Efficient Seasonal Forecasting of Application Demand with ELF

by

Priscilla Wu

S.B., Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

Author . . . .
Department of Electrical Engineering and Computer Science
May 14, 2021

Certified by . . . .
Samuel Madden
Professor of Computer Science, MIT
Thesis Supervisor

Certified by . . . .
Alma Dimnaku
Principal Software Engineer, NetApp
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Efficient Seasonal Forecasting of Application Demand with ELF

by

Priscilla Wu

Submitted to the Department of Electrical Engineering and Computer Science
on May 14, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Increased use of data insights to guide ventures have led to an explosion of needs in data services such as data accessibility, mobility, availability, and protection. Particularly in cloud enterprises, this expansion of data services has led to an increased need of AIOps, or intelligent systems that can offer consistent operation while dynamically adjusting their operation for the data services requested. In the field of storage systems, self-management features include proactive management of resources through knowledge of demand and their changing patterns. Previous research on classification, forecasting, trending, and pattern recognition in storage workloads have concluded that there is no universally best predictor for all workload patterns. In addition, these researched methods and their comparisons focus more heavily on accuracy without considering the limitations on overhead and computation power present in a system-oriented approach. This thesis analyzes design tradeoffs and presents ELF, a generic forecasting algorithm of storage workload data that optimizes computation costs in the context of a real-life production system. ELF takes advantage of the fact that the majority of storage workloads possess activity too simple to warrant complex forecasting models. Using a customized classification approach, ELF selects the appropriate predictive model based on the workload's observed activity and produces accurate forecasts 92 times faster than a generic baseline algorithm while storing 97.5% less data.

Thesis Supervisor: Samuel Madden
Title: Professor of Computer Science, MIT

Thesis Supervisor: Alma Dimnaku
Title: Principal Software Engineer, NetApp

Acknowledgments

There are so many people that have helped and supported me on this journey to create ELF. I would first like to thank my team at NetApp, for welcoming me into the family and providing me the opportunity to pursue and develop something meaningful. My supervisor Alma, who met with me almost daily and was a source of constant support and guidance. My coworker Deepan, who taught me the reins around NetApp and took the lead in much of the ML model research. And my manager Rakesh and my mentor Rosa, for their regular check-ins of advice, support, and motivation.

At MIT, I would also like to thank my advisor, Sam, for bearing with me as I struggled to convey my thoughts onto paper and for the constant proofreadings and edits. And my undergrad advisor, Fredo, for keeping me accountable and motivated to work on this thesis.

Finally I would like to thank my friends and family. Nanette, my roommate, who struggled through all the same classes with me this semester and was a constant through the COVID times. All my friends that I randomly badgered for their suggestions and opinions as well as for their support and commiseration. And finally, my family, for all their regular check-ins, advice, and prayers.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data insights have become the cornerstone for business ventures and future designs. Particularly in enterprise cloud computing, this has led to a sharp increase in various data services offered. In storage systems, such services can include data accessibility, mobility, availability, and protection. Due to this explosion of services, a new need has arisen to aide users and customers in their management. Much of recent storage management system developments have been in the pursuit of a self-managing, hands-off system, often in the form of AIOps, or Artificial Intelligence for IT Operations. The development of such features relies on accurate predictors of workloads, which can then lead to proactive and automated service responses. As accurate forecasts by nature include the detection of regular data patterns, the development of more advanced models should also support the identification of abnormalities and changes in pattern. Are changes in pattern a transient abnormality or a non-transient change in activity? In addition, the trend of any particular resource operation may gradually change over time. Accurate models must be able to detect these changes in trend and adjust themselves accordingly. Models that can accurately account for these circumstances are more widely desired in general use cases including self-managing systems.

As such, most time series predictors are designed for accuracy, but often without the needs of a system kept in consideration. In particular, for popular forecasting methods, the more accurate the predictor, the more complex and computationally heavy the model is. In a storage system where tens of thousands of workloads are managed simultaneously, running a computationally complex predictor that requires large amounts of data for each workload is infeasible. Thus, in designing and choosing an ideal predictor for all the workloads in an ongoing system, one must optimize for accuracy while considering scalability as well as storage or computation time restrictions.

Furthermore, in real-life applications, workload data is presented in streams. The massive amounts of data at rest that are used in many time series analyses are not practical for use in online situations, as the data is too large to continuously store in cache and the time needed to pull the data is too significant. In a real-life system, analytic assessments should also be provided in real-time, without needing to wait hours for data to be pulled. Consequently, in order to provide real-time forecasts, evaluated forecasting approaches and predictive analysis techniques should be compatible with data streams rather than solely using stale data.

1.1 Motivation

The storage system at NetApp on which ELF was developed manages tens of thousands of workloads simultaneously. The large number of workloads meant that scalability was a primary goal in developing this algorithm. Consequently, attributes such as efficiency, lowered computation costs, and lightweightedness were prioritized. In evaluating methods on these attributes, we focused on two aspects: computation time and storage requirements. As ELF must run concurrently with other services, the amount of memory available for this algorithm was limited. In addition, in consideration of the fact that a majority of the computation time for workload analysis in the current system is attributed to pulling stale data from the database, methods revolving around streaming calculations and statistics were studied in order to minimize the amount of data to be pulled.

The system additionally monitors several different metrics for each workload, such as IOPS, capacity used, and latency. The different metrics and the potential resulting insights motivated a need for a generic algorithm. Rather than a method that can only be used with one specific metric, a method designed to be generalizable to several metrics would serve the greatest benefit and versatility.

1.2 Contribution

This thesis studies the problem of efficient forecasting of streaming storage data. The focus on storage system workloads presents different considerations than generic time series forecasting problems. It impacts method accuracy, as methods may perform differently on data from various environments and sources, while introducing limitations on computational overhead.

The technique we present, ELF, involves classification of the given workloads to best utilize various forecasting methods and maximize accuracy while minimizing computation. With a focus on lightweightedness as well as accuracy, ELF is designed for use in a real-life production system. In addition, it is created to be generic and modularizable. Specifically, it allows different portions of the algorithm pipeline to be detached and used to fulfill other services or analytics. As a whole, ELF can be run as a parallel service that does not require integration into the environment to be run. Its generic nature allows it to be used with any metric; although this thesis primarily studies its use with the I/Os per second (IOPS) metric, ELF's generalizability to other metrics such as latency or capacity is further discussed in Section 4.4. Metric generalizability allows for ELF to be used in many different use cases such as provisioning, planning, or protection.

As a general overview, ELF classifies workloads based on their activity into four predefined classes on a 24-hour basis. Given its classification, each workload is then forecasted using a predictor tailored for workload activity of that class. Generally, ELF runs simple models on workloads with simple activity and complex models on workloads with complex or seasonal activity. Using this approach, when run on about 20,000 workloads of real data, ELF performs 92 times faster while storing 97.5% less data than a generic baseline algorithm that does not utilize a classification approach.

Chapter 2

Background

2.1 System Description

The storage system on which ELF was developed is a storage array consisting of physical and logical components. A physical controller manages drives that provide the physical storage for data. Drives are logically grouped into pools, which can then be used to created volumes. Volumes are the logical component through which a host server can send I/Os and access storage on the storage array. For each volume, data metrics such as IOPS, capacity, and latency are collected at 5 minute intervals. The streamed workload data is then stored into a database for future analysis. Then, every 24 hours, the system pulls the stale data from the database to conduct heavy analyses and forecasts. ELF was designed in consideration of this current system process, using a database containing data from about 20,000 volume workloads and conducting its offline computations every 24 hours.

Throughout this thesis, "cache" will refer to RAM on the controller and "online" computations will be used to describe calculations that are done while the data is being streamed every 5 minutes and the results of which would be stored in cache. "Offline" computations will refer to the heavier calculations and model creations done every 24 hours, where the input data is the data pulled from the database. With regard to online computations, as ELF would be running concurrently with other programs, our goal was to limit cache memory usage to 30 MB of data at a time.

2.2 Workload Characterization and Classification

In order to forecast or analyze workloads, understanding the characteristics of the data is imperative. Workload activity changes in a variety of manners and its analysis allows for comprehension of demand and existing patterns in storage workloads. For example, Kim et al. researched I/O workload characteristics by studying different metrics of the workloads, such as I/O bandwidth distribution, read to write ratio, and request size distribution [17]. In this research, metric patterns and distributions were studied through methods such as CDF and PDF functions and percentile analysis. Then, when modeling the distributions, it was found that the read and write I/O bandwidth usage as well as inter-arrival time of requests could be modeled as a Pareto distribution. Understanding workload characteristics is also important when recognizing patterns and characteristics that are application or environment specific. For example, an analysis of disk drive workloads measured in various systems resulted in the discovery of characteristics that were common across all traces as well as characteristics that were specific to application or to environment [21]. In order to develop generalized classification techniques, common workload characteristics must be identified. Thus when designing ELF, characterization of workload activity was conducted to further understand the common characteristics present in the data.

In addition to understanding the characteristics of I/O workloads, classifying them into various groups allows for more in-depth analysis. Seo et al. developed a set of such classifiers by first determining a list of I/O trace features that best represented the workloads [24]. Using various clustering algorithms, the features were used to determine representative access patterns of general workloads. Wang et al. studied clustering based on other certain characteristics of the data, such as trend, seasonality, and periodicity [26]. Analysis of different kinds of clustering algorithms was also explored in Chethan et al. [6]. The classes determined by Seo et al. are hierarchical by nature, allowing them to be organized into a tree-like structure, thereby constructing a decision-tree-based classifier to classify other I/O trace samples. Another approach is explored in M. Alshawabkeh et al. [1], which classifies storage workloads into tiers of I/O intensity by using Markov chains to form clusters of similarly behaving workloads. Other studies have also executed storage tiering through a Markovian-based model to capture patterns of high or low traffic intensities [30]. By capturing the duration of these patterns, the model also distinguishes temporal patterns such as weekday/weeknight or day/night activity. In classifying workloads, ELF takes a computationally simpler approach than most of these algorithms. In a relatively linear fashion, workloads are classified into four different classes on the basis of the level, range, and seasonality of its activity.

2.2.1 Characterization

When characterizing our given workloads, we decided to primarily focus first on the I/Os per second (IOPS) metric of the storage workload data. As ELF is intended to be generic, once it is fully developed, it can then be generalized to other metrics. In addition, though information from different metrics can be combined to gather further insight about a workload [13][3], ELF only considers one metric at a time.

In studying IOPS storage workloads, we detailed characteristic attributes of such workloads. First, though some workloads possess explicit seasonal activity or patterns, such as a regular burst in activity every 30 minutes, a majority of IOPS workloads consistently display little to no activity. In addition, among workloads with seasonal activity, lengths of detected patterns vary greatly from workload to workload. At a time scale of about half a day, pattern length can range from 30 minutes to 6 hours. In increasing the time scale, patterns can also be found at a daily, weekly, or monthly granularity. Aside from patterned and low activity workloads, other workloads may exhibit irregular, nontrivial activity with no observable pattern. These workloads are characterized with bursty and random activity.

Specifically for the workloads in our system, the activity of almost all the workloads falls below 10,000 IOPS. With significant activity (defined as having activity above 100 IOPS), the total range of operation of individual workloads usually resides around a few hundred to a few thousand IOPS. Naturally, if the activity is higher, the range of operation is also larger. However, despite the differences in activity level

Types of Workload Classes	
Type	**Activity Characteristics**
Idle	Little to no activity
Constant	Narrow range of operation
Seasonal	Explicit, repeating patterns
Random	No observable pattern; bursty activity; wide range of operation

Table 2.1: Different classes of workloads defined by characteristic activity. Examples of each type of workload are shown in Figure 3-2.

between individual workloads, workloads themselves generally operate at a consistent level of activity. For instance, even if the range of operation of a particular workload is large, the median of the data would stay relatively constant. Though level changes in the data do occur, they are relatively infrequent (occurring once or less in a month for any given workload). These observations show that, among IOPS workloads, there is little to no trend expected in the data.

