AN INFLUXDATA CASE STUDY

How Veritas Technologies Uses InfluxDB to Enable Time Series Forecasting at Scale

Marcello Tomasini
Senior Data Scientist, Veritas Technologies

MAY 2019

Company in brief

Veritas Technologies empowers businesses of all sizes to discover the truth in information — their most important digital asset. Using the Veritas platform, customers can accelerate their digital transformation and solve pressing IT and business challenges including multi-cloud data management, data protection, storage optimization, compliance readiness and workload portability — with no cloud vendor lock-in. Ninety-seven percent of Fortune 100 and eighty-six percent of Fortune 500 companies rely on Veritas today to reveal data insights that drive competitive advantage. Veritas has been awarded as a leader in Gartner's Magic Quadrant for data center backup and recovery for the past 15 years. It offers a complete family of data protection and long-term retention (LTR) appliances that ensure data availability, reduce costs and complexity while increasing operational efficiency.

Case overview

Veritas needed to automate storage forecasting in its SaaS platform, Veritas Predictive Insights. This platform uses artificial intelligence (AI) and machine learning (ML) to deliver predictive support services for Veritas appliance customers. Previously, Veritas measured problems through their auto-support capabilities. They had years of Veritas AutoSupport information and hundreds of millions of telemetry data points from more than 10,000 Veritas appliances. But they didn't have any analytics for forecasting to enable preventing problems from happening. Visibility was a rear-view mirror.

The availability of a vast amount of time series data (collected for use internally from Veritas' Auto Support capabilities) enabled forecasting for a multitude of use cases from application performance optimization to workload anomaly detection. Yet the challenge was to automate a historically manual process handcrafted for the analysis of a single data series of just tens of data points to large-scale processing of thousands of time series and millions of data points.

Veritas chose InfluxDB to implement a solution to tackle the issues of time series forecasting at scale, including continuous accuracy evaluation and algorithm hyperparameters optimization. Veritas uses InfluxDB for their storage forecasting implementation of data in Veritas Predictive Insights which is capable of training, evaluating and forecasting over 70,000 time series daily.

> *“We need to run the forecast in an automated way. And when we try running in an automated manner, there are a lot of issues because we have more than 10,000 appliances. For each appliance, we are forecasting for each type of storage partition.”*
>
> ***Marcello Tomasini****, senior data scientist*

The business problem

Originally a division of Symantec from 2004 to 2014, Veritas Technologies is a leader in multi-cloud data management with a 360-degree approach to data management — from handling data cloud to on-premise and hybrid solutions. Veritas is most known for two products: software-defined storage InfoScale (a distributed file system) and NetBackup (a leading backup and recovery software for enterprises, commonly used for backing up data centers).

Veritas has more than 10,000 NetBackup appliances deployed in the wild actively using Auto Support and daily reporting, internally, several types of telemetry data.

NetBackup Appliances

Veritas implements forecasts for several use cases. The most important use case is storage forecasting, which runs in Veritas Predictive Insights to track storage consumption of NetBackup appliances and reduce downtime. If the appliance runs out of storage, then the backup fails. A backup fail means that at that point, if any type of event happens in the infrastructure of a company, there is a risk of data loss.

Veritas needed to proactively reduce downtime with NetBackup Appliances in order to lower risk and save cost for its customers, so they built Veritas Predictive Insights , a SaaS platform that uses artificial intelligence (AI) and machine learning (ML) to deliver predictive support services for both new and existing Veritas appliance customers.

Veritas Predictive Insights is built on years of Veritas AutoSupport information and hundreds of millions of telemetry data points from over 10,000 Veritas appliances. Veritas had been using this data before internally, for their support engineers, when they would take a call about a particular problem. The customer could also access this data by logging into their support account. But there were no predictive capabilities from it. They didn't have the engine.

The technical problem

Veritas' more than 10,000 NetBackup appliances, which actively use Auto Support, daily report several types of telemetry data to Veritas:

- The type of data of most interest to Veritas is storage data, which reports **1 point every 15 minutes** (96 points/day) per storage partition type in the appliance (MSDP, Log, etc)
- Veritas has **2+ years of this historical data** (0.5M+ points/appliance → ~ 5B data points).

Veritas wanted to utilize storage usage forecasts for:

- **Resource planning** (predicting and allocating storage to prevent downtime and data loss)
- **Detection of workload anomalies** (such as sudden spikes in the storage used for backups)
- **Identifying possible data unavailability or SLA violations** (that could be caused by running out of storage)
- **Capitalizing on sales opportunities** (visibility into whether a customer is running out of capacity and then selling additional capacity based on their predicted needs)

However, once they built the hardware setup in their ML platform, they needed to automate storage forecasting. This presented a lot of challenges because they had more than 10,000 appliances. For each appliance, they were forecasting for each type of storage partition. It was impossible to run the forecast manually for this massive data volume, so automation was necessary.

Veritas embarked on an automation journey that began with the challenges of time series forecasting at scale and that ended with overcoming the challenges of automating storage forecasting.

Technical journey

Forecasting is the process of making predictions of the future based on present and past data — it does not mean predicting the future. We make predictions with a clear understanding that they are not exact or perfect but rather error-prone. Though nobody/nothing can *exactly* predict the future, what makes forecasts useful is when that error is small for the type of use case being addressed.

> *"Essentially, all models are wrong,...but some are useful."*
>
> ***George Box**, one of the most influential statisticians of the 20th century and a pioneer in the areas of quality control, time series analysis, design of experiments and Bayesian inference*

The predictability of an event or a quantity depends on several factors:

- how well we understand the factors that contribute to it;
- how much data is available;
- whether the forecast can affect what we are trying to manage.

What can be forecasted?

Forecasting requires an understanding of what we want to forecast. Along with that, forecasting requires data. With the advent of IoT and modern time series databases, it's a lot easier to work with high volumes of time series. Since there is a lot more data, forecasting is possible for a lot of use cases.

The key assumption behind forecasting is that the way in which the environment is changing will continue into the future — that what we are trying to forecast is based on the past history. Since we cannot see the future, it means that we **assume that whatever happened in the past is a good indicator of what will happen in the future**. If that is not true, we cannot forecast.

Explanatory Model vs. Time Series Model

There are several types of models for forecasting:

- *Explanatory models (ML-based models that use multiple variables called predictors)*

$$y_{t+h} = f(\mathrm{VAR}_1, \mathrm{VAR}_2, \ldots, \mathrm{VAR}_n, error)$$

- *Time series models*

$$y_{t+h} = f(y_t, y_{t-1}, \ldots, y_{t-n}, error)$$

Time series models have been the most popular because they don't require knowing all the predictor variables or extracting them to train the model. Instead, time series models just look at the past values of the quantity that we want to forecast, and they forecast the future. Also, a simple time series model often performs as well as a more complex machine-learning-based model.

Two other important concepts fundamental to forecasting are additive seasonality and multiplicative seasonality. The difference is just the spread of the seasonality:

- In an additive model, seasonality stays constant.
- In a multiplicative, seasonality increases over time.

