STORAGE DEVELOPER CONFERENCE

SDC 23

BY Developers FOR Developers

Applying AI/ML Methodologies to Categorize Storage Workloads and Replaying them in Standard Test Environments

Dhishankar and Padmanabhan
Hewlett Packard Enterprise

Agenda

- Background (Abstract, Solution)
- Extracting the data from ASUP/Call home (Sampling)
- Model Details
  - Feature/Parameters Sampling
  - Model output
- Modeling
  - Details
  - Code Input
  - Layers
  - Output and Training
- Usage
- Conclusion

Abstract

- With the complexity of applications increasing every day, the workloads generated by these applications are complicated and hard to replicate in test environments. We propose an efficient method to synthesize a close approximation of these application workloads based on analyzing the historic autosupport (call-home) data from field using an iterative mechanism and also a method to store and replay these workloads in the test environment for achieving the goals of customer driven testing.

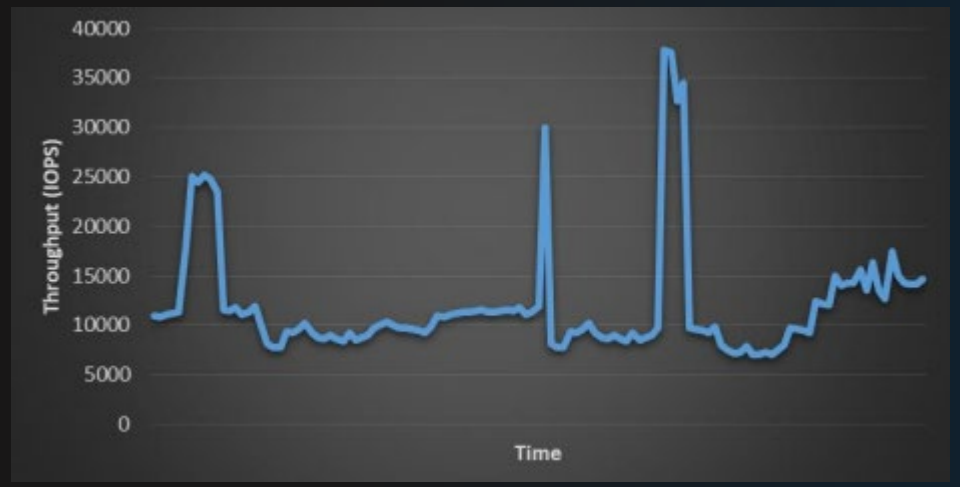
- Problem Statement: As we align more towards customer driven testing, the quantity and complexity of workloads grows exponentially. Most of our regression testing uses workloads designed for testing the functionality and stressing the array but some of the corner cases or race conditions are only caught using complex customer workloads. It is very difficult, and time consuming to model and synthesize customer workloads whenever there is a need to reproduce any escalations or POCs (proof of concept). The existing IO tools don't have any direct mechanism to simulate these customer workloads. Some tools do provide capability to capture and replay workloads but only at the host level. They also lack the capability to analyze array stats which is more significant for modelling customer workloads.

