SDC 18
September 24-27, 2018
Santa Clara, CA
www.storagedeveloper.org

Machine Learning to Detect Complex Workloads in Real-Time

Kenny Sun
Veritas Technologies LLC

Machine Learning to Detect Workloads

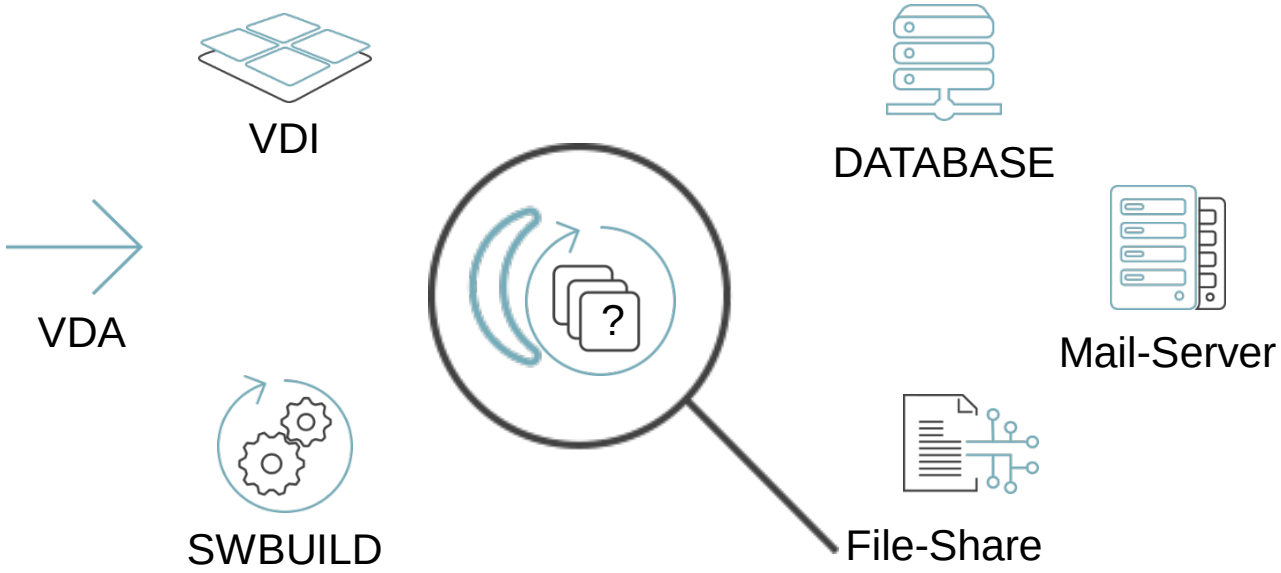
- Motivation
  - Complex new-age workloads
  - Hardware agnostic applications
  - Virtual layers in cloud infrastructure
  - Dynamically changing requirements
  - SLAs becoming more important
  - Importance of optimal use of resources

Setting the Stage

- Six different workloads with different patterns
  - VDI (VM desktops)
  - VDA (Streaming)
  - SWBUILD (DevOps)
  - Database
  - Mail-server
  - File-share