Trend and Seasonality

Additive Decomposition

$$y_t = S_t + T_t + R_t$$

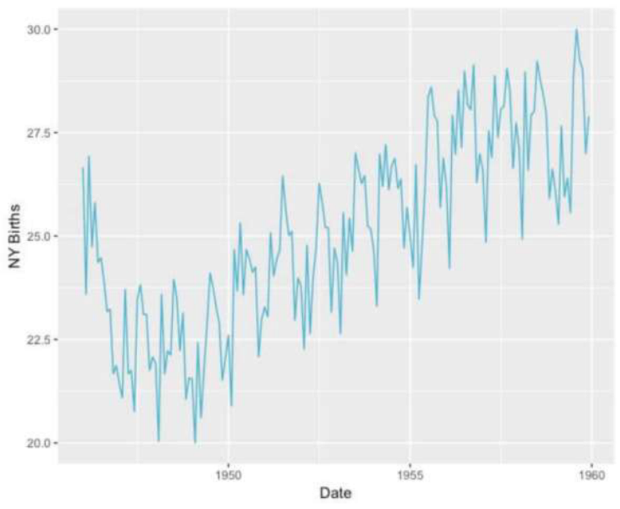
Multiplicative Decomposition

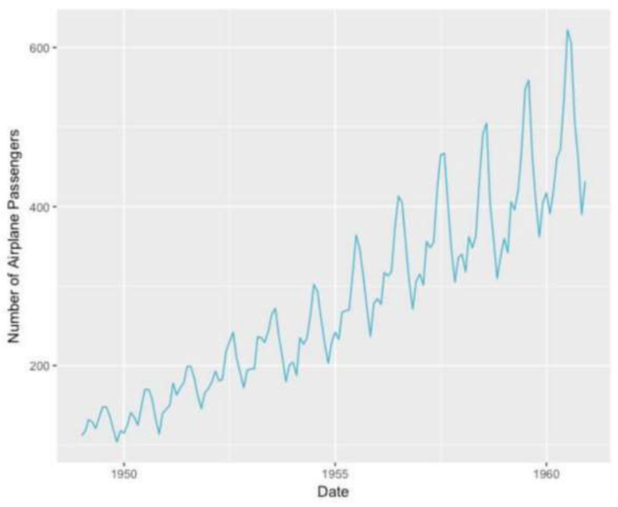
$$y_t = S_t \times T_t \times R_t$$

In the above formulas: $\mathbf{y_t}$ is the data
$\mathbf{S_t}$ is the seasonal component
$\mathbf{T_t}$ is the trend-cycle component
$\mathbf{R_t}$ is the remainder component, all at period t

Depending on the type of data being forecast, it's important to remember the distinction between additive and multiplicative seasonality.

Additive vs. Multiplicative Seasonality



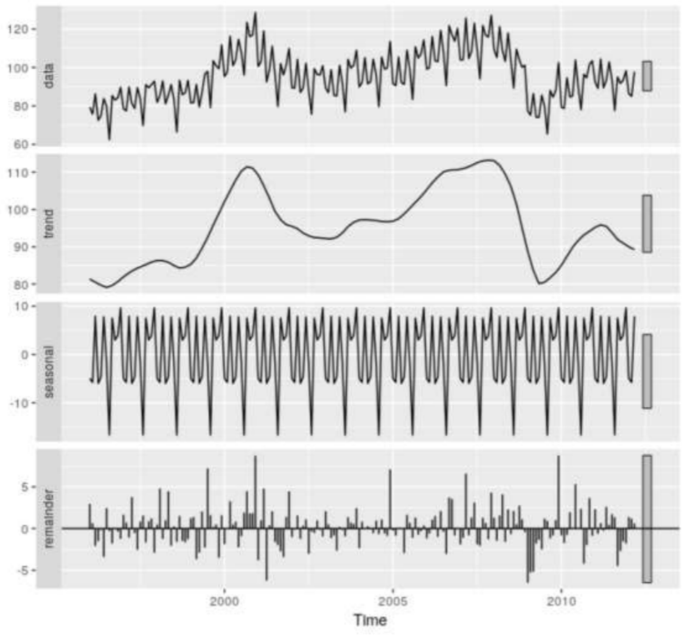


A common pattern to deal with time series is to:

1. Find the trend in the original data and then remove the trend from it
2. Find the seasonality (recurrent pattern) in the data and then remove the seasonality from it
3. Assume that the remainder has a mean of zero and no auto-correlation

If the mean in the remainder is not zero or there is auto-correlation, it means that we didn't do a very good job of extracting the trend and seasonality.

Trend and Seasonality Decomposition



The solution

> *“All this data comes in as time series. So the InfluxDB time series database is being used by multiple algorithms. For that reason, it has to be able to serve a heavy workload. And at the same time, it is used to serve the data back to the UI quickly.”*

Why InfluxDB?

Veritas chose InfluxDB for use in three use cases:

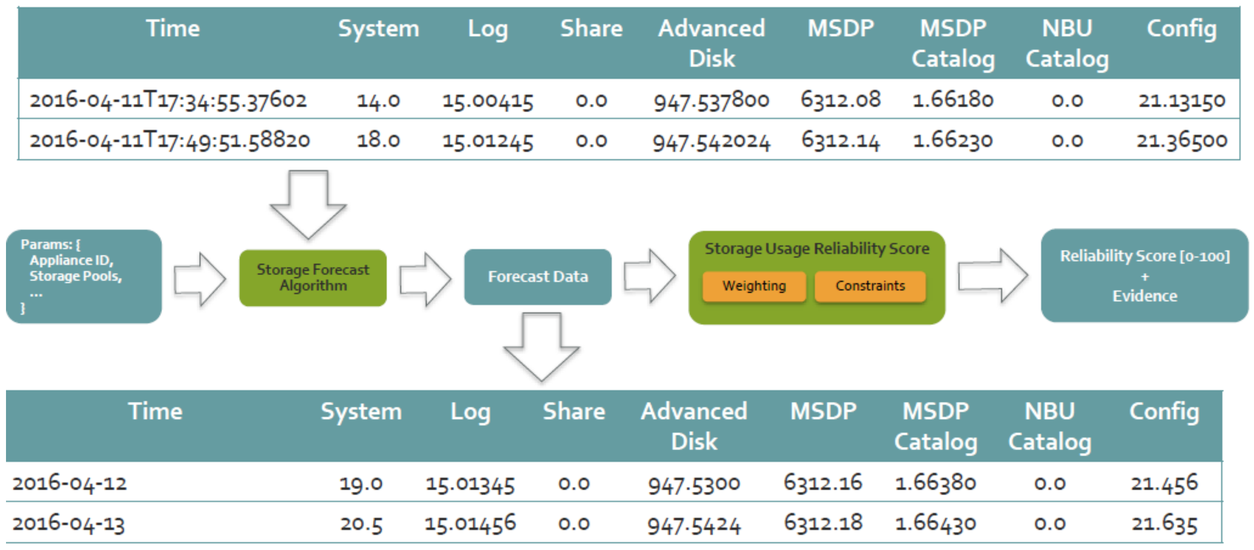
1. **Storage:** storing several types of time series data (storage, cpu usage, disk iops, network iops data)
2. **Use by multiple algorithms:** storage, forecasting, workload anomaly detection, etc.
3. **Serving REST API:** Serve the data back to the UI (the queries for the UI that displays the forecast go directly to InfluxDB, which in itself provides support for REST API)

Veritas also chose InfluxDB because it is purpose-built for time series data. This made it easier to work with time series data than other types of databases. Other InfluxDB features that Veritas liked include:

- Ability to naturally deal with `GROUP BY time()` and its simple query language
- Performance (ingest and storage space)
- Python DataFrameClient: Veritas found the DataFrameClient, which is part of the Python library to interface with InfluxDB, super-useful because by just querying InfluxDB, it will return a Pandas dataframe. This makes most of the data science work trivial because the data is already laid out in your dataframe.