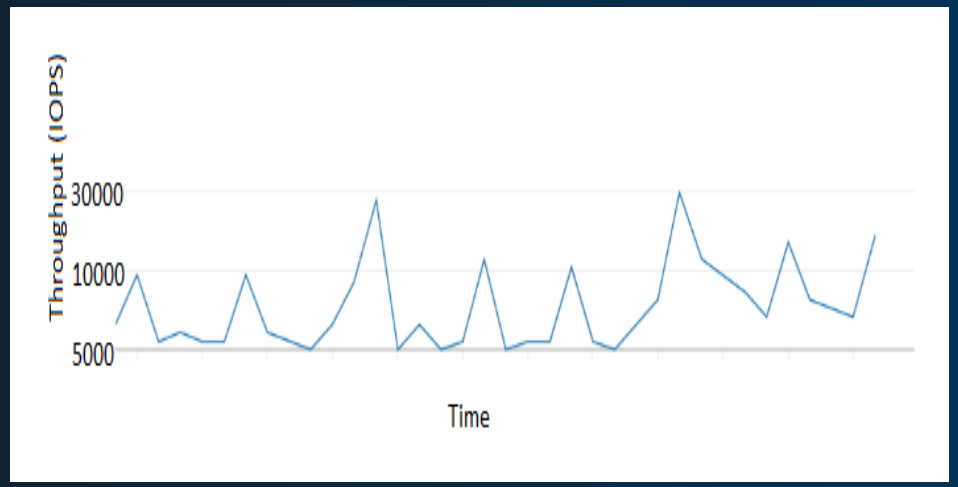
Solution

- Solution: In this paper we describe two solutions, viz. Workload Analyzer and Synthesizer (WAS) which analyses the array autosupport (ASUP) data to synthesize customer workloads and the Workload Matrix Solution1 (WMS) to integrate and deploy these synthesized workloads in existing test environments.

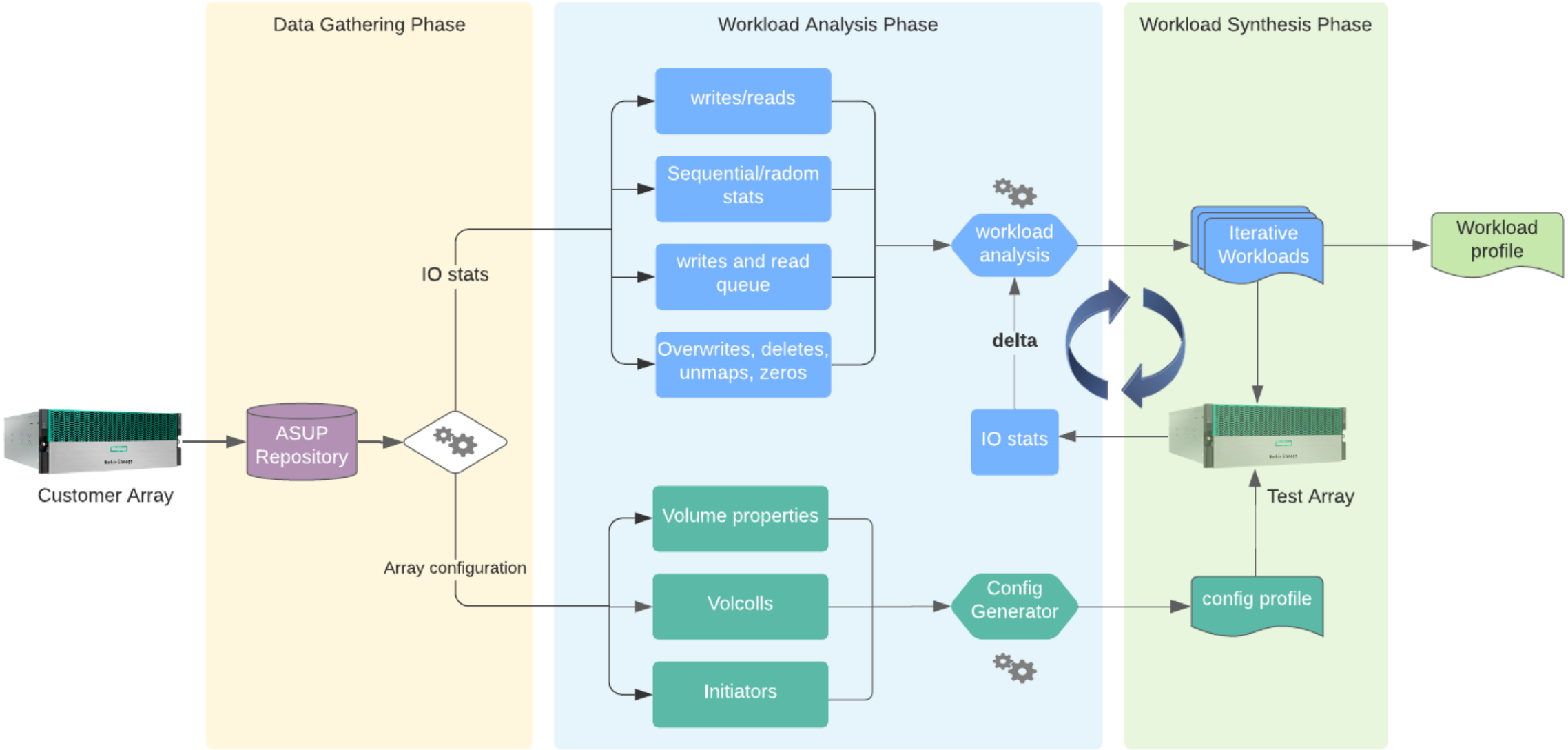
Existing Reference: AI/ML Based Customer IO Workload Simulation by Susanta Dutta (HPE)