These characterizations were the justification behind much of the overall algorithm design. Table 2.1 details workload classes that were defined through consideration of these observed workload characteristics. The methodology of classifying workloads into each of these classes is outlined in Section 3.4. In addition, not only did the observed typical activity motivate the classification of the workloads, but the significant observation that the majority of workloads had either simple or little to no activity indicated that the majority of workloads did not warrant complicated predictive algorithms.

2.2.2 Classification

Though there are many different types of characteristics of time series workloads that we could utilize to classify our workloads, such as trend, skewness, or chaos [28][27], we focused mainly on seasonality and self-correlation. In classifying workloads based on their seasonality, the seasonality of the workload must first be detected. Algorithms that do this utilize a variety of different methods such as auto-correlation functions, partial correlation functions, or Fourier transforms, to name a few. Though many of these methods were explored, for logical and computational simplicity, solely a combination of auto-correlation functions and heuristics was used in ELF to determine seasonality.

2.3 Time Series Analyses

While many classification approaches may naturally lend themselves to pattern prediction, there are various time series forecasting techniques that can accurately predict workload activity, usually through machine learning. Studies comparing the accuracy of various forecasting strategies have been extensive, including methods from simple naïve forecasting and moving average to more complicated models such as ARIMA and TBATS [14] [15] [25] [16]. While naïve methods are quick and lightweight, they suffer from poor accuracy. Conversely, while complicated models such as ARIMA can be much more accurate in forecasting, the overhead computation cost is heavy.

In addition, another evaluating factor of an accurate forecasting technique is its ability to forecast different types of time series. As different models will perform better with different types of data, it is difficult to know which workload predictor will work best for any particular workload as there is no set standard [15] [16] [14]. To combat this issue, Herbst uses a dynamic approach that selects the suitable method for a given situation based on a decision tree and direct feedback cycles [14]. With ELF, we take a simplified version of this approach where different classes of workloads are forecasted with different predictors.

Furthermore, in real-life systems, forecasts are regularly used to fulfill requests from dynamic use cases such as resource provisioning. This requires time series forecasters that are both efficient as well as accurate. The issue of workload prediction for use in dynamic provisioning has been commonly explored [25][3] and is also examined in this thesis.

2.3.1 ML Models

In selecting a machine learning model for use, a baseline model must first be determined. As a model traditionally used in time series forecasting and also commonly used as a baseline for other models [29][10], the ARIMA model was used as a baseline for accuracy and efficiency. ARIMA models are made of an autroregression portion (AR), a moving average portion (MA), and an integration portion (I). To customize these three parts for the workload, three parameters (p, d, and q) are passed in. The seasonal version of the ARIMA model, SARIMA, has twice the number of parameters (p, d, q, and P, D, Q) to represent the seasonal component.

In addition to the ARIMA model and its variants, other models mentioned in this thesis include multiple linear regression, Holt-Winters, and Facebook Prophet. A multiple linear regression model captures relationships between two or more features and a response variable by fitting a linear equation to the given data. When used in time series forecasting, features are set as the values at preceding time steps and the response variable is set as the value at the next time step. Holt-Winters is a triple exponential smoothing model that models three aspects of time series behavior: average value, trend, and seasonality. In contrast, Facebook Prophet is a complex additive regression model best suited for workloads over extended time periods with multiple seasonalities and irregular events. Depending on the data, ELF uses either a Holt-Winters or Facebook Prophet model.

2.4 Data Streams

In addition to heavy computation cost, another downside to forecasting using machine learning models is the large amounts of data needed to train the models. In situations where methods are studied and compared only for their accuracy, finding and using extensive datasets for training is inconsequential. However, in a real-life, long-running system, assessments need to be made in real-time. Thus, storing large quantities of incoming data for computation is, storage-wise, infeasible, and pulling a dataset of that size for computation would take an unideal amount of time. In addressing this issue, methods that operate on streaming data are considered. Past research has explored time series prediction and analytic methods on streaming data for various use cases. For example, Rodrigues et al. performs an online prediction of streaming sensor data through incremental clustering and neural network training [22]. Ziehn et al. conducts similarity evaluations on time series in real-time using streaming data [31].

Using analytic methods that are compatible with data streams are ideal when classifying data as well. Such analytic methods often involve the computation of counts, quantiles, and/or histograms. There are several algorithms that allow the computation of quantiles in one-pass, allowing continual computation in an ongoing system environment. The most popular of these seems to be the algorithm developed by Greenwald-Khanna [12][9]. Cormode et al. expands on this algorithm by accounting for distributions containing skew, a common feature of traffic streams [7]. In addition to one-pass algorithms, extensive research and development has been done in computing counts, quantiles, and histograms over sliding windows of data [19][2][8][11]. As we were limited in memory storage to compute these types of calculations, efficiency in data storage was also of note. Buragohain and Suri examine various algorithms that compute quantiles on streams and evaluate them on space efficiency [5]. Ma et al. [18] study "frugal streaming," in which only one or two units of memory are needed to estimate quantiles. In terms of calculating counts in one-pass or with windowed streaming data, Welford's method to calculate variance in one pass [23] is a commonly known approach. In addition, Becchetti and Papapetrou et al. present sketches to calculate maxes and mins of windowed data streams [4][20].

2.4.1 Streaming Modules

In the development of ELF, Greenwald-Khanna and Welford's method were briefly used to keep counts and statistics over the streamed data. However, we found that these methods, despite being the simplest out of most algorithms mentioned previously, were still too complex for our use case. Rather than keeping a complex data structure to calculate hyper-accurate statistics over streamed data, we sacrificed some accuracy in statistics to keep a vastly simplified structure instead. In addition, when considering machine learning models that are compatible with data streams, we found that most models were infeasible due to our limitation on memory usage.

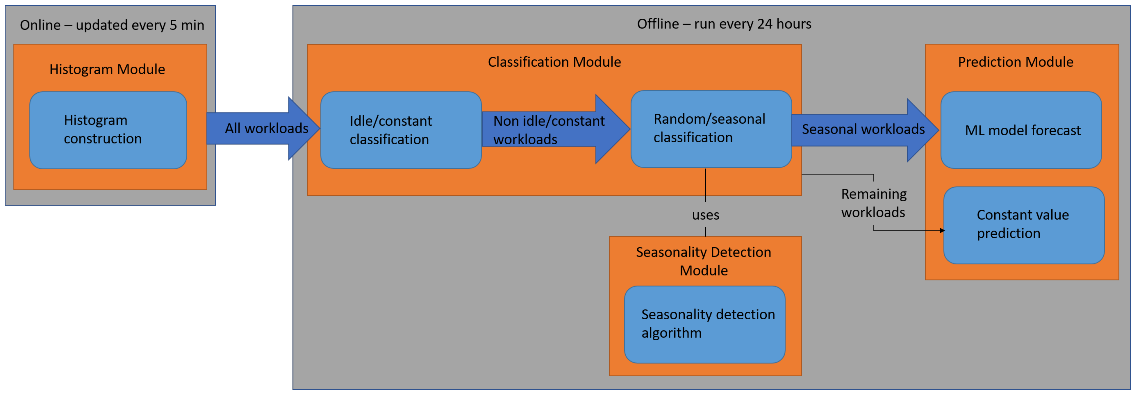
Chapter 3

Methodology

3.1 Goals and Considerations

In designing ELF, we considered several trade-offs in computation. A primary one was deciding which aspects of the method to run online and which to run offline. In a perfectly optimized algorithm, all computations would be run incrementally online as data streams in. With this approach, there would be no need to set aside time each day to pull several gigabytes of data from the database and then run analyses over them. However, this design is impractical as streaming calculations would thus be limited to a run time of 5 minutes (the time difference between each data point collection) and a computation size of about 30 MB, as input data and computed values from a streaming calculation would have to be stored in cache. While the time restriction of 5 minutes was not excessively restraining, the size restriction of 30 MB severely limited the complexity of computations that could be run online. Thus, a combination of calculations run online with data stored in cache and calculations run offline with data pulled from the database each day was needed.

To summarize the design goals of ELF, in order to reduce overall computation and create a more efficient algorithm, we planned to reduce the amount of data needing to be pulled from the database while also computing as many calculations as possible online. In forecasting workload activity, the portion that is the most computationally complex and requires the largest amount of data is the machine learning module. As



Online – updated every 5 min
Histogram Module
Histogram construction
All workloads
Offline – run every 24 hours
Classification Module
Idle/constant classification
Non idle/constant workloads
Random/seasonal classification
Seasonal workloads
Prediction Module
ML model forecast
uses
Remaining workloads
Constant value prediction
Seasonality Detection Module
Seasonality detection algorithm

Figure 3-1: Overview of ELF and its different modules. Arrows indicate the workloads that are sent to each specific module or sub module.

concluded from Chapter 2, the main technique behind reducing overall computation was to run simplified models over workloads that had simple activity. In doing this, heavy computational actions would be broken up or minimized simply by running them fewer times and on smaller sets of data.

Another trade-off to consider was whether to design the system in a modularized nature, where each module could be used as a separate microservice, but at the cost of some computational optimization. It was ultimately determined that the optimizations gained from a fully integrated system did not outweigh the benefits of a multi-use, versatile modular system.

3.2 Algorithm Overview

The overall design and flow of ELF is summarized in Figure 3-1. The pipeline can be broken down into four modules: histogram construction, workload classification, seasonality detection, and prediction. As an overview, the histogram module stores histogram data for each workload, the classification module classifies each workload based on its activity, the seasonality detection module is utilized by the classification module when classifying workloads, and the prediction module forecasts workloads based on their classifications. Out of the four modules, only the histogram module is run online. All other modules are run offline every 24 hours.

In general, each successive module in the pipeline requires more computationally intensive calculations and more data than the previous module but is run on fewer workloads. From the histogram module, which is run on all workloads and is the least computationally intensive module, to the seasonality detection module, which is more computationally intensive and requires more data but is only run on 10% of workloads, to the machine learning model forecast in the prediction module, which requires the most data and is the most computationally intensive but is only run on 2% of workloads. Though the pipeline is generally linear, modules can interact in different ways. For example, the classification module utilizes both the histogram module and the seasonality detection module to classify workloads and assign them to the appropriate prediction technique.

The following sections will outline each of these four modules and how they cooperate in order to efficiently classify and forecast tens of thousands of workloads simultaneously.

3.3 Streaming Histogram Creation

The first module in ELF is histogram creation, where a histogram with 10 bins is created and stored for each workload every 24 hours. Histograms are useful in that they can provide several statistics such as quantiles, ranges, or distributions without needing to store large amounts of data. Due to the considerable number of workloads in the system, calculating workload statistics from histograms rather than the entire workload dataset enables the reduction of computation load.

In designing this module, we considered tradeoffs between creating histograms for each workload incrementally as data was streamed and storing all 20,000 histograms in cache, or pulling 24 hours of data each day and creating the histograms offline. We determined that the benefits of redistributing offline computation to online and reducing the amount of data to be pulled offline outweighed the storage cost of keeping the histograms in cache. In addition, using a fixed-size histogram with predetermined ranges kept the cache cost relatively minimal. Another design tradeoff we considered was whether to integrate the histogram code into the program and system code to reap maximum computational efficiency or to separate out the code as an entirely detachable module. If these histograms were solely used for ELF and nothing else, the former option might be more attractive. However, as stated previously, histograms can provide numerous useful data statistics and metrics. Offering this histogram module as a separate microservice allows it to be utilized by other services as well. As such, this module was implemented as an independent microservice to expand its usability and versatility.

