distance calculation. flc_i refers to the lifecycles of the file i and fnm_i to its filename.

D. Database creation

In order to compare files with existing ones, it is necessary to first establish a non-redundant database of profiles. As previously mentioned, the success of non-parametric regression relies heavily on the quality of the available neighbor database. Ideally, a substantial database encompassing a wide range of scenarios the system may encounter is desirable for accurate modeling. However, while a larger database enhances model accuracy, it also introduces challenges regarding the execution speed of the model. Finding the right trade-off between database size and time execution becomes crucial in achieving effective nonparametric regression results. To achieve this, we take previously extracted profiles from NEMO [18], NAMD [17] and LQCD [5] and add them one by one to the database, considering only the profiles that do not have a distance below a specified threshold to any existing profile in the database. The sizes of these databases were chosen experimentally by setting specific thresholds that would result in databases of approximately desired sizes. These thresholds were determined to ensure that the databases offer distinct scenarios with different sizes, meeting specific criteria for experimentation or analysis So, the number of stored files is reduced from 480 to 15 for both read and write databases. With a smaller dataset, the algorithm can process and analyze the data more efficiently.

Algorithm 1 Databases building

Require: *ExistingProfiles*: FLC's data
Threshold: Threshold of similarity
d: custom distance metric
Ensure: *db*: Non-redundant database.
function DB_BUILD(ExistingProfiles, Threshold, d)
    Initialize an empty database dictionary, $db = \{\}$
    for each $CandidateProf$ in $ExistingProfiles$ **do**
        $minDistance \leftarrow \infty$
        for each $ExistingProfile$ in db **do**
            $dist \leftarrow d(CandidateProf, ExistingProf)$
            if $dist < minDistance$ **then**
                $minDistance \leftarrow dist$
        if $minDistance > Threshold$ **then**
            $db[key] \leftarrow CandidateProf$

We thus ensure that only non redundant profiles are added to the database, avoiding duplicates or profiles that are very similar to existing ones. This helps in maintaining a compact and efficient database while reducing unnecessary comparisons in the second phase of the algorithm that follows.

E. Prediction

Once the non-redundant database is constructed, it serves as a reference for prediction. The prediction process is subdivided into two sub-algorithms: the first one is responsible of finding the most similar profile from the database by performing a nearest neighbor search using the $Comb\_Dist$ as distance metric.

Algorithm 2 Forecasting process

Require: *db* : Database of profiles for NN
ts : FLC's data
cle : Key to compare
Ensure: *best_key* : Key of the most similar profile in the database
function FILE_NN(db, ts, cle, d)
    $best\_key \leftarrow$ first key of *db*
    $min\_dist \leftarrow \infty$
    for each key in keys of *db* **do**
        $dist \leftarrow$ COMB_DIST(key, cle)
        if $dist < min\_dist$ **then**
            $min\_dist \leftarrow dist$
            $best\_key \leftarrow key$
    return $best\_key$

The algorithm then retrieves the forecasted data from the database for the identified nearest file neighbor. This forecast data represents the future values of the time series starting from the current time point (n) and extending up to the specified forecast horizon (h). Then with the forecasted data, the algorithm makes predictions for the behavior of the time series beyond the current time point. Specifically, it predicts the behavior of the time series from the current time point (n) to the next h time points (n + h).

V. Validation

We conduct a set of experimentations to compare the NN algorithm with ARIMA for file order prediction. To do this, we run the prediction on each of the application files and compare the forecast to the ground truth as described in section V-C. In this first part, only the intra-timeseries behavior is used for prediction, which includes the historical time series. In a second step, we consider a prediction that takes into account inter-timeseries relationships as described in section IV. Since we have defined a new distance metric, we compare this new metric, which incorporates both Euclidean distance and string edit distance, against the Euclidean distance alone.

A. Hardware and software

To capture their I/O behavior, each application is run on a single compute node, with 134GiB memory, an AMD EPYC 7H12 processor with 64 cores. The traces that we are working with have been extracted using our FiLiP profiling tool [12]. This tool makes the extraction and profiling time series data from file access patterns. By capturing the temporal dynamics of file access in Json format as described in listing 1.

Listing 1: Example of FileLifecycle

```
"file_1": [
    {
    "timestamp": 4096,
    "type": "WRITE"
    },
    {
    "timestamp": 4098,
    "type": "READ"
    },
]
```

B. Selected applications

Three scientific applications have been selected to evaluate the performance of our algorithm : NEMO [18], NAMD [17] and LQCD [5].