Actual Example of Customer Workload





Extracting the Data



Configuration Parameters

- No. of volumes
- Volume attributes
  - Size
  - Cache settings
  - App aware settings
  - Dedupe settings
  - Compression settings
- Consistency Groups
- Snapshot schedules
- Replication (synchronous/asynchronous)

Workload Parameters – Model Input

Sensor/stats in Time series format (per second interval)

1. Histogram (range) of block sizes
2. Histogram (range) of latencies
3. No. of writes
4. No. of reads
5. No. of sequential blocks
6. No. of random blocks
7. Partial block writes
8. Volume background operations
9. System background operations
   a. Deletions
   b. Replication
   c. Garbage collection (GC)
10. No. special scsi operations
    a. write zeros
    b. unmaps
11. Cache usage
12. Memory usage

Model Details – Output

- Recommended Configuration Parameters (Via Setup Script)
  - Volume, count, size, other parameters
  - Features in Play (Dedupe%, Compression Ratio, Snapshots, etc)
  - Background Operations.
- Recommended IO Parameters
  - Range of block sizes
  - % Writes vs Reads
  - % Random vs Sequential
  - % Partial blocks (Unaligned IOs)
  - Queue Depth

Model Used

CNN layer (Conv1D) - to capture spatial patterns with histogram data

- Range of block sizes
- Range of latencies

LSTM (Long Short Term Memory) layer to capture temporal dependencies in time series data

- Background operations sensor data
- IO pattern – Random vs Sequential, Writes vs Reads, Partial Writes, Unmaps, write zeros.

Concatenate all the models to get the required output

Training/test data samples for the model is obtained from running IO on the storage array using known parameters (IO tool parameters - Y) and extracting the relevant features (sensor/stats data – x)

Code - Inputs

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv1D, Dense, Concatenate, Flatten
from tensorflow.keras.metrics import MeanAbsoluteError, CategoricalAccuracy


# X_hist_blocks: Histograms of block ranges time series data
# X_hist_latencies: Histograms of latencies time series data
# X_load: System load time series data
# X_random: Random I/O type values time series data
# X_sequential: Sequential I/O type values time series data
# X_partial: Partial I/O type values time series data
# y_read: Ground truth for read% workload parameter
# y_write: Ground truth for write% workload parameter

# Define the inputs
hist_blocks_input = Input(shape=(num_time_steps, num_bins))
hist_latencies_input = Input(shape=(num_time_steps, num_bins))
load_input = Input(shape=(num_time_steps, num_load_parameters))
random_input = Input(shape=(num_time_steps, num_io_parameters))
sequential_input = Input(shape=(num_time_steps, num_io_parameters))
partial_writes_input = Input(shape=(num_time_steps, num_io_parameters))
```


Code – Model Layers

