Machine Learning as a Service on Nimbus

Yukai Jin
Devtools, VMware Inc.
jinyuj@vmware.com

Jinghua Zhu
Devtools, VMware Inc.
jinghuaz@vmware.com

Yiting Cheng
VCAN, VMware Inc.
ycheng@vmware.com

Jinpeng Li
Devtools, VMware Inc.
jinpengl@vmware.com

Samson Wu
Devtools, VMware Inc.
samsonw@vmware.com

Jiayi Liu
Devtools, VMware Inc.
jiayil@vmware.com

Abstract

Machine learning requires large computing resources. We present a service of machine learning over container cloud of Kubernetes on infrastructure of VMware internal cloud of Nimbus[12] with CPU and GPU support. The service has user-friendly interface and consists of several micro services to launch TensorFlow container to train models. It aims to manage large computing resources in a center place to reduce company costs; it saves time for developers and data scientists to configure distributed machine learning platform. We present experiments on the performance comparison between CPU, GPU, multiple CPUs on our service.

Key Word

machine learning, machine learning as a service(MLaaS), tensorflow, kubernetes

1 Introduction

Nowadays, machine learning gets more and more popular, for example in audio recognition, face detection, trend prediction, word embedding for natural language understanding and etc. As machine learning algorithm becomes more and more complex, more computing power is needed. GPU resources are leveraged to get better result on accelerating the process.

In VMware, there also emerge various products using machine learning technology, such as VMware Analysis Cloud. Fortunately, VMware has a platform of Nimbus[1], which is an internal cloud infrastructure, and it is ready to make a machine learning service on it.

We build MLaaS based on Nimbus leveraging CPU and GPU resources. The service supports environment configuration, data and code management, log monitoring, running status visualization. It benefits data scientists a lot by simplifying their work on machine learning program runtime environment setup and accelerating program running speed.

In this article, we will describe our MLaaS on Nimbus by introducing basic user workflow, main architecture, evaluation and future work.

2 Related Work

The three public cloud companies build good examples of machine learning as a service. Google uses TensorFlow [2] on Google Cloud, meanwhile Amazon uses MXNet [3]. Microsoft has its own machine learning ecosystem on Azure and it has had a complete business intelligence platform for Microsoft SQL server. They all provide GPU servers to get higher performance on machine learning. In China, there are many companies working on building machine learning service as well. For example, Tecent, Baidu, Alibaba and so on. Most of services support 2 kinds of usage: build a machine learning train graph in UI, uploading data and training; the other is to upload train source code for learning.

3 Workflow definition

With this platform, a workflow within the online website that helps teams and developers to get machine learning to work in a short period is designed.

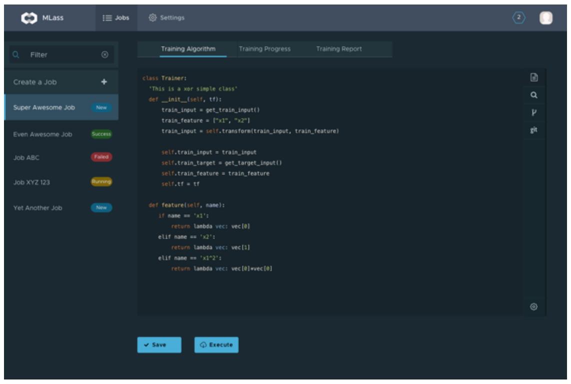
The workflow goes in several stages:

3.1 Job creation and environment configuration

A job is the minimum unit of machine learning application in the MLaaS system. The system allows user to create job based on abstracted computing resource regardless of details on hardware and system.

3.2 Algorithm creation

To allow a user make everything done in the same context, an online editor is integrated within the platform, with the real time executing.



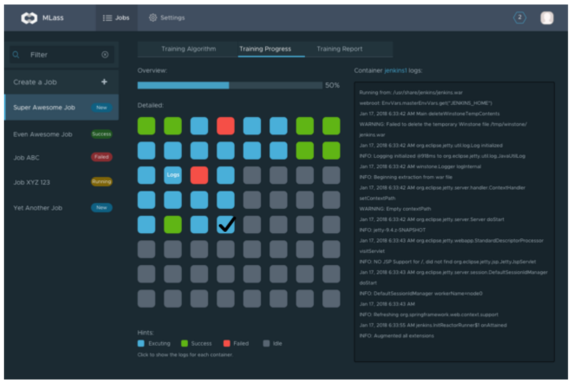
```
class Trainer:
  'This is a xor simple class'
  def __init__(self, tf):
      train_input = get_train_input()
      train_feature = ["x1", "x2"]
      train_input = self.transform(train_input, train_feature)

      self.train_input = train_input
      self.train_target = get_target_input()
      self.train_feature = train_feature
      self.tf = tf

  def feature(self, name):
     if name == 'x1':
         return lambda vec: vec[0]
     elif name == 'x2':
         return lambda vec: vec[1]
     elif name == 'x1^2':
         return lambda vec: vec[0]*vec[0]
```

3.3 Training, monitoring and debugging

To improve the experience of monitoring distributed machine learning execution, a monitor is designed to track each executor's status and fetch real-time logs for debugging.

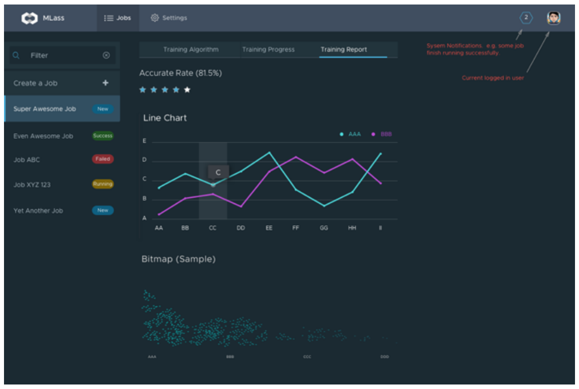


Container jenkins1 logs:

```
Running from: /usr/share/jenkins/jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
Jan 17, 2018 6:33:42 AM Main deleteWinstoneTempContents
WARNING: Failed to delete the temporary Winstone file /tmp/winstone/
jenkins.war
Jan 17, 2018 6:33:42 AM org.eclipse.jetty.util.log.Log initialized
INFO: Logging initialized @918ms to org.eclipse.jetty.util.log.JavaUtilLog
Jan 17, 2018 6:33:42 AM winstone.Logger logInternal
INFO: Beginning extraction from war file
Jan 17, 2018 6:33:42 AM org.eclipse.jetty.server.handler.ContextHandler
setContextPath
WARNING: Empty contextPath
Jan 17, 2018 6:33:42 AM org.eclipse.jetty.server.Server doStart
INFO: jetty-9.4.z-SNAPSHOT
Jan 17, 2018 6:33:43 AM org.eclipse.jetty.webapp.StandardDescriptorProcessor
visitServlet
INFO: NO JSP Support for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
Jan 17, 2018 6:33:43 AM org.eclipse.jetty.server.session.DefaultSessionIdManager
doStart
INFO: DefaultSessionIdManager workerName=node0
Jan 17, 2018 6:33:43 AM
INFO: Refreshing org.springframework.web.context.support
Jan 17, 2018 6:33:55 AM jenkins.InitReactorRunner$1 onAttained
INFO: Augmented all extensions
```

3.4 Evaluation and visualization

The system also provides approaches to auto visualize evaluation results according configuration.



3.5 Packaging and versioning

Deploying an algorithm as application goes far beyond the domain of machine learning itself. Thus, as a developer oriented machine learning solution, we designed a one- click deployment mechanism to provide a simple approach to package models into application and keep track of versions.

4 Architecture Design

Our platform mainly consists of four parts: UI, Management Service, Contractor, Booster. User will communicate with MLaaS through UI. Management service is responsible for handling user requests and send necessary message to message queue. Contractor listens to message queue and call Booster to allocate computation resources. After Booster completes work, it will tell Contractor through message queue. Then Contractor will deploy code and execute it. Result will be pushed to message queue so that management service can notify UI.

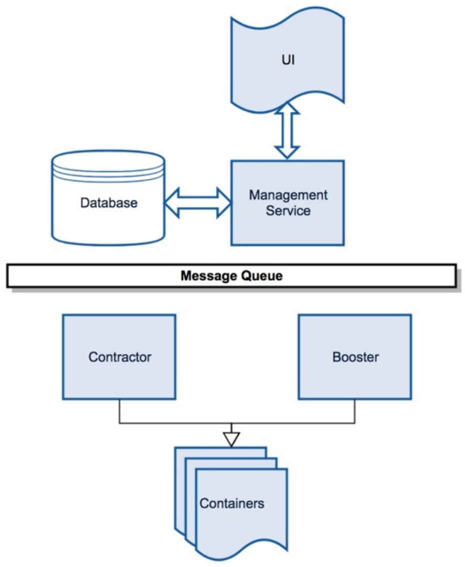


Figure 4.1 General Architecture of MLaaS.

4.1 Management Service

Management Service is a light layer to manage user sessions, job records, and translate user request into proper job start request and send it to message queue.

Public API Summary:

```
GET /api/job
GET /api/job/{id}
POST /api/job
PATCH /api/job/{id}
DELETE /api/job/{id}
POST /api/job/{id}/code
GET /api/job/{id}/code
GET /api/job/{id}/code/{versionId}
PUT /api/job/{id}/code/{versionId}
POST /api/job/{id}/run(stop)
GET /api/job/{id}/node
GET /api/job/{id}/node/{id}
GET /api/job/{id}/node/{id}/log
GET /api/job/{id}/download?file_link=<link>
```

Figure 4.2 Restful API to manage services.

A job object includes code version, data source, execution parameters, environment parameters and results. They are stored in a NoSQL [4] database, Redis [5]. Running status and job logs are real time information. Management Service will sync them from Booster and send to UI periodically through web socket.

4.2 Contractor

Contractor is a scheduling engine. The contractor will schedule Booster and Monitor to assign a specific user job, and dynamically select template generated code to inject into the worker node according to the configuration of the user job. Contractor consists of two parts, one is the job engine, another one is the code engine.

Job engine is a state manager. By monitoring the message queue, Contractor get the information that the user's task configuration and the status of the corresponding Booster. When the contractor get both the user's task configuration and Booster ready response, the job engine will notify the code engine to modify the user's code. After the code engine modifies the code, the job engine will package the user code and deliver it to the Booster, informing Booster about starting the user's task. Job engine will track the user's task status through Booster. When the user's task is completed to some specified state, the job engine will select different operations to affect the task according to different status.

```
while true
  sleep(5)
  worker_exits = db.get("exit-#{job_id}")
  puts "ps server checking all exists:
      #{worker_exits}"
  next if worker_exits.nil?
  break if (worker_exits.to_i ==
      worker_ip_list.length)
end
booster.stop_jobs(job_id)
```

Figure 4.3 Job Engine of handling exit status.

As shown in the figure, when all the user's worker nodes have completed their job, the job engine will ask booster to stop and destroy all nodes.

Code engine is a code modifier. Code engine will choose different ways to modify the user's code to achieve the rational use of Booster's resources based on the user's different configuration information.

```
if job['type'] == 'composed'
  # generate hidden_layer_dim
  hidden_layers = job['hidden_layers']
  hidden_layer_dim = hidden_layers.map {
|hidden_layer| hidden_layer['neurons']}
  # generate activate function
  activation = (hidden_layers.first &&
hidden_layers.first['activation'])
  # generate learning rate
  learning_rate = (hidden_layers.first &&
hidden_layers.first['learning_rate'])
  # generate optimizer name
  optimizer_name = job['output']['type']
  trainer_klass_content = TRAINING_FACTORY.call(
      /^\.\/(.*)\.py$/
          .match(train_input_file)[6],
      hidden_layer_dim,
      activation,
      learning_rate,
      optimizer_name,
      *method_names
  )
```
Figure 4.4 Code Engine of modifying user code.

As shown in the figure, the code engine configures the code factory based on the user's task configuration. The code factory injects the user's code, or splices the user's code into a distributed template.

4.3 Booster

The Booster is an API service to talk to Kubernetes [7] to create and manage containers. It supports:

1. create CPU or GPU based container.

GPU is core to train model in machine learning. We understand GPU resource is expensive and less than CPU ones. If the training workload is not very heavy, user can also consume lots of CPU resources. So, Booster provides user two options. For example, user can choose to launch TensorFlow worker with both CPU or GPU featured container.

2. create container by specified resource.

Some cloud providers, like Amazon Web Service, only provides user with pre-defined resource template. For example, user can request 1G or 2G memory virtual machine but not 1.5G. In Booster, user can input any value of CPU, GPU and memory resource.

3. launch container in different Kubernetes cluster.

Booster can work with Kubernetes in private cloud, public cloud, and on-premise environment.

4. monitor container lifecycle.

There is an informer in Booster to watch the container status, from creation, pending, running, to completed. It can define action rule for any change of the status.

5. get training progress from container in stream.

Booster can get the training progress or log from any container in any time.

6. dynamically talk to different Kubernetes cluster.

If some Kubernetes clusters are up or down, Booster can connect or disconnect to them without any restart. For end user, he will not be aware of any change.

4.4 Code Modification

To clarify the code modification process mentioned in 4.2, this part will focus on details of how to do post-processing on user-uploaded code files to adapt to server runtime and guarantee that code can be correctly executable and monitor-able.

Three modes are supported in MLaaS system. The first one is guiding mode, which is used for users green to machine learning. A code template is provided and several functions are required to be filled such as `define_model`, `setup_train`, `step_train`. Once they are ready, MLaaS has a template engine to insert these functions into a framework following the workflow like below:

```
def main():
   model = define_model()
   env = setup_train(model)
   with tf.Session() as session:
      track(model, session)
      while step_train(model, env)
```
Figure 4.5 Process of user green mode training.

In this mode, `model` and `session` object both are fully manageable. It is convenient to read model graph and track operator node like `loss` and `accuracy`. However, the workflow is invisible to users. It is hard to test model design without MLaaS; in another word, users cannot use what they provided running in their local environment directly.

The second one is annotation mode. Comment annotation is used to set tracking mark. For example, users write complete machine learning code like:

```
def main():
   model = customized_define_mode()
```

```
    env = customized_setup_train(model)
    with tf.Session() as session:
       #@define: model, session
       while customized_step_train(model, env)
```
Figure 4.6 Make annotation and insert code.

Then user can run complete code in his local and the code can be uploaded to MLaaS for further running as well. To make code monitor-able, it is needed to add comment annotation `#@define: model, session` so that MLaaS is able to inject code like `track(model, session)`. In this mode, comment annotation is responsibility of user to tell MLaaS how to track session and model.

Non-intrusive mode is the third one. `Non-intrusive` is for code from user; meanwhile, `Session` class in tensorflow should be patched. There is a wrap for the original `Session` and overwrite `tf.Session`. Once machine learning code initialize a session as `tf.Session()`, it actually creates patched `Session`, which can set up monitor to track training.

Ideally, the third mode would be enough for all kinds of cases. However, we find that there exists tensorflow code does not use `Session`; for example, `DNNClassifier` class directly invoke a customized session of `MonitoredSession`. Currently we cannot guarantee the third mode 100% works; therefore, we implement all of the 3 modes.

5 Discussion

Several advantages and challenges are listed below.

5.1 Build machine learning on SaaS

VMware has run SaaS inside and outside company in many areas but hasn't made an attempt to integrate SaaS with Machine learning yet. Our research showed that, machine learning is definitely one of the domains that would benefit a lot from SaaS model.

- Running Machine Learning on SaaS improves productivity of Machine Learning experts

Comparing to research, running a real business on machine learning requires far more work to handle, e.g. hardware setting up, maintenance and service hosting.

By leveraging the workflow of SaaS, we move these affairs to cloud management experts thus enabling Machine Learning experts to focus on their domain as well as improving the infrastructure stability.

- Running Machine Learning on SaaS reduces Machine learning infrastructure costs

Though requiring large amounts of computing resource on the first training, machine learning workloads go down later according to our tests and experience. That means, by serving a centralized SaaS service, we introduce a time-based resource-sharing across the whole company. At the same time, guarantee each team a larger amount of computing resource on their critical jobs.

5.2 Scheduling system based on Kubernetes

Without Contractor or Booster, users have to manually write yaml file and run command lines to create containers. They have to do this each time and have to standby to monitor the container health status and training progress from time to time.

1. Create containers

With our services, user only needs to choose machine learning framework and computing resources in UI. Contractor and Booster can automatically create containers to meet user's requirement. Without them, user has to manually write lots of yaml files. For example, one for CPU featured Tensorflow, one for GPU featured Tensorflow, one for Keras[8] framework, …etc. Besides, user also needs to test their models before adopting them into production environment. This will cost several hours. But in MLaaS, it only needs couple of seconds.

2. Kubernetes yaml files maintenance

If a user wants to upgrade his Kubernetes yaml files to use latest version of machine learning frameworks, it would takes couples of minutes for him to verify it. With MLaaS, we'll cover this for them. User doesn't need to take care of it.

3. Kubernetes cluster management

In a local machine learning environment, user needs to take care of the configuration files of Kubernetes cluster. If a user wants to launch container in different Kubernetes clusters, he needs to manually run the same commands or write a script to do that. In MLaaS, since Booster can dynamically connect to different Kubernetes clusters, user doesn't need to pay any time for it.

5.3 Key challenges

1. Resource competition.

The computing resource is not infinite. If a job requests more resource than how much the system has now, Contractor and Booster can launch training worker with as many resources as it can consume at that moment. Then it will schedule the job's remaining request into a queue and assign resource to it when other jobs are finished.

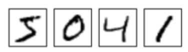
2. High availability

As mentioned before, the Booster can talk to many Kubernetes clusters. If a data center is down, Contractor and Booster can detect it and launch new containers in other Kubernetes clusters to continually work on the training.

6 Performance Analysis

At present, MLaaS platform has multi-CPU distributed computing capabilities. In future, multi-GPU distributed computing will be enabled. In order to further understand the computational relationship between CPU distributed computing and GPU computing, we chose one of the most commonly used machine learning case – MNIST [9].

MNIST is a simple computer vision dataset. It consists of images of handwritten digits like these:

For example,  the labels for the above images are 5, 0, 4, and 1. The computer will read all these pictures and attempt to classify them from one to ten.

Our test environment is as follows:

Using 1 parameter node with a variable number of working nodes.

Unified use of neural network algorithm to calculate the number of iterations 20,000 times and each iteration using 50 data sets for calculation.

The computing devices we chose were:

GPU: Nvidia Geforce GTX 1060

Single Host: Intel (R) Xeon (R) CPU E5-2680 v4 @ 2.40GHz x 4,

CPU distribute: n x CPU Intel (R) Xeon (R) CPU E5-2680 v4 @ 2.40GHz, 2Ghz limitation on each CPU.

Device\ Layers	2	3	4	5	6
GPU	108s(1.57x)	170s(1.74x)	296s(2.4x)	711s(2.88x)	2052s
Single Host	1153s(1.62x)	1870s(2.39x)	4477s(3.02x)	13557s	\
4 CPU	890s(1.88x)	1682s(2.42x)	4080s	\	\
8 CPU	(1.21x) 738s(1.48x)	(1.53x) 1095s(2.74x)	(1.36x) 3011s	\	\
16 CPU	(1.44x) 512s(1.88)	(1.14x) 960s(1.93)	(1.61x) 1860s(3.18)	5920s	\
32 CPU	(1.14x) 451s(2.16)	(0.98x) 975s(2.02)	(1.05x) 1970s(2.31)	(1.30x) 4557s	\

Figure 6.1 Seconds of finishing the training job.

In figure 6.1, we can see that, compared with a single host, GPU has different calculation ratios according to different algorithms. The experimental results show that the ratio ranges from about 10 to 19x between one GPU and a single host, and the greater the complexity of the algorithm, The greater the difference between the computing power. The gap between GPU and single host has reached 19x when the layer number go up to 6 layers.

In addition, we compare the performance of single-host computing and distributed computing. From the experimental results, we can see that the computing power of single-host is close to that of 4x2Ghz multi-CPU distributed computing.

For multi-CPU distributed computing, we can get the CPU number and the calculation of the length of the graph:

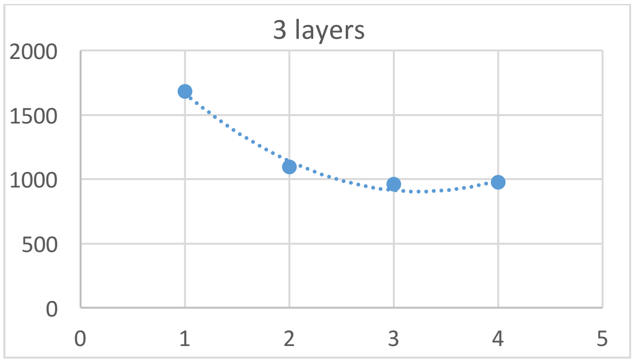


Figure 6.2 Duration-CPUs table, seconds/$2^{n+1}CPU$

In Figure 6.2, we selected three layers of network as the sample. In the experimental environment, the computing power got the highest when the number of CPU worker node is 16. When the number exceeded 16, due to the influence of network loss, the computing performance didn't improve significantly.

Due to the limitation of experiment number and the experiment environment, the data above should only be a reference value. From the above experimental results we can make the following summary:

1. In neural network computing, there is a large performance gap between GPU computing and CPU computing. When the algorithm complexity gets higher, the gap get larger.

2. There is performance improvement when comparing multi-CPU distributed computing with single-host CPU computing. However, when the number of CPUs reaches a certain value, the increase of computing power will be made a negative impact by factors such as network bandwidth or the number of parameter nodes.

Based on the above summary, we can get the following corollary:

1. GPU calculation has irreplaceable advantage compared with CPU calculation.

2. With reference to the experimental results of multiple CPUs, we have reason to believe that multi-GPU computing can really improve the computing power and save time. According to the results of the fitting curve, the initial increase of the number of GPUs can achieve nearly linear performance improvement. However, when the number of GPUs increases to a certain extent, performance bottlenecks will occur. How to divide the GPU into a group of training, or how to optimize after the performance bottleneck will be the problem to be solved.

7 Conclusion

What we provide is a unified solution for running machine learning code on the cloud. By leveraging idle computation resources on Nimbus, we build a platform that allows user to upload machine learning program, specify data source, allocate computation resources as the user needs. Several main advantages prove its value. First, user can have access to more computation resources including GPU on our platform if he/she is going to run a time-consuming program. Second, we provide distributed runtime environment so that user program will be executed faster than he/she does in local environment. Third, we support some auto code completion and various modes of code adaption so that user can build their program with simplicity and flexibility.

8 Future Work

Although the works of MLaaS have achieved fruitful goals, there are still challenges:

1. For incoming train data, integrate with more data flow system such as Spark, Data Warehouse, etc.

2. In this paper, we met performance bottleneck when adding CPU worker nodes to a specific number. How to handle this issue would be a challenge for us.

3. Currently, MLaaS supports TensorFlow and Keras. To make it for widely adoption, we will support more machine learning frameworks and tools in future.

4. We are now integrating our platform with internal teams who needs training service, such as VAC [10] and Ada [11] from EUC.

9 Reference

[1] https://deepmind.com/research/alphago/

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A System for Large-Scale Machine Learning. In The 12th USENIX Symposium on Operating Systems Design and implementation. November, 2016.

[3] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. arXiv:1512.01274v1. 2015.

[4] J. Han, E. Haihong, G. Le, and J. Du. Survey on NoSQL Database. In The 6th International Conference on Pervasive https://confluence.eng.vmware.com/pages/viewpage.action?spaceKey=SKYD&title=Adaer, 2011.

[5] https://redis.io/

[6] M. Daoust. Example of MNIST with Neutral Network. Jun 28, 2017 https://github.com/tensorflow/tensorflow/tree/r1.4/tensorflow/examples/tutorials/mnist

[7] D. Bernstein. Containers and Cloud: From LXC to Docker to Kubernetes. Page: 81 - 84, Volume: 1, Issue: 3, IEEE Cloud Computing. September, 2014.

[8] https://keras.io/

[9] L. Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. In IEEE Signal Processing Magazine. Pages: 141-142, Volume: 29, No. 6, November, 2012

[10] A. Dayanov. VMware Analyze Cloud.

https://wiki.eng.vmware.com/VAC

[11] W. Fan. Anomaly Diagnosis through log Analysis

https://confluence.eng.vmware.com/pages/viewpage.action?spaceKey=SKYD&title=Ada

[12] https://confluence.eng.vmware.com/display/DevToolsDocKB/Nimbus_Manual