For each partition type, Veritas collects the amount of storage used in bytes. Forecast data is also consumed by other algorithms.

Time-Stamped Data for Each Partition Type



Time	System	Log	Share	Advanced Disk	MSDP	MSDP Catalog	NBU Catalog	Config
2016-04-11T17:34:55.37602	14.0	15.00415	0.0	947.537800	6312.08	1.66180	0.0	21.13150
2016-04-11T17:49:51.58820	18.0	15.01245	0.0	947.542024	6312.14	1.66230	0.0	21.36500

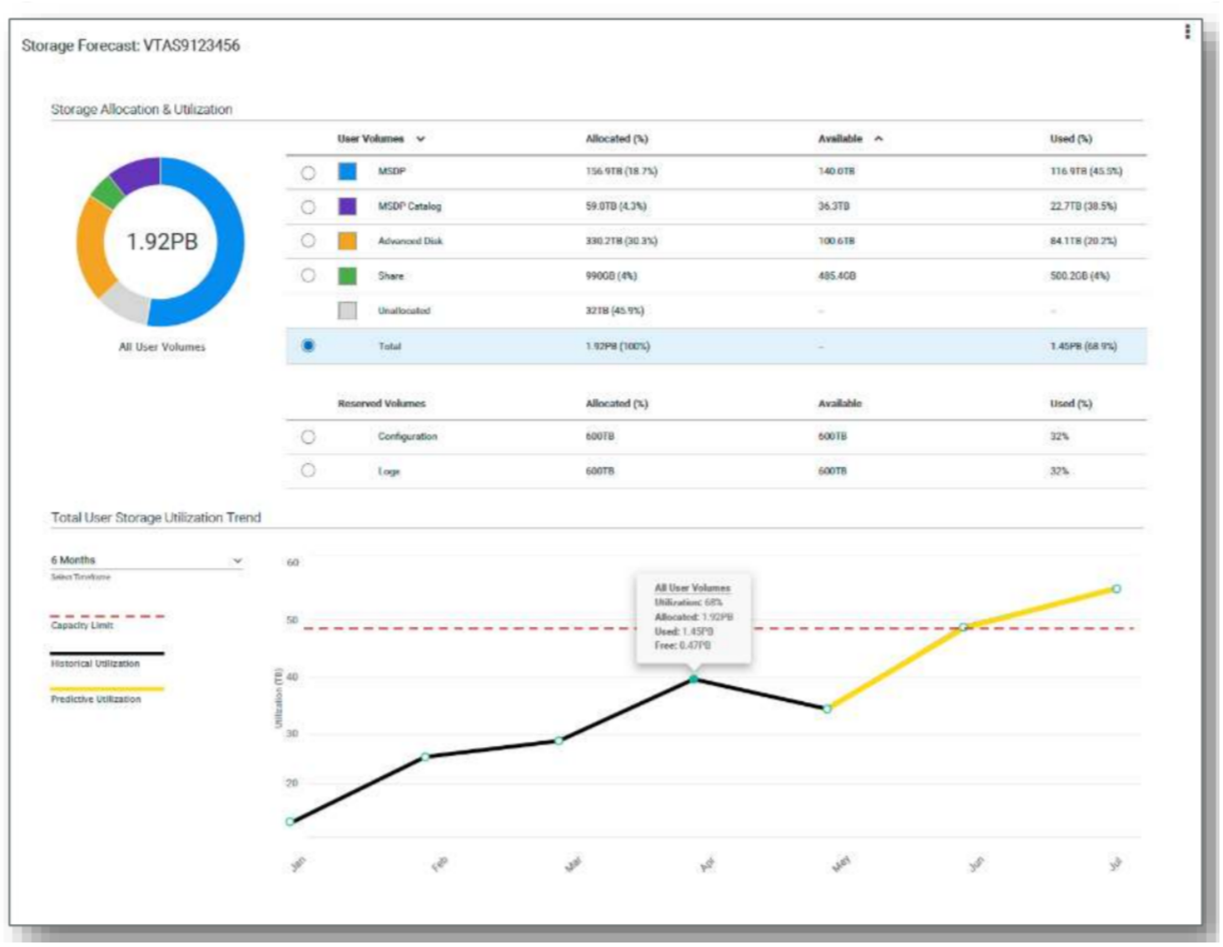
Params: [
Appliance ID,
Storage Pools,
...
]
Storage Forecast Algorithm
Forecast Data
Storage Usage Reliability Score
Weighting
Constraints
Reliability Score [0-100]
+
Evidence

Time	System	Log	Share	Advanced Disk	MSDP	MSDP Catalog	NBU Catalog	Config
2016-04-12	19.0	15.01345	0.0	947.5300	6312.16	1.66380	0.0	21.456
2016-04-13	20.5	15.01456	0.0	947.5424	6312.18	1.66430	0.0	21.635

The telemetry data generates a System Reliability Score [SRS] for each appliance, that is visible in a dashboard for Veritas appliances services personnel, and will also be accessible to customers at a future time. The SRS is what leads support and customer teams to take proactive actions as identified by the analytics. This could include a notification to install a patch to dispatching service personnel or making prescribed, on-site services.

Veritas also has a currently-internal UI (shown below) that charts the storage consumption and forecast for each partition.

Storage Usage Forecast at Veritas Predictive Insights



Storage Forecast: VTAS9123456

Storage Allocation & Utilization

		User Volumes	Allocated (%)	Available	Used (%)
○	🟦	MSDP	156.9TB (18.7%)	140.0TB	116.9TB (45.5%)
○	🟪	MSDP Catalog	59.0TB (4.3%)	36.3TB	22.7TB (38.5%)
○	🟧	Advanced Disk	330.2TB (30.3%)	100.6TB	84.1TB (20.2%)
○	🟩	Share	990GB (4%)	485.4GB	500.2GB (4%)
	⬜	Unallocated	32TB (45.9%)	-	-
●		Total	1.92PB (100%)	-	1.45PB (68.9%)

	Reserved Volumes	Allocated (%)	Available	Used (%)
○	Configuration	600TB	600TB	32%
○	Logs	600TB	600TB	32%

Total User Storage Utilization Trend

All User Volumes
Utilization: 68%
Allocated: 1.92PB
Used: 1.45PB
Free: 0.47PB

This forecasting process and other machine learning algorithms run on Veritas' custom ML platform.

For building its ML engine, Veritas has taken two years of its auto-support data and its hundreds of millions of data points, combined with the real-world experience from support engineers, to add new proactive support services including:

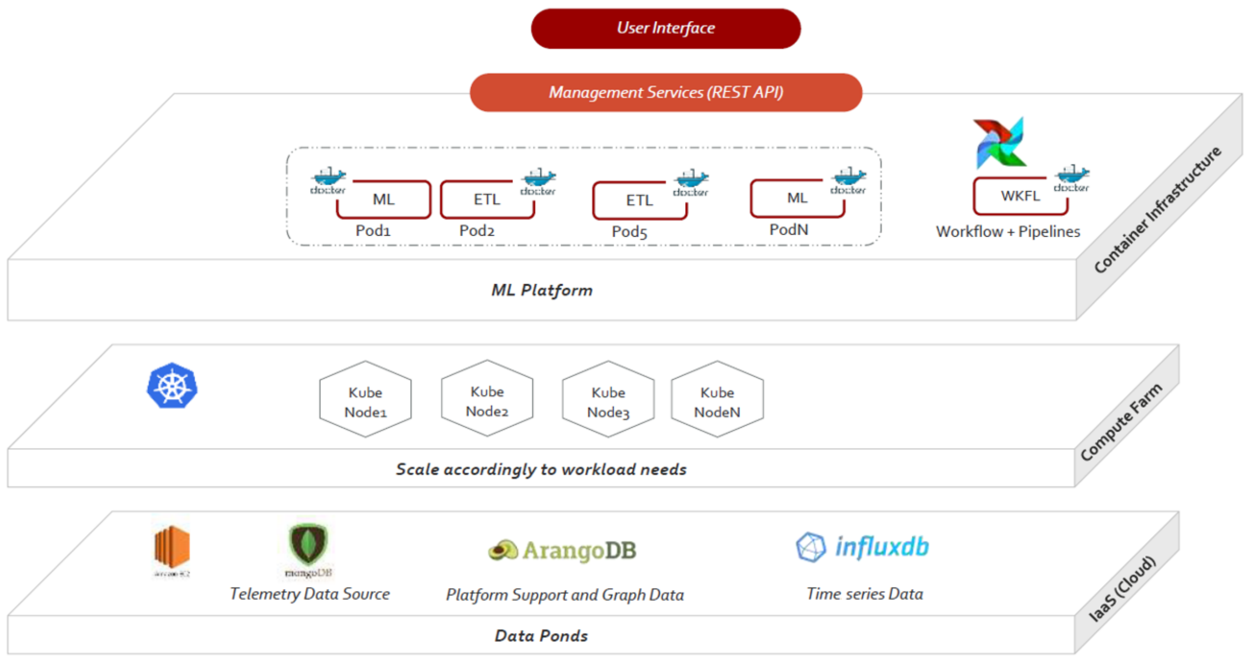
- Improving operational efficiency
- Reducing unplanned maintenance and downtime
- Increasing ROI

The engine is self-learning, and can look at what is happening and apply knowledge across all appliances. It provides proactive monitoring of customers' appliances.

Technical architecture

> *"We try to aim for the best accuracy. So the solution is to leverage InfluxDB in this case. Then we compute the error online and save it as a time series. That's why we use InfluxDB."*

InfluxDB in Veritas' Custom ML Platform



The ML platform's design encompasses a container infrastructure, compute farm and IaaS (Cloud):

- The ML platform's pods run on a Kubernetes cluster that scales according to workloads needs.
- The Kubernetes cluster has four nodes for computational purposes (this is where the forecast gets run).
- Apache Airflow manages the ML pipelines.
- The Data Ponds layer has document-based storage (MongoDB and ArangoDB).
- InfluxDB is used for all the time series data.
- The platform uses Amazon S3 buckets.

Hardware Setup

K8s cluster:

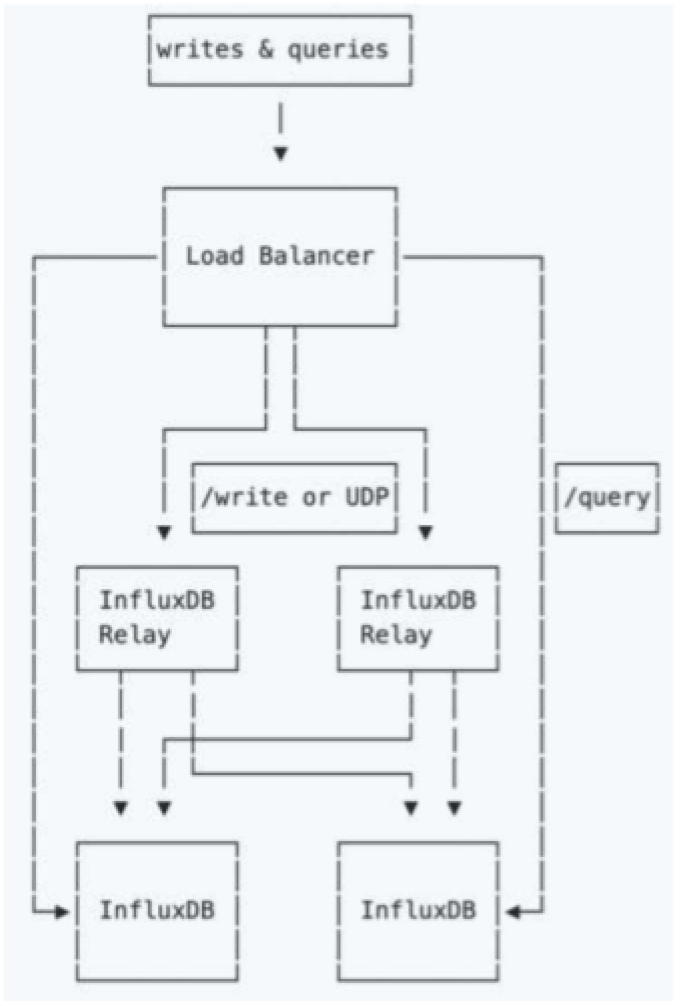
- 4 x m4.2xlarge (gen purpose)
- 8 vCPU
- 32 GiB RAM

InfluxDB:

- 2 x r4.xlarge (mem opt)
- 4 vCPU
- 30.5 GiB RAM
- gp2 SSD 3000 IOPS

Nginx load balancer:

- 5 workers
- 4096 connections/worker



The machines that Veritas uses to run InfluxDB are not really big from a computational viewpoint but have a lot of RAM and SSD storage with high IOPS in order to support a high throughput. This is because InfluxDB requires faster storage and a lot of RAM to deliver high performance.

The setup uses an Nginx load balancer, which runs on its own machine, on one of the two InfluxDB nodes. Veritas found that Nginx can be, in itself, a bottleneck and recommend ensuring that there are enough workers and connections allowed on it to prevent it from becoming the bottleneck. They found this to be the case especially with heavy workloads, where Nginx starts caching and consuming the RAM of one of the two nodes. They plan to move from the Nginx load balancer to an Elastic Load Balancer in the production deployment in AWS.

Once Veritas completed the hardware setup in their ML platform, they needed to run the forecast in an automated way. They have more than 10,000 appliances, and for each, they are forecasting for each type of storage partition. The system has to be able to detect any anomaly and take action based on whatever it finds.

To automate storage forecast implementation, they had to overcome the three challenges of forecasting automation:

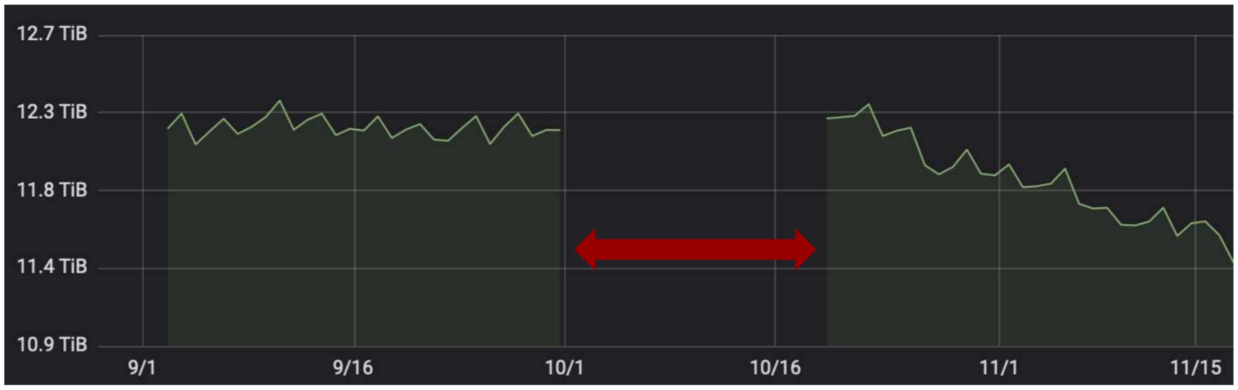
1. **Model selection** – Which model is the best?
2. **Model evaluation (validation)** - How accurate is the model currently running in production?
3. **Model update (online tuning)** – How to keep tuning/improving the model?