```
# Define CNN layers for histograms
conv_hist_blocks = Conv1D(filters=32, kernel_size=3, activation='relu')(hist_blocks_input)
conv_hist_latencies = Conv1D(filters=32, kernel_size=3, activation='relu')(hist_latencies_input)

# LSTM layer for load parameters
lstm_load = LSTM(units=64, return_sequences=True)(load_input)
lstm_output_load = LSTM(units=64)(lstm_load)

# LSTM layer for I/O types
lstm_io = LSTM(units=64, return_sequences=True)(Concatenate()([random_input, sequential_input, partial_writes_input]))
lstm_output_io = LSTM(units=64)(lstm_io)

# Concatenate all the outputs
concatenated_output = Concatenate()([conv_hist_blocks, conv_hist_latencies, lstm_output_load, lstm_output_io])

# Fully connected layers
fc_layer = Dense(128, activation='relu')(concatenated_output)
fc_layer = Dense(64, activation='relu')(fc_layer)
```


Code – Output Layers, Build and Compile

```
# Output layers for each parameter
output_block_range = Dense(10, activation='linear', name='block_range')(fc_layer)  # Predict a range for each of the 10 block size ranges
output_latencies_range = Dense(num_bins, activation='linear', name='latencies_range')(fc_layer)  # Predict a range for each latency bin
output_io_counts = Dense(2, activation='linear', name='io_counts')(fc_layer)  # Predict number of reads and writes
output_io_types = Dense(4, activation='softmax', name='io_types')(fc_layer)  # Predict the probabilities of 4 I/O types
output_partial_unmaps_zeros = Dense(3, activation='softmax', name='partial_unmaps_zeros')(fc_layer)  # Predict the probabilities of 3 categories


# Build the model
model = tf.keras.models.Model(
    inputs=[hist_blocks_input, hist_latencies_input, load_input, random_input, sequential_input, partial_input],
    outputs=[output_block_range, output_latencies_range, output_io_counts, output_io_types, output_partial_unmaps_zeros])


