MULTI-FACTOR DESIGNED EXPERIMENTS FOR COMPUTATIONAL PERFORMANCE MEASUREMENT ANALYSIS (CPM&A) OF PARALLEL DISTRIBUTED BIG DATA SYSTEMS AND SPAMHAUS DATA ANALYSIS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Yuying Song

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. William S. Cleveland, Chair

Department of Statistics

Dr. Jun Xie

Department of Statistics

Dr. Chuanhai Liu

Department of Statistics

Dr. Wenwen Tung

Department of Earth

Approved by:

Dr. Jun Xie

Head of the School Graduate Program

Faith, Hope and Love

For My lovely Family

ACKNOWLEDGMENTS

It is always a magnificent and challenging experience to earn a doctorate degree. It was hardly possible to be where I am without many people's help.

I would like to express my greatest appreciation to my advisor Dr. William S. Cleveland who is an outstanding supervisor and a distinguished researcher. He is very patience in teaching and knowledgeable in big data analysis and data visualization. His instructions deeply influence me on visualizing data and carrying out the data analysis in my current and future work.

I would also like to thank my other committee members Professor Wenwen Tung, Professor Jun Xie and Professor Chuanhai Liu for their valuable and constructive suggestions in the research work and my graduate life.

I wish to acknowledge the help provided by Professor Bowei Xi for her great help, discussions and instructions during my research.

I want to express my appreciation to our star staff: Doug Craig for his valuable and constructive suggestions during the development of this research work.

I want to thank my former and present colleagues: Jeff, Ryan, Xiaosu, Philip, Barret, Ashrith, Qi, Aritra for all the help and discussions along the research and teaching.

I also want to thank all my friends at Purdue for all the good memories: working on projects, preparing for finals, hot pot parties, game night. Thank you for being there in my life.

I would like to express my very great appreciation to my family: my husband Hai Dong, my daughter Ella and my parents. Thank you for all the love and support.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

ABBREVIATIONS

D&R	Divide and Recombine
KB	Kilobyte
MB	Megabyte
GB	Gigabyte
TB	Terabyte
HDFS	Hadoop Distributed File System
RHIPE	R and Hadoop Integrated Programming Environment
API	Application Programming Interface
DNS	Domain Name System
DNSBLs	DNS-based Blocklists

ABSTRACT

Song, Yuying. Ph.D., Purdue University, December 2020. Multi-Factor Designed Experiments for Computational Performance Measurement Analysis (CPM&A) of Parallel Distributed Big Data Systems and Spamhaus Data Analysis. Major Professor: William S. Cleveland.

Big Data analysis is of great challenges in practice. The data set sizes will grow quickly. And analysing big dataset in a timely manner is critical. In fact, computational performance depends very heavily, not just on size, but on the computational complexity of the analytic routines used in the analysis. Datasets that have computational challenges have a very wide range of sizes. Furthermore, the hardware power available to the data analyst is also an important factor. Improvements in performance from better measurement and analysis can be provided for wide ranges of dataset size, computational complexity, and hardware power. In my first part of dissertation, I will develop an overall framework of practices that can provide guidance to big data platform performance measurement. It has two main impacts, one is to provide a rigorous and comprehensive performance experiment framework on computing methods and systems for big-data analytics by bringing the statistical thinking, the statistical experimental design, and the statistical modeling to the research community. The other is to enhance performance improvement by providing a much better understanding of performance.

In the second part of the dissertation, I will analyze a 1TB Spamhaus Blacklisted Data by applying Divide and Recombine on Hadoop. The spamhaus data was collected from the Stanford mirror of the Spamhaus Internet IP address and domain name blacklist site. The Spamhaus service classifies IP addresses and domain names as blacklisted or not based on many sources of information and many factors such as being a major conduit for spam. Queries are sent to the site about the status of an

IP address or domain name, whether it is blacklisted or not, and if blacklisted the cause. The processed data consist of values of 13 variables for each of 13,178,080,366 queries during 8 months. Subject matter divisions will be carried out and the blacklisted properties will be analyzed. The data were analyzed on Professor Cleveland's 10-node cluster, wsc; it has a little over 1 TB of memory and 200 cores. An important property of the blacklisting and its cause were discovered.

1. MULTI-FACTOR DESIGNED EXPERIMENTS FOR COMPUTATIONAL PERFORMANCE MEASUREMENT ANALYSIS (CPM&A) OF PARALLEL DISTRIBUTED BIG DATA SYSTEMS

Big Data analysis is of great challenges in practice. The data set sizes will grow quickly. And analyzing big dataset in a timely manner is critical. In fact, computational performance replies greatly, not just on input size, but also on the analytic methods' computational complexity of the analysis. Datasets that have computational challenges have a very wide range of sizes. Additionally, the power of hardware available for the computation is also an important factor. Improvements in computational performance from better measurement and analysis can be provided for wide ranges of dataset size, computational complexity, and hardware power. This project will develop a general framework of practices that can provide guidance to big data platform performance measurement. There are two main impacts, one is to provide a rigorous performance experiment framework on computing methods and systems for big-data analysis by bringing the statistical thinking, the statistical experimental design, and the statistical modeling to the research community. The other is to enhance potential performance improvement by providing a much better understanding of computations and tuning parameters interactions.

For the data analyst, what matters is the elapsed times of analytic methods of big data. The common practice is benchmarks to measure the CPM&A of big data distributed system, which almost always do not take account of the factors, nor do they measure run time of analytic methods. For example, CPM&A is often done using benchmarks such as sort. We want sort to run fast, for analytic methods that depend on sort, any improvement for the performance of an analytic method benefits analysis. However our goal is improvements for many analytic methods when confronted with big data. To optimize the computational performance, we need to carry out rigorous, statistically designed multi-factor experiments for CPM&A on analytic methods of big data. Divide and Recombine (D&R) is a very popular statistical method in big data analysis. And Hadoop, the parallel distributed system will be the big data platform for our performance measurement. There are many factors that can affect performance, and those factors have interactions that are complex. This calls for full-factorial experiments focused on D&R with Hadoop. And the studied response for our experiments is elapsed times of analytic routines.

1.1 D&R for Big Data Analysis

1.1.1 D&R Statistical Theory and Method

This Performance experiment project is focused on D&R ([1] [2]) statistical method and Hadoop platform. D&R is a statistical approach to the analysis of big data. If the dataset size is too large to fit into the memory, it will be very difficult to analyze or draw any useful conclusions from the input. Instead, D&R provides a more feasible and applicable way to analyze big data. First, the original input dataset will be divided to small subsets by some division methods. This step is called D[dr] computation. Second, the analytic methods will be applied to each subset independently without any communications among the subset. This is called A[dr] computations. Lastly, a recombination step will be carried out to each output to get a D&R result. There can be communications between the outputs, but often there can be components that are embarrassingly parallel. This step is the R[dr] computation. Figure 1.1 on page 3 illustrates the D&R framework([3]). It has many applications in time series models, MCMC simulations, Spatial and Seasonal-Temporal Modeling and other statistical models ([4], [5], [6], [7]).