Of the four modules outlined in Figure 3-1, this is the only one that operates on streaming data. For each workload, the system stores a histogram of 10 bins, with predetermined bin ranges. For the IOPS metric, the bin ranges chosen are outlined below:

$$\begin{array}{ccc} [0-100] & (100-400] & (400-700] \\ (700-1,000] & (1,000-2,000] & (2,000-4,000] \\ (4,000-6,000] & (6,000-8,000] & (8,000-10,000] \\ (10,000-\infty) & & \end{array}$$

These bin ranges were defined in consideration of typical IOPS activity in storage workloads, with further justification for these bin ranges given in Section 3.4. Each histogram is kept in cache and updated every 5 minutes as each data point is streamed. Every 24 hours, the histogram for each workload is saved into the database and a new histogram is constructed over the next day. Thus, at any one point, only one histogram per workload is kept in cache. Historical records in the database store all daily histograms constructed from previous days, recording one histogram per day for each workload. In terms of memory usage, given that a single histogram has a size of about 144 bytes, with 20,000 workloads, less than 3 MB of data is kept in cache at any one point.

In the overall pipeline, these histograms are utilized both by the classification module and the prediction module. Since histograms are created for every workload and are small in size, they are readily used for simple computations. In fact, the first step of the classification module is to pull all the histograms of the day and use them to filter workloads with simple activity. For workloads classified with simple activity, the only data pulled from the database throughout the entirety of ELF is their histogram data.

3.4 Workload Classification

Of the offline processes, the classification module is the first to be run. In order to best determine which prediction method should be applied to each workload, this module classifies workloads into one of four groups, or states, based on their activity. These states were derived from the workload characterization as described in Section 2.2.1 and their computations are defined below:

- **Idle**: 95% of all points reside in the first bin of the histogram
- **Constant**: 95% of all points reside in any one bin of the histogram (besides the first)
- **Seasonal**: contains a detected seasonality longer than 30 minutes
- **Random**: not idle, constant, or seasonal

3.4.1 Idle and Constant Workloads

Workloads defined as having simple activity are classified as idle or constant. To keep computation for these workloads minimal, classification of idle and constant workloads only require the histogram data of the workloads. The first step of the classification module is to inspect the histograms of all workloads and classify them as idle, constant, or neither.

In defining an idle workload from a storage perspective, since an IOPS value smaller than 100 is considered trivial, a workload that operates primarily in that range is essentially idle. The first bin range of the histogram (0 - 100) was thus chosen to allow workloads with little to no activity to be determined quickly. If 95%

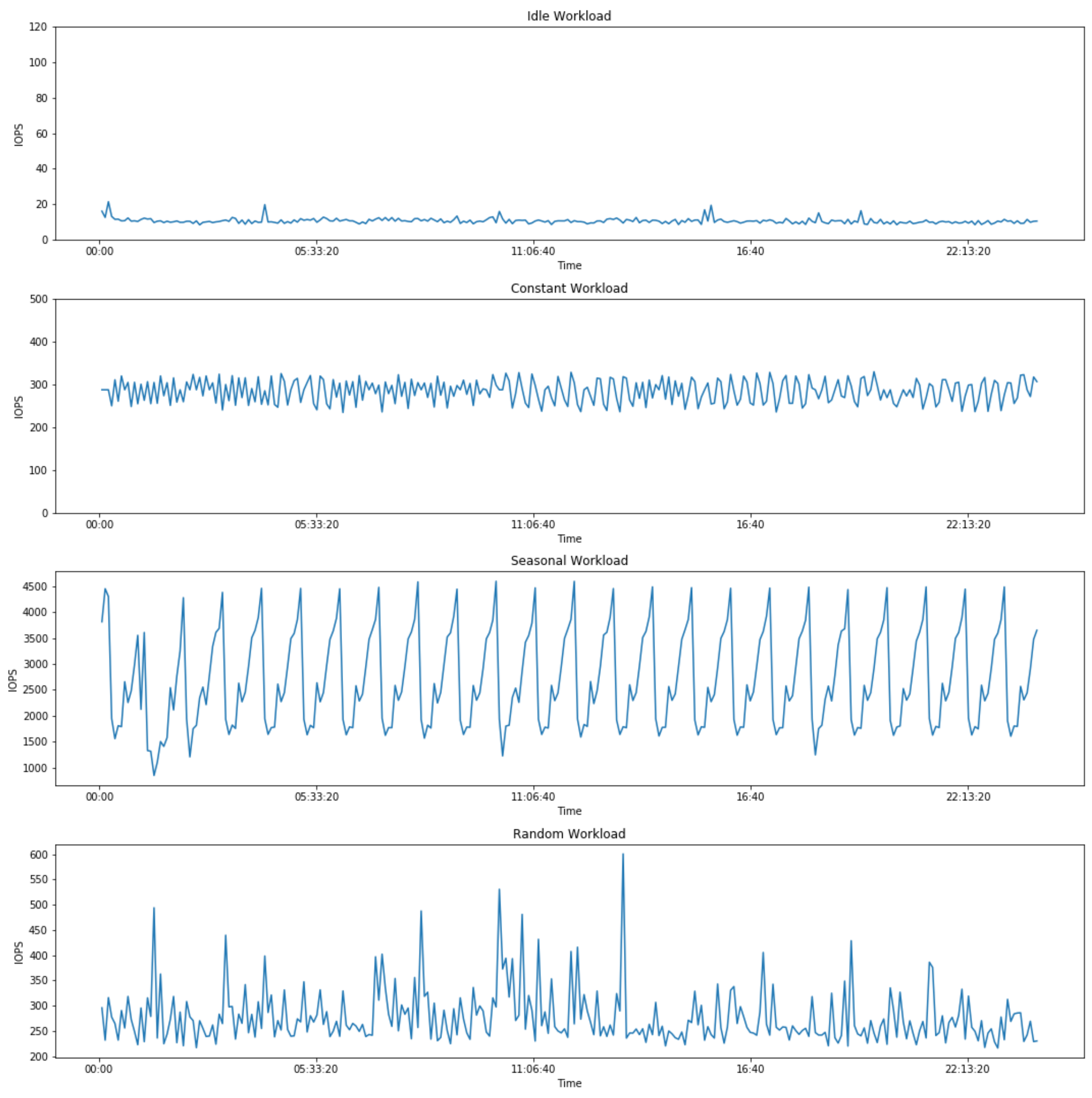


Figure 3-2: Example idle, constant, seasonal, and random IOPS workloads classified by the definitions stated in Section 3.4. Note that while each workload is an example of one of the four workload states, their specific activities are not representative of all workloads of that same type.

of a workload's IOPS activity lies within the first bin of the histogram, or under 100 IOPS, for that day, it is classified as idle. The 95% threshold is an empirically derived benchmark to filter out outliers and abnormalities.

Workloads with constant activity are classified using the remaining bin ranges. These remaining bin ranges detail thresholds of significantly different workload activity. For example, though the (8,000 - 10,000] IOPS bucket range is much larger than the (100 - 400] range, a difference of a few hundred IOPS is much less significant when the general level of activity is in the several thousands rather than the few hundreds. Using this rationale, if the majority of a workload's activity falls within one histogram bin, it is classified as constant.

Given that the classification and any subsequent computation for these two types of workloads only require histogram data, the total computation for simple workloads is relatively trivial. Since idle and constant workloads comprise about 90% of all workloads, classifying workloads in this manner enables us to save significant amounts of computation. As a note, since the classification of idle and constant workloads is the first filtering step in the algorithm pipeline (refer to Figure 3-1), idle and constant classifications take precedence over seasonal and random. Thus, even if a workload displays a seasonal pattern, it will not be classified as seasonal unless its range of activity is significant enough.

3.4.2 Random and Seasonal Workloads

For all non-idle and non-constant workloads, which compose about 10% of the workloads, additional data needs to be pulled. In order to classify seasonal workloads, workload data is run through a seasonality detection algorithm, to be described in Section 3.5. At a 5-minute data collection granularity, the seasonality detection algorithm requires about a day's worth of data (288 points). Thus, for the workloads not classified as idle or constant, the last 24 hours of IOPS data is pulled from the database to be run through the seasonality detection algorithm.

If a workload possesses a seasonal pattern longer than 30 minutes, it is classified as seasonal. Of the 20,000 workloads in the system, seasonal workloads comprise about 2%. The remaining workloads that do not have a detected seasonality are classified as random. Random workloads are generally characterized by bursty activity operating over a large range. Despite having nontrivial activity in contrast to idle and constant workloads, random workloads are also unfit for complex machine learning model predictions due to their widely unpredictable nature. Without a general trend or an established pattern, running random workloads through a machine learning model results in predictions with high validation error and consequently wasted computation.

3.4.3 Classification Discussion

Of these four groups of workloads, only workloads classified as seasonal have activity meaningful enough to be predicted using a machine learning model. Due to triviality or unpredictability, the remaining three groups are predicted using simpler means. To maximize efficiency, one of ELF's design goals is to reduce computation of workload forecasting by reducing the number of times modules with a high computational load are run. The histogram module enables this goal by keeping statistics summarizing the activity of each workload at a low computational and storage cost. Using the histograms, the classification module then trivially filters out workloads with simple or unpredictable activity, allowing the computationally heavy machine learning model to be run on fewer workloads (specifically, about 2%).

When expanding the use of ELF to different metrics, state definitions and bin ranges must be customized. The relevant classification states will depend on the metric being analyzed and the bin ranges will consequently follow the defined states. For IOPS, the first bin range was designed for idle classifications and the remaining ranges were designed to identify constant workloads. In general, defined classification states and bin ranges utilize workload characterizations to represent possible workload activity as well as ranges of significant activity.

3.5 Seasonality Detection

The seasonality detection module is used solely by the classification module to distinguish seasonal and random workloads. The seasonality detection algorithm used is an original method developed through the combination of heuristics and concepts presented in related work.

For this algorithm, 24 hours of data at a 5 minute granularity, or 288 points, is passed in as input. From a purely technical standpoint, the algorithm would operate similarly with almost half as many points, around 180, but a time period of 15 hours is not as significant as 24 hours when studying patterns in workload activity. And because much of workload patterns stem from societal time periods, such as a work day or work week, special care must be taken in choosing the amount of data to run through the algorithm. In addition, because offline processes are only run once every 24 hours, passing 24 hours of data as input ensures that all workload activity is analyzed when detecting patterns.

First, the input data is preprocessed in two steps: filtering and smoothing. Values smaller than the 1st percentile and greater than the 99th percentile are replaced with the median of the data, as pure filtering is avoided to keep time sequences constant. Next, the data is smoothed with a moving average of 3, where the first and last points are kept as their original value. This smoothing allows peaks formed by the next step of processing to appear more distinct, as shown in Figure 3-3.

