Predicting Resource Availability in a Multimedia Fog Computing Platform

Yi-Ying Huang

October, 2016

Outline

- Motivation
- Research Problem
- System Overview
- Proposed Solution
- Trace-Driven Simulations
  - Trace Collection & Used Datasets
  - Setup
  - Results
- Conclusion & Future Work

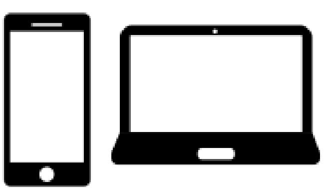
Outline

- **Motivation**
- Research Problem
- System Overview
- Proposed Solution
- Trace-Driven Simulations
  - Trace Collection & Used Datasets
  - Setup
  - Results
- Conclusion & Future Work

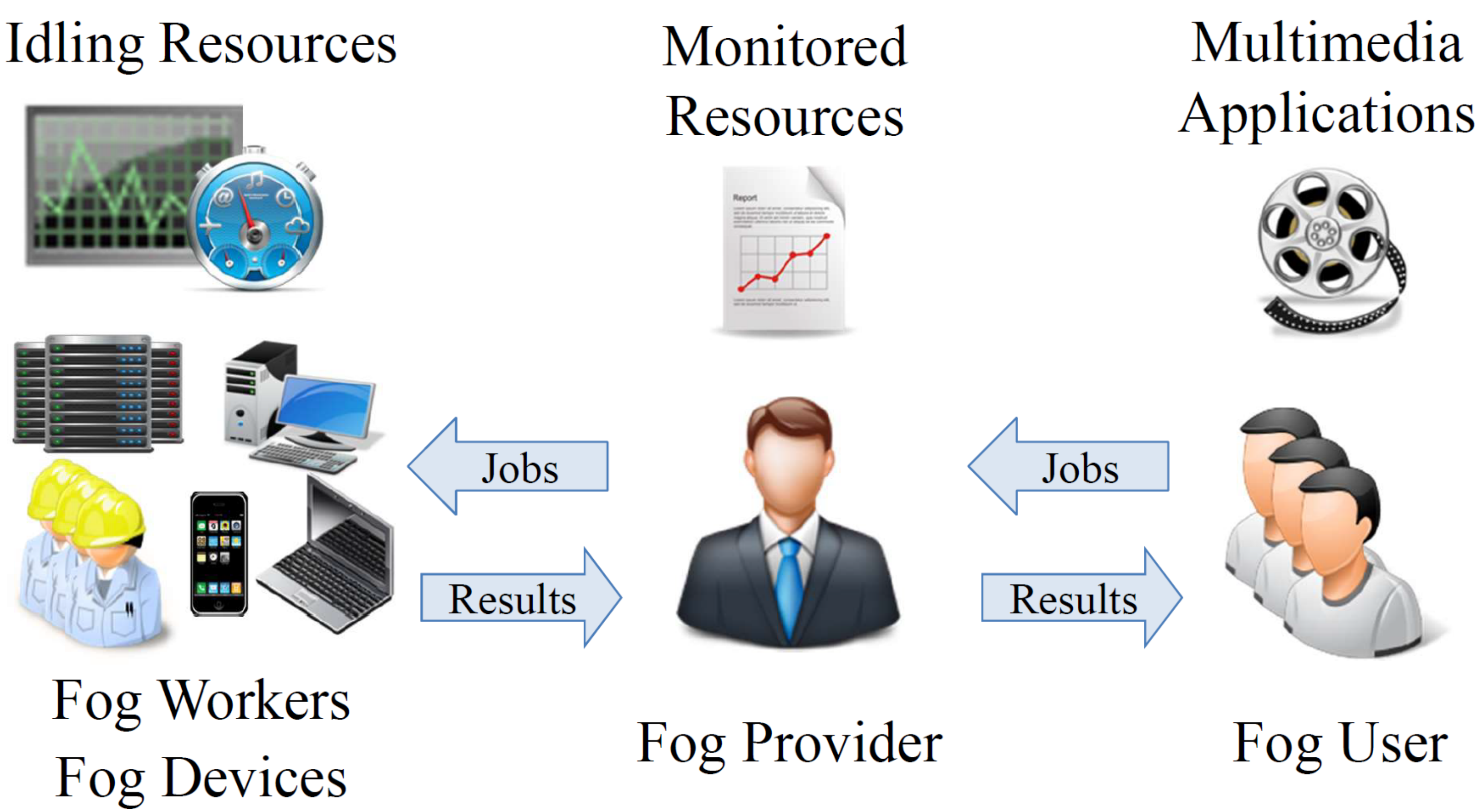
Motivation

- Increasing demands of resource-hungry multimedia jobs
- Expensive cloud service
- Advancing personal devices

⇨ Possible solution: **fog computing**

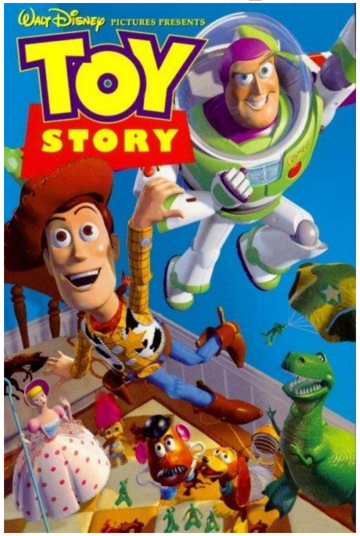


Multimedia Fog Computing Platform



Application: Animation Rendering

- In 1995, Toy Story required 800,000 machine hours to render at 2 to 15 hours per frame [1]
- In 2001, Pixar spent about 12 hours to render a single frame with the main character in it [2]
- In 2014, Disney even needed to render Big Hero 6 on a 55,000-core supercomputer [3]



[1] http://collider.com/pixar-numbers-toy-story-brave/.
[2] http://collider.com/pixar-numbers-monsters-university/.
[3] https: //www.engadget.com/2014/10/18/disney-big-hero-6/.

Outline

- Motivation
- **Research Problem**
- System Overview
- Proposed Solution
- Trace-Driven Simulations
  - Trace Collection & Used Datasets
  - Setup
  - Results
- Conclusion & Future Work

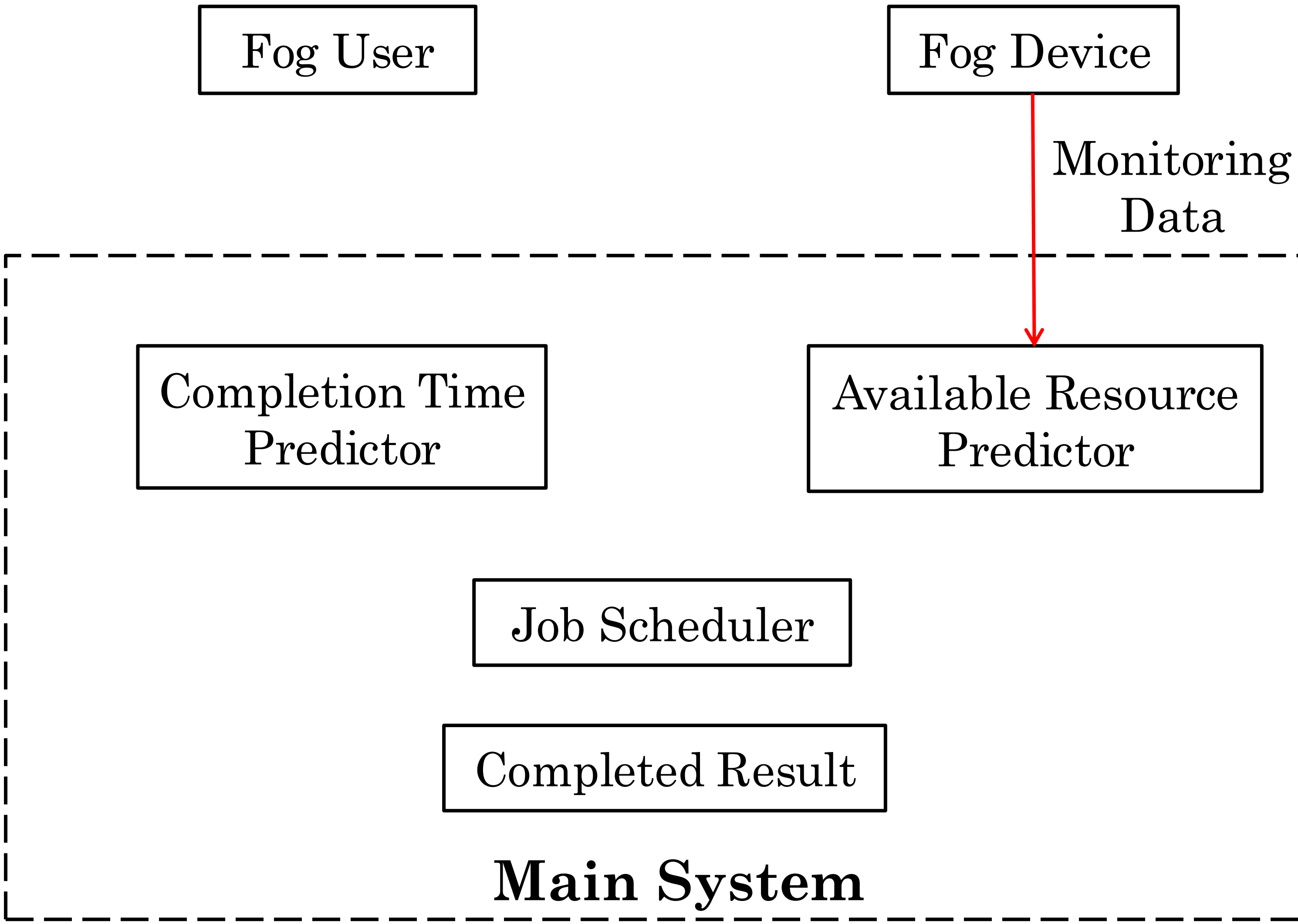
Research Problem

- Accurate prediction of resource availability helps job scheduling in our multimedia fog computing platform
- Each fog user may have his own usage pattern, which leads to daily and weekly regular pattern
- We use machine learning predictors to predict the available resource of a future time period

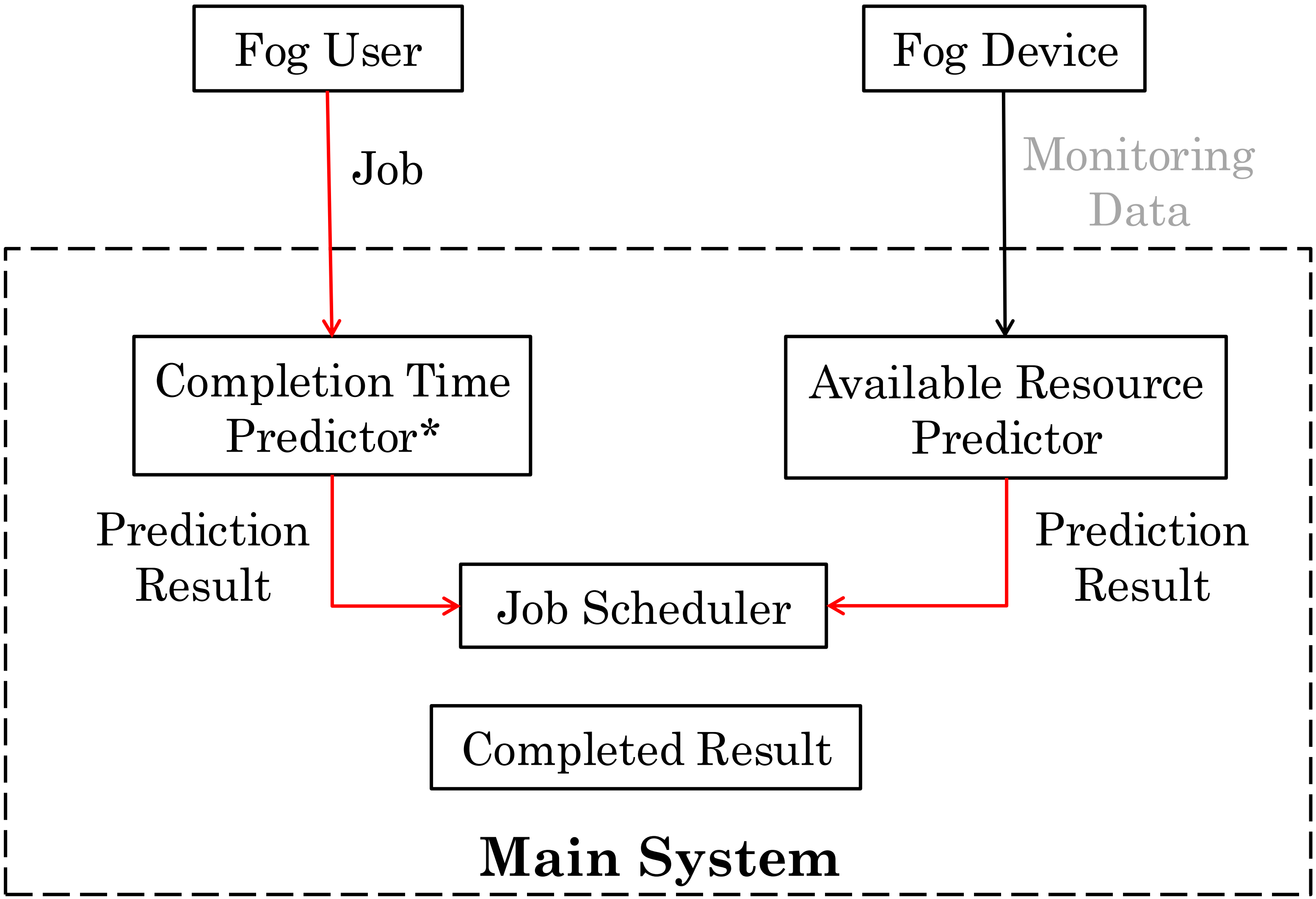
Outline

- Motivation
- Research Problem
- **System Overview**
- Proposed Solution
- Trace-Driven Simulations
  - Trace Collection & Used Datasets
  - Setup
  - Results
- Conclusion & Future Work