There are many ways to carry out D[dr] computations. One major class of division methods is by conditioning-variable division. In very many cases, it is natural to divide the data based on the subject matter. The data are divided by conditioning

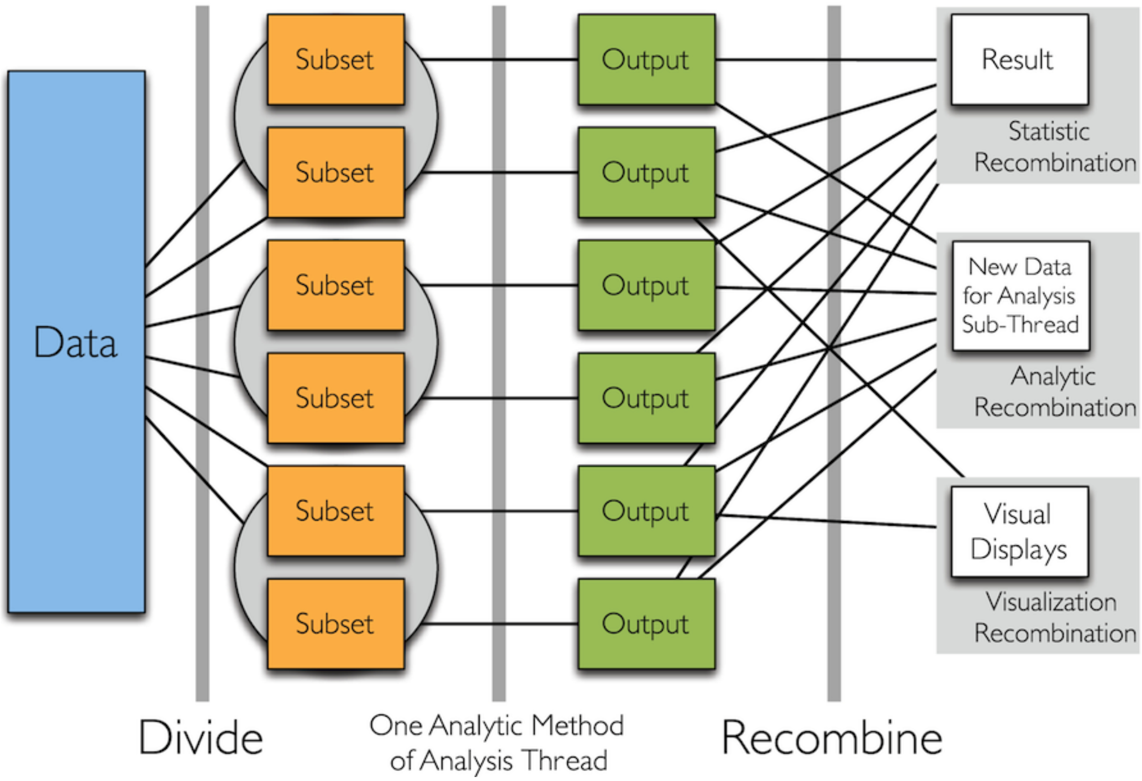

Fig. 1.1.: Divide and Recombine structure diagram

on the values of variables important to the analysis. For example if the data are 50 years of variables for 20,000 banks in the U.S., it makes sense to divide the data by bank and by time. Actually, this strategy of analysis has been used in the past in many ways because it makes sense for data of any size [6], [7]. For D&R, it is also done for computational performance. Another major class of division is sampling division. Each subset is viewed as a sample of the data. Subsets are replicate samples. For example, the division method can be random sampling. In this case, we seek a single result for all of the data. One place this arises is when subsets from conditioning variable division are too large. When it is needed, sampling division is critical, but in practice, it is used far less than conditioning-variable division.

The A[dr] computations are selected by the researches and analysts based on different projects. There are no restrictions to the analytic methods applied to the division subsets. This is called embarrassingly parallel computation because there is no any communications between the subsets in the A[dr] step.

The R[dr] computations have mainly three different output formats. The first output type can be calculated by applying statistical recombination to each output of the A[dr] step. The second format, the most common, is to return a new division dataset which the further analysis will be based on. The last common format is draw statistical conclusions by visual display [8].

The statistical accuracy of the D&R result is typically less than that of the direct all-data result. The D&R research in statistical theory seeks to maximize the statistical accuracy of D&R results [7].

1.1.2 DeltaRho Platform and Hadoop

The DeltaRho software platform [3] applies the D&R theory into practice. It combines R [9] as the front end and Hadoop [10], [11] as the backend to enable the big data analysis. The front end R is which the data analysts use to write code for modeling and analysis. Its back end Hadoop is a platform for distributed computing and large-scale data process. *Rhipe* R package, [12], [13], the R and Hadoop Integrated Programming Environment provides communications between R and Hadoop. The analyst specifies R code for the three D&R tasks: D[dr], A[dr], and R[dr] using R commands through *Rhipe* functions. And Hadoop will carry out the real computations.

The Hadoop ecosystem mainly consists of three components: YARN, HDFS and MapReduce. YARN is the resource manager in a computing cluster. It is a new feature introduced to Hadoop 2.0. It can dynamically allocate different resources and schedule the application processing for large volume of data processing. Clients submit MapReduce jobs and YARN decides when and what nodes are available for those tasks. HDFS, Hadoop Distributed File System [14], is the storage unit of Hadoop. It splits the big data as a sequence of blocks and distributes them across the cluster. In addition, each block is replicated for fault tolerance, that means, if some blocks of data are lost due to the technical issues of some nodes or servers, HDFS can recover the missing pieces from the block replicates saved on other nodes, making the big data storage more robust to hardware failures. YARN and HDFS are also used in Spark platform [15]. Lastly, MapReduce [16] is the computing engine. It consists of Mappers and Reducers. As mentioned in HDFS part, the Hadoop file is saved as a sequence type consisting of blocks in HDFS and each block is made up of different key-value pairs. The Map operation will take each block of the data as input and apply the user defined functions to each key-value pair and return the output key-value pairs in parallel. There is no communications among different key-value pairs or each block, making the map process embarrassingly parallel. The output key-value pairs become the input for Reduce operation. The Reduce transform is carried out to aggregate all values if they share the same key and write a new key-value pair output to the HDFS.

There are three common input types for *Rhipe*: text files, sequence files and numeric simulation types [13]. The text file is one of the most widely used data input format in the data science world where data is laid out in lines and each line is a record. Sequence files [17], consisting of key-value pairs, are designed for Hadoop. It has a great of application in Hadoop as the MapReduce input and output. Besides, it is saved in binary files, providing faster data read and writing speed than text files. As a result, it is a common practice to source the raw input text file, form a meaningful division and save the division to Sequence file by utilizing *Rhipe* functions. It is important to keep in mind that this new division operation will only be carried out once and the further data analysis will be based on the new division instead of the raw text input.

Here are the detailed steps to illustrate this workflow through *Rhipe* by connecting R and Hadoop with a text input. Imagine that the raw data is in the text format and the job is to divide the data input by a conditional variable and save the new division as the output to HDFS. Suppose the conditional variable is a timing variable *Year* without loss of generality, the output key-value pair is *Year* and its corresponding data frame sharing the same value of *Year*. The analyst needs to specify R functions for the three stages: Map, Reduce and Executions. First, the Map expression will take the Map input key-value pair, carry out the divide computation and pass the output key-value pair to the Reduce stage. The Map input key-value pair in this example is the line number identifier and every line from text file. The output key-value pair is *Year* and one-row data frame including all the information of the line record. The Map code is an R expression to take the long string of every line, split it to vectors and reform it to a dataframe format. This is the D[dr] computation. Each map will take a block of data consisting of many key-value pairs and carry out the operations above in parallel. Second, a Reduce expression is written to take the Map output key-value pairs as input and aggregate different values if they share the same key value of *Year*. The output key-value pair of the Reduce Stage is *Year* and a big dataframe including all the records of the *Year*. Both Map and Reduce stages are R expressions returning a list of arguments to be evaluate by the last step execution function. The execution function in *Rhipe* will specify data input path and output path in HDFS, take the Map expression and Reduce expression as arguments and pass all the information to Hadoop for computation. Furthermore, the execution function provides more flexibility to interact with Hadoop by allowing passing Hadoop parameters to tune the Hadoop configuration. In summary, all the code are rewritten in R commands by *Rhipe* functions but Hadoop is the platform to actually carry out the big data analysis.