After preprocessing, the data is auto-correlated to a lag of 60. As the auto-correlation function, or ACF, of a time series shows areas where the data is highly correlated with itself, lags greater than zero that display high self-correlation likely indicate the presence of a pattern. Thus, in order to identify patterns in the data, the algorithm searches for peaks in the ACF. However, since the ACF is only calculated to a lag of 60, patterns with a length greater than that are difficult to capture. In the circumstances where the pattern is too long to be captured at a 5 minute granularity, we look to larger granularities, to be discussed in Section 4.4.2. The number of lags calculated is a parameter that can be customized, but the specific value of 60 used

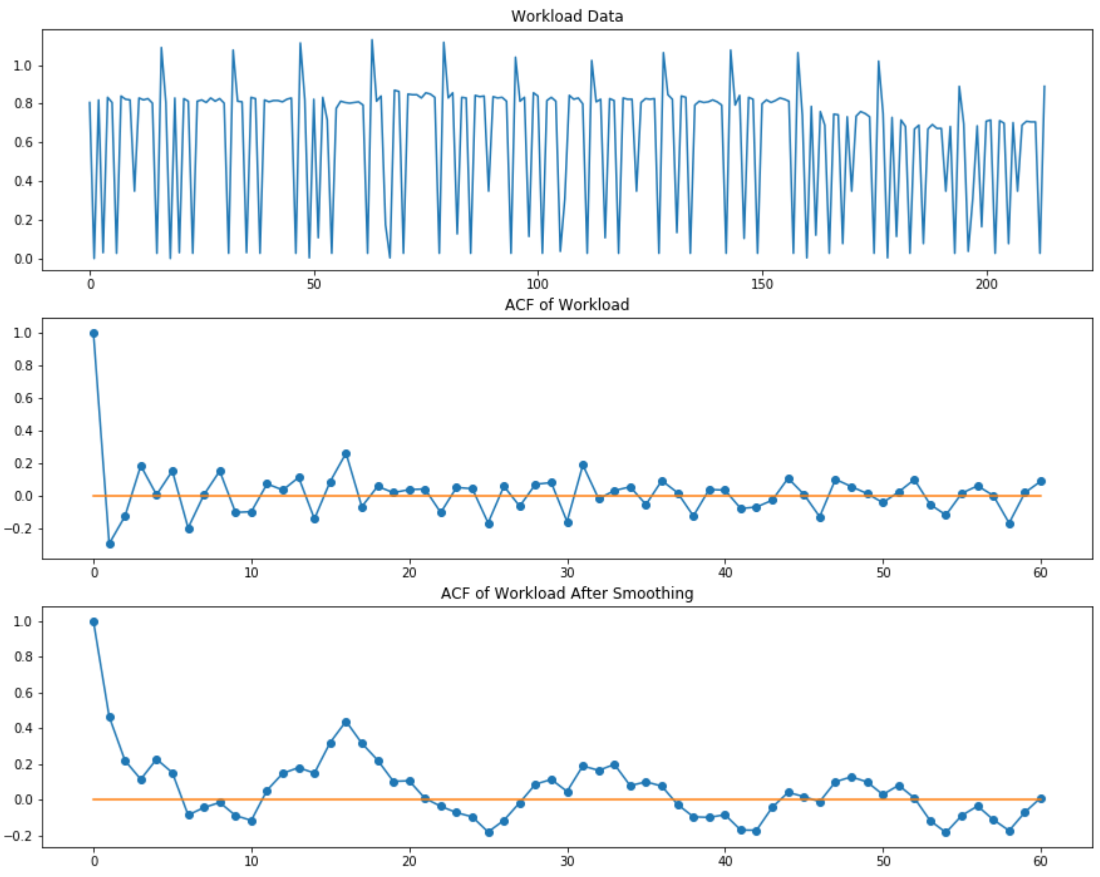


Figure 3-3: Given the workload data displayed in the top plot, the second two plots show the ACF of the data with and without smoothing. Note that the peaks become much more apparent after smoothing.

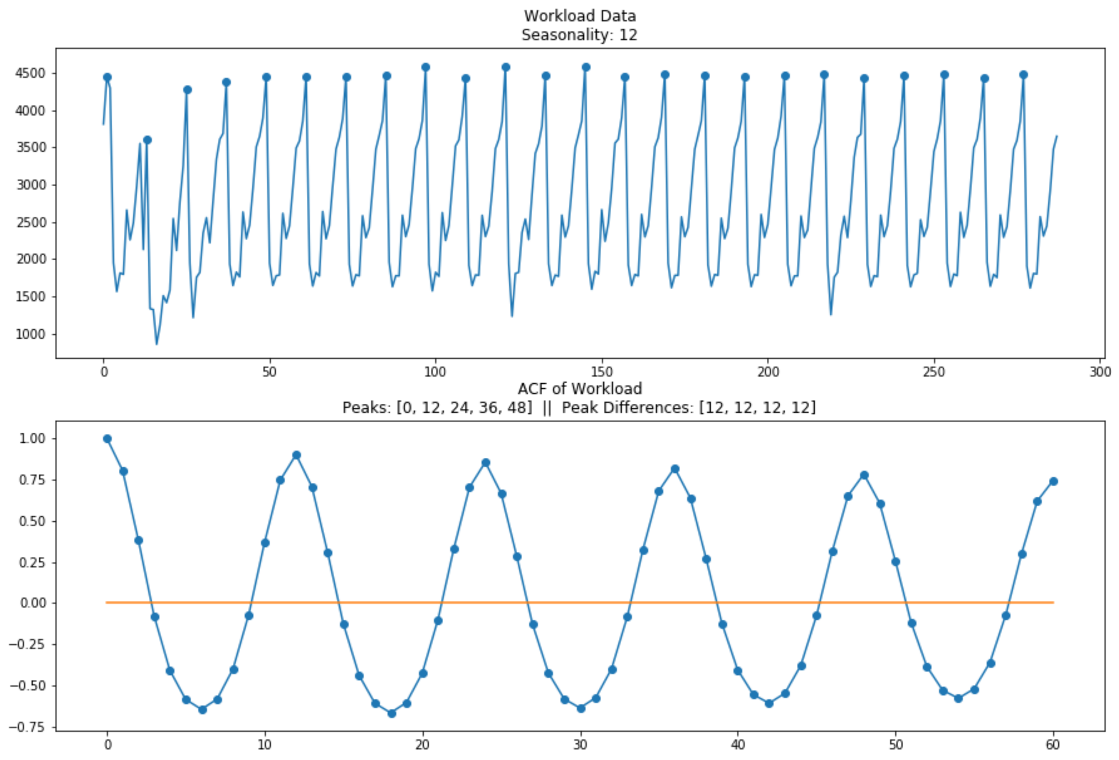


Figure 3-4: The ACF of a workload and the peaks and peak differences calculated from it. The resulting calculated seasonality is indicated on the workload data plot.

was determined empirically. As the ACF is a computationally expensive function, a lag of 60 was found to optimize computation cost without too much sacrifice in the types of patterns that we wanted to capture. With data at a 5 minute granularity, since relevant patterns are at most a few hours in length, a lag of 60 is sufficient.

Once the ACF is computed, lags at which peaks occur are detected. Peaks are defined as the max value in a range of consecutive positive points. Once these lags are identified, the differences between each of them are calculated, resulting in a list of potential pattern lengths. The final detected seasonality is the mode of these potential pattern lengths, provided that the frequency of the mode value is greater than one and the mode corresponds to a pattern length longer than 30 minutes (at a 5 minute granularity, this corresponds to a value greater than 6). Figure 3-4 displays example resulting calculations when this process is run on a workload.

Once the seasonality of the workload is determined, the seasonality detection module returns it to the classification module, which uses it to classify seasonal and random workloads. In addition, if a seasonality exists, the length of the pattern is returned for the classification module to pass to the prediction module.

3.6 Forecast and Prediction

The final module in ELF is the prediction module, which computes a 24 hour forecast for all workloads. Of the four states in our classification set, only workloads classified as seasonal are forecasted using a machine learning model. Idle and constant workload activity is too trivial to warrant the computation of a machine learning model and random workload activity is too unpredictable for a machine learning model to forecast accurately. This section will outline the prediction methods used for each classification state, particularly the machine learning model chosen to forecast seasonal workloads.

3.6.1 Constant Value Prediction

For workloads classified as any state other than seasonal, the predicted activity for the next day is a constant value prediction, the value of which as outlined below:

- **Idle**: zero
- **Constant**: median of the day's data
- **Random**: 75th percentile of the day's data

As workloads classified as idle have activity that is essentially zero from a storage management perspective, its prediction follows similarly.

The constant value prediction for constant workloads is the median of all values from that day. As the full day's worth of data is only pulled for workloads classified as random or seasonal, the median of constant workloads must be calculated solely using data from the histogram or stored in cache. Thus, in order to calculate a more accurate median, a total sum of values for each bin is stored for each workload. This means that a total of 20 values for each workload is now stored in cache, 10 histogram values and 10 sum values. The median of the day's data can then be more accurately estimated by calculating the mean value of the bin in which the 50th percentile would fall.

As the full day's worth of data for random workloads has already been pulled for use in the seasonality detection algorithm, the 75th percentile can be calculated straight from the data. As the day's dataset is small (288 points), the computation is trivial enough to calculate the actual percentile rather than estimate it using the histogram. The 75th percentile was chosen through a rule-of-thumb heuristic that the 3-quarter threshold provides a good representation of workload activity for general use. However, the prediction percentile could potentially be raised or lowered depending on specific use case. For example, if the prediction was to be used for provisioning, the percentile could be raised to 85 or 90. If the use case needed the most accurate representation of activity, the value could be lowered to return the mean or median.

3.6.2 Machine Learning Model Forecast

The machine learning model in the prediction module is the most computationally heavy part of ELF. Not only does the training and forecast of the model require additional computation time, but the model also requires a large amount of data in comparison to the previous modules. In order to present a forecast of 24 hours, an additional 3 days of data is required to train the model. Thus, for all seasonal workloads, the prediction module pulls the past 72 hours of data from the database. The following section will outline the different models that were explored and the ones chosen for use in ELF.

Initial Exploration

The exploration conducted to find the most appropriate machine learning model to suit our needs was done primarily using open source models and libraries. Models that were studied more extensively included multiple linear regression (MLR), ARIMA, Holt-Winters, and the Facebook Prophet model. As one of the most standard time-series forecasting models, ARIMA was used primarily as a baseline for timing and accuracy. Since the ML model would only be run on data that possess heavy seasonal components, we found that seasonal ARIMA (SARIMA) produced better accuracy on the seasonal workloads than ARIMA. In comparison to the forecast accuracy produced by ARIMA, SARIMA forecast accuracy was generally 10% better. However, due to the additional seasonal parameters that SARIMA needs, the seasonal model took several orders of magnitude longer to run than the nonseasonal ARIMA model. Nonetheless, because the primary aim when designing ELF was to find an alternate model that could match the accuracy of a baseline model while decreasing overall computation time, the achieved accuracy was of most interest for our baseline model.

The first model to be compared to the ARIMA models was multiple linear regression. The MLR model was able to match the accuracy of SARIMA and performed about 70 times faster than ARIMA. However, to achieve this speed and accuracy, the MLR model required much more data than either of the ARIMA models. With 6 features, where each feature was the value at the preceding timestep up to 6 timesteps backwards, we would need to input 6 times more data into the MLR model than what was needed for ARIMA.

However, beyond the problem of input size, both the ARIMA model and MLR possessed more pressing issues: they forecasted in steps, in which additional data needed to be provided for each forecast step. With a static set of data, both models were limited in the length of accurate forecasts they could produce. If the models were forced to predict for longer steps, their forecasts would become wildly inaccurate. Since the nature of the system required a model that could take in a static set of data and produce an accurate forecast of at least 24 hours, or 288 points at a 5 minute granularity, other models needed to be explored.

Holt-Winters and Facebook Prophet

After additional exploration, the two models chosen were the Holt-Winters model and the Facebook Prophet model. While both models are seasonal time-series forecasting techniques, they differ mainly in the types of seasonalities that are supported and the degree of customizable parameters. In testing these models, the grid search technique was utilized to find the parameter combination that resulted in the most accurate forecast. Grid search takes in multiple inputs for each given parameter, runs the model with all possible combinations of the inputs, and returns the best result. As it is a computationally expensive technique, in order to determine the most accurate model with reasonable computation time, both models were studied with and without the use of grid search. For both models, the test size, or forecast length, was one full day (288 points), as necessitated by the system design. In order to gain an accurate forecast of this length, the training size was 3 full days (864 points).