System Overview

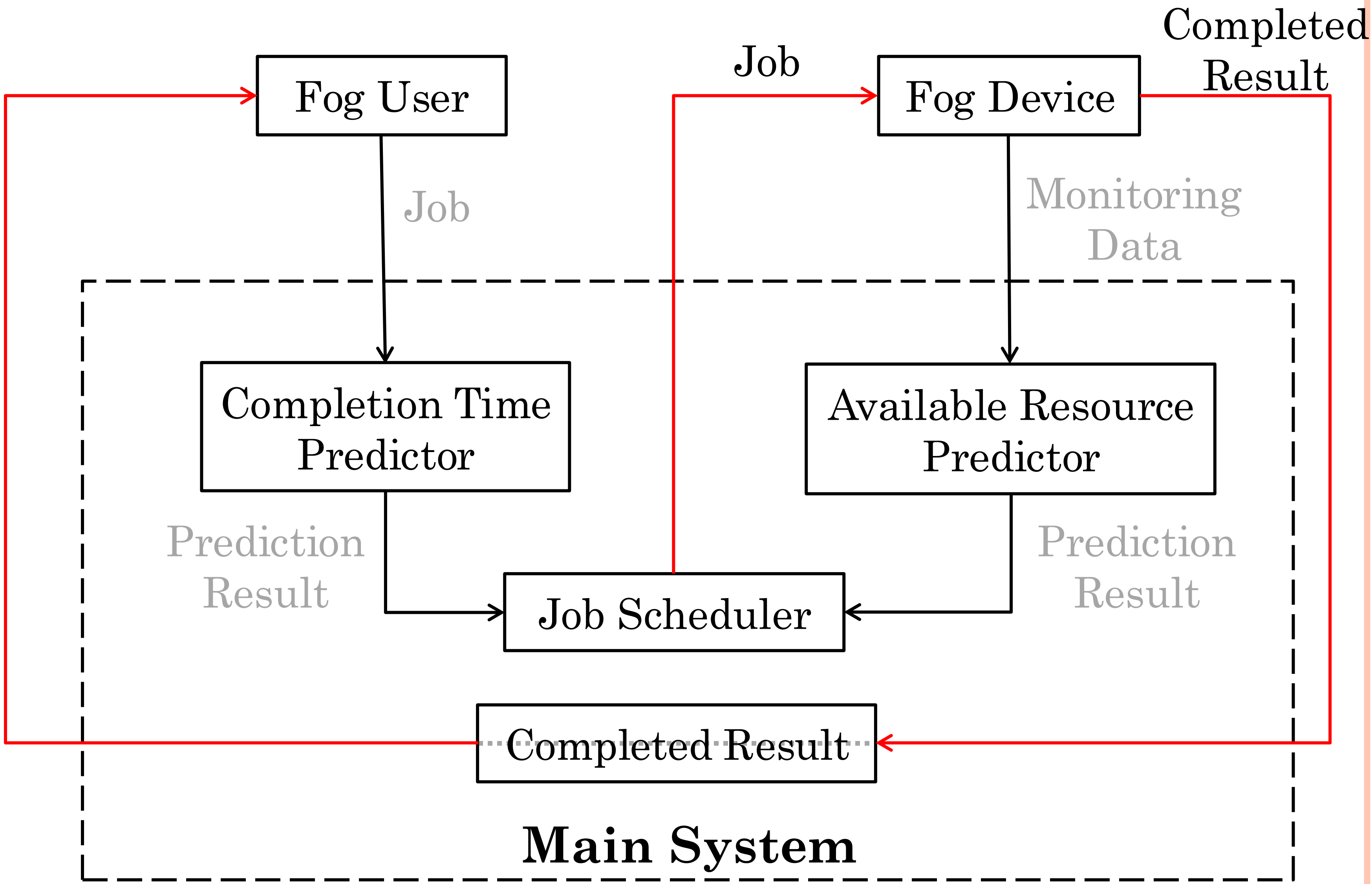


System Overview

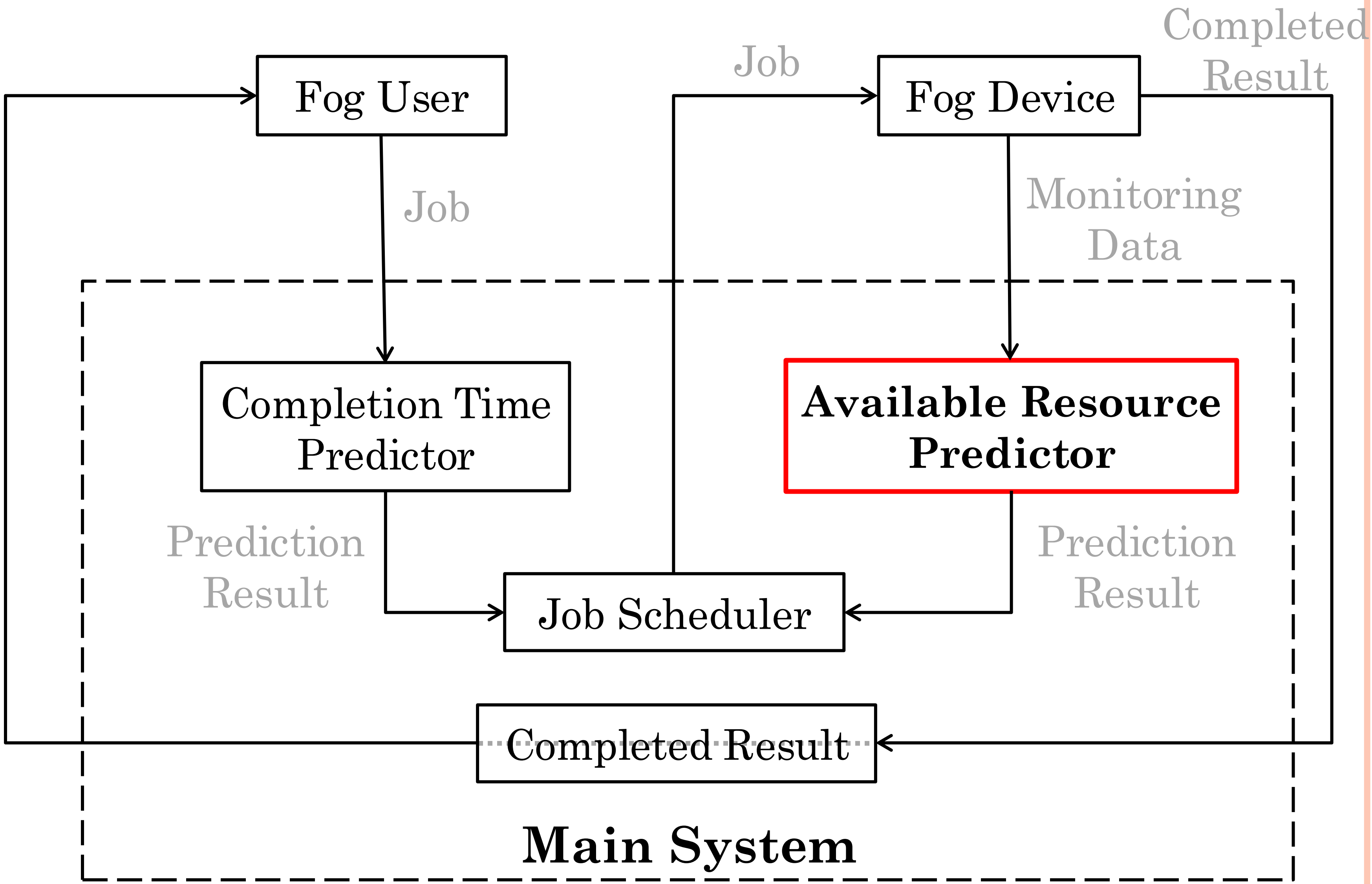


* H. Hong et al., Animation rendering on multimedia fog computing platforms. In *IEEE 8th CloudCom*. 2016.