All of this must be done unsupervised.

Model selection

Selecting the best model for the type of data they have can be done manually because they assume that the data is coming from a similar source. If you find a model that works for one type of time series, it is reasonable to expect it to perform across all the similar time series. But numerous issues remain:

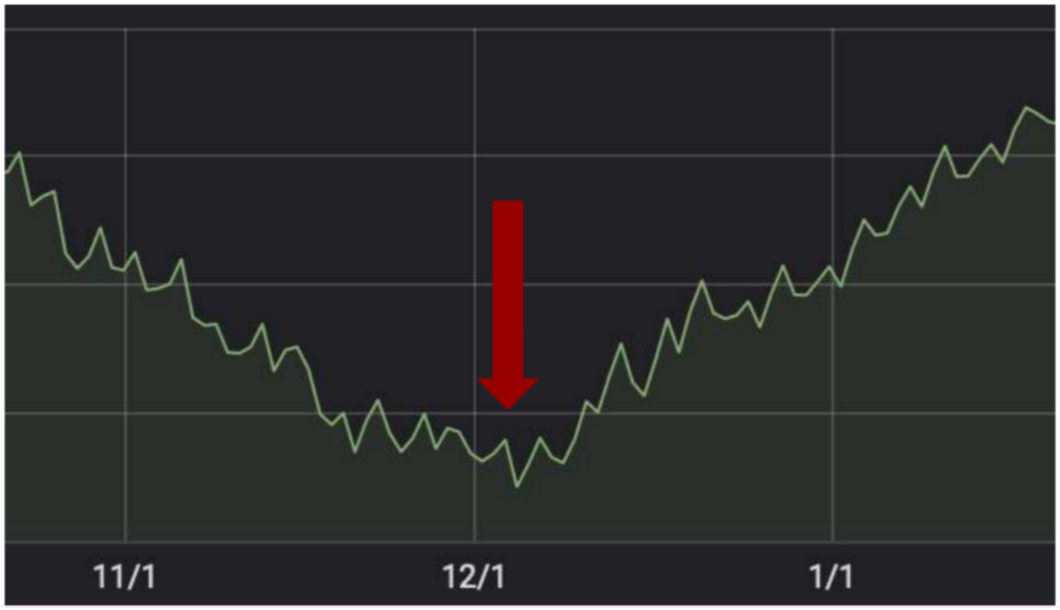
- **How they handle missing values:** If they suspect missing values, they conduct a process called data imputation where they fill in those missing values. But sometimes the NetBackup appliances might not record the missing values for an extended period, and the algorithm has to be able to handle this missing data.



- **How they handle outliers:** There might be outliers, due to some bug or because outliers actually exist. They might decide to remove them so that they don't skew the forecast. But in order to do that, they need to be able to automatically detect them. That in itself is a tough problem.

- **How they handle trend & seasonality:** There are considerations like trend and seasonality decomposition, which have to be extracted in an automated manner. Also, seasonality exists in different types. There could be multiple seasonalities, with different periods (daily or weekly seasonality). These differences need to be detected in the data.

- **How they handle trend changepoints:** When they extract the trend, the time series data is not just monotonic but might actually have a change in the trend. Trying to fit a simple linear model without taking into account those changepoints would result in an invalid model.



- **How they choose algorithm parameters:** Once they choose their model, they still have to tune it. They have to find a set of parameters of the model that provides the best performance. Running a cross-validation, since they have 70,000+ time series, would be a really expensive process. With real-world data being “messy” — and given their need to tackle each case while facing impossible deadlines — they settled on a model that solves the problem of choosing algorithm parameters: Prophet, a tool developed by Facebook for forecasting at scale.

Prophet has the following three components:

1. Growth component, **g**
2. Seasonal component, **s**
3. Holiday component, **h** (to account for extreme events like Christmas or the Superbowl)

Veritas selected Prophet for its functionalities:

- Decomposable time series model

$$y_t = g(t) + s(t) + h(t) + \epsilon_t$$

- Incorporate trend changes in the growth model by explicitly defining changepoints
- Fourier series to provide a flexible model of periodic effects (seasonality)
- Holidays as an additional regressor
- Observations do not need to be regularly spaced
- Handle missing values
- Robust to outliers (it fits whatever data you have, which makes it easier to run at scale)

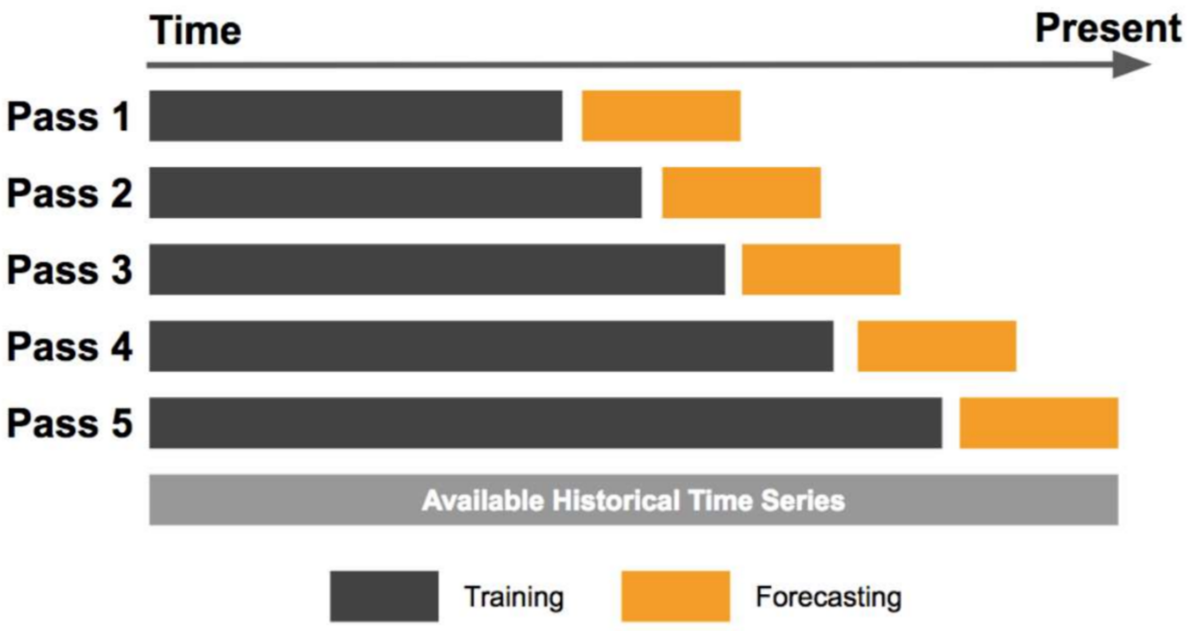
Model evaluation

To validate that their model actually performs, Veritas needed to have some accuracy number, which is their error. There are different types of error measures for time series data. But it is important to keep monitoring the model as it runs in production in order to avoid the risk of having an outdated model.

The cross-validation process in time series data is a little different than the usual cross-validation process in machine learning because time series data is ordered by time, so you can't take a random sample of the data. You have to take a continuous partition of data in order to train the model.

