FACTORS AFFECTING MACHINE LEARNING ALGORITHMS AND LIBRARIES SELECTION

Boris Zibitsker, PhD, BEZNext;
Dominique Heger, PhD, Data Analytica;

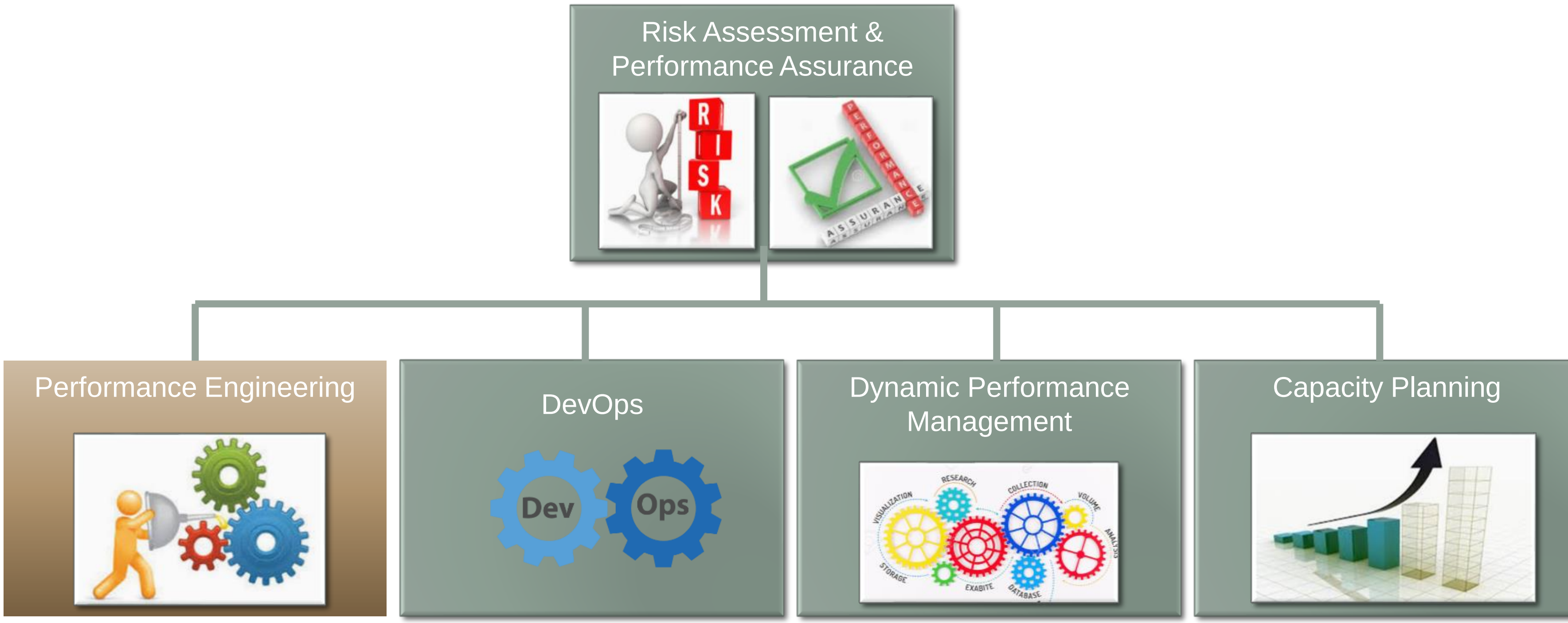
ICPE 2018

Outline

- Introduction
- Challenges
- Objective
- Benchmarking and Modeling
- Collaboration
- Methodology
- Data Collection Results
- Modeling Results
- Recommender
- Summary

Introduction

Performance Engineering and Performance Assurance



Challenges

Business

Data Scientists

ML Algorithm

Application

- Business Requirements to IT: Accuracy, Timeliness, Throughput, Scalability
- Selection of ML Algorithms affects ability of IT meeting Business Goals
- ML Algorithms are "Atomic Components" of IT
- ML Algorithm performance depends on Number of observations / rows, number of columns / predictors, hardware and software configuration

Challenges
Selection of ML Algorithms Affects Business and IT

IT

- Users
  - Customers, Local, Partners, Vendors
- Measurement Data
  - Performance, Resource Usage
  - Accuracy
  - Scalability
  - Cost
- Hardware and Software Systems
  - Big Data Clusters, EDW, Data Center, Cloud
- Workload
- Application
- Data
- **ML Algorithm / ML Library**

Business

- Business Measures
  - P&L, Balance Sheet, etc.
- Business Plan
  - Growth
- Service Level Goals
- Line of Business

Challenges: A lot of Options for selection of Regression Algorithms

Regression Algorithms	Method	Python	
		Python Library	Python Function
Focus on the highlighted MLAs	Ordinary Least Squares Regression	sklearn.linear_model	LinearRegression()
	Lasso (least absolute shrinkage and selection operator)	sklearn.linear_model	linear_model.Lasso()
	Multi task lasso	sklearn.linear_model	linear_model.MultiTaskLasso()
	Elastic Net	sklearn.linear_model	linear_model.ElasticNet()
Ridge Regression		sklearn.linear_model	linear_model.Ridge()
	Ridge	sklearn.linear_model	linear_model.Ridge()
	Lasso	sklearn.linear_model	linear_model.Lasso()
	LassoLars	sklearn.linear_model	linear_model.LassoLars()
	MultiTaskLasso	sklearn.linear_model	linear_model.MultiTaskLasso()
Elastic Net	Elastic Net	sklearn.linear_model	linear_model.ElasticNet()
	MultiTaskElasticNet	sklearn.linear_model	linear_model.MultiTaskElasticNet()
Lars	Lars	sklearn.linear_model	linear_model.Lars()
Bayesian Regression	blr	sklearn.linear_model	
	brm	sklearn.linear_model	
	BayesianRidge	sklearn.linear_model	linear_model.BayesianRidge()
	ARDRegression	sklearn.linear_model	linear_model.ARDRegression()
	many	PyMC3	
	many	bayespy	
kernel regression	kernel Ridge	sklearn.kernel_ridge	kernel_ridge.KernelRidge()
	kernel Ridge	mlpy	kernelRidge()
	nonparametric Kernel Reg	statsmodels	nonparametric.kernel_regression.KernelReg()
	ksmooth	sklearn.linear_model	
SVR (Suppport Vector Regression)	svm	sklearn.svm	svm.SVR(), SVR()
	ksvm	sklearn.linear_model	
SGD (Stochastic Gradient Descent)		sklearn.linear_model	linear_model.SGDRegression()
Gaussion Processs Regression	GaussianProcessRegressor	sklearn.linear_model	gaussian_process.GaussianProcessRegressor()
Regression Tree		sklearn.tree	tree.DecisionTreeRegressor()
Bagging	ensemble.BaggingRegressor	sklearn.linear_model	
	bagging	sklearn.linear_model	
Random Forest	ensemble.RandomForestRegressor	sklearn.linear_model	
	ensemble.ExtraTreesRegressor	sklearn.linear_model	
	randomForest	sklearn.linear_model	
AdaBoosting	ensemble.AdaBoostRegressor	sklearn.linear_model	
Gradient Boosted Regression Trees	ensemble.GradientBoostingRegressor	sklearn.linear_model	
	gbm	sklearn.linear_model	
Neural Network	neural_network.MLPRegressor	sklearn.linear_model	neural_network.MLPRegressor()
K Neariest Neighbours	KNeighborsRegressor	sklearn.linear_model	neighbors.KNeighborsRegressor()
	Least Angle regression	sklearn.linear_model	
	Orthogonal matching Pursuit	sklearn.linear_model	
	Automatic Relevance Determination (ARD)	sklearn.linear_model	
	Passive Aggressive Algorithms	sklearn.linear_model	
	Robustness regression	sklearn.linear_model	
	Perception	sklearn.linear_model	
	Polynomial regression	sklearn.preprocessing	
	Stepwise Linear regression	sklearn	
	Survival Regression	sklearn.linear_model	
	Isotonic Regression	sklearn.linear_model	

Classification Algorithms	Algorithm	Python	
		Library	Function
Linear Classifier	Binary Logistic Regression	sklearn.linear_model	LogisticRegression()
	Multinomial Logistic Regression	sklearn.linear_model	LogisticRegression()
	Ordinal Logistic Regression	N/A	N/A
	Fisher's Linear Discriminant Analysis	sklearn.discriminant_analysis	LinearDiscriminantAnalysis()
	Naive Bayes classifier	sklearn.naive_bayes	MultinomialNB(), BernoulliNB(), GaussianNB()
Quadratic Classifier	Quadratic Discriminant Analysis	sklearn.discriminant_analysis	QuadraticDiscriminantAnalysis ()
Decision Trees	Classification Tree	sklearn.tree	DecisionTreeClassifier()
Ensemble Model	Bagging	sklearn.ensemble	BaggingClassifier()
	Random Forest	sklearn.ensemble	RandomForestClassifier()
	Gradient Boosting	sklearn.ensemble	GradientBoostingClassifier()
	AdaBoosting	sklearn.ensemble	AdaBoostClassifier()
Support Vector Machines	-	sklearn.svm	SVC(), LinearSVC(), NuSVC()
Neural Network	-	sklearn.neural_network	MLPClassifier()
Stochastic Gradient Descent		sklearn.linear_model	SGDClassifier()
Nearest Neighbors	KNeighbors	sklearn.neighbors	KNeighborsClassifier()
	RadiusNeighbors	sklearn.neighbors	RadiusNeighborsClassifier()
Gaussian Processes		sklearn.gaussian_process	GaussianProcessClassifier()

Challenges:
A lot of Options for selection of
Classification Algorithms

Clustering Algorithms	Algorithm	Python	
		Library	**Function**
Centroid-based	K-means clustering	sklearn.cluster	KMeans(), MiniBatchKMeans()
Distributed-based	Gaussian mixture models	sklearn.mixture	GaussianMixture()
	LDA	gensim.models	LdaModel()
Connectivity-based	Hierarchical clustering	sklearn.cluster	AgglomerativeClustering()
Density-based	DBSCAN	sklearn.cluster	DBSCAN()
	Birch	sklearn.cluster	Birch()
	Spectual clustering	sklearn.cluster	SpectralClustering()