# Compile the model
model.compile(optimizer='adam',
              loss={'block_range': 'mean_squared_error',
                    'latencies_range': 'mean_squared_error',
                    'io_counts': 'mean_squared_error',
                    'io_types': 'categorical_crossentropy',
                    'partial_unmaps_zeros': 'categorical_crossentropy'},
              metrics={'block_range': MeanAbsoluteError(),
                       'latencies_range': MeanAbsoluteError(),
                       'io_counts': MeanAbsoluteError(),
                       'io_types': CategoricalAccuracy(),
                       'partial_unmaps_zeros': CategoricalAccuracy()})
```

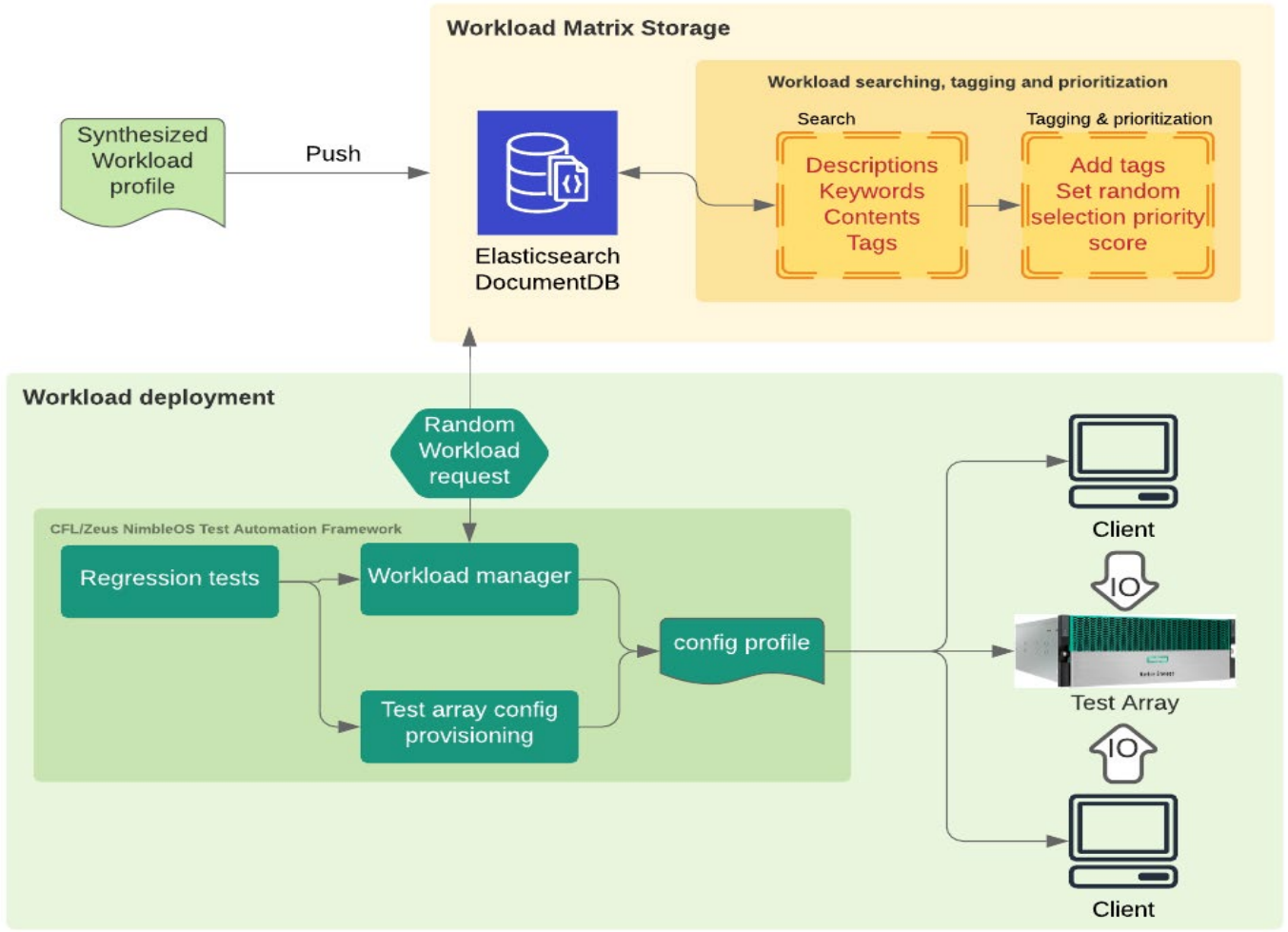

Train and Evaluate the Model

```
# Train the model
model.fit(
    {'hist_blocks_input': X_train_hist_blocks,
     'hist_latencies_input': X_train_hist_latencies,
     'load_input': X_train_load,
     'random_input': X_train_random,
     'sequential_input': X_train_sequential,
     'partial_input': X_train_partial},
    {'block_range': y_train_block_range,
     'latencies_range': y_train_latencies_range,
     'io_counts': y_train_io_counts,
     'io_types': y_train_io_types,
     'partial_unmaps_zeros': y_train_partial_unmaps_zeros},
    epochs=num_epochs,
    batch_size=batch_size,
    validation_split=0.2)

# Evaluate the model
loss, block_range_mae, latencies_range_mae, io_counts_mae, io_types_accuracy, partial_unmaps_zeros_accuracy = model.evaluate(
    {'hist_blocks_input': X_test_hist_blocks,
     'hist_latencies_input': X_test_hist_latencies,
     'load_input': X_test_load,
     'random_input': X_test_random,
     'sequential_input': X_test_sequential,
     'partial_input': X_test
```


SDC 23

Usage:



VDBench Example:

```
dedupratio=4
dedupunit=4k
compratio=4
wd=wd_vsi_read,rdpct=100,xfe
rsize=(8k,16.5k,32.6k,124k),
sd=sd*
wd=wd_vsi_write,rdpct=0,xfer
size=(4,3k,8.5k,32k,124.2k),
sd=sd*
```


How Frequently do we revisit

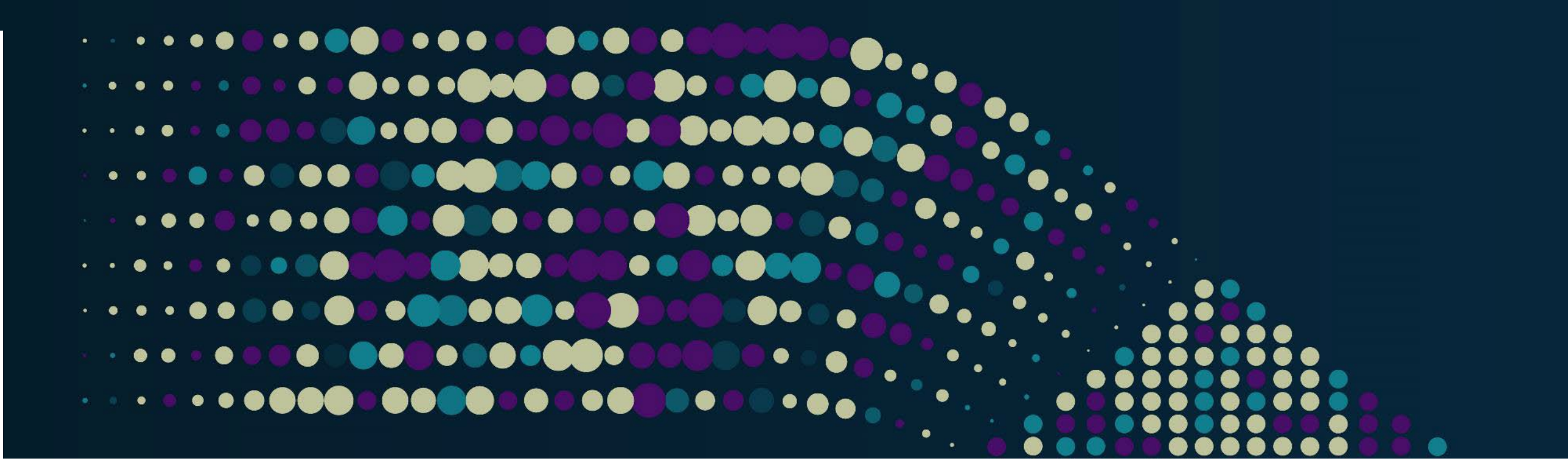
- Systems are classified as Low, Mid and High-End Storage Systems
  - We use 3 different models separately for each category.
- Training happens based on phases in development and features.
  - Every Major/minor releases
  - Any new Customer Issues encountered

Conclusion

- Not new, but newer techniques and models are emerging
  - External tools like vdbench although can replay workloads using the Sun StorageTek Workload Analysis (SWAT)2 feature, but don’t have a way to analyze the array side stats and automatically generate profiles
- Getting closer to real world actuals
  - All the relevant factors that impact workloads have been carefully considered in synthesizing new workload profiles. We’re also providing an iterative mechanism to tune the workload parameters based on delta from customer ASUP stats. Thus, we strongly believe that the outcome of this solution should be as close as real customer workloads.
- Getting to Customer Issues quicker
  - By continuously iterating through the customer ASUP data, we are able to model the workloads closer to actual usage scenarios over a longer period of time which is possible to get to with synthetic workloads in shorter period of time

Thank You

- For queries please reach out to
  - Dhishankar Sengupta – dhishankar@hpe.com
  - Padmanabhan Pandurangan - padmanabhan.pandurangan@hpe.com



Please take a moment to rate this session.

Your feedback is important to us.