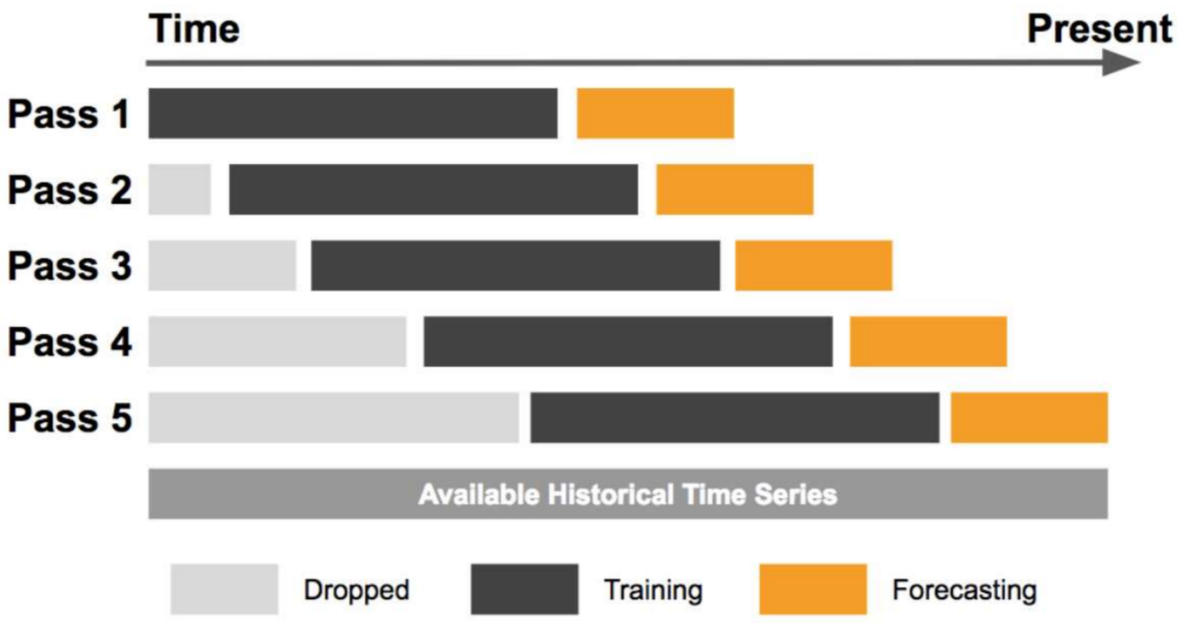
They use the procedure called Expanding Window Validation where — given available historical time series (such as one year of data) — they start with a small amount at the beginning. Then, they forecast, and compare against historical data so they can compute the error. Then, they increase the data they used and forecast again. They repeat this process until they use all the available historical data.

Cross Validation: Expanding Window Validation



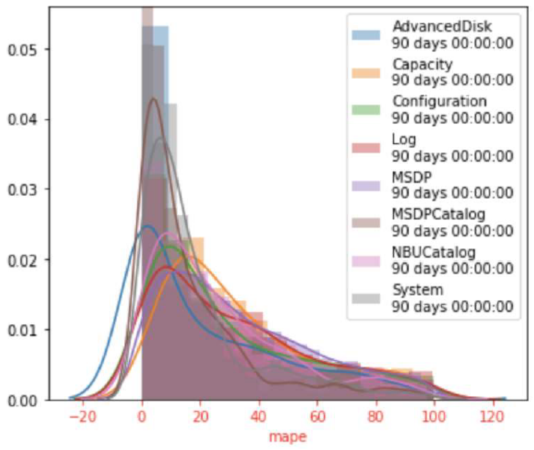
Another way to do this cross-validation process is Sliding Window Backtesting. At the points to their training data, they use a fixed-size window. They forecast and compare against available historical data. This second procedure is more useful when, for example, they know that the process that is generating the data is slowly changing its characteristics. They want to use only a recent window of data for the forecast. Otherwise, the forecast might take into account all the data that is not relevant and thereby generate an invalid forecast.

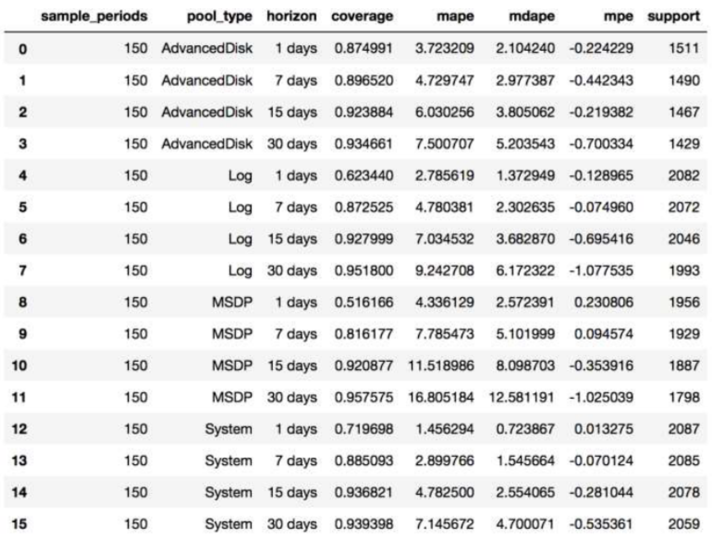
Cross Validation: Sliding Window Backtesting



This procedure has to be run for every time series, for every model, and for all the sets of hyperparameters they want to test. So it's a very expensive procedure, but if they run it, they obtain results as shown below. For each partition, they have a horizon — which is how far in the future they forecast — and they can generate a set of error measures.

Cross Validation: Experimental Results





	sample_periods	pool_type	horizon	coverage	mape	mdape	mpe	support
0	150	AdvancedDisk	1 days	0.874991	3.723209	2.104240	-0.224229	1511
1	150	AdvancedDisk	7 days	0.896520	4.729747	2.977387	-0.442343	1490
2	150	AdvancedDisk	15 days	0.923884	6.030256	3.805062	-0.219382	1467
3	150	AdvancedDisk	30 days	0.934661	7.500707	5.203543	-0.700334	1429
4	150	Log	1 days	0.623440	2.785619	1.372949	-0.128965	2082
5	150	Log	7 days	0.872525	4.780381	2.302635	-0.074960	2072
6	150	Log	15 days	0.927999	7.034532	3.682870	-0.695416	2046
7	150	Log	30 days	0.951800	9.242708	6.172322	-1.077535	1993
8	150	MSDP	1 days	0.516166	4.336129	2.572391	0.230806	1956
9	150	MSDP	7 days	0.816177	7.785473	5.101999	0.094574	1929
10	150	MSDP	15 days	0.920877	11.518986	8.098703	-0.353916	1887
11	150	MSDP	30 days	0.957575	16.805184	12.581191	-1.025039	1798
12	150	System	1 days	0.719698	1.456294	0.723867	0.013275	2087
13	150	System	7 days	0.885093	2.899766	1.545664	-0.070124	2085
14	150	System	15 days	0.936821	4.782500	2.554065	-0.281044	2078
15	150	System	30 days	0.939398	7.145672	4.700071	-0.535361	2059

Here, they used Mean Absolute Percentage Error (MAPE). MAPE is the difference between the forecast data and the actual data divided by the value of the actual data, and transformed as a percentage.

Backtesting vs. Online Testing

There are thousands of series, and they have to run the validation procedure for each series. They might also want to try more than one model per time series. So it becomes very expensive to compute all this validation data.