Challenges:
Several Options for Selection
of the Clustering Algorithms

Challenges

How to choose Right Machine Learning Algorithm and ML Library

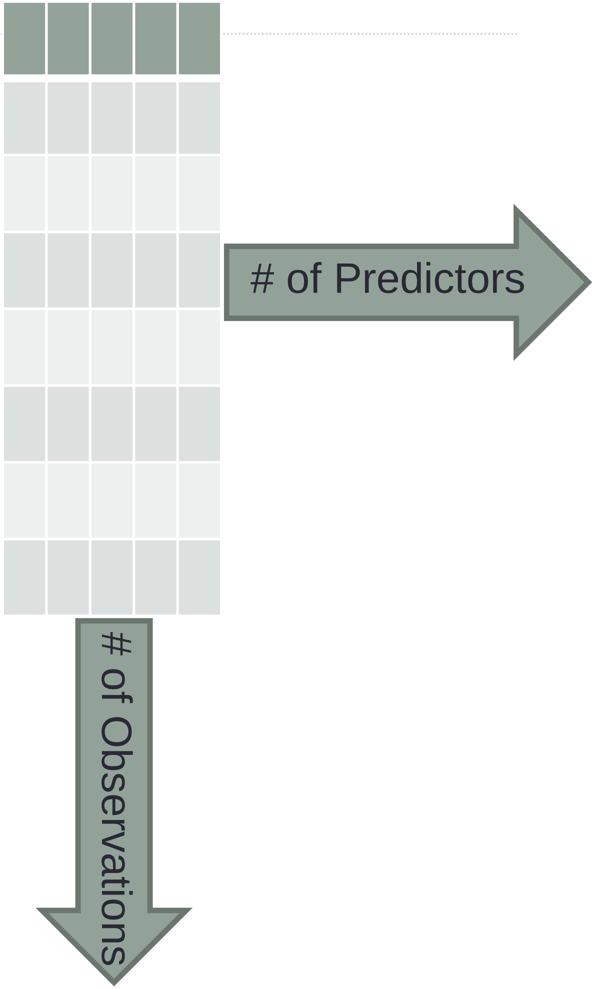
How to choose Right Hardware and Software configuration

Objectives of our project

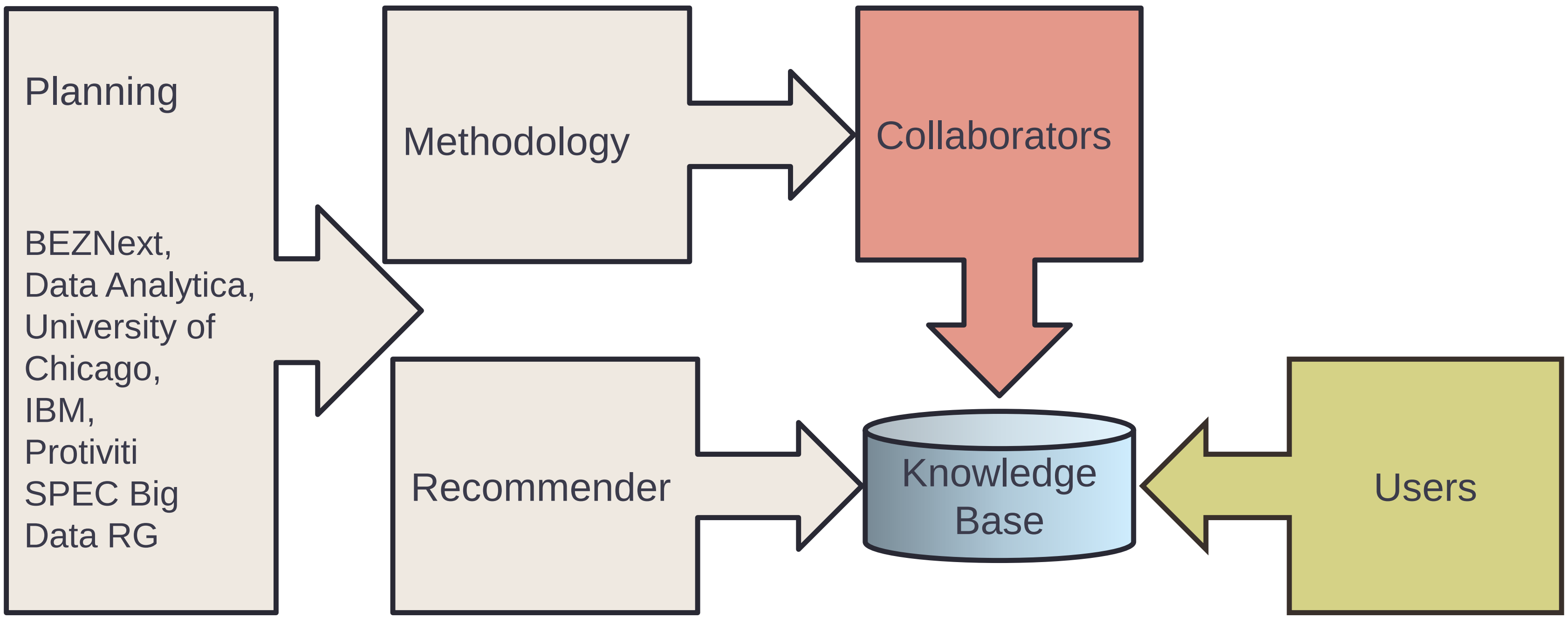
- Develop Methodology
  - How to select appropriate ML algorithm and ML library providing sufficient accuracy, response time, scalability, minimize usage of resources and cost
  - Select Data Set
  - How to collect performance measurement data
- Use Models
  - Develop Models expanding measurement results
  - Use Models to compare benchmarks results done in different environment
- Organize Collaboration
  - Organize a Collaboration to benchmark different algorithms in parallel
  - Use common methodology to benchmark ML algorithms and libraries
- Develop Recommender for Data Scientists and Application Developers
  - Create knowledge base and web application
  - Develop algorithm selecting appropriate ML algorithm and ML library based on business requirements
  - Determine minimum data set size to achieve desired level of accuracy and time of model training

Role of Benchmarks and Modeling

- Evaluate performance, usage of resources, scalability and accuracy of ML algorithms and libraries
- Analyze the impact of the size of the data set on time of the models training
- Benchmarking process includes the following steps:
  - Preparing the Data Sets with different numbers of observations and predictors
  - Preparing and running the Benchmark test:
    - Writing Python programs
    - Creating the benchmark environment
  - Collecting measurement data: response time, accuracy, CPU, memory usage, I/O rate and network utilization
- Modeling:
  - Building models to predict performance characteristics of the different ML algorithms and ML Libraries for different sizes of the data sets not included into the benchmarks
- Model validating:
  - Comparing the prediction results with actual measurement data for data set with different number of observations and predictors.



Collaborative Approach

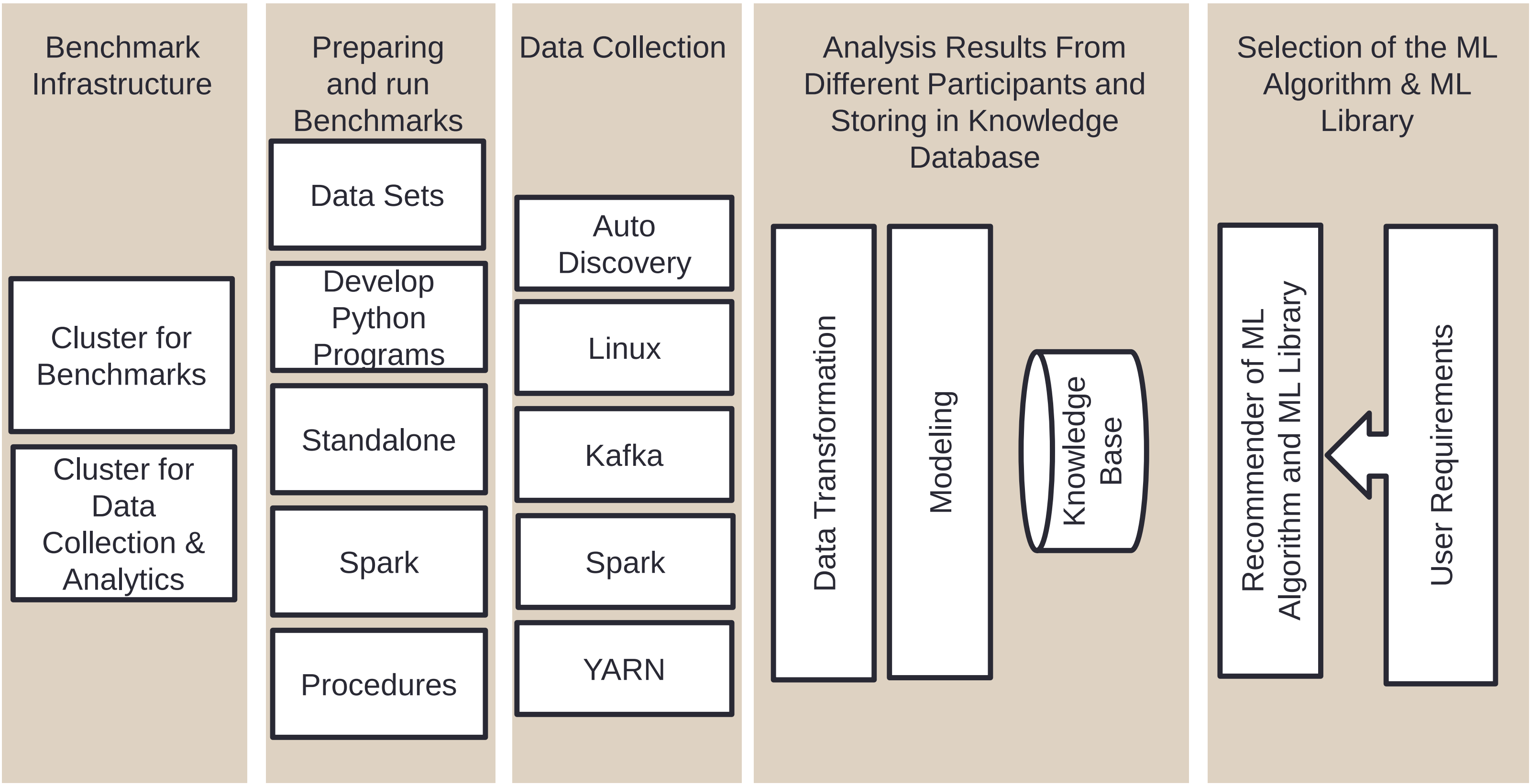


Example of Business Requirements

Type	Relative Weight
Accuracy	0.2
Response Time	0.4
CPU Utilization	0.2
Memory Utilization	0.2
Total	1

Relative importance of the different requirements to new application

Benchmarking Process



Infrastructure
Spark Cluster with 4 Data Nodes Provided by IBM

Host	uchicago-hadoop-host-01.bigdatauniversity.com	uchicago-hadoop-host-02.bigdatauniversity.com	uchicago-hadoop-host-03.bigdatauniversity.com	uchicago-hadoop-host-05.bigdatauniversity.com
IP	10.114.57.125	10.114.57.122	10.114.57.115	10.114.57.117
Linux	Ubuntu 16.04.2 LTS (4.4.0)	Ubuntu 16.04.2 LTS (4.4.0)	Ubuntu 16.04.2 LTS (4.4.0)	Ubuntu 16.04.2 LTS (4.4.0)
CPU	8x2.6 GHz	8x2.6 GHz	8x2.6 GHz	8x2.6 GHz
RAM	15.7 GB	15.7 GB	15.7 GB	15.7 GB
Space	4TB	4TB	4TB	
Services	datanode-3 : 10.42.82.202 yarn-nodemanager-3 : 10.42.12.72 spark-worker-2: 10.42.205.153	datanode-2 : 10.42.47.208 yarn-nodemanager-4 : 10.42.175.132 spark-worker-3 : 10.42.36.4 jupiter-1 : 10.42.204.60 zeppelin-1 : 10.42.255.224	datanode-1 : 10.42.121.65 yarn-nodemanager-1 : 10.42.59.195 spark-worker-1 : 10.42.13.255 hue-1 : 10.42.73.11	datanode-4: 10.42.219.248 yarn-nodemanager-2 : 10.42.18.233 spark-worker-4: 10.42.194.67