1) NAMD:

NAMD [17] is a parallel molecular dynamics code designed for high-performance simulation of large bio-molecular systems. It has the particularity of being very dependent on the storage hardware, due to its large I/O bursts, and is thus a good use-case.
For our experiment, we use the Satellite Tobacco Mosaic Virus (STMV-28M) configuration. This is a 3x3x3 replication of the original STMV dataset from the official NAMD site, containing roughly 28 million atoms.
NAMD execution goes through 50 steps corresponding to the number of simulation time steps to achieve. Another parameter defines the number of steps after which a checkpoint is performed that is set to 5 to obtain ten checkpoints per run for a significant I/O activity. In this configuration, a total of 2 GB of data is read and 5 GB of data is written, resulting in

a total of 10226 read operations and 1079 write operations. These operations involve a total of 14 files, 10 of which are heavily re-used.

2) NEMO: NEMO [18] (*Nucleus for European Modeling of the Ocean*) is a state-of-the-art modeling framework for research activities in ocean and climate sciences. It is characterized by a significant file re-use, highlighting the importance of a custom file placement policy in the hierarchical storage to keep the most accessed files in the most efficient level. Additionally, we can see that NEMO constantly manipulates a certain number of files, whether in read or write mode, which offers rich patterns for both READ and WRITE operations. It presents a higher activity in reading at the beginning of the application and a higher activity in writing at the end of the application, which is explained by the reading of input files at the beginning and the writing of output files at the end of the application.
For our experiment, we use the GYRE configuration, which simulates the seasonal cycle of a double-gyre box model, and which is often used for I/O benchmarking purpose as it is very simple to increase grid resolution and does not require any input file. In our case, the grid resolution is set to 5 and the number of MPI processes to 32 to increase the I/O activity. In this configuration, a total of 65 GB of data is read and 50 GB of data is written, These operations involve a total of 184 files, among which 21 are heavily reused.

3) LQCD: Lattice QCD is a well-established non-perturbative approach for solving the quantum chromodynamics (QCD) theory of quarks and gluons. It is a lattice gauge theory formulated on a grid or lattice of points in space and time. When the size of the lattice is taken infinitely large and its sites infinitesimally close to each other, the continuum QCD is recovered [5].
In this lattice quantum chromodynamics (LQCD) simulation, the parameters are set as follows: the quark mass (qmass) which represents the mass of the quark is set to 0.02, and the gauge coupling strength (beta) which represents the strength of the interactions between quarks and gluons in the lattice simulation is set to 0.2. The simulation is performed with a spatial extent of 16 lattice units and a temporal extent of 36 lattice units. The simulation is divided into a total of 8 tasks in the temporal direction, 4 tasks in the spatial direction, 4 tasks in the y-direction, and 4 tasks in the x-direction, resulting in a total of 512 tasks.
In this configuration, a total of 27 GB of data is read and 40 GB of data is written, These operations involve a total of 284 files, among which 21 are heavily reused.

C. Evaluation metrics

To evaluate the performance of our prediction algorithms, we use standard multi-class supervised learning metrics: accuracy, precision and MCC for Matthews Correlation Coefficient [4]. The objective is to predict the order of file activity, so we subdivide the order of file prediction into four quartiles, and treat each of these as a specific category. We define for each quartile:

- True Positives (TP): number of correctly predicted instances for a specific quartile.
- True Negatives (TN): number of correctly predicted instances for all other classes excluding the specific quartile.
- False Positives (FP): number of incorrectly predicted instances as the specific class.
- False Negatives (FN): number of instances belonging to the specific class but incorrectly predicted as other classes.

The different metrics are then computed and averaged for quartile:

- Accuracy = $\frac{1}{4} \times \sum_{i=1}^{4} \frac{TP_i+TN_i}{TP_i+FP_i+FN_i+TN_i}$
- Precision = $\frac{1}{4} \times \sum_{i=1}^{4} \frac{TP_i}{TP_i+FP_i}$
- MCC = $\frac{1}{4} \times \sum_{i=1}^{4} \frac{TP_i \cdot TN_i - FP_i \cdot FN_i}{\sqrt{(TP_i+FP_i)(TP_i+FN_i)(TN_i+FP_i)(TN_i+FN_i)}}$

For example, an accuracy of 100% means that every file was predicted in its right quartile. We check the quality of our prediction via these metrics each $w \times 10ns$, then average the results over the whole application execution.