1.1.3 D&R with DeltaRho Impact

D&R with DeltaRho enables deep analysis of big data. Deep analysis refers to analyzing data at their finest granularity, and not just summary statistics and provides high statistical accuracy and not having to substantially sacrifice this to cope with big data. This includes visualization, so model building and diagnostics can be used in place of blind analyses. Deep analysis also means that the analyst can use the thousands of methods of statistical models and data visualization. To date, there are 15,681 packages available through the repository maintained by Comprehensive R Archive Network [18], which makes R the perfect programming language for statistical computing and analysis. There is a vast R user community and the large community commits new model packages to R everyday. However, R is not designed for big data analysis because R can only analyze data fitting into the computer's memory. Hadoop written in Java, on the other hand, provides a powerful platform to manage the big data storage and computation but brings a challenging learning curve for many statisticians. Hence, *Rhipe* enables statisticians to write R code by taking advantage of the various implemented R packages and Hadoop platform to carry out the deep analysis of big data. D&R with DeltaRho can be used now to carry out analyses of big data. How big the dataset size depends on the hardware power of the cluster used. As the power increases the size of the data that can be feasibly and practically analyzed increases.

1.2 D&R Experiment Design

There are many critical factors that will impact the overall performance [19] [20], [21] [22] and the world of computer systems for data analysis approaches optimization by a very weak approach of changing important factors one at a time and relying on canonical benchmarks [23]. We are proposing a more rigorous approach in which we measure the time of D&R statistical computations and study many critical factors by multi-factor statistical designed experiments. This experiment framework was first developed by Jeff Li [19] on Hadoop1 and we expand it to Hadoop2 and introduce other important factors.

The experiment is based on the simulated Data input written to HDFS. And the statistical model to analyze the input data is logistic regression [24] without loss of generality. The logistic regression is a common statistical model for binary classification. It is applied to each subset by the R function *glm.fit* [25] in the A[dr] step or the map stage. The R[dr] reduce stage is to combine the fitted coefficients of each subset and return their average across the subsets. There are three sets of factors of interest: statistical factors, Hadoop factor and Hardware factor. There are two types of time response variables for this experiment, one of the object time O to read data from HDFS and load it in memory as an R object. And the other response variable is the logistic regression time L which starts the measurement after O is finished, including the logistic regression A[dr] and R[dr] computations. The summation of O and L is the total elapsed time T required to finish the whole process of the logistic regression. Details of the response variables will be explained in later sections. In as summary,different types of elapsed time are measured under various combinations of factors.

The whole experiment performance is well designed to eliminate potential system noises. Firstly, the cluster is exclusively occupied by the performance experiment, no other process or users are allowed to use it during the experiment time. Secondly, each experiment run (one combination of factor levels is one run) should be independent from each other, that is, the process of last run should not impact the next run. As a result, sleep time is introduced between different runs so that the occupied cluster resources will be released and get ready for the next run. Lastly, there are three replicates with each setting of factors for the purpose of the experiment error control. Replicates are necessary to estimate the factor effects. The performance experiment is carried out on two clusters. Cluster operation is a very complex process and there might be unknown system operations going on during the experiment. Rerunning is allowed if one run is interfered by other operations. Error diagnostics and analysis will be carefully carried out in the model building step.

1.2.1 Experiment Response Variables

To measure the CPM&A, the response variables are the elapsed time of an analytic routine. We break total elapsed time T, into two components to have a more informative way to understand the computations of big data. As a result, there are two different timing response variables. The data input is saved as sequence files on HDFS and the objective is to carry out a logistic regression with Rhipe on it. The elapsed time T is the total time it takes to finish this process. It can be broken into two parts: the first response variable Object Time O and the second response variable Logistic Time L. O measures the elapsed time required to read the raw data saved on HDFS and form key-value pairs as R objects for the Hadoop job. L time begins its measurement after O step finishes. Once the data is available as R object format in R, L time starts by carrying out the logistic regression in Map, recombining model coefficient and writing output to HDFS in Reduce. So their mathematical relationship is $T = O + L$.

It is important to notice that L is a latent response variable, which means there is no way to measure it directly since we need to have data available first before we can start the L measurement. As a result, we will measure O and T independently from different runs with assumption that L can be derived from $T = O + L$. This assumption will be carefully examined in the model building and diagnostics steps. The analysis of the two response variables O and L can help us to better understand the time distribution of the big data analysis and of vital importance to understand the different factor impact to the computation. The statistical models with deconvolution methods will be carried out for the L analysis.

1.2.2 Experiment Factors

There are three sets of experiment factors that we are interested in: Statistical Variables, Hadoop Variable and Hardware Variable. The details are discussed in the following sections.

Statistical Variables

The input data can bring challenges to computations and modeling from two perspectives: the number of observations N and the number of variables V. V consists of one binary response variable and $P = V - 1$ explanatory variables. The larger N or V is, the more complex the computation is. The data size is growing fast with the improvement of the database ability and big data with TB, PB size are quite common in the modeling nowadays. D&R provides feasible ways to analyze big data with different scales by dividing the raw data into different subsets. The typical routine for D&R is to read the raw data (often in text format) once, divide the raw data and save the new division as the sequence file format to HDFS. And the new division becomes the new input for the future analysis. The sequence file has much more prevailing advantages than text file for big data analysis. It has more efficient storage power with binary storage, suitable for key-value pair and applicable for parallel computing. As a result, the simulated sequence file with different subset divisions becomes our input in the performance experiment. That is, a sequence of key-value pairs saved as sequence file are simulated as input, where the key is the subset number and the value is subset.

The number of observations in the subset M really matters in the computation. M and V is the number of rows and number of columns for the subset data frame respectively, so the subset dataset size can be fully determined by M and V. The total number of observations N is fixed: $N = 2^{30}$ and M is a changing variable. Suppose R is the subset numbers after division, their relationship is $N = M \cdot R$. There are R key-value pairs in the input data and each value is a $M \cdot V$ data frame. So M value is directly related to the data division. The log scale is applied for the purpose of symbol simplicity:

$$
\begin{aligned}
n &= log2(N) = 30 \\
r &= log2(R) \\
m &= log2(M) \\
v &= log2(V)
\end{aligned}
$$

The Table 1.1 shows the overall data input size where the smallest input size is 64GB and the largest input is 512 GB or 0.5 TB. The Table 1.2 displays the two dataset variables and their values. The log2 scale of subset observations m has 9 values, changing from 9 to 17 with increment of value one. The log2 scale of variables v has four values, 3, 4, 5 and 6. The subset size $s = 2^{m+v+3}$ varies from 32 KB to 64 MB. The number of subsets R varies from 8,192 to 2,097,152.

Table 1.1.: Input Data Size

v	3	4	5	6
Data Size(GB)	64	128	256	512

Table 1.2.: Dataset Variable

Variable	**Value**
m	9, 10, ..., 16, 17
v	3, 4, 5, 6

In the experiment, there are $N = 2^{30}$ observations of V variables. One is the binary response variable: 0 or 1 with equal probability. And there are $V - 1$ explanatory variables where each is randomly and independently generated from normal distribution with mean 0 and variance 1. The data are simulated and written to the HDFS by D[dr] computation. It is worthy noticing that there is no relationship between the response variable and $V-1$ explanatory variables in our simple data simulation. We can develop more elegant simulation methods such as developing the response variables given a preset list of model parameters. Our current simulation provides a framework to understand the elapsed time it takes to apply a statistical model with D&R, so the simple data input simulation is sufficient to serve our elapsed time measurement purpose. Also, the data simulation time is not part of measurement of the experiment.