Infrastructure

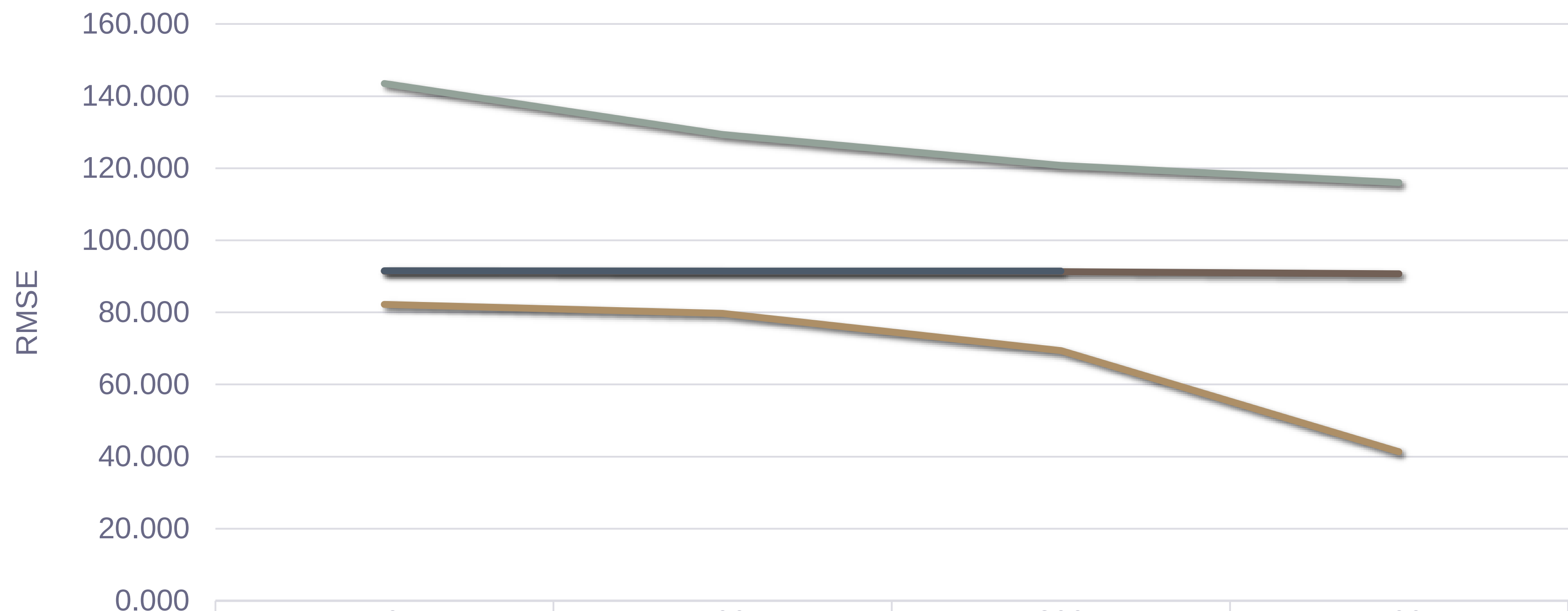
Spark Cluster with Four Nodes are used for Data Collection and Management

Host	uchicago-misc-host-01.bigdatauniversity.com	uchicago-hadoop-host-04.bigdatauniversity.com	uchicago-spark-host-01.bigdatauniversity.com	uchicago-spark-host-02.bigdatauniversity.com
IP	10.114.57.118	10.114.57.116	10.114.57.126	10.114.57.114
Linux	Ubuntu 16.04.2 LTS (4.4.0)	Ubuntu 16.04.2 LTS (4.4.0)	Ubuntu 16.04.2 LTS (4.4.0)	Ubuntu 16.04.2 LTS (4.4.0)
CPU	4x2.6 GHz	8x2.6 GHz	4x2.6 GHz	4x2.6 GHz
RAM	7.83 GB	15.7 GB	7.83 GB	7.83 GB
Space	4TB	4TB	4TB	4TB
Services	HDFS Client 1 Spark client BEZVision BEZVision Repository Oracle Analytics WebApp WebApp Repository MariaDB Analytics Python scripts Benchmarks scripts	HDFS namenode-primary-1 : 10.42.150.176 **yarn-resourcemanager-1** : 10.42.9.96 Spark jobhistory-server-1 : 10.42.85.19 **spark-master-1 :** 10.42.171.182 Kafka broker-1 : 10.42.114.64 Zookeeper zk-1 : 10.42.52.9	spark-master-1 : 10.42.188.22 spark-worker-2 : 10.42.107.244 datanode-2 : 10.42.88.226 namenode-primary-1 : 10.42.245.223 yarn-resourcemanager-1 : 10.42.82.156 jobhistory-server-1 : 10.42.170.253	spark-worker-1 : 10.42.242.8 datanode-1 : 10.42.204.132 yarn-resourcemanager-1 : 10.42.152.78 BEZ Agent Manager Linux agents, YARN and Spark agents UDT/YLT/BVT

Data Collection Process

- Benchmarking Ordinary Least Squares Regression (OLS), Ridge Regression (Ridge) and Random Forest (RF)
  - Standalone Python and Python on Spark
  - Each algorithm was tested for 3 dataset sizes of 5k, 50k and 1m observations and for 4 different number of predictors
- Measurement data include
  - Accuracy, Response time, CPU utilization, Memory usage, I/O rate and Network throughput

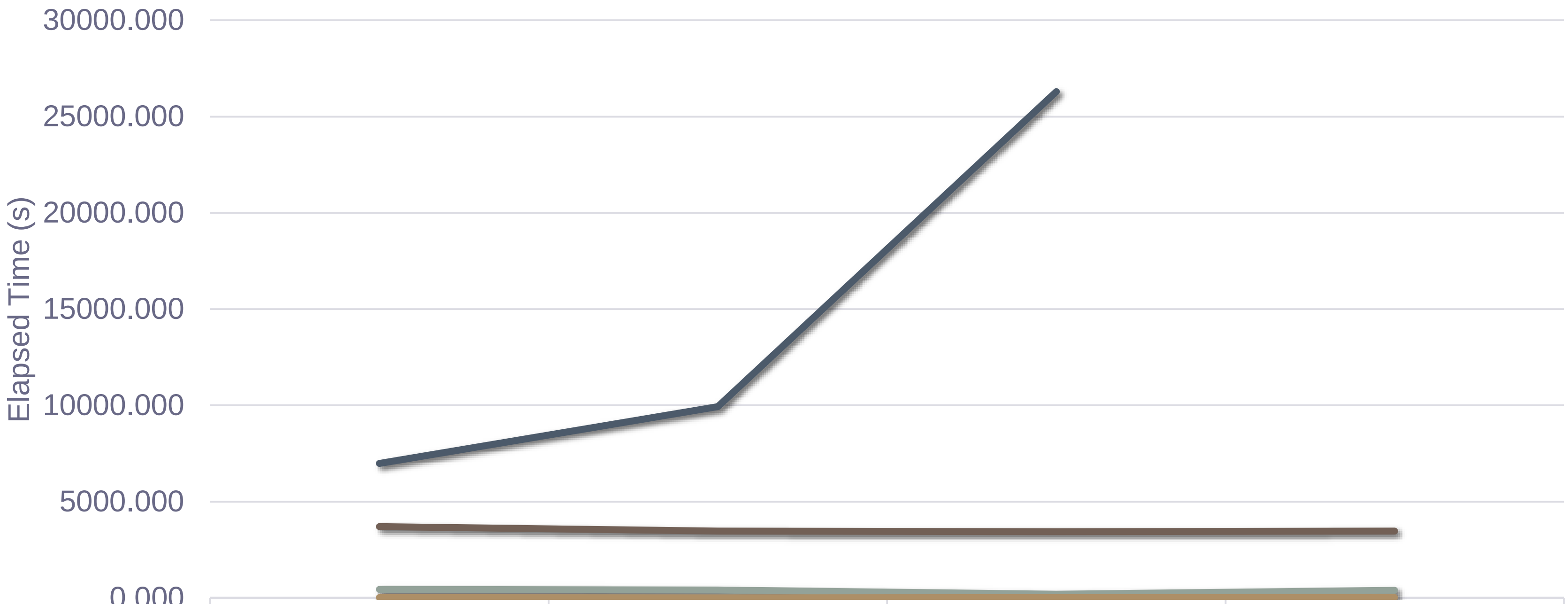
Accuracy RMSE vs # Predictors



	50	100	200	400
OLS 1M	143.489	129.341	120.801	116.012
OLS 50K	82.248	79.751	69.382	41.351
Ridge 1M	91.471	91.359	91.281	90.702
RF 1M	91.542	91.495	91.502	