Objective

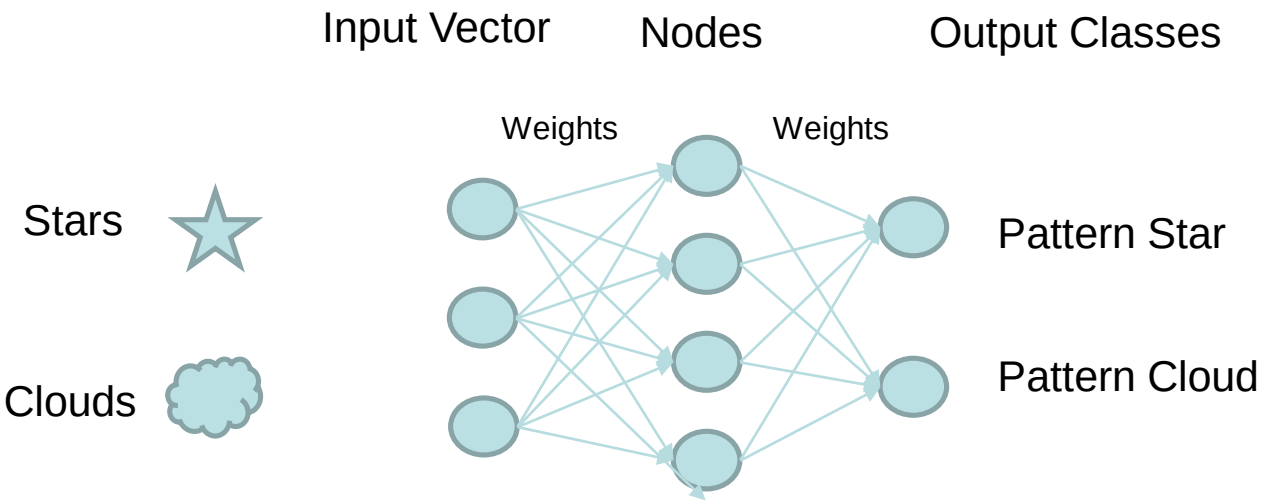
- Detect the workloads from one another



Method Overview

- Supervised Neural Networks (4 x 3)
- Use file system counters as inputs
- Predict workload type based on counter DNA
- Probability of each of the 6 types as outputs

Basics of Supervised Neural Networks



Feed forward the values →
Adjusting weightages in ← back-propagation of errors

Collecting Sampling and Pre-Processing

- Raw data for this experiment was file system counters from the VxFS file system. There are 283 file system counters that cover operations such as allocation and flushing of page cache, usage of inode cache, and all the vnode data and meta-data operations

Collecting Sampling and Pre-Processing

- These counters reflect the workload activities happening on the server as the IOs trickle down from applications to storage system
- Real-time data was collected for both training and testing the network (283 counters collected with 10 sec intervals)

Collecting Sampling and Pre-Processing

- Pre-processing is done by
  - Removing unused counters
  - Normalize by the highest count
  - Scaling data from 0 to 10
- The final input vector used for neural network had 119 counters

Samples of Raw and Processed Data

VXFSSTAT Counter data for File Share workload

vxi_alloc_emap	158346	vxi_superwrite	0
vxi_alloc_smap	0	vxi_write_asynccnt	0
vxi_alloc_expand_retry	0	vxi_write_synccnt	0
vxi_alloc_find_retry	0	vxi_write_dio	0
vxi_alloc_findfail	3	vxi_write_donetran	0
vxi_alloc_sumsum	16843	vxi_write_logged	0
vxi_alloc_findfix	0	vxi_write_logonly	0
vxi_alloc_mapflush	0	vxi_write_rand	0
vxi_alloc_prev	2821	vxi_write_seq	33674
vxi_alloc_search	33150	vxi_anonpgin	0
vxi_alloc_sumclean	0	vxi_anonpgout	0
vxi_alloc_try	35971	vxi_sectout	7776236
vxi_execpgin	0	vxi_pagecluster	0
vxi_execpgout	0	vxi_pagestrategy	0
vxi_bawrite	10536	vxi_pgout	74071
vxi_bwrite	42561	vxi_pgpgout	983420
vxi_btwrite	0	vxi_sync_delxwri	0
vxi_lwrite	376513	vxi_sync_inode	0
vxi_bdwrite	333952	vxi_sync_page	0
vxi_bdwrite_loglow	0	vxi_write_only	0
vxi_bcache_curkbyte	1693416	vxi_write_throttle	0
vxi_bcache_maxkbyte	6804512	vxi_read_to_map	0
vxi_bc_chunksteal	0	vxi_dirc_setup	0
vxi_bc_hits	604603	vxi_dirc_purge	0
vxi_bc_lookups	608783	vxi_bmap_lookup	430107
vxi_bc_reuse	4179	vxi_bmap_indir_lookup	0