Hadoop Variable

There are many critical tuning parameters for a Hadoop cluster [20], [21], [22], [26], [27]. The cluster administrator will configure a Hadoop cluster and there are many parameters that users can actively interact with in their map-reduce jobs to tune the computation performance. Block size (B) is one of the most important tuning parameters in Hadoop. The Hadoop HDFS puts subsets into multiple same size blocks. B in a single disk is the minimum amount of data that can be read or written at one time. Similarly, B in HDFS is the minimum computation and storage unit where the large files in HDFS will be broken into numbers of blocks and each map task will operate on a block of data every time in parallel [10]. There are many benefits to operate on block abstractions for HDFS. Firstly, HDFS is designed to support very large files storage so it is very common that a single file is too large to save in a single disk. Instead, a large file is broken into multiple blocks and blocks can be saved across the cluster. Block abstraction is applicable in big data practice and simplifies the storage system. Furthermore, blocks works well with the fault tolerance feature of HDFS. A Hadoop cluster can be simplified as a integrity of multiple k computers and each computer has a fixed rate of failure p. If one of the computer is crashed (such as hard-disk failure, RAM crash) without any data backup, data is lost and any job depending on this job will fail. The failure rate of the cluster equals to $1-(1-p)^k$.

If a cluster consists of 1000 computers with $p = 0.1\%$ failure rate, the failure rate for the cluster is as high as 63.2%. As a result, HDFS in Hadoop2 achieves high fault tolerance by replicating blocks of data on different datanodes or machines. If any of the machines fails, the data block is still available from the copies saved on other computers.

The default block size on HDFS is 64 MB for Hadoop1 and 128 MB for Hadoop2. The HDFS block size is much larger than a computer disk block size (usually 512 byte [28]). The main reason for a large HDFS block is to reduce the cost of disk seek time and internet traffic. The total time to read a block data is the summation of seek time and transfer time where seek time is the time required to locate the file and transfer time is to read its content off disk without doing any more seeks. The transfer rate is a fixed feature determined by disk and the transfer time = File size/transfer Rate. Disk seeks are generally expensive operations on large data and here is one example to illustrate the total time operation. Suppose that the seek time is 10ms, the transfer rate is $100MB/s$, file size is 100MB and block size is 100MB, so this file is only saved into one block. The time required to access this data is $10ms + 100MB/(100MB/s) = 1.01s$. On the other hand, if the block size is small with 1MB, the 100MB input dataset will be saved into 100 blocks and the access time is increased to $100 * 10ms + 100MB/(100MB/s) = 2.0s$. Large data sets with TB or PB are very common in Hadoop framework and they are chopped into fixed size of blocks, so large block size is expected to reduce the seek time. It doesn't mean the larger B is, the better performance is. There are constrains for B maximum value from RAM and performance perspective. And Block size also impacts the parallelism of mappers. The B effects will be carefully examined in our performance experiment. In the experiment, this factor has three values: 64 MB, 128 MB, and 256 MB (Table 1.3).

Table 1.3.: Hadoop Variable

Variable	**Value**
BLK (MB)	64, 128, 256

Hardware Variable

The third set of experiment factors is Hardware factor. It includes two clusters wsc cluster and wolf cluster (Table 1.4) where the same experiment will be carried out. The wsc computation cluster is maintained by Purdue ITaP Research Computing Center(RCAC) [29], suitable for high performance computing. It consists of 10 nodes, 20 cores per node, 128GB RAM per node and 160TB total disk storage. The wolf cluster, maintained by system administrators at the Department of Statistics Purdue University, is a pseudo cluster [30] because it only has one single node. The single node wolf cluster has 24 cores, 192GB RAM in total and overall 4TB disk storage. Both clusters run Apache Hadoop2. Table 1.5 compares the different configurations of the two clusters. There are 200 cores in wsc cluster and one container is corresponding to a core. One of the containers will be assigned to launch the Application Master so there will be at most 199 containers running in parallel for a task. Similarly, wolf cluster has at most 23 containers running in parallel. The total disk storage of wsc is 40 times of that of the wolf cluster. The RAM of wsc is more than 6 times of wolf. Obviously, wsc cluster has considerably more compute power than wolf. But quantifying the difference is one of the most important goals of our experiment. It provides actionable information on how much needs to be spent on hardware for a project.

All the experiment runs are carried out on a Hadoop cluster but Rhipe protects the analyst from many details of Hadoop. The analyst does not have to manage key-value pairs, Map, and Reduce, and thinks solely in terms of division methods

Table 1.4.: Hardware Variable

Variable	**Value**
cluster	wsc, wolf

Table 1.5.: Two Cluster Settings

Software Version	**WSC**	**WOLF**
R	3.4.3	3.6.1
Hadoop	2.7.4	2.9.1
Protocol Buffer	2.5.0	2.5.0
Java	1.8.0_191	11.0.8
Rhipe	0.75.2	0.76.0
Processor	Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz	Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz
Namenodes	wscadm.rcac.purdue.edu	wolf.stat.purdue.edu
Number of Nodes	10	1
Number of Cores Per Node	20	24
RAM Per Node	128GB	192GB
Number of Disk Per Node	4	2
Storage Per Drive	4TB	2TB
Total Disk Storage	160TB	4TB
Drive Technology	Spinning	SSD
Number of Replication	3	1

and recombination methods developed as part of our statistics research. And Hadoop will carry out the computations.

Experiment Factors Summary

Knowledge of the functioning of the computational environment provides information that helps greatly in choosing the factors and their levels. However, that knowledge does not provide strong insight into interactions. We must rely on empirical study for this. For this reason the design is full factorial. There are 9 values of m, 4 values of v, 3 values of B, 2 values of *cluster*, 2 measurement types, so there are 432 combinations of the factors. Table 1.6 summarizes the three sets of experiment

Table 1.6.: Experiment Factors and Response

Variables	**Name**	**Levels**	**Value**	**Description**
Statistical Factor	m	9	9,10, ..., 17	log num of subset rows
	v	4	3,4,5,6	log num of subset columns
Hadoop Factor	B(MB)	3	64, 128, 256	HDFS block size
Hardware Factor	*cluster*	2	wsc, wolf	two clusters
Response Variables	*compute type*	2	o, l	two types of log elapsed time

factors and response variables. There are 3 replicates for each combination so there are 1,286 runs in total.

The experiment is carefully designed and carried out to eliminate the potential noises. Three replicates are necessary since there is error variation that cannot be controlled in the experiment. Most variation is eliminated by giving each run exclusive use of the hardware so there is no contention from other users or other runs. But system processes that keep the hardware running must continue. They use a variable amount of capacity on the cluster. They are complex and cannot be easily measured, so they become error variation. We expect this variation to be small in magnitude compared with the effects of the experimental factors. Cluster operation is a very complex process, including unknown system operations going on during the experiment. So re-running is allowed if one run is interfered by other operations. Error diagnostics will be carefully carried out in the model building step.

1.3 Exploratory Data Analysis

Runs with different factors combinations are carried out in the two clusters and their elapsed time O and T are recorded for the further analysis. The interactions between different factors and the main effects might be very complicated. Data visualization is the first step to analyze data and trellis display is heavily involved in the exploratory data analysis to propose plausible and reasonable model fittings.