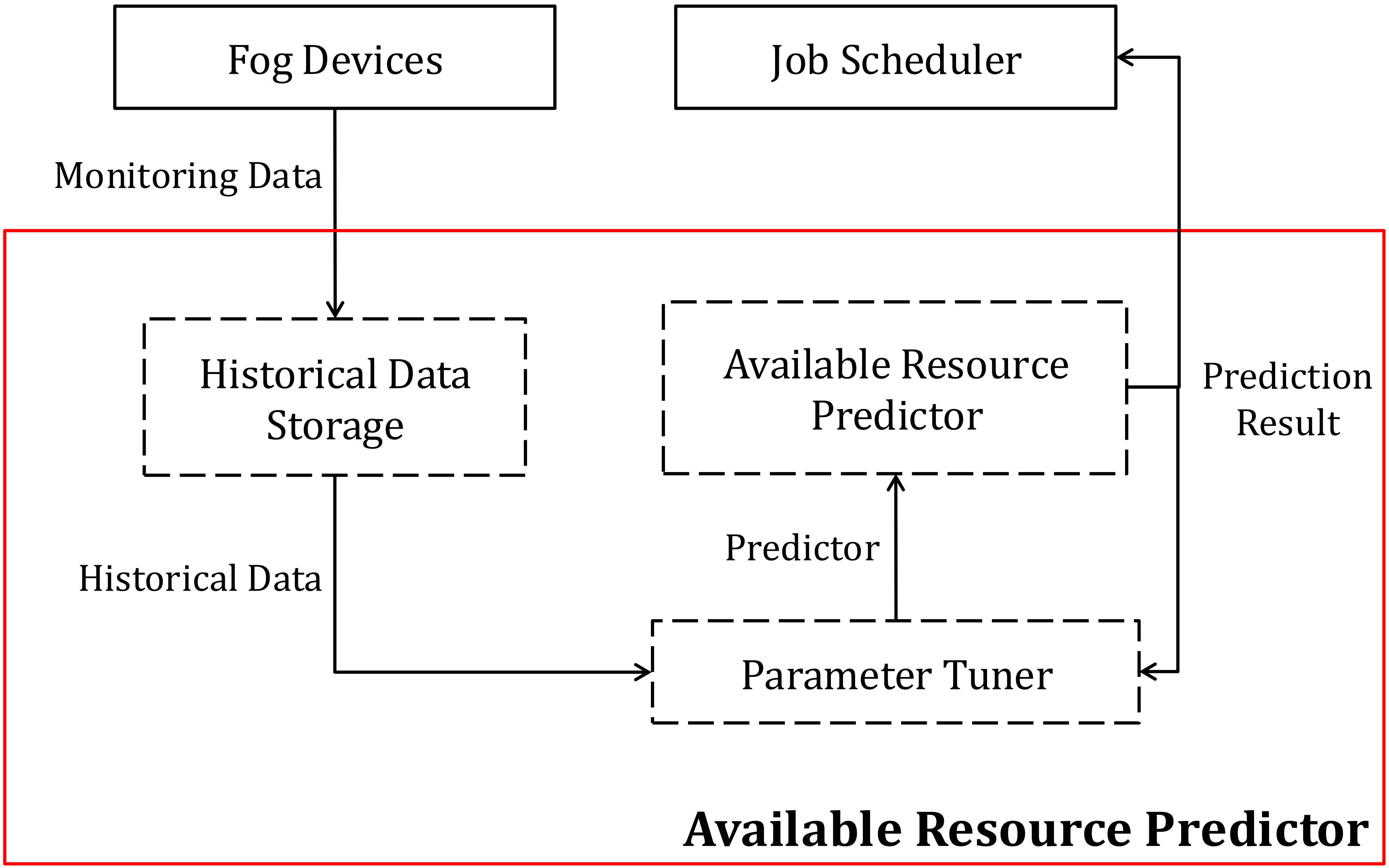
System Overview



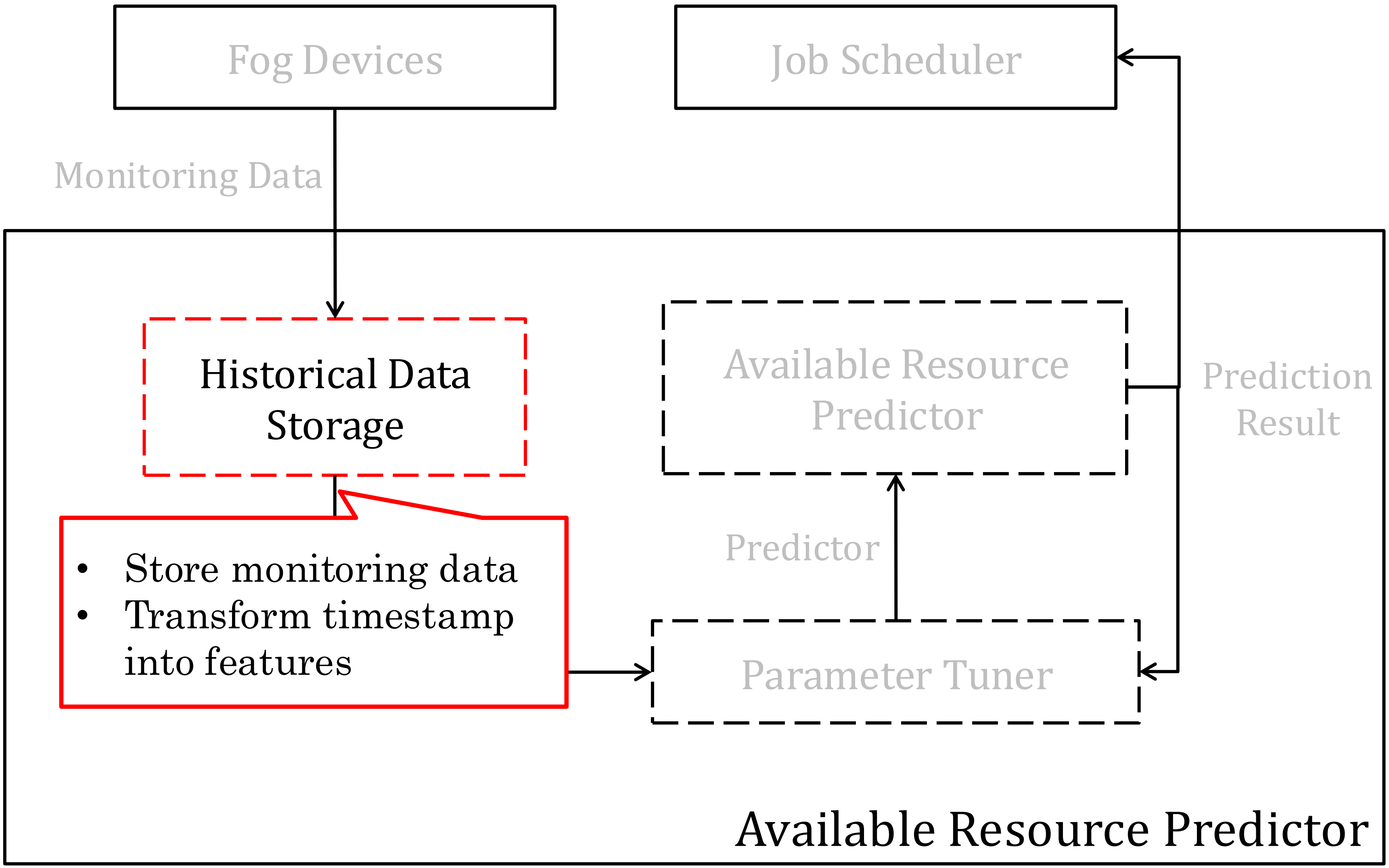
System Overview



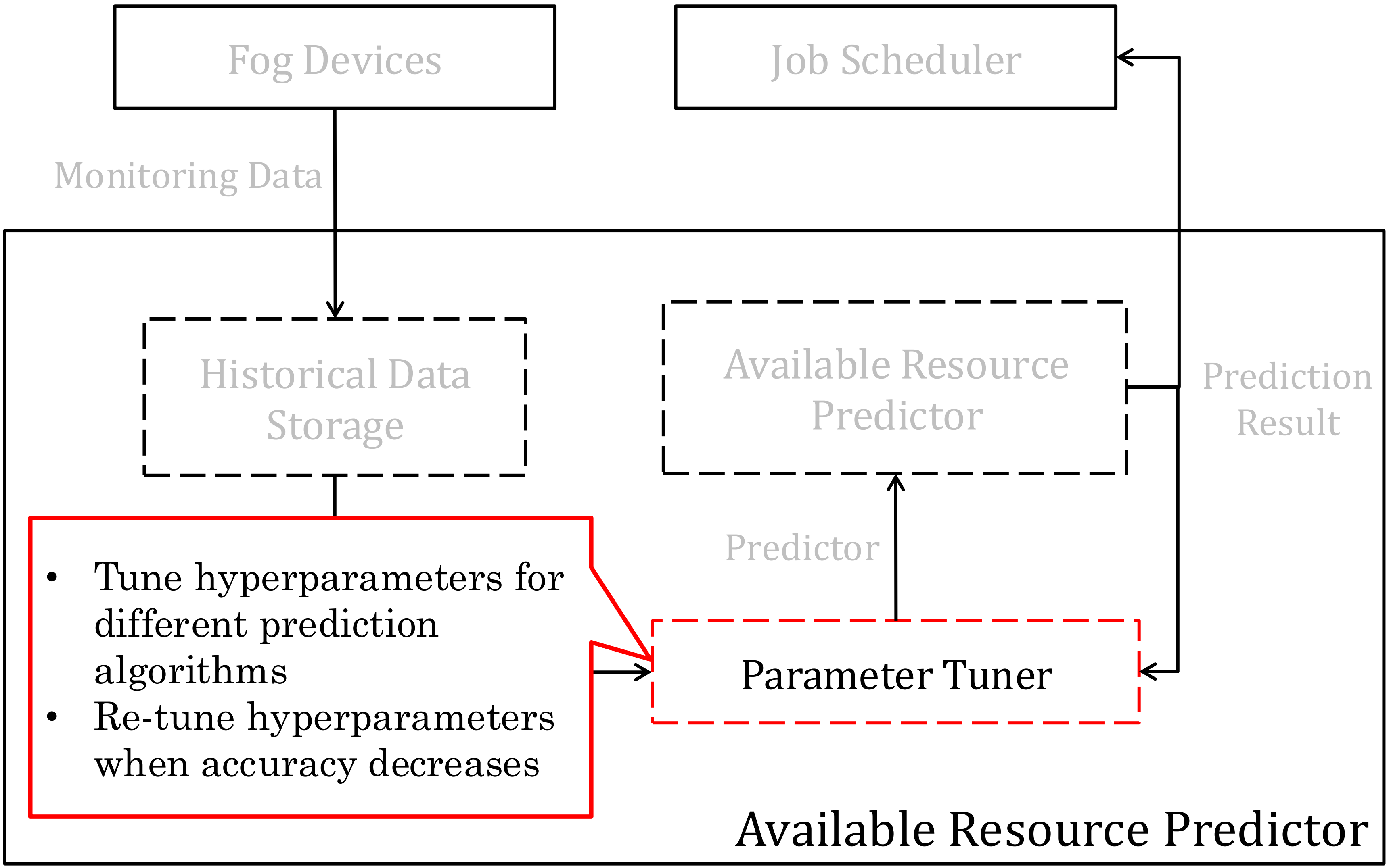
System Overview



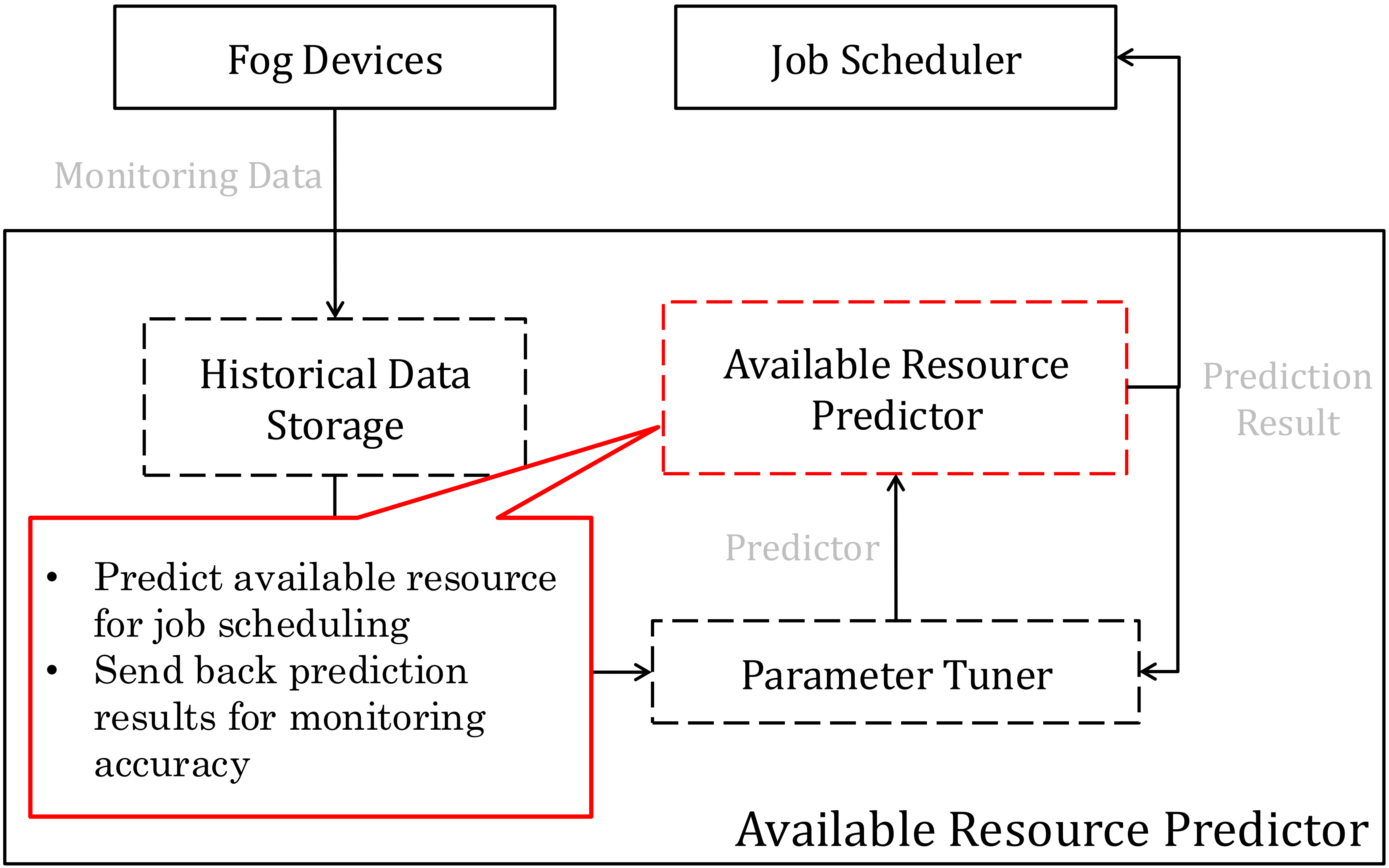
System Overview



System Overview



System Overview



Outline

- Motivation
- Research Problem
- System Overview
- **Proposed Solution**
- Trace-Driven Simulations
  - Trace Collection & Used Datasets
  - Setup
  - Results
- Conclusion & Future Work