The Holt-Winters model is a seasonal forecasting method that primarily uses exponential smoothing. The model takes in seasonality, trend, and level as its three parameters to fit its three smoothing equations. In using grid search with Holt-Winters, other than the seasonality returned from the seasonality detection algorithm, seasonality inputs explored included the given seasonality plus or minus one as well as other pattern lengths detected by the seasonality detection algorithm that were not the most frequently occurring. However, most seasonal workloads at a 5 minute granularity were explicitly patterned (see Figure B-1 for examples) with a stable seasonality, level, and trend. Thus it was concluded that Holt-Winters could perform accurately even without the use of grid search, as the stable seasonality and lack of trend allowed parameters to be accurately set with one value.

The Facebook Prophet model, in contrast to Holt-Winters, is a more complex model with higher customization capabilities. The benefit of this model is its support for workloads with multiple seasonalities or irregular seasonalities (such as holidays in a yearly calendar). As such, its capabilities are preferred in forecasting workloads with light or more complex seasonal patterns as well as workloads with longer patterns, such as weekly and yearly patterns. Because the Prophet model's strength is in its customizable parameters and multiple components, which are used to describe more complex seasonalities, using grid search with this model to forecast workloads possessing such qualities is advantageous. Between the Holt-Winters model and Facebook Prophet with grid search, due to the nature of workload activity at a 5 minute granularity, the Holt-Winters model works best, with its comparably lightweight computational load and high accuracy. Section 4.4.2 will detail other circumstances and use cases where Facebook Prophet will be used in the system.

Chapter 4

Validation

As the original goal of this thesis was to develop a generic algorithm to forecast storage workloads that reduced computation but still performed accurately, ELF was evaluated over four characteristics: accuracy, storage space, computation time, and flexibility. In the validation experiments, a simulated environment was created where 20,000 workloads streamed 7 days of real data that was previously collected to the algorithm (either to ELF or the baseline) and validation statistics were recorded.

4.1 Accuracy

Since ELF is comprised of two parts, classification and forecasting, the evaluation of accuracy was also thus split. In general, workload classification accuracy was determined trivially or through manual inspection and analysis while workload forecast accuracy was determined through a variety of error metrics. For manual inspection, a subset of 200 workloads containing workloads with typical activity as well as special cases was examined closely. For the remaining workloads, further inspection only occurred when certain validation metrics or checkpoints revealed unusual activity.

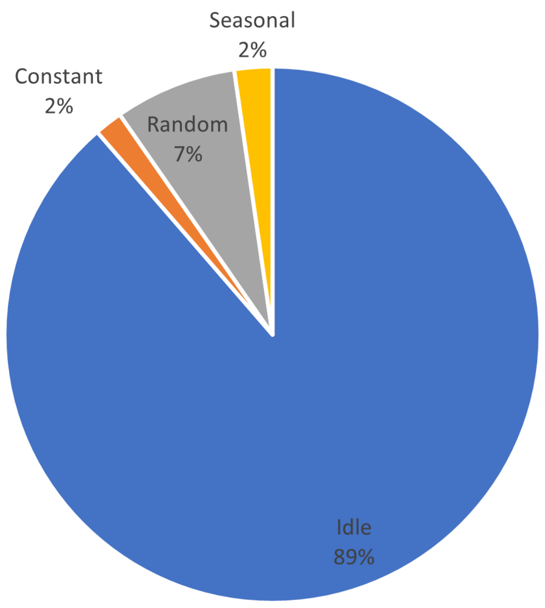


Figure 4-1: Average workload classification distribution over 7 days

4.1.1 Classification

Figure 4-1 shows the average classification distribution of workloads over a period of 7 days. From day to day, the distribution percentages varied very little, with the range of fluctuation remaining around 1%. The validation of these classifications was mainly conducted manually through graph visualizations and inspections of outliers or abnormalities. This section will detail the processes for how each individual state was evaluated for accurate classification.

Idle and Constant classifications

The validation for idle workloads is relatively trivial; since 100 IOPS was predetermined to be the threshold of significant activity, any workload that primarily operates under that threshold is classified as idle. "Primarily" was defined as 95% of all data points, a threshold chosen to filter out outliers that influenced the classification. This threshold was originally 99%, but was changed as it still allowed inactive workloads to be classified as random due to a few abnormalities.

Constant workloads were classified and validated in a similar fashion, also with 95% of points considered. The additional factor in constant classifications were the

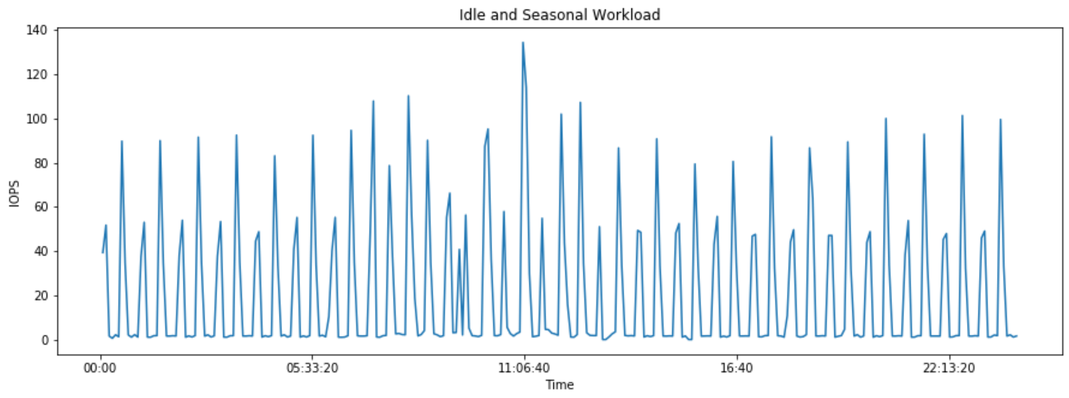


Figure 4-2: An example of a seasonal workload that operates primarily under 100 IOPS, so it is classified as idle.

bin ranges of the remaining histogram bins. These ranges were tuned according to both workload characterization and visual validation. Some incorrect classifications of constant workloads occurred when the level of activity straddled a bin range boundary. In these cases, since 95% of workload points did not fall into any one bin, the workload was not classified as constant despite operating within a small range. However, as bin ranges are generally rather large and successively increase in range, with the latter few being several thousand IOPS, this circumstance occurred relatively infrequently.

Random and Seasonal classifications

As idle and constant classifications take precedence over random and seasonal classifications, it is possible to have workloads that are explicitly seasonal but are classified as idle or constant, an example shown in Figure 4-2. As this precedence is intentional, such classifications are not considered incorrect.

The majority of incorrect classifications are consequently seasonal workloads being classified as random or random workloads classified as seasonal. The former can occur due to the workload containing too much noise to be accurately classified as seasonal or having a pattern that is too long to be detected by the seasonality detection algorithm. The latter can occur by chance, when a random workload happens to have points in the right places to be given a seasonality. Due to the consistent nature of workloads, incorrect classifications due to chance circumstances or noise are

Constant Value Prediction Errors				
Workload Type	**Constant**		**Random**	
Error Metric	**Mean MAPE**	**Median MAPE**	**Mean MAPE**	**Median MAPE**
Overall	83.5	5.7	>500	111.7
Day 1	3.5	7.2	>500	110.1
Day 2	12.4	6.7	>500	120.1
Day 3	418.0	5.6	>500	102.8
Day 4	7.8	5.0	>500	104.6
Day 5	7.5	4.7	>500	136.5
Day 6	6.6	4.8	>500	107.9
Day 7	6.3	4.9	>500	126.0

Table 4.1: The percent error of the constant value prediction for constant and random workloads. The error shown is the mean or median of MAPEs across all constant or random workloads on the day specified. Errors greater than 500% are too large to be significant and are thus not explicitly listed.

not stably repeated from day to day. Using this observation, modifications to ELF that address these circumstances will be discussed in Section 5.2.2. Workloads with patterns too long to be caught with the seasonality detection algorithm will be caught at higher granularities, to be discussed in Section 4.4.2.

4.1.2 Forecast

ELF forecasts workloads using two different methods: for idle, constant, and random workloads, a constant value prediction is used. For seasonal workloads, the Holt-Winters model is used for forecasting. Since there are different use cases and considerations for different types of workloads and prediction methods, accuracy evaluation for these two prediction methods were conducted separately.

Constant Value Prediction

In validating the prediction for idle workloads, use cases must be considered. Since the constant value prediction for idle workloads is 0, error metrics do not yield any additional useful information. It is unlikely for predictions of idle workloads to be used in additional analysis as idle workloads are likely to be filtered out of most

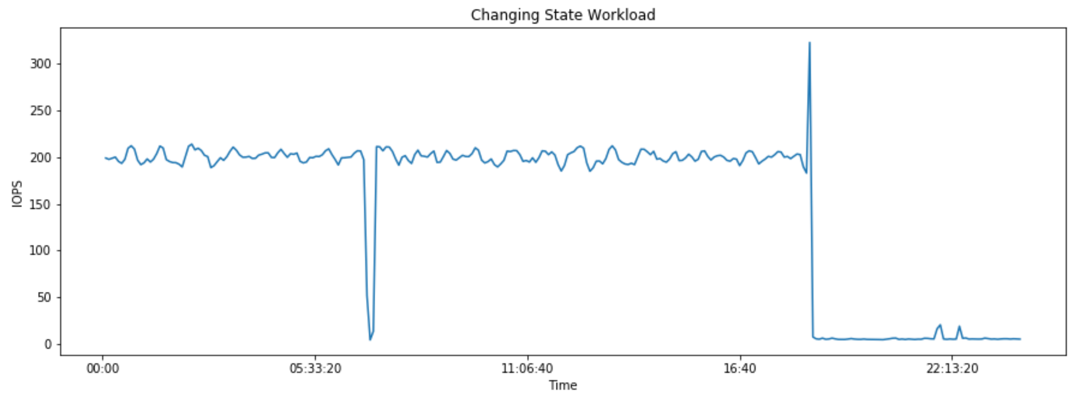


Figure 4-3: A workload that was initially constant but became idle during the day that was predicted. A state change such as this would result in a high error for the constant value prediction.

analyses, so idle workload predictions were left as is.

In evaluating the accuracy for constant and random workload predictions, we used the mean absolute percent error (MAPE) over all points in a prediction. Table 4.1 displays the mean or median of all MAPEs calculated for a certain day or type of workload. These error statistics give the representative MAPE for all workloads of a certain type that were predicted with a constant value prediction. As a single MAPE returns the average percent error for a single workload day, the mean or median of all MAPEs across workloads of the same type displays a metric for how well constant value predictions perform across workloads of that type. In inspecting the error statistics for outliers, we found two circumstances that resulted in unusually high error. The first was the occurrence of abnormalities such as random spikes or dips that cannot be predicted. The second was state or level changes in the workload, as shown in Figure 4-3. Due to the nature of 24 hour classification and prediction, if the level of workload activity changes, it will take a day for the classification and prediction to adjust.

To account for high errors due to abnormalities, for each MAPE calculated, the 5th and 95th percentile of errors were filtered before taking the mean of the absolute errors. The mean and median of all such calculated MAPEs are the values shown in Table 4.1. In considering prediction errors without errors due to state changes, we can examine the median of all MAPEs. However, since ability to adapt to level changes is a desired quality in time series forecasters, the mean of all MAPEs is still a pertinent evaluation metric. The day-to-day distribution in Table 4.1 of the mean of MAPEs for constant workloads more prominently displays days in which workloads undergo an activity level change, indicated by abnormally high percent errors.