Predictors

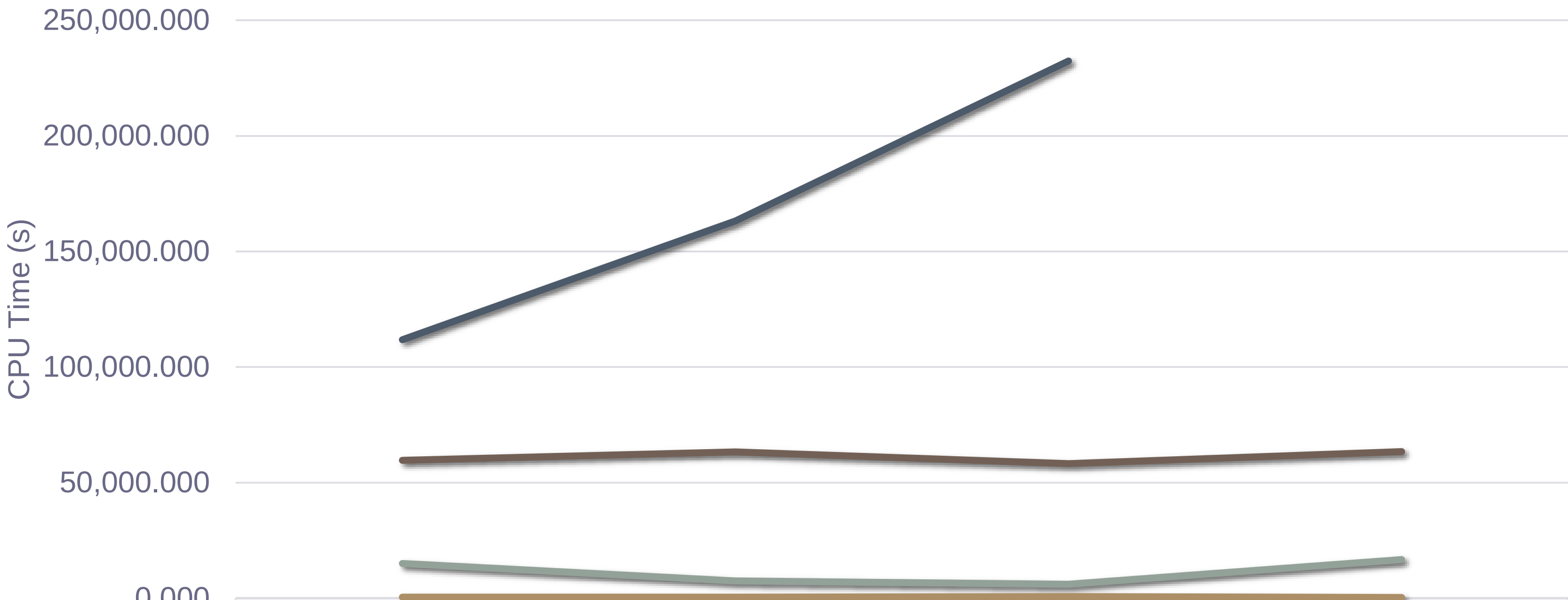
Elapsed Time vs. #Predictors



	50	100	200	400
OLS 1M	439.270	414.397	197.005	393.006
OLS 50K	22.442	21.230	23.502	22.446
Ridge 1M	3,703.687	3,475.132	3,446.311	3,467.036
RF 1M	6,984.629	9,939.694	26,305.126	

Predictors

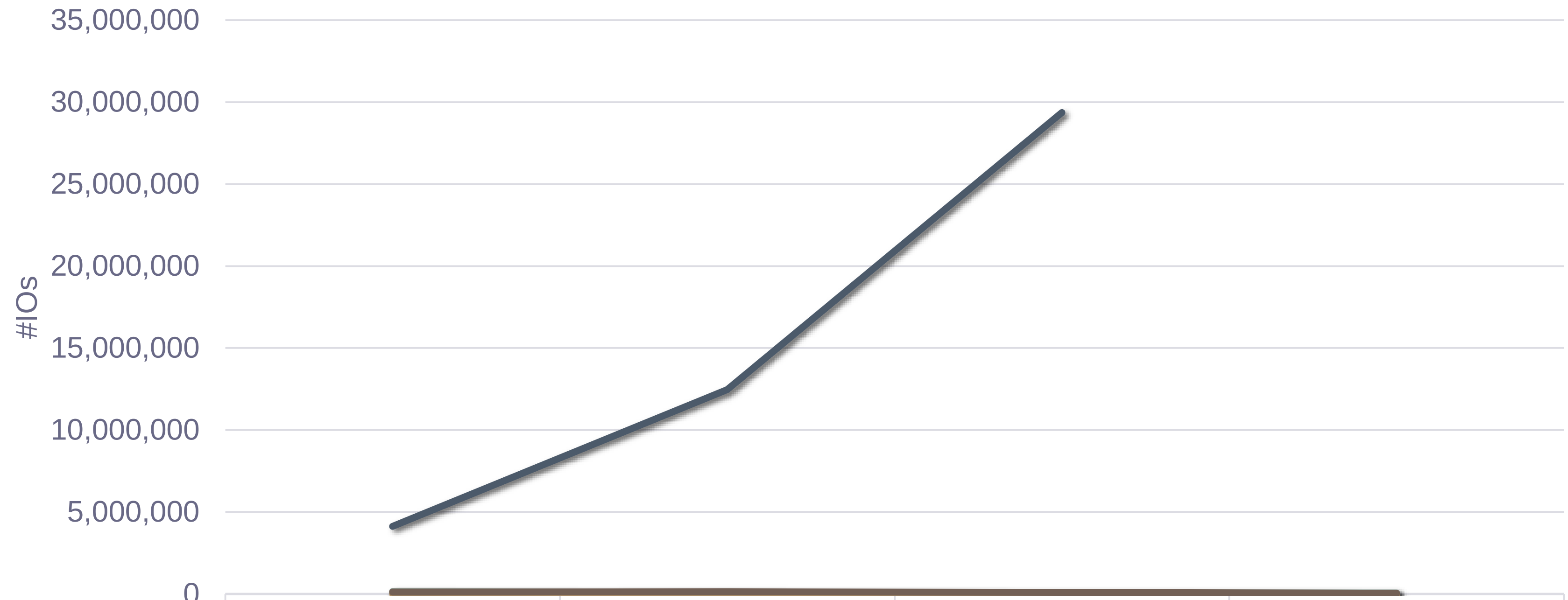
CPU Time Across All Nodes vs # Predictors



	50	100	200	400
OLS 1M	15,051.570	7,456.370	6,102.760	16,714.120
OLS 50K	435.400	440.160	732.820	261.180
Ridge 1M	59,635.940	63,311.340	58,236.210	63,462.430
RF 1M	111,756.520	163,183.570	232,357.130	

\# Predictors

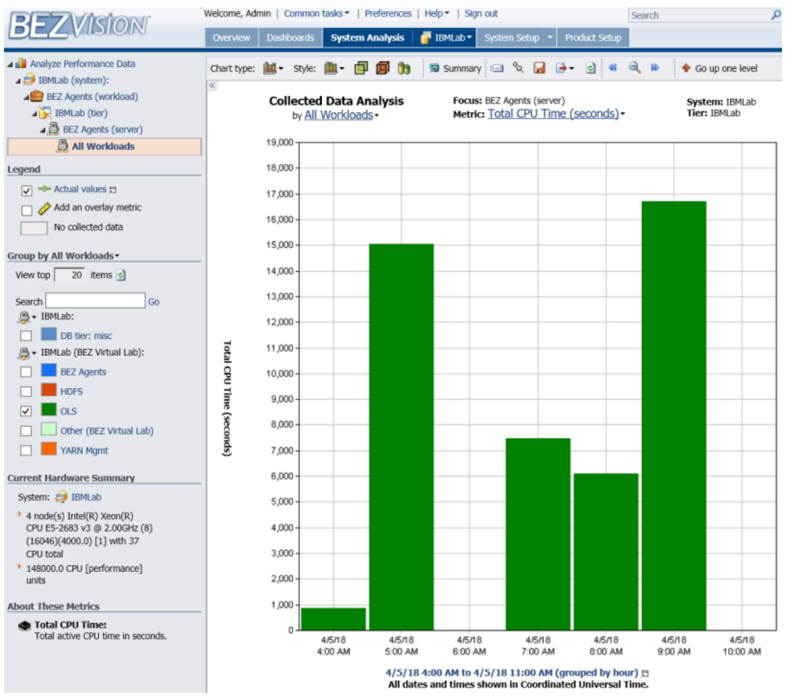
Total I/O across All Nodes vs #Predictors

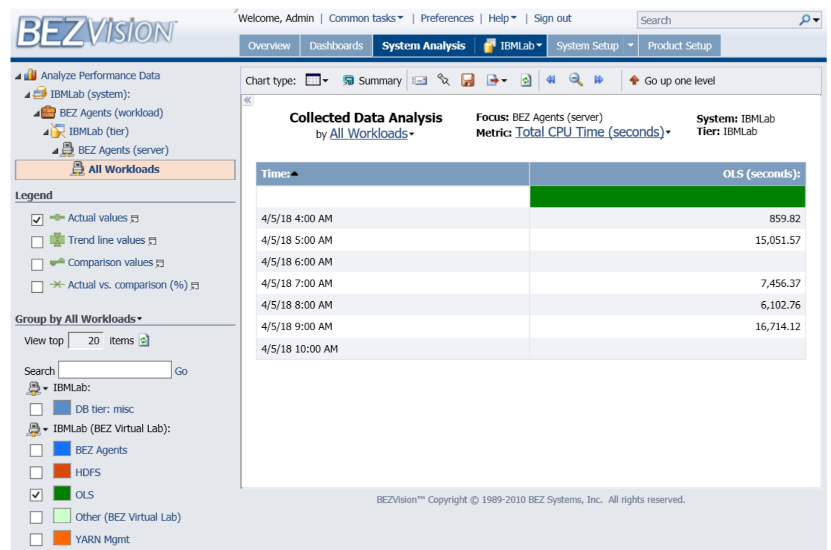


	50	100	200	400
OLS 1M	157,820	75,032	58,752	55,969
OLS 50K	307	372	1,018	96
Ridge 1M	109,957	132,909	107,311	50,803
RF 1M	4,121,545	12,466,652	29,362,640	

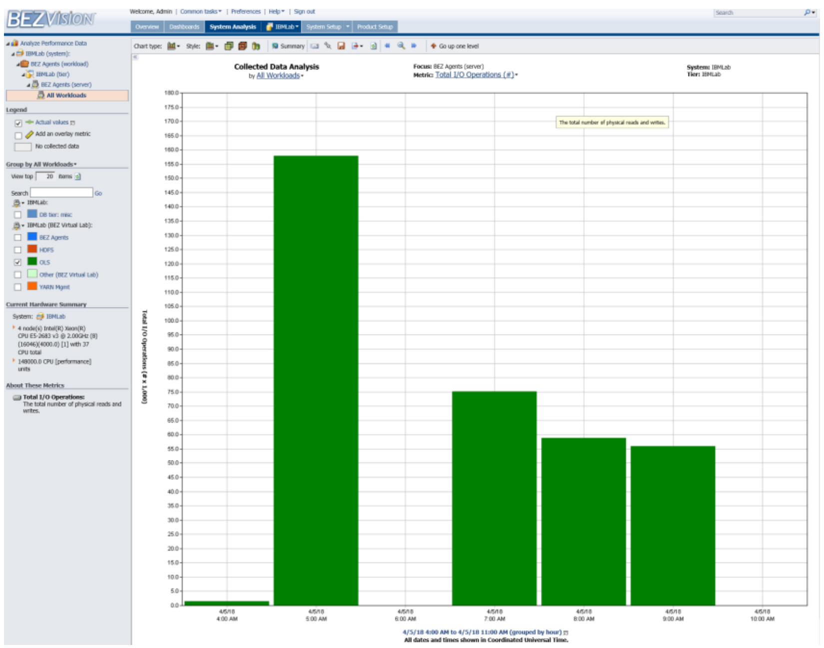
Predictors

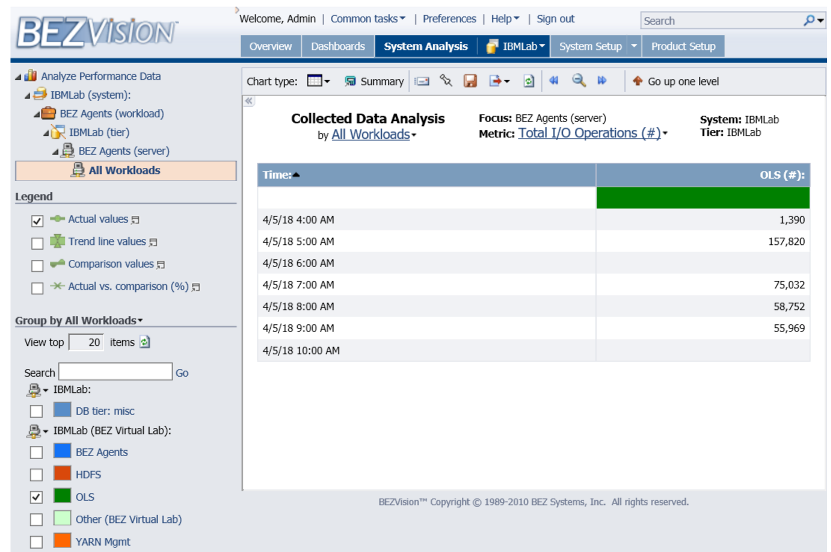
Total CPU time used for OLS 1M rows and 50, 100, 200, 400 predictors starting at 5, 7, 8 and 9 AM