Proposed Solution

- Random Forest (RF)
  - Construct a multitude of decision trees
  - Average all trees' prediction results
- Gradient Boosting Tree (GBT)
  - Consists a sequence of trees
  - Each successive tree is to predict the residuals of the preceding one
- Neural Network (NN)
  - Consists input layer, hidden layers, and output layer
  - Each layer contains multiple neurons

- Lack of representative instances incurs high negative impact for GBT
- With large enough training dataset, GBT often outperforms RF
- ➢ RF and GBT have complementary properties

Outline

- Motivation
- Research Problem
- System Overview
- Proposed Solution
- **Trace-Driven Simulations**
  - **Trace Collection & Used Datasets**
  - Setup
  - Results
- Conclusion & Future Work

Trace Collection & Used Datasets

- Dataset 1: **Datacenter Dataset***
  - # of nodes: 500
  - Period: Jul.~Sep. 2013 (3 months)
  - Sampling frequency: 1 record/5 minutes
  - Contents: VM resource usage
- Dataset 2: **Desktop Dataset**
  - # of volunteers: 25
  - Period: late May~Jun. 2016 (1 month)
  - Sampling frequency: 1 record/10 seconds
  - Contents: real users' resource usage

Resource usage includes CPU usage, memory usage, disk usage, and network rx/tx throughput

* Datacenter dataset resource: http://www.bitbrains.nl/solvinity-en

Sample Statistics of Datasets

	Datacenter Dataset	Desktop Dataset
Total # of nodes	500	25
Period	3 months	1 month
Total # of records	12,496,728	2,967,335
Avg. # of records	24,993	118,693
size of training set	9,997,696	2,373,909
size of testing set	2,499,032	593,426
# of features	9	9
Features	id, epoch, dayInMonth, dayInWeek, isWeekend, hourInWeek, hourInDay, minute, daySlot	id, epoch, dayInMonth, dayInWeek, isWeekend, hourInWeek, hourInDay, minute, daySlot
Prediction Target	cpuUsagePercent	cpuUsagePercent

- We split each dataset into (a) training set (80%) and (b) testing set (20%)*

* Abu-Mostafa et al., *Learning from data*, volume 4. AMLBook Singapore, 2012.

Procedure of Generating Model

D

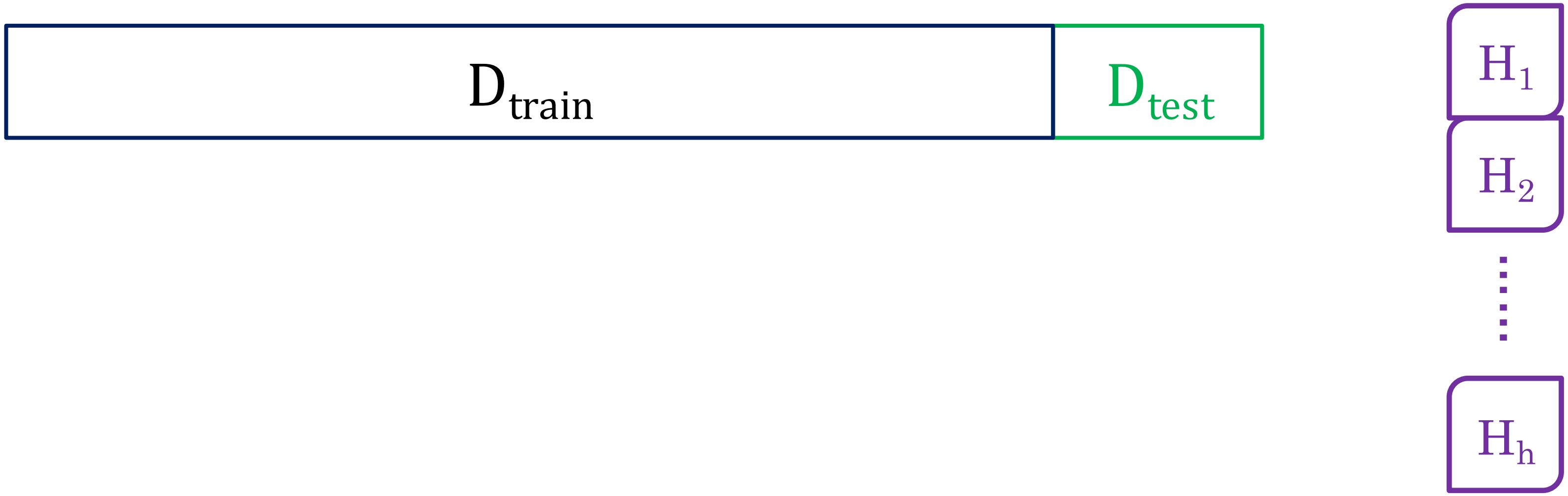
Procedure of Generating Model (Cont.)

1. Divide the data into training and testing set

D_{train}	D_{test}

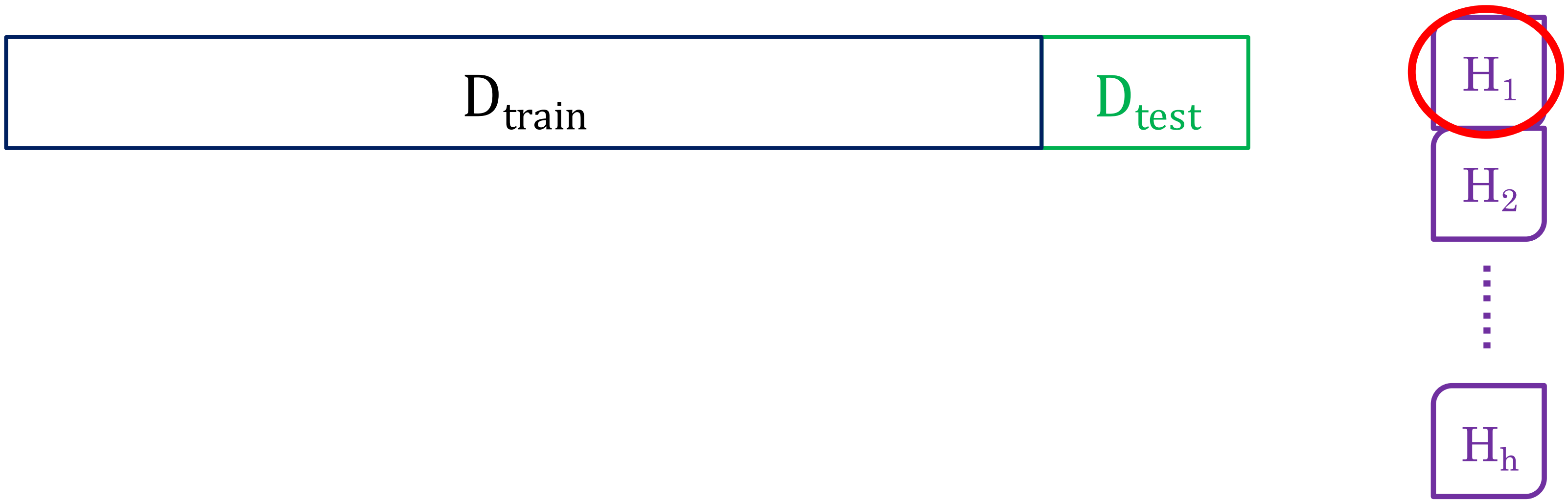
Procedure of Generating Model (Cont.)

2. Test different combination of hyperparameters



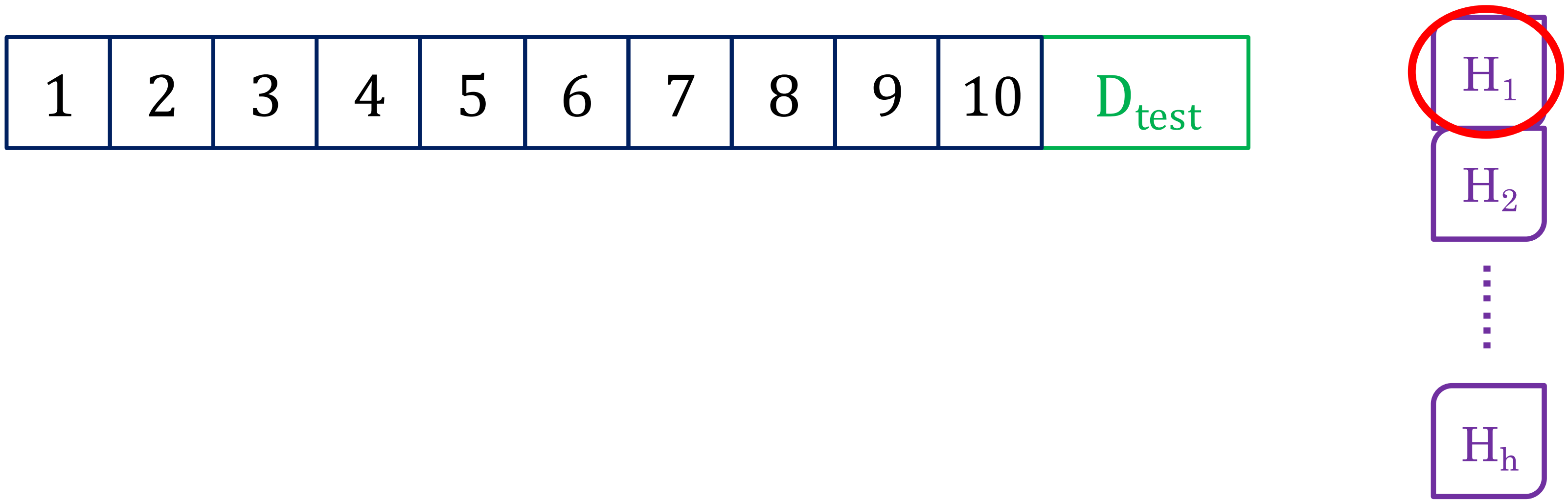
Procedure of Generating Model (Cont.)

3. Perform **10-fold cross validation** for each combination of hyperparameter using training set



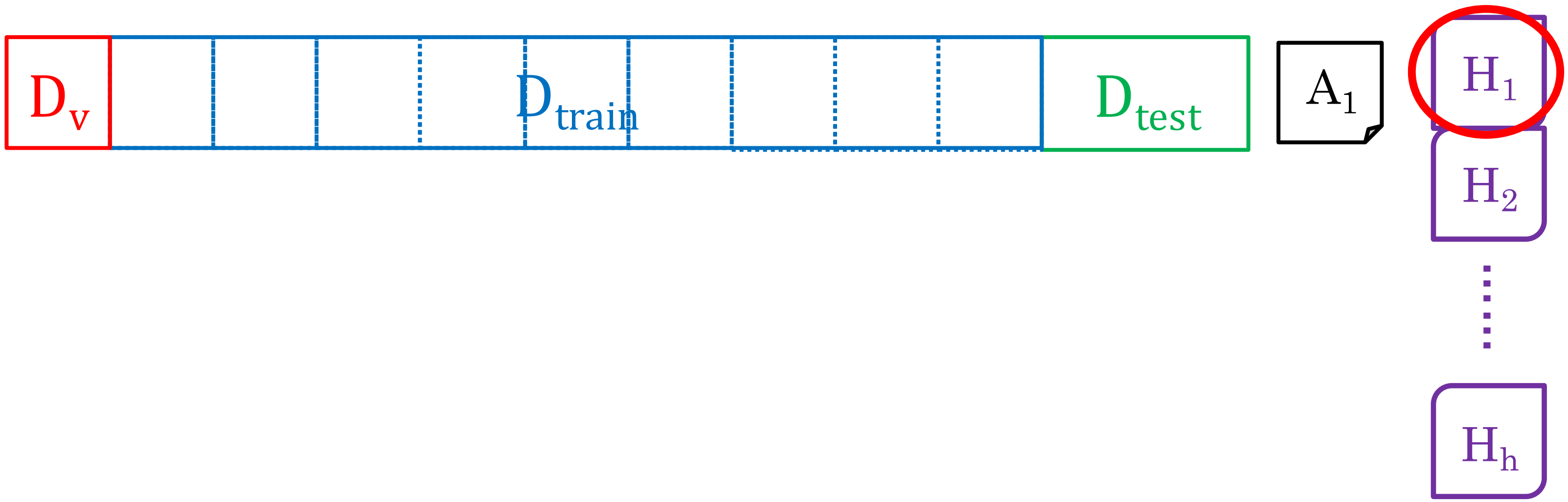
Procedure of Generating Model (Cont.)