D. Tested hyperparameters

Our prediction also requires several hyperparameters. One of these hyperparameters is the window size (w), which is used to determine the number of IO requests to consider when creating the time serie. We selected those values based on the execution time of the application, to have 1000 data point in total within the time series. The value is then rounded to the nearest power of 10. The second hyperparameter (h) refers to the prediction horizon, which represents the number of observations we want to forecast.

TABLE I: Time serie tested hyperparameters

Application	h	w
LQCD	40	10 000 000
NEMO	40	1 000 000
NAMD	40	1 000 000

The second set of hyperparameters concerns the distance metric introduced in section IV-C, which are λ_1 and λ_2, both set to 0.5 in order to have an equal balance of distance between ed and levenshtein distance.The last hyperparameter that we varied in these experiments is the database size which depends on the threshold, described bellow

TABLE II: DataBase Tested hyperparameters

Threshold	Dbsize
0.1	25
0.35	15
0.5	4

E. Time consumption

We will also perform a validation of the time execution for the forecast algorithm. For this purpose, we will compare the execution of an intra-timeseries approach using the ARIMA algorithm against our inter-timeseries approach, using different

database sizes. Moreover, the comparison will include varying the sizes of the databases used in the inter-timeseries approach. This variation will help us understand how the database size affects the execution time. Larger databases may require more processing time and memory, while smaller databases may yield faster searches but could sacrifice accuracy due to limited data. It should be noted that accuracy increases with the size of the database, making it essential to strike the right trade-off between database size and performance.

VI. Results and Discussion

In this section, we analyze the results of the file ranking, evaluating the prediction quality by comparing the intra-timeseries approach represented by ARIMA with the inter-timeseries approach with different database sizes.

A. NEMO

The results in tables III show NEMO's forecasting results evaluation for inter-timeseries prediction models (NN15) and intra-timeseries ARIMA prediction model. Firstly, the inter-timeseries prediction models exhibit significantly better performance compared to the intra-timeseries ARIMA models. The NN15 model achieves high accuracy, with 92.8% for READ operations and 91.9% for WRITE operations. Additionally, the impact of database size on forecasting performance is clearly evident. When using a small database with just 4 time series (NN4), the precision drops considerably, with values of 43.6% for READ and 75.9% for WRITE. However, as the database size increases to 15 time series (NN15), the precision improves dramatically, reaching the double 85.7% for READ and 83.8% for WRITE. This demonstrates the critical importance of having a diverse and representative dataset for training the forecasting models.

Furthermore, the analysis reveals the importance of finding the right balance between database size and forecasting precision. While increasing the database size up to 15 time series significantly enhances forecasting performance, further expansion to 25 time series (NN25) does not result in notable improvements. In fact, the precision values for NN15 and NN25 are nearly identical, except for the difference in execution time, as depicted in 4. This suggests that beyond a certain point, adding more time series might introduce unnecessary complexity without yielding substantial gains in accuracy.

TABLE III: NEMO forecasting results evaluation

	Metrics	Accuracy	Precision	MCC
READ	NN4	71.8%	43.6 %	24.8 %
	NN15	92.8%	85.7%	80.9%
	NN25	92.8%	85.7%	80.4%
	ARIMA	82.7%	65.4%	53.9%
WRITE	NN4	87.9%	75.9%	67.9%
	NN15	91.9%	83.8%	78.4%
	NN25	92.0%	84.0%	78.7%
	ARIMA	85.5%	71%	61.3%

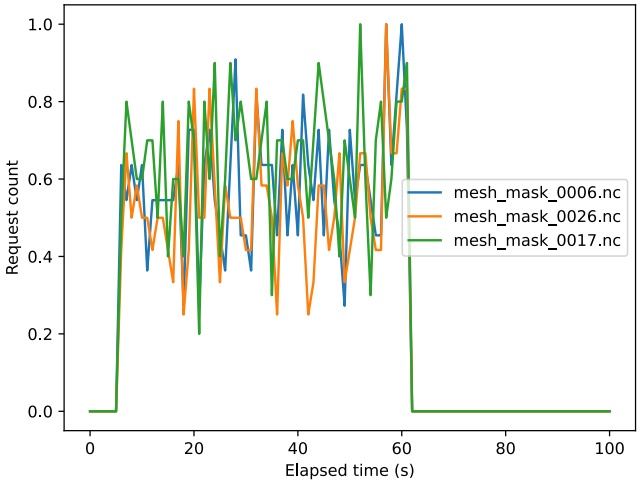

Fig. 2: Similarities between files from NEMO application

B. NAMD

Similar to the findings in NEMO, the inter-timeseries (TS) prediction models (NN15 and NN25) demonstrate their superiority over the intra-timeseries ARIMA models. Notably, for READ operations, NN15 and NN25 achieve an outstanding accuracy of 97.3%, while ARIMA lags behind at 94.6%. The same trend is observed for WRITE operations, with all three models (NN4, NN15, and NN25) attaining identical accuracy levels of 87.0%, outperforming ARIMA's accuracy of 83.9%.