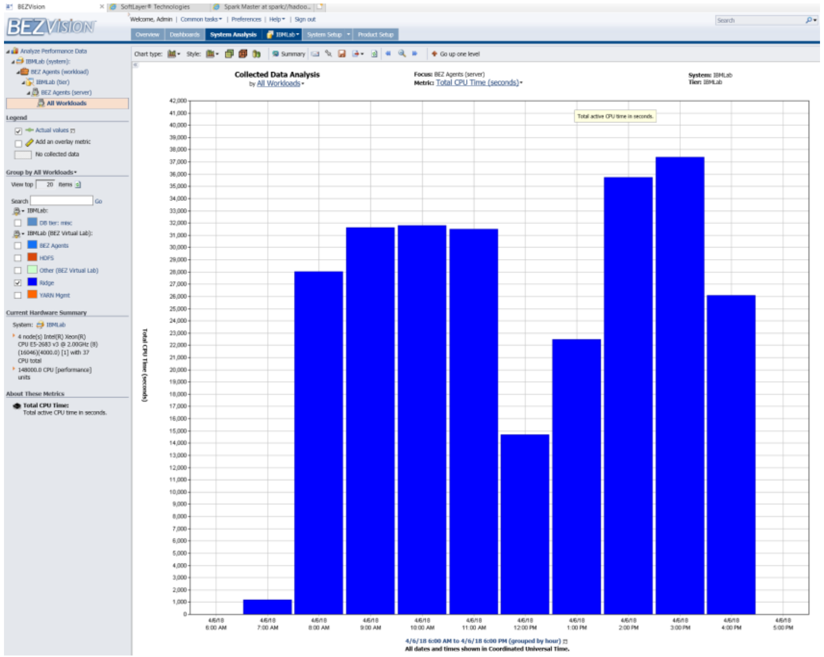
Total IO count for OLS 1M rows and 50, 100, 200, 400 predictors for Benchmark starting at 5, 7, 8, and 9 AM

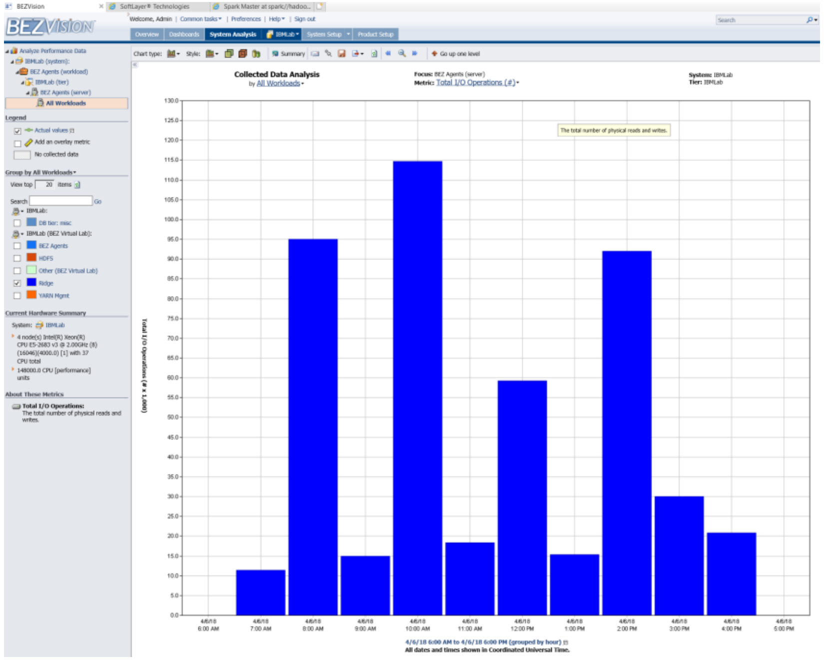




Total CPU time and IO count: Ridge 1M rows and 500 predictors

Start at 8, 10 AM, 12PM 2PM with 2 hours each run with most of I/O happens during the first minute

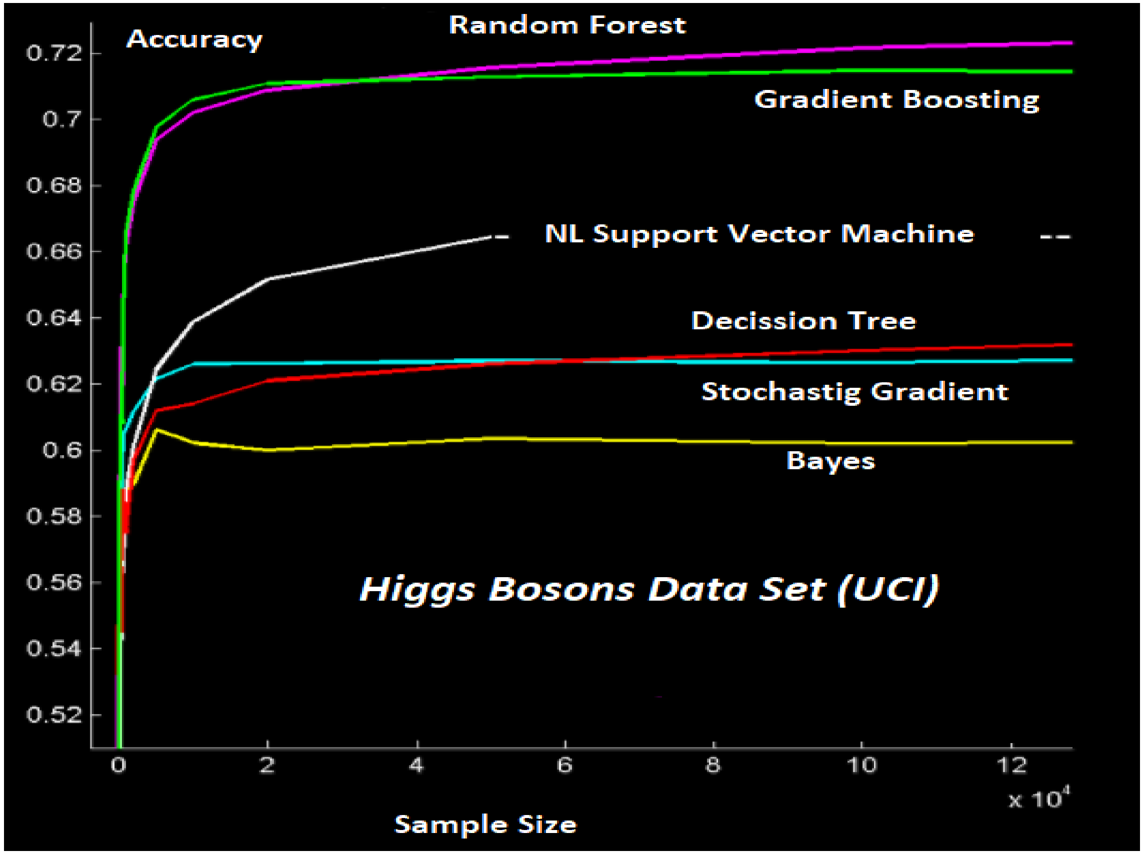




Surprises

- Spark response time exceeds Standalone Python
- CPU Utilization and #I/Os
- Memory utilization
- ML algorithms accuracy depends on training Data Set size
- Partitioning
- Size of the data size
- Machine Learning Libraries Implementation
- Docking containers

ML algorithms accuracy depends on training Data Set size

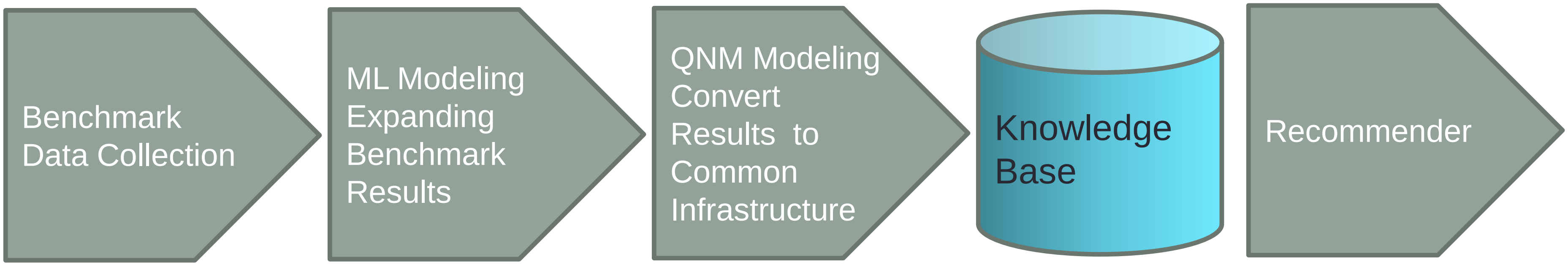


- Higgs Boson Data Set (Cern Benchmark) with 11m rows and 28 predictors https://archive.ics.uci.edu/ml/datasets/HIGGS
- Results of testing ML algorithms on various training data set sizes.
- Training data size affects training and running time.

Modeling Expands the Results of Benchmarks

- Preparation and conducting the benchmark test is time consuming. In our case each benchmark test run for 2 hours. Therefore, the benchmark results include the limited number of data points.
- In order to expand the results of the benchmark tests we applied ML and Queueing Network Models.
- For ML based models the measurement data were split into two data sets: 80% of data were used for model training and 20% of data were used to compare the actual measurement data with prediction results. Trained models are used to predict Response Time, CPU Utilization, I/O rate, Memory Utilization, and Accuracy for each ML Algorithm and ML Library

Role of Models



- Benchmarks are done by collaborators in parallel;
- ML models are used to expand benchmark results;
- Queueing network models (QNM) are used to convert measurement data collected by different collaborators on different clusters into baseline configuration
- QNM are used to justify capacity management measures

Modeling Expands Results of the Benchmark Tests

- Number of benchmarks tests is limited
- ML models can be used to fill gaps and enable evaluation of scenarios where tests were not performed
- Different ML algorithms are evaluated to predict the metrics and estimate the accuracy by comparing prediction results with measurement data
- Measurement data were split into two data sets:
  - 80% of data were used for model training and
  - 20% of data were used to compare the actual measurement data with prediction results.
  - Trained models are used to predict Response Time, CPU Utilization, I/O rate, Memory Utilization, and Accuracy for each ML Algorithm and ML Library.