As we discussed previously, L is a latent variable and couldn't be measured directly. We measure O and T instead and will derive the potential L format from data visualization and modeling. This requests a deconvolution model for L.

1.3.1 Elapsed Time vs Subset Rows m

The following six Figures: Figure 1.2, Figure 1.3, Figure 1.4, 1.5,Figure 1.6 and Figure 1.7 display the relationship between the log elapsed time o, t vs m conditional on different variables. We will explore the interactions of m with other factors by conditional and grouping on different variables in trellis display.

Figure 1.2 plots o, t vs m superpose on v conditional on B, *compute type* and two clusters. It has 12 panels and each panel is the elapsed time under different combinations of factors. A line is drawn by connecting average of three replicates in each combination of factors. As we can see from Figure 1.2 and within the same panel:

- o is moving slowly, almost constant for smaller m values. It increases as m increases when m gets larger.
- the relationship between t and m is more complicated. It has different interactions with different v values. For example, when $v = 6$, t is increasing as m increases. But for other v values, t first decreases as m increase and then increases.
- o and t increases as v increases.

Figure 1.3 plots o, t vs m superpose on B conditional on v, *compute type* and two clusters. It has 16 panels and each panel is the elapsed time under different combinations of factors. A line is drawn by connecting average of three replicates in each combination of factors. As we can see from Figure 1.3 and within the same panel:

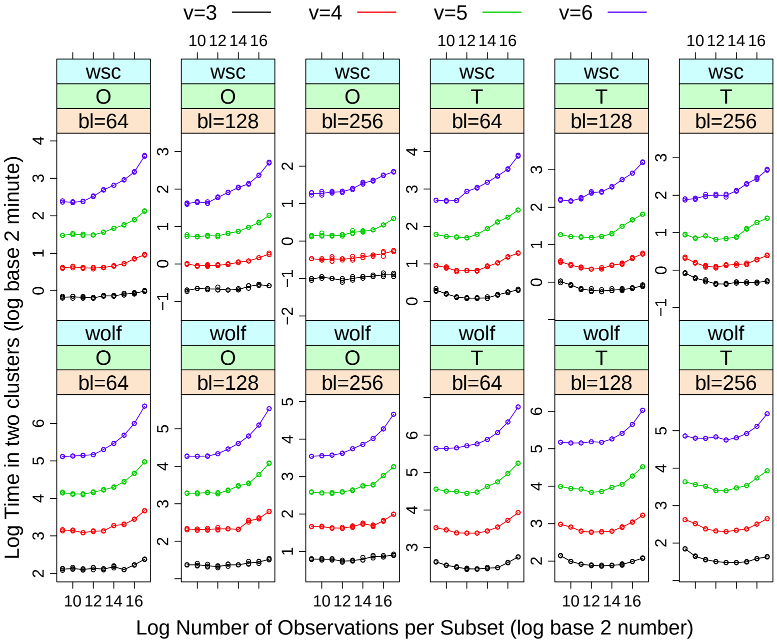

Fig. 1.2.: Log time vs m superpose v

- o is moving slowly, almost constant for smaller m values. It increases as m increases when m gets larger.

- It is clearer than Figure 1.2 that the relationship between t and m is more complicated. It has different interactions with different v values. For example, when $v = 6$, t is increasing as m increases. But for other v values (other panels), t first decreases as m increase and then increases.

- The larger the B is, the smaller o and t are.

Figure 1.4 and 1.5 plot o, t vs m superpose on *compute type* conditional on v, B for two clusters. Each has 12 panels and each panel is the elapsed time under different combinations of factors. A line is drawn by connecting average of three replicates in each combination of factors. As we can see from the two figures and within the same panel:

- o takes a large portion of t.

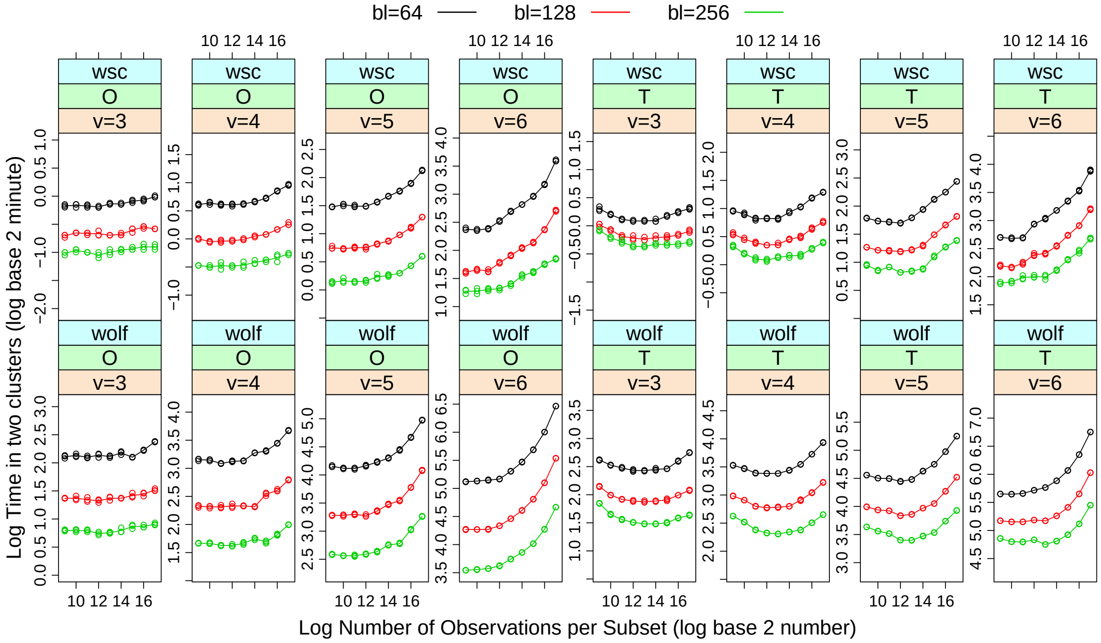

Fig. 1.3.: Log time vs m superpose B

- The portion of o in t is changing as B and m change.

Figure 1.6 and Figure 1.7 plot o, t vs m superpose on *cluster* conditional on v, B for two *compute type* o and t. Each has 12 panels and each panel is the elapsed time under different combinations of factors. A line is drawn by connecting average of three replicates in each combination of factors. As we can see from the two figures:

- *wsc* cluster is faster than *wolf* cluster in each panel.

- The elapsed time ratio of *wolf* vs *wsc* is larger as v increases across panels.

1.3.2 Elapsed Time vs Subset Columns v

Figure 1.8 shows o, t vs v superpose on m conditional on B, *compute type* and two clusters. Each has 12 panels and each panel is the elapsed time under different

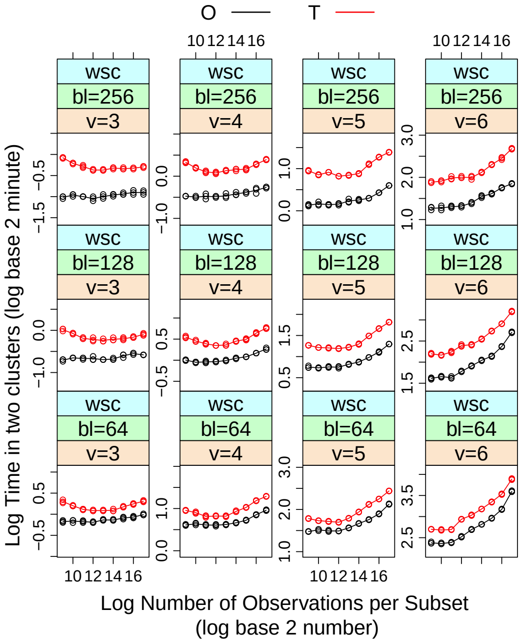

Fig. 1.4.: wsc: Log time vs m superpose compute type

combinations of factors. A line is drawn by connecting average of three replicates in each combination of factors. As we can see from Figure 1.8:

- The larger v is, the larger o and t are.
- the impact of v on o and t is interacted with m.
- the relationship of v on o and t is very closed to linear in wolf cluster.