The results reaffirm the benefits of considering global patterns and relationships across multiple time series, as exhibited by the inter-timeseries models, which lead to significantly improved forecasting performance. However, an interesting observation emerges from the WRITE operations, where the accuracy remains the same for NN4, NN15, and NN25, without any notable increase as the database size grows. This suggests that certain characteristics of WRITE operations might not exhibit significant variations that can be effectively captured by expanding the dataset further. As with NEMO, the findings emphasize the importance of choosing the right model and dataset size for accurate forecasting without introducing unnecessary complexity. However, for write, there is no significant difference observed between the three database configurations, this can be explained by the fact that write files behave almost the same, so having more of them in the database does not imply significant differences.

C. LQCD

In the context (LQCD), it is essential to acknowledge that a significant number of time series behave quite differently, which poses a challenge for our algorithm based on the similarity of access patterns. Furthermore, given the already very high accuracy of ARIMA, with 97.8% for reading and 95.2% for writing, it is challenging to achieve significant improvements. As seen in Table V, the inter-timeseries prediction models (NN15 and NN25) exhibit higher accuracy compared

TABLE IV: NAMD forecasting results evaluation

	Metrics	Accuracy	Precision	MCC
READ	NN4	90.1%	79.9%	73.3%
	NN15	97.3%	94.5%	92.7%
	NN25	97.3%	94.5%	92.7%
	ARIMA	94.6%	89%	85.4%
WRITE	NN4	87.0%	72.9%	64.3%
	NN15	87.0%	72.9%	64.3%
	NN25	87.0%	72.9%	64.3%
	ARIMA	83.9%	66.6%	55.9%

to the intra-timeseries ARIMA models for both READ and WRITE operations. For READ, NN15 and NN25 achieve impressive accuracy levels of 98.3%, while ARIMA lags slightly behind at 97.8%. Similarly, for WRITE operations, all three models (NN4, NN15, and NN25) attain comparable accuracy levels of 95.7%, outperforming ARIMA's accuracy of 95.2%.

Despite the algorithm's limited improvement due to diverse file behaviors, the precision achieved remains relatively high. This can be attributed to the prevalence of inactive files in the LQCD system. As observed in Table V, NN15 and NN25 achieve precision scores of 96.6%, whereas ARIMA reaches 95.7% for both READ and WRITE operations. In many instances, a substantial number of files remain inactive, and predicting them as null can be sufficient to achieve good precision scores. This is an area where traditional ARIMA approaches excel, as they can effectively capture and model such stable behaviors.

TABLE V: LQCD forecasting results evaluation

	Metrics	Accuracy	Precision	MCC
READ	NN4	97.3%	94.6%	92.9%
	NN15	98.3%	96.6%	95.4%
	NN25	98.3%	96.6%	95.4%
	ARIMA	97.8%	95.7%	94.3%
WRITE	NN4	95.6%	91.3%	88.4%
	NN15	95.7%	91.5%	88.7%
	NN25	95.7%	91.5%	88.7%
	ARIMA	95.2%	90.4%	88.7%

D. Time consumption

We can observe in figure 4 that the inter-file approach exhibits a temporal complexity of the same order as the intra-file approach, especially when the database size is 4. It is evident that the prediction time increases with the size of the database. This highlights the importance of finding the right trade-off between execution time and accuracy.