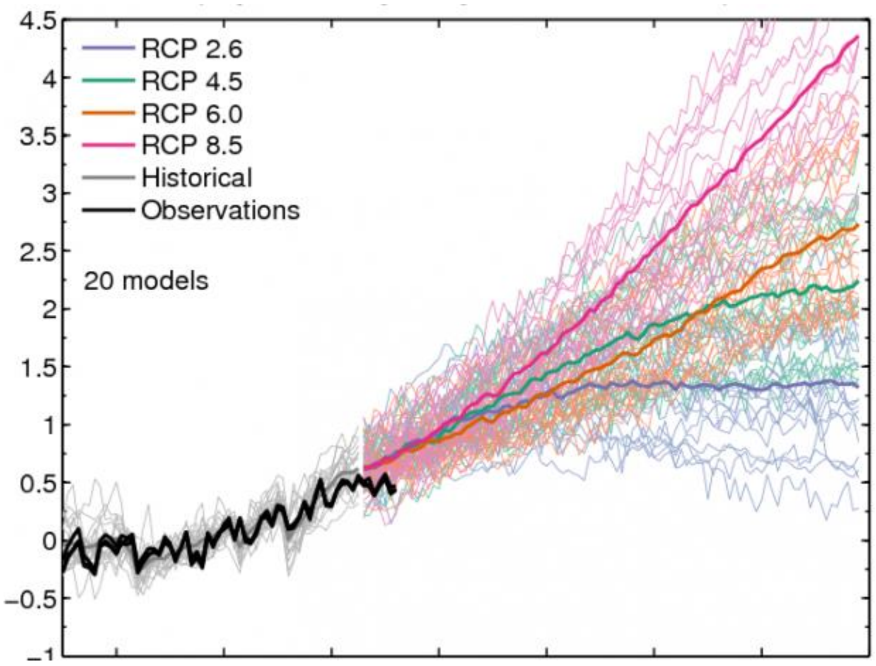
Prediction Models

Accuracy = f(Algorithm, Library, # of Observations, log(log(# of Features));

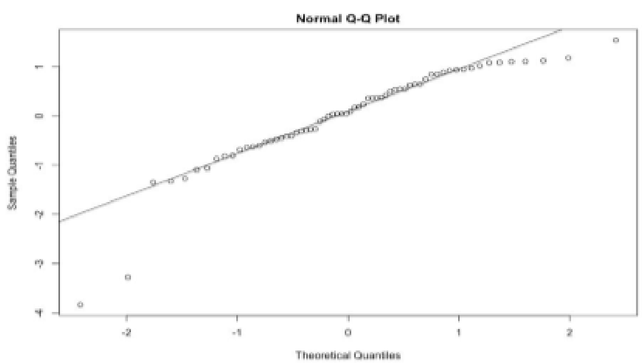
CPU Utilization= f(Algorithm, Library, # of Observations, # of Features);

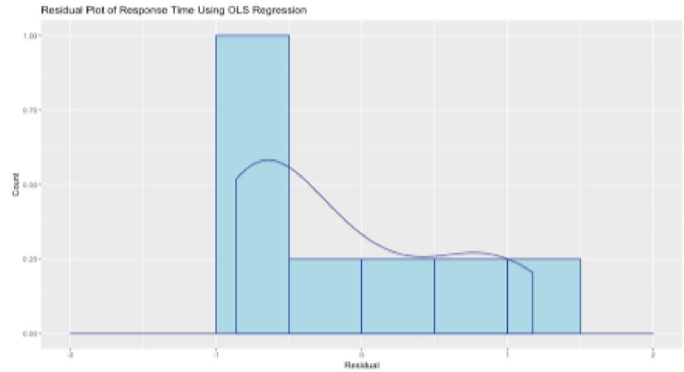
log(Memory Usage) = f(Algorithm, Library, # of Observations, # of Features);

log(Response Time) = f(Algorithm, Library, # of Observations, # of Features);



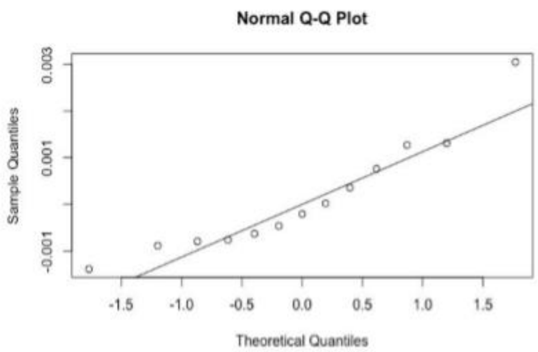
Ordinary Least Squares Regression outperforms Random Forest and Regression Tree Models in Predicting Response Time

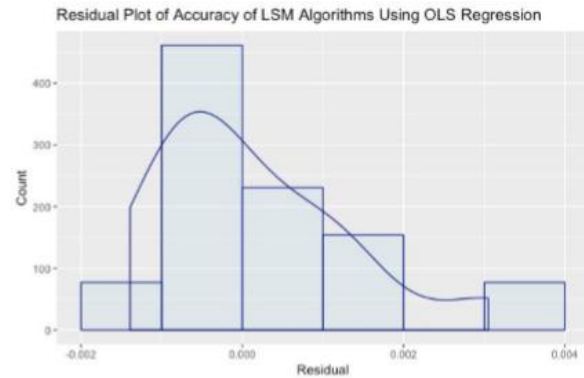


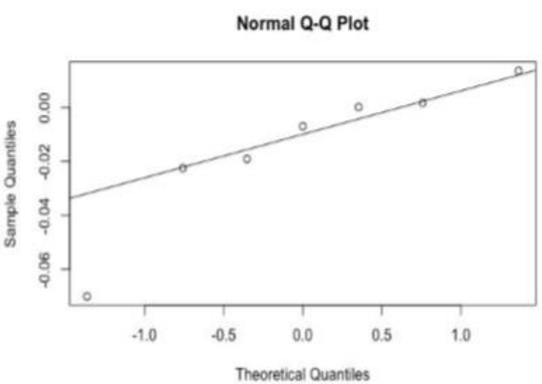


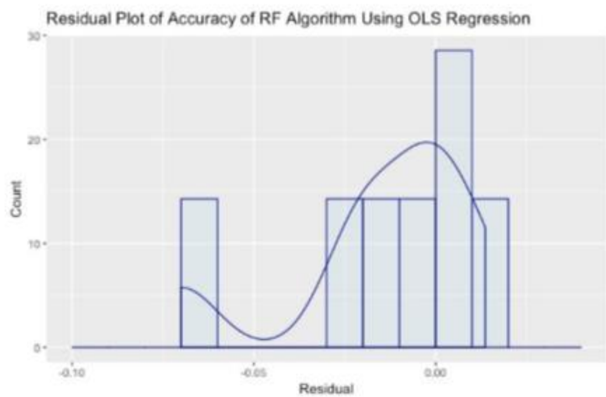
Model	Train R2	Test R2	Final Model
OLS	89%	91%	Yes
RF	70%	96%	No
Tree	80%	88%	No

Ordinary Least Squares Regression Models Provide highest Accuracy for LSM and RF Algorithms



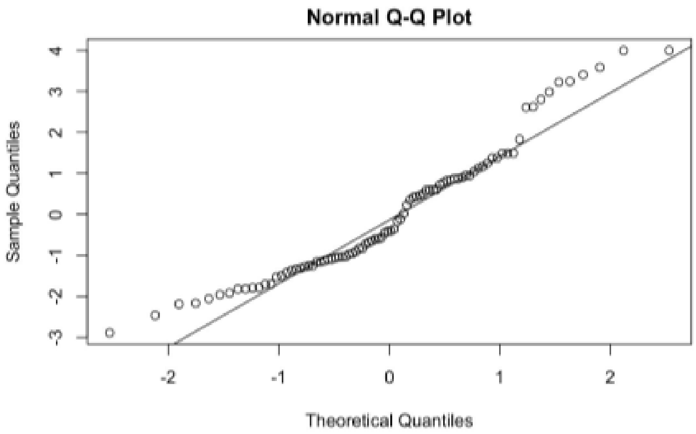


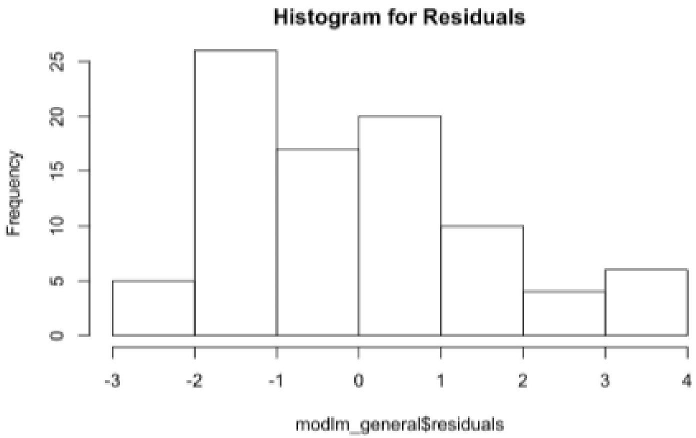




Algorithms	Model	Train R2	Test R2	Final Model
LSM	OLS	87%	84%	Yes
	RF	83%	95%	No
RF	OLS	99%	99%	Yes
	RF	48%	89%	No

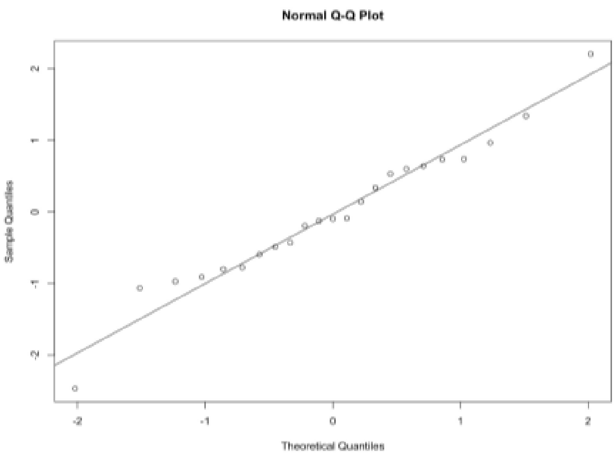
RF Model to Predict CPU Utilization

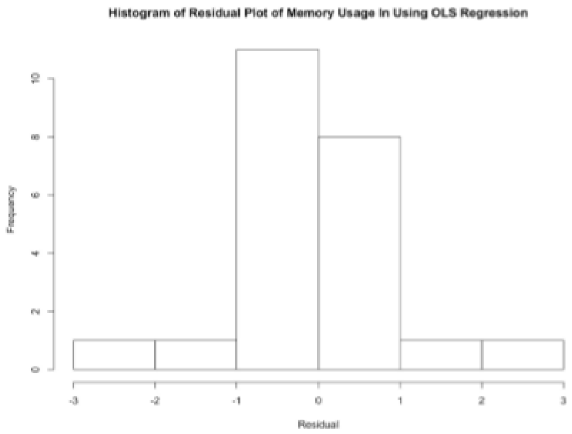




Regression Tree model provides higher accuracy of CPU Utilization prediction (the difference between train and test results is smaller).