a. Divide training set into 10 portions



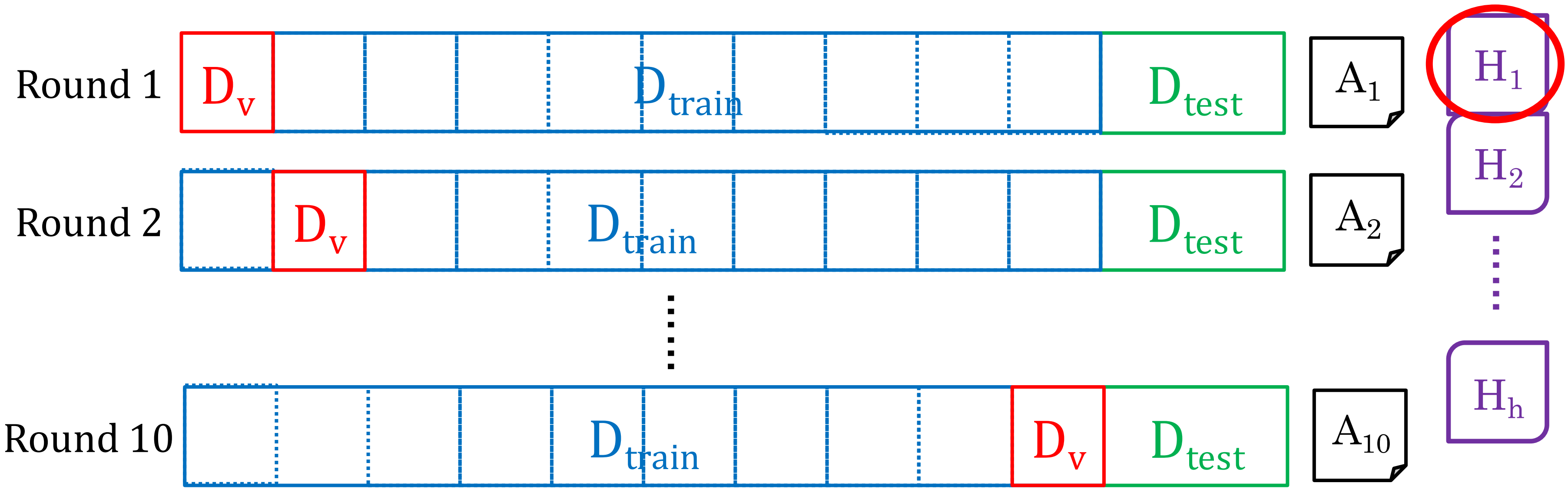
Procedure of Generating Model (Cont.)

b. Use one portion as validation set and the rest as training set
c. Train the model using the training set
d. Validate the model using the validation set and get accuracy *A*



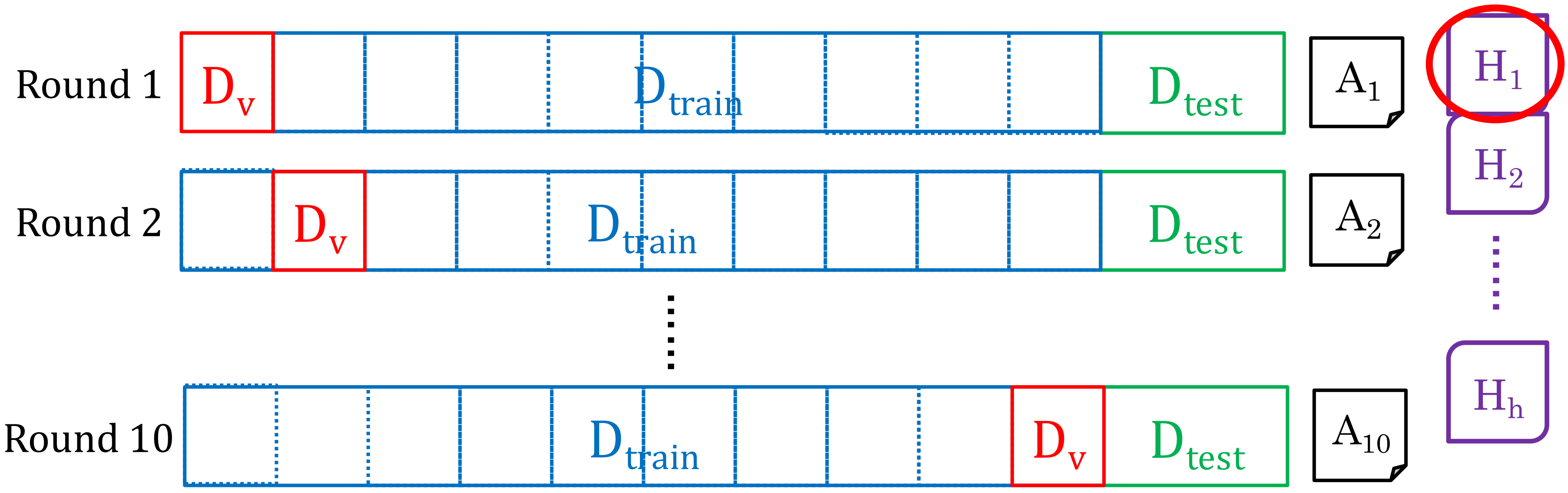
Procedure of Generating Model (Cont.)

e. Repeat c. and d. 10 times until every portion has been used for validation



Procedure of Generating Model (Cont.)

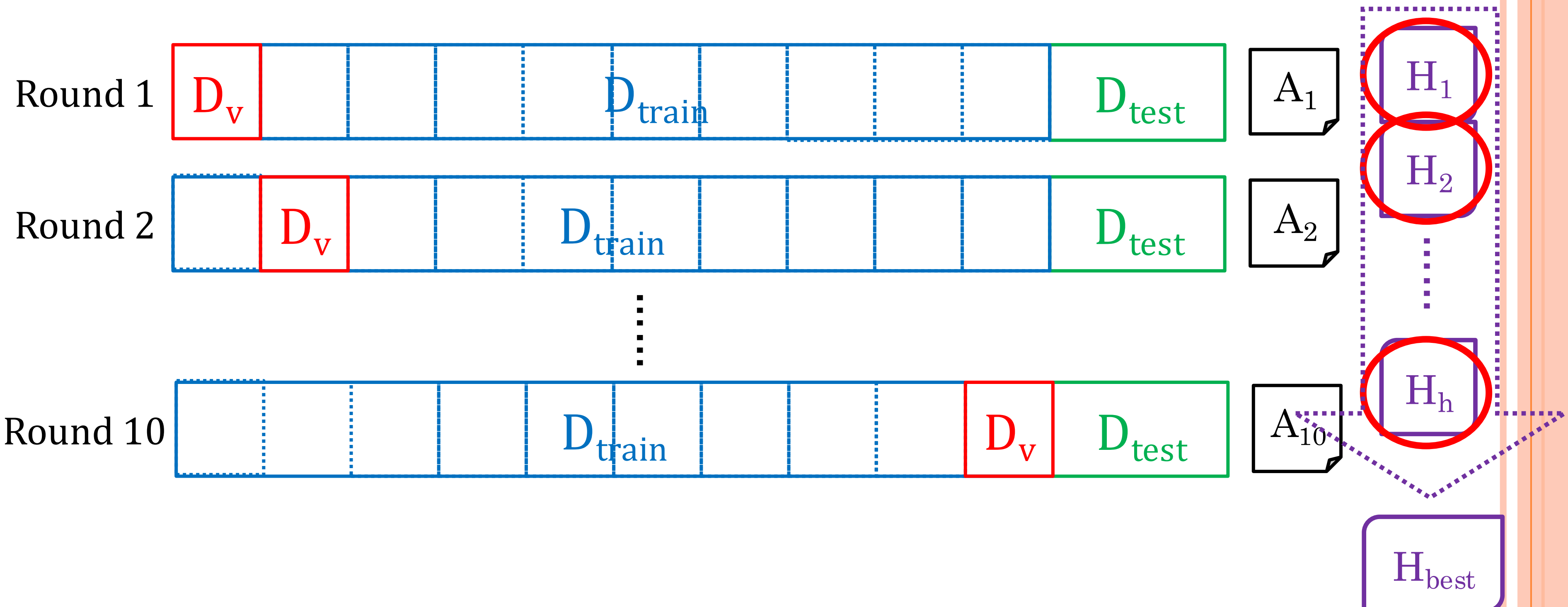
f. Average all 10 accuracy scores as the result



$$\text{Prediction Accuracy } A = \frac{1}{N}\sum_{i=1}^{N} A_i$$

Procedure of Generating Model (Cont.)

4. Select the combination of hyperparameter with the highest accuracy score

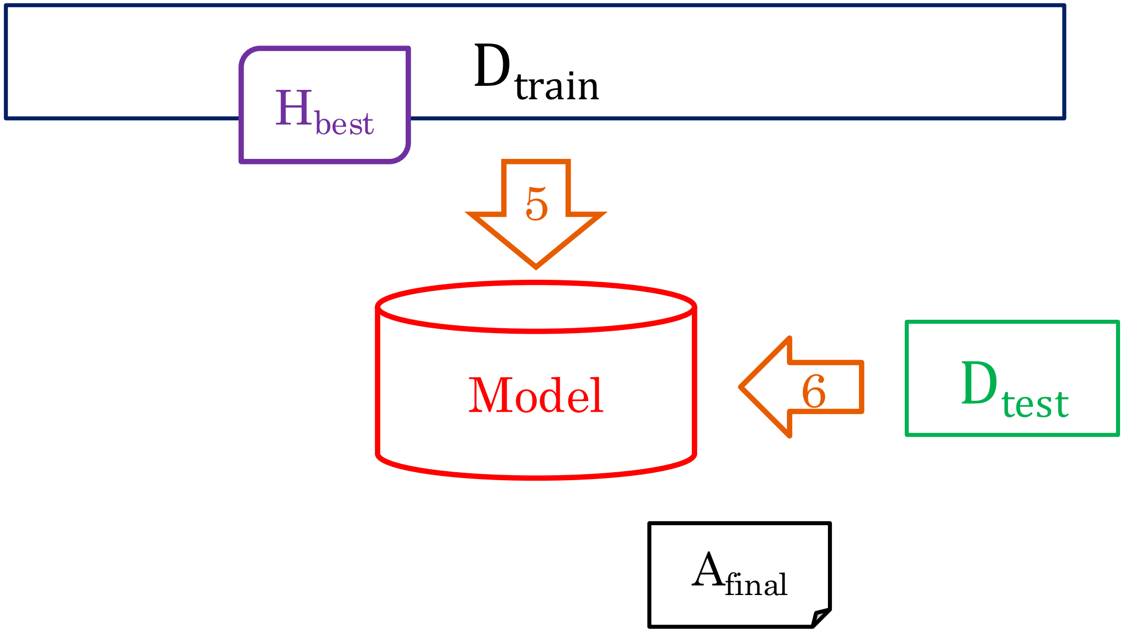


Prediction Accuracy $A = \frac{1}{N}\sum_{i=1}^{N} A_i$

Procedure of Generating Model (Cont.)

5. Generate model with the selected combination using the training set

6. Test the model using testing set and report the final accuracy



Procedure of Generating Model (Cont.)