This comparison of execution time do not include the additional steps involved in ARIMA, such as tuning that increases linearly with the the number of tested combinations as we can see it in figure 5, stationarity checks, and eventually stationnarisation and destasionnarisation which are variable

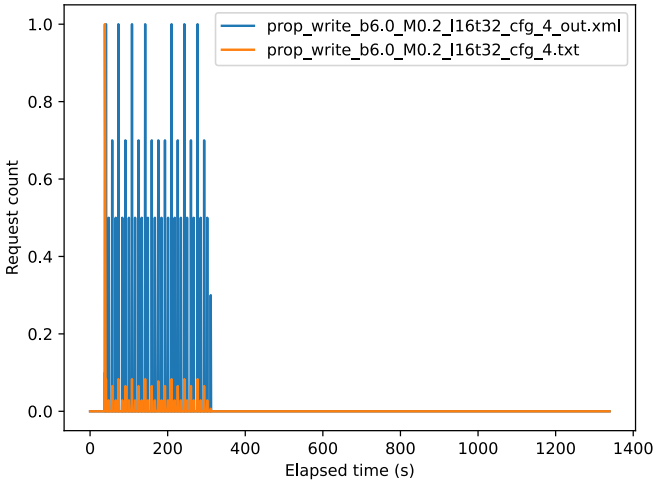

Fig. 3: Exemple of similarity between file for LQCD app on write

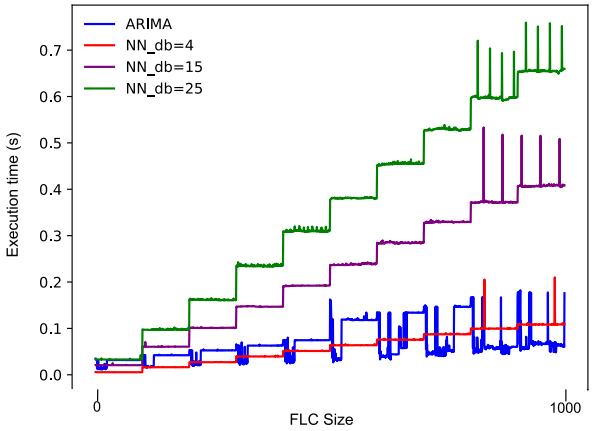

Fig. 4: Execution time of the forecasting algorithm

but non-negligible costs. These steps can be computationally expensive and time-consuming. In contrast, neighrest neighbor prediction may not require these additional preprocessing steps, making it potentially faster overall.

VII. Conclusion and Further Works

In this study, we have demonstrated the benefits of adopting an inter-timeseries approach, which not only showed significant improvements in accuracy across all tested scientific applications, but also maintained a balance with reasonable execution time. Our findings show significant improvements across all tested applications. Notably, the NEMO application's precision saw an increase, going from 65.4% to 85.7%. The results we obtained are promising, regarding time series ranking prediction accuracy. Our proposed inter-timeseries method increses performance, achieving accuracy rates ranging from 72.9% to 83.8% in challenging scenarios

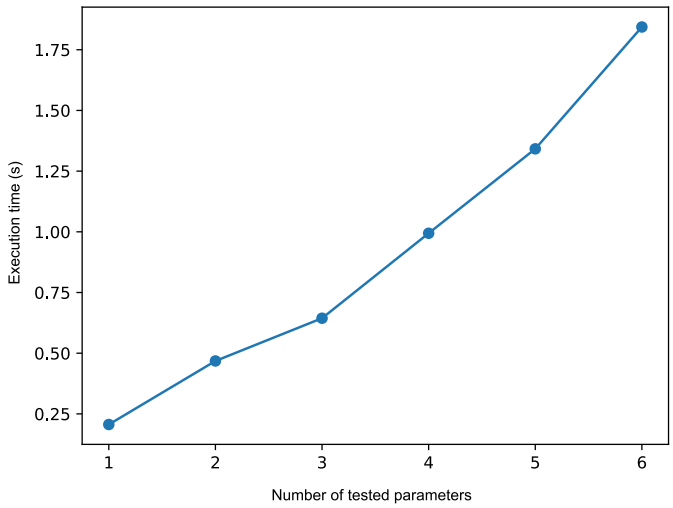

Fig. 5: Execution time of the tuning algorithm

and up to 97% in best cases. These findings demonstrate the effectiveness and robustness of our approach in predicting the order of time series activity. Looking ahead, we aim to further enhance our approach by optimizing the execution time. To achieve this, we plan to explore clustering techniques for file types. By analyzing and grouping clusters separately, we expect to enhance the process and reduce execution time even more.

REFERENCES

[1] Dima Alberg and Mark Last. Short-term load forecasting in smart meters with sliding window-based arima algorithms. *Vietnam Journal of Computer Science*, 5:241–249, 2018.

[2] Kasun Bandara, Christoph Bergmeir, and Slawek Smyl. Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. *Expert systems with applications*, 140:112896, 2020.

[3] Peter Bloomfield. *Fourier analysis of time series: an introduction*. John Wiley & Sons, 2004.