Since it's computationally expensive, the process might be run just as a one-off batch once a month to check system status, but this means results are always outdated (lagging behind what they have in production). That might be an issue because they try to aim for the best accuracy. So the solution is to leverage InfluxDB. Then they compute the error online and save it as a time series.

How they do online computation of the accuracy data

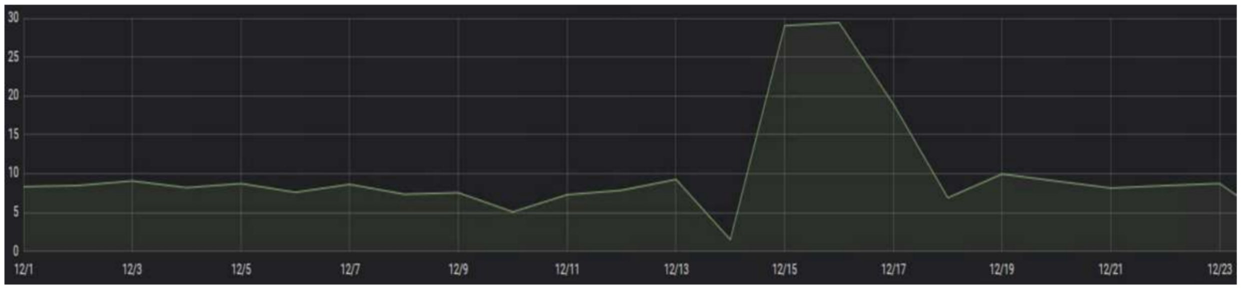
1. They save all previous **forecast data** along with the historical data because it is used to compute the error metrics. In the previous procedure, they were just iterating over each time

series, but in this case, they have to store the data and then retrieve it at a later point in time to get those accuracy metrics. They save forecast data for each appliance and storage type.

2. Also, since this data gets queried in the future, they do **multi-horizon forecasts**. That's because if they do the Sliding Window or Expanding Window validation process, they can repeat the process for multiple horizons. In this case, once the forecast has been computed, there's no way to go back in the past and update the forecast with a different horizon, unless they rerun the forecast on the past data. And that will defeat the purpose of avoiding all the repeated computation.
3. As they generate new forecasts, they compute **accuracy data** based on past forecasts and past historical data. This step is useful because it:
    - Is very computationally efficient since the computation is distributed over time.
    - Enables A/B testing. You can run two models at the same time and keep track of their error as a time series to see who's performing better.
    - Enables real-time monitoring. You can check the error again. If the error, for some reason, spikes up, then you know that something changed in the underlying process of data, and you can examine what's happening.

They forecast the error as a time series per appliance, per horizon and per storage type.

Backtesting vs. Online Testing



Forecast Data

How they save the data in InfluxDB

Solution 1:

In Solution 1, they have one measurement for all the historical data and one measurement for all the forecasts.

The tags they set are:

- `Type` (of storage)
- `Serial` (appliance identificator)
- **`History_cutoff`** (has the same value as **`Cutoff`** field in the list below)

`History_cutoff` represents the timestamp of the last point in the history of the data used to train that specific forecast. When they `GROUP BY history_cutoff`, they group by a specific forecast.

The fields they set are:

- `Yhat` (for historical data)
- `Yhat lower` and `Yhat upper` (both for forecast data)
- `Cap`
- **`Cutoff`**

This approach was simple and very effective, but presented two problems:

1. **Fetching data twice:** Every time they needed to compute the accuracy data, they had to query both the historical data measurement and the forecast measurement.
2. **Can't have forecasts in same table as histories:** They faced cardinality in the measurements. This is because the serial has a cardinality of about 10,000; type has a cardinality of around 10; and **history_cutoff** might grow in cardinality to a large number (because if, for example, they run the forecast every day, then they will have a new value for **history_cutoff** every day).

Solution 2:

In order to scale the approach in Solution 1 above, they moved to Solution 2 where:

- They have one measurement per appliance.
- Each measurement contains both historical and forecast data.
- They had to add a Boolean flag in history (as a tag, used to identify historical vs. forecast data).

This approach scaled well and solved their cardinality issue.

The challenge remaining was that they might not want to retain all the forecast data or might want to retain it for a short period compared to historical data. To solve it, they set up a retention policy specifically for forecast data or have a DELETE QUERY running as a background cleanup process.

The tags they set for Solution 2 are:

- `Type`
- `Serial`
- **`Is_history`**
- `History_cutoff`

The fields they set are:

- `Yhat (for historical data)`
- `Yhat lower (for forecast data)`
- `Yhat upper (for forecast data)`
- `Cap`
- **`Cutoff`**

Data

How they lay out the accuracy data in InfluxDB

Solution 1:

They have 1 measurement for all the accuracy data. They set as tags the storage types, the serial of the appliance, and then the horizon. They forecast for multiple horizons so that they can compute the accuracy metrics for multiple horizons. The fields are those numerical values for the errors.

Tags:

- `Pool_type`
- `Serial`
- **`Horizon`**
- `Train_freq`
- `Sample_periods`
- `Other` **`hyperparams`**

Fields:

- `coverage`
- `mape`
- `mdape`
- `smape`
- `etc`

This setup proved very useful if they needed to rely on cross-analysis. If they want to build the “Cross Validation: Experimental Results” table previously shown with all the error metrics and the error distributions across the appliances , this data layout proved very easy to deal with.

However, the cardinality of forecast data becomes even worse with this solution.

Solution 2:

To solve the cardinality issue, they moved to a solution where they split the accuracy data per appliance (one measurement per appliance).

They set as tags:

- `Pool_type`
- `Serial`
- **`Horizon`**

They set as fields:

- `coverage`
- `mape`
- `mdape`
- `smape`
- `etc`

Solution 2 scaled very well but presented the following problems:

- Cross-analysis means fetching from 10K+ measurements.
- They could not track model updates/configuration (what model generated the accuracy numbers and what hyperparameters of the model were used to generate accuracy data).

Before covering the solution they developed for this last issue, it's helpful to look at how they do the online tuning. As mentioned, they could run the Sliding Window or Expanding Window cross-validation for the tuning. That is a good one-off run to get the initial values of the hyperparameters of the model. It is not enough to keep updating the model while it's running in production so that it always has the best performance.

Model update (online tuning)

They had to solve several model update problems:

- They have thousands of models.
- Each model needs to be tuned for a specific time series (they need to tune more than 70,000 models).
- Need to adapt to changes in underlying process (otherwise, the performance of their model will drift. Again, the backtesting procedure works well if it's run one-off. It doesn't really work well if they need to run it often.)
- Backtesting is computationally too expensive (but they have the online validation data).

To solve these problems, they rely on an effective mathematical tool called Sequential Model-Based Optimization (SMBO). SMBO consists of three components:

- **Hyperparameter optimization:**

$$x^* = \arg\min_{x \in \chi} f(x)$$

  where f(x) represents an objective score (the error) to minimize — such as Root Mean Square Error (RMSE) — evaluated on the validation set.