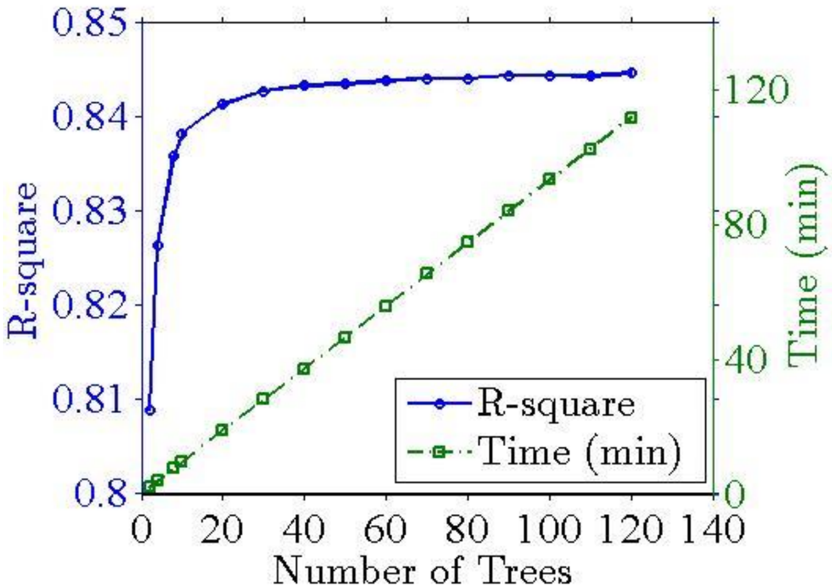
1. Divide the data into training and testing set
2. Test different combination of hyperparameters
3. Perform 10-fold cross validation for each combination of hyperparameter using training set
    a. Divide training set into 10 portions
    b. Use one portion as validation set and the rest as training set
    c. Train the model using the training set
    d. Validate the model using the validation set and get accuracy score
    e. Repeat c. and d. 10 times until every portion has been used for validation
    f. Average all 10 accuracy scores as the result
4. Select the combination of hyperparameter with the highest accuracy score
5. Generate model with the selected combination using the training set
6. Test the model using testing set and report the final accuracy

Hyperparameters of the Predictors

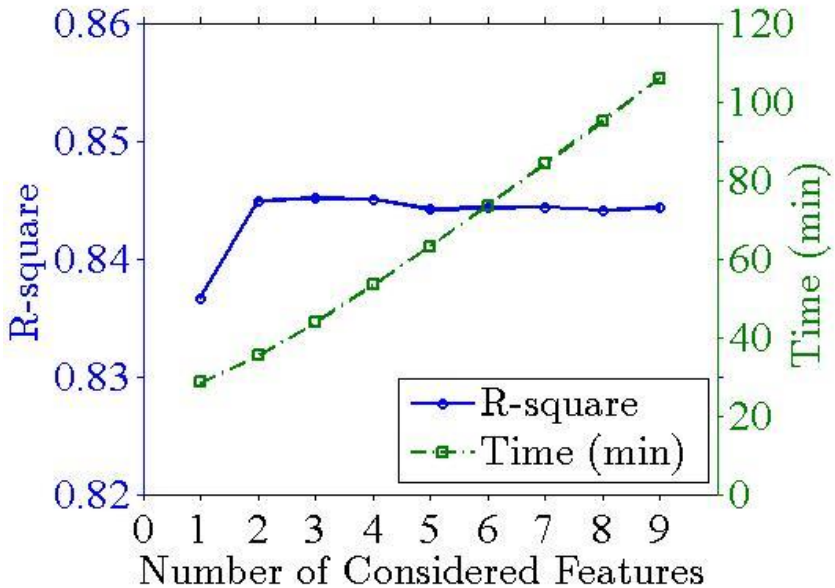
- RF-based predictor
  - The number of trees
  - The number of considered features
- GBT-based predictor
  - The number of trees
  - The maximal depth of each tree
  - The shrinkage (i.e., the learning rate)
- Metrics: execution time and r^2 score

Tuning Hyperparameters of RF

- Desktop dataset
  - The number of trees [2, 4, 8, 10, 20, …, 150]
  - The number of features [1, 2, …, 9]



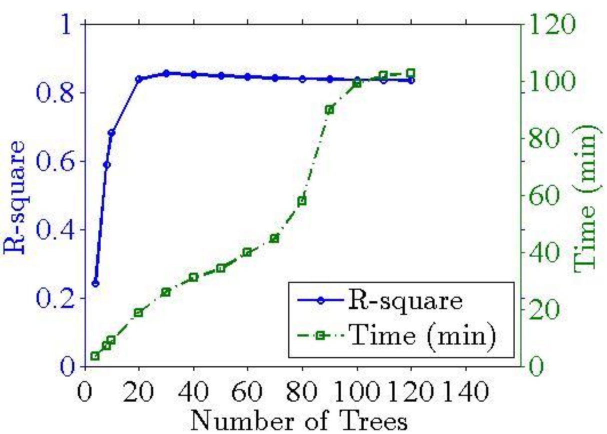
Chosen trees = 80



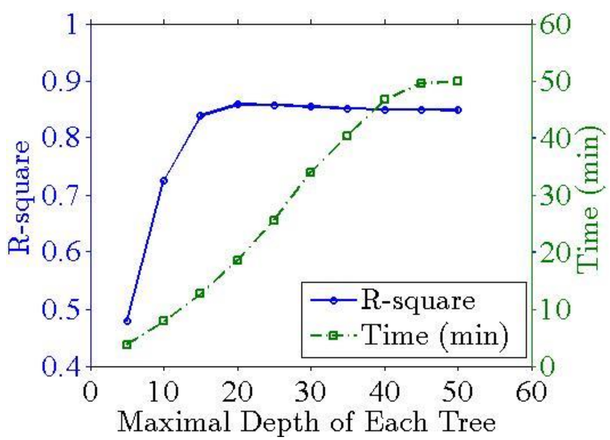
Chosen features = 9

Tuning Hyperparameters of GBT

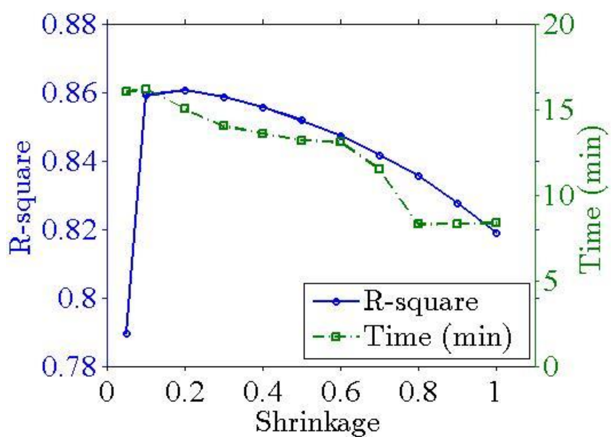
- Desktop dataset
  - The number of trees [2, 4, 8, 10, 20, …, 130]
  - The maximal depth of each tree [5, 10, …, 50]
  - The shrinkage [0.05, 0.1, 0.2, …, 1]



Chosen trees = 30



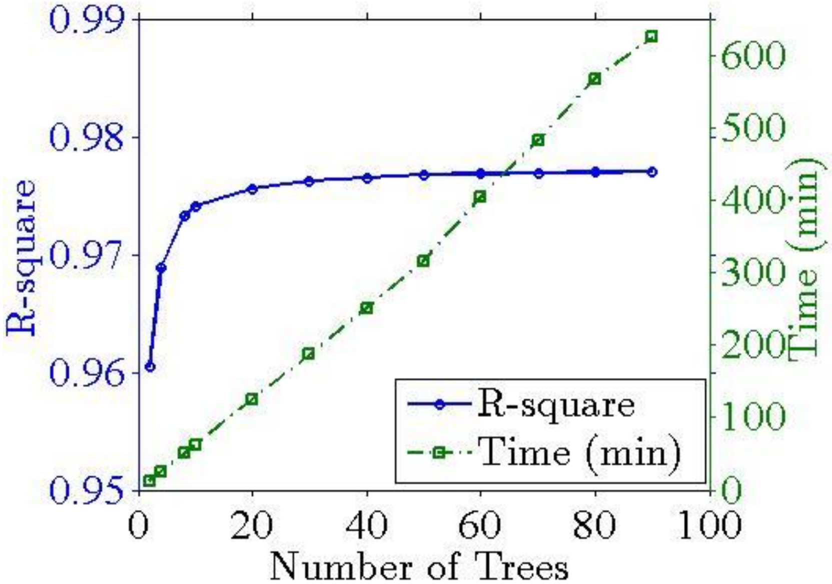
Chosen depth = 20



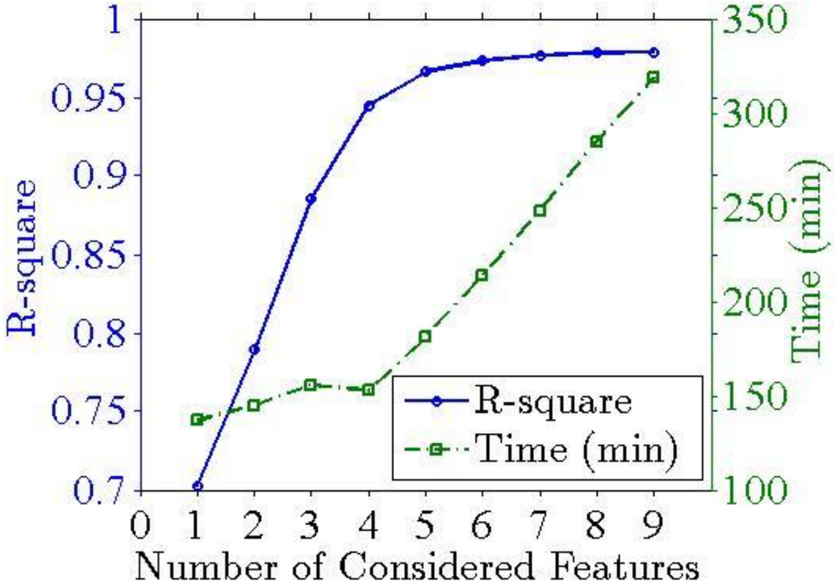
Chosen shrinkage = 0.2

Tuning Hyperparameters of RF

- Datacenter dataset
  - The number of trees [2, 4, 8, 10, 20, ..., 90]
  - The number of features [1, 2, ..., 9]



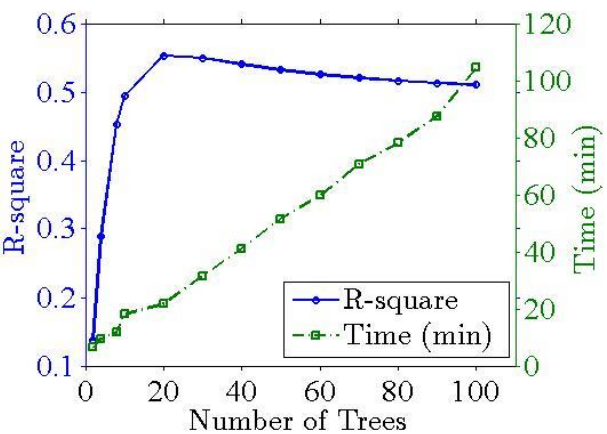
Chosen trees = 40



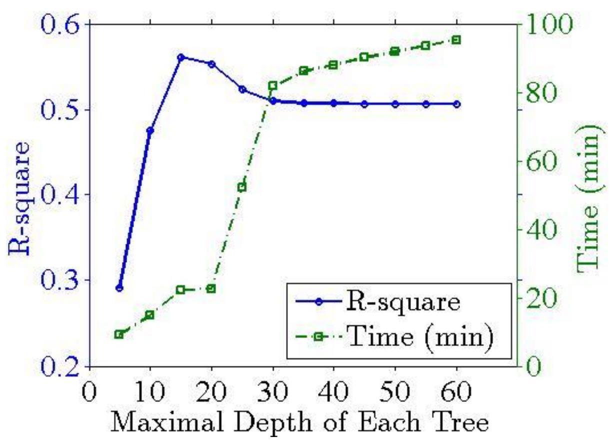
Chosen features = 9

Tuning Hyperparameters of GBT

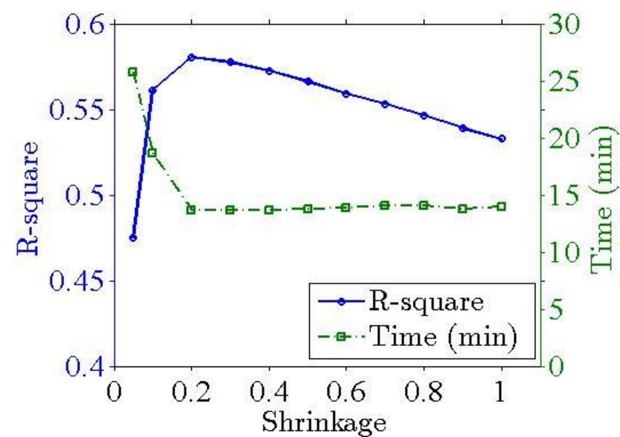
- Datacenter dataset
  - The number of trees [2, 4, 8, 10, 20, ..., 100]
  - The maximal depth of each tree [5, 10, ..., 60]
  - The shrinkage [0.05, 0.1, 0.2, ..., 1]