[4] Francieli Zanon Boito, Ramon Nou, Laércio Lima Pilla, Jean Luca Bez, Jean-François Méhaut, Toni Cortes, and Philippe OA Navaux. On server-side file access pattern matching. In *2019 International Conference on High Performance Computing & Simulation (HPCS)*, pages 217–224. IEEE, 2019.

[5] Christine TH Davies, E Follana, A Gray, GP Lepage, Q Mason, M Nobes, J Shigemitsu, HD Trottier, M Wingate, C Aubin, et al. High-precision lattice qcd confronts experiment. *Physical Review Letters*, 92(2):022001, 2004.

[6] K Lalitha Devi and S Valli. Time series-based workload prediction using the statistical hybrid model for the cloud environment. *Computing*, 105(2):353–374, 2023.

[7] Roshani W Divisekara, GJMSR Jayasinghe, and KWSN Kumari. Forecasting the red lentils commodity market price using sarima models. *SN Business & Economics*, 1(1):20, 2020.

[8] Bin Dong, Xiuqiao Li, Qimeng Wu, Limin Xiao, and Li Ruan. A dynamic and adaptive load balancing strategy for parallel file system with large-scale i/o servers. *Journal of Parallel and distributed computing*, 72(10):1254–1268, 2012.

[9] Everette S Gardner Jr. Exponential smoothing: The state of the art. *Journal of forecasting*, 4(1):1–28, 1985.

[10] Andrew Harvey and Vitaliy Oryshchenko. Kernel density estimation for time series data. *International journal of forecasting*, 28(1):3–14, 2012.

[11] Rohit J Kate. Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery*, 30:283–312, 2016.

[12] Adrian Khelili, Sophie Robert, and Soraya Zertal. Filip: A file lifecycle-based profiler for hierarchical storage. *Infocommunications Journal*, 14(4):26–33, 2022.

[13] Lei Li and B Aditya Prakash. Time series clustering: Complex is simpler! In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 185–192, 2011.

[14] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15:107–144, 2007.

[15] Robert B Penfold and Fang Zhang. Use of interrupted time series analysis in evaluating health care quality improvements. *Academic pediatrics*, 13(6):S38–S44, 2013.

[16] Donald B Percival and Andrew T Walden. *Wavelet methods for time series analysis*, volume 4. Cambridge university press, 2000.

[17] J. C. Phillips, D. J. Hardy, J. D. C. Maia, J. E. Stone, J. V. Ribeiro, R. C. Bernardi, R. Buch, G. Fiorin, J. Henin, W. Jiang, R. McGreevy, M. C. R. Melo, B. K. Radak, R. D. Skeel, A. Singharoy, Y. Wang, B. Roux, A. Aksimentiev, Z. Luthey-Schulten, L. V. Kale, K. Schulten, C. Chipot, and E. Tajkhorshid. Scalable molecular dynamics on CPU and GPU architectures with NAMD. *Journal of Chemical Physics*, 2020. DOI : 10.1063/5.0014475.

[18] F. Sevault, S. Somot, and J. Beuvier. A regional version of the NEMO ocean engine on the Mediterranean Sea: NEMOMED8 user's guide. Technical report, Meteo-France, CNRM, 2009.

[19] Jin Shieh and Eamonn Keogh. i sax: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631, 2008.

[20] Yunsheng Song, Jiye Liang, Jing Lu, and Xingwang Zhao. An efficient instance selection algorithm for k nearest neighbor regression. *Neurocomputing*, 251:26–34, 2017.

[21] Mathieu Stoffel, François Broquedis, Frédéric Desprez, and Abdelhafid Mazouz. Phase-ta: Periodicity detection and characterization for hpc applications. In *HPCS 2020-18th IEEE International Conference on High Performance Computing and Simulation*, pages 1–12. IEEE, 2021.

[22] Nancy Tran and Daniel A Reed. Arima time series modeling and forecasting for adaptive i/o prefetching. In *Proceedings of the 15th international conference on Supercomputing*, pages 473–485, 2001.

[23] Ruey S Tsay. *Analysis of financial time series*. John wiley & sons, 2005.

[24] Frans Van Den Bergh, Konrad J Wessels, Simeon Miteff, Terence L Van Zyl, Albert D Gazendam, and Asheer K Bachoo. Hitempo: a platform for time-series analysis of remote-sensing satellite data in a high-performance computing environment. *International journal of remote sensing*, 33(15):4720–4740, 2012.

[25] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095, 2007.