- **Surrogate function:** Bayesian optimization uses a surrogate function to build a probability model of the objective.

$$P(f(x)|x)$$

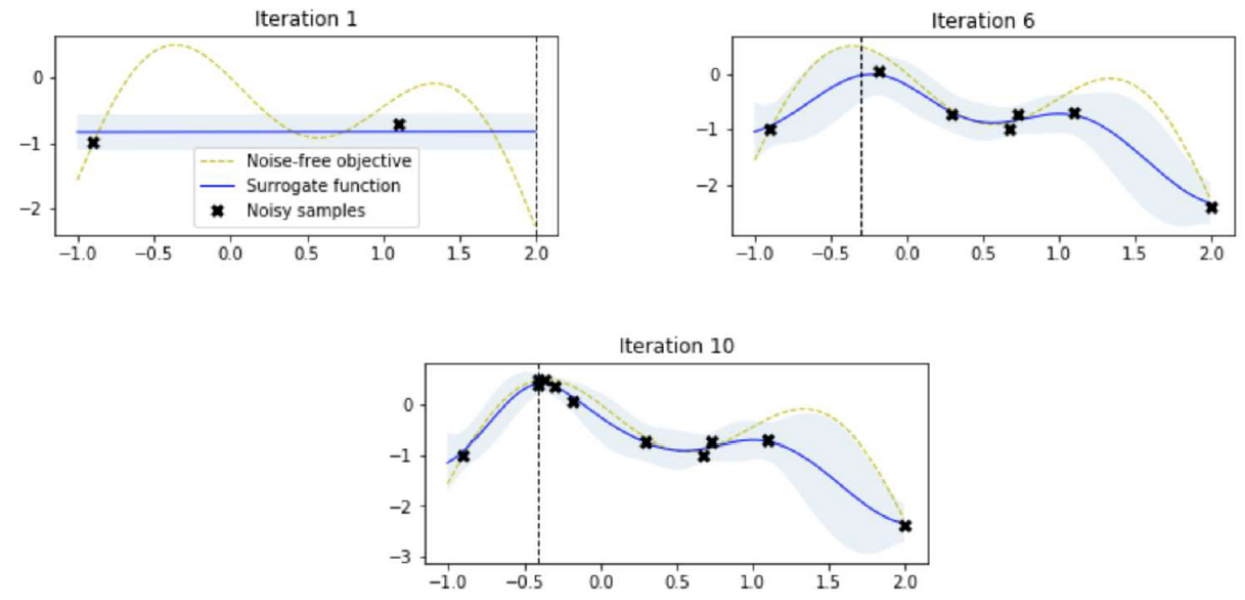
  The surrogate function tries to model — given model **f** and the set of hyperparameter **x** — the future performance of the model so that they can use this in combination and repeat the process using a selection function (a function that gives you the set of hyperparameters **x**). They repeat this procedure in iterations until they minimize the objective score.

  Bayesian optimization attempts to find the global optimum in a minimum number of steps. It incorporates prior belief about **f** and updates the prior with samples drawn from **f** to get a posterior that better approximates **f**. The model used for approximating the objective function is called *surrogate model*.)

- **Selection function:** This is used to choose the next set of hyperparameters.

The graph below shows how SMBO works using the surrogate function.

Sequential Model-Based Optimization (SMBO)



At the beginning, all the hyperparameters are equally probable because there's no prior knowledge of the error. There is some function of the error — shown in the yellow dashed line:

1. They sample, starting start from an initial set of parameters.
2. They check the error and update the surrogate function.
3. The surrogate function tends to converge towards the actual error function.
4. After some iterations, they select the set of parameters that has the smallest error.

They can run this procedure online while doing the forecast because they have the past forecast data and the past historical data. So they can compute the error and then update, after every run, the surrogate function and then get a new set of hyperparameters.

To enable sequential model-based optimization, here is how they lay out the data in InfluxDB. They go back to Accuracy Data Solution 2 above and follow it with subsequent solutions, Solution 3 and Solution 4 below).

Solution 3:

In Solution 3, they added two more tags to those in Solution 2:
- `model_type` to identify what model was used to generate that accuracy data
- `hyperparameters` used to generate that accuracy data

However, Solution 3 still presented the same problem of cross-analysis as Solution 2, but also introduced new problems because `model_type` and `hyperparameters` usually change:

- **Hyperparameters are different for different algorithms:** The hyperparameters are specific for each model.
- **Hyperparameters cardinality can be really high:** If they switch models often, then they have an explosion of the cardinality of accuracy data. Also, some models have a lot of hyperparameters. For example, neural networks have tons of parameters, and it's not realistically feasible to store all of them as tags. There is no simple solution for that.

Solution 4

Rather than relying solely on InfluxDB to solve the issues (faced in Solution 3 above), they externalized the hyperparameters. In Solution 4, cross-analysis meant fetching from 10K+ measurements *and referencing another database*:

- They replaced the tags **model_type** and **hyperparameters** with a new tag — the hyperparameter identifier **hyperparams_doc_id**. This identifier is a foreign key to a document-based store (in this case ArangoDB).
- For each document, they have the model type and all the parameters of that model. So when they query this data, the **hyperparams_doc_id** identifies both the model and set of parameters that were used to generate that accuracy data.
- Whenever Veritas' SMBO process trains their model, they can pass the new settings to the forecast algorithm and the data will automatically save the **hyperparams_doc_id** associated with that model and set of parameters as part of the accuracy data.

Results

> *"We saw that InfluxDB is a very powerful tool to enable forecasting at scale."*

Recognizing the potential and versatility of InfluxDB as an open source purpose-built time series database, Veritas used it in all their storage use cases. They put to work its attributes (high write throughput, high query throughput, low disk storage requirements) to enable forecasting at scale:

- They carefully chose a setup that optimizes IOPS and memory
- They also addressed the series cardinality issue.
- They provided a basic data layout to support online accuracy evaluation and algorithm tuning for time series forecasting.

Powered by InfluxDB as its time series database, and available for Veritas appliances, Veritas Predictive Insights delivers immediate value for both new and existing installations with prescriptive support services that can mitigate problems before they occur.

Veritas Predictive Insights enables Veritas to provide operational support by monitoring system health, detecting potential issues and launching proactive remediation. Veritas Predictive Insights is available as a service feature for Veritas appliances. It is automatically turned on without additional cost to customers or need for licensing.

Continuous AI/ML self-learning processes in Veritas' platform constantly improve insights and accuracy, identifying patterns, predicting trends and optimizing resiliency and utilization with intelligent forecasting and predictive maintenance. By introducing AI-based operational capabilities inside Veritas appliances, Veritas is giving its customers better resiliency and reduced downtime risk. Veritas Predictive Insights:

- **Gives customers a more holistic view of their environment.** Its capacity forecasting helps customers flatten out acquisition models. Because customers couldn't forecast consumption needs, that led to issues with spikes in the procurement process, and in the worst cases led to downtime.
- **Eliminates 'alert fatigue' through prescriptive maintenance.** This saves techs the need to deal with thousands of alerts in an enterprise environment and thereby increases operational efficiency. This benefit also helps determine what should be done first at the remediation level and levels the playing field for support tech experience, by making less experienced techs more able to take action.

By translating data into actionable value for its NetBackup customers, Veritas is fulfilling its purpose of empowering businesses of all sizes to discover the "truth in information".

About InfluxData

InfluxData is the creator of InfluxDB, the leading time series platform. We empower developers and organizations, such as Cisco, IBM, Lego, Siemens, and Tesla, to build transformative IoT, analytics and monitoring applications. Our technology is purpose-built to handle the massive volumes of time-stamped data produced by sensors, applications and computer infrastructure. Easy to start and scale, InfluxDB gives developers time to focus on the features and functionalities that give their apps a competitive edge. InfluxData is headquartered in San Francisco, with a workforce distributed throughout the U.S. and across Europe. For more information, visit influxdata.com and follow us @InfluxDB.

548 Market St. PMB 77953, San Francisco, CA 94104