1.3.3 Elapsed Time vs Block Size *B*

Figure 1.9 shows the relationship between time and B. It displays o, t vs B superpose on V conditional on m, *compute type* and two clusters. It has 36 panels and each panel is the elapsed time under different combinations of factors. A line is drawn by connecting average of three replicates in each combination of factors. As we can see from Figure 1.9:

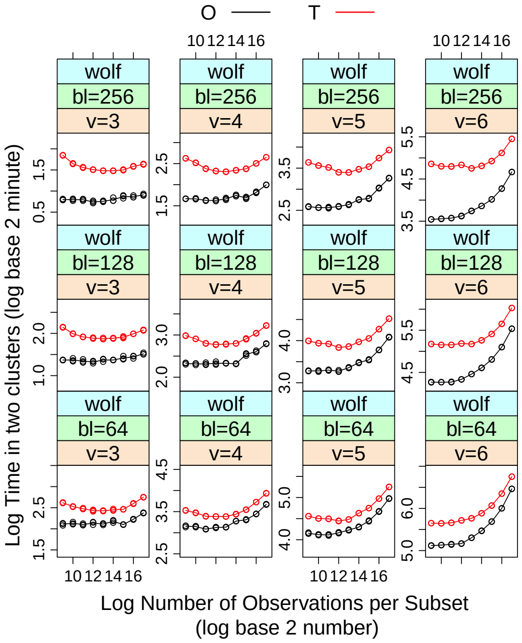

Fig. 1.5.: wolf: Log time vs m superpose compute type

- The larger B is, the smaller o and t are.
- the relationship of B on o and t is very closed to linear.

1.3.4 Elapsed Time vs Replicates

All the figures above display all the experiment results including the three replicates. The replicates almost overlap with each other or only divert by a small amount. That means the experiment noise is relatively small. Figure 1.10 and Figure 1.11 give better view of experiment noises by grouping on replicates. This is illustrated by focusing on $BLK = 256MB$ for two clusters and similar patterns hold for other block sizes. As we can see from Figure 1.10 and Figure 1.11, three lines of replicates almost overlap with each other so the noises are very small. And there is no obvious patterns in the run orders. Experiment noises and errors will be further examined in the model building steps.

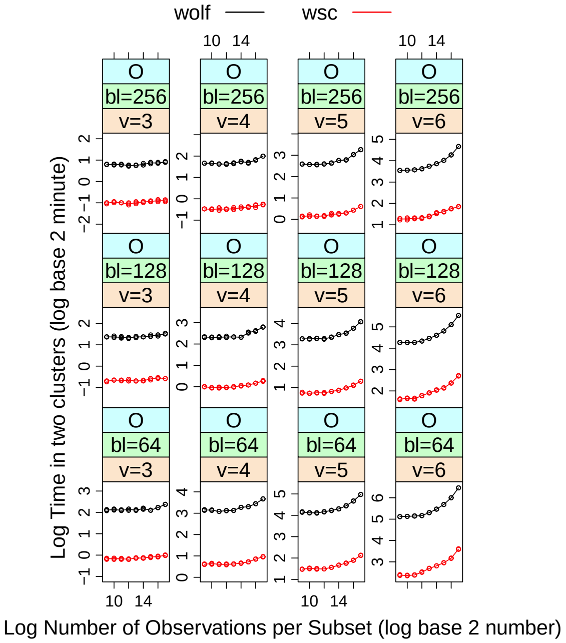

Fig. 1.6.: o vs m superpose cluster

1.4 Categorical Model Building

The exploratory analysis of the experiment results gives us good insight of potential relationships between elapsed time and factors:

- As v increases, the o and t increase monotonously with other factors hold as constant. This is expected since as v increases, the whole dataset size increases accordingly. Larger datasets will need more computation resources and take longer time. We would expect l increases as v increases, too.

- As m increase with other factors holding as constant, o first stays unchanged then begins to increase for large m values. When the total number of observations N and V are fixed, the input data size is fixed. And the total number of blocks are only subject to the value of B and one block of data can consist of multiple key-value pairs or subsets. As a result, the varying values of m would only impact the subset size and number of subsets R. The number of blocks

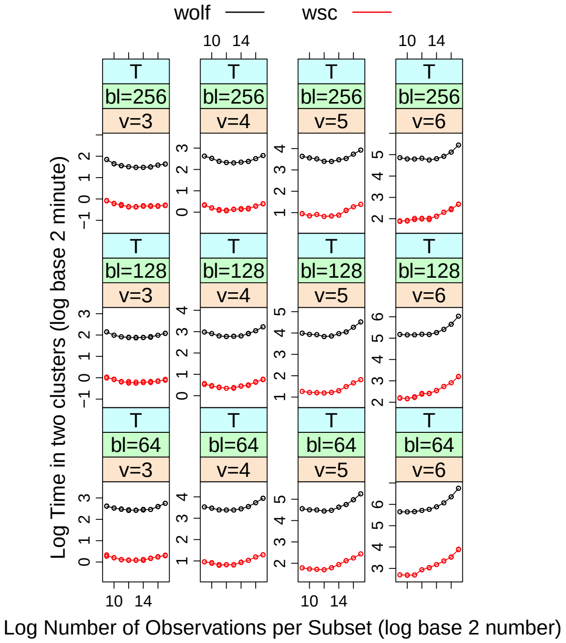

Fig. 1.7.: t vs m superpose cluster

stays the same while the number of subsets in one block would be different as m changes. The number of mappers are determined by the number of blocks so the number of mappers stay the same as m changes. When m values are relatively small, o stays unchanged. However, when m gets larger, the subset size gets larger and one subset might be spilled across two blocks and that increases the IO time and network traffic. As a result, o increases as m increases.

- The relationship between t and m is more complicated. As we can see from Figure 1.3, t first decreases as m increases then increases. This suggests l probably has a more complicated up and down relationship with m. For larger m values, it will require more memory to carry out the logistic regression which might cause the map running longer. On the other hand, some very small m values will increase the number of key-value pairs and it might increase the

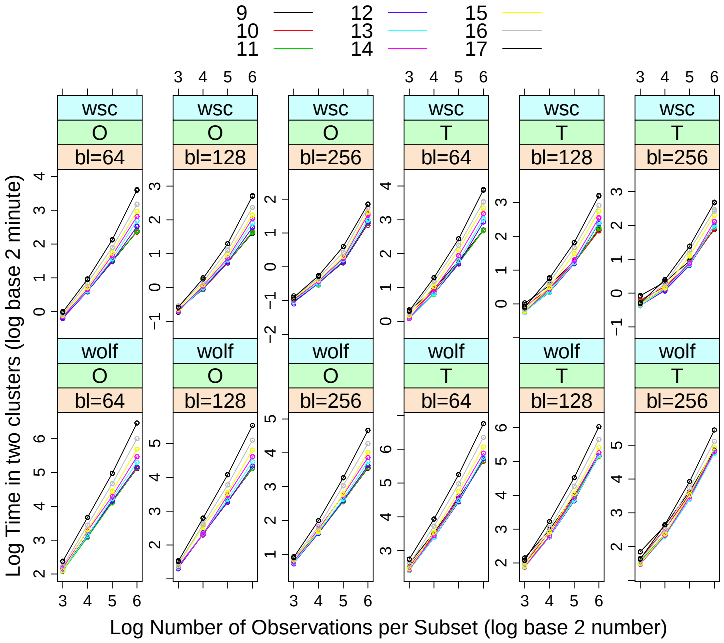

Fig. 1.8.: Log time vs v superpose m

number of intermediate files written to local disk between map and reduce stage. This in turn increase the total time.