After considering these adjustments and circumstances, the error for constant workload predictions is appropriately acceptable in providing an accurate representation of activity. However, the error for random workload predictions is consistently high, even when examining the median of MAPEs. This is to be expected, since we are predicting workloads characterized with a wide range of activity, often several thousand IOPS, with a constant value. Because random workloads are difficult to predict even with an ML model, as shown in Table 4.2, random workloads will always have to be overprovisioned or dealt with conservatively in certain use cases.

ML Model forecast

Two different types of validation metrics were used when evaluating prediction results from the ML model. The first, the RMSE, gives a representation of how well the forecasted result fits the actual data. In order to display a generically comparable metric, we divided the RMSE of each prediction by the total range of activity of the validation data. Table 4.2 shows the average percent error of this metric over all the given workloads for a specific day. To reduce the effect of outlier noise, the mean or interquartile range could also be used to normalize the RMSE. Since the appearance of outlier noise did not occur frequently in seasonal workloads, we used the total range when calculating the error metric. The second error metric is an up/down classification accuracy metric, which provides a representation of how well the forecast predicts peaks and valleys in the data. This metric is useful in cases where on/off classifications are particularly significant, such as provisioning and scheduling. The accuracy of this metric is given by the percentage of points in the forecast that are above or below the median value at the same time the equivalent point in the validation data is above or below the median. A forecast with perfect classification

ML Model forecast (Holt-Winters)						
Validation Metric	**RMSE / Range Percent Error**			**Up/Down Classification Percent Accuracy**		
Workload Types Forecasted	**Seasonal**	**Random and Seasonal**	**All**	**Seasonal**	**Random and Seasonal**	**All**
Total Average	18.4	264.9	59.4	79.8	67.4	74.3
Day 1 Average	22.7	968.0	48.5	74.6	66.0	73.9
Day 2 Average	15.6	79.3	141.2	80.8	67.4	75.8
Day 3 Average	15.1	95.4	35.5	80.6	66.2	74.9
Day 4 Average	21.7	53.9	37.5	82.8	66.2	73.6
Day 5 Average	15.8	28.6	48.9	83.0	68.0	73.8
Day 6 Average	13.9	26.5	48.1	84.7	71.4	74.0
Day 7 Average	15.0	26.2	19.8	87.0	70.0	73.4

Table 4.2: Validation metrics of workloads forecasted by the Holt-Winters ML model. The RMSE value represents the mean percent error over all workloads of that type on the day specified, where the error is the RMSE divided by the total range of activity that day for that workload. The up/down classification is the percent accuracy of when the predicted forecast is above or below the median in comparison to the actual. A 100% up/down classification rate is perfect accuracy.

ARIMA vs Holt-Winters		
ML Model	**RMSE / Range**	**Up/Down Classification**
ARIMA	25.1	75.9
Holt-Winters	18.4	79.8

Table 4.3: The average RMSE and up/down validation metrics as described in Table 4.2 of the prediction given by the ARIMA and Holt-Winters model over all seasonal workloads classified in the 7 day period.

accuracy will have a percentage of 100%.

To provide further baselines of validation for classification, two more experiments were run, where additional non-seasonal workload types were passed to the ML model to predict. To account for cases where seasonal workloads were incorrectly classified as random and to demonstrate how accurately the ML model could forecast random workloads, a conservative algorithm was run where both random and seasonal workloads were passed to the ML model. For a generic algorithmic baseline, another experiment passed all workloads to the model to see how ELF would perform in comparison. These baseline results are indicated in Table 4.2. For a validation baseline for the Holt-Winters model itself, the ARIMA model was used to predict all seasonal workloads, the results of which are shown in Table 4.3.

First, while examining how well Holt-Winters forecasted only seasonal workloads, as is originally designed in ELF, we found two types of seasonal workloads, each resulting in different levels of error. Strongly seasonal workloads similar to those shown in Figure B-1 usually possessed around 1-5% RMSE error. Weakly seasonal workloads similar to those shown in Figure B-2 were usually predicted with about 20-30% error, resulting in averages that hovered around 15-20% (see Table 4.2). In comparison, the ARIMA model baseline performed with 25% error (see Table 4.3), indicating a stronger performance from the Holt-Winters model.

In comparison to solely forecasting seasonal workloads, the prediction errors for random and seasonal workloads were unsurprisingly higher. Despite the model generally performing better on random workloads than the constant value prediction, the comparison becomes arbitrary as the average error is above 250% even when averaged with errors from seasonal workloads, an error too great for the prediction to be used reliably. This result reinforces the assumption that random workloads are too unpredictable for an ML model and are better off predicted with a constant value to save computation. The cases where this conservative algorithm would perform better than ELF is in cases of seasonal workloads being incorrectly classified as random. For this circumstance, we look to modifications to improve the classification algorithm instead, as will be discussed in Section 5.2.2.

Finally, in studying the results from the general algorithm baseline where all workloads were passed to the model, errors are also higher in comparison to just forecasting seasonal workloads. Though an element of the increased error is due to the high error from predicting random workloads, the general error metric of RMSE divided by range is ill-suited to represent prediction errors for idle and constant workloads because their ranges are so narrow. Thus, in properly summarizing the accuracy of idle and constant predictions, other validation metrics should be considered. In examining the up/down classification accuracy between only seasonal workloads, seasonal and random workloads, and all workloads, the comparative accuracy indicates that idle and constant workloads forecasted by an ML model have around similar accuracy to just seasonal workloads. Since idle and constant workloads should be trivial for an ML model to predict, we can conclude that seasonal predictions are well-predicted by Holt-Winters.

4.1.3 Accuracy Overview

To summarize the accuracy validation findings of ELF, first, we found that about 89% of all classifications were idle, 2% were constant, 7% were random, and 2% were seasonal. While idle and constant classifications were relatively accurate, additional modifications can be made to ELF to reduce incorrect classifications between random and seasonal workloads. In terms of the prediction of these workloads, constant value predictions worked well and accurately for idle and constant workloads. For random workloads, neither constant value predictions nor an ML model produce predictions accurate enough to use reliably. Thus, random workloads must always be dealt with conservatively and computation time is saved by using a constant value prediction to report representative activity. When predicting using a ML model, forecasting solely on seasonal workloads provides the best performance. In addition, the performance of the Holt-Winters model on seasonal workloads is satisfactorily accurate, performing better than the baseline ARIMA model and performing similarly to forecasts on trivial activity.

Data from online calculations stored in cache		
Data structure	**Size per workload (bytes)**	**Total size (MB)**
Histogram	144	2.88
Bin sums	144	2.88
Total	288	5.76

Table 4.4: Size of data stored in cache at any one point in time. Total size was estimated using 20,000 workloads.

4.2 Storage

For storage validation, ELF was evaluated on the size of data needed for online computations as well as offline computations. In Chapter 2, it was mentioned that we were limited to 30 MB of storage in the cache. While there was no such limit for offline computations, minimizing the amount of data needed for those calculations would greatly affect the speed of overall computation as pulling data from the database is a costly operation.

4.2.1 Online Computations

The only online computation that ELF executes is the creation and update of histograms for each workload. In order to have an accurate constant value prediction for constant workloads, a sum count of values for each bin is also kept. Since the histograms are a fixed size of 10 bins, this results in a total of 20 values stored for each workload. Table 4.4 lists the size of data kept for each workload and for all 20,000 workloads. Cumulatively, the data stored in cache is less than 6 MB, far below our 30 MB limit.

4.2.2 Offline Computations

The relevant storage evaluation for offline computations is the amount of data needed to be pulled from the database for each operation. In ELF, for each non-idle and non-constant workload, a day of data is pulled for the seasonality detection algorithm. For each classified seasonal workload, three days of data are pulled for the ML model. In

Data pulled from the database in offline calculations			
Operation	**Size per workload (bytes)**	**ELF total size (MB)**	**Baseline total size (MB)**
Seasonality detection algorithm	9368	18.7	0
ML model	208,112	83.2	4162.2
Total	217,480	101.9	4162.2

Table 4.5: Amount of data pulled from the database in one day of operation. Total size was estimated using 20,000 workloads and using the percentages that 10% of workloads are non idle or constant (and run through seasonality detection) and 2% of workloads are seasonal (and run through the ML mode).

a baseline algorithm without the classification aspect, 3 days of data are pulled for every workload. Table 4.5 shows the size of data that ELF or the baseline algorithm would pull from the database in one day of operation. Total size statistics for ELF are estimated using the percentages that 10% of workloads are non-idle and non-constant and 2% of workloads are seasonal. In total, it is shown that ELF pulls approximately 40 times less data than the baseline.

4.3 Timing

In validating ELF on the basis of timing, it was compared against two baselines. The first baseline validated the classification aspect of ELF's design and the second baseline validated the machine learning model that was chosen. The classification baseline algorithm is the same as ELF but differs in that it does not classify workloads and instead runs all workloads through the ML model. The ML model baseline algorithm is the same as ELF except for the ML model that was run. Rather than the Holt-Winters model, seasonal workloads were forecasted with ARIMA.

Table 4.6 details offline process time (not real time) spent doing individual module operations cumulatively over the 7 days. Note that while the histogram module is additional computation that ELF executes that the baseline does not, as all computation for the histograms is online, the computation time of the system as a whole does

Computation Time of ELF and Baselines (in seconds)			
Operation	**ELF**	**Classification Baseline**	**ML Model Baseline**
Idle/Constant classification	1.38	0	-
Pulling 1 day of data for seasonality detection	92.35	0	-
Random/Seasonal classification	16.24	0	-
Constant value prediction	2.02	0	-
Pulling 3 days of data for ML model	25.06	1824.76	-
ML model forecast	1871.64	182,680.11	19,168.66
Total	2008.69	184,504.87	-

Table 4.6: Cumulative offline process time spent running each listed module operation over a period of 7 days. The classification baseline algorithm forecasted all 20,000 workloads directly with the Holt-Winters model without any classification of the workloads. The ML model baseline ran the ARIMA model over all workloads that the Holt-Winters model was run on in ELF (seasonal workloads).

not increase due to the histogram module. As such, the computation time for the histogram module has been excluded from the comparison. In addition, in regards to pulling data from the database, though the validation experiments were not run in the actual system mentioned in Section 2.1, both ELF and the baselines were run in the same simulated environment, which possessed similar environment variables to the aforementioned system. Thus the time taken to pull data from the database is comparable between the algorithms.

As shown, a significant portion of the overall computation time is dedicated to pulling data from the database. With ELF, this is minimized with its classification portion by allowing data to be pulled from fewer workloads. The additional computation that ELF completes due to the classification of workloads is trivial in comparison to the computation saved from neither running the Holt-Winters model over all 20,000 workloads nor pulling all the data needed to do so. In total, ELF runs about 92 times faster than its baseline. In addition, the chosen model Holt-Winters performs about 10 times faster than its ARIMA baseline when run on the exact same data.

4.4 Flexibility

Since ELF was designed to be generic, it was also evaluated on its ability to operate on different types of data. In considering different types of data, we examined the use of ELF with different metrics as well as with data at different time granularities.

4.4.1 Metrics

As a general workload forecasting algorithm, ELF can be utilized for the various use cases that come with forecasting workload data of different metrics such as capacity and latency. However, specific portions of the algorithm need to be tuned to account for metric characteristics. The design decisions described in Chapter 3 that were made specifically for the IOPS metric include histogram bin ranges, classification states, and consequently the values for constant value predictions. In order to customize these settings for different metrics, workload characterization as well as use case analysis is needed for each metric.

For example, in considering latency, use cases could include process scheduling or general usage analysis, in which case it would be useful to detect and predict patterns, similarly to IOPS. Depending on environment, bin ranges could include under 1 millisecond, a few milliseconds, greater than a few milliseconds, etc. Classification states could mirror the IOPS states, such as low latency, high but constant latency, and patterned latency, with constant value prediction values following similarly.