Regression Tree Models provide better accuracy in Memory Usage prediction





Model	Train R2	Test R2	Final Model
OLS	78%	87%	No
Tree	87%	81%	Yes
RF	60%	62%	No

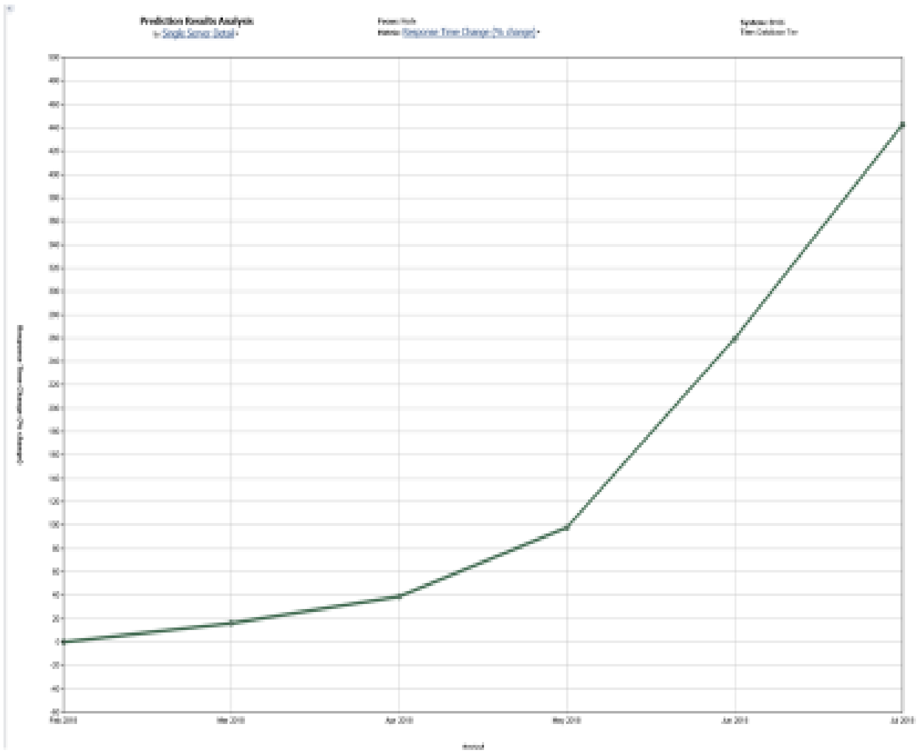
Regression Tree model provides better accuracy of Memory Usage prediction

Comparing algorithms:

Ordinary Least Squared provides better results in prediction of Accuracy and Response Time. Regression Tree algorithm provides better results in predicting CPU and Memory utilization.

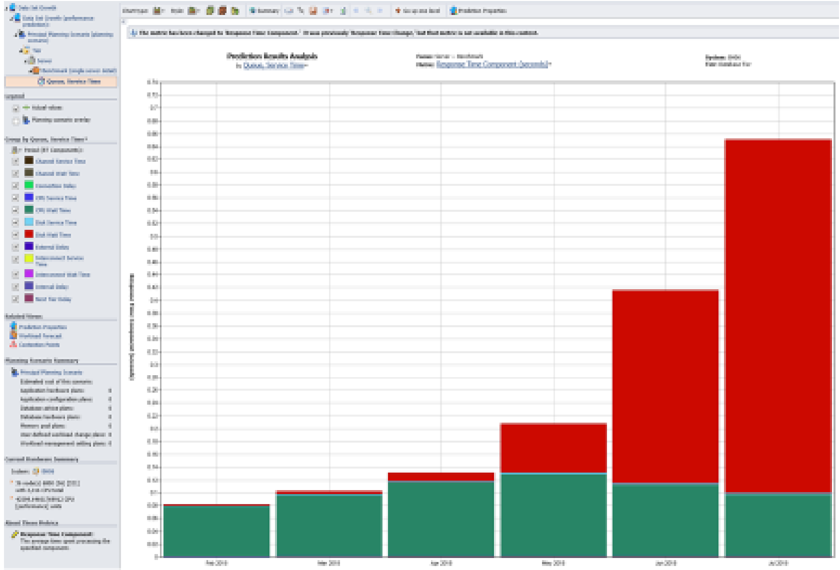
Measure	Algorithms	Model	Train R2	Test R2
Accuracy	LSM	OLS	87%	84%
	RF	OLS	99%	99%
Response Time	LSM	OLS	89%	91%
	RF	OLS	73%	85%
Memory Usage	LSM & RF	Tree	87%	81%
CPU Utilization	LSM & RF	Tree	97%	96%

Predicted impact of the #Observations increase by 25% each period on OLS elapsed time

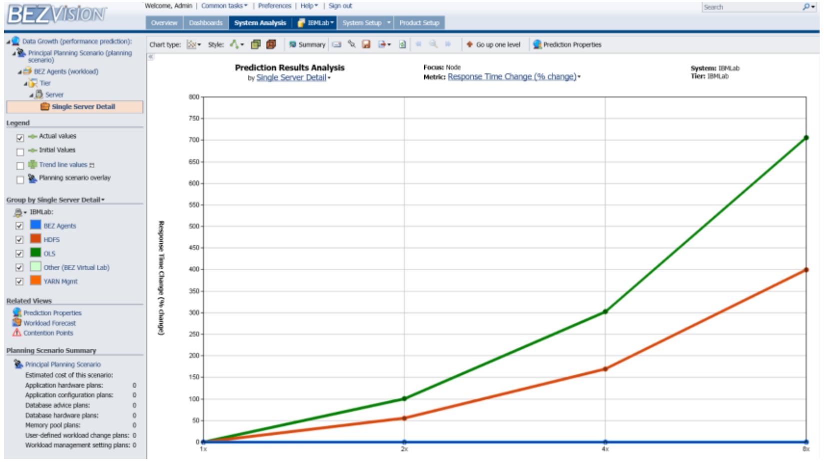


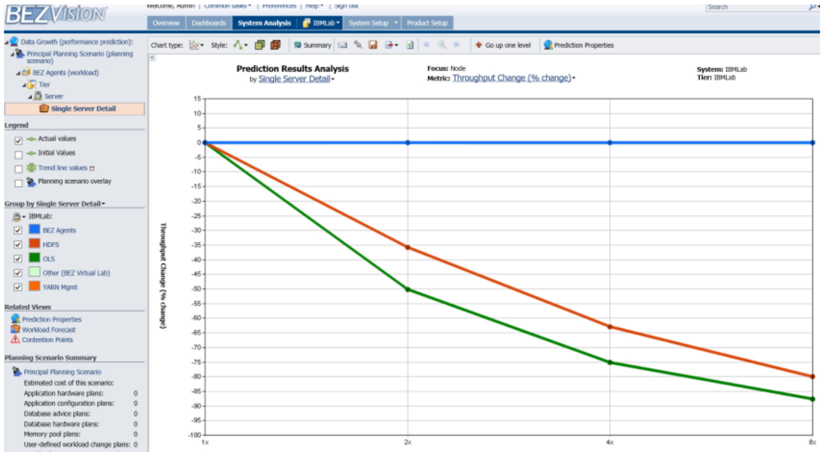
Waiting for I/O will become a bottleneck for OLS

Predicted Elapsed Time Components for Data Set Size Growth 25% per step



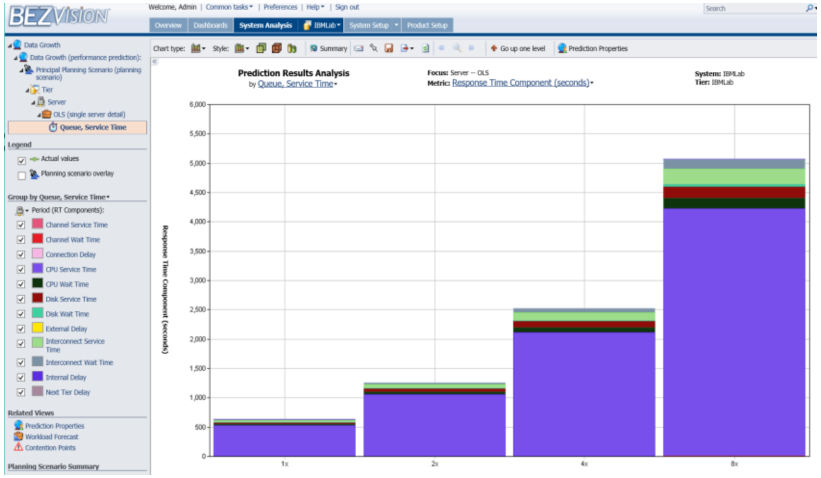
Predicted Elapsed Time and Throughput Change (%) for 50, 100, 200 and 400 Predictors



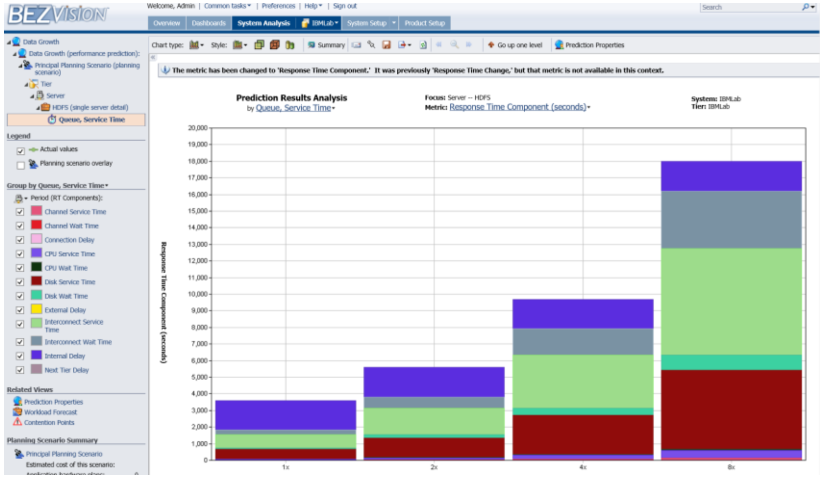


Predicted Elapsed Time Components OLS and HDFS for for 50, 100, 200 and 400 Predictors