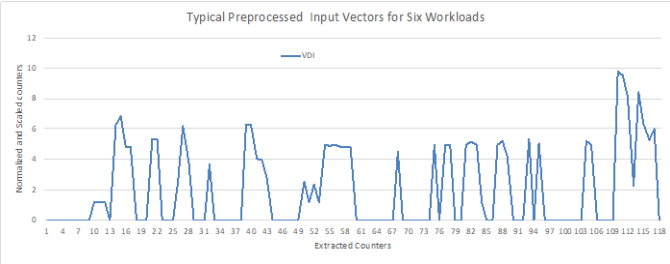
Sample of Raw vxfsstat Data

VDI	VDA	SWBUILD	DB	Postmrk	FileShare
0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000
0.000	1.146	0.000	0.477	5.341	5.200
0.000	0.301	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	1.531	0.000
0.000	0.000	0.000	0.000	2.676	0.602
0.000	0.000	0.000	0.000	5.103	4.226
0.000	0.000	0.000	0.000	1.322	3.451
0.000	0.000	0.000	0.000	5.165	4.520
0.000	0.000	0.000	0.000	5.165	4.556
1.146	1.681	4.336	1.447	3.855	4.023
1.146	1.681	4.665	1.462	4.949	4.629
1.146	1.869	4.888	1.531	5.910	5.576
0.000	1.431	4.491	0.778	5.859	5.524
6.241	6.242	6.133	5.985	6.229	6.229
6.833	6.833	6.833	6.833	6.833	6.833
4.817	2.956	5.541	4.363	6.211	5.781
4.817	2.956	5.541	4.363	6.211	5.784
0.000	0.000	1.204	0.000	1.898	3.621
0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.477	3.184	0.301	2.621	2.746
5.318	3.973	5.284	5.090	6.227	5.636
5.318	4.091	5.307	5.090	6.276	5.668
0.000	0.000	1.204	0.000	0.000	3.156
0.000	1.964	2.671	0.000	3.543	3.817
0.000	0.301	0.000	0.000	0.000	0.000
2.545	2.545	0.000	0.000	2.545	2.545