In considering capacity, use cases could include capacity planning, in which it would be useful to know when capacity has reached a certain threshold and at what rate it is increasing. Bin ranges could be certain percentage thresholds, based on workload characterization, and states could include low usage (the IOPS idle equivalent), high usage (a constant equivalent), and increasing usage (seasonal equivalent). Both workloads classified as high usage and increasing usage would require the appropriate additional attention and flags.

Though customizations are required to run ELF with different metrics, the basic functionality and structure remain the same. In order to efficiently predict various workloads for various use cases, workloads are classified into states that allow them to be forecasted and handled in a systematic and economical manner.

4.4.2 Time Granularities

When evaluating the use of ELF on different time granularities, we considered not only its ability to operate with such data but also the different use cases and problem solutions it could fulfill. Throughout the thesis, only data at a 5 minute granularity was discussed. However, the system on which ELF was developed also processed hourly and daily data, which were also tested when designing the algorithm. Though hourly and daily workloads were not studied as deeply as 5 minute workloads, they were used sufficiently to ensure that ELF functioned properly on alternate time granularities.

The use of data at different time granularities provides solutions to many different needs. For example, using multiple granularities, ELF can detect and model different patterns that occur at different time scales. When utilizing 5 minute data, hourly patterns can be captured and forecasted. With hourly data, daily or weekly patterns could be captured and, similarly, with daily data, monthly or quarterly patterns could be found. In a typical business cycle, since patterns may occur from working hours to nighttime, from weekday to weekend, or along quarterly deadlines, running the model at different granularities could capture multiple patterns in a single workload. Figure 4-4 shows a workload with both hourly and daily patterns.

This feature would also allow workloads with unstable classifications to be more accurately classified. In the case where a particular workload is lightly seasonal or possesses a seasonality right on the border of the seasonality detection algorithm's maximum pattern length limit, its classification may oscillate between random and seasonal from day to day. Using a higher granularity in situations such as these would allow more insight into which classification would be more accurate for that particular workload. In addition, seasonal workloads that have patterns too long for the seasonality detection algorithm would be classified as random at a smaller time granularity, but consequently would be correctly classified as seasonal at a larger

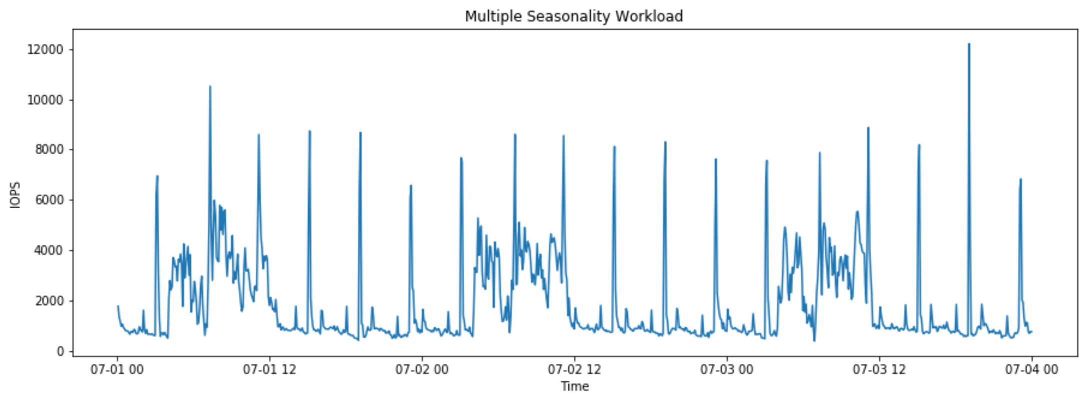


Figure 4-4: The activity of a workload over three days, from July 1st to July 3rd. Note that the workload contains both an hourly pattern of about 4 hours as well as a daily pattern of raised activity around 6am to 12pm.

granularity.

In addition, it was found that different machine learning models would be preferable at larger granularities. Patterns found at an hourly granularity are similar to those found at a 5 minute granularity, making Holt-Winters the ideal model for hourly data as well. However, at a daily granularity, we found that patterns are not as explicit as they are at a 5 minute or hourly granularity. The lightly seasonal workloads that are typically found in daily data are forecasted more accurately with the Facebook Prophet model, which allows for more parameters and customizations. In addition to capturing more difficult patterns, the Prophet model possesses explicit parameters for custom seasonalities such as holidays or weekends, seasonalities that only apply at a daily granularity. Though the Prophet model is a more computationally complex model than Holt-Winters, because forecasts at a daily granularity may not have to be run as often as forecasts at the other two granularities, the effects of this drawback can be minimized.

Chapter 5

Conclusion

5.1 Summary

This thesis presents ELF, an algorithm that addresses the problem of efficient forecasting of storage workloads. It was designed to be scalable to tens of thousands of workloads and flexible to different data metrics and use cases. To achieve this, it utilizes concepts such as online computation, classification of workloads, and specialized prediction methods. Its main approach to efficiency is to reduce computational overhead by executing complex computations on less data. The basic architecture consists of four modules: the histogram module, classification module, seasonality detection module, and prediction module. The histogram module collects compact statistics about each workload that allow 90% of all workloads to be classified as idle or constant with trivial computation. The classification module then utilizes the seasonality detection module to classify the remaining 10% of workloads as random or seasonal. Finally, the prediction module forecasts the 2% of workloads classified as seasonal with a machine learning module and the remaining 98% of workloads with a constant value prediction. With this design, ELF performs 92 times faster and stores 40 times less data than baseline algorithms while still matching their accuracy.

5.2 Discussion

5.2.1 Contribution

In addition to providing an efficient, lightweight, and accurate method to forecasting workloads, our approach provides further benefit with its generic and modularized nature. Its modularization allows for the multi-use of specific modules, such as the histogram module and seasonality detection module, in other use cases and services that do not run the entirety of the algorithm. Its generic nature allows it to be used with many different data metrics, consequently also fulfilling the additional use cases that follow. In summary, ELF not only provides an efficient solution to forecasting storage workloads, but its versatility allows it to be utilized in many different use cases and solutions.

5.2.2 Future Algorithm Modifications

In moving forward with ELF, there are many different modifications that could be made to improve efficiency or accuracy. Under different resource constraints, more of the algorithm could be completed online. If we could store more than 30 MB of data in the cache at a time, the only limiting factor on online computations would thus be a computation time limit of 5 minutes or the data collection granularity. With this change, we would likely be able to run all modules except for the ML model forecast online, thus increasing efficiency by decreasing both offline computations and the amount of data to pull from the database.

To address misclassifications between seasonal and random workloads, we consider both the issue of seasonal workloads misclassified as random and random workloads misclassified as seasonal. For the former, an initial exploration was completed, coined as the "conservative approach," that sent all random and seasonal workloads to the ML model. Though this approach would resolve this particular issue, the costs to accuracy (Table 4.2), computation time (Table A.1), and storage (Table A.2) were too great to consider it as a valid modification. Instead, we conducted further studies to develop a heuristic that could indicate workloads that are more likely to be seasonal rather than random.

For misclassifications between seasonal and random workloads not due to the fact that the seasonal pattern is too long to be detected by the detection algorithm, we assume that all such misclassified workloads are lightly seasonal or contain activity that occasionally resemble seasonal patterns. These workloads are characterized with unstable day-to-day classifications, such as switching between seasonal and random classifications multiple times within a sequence of days. Because most workloads have relatively stable activity, this occurrence indicates a workload containing both random and seasonal characteristics. To properly distinguish seasonal and random workloads out of these types of workloads, we can use two strategies. The first is to examine the workload at a greater time granularity. Oftentimes the workload activity will stabilize and become more distinct once viewed at either hourly or daily granularities. Combining results from ELF running with 5 minute data and from it running with hourly or daily data as described in Section 4.4.2 will allow us more insight into the correct classification of the workload. The second strategy is to utilize a heuristic based on histogram point concentration, error, and classifications of a workload throughout multiple days. In analyzing histogram point concentration, we found that seasonal workloads generally tend to have a higher concentration of points among one or two histogram bins while random workloads tend to have a larger distribution among multiple bins. When predicted with a constant value prediction, seasonal workloads consistently tend to have a smaller error than random workloads. In combining these two observations, we can develop a heuristic that, given workloads that have both seasonal and random characteristics, can more accurately distinguish between workloads that are mainly random and those that are mainly seasonal.

Finally, another adjustment to ELF could be to improve classifications that are based on histogram distributions, mainly constant workload classifications. In the case where a constant workload's activity straddles a bin boundary, it would be classified as random rather than constant. To combat this issue, we may develop a heuristic that examines bin distribution or create a second set of histogram bins with ranges that straddle the first histogram bin ranges. However, before doubling the amount of data we would need to store in cache, proper analysis would have to be conducted to ascertain its need.

5.2.3 Additional Insights

An important aspect of workload analysis and forecasting is abnormality detection. Abnormalities may include random spikes or dips in usage, changes in state or level of activity, and changes in pattern or activity type. Many of these changes would be useful to detect and flag for future inspection or analysis. Though abnormality detection was not a focus for this thesis, due to the nature of ELF and its validation, detecting these changes would be simple. For example, changes in state or level of activity would be indicated by a shift in classification for a particular workload or a change in its constant value prediction. A change in pattern could be indicated by a change in seasonality length as returned from the seasonality detection module. The addition of these features would further expand the versatility and usage of ELF.

Appendix A

Additional Tables

Computation Time of ELF vs Conservative Algorithm (in seconds)		
Operation	**ELF**	**Conservative Algorithm**
Idle/Constant classification	1.38	1.41
Pulling 1 day of data for seasonality detection	92.35	-
Random/Seasonal classification	16.24	-
Constant value prediction	2.02	0.02
Pulling 3 days of data for ML model	25.07	108.57
ML model forecast	1871.64	7326.49
Total	2008.70	7436.49

Table A.1: Cumulative offline process time distribution over a period of 7 days between ELF where only seasonal workloads are sent to the ML model and a conservative version, where both random and seasonal workloads are sent.

Data pulled from the database in offline calculations			
Operation	**Size per workload (bytes)**	**ELF total size (MB)**	**Conservative Algorithm total size (MB)**
Seasonality detection algorithm	9368	18.7	0
ML model	208,112	83.2	374.6
Total	217480	101.9	374.6

Table A.2: Amount of data to be pulled from the database in one day of operation between the original algorithm where only seasonal workloads are sent to the ML model and a conservative version, where both random and seasonal workloads are sent.