Chosen trees = 20



Chosen depth = 15



Chosen shrinkage = 0.2

The Chosen Hyperparameters

- RF-based predictor

	Datacenter Dataset	Desktop Dataset
Number of Trees	40	80
Number of Features	9	9

- GBT-based predictor

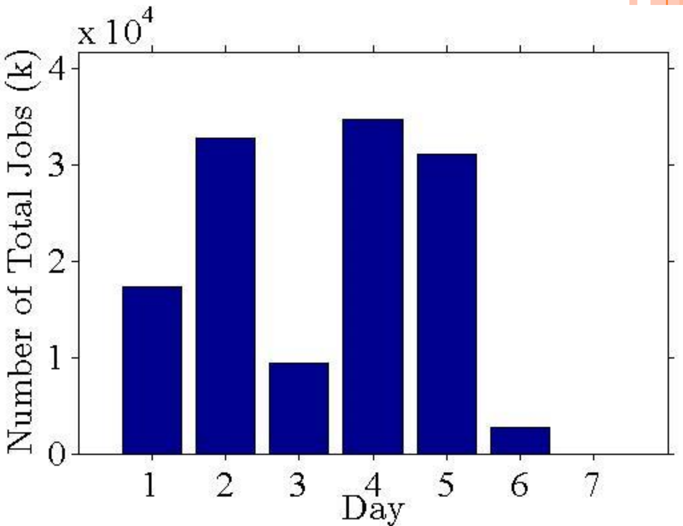
	Datacenter Dataset	Desktop Dataset
Number of Trees	20	30
Depth of Trees	15	20
Shrinkage	0.2	0.2

Outline

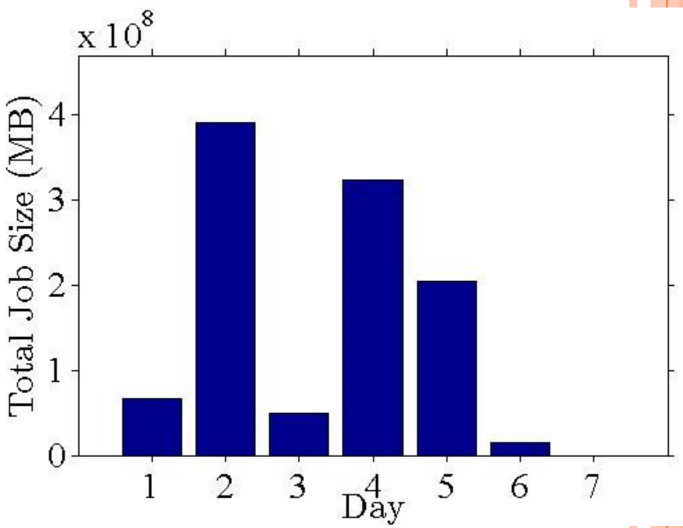
- Motivation
- Research Problem
- System Overview
- Proposed Solution
- **Trace-Driven Simulations**
  - Trace Collection & Used Datasets
  - **Setup**
  - Results
- Conclusion & Future Work

Setup

- Input: Animation job rendering dataset
  - 127,791 records collected between Sep. and Nov. 2015
  - Job's arrival time and size



Feature	Mean	Std.
CPU Usage (%)	19.7	11.7
RAM Usage (KB)	380.7	147.5
# of Frames	113.9	76.7
# of Polygons	63512.6	332868.8
Image Size (Pixels)	131161.6	17453.5
Completion Time (s)	104.1	194.2



Setup (Cont.)

- Input: Available resource dataset (Datacenter/Desktop dataset)
  - 2,499,032 records / 593, 426 records
  - Actual CPU availability of the recorded period
  - Use Poisson process with a mean arrival rate $\lambda = 30$ mins to generate the devices' arrival time
- Earliest Start Scheduling (ESS)*
  - Batch arrived jobs every day, schedule at 23:59, and starts processing them in the next day

* P. Brucker and S. Knust. *Complex Scheduling*. Springer, 2012.

Setup (Cont.)

- Implement the perfect scheduling, Oracle
- Implement the simulator using Java, run simulations for each solution and each dataset for 10 times and present the 95% confidence intervals whenever applicable
- Implement the algorithms using open-source libraries, scikit-learn [1] and xgboost [2]

[1] http://scikit-learn.org/
[2] https://github.com/dmlc/xgboost/

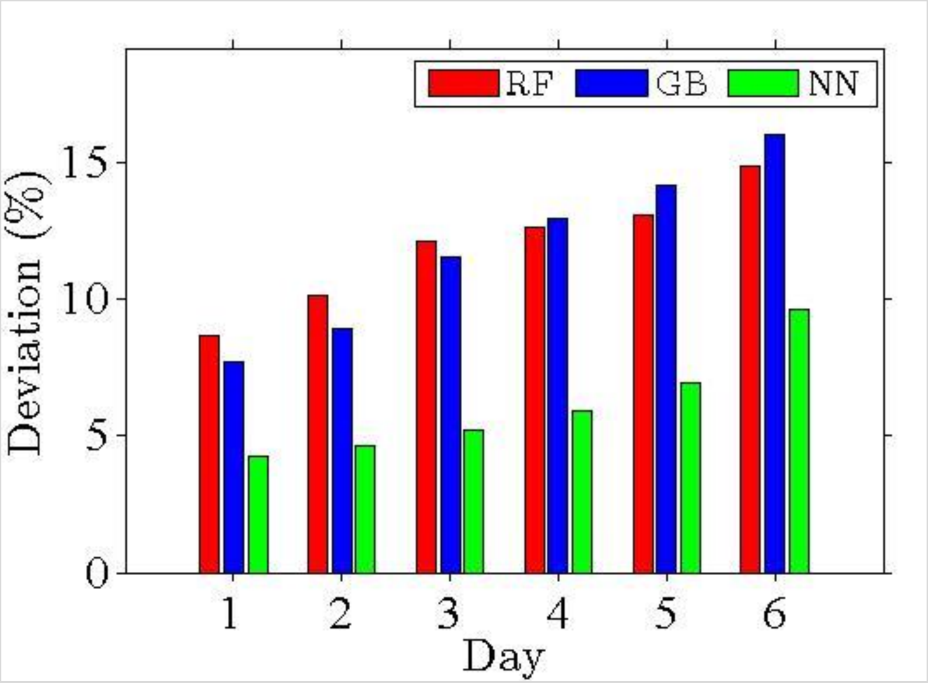
Performance Metrics

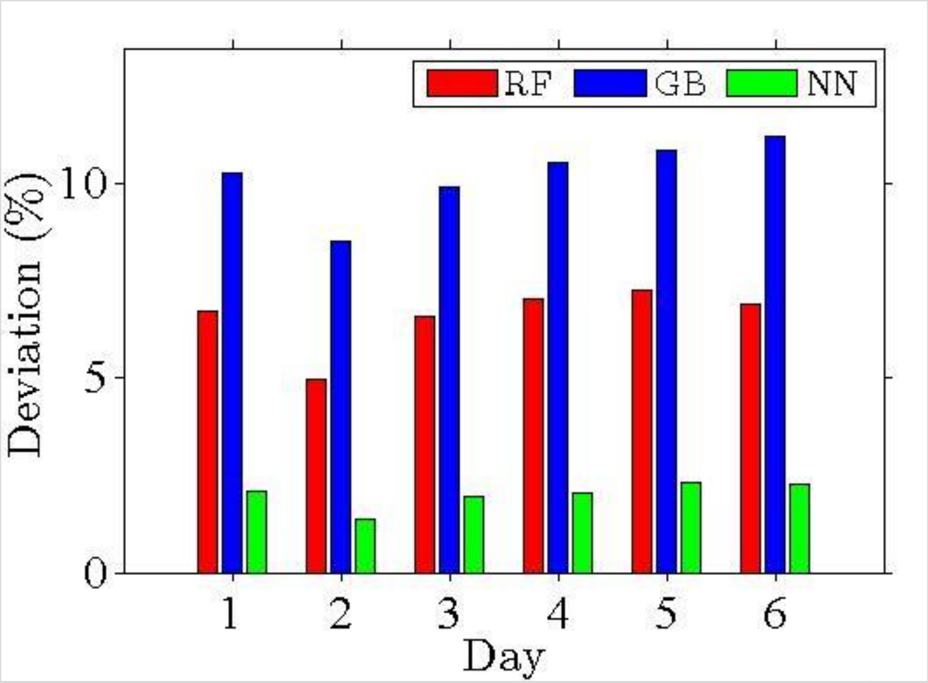
- Deviation
  - $\text{abs}(\hat{Y} - Y)$
- Completed job ratio
  - Ratio of # of completed jobs to that of total jobs
- Makespan
  - Total time to complete a set of jobs (including execution time and waiting time)
- # of failed jobs
  - # of jobs that are not completed when the day ends
- Normalized CPU consumption
  - CPU consumption normalized to that of the Oracle

Outline

- Motivation
- Research Problem
- System Overview
- Proposed Solution
- **Trace-Driven Simulations**
  - Trace Collection & Used Datasets
  - Setup
  - **Results**
- Conclusion & Future Work