- There is a clear interaction between m and v.
- Larger value of B will reduce both o and t.
- Factor B interacts with v on o and t.

1.4.1 Categorical Model on m and v

There are four factors with potential complicated interactions. And the hardware variable - two clusters have quite different configuration which brings more challenges to the model building. The m and v are statistical variables and it is determined by the data analyst, data set input and analytic method so We will focus on the categorical model building on m and v first. That means, we will fit a full categorical

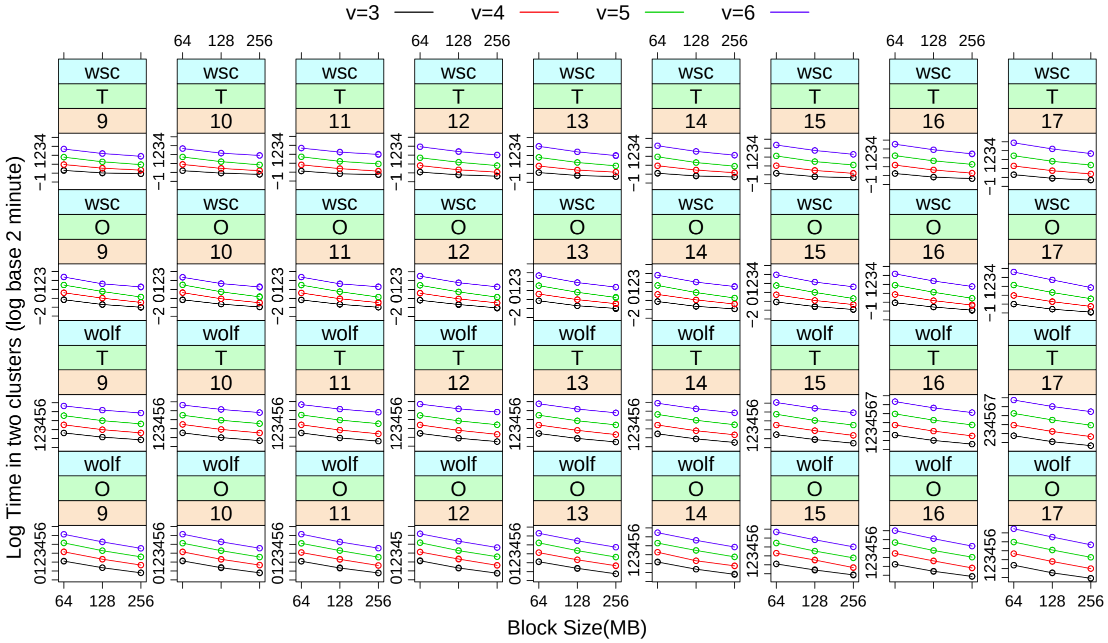

Fig. 1.9.: Log time vs B superpose v

model of o and l on m and v for each combinations of B and *clusters*. There will be a six sets of fittings for each o and t. The full categorical model is essential since we have little knowledge about the interactions.

We specify the mathematical annotations for m, v, o, l and t as following for the purpose of model simplicity.

$$m_i = log2(M_i),$$

$$v_j = log2(V_j),$$

$$o_{ijk} = log2(O_{ijk}),$$

$$l_{ijk} = log2(L_{ijk}),$$

$$t_{ijk} = log2(T_{ijk})$$

$$where\ i = [1, 2, ..., 9], m\ levels;\ j = [1, 2, 3, 4],\ v\ levels;$$

$$k = [1, 2, 3],\ replicates.$$

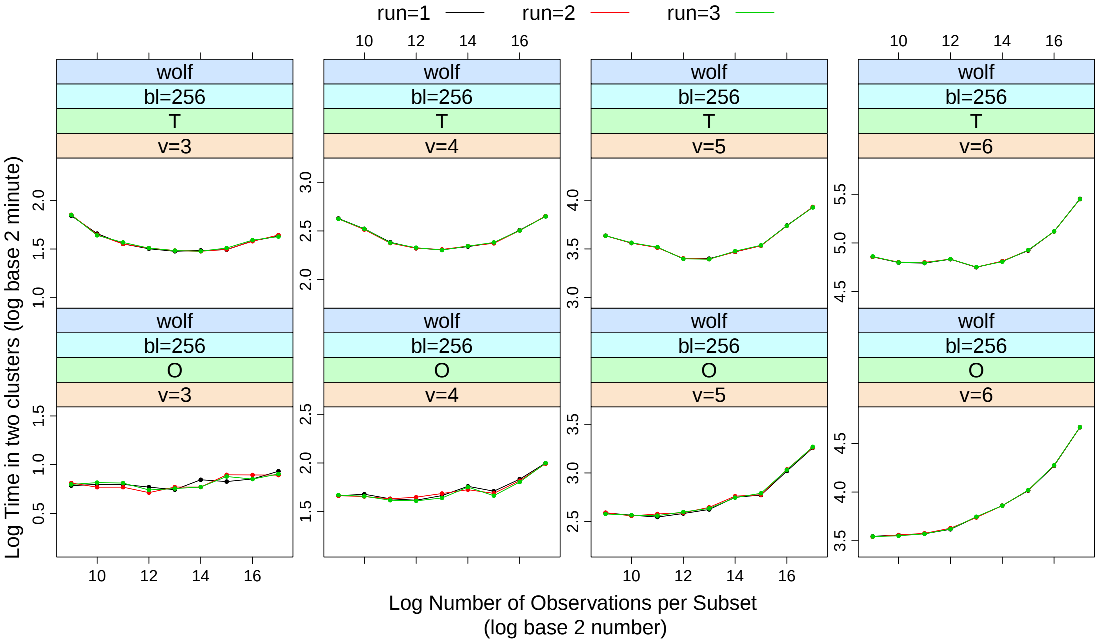

Fig. 1.10.: Log time vs m superpose *replicates*

We propose a tentative general function form for o and t:

$$O_{ijk} = \{2^{g_o(m_i,v_j)}\} \cdot 2^{\epsilon_{1,ijk}}, \tag{1.1}$$

$$T_{ijk} = \{2^{g_o(m_i,v_j)} + 2^{g_l(m_i,v_j)}\} \cdot 2^{\epsilon_{2,ijk}} \tag{1.2}$$

where:

- g_o and g_l are functions related to m and v.
- Error term $\epsilon_{1,ijk} \sim$ i.i.d N(0, σ_1^2).
- Error term $\epsilon_{2,ijk} \sim$ i.i.d N(0, σ_2^2).
- $\epsilon_{1,ijk}$ and $\epsilon_{2,ijk}$ are independent from each other.