Appendix B

Additional Figures

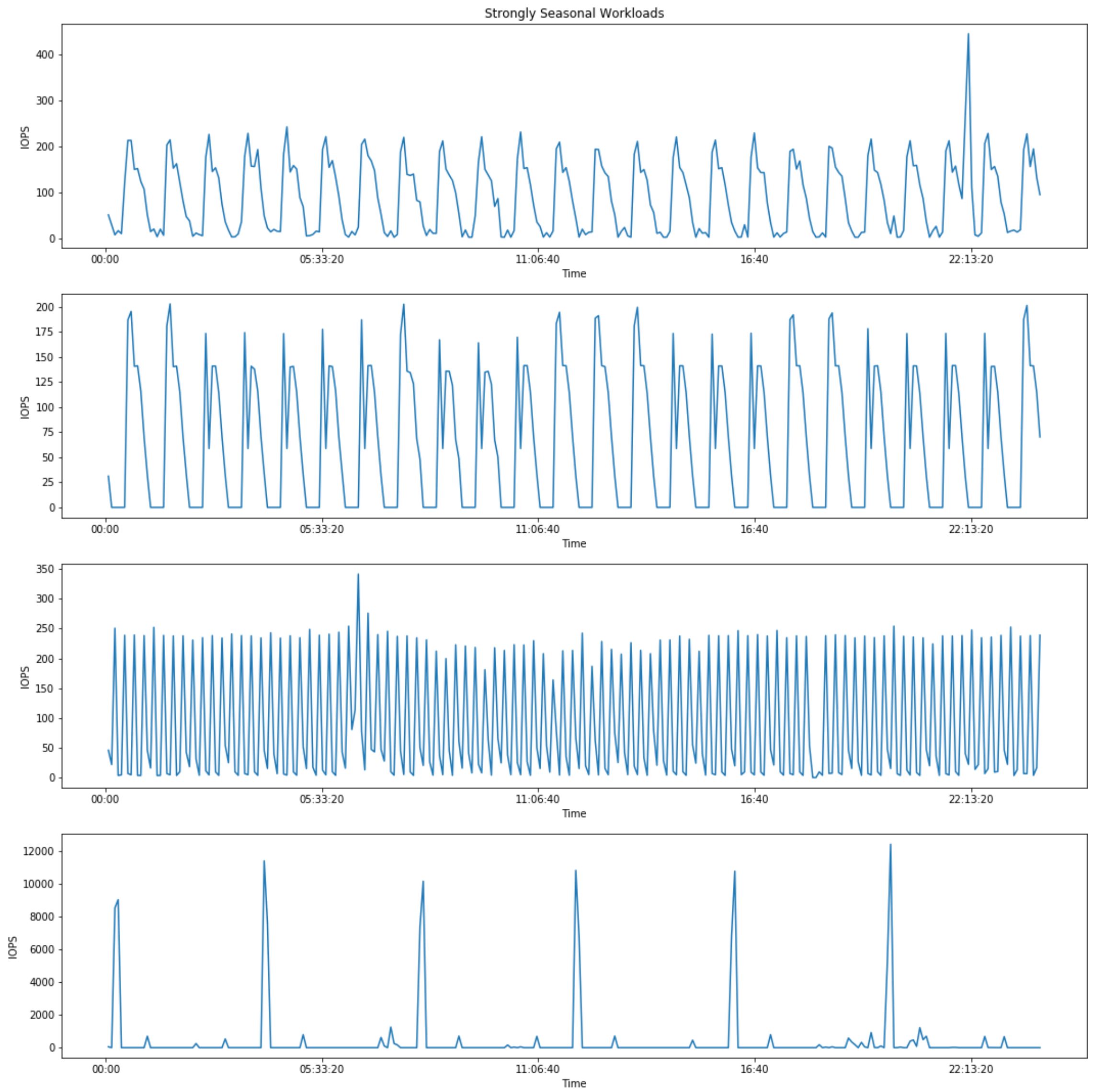


Figure B-1: Examples of strongly or explicitly seasonal workloads. These are characterized by consistent activity with little variation.

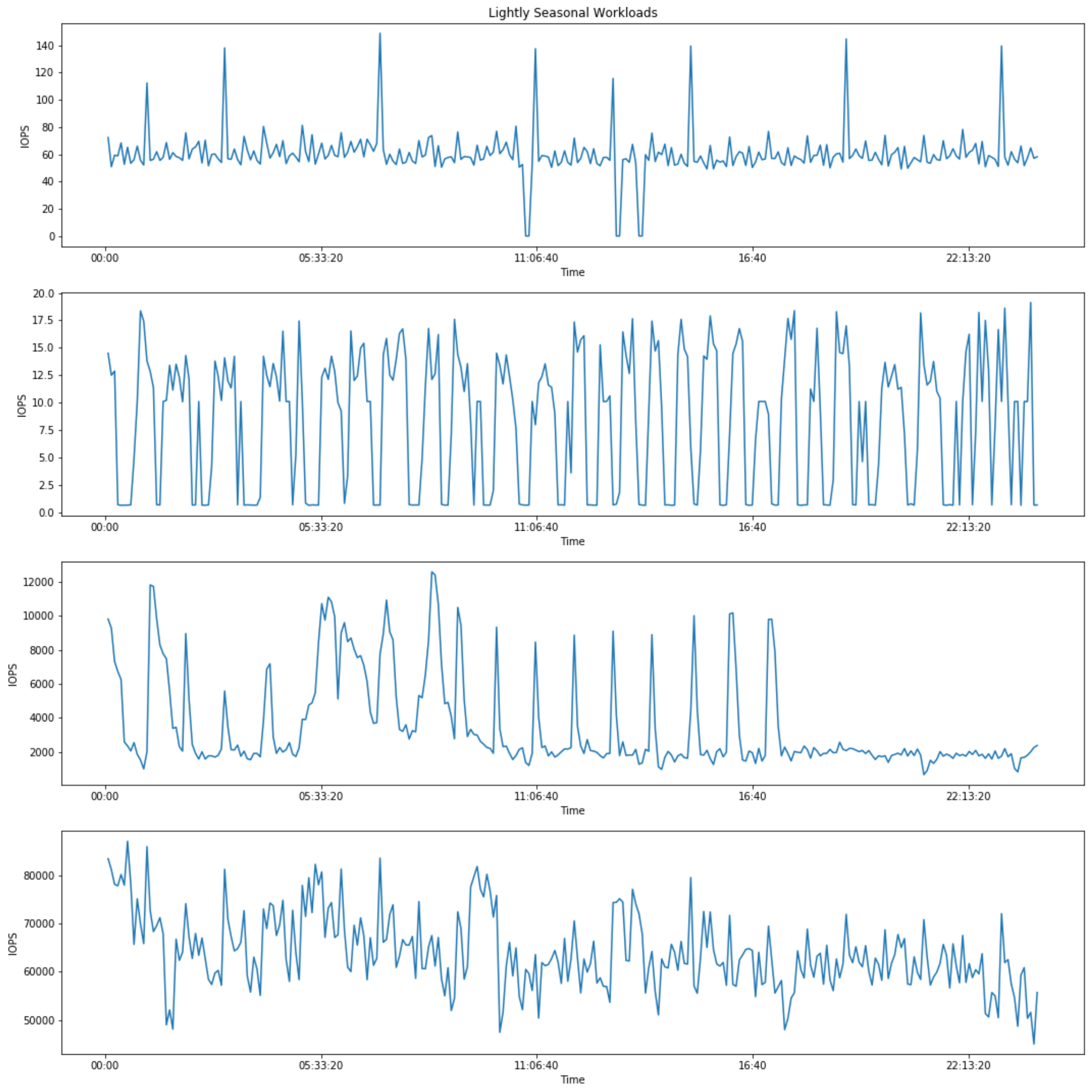


Figure B-2: Examples of lightly or weakly seasonal workloads. The activity of these workloads could range from seasonal with multiple abnormalities, as shown at the top, to workloads with slightly ambiguous or changing patterns.

Bibliography

[1] M. Alshawabkeh, A. Riska, A. Sahin, and M. Awwad. Automated storage tiering using markov chain correlation based clustering. In *2012 11th International Conference on Machine Learning and Applications*, volume 1, pages 392–397, 2012.

[2] Arvind Arasu and Gurmeet Manku. Approximate counts and quantiles over sliding windows. volume 23, pages 286–296, 01 2004.

[3] Jayanta Basak and Madhumita Bharde. Dynamic provisioning of storage workloads. In *29th Large Installation System Administration Conference (LISA15)*, pages 13–24, Washington, D.C., November 2015. USENIX Association.

[4] Luca Becchetti and Elias Koutsoupias. Competitive analysis of aggregate max in windowed streaming. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming*, pages 156–170, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[5] Chiranjeeb Buragohain and Subhash Suri. Quantiles on streams. 01 2009.

[6] Chethan, M. Pushpalatha, and Dr. Ramesh Boraiah. A survey on analysis and classification of workload in cloud. 2016.

[7] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Effective computation of biased quantiles over data streams. In *21st International Conference on Data Engineering (ICDE'05)*, pages 20–31, 2005.

[8] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows (extended abstract), 2002.

[9] Steven Engelhardt. Calculating percentiles on streaming data, 3 2018.

[10] UK Centre for the Measurement of Government Activity. From holt-winters to arima modelling: Measuring the impact on forecasting errors for components of quarterly estimates of public service output, 2008.

[11] Phillip B. Gibbons and Srikanta Tirthapura. Distributed streams algorithms for sliding windows. In *Proceedings of the Fourteenth Annual ACM Symposium on*

Parallel Algorithms and Architectures, SPAA '02, page 63–72, New York, NY, USA, 2002. Association for Computing Machinery.

[12] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *In SIGMOD*, pages 58–66, 2001.

[13] Ajay Gulati, Chethan Kumar, Irfan Ahmad, and Karan Kumar. Basil: Automated io load balancing across storage devices. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, page 13, USA, 2010. USENIX Association.

[14] Nikolas Herbst. *Workload Classification and Forecasting.* PhD thesis, 01 2012.

[15] Antony S. Higginson, Mihaela Dediu, Octavian Arsene, Norman W. Paton, and Suzanne M. Embury. Database workload capacity planning using time series analysis and machine learning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 769–783, New York, NY, USA, 2020. Association for Computing Machinery.

[16] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey. Empirical evaluation of workload forecasting techniques for predictive cloud resource scaling. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 1–10, 2016.

[17] Youngjae Kim and Raghul Gunasekaran. Understanding i/o workload characteristics of a peta-scale storage system. *Journal of Supercomputing*, 71(3), 11 2014.

[18] Qiang Ma, S. Muthukrishnan, and Mark Sandler. Frugal streaming for estimating quantiles: One (or two) memory suffices. *CoRR*, abs/1407.1121, 2014.

[19] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. Sketching distributed sliding-window data streams. *The VLDB Journal*, 24:345–368, 06 2015.

[20] Odysseas Papapetrou, Minos N. Garofalakis, and Antonios Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *CoRR*, abs/1207.0139, 2012.

[21] Alma Riska. Disk drive level workload characterization. In *In USENIX Annual Conference*, 2006.

[22] Pedro Pereira Rodrigues and João Gama. Online prediction of streaming sensor data.

[23] Joni Salonen. Welford's method for computing variance, 2013.

[24] B. Seo, S. Kang, J. Choi, J. Cha, Y. Won, and S. Yoon. Io workload characterization revisited: A data-mining approach. *IEEE Transactions on Computers*, 63(12):3026–3038, 2014.

[25] C. Vazquez, R. Krishnan, and E. John. Time series forecasting of cloud data center workloads for dynamic resource provisioning. 6:87–110, 09 2015.

[26] Xiaozhe Wang, Kate Smith-Miles, and Rob Hyndman. Characteristic-based clustering for time series data. *Data Min. Knowl. Discov.*, 13:335–364, 09 2006.

[27] Xiaozhe Wang, Kate Smith-Miles, and Rob Hyndman. Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series. *Neurocomputing*, 72(10):2581–2594, 2009. Lattice Computing and Natural Computing (JCIS 2007) / Neural Networks in Intelligent Systems Designn (ISDA 2007).

[28] Alan Wolf, Jack B. Swift, Harry L. Swinney, and John A. Vastano. Determining lyapunov exponents from a time series. *Physica D: Nonlinear Phenomena*, 16(3):285–317, 1985.

[29] J. Xue, F. Yan, R. Birke, L. Y. Chen, T. Scherer, and E. Smirni. Practise: Robust prediction of data center time series. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 126–134, 2015.

[30] Jing Xue. *Workload prediction for efficient performance isolation and system reliability.* PhD thesis, 2017.

[31] Ariane Ziehn, Marcela Charfuelan, Holmer Hemsen, and Volker Markl. Time series similarity search for streaming data in distributed systems. In *EDBT/ICDT Workshops*, 2019.