Results – Deviation

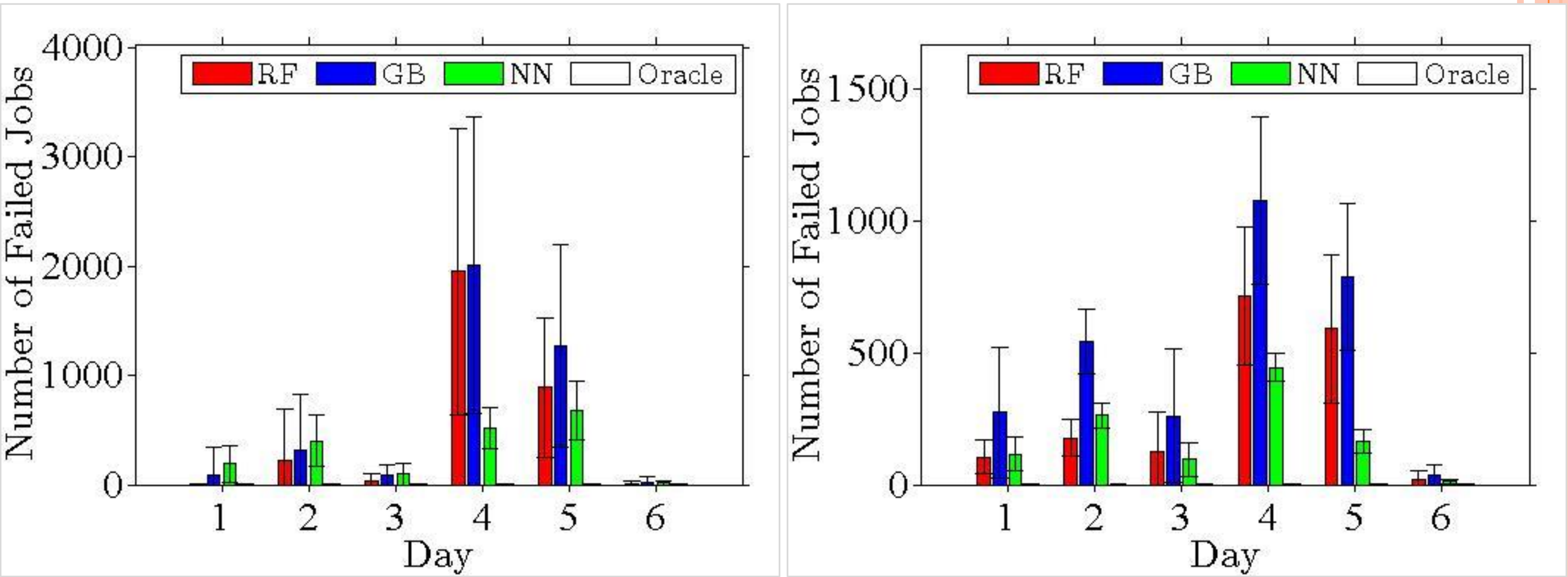




- Deviation of three solutions for desktop / datacenter dataset

⇨ NN-based algorithm performs the most accurate prediction for both datasets

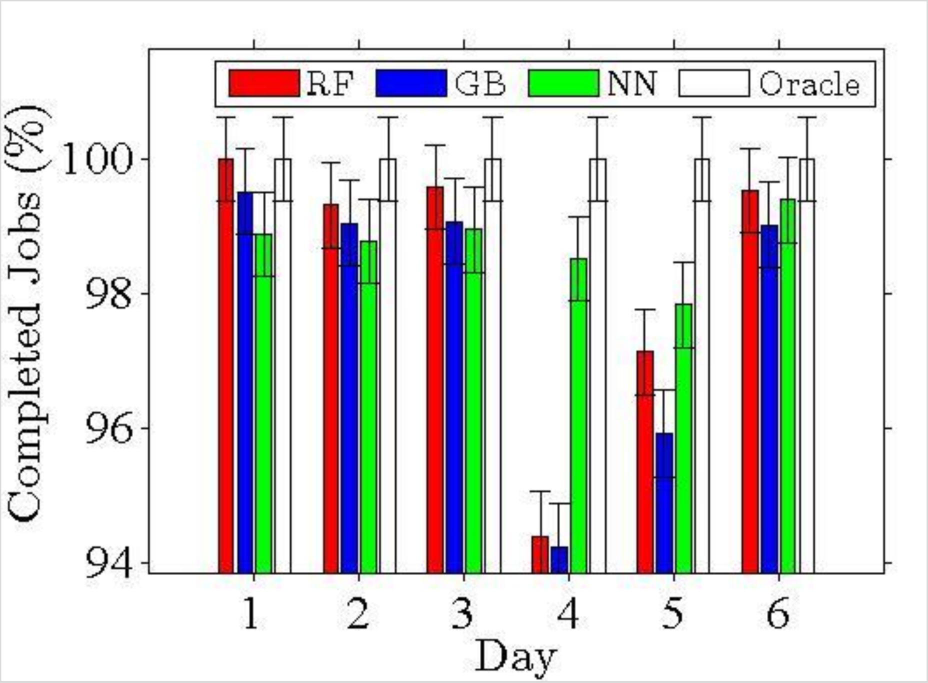
Results – # of Failed Jobs

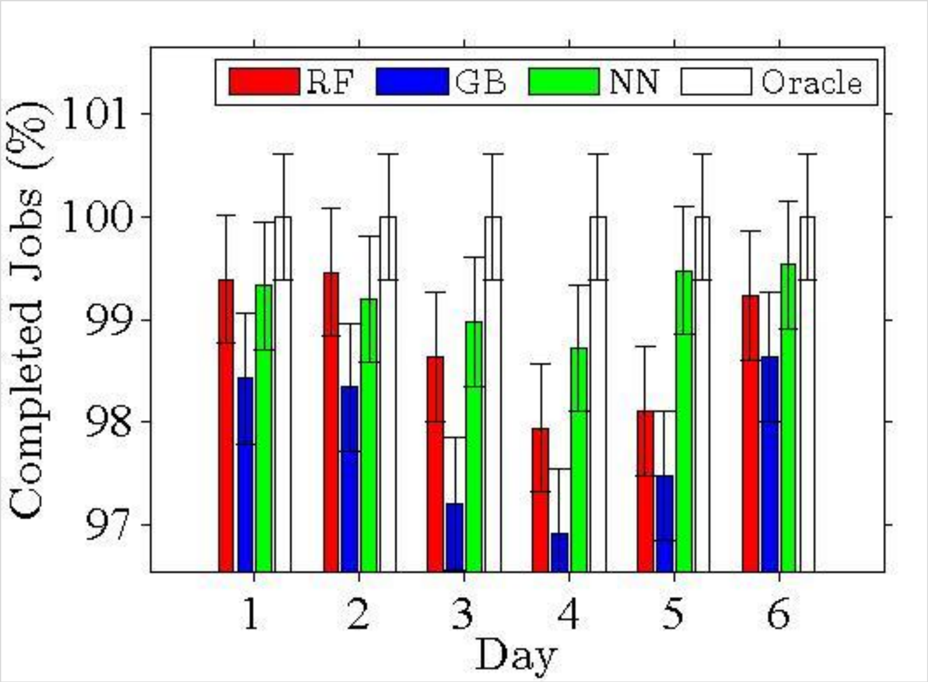


- # of failed jobs of three solutions for desktop / datacenter dataset

⇨ More accurate prediction leads to less failed jobs

Results – Completed Jobs Ratio

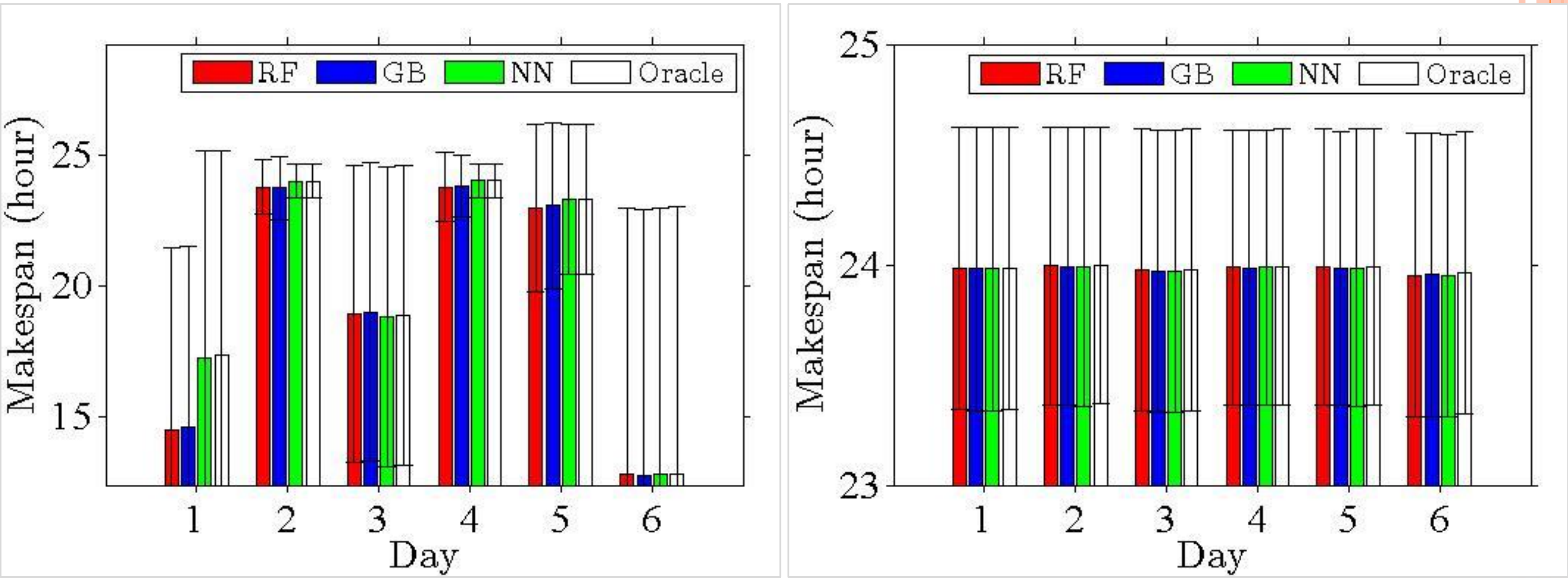




- Completed jobs ratio of three solutions for desktop / datacenter dataset

⇨ **Three solutions perform close to Oracle**

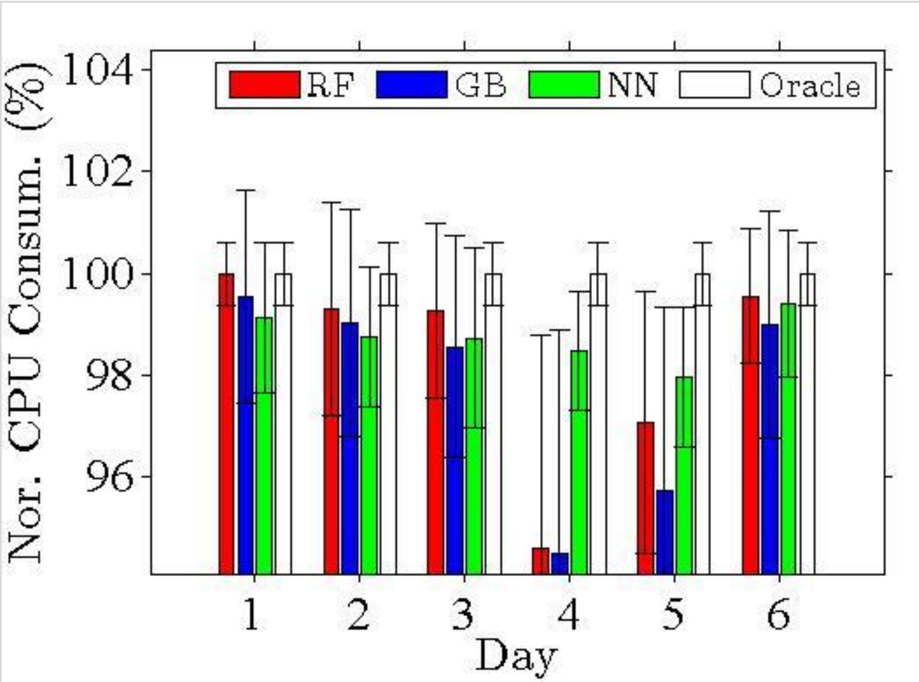
Results – Makespan

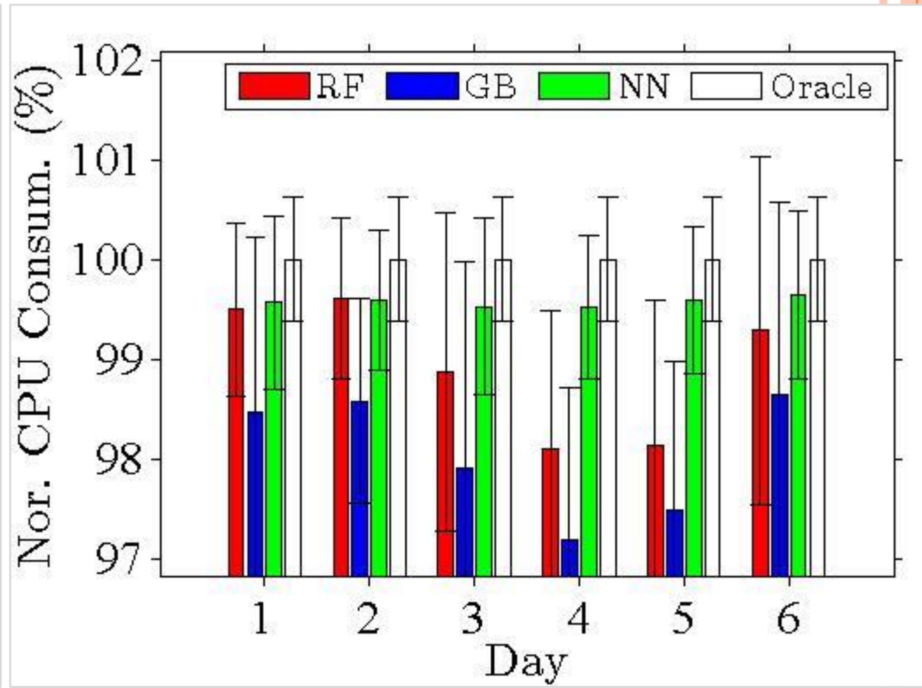


- Makespan of three solutions for desktop / datacenter dataset

⇨ Three solutions perform close to Oracle

Results – Nor. CPU Consumption





- Normalized CPU consumption of three solutions for desktop / datacenter dataset

⇨ **Three solutions perform close to Oracle**

Outline

- Motivation
- Research Problem
- System Overview
- Proposed Solution
- Trace-Driven Simulations
  - Trace Collection & Used Datasets
  - Setup
  - Results
- **Conclusion & Future Work**

Conclusion

- Propose the multimedia fog computing platform
  - Utilize idling resources of fog devices
- Use three machine learning algorithms: RF, GBT, and NN
  - NN-based algorithm performs the most accuracy prediction results
- Simulation results show that more accurate prediction leads to fewer failed jobs

Future Work

- For predicting the resource availability
  - Collect more user/device information as features
  - Adopt more machine learning algorithms suitable for time series prediction
- For the multimedia fog computing platform
  - Study the scheduling problem
  - Deal with the dynamicity of fog users' requests
  - Provide QoS guarantees on the resource limited fog devices