It is worth to notice that the term $2^{(g_l(m_{ij},v_{ij}))}$ in Equation 1.2 is actually a latent variable since it quantifies the formula of L time and we couldn't observe or measure

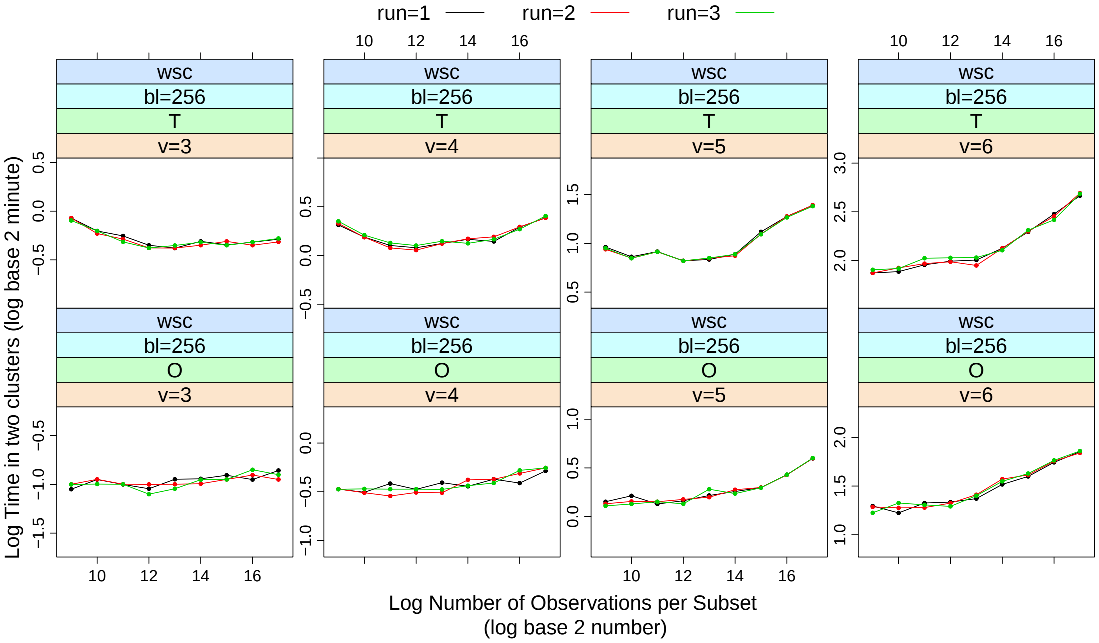

Fig. 1.11.: Log time vs m superpose *replicates*

it directly. However, we have O measurement and T measurement that O is part of T. So the format of $2^{(g_l(m_{ij},v_{ij}))}$ can be derived from our data. We will examine this assumption carefully in the model fitting and diagnostics steps.

In the categorical model fitting with m and v, we allow full interactions between m and v, thus we fit a full categorical models for each of the six combinations of B and *cluster*:

$$g_o(m_i, v_j) = \mu^o + \tau_i^o + \beta_j^o + (\tau\beta)_{ij}^o \tag{1.3}$$

$$g_l(m_i, v_j) = \mu^l + \tau_i^l + \beta_j^l + (\tau\beta)_{ij}^l \tag{1.4}$$

The model parameter fit is achieved by minimizing the sum of square errors in the Equation 1.7 of o and t with fitted g_o and g_l. There are three replicates for each of the six combinations of block size factor and hardware factor, combining two compute types, there are $9 \cdot 4 \cdot 3 \cdot 2 = 216$ runs in total. Hence we have enough observations to fit the $9 \cdot 4 \cdot 2 = 72$ parameters in the full categorical model 1.3 and 1.4.

$$SSE_o = \sum_{i,j,k=1} (o_{ijk} - \hat{g}_o(m_{ijk}, v_{jjk}))^2 \tag{1.5}$$

$$SSE_t = \sum_{i,j,k=1} [t_{ijk} - \log_2(2^{\hat{g_o}(m_{ijk},v_{ijk})} + 2^{\hat{g_l}(m_{ijk},v_{ijk})})]^2 \tag{1.6}$$

$$SSE = SSE_o + SSE_t \tag{1.7}$$

1.4.2 Model Diagnostics

The categorical models are fitted and we carry out the model diagnostics to evaluate the model assumptions and goodness of fit. Figure 1.12 and 1.13 show the residual plots. There is no obvious patterns from the residual plots: residual plots are distributed symmetrically around horizontal line $y = 0$ and constant variance assumptions hold. Figure 1.14 displays the normal quantile distribution plot of residuals: normality holds for most panels. There are several big outliers or extreme values in the tail for wolf cluster that divert a little big from the normality assumption. But the errors are small in magnitudes so we will skip the further interventions.

1.4.3 Model Fitting

The model diagnostics show no lack of fit for our fitted categorical model. We will check the o, t and l fit and different factors effects and their interactions in the following sections.

o Fit and *t* Fit in the Categorical Model

Figure 1.15, 1.16, 1.17 and 1.18 compare the fitted value with the actual. The fitted values are falling in between the three replicates so there is no obvious lack of fit. Figure 1.19 displays the fitted o and t vs m superpose on v conditional on B,*cluster* and *compute type*. We can draw the same conclusion as the exploratory data analysis section:

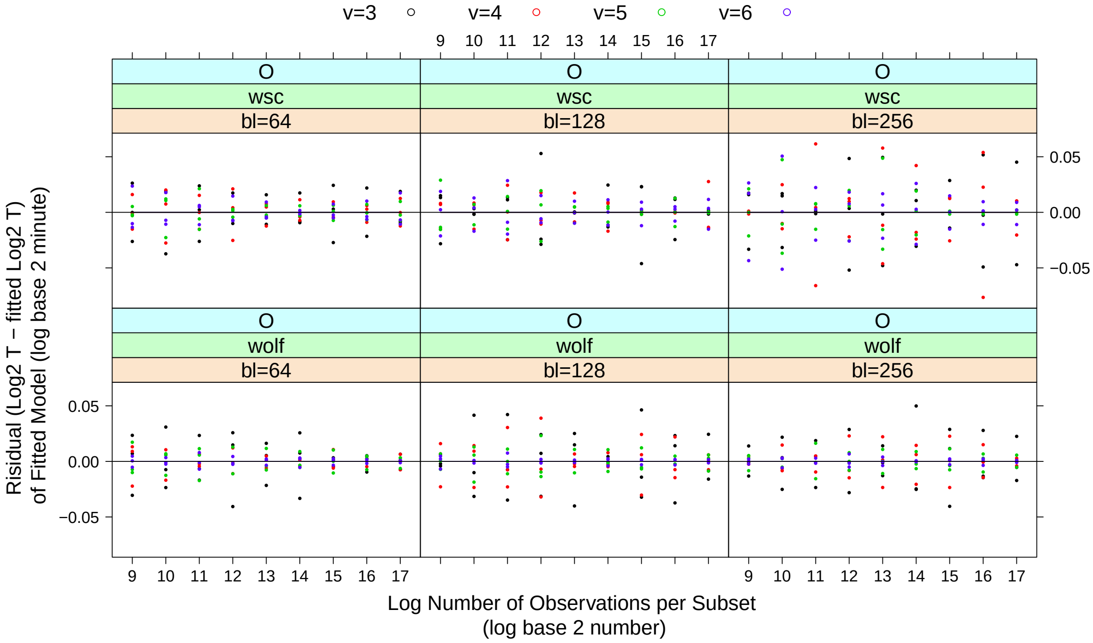

Fig. 1.12.: o fit: Residual vs m

- o is moving slowly, almost constant for smaller m values. It increases as m increases when m gets larger.
- there is an up and down curve between t and m.
- there is an interaction effect between m and v.
- the larger v is, the larger o and t are.
- wolf cluster is slower than wsc cluster under the