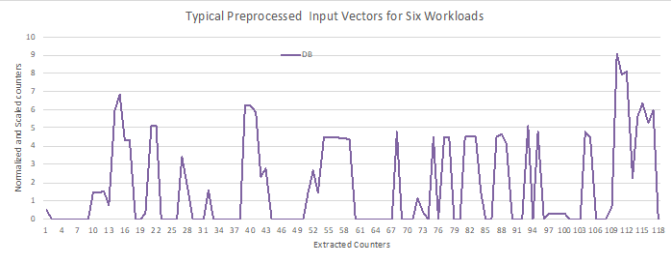
Sample of Raw vxfsstat Data

Input Vector

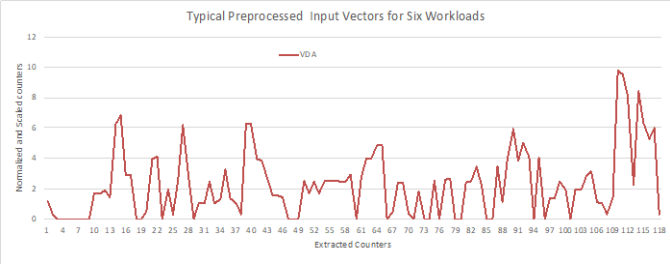
VDI



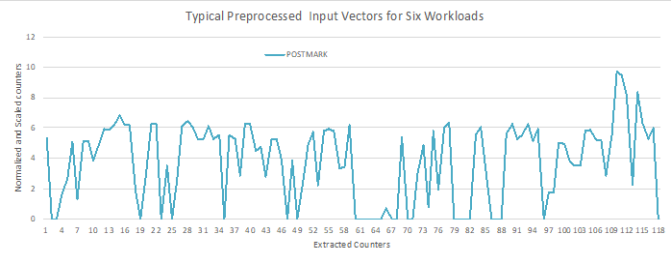
DATABASE



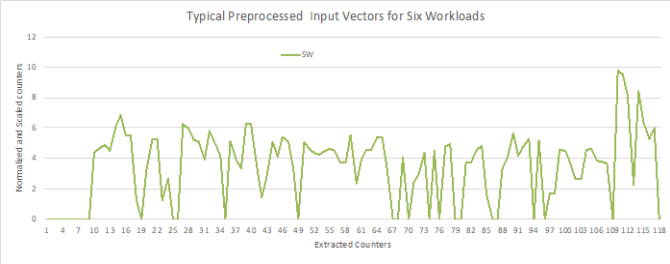
VDA



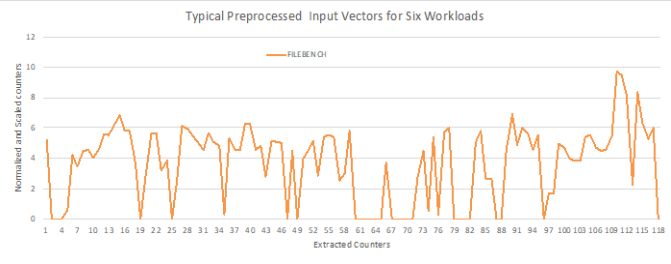
MAIL



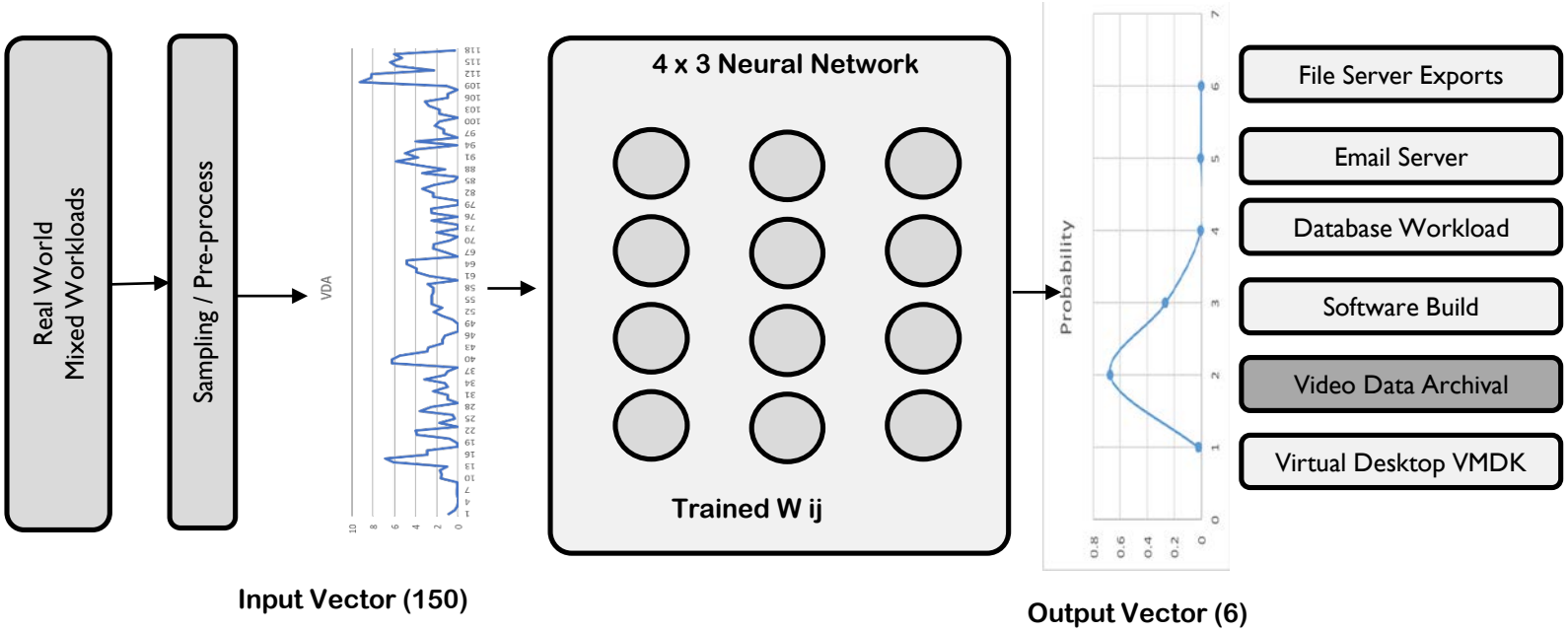
SWBUILD



FILE



Training and Testing of Neural Network



The Training Cycle

- 50 pre-processed input vectors were used for each workload (119 counters each vector) .
- A 3-layer / 4-node neural network was found sufficient to learn for classification. The Neural Network was trained iteratively using these 50 x 6 training patterns. Output classes were defined to be the six workloads.

The Training Cycle

- The network was trained by iteratively reducing errors in classification until classification efficiency was achieved
- The final set of weights were stored to be used during real-time testing of the workloads

Testing the Network

- The network was then applied to hundreds of un-seen test vectors gathered from real-life six benchmark workloads which were randomly fed to the trained network
- The network generated classification for each input vector as output

Output Vector for Six Workloads

	Six Noisy workloads generated by different benchmarks						
Classification by ML		VDI	VDA	SW	DB	Postmark	Filebench
	VDI	0.97970	0.00147	0.00103	0.00584	0.00173	0.00052
	VDA	0.00000	0.98072	0.00004	0.00014	0.00000	0.00000
	SW	0.00010	0.00323	0.97758	0.00018	0.00003	0.00061
	DB	0.00167	0.00005	0.00000	0.95408	0.00007	0.00072
	Postmark	0.00039	0.00020	0.00001	0.00013	0.99474	0.00001
	Filebench	0.00020	0.00006	0.00340	0.00230	0.00041	0.97522

Conclusion of the Experiment

- Small neural networks can be trained to identify workloads based on activity patterns
- Workload detection has a high success rate
- Detection can be done in real-time to take action
- Applications can be built to manage cloud infrastructure with different workloads

Possible Application Scenarios

- Detect anomalies, unbalanced use of resources
- Take proactive action to improve application performance
- Monitor and improve SLAs
- Help optimal scheduling of different applications

Q&A

- Thank You!