OLS

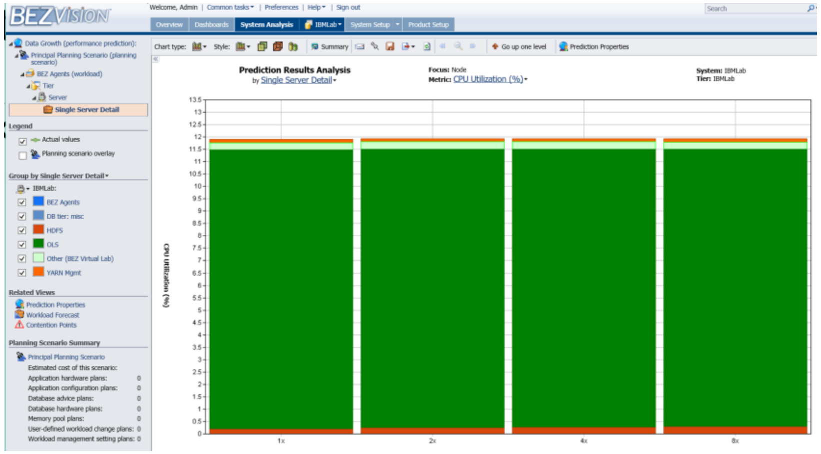


HDFS

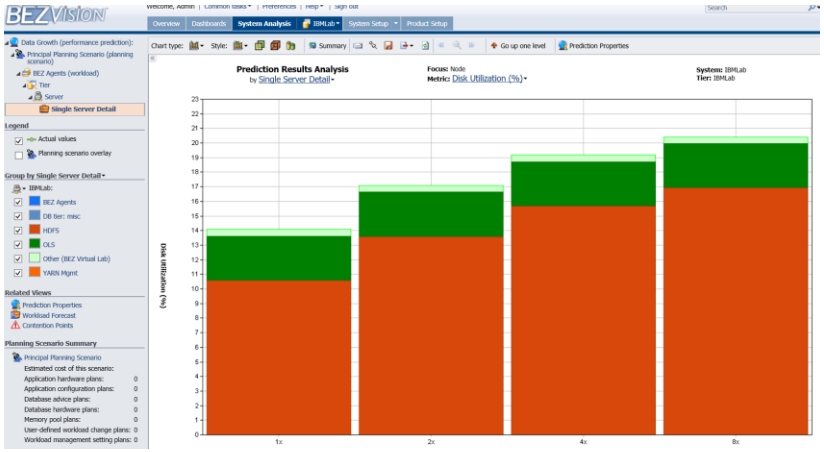


Predicted Resource Utilization for OLS and HDFS for 50, 100, 200 and 400 Predictors

CPU

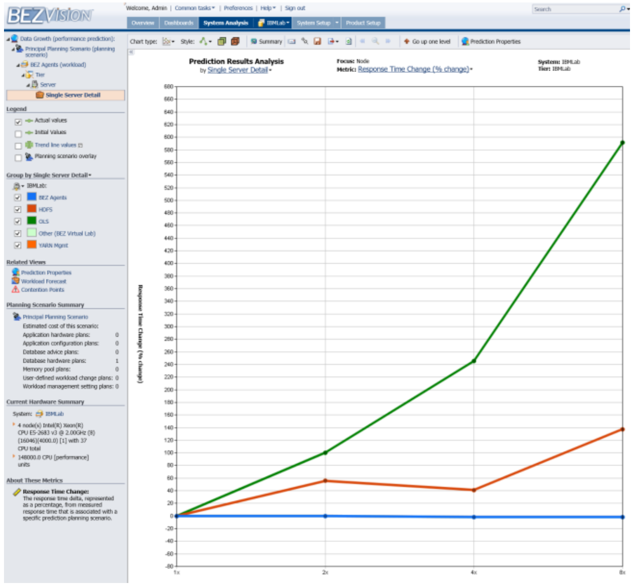


Disk

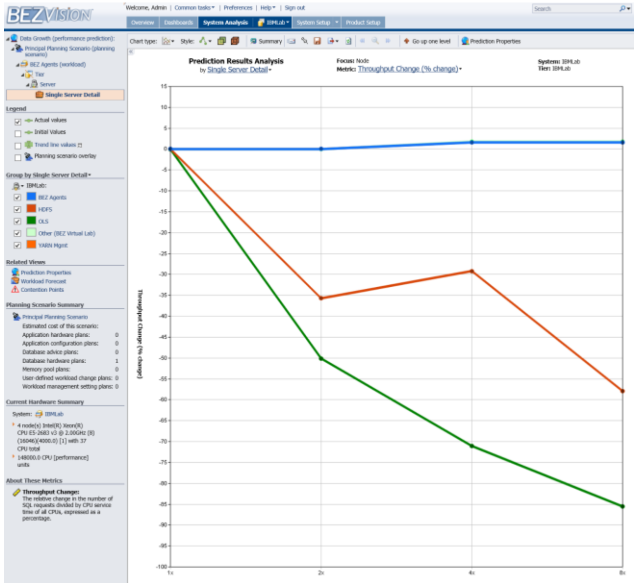


Predicted Impact of Increase Number of Nodes on OLS and HDFS Workloads Elapsed Time and Throughput

Elapsed Time relative change

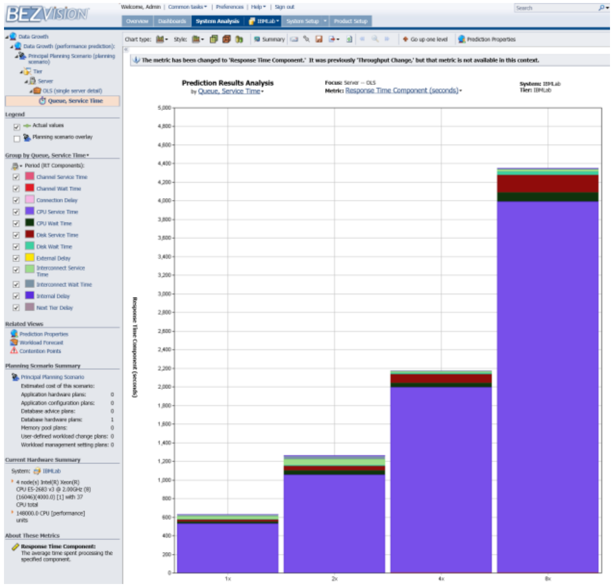


Throughput relative change

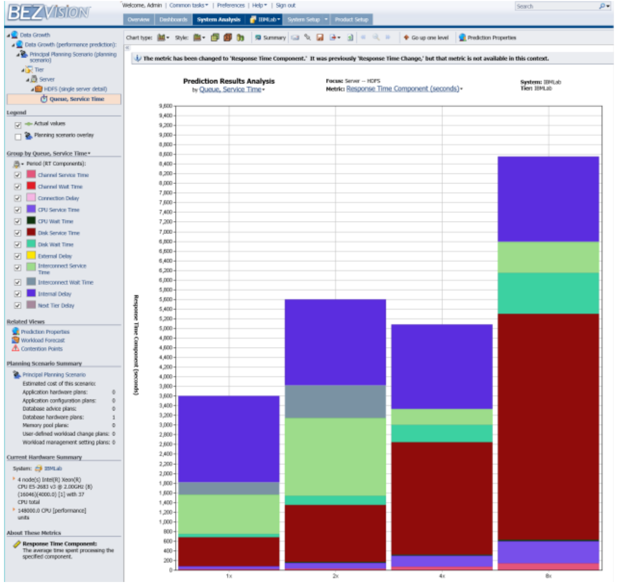


Predicted Impact of Increase Number of Nodes on OLS and HDFS Elapsed Time Components

OLS



HDFS



Example of Business Requirements

- The Score takes into consideration the type of ML algorithm, Number of Observations and Features / Predictors in Data Set, the relative importance of the different criteria, like response time, Accuracy, CPU Utilization, Memory utilization, Number of I/O operations, and other parameters:

Score = w1 * Accuracy + w2 * Response Time + w3 * CPU Utilization + w4 * Memory Utilization +w5 * Scalability , etc

Where the weighting coefficients wi represent business priorities between 0 and 1.

Type of Requirement	Relative Weight
Accuracy	0.2
Response Time	0.4
CPU Utilization	0.2
Memory Utilization	0.2
Total	1

Example of Recommendation

- Response Time can vary between 0 and infinity. We transform the response time as 1 / (1 + RT) to make it as a number between 0 and 1, where 1 is better. In addition to calculating the score we check if predicted CPU Utilization and Memory Usage are less than 1. -
- Value of score is used to recommend the appropriate ML algorithm and ML Library.

Algorithm	library	pred_score	pred_rank	true_score	true_rank
OLS	Python Sklearn	0.962057911	1	0.936165261	1
OLS	Pyspark ML	0.876712666	2	0.753752225	2
Ridge	Python Sklearn	0.781980143	3	0.725268522	3
Ridge	Pyspark ML	0.722426161	4	0.659234146	4
RF	Python Sklearn	0.476284999	5	0.429752013	5
RF	Pyspark ML	0.465422159	6	0.415271967	6

- ML OLS Algorithm using Python Sklearn ML library is the most appropriate algorithm to satisfy business requirements presented in example above.

Summary

- Business Requirements
- Challenges
- Collaboration
- Modeling Expands Benchmarks
- ML Models and QNM Models
- Surprises
- Recommender

References

- Daniel A. Menasce “Software, Performance, or Engineering?” WOSP '02 Proceedings of the 3rd international workshop on Software and performance Pages 239-242
- Max Kuhn, Kjell Johnson, Applied Predictive Modeling, Springer, 2013
- B. Zibitsker, IEEE Conference, Delft, Netherlands, March 2016, Big Data Performance Assurance
- B. Zibitsker, Key note presentation “Role of Big Data Predictive Analytics” Big Data Predictive Analytics Conference, Minsk 2015
- B. Zibitsker, Key Note Presentation on “Big Data Advanced Analytics”, Big Data Advanced Analytics Conference, Minsk 2016,
- Dominique A. Heger “Big Data Predictive Analytics, Applications, Algorithms and Cluster Systems” ISBN:978-1-61422-951-3
- Dr. Joseph Hellerstein, *Microsoft Corp,* Yixin Diao, *IBM* Engineering Performance Using Control Theory
- B. Zibitsker, Proactive Performance Management for Data Warehouses with Mixed Workload, Teradata Partners, 2008, 2009
- J. Buzen, B. Zibitsker, CMG 2006, “Challenges of Performance Prediction in Multi-tier Parallel Processing Environments”
- B. Zibitsker, CMG 2008, 2009 “Hands on Workshops on Performance Prediction for Virtualized Multi-tier Distributed Environments”
- Heger, D., “Introduction to Apache YARN Schedulers & Queues”, Data Analytica, www.mlanalytica.com, 2016
- Heger, D. "Machine Learning in the Realm of Big Data Analytics", Fundcraft Publication, ISBN 978-0-578-19095-2, March 2017.

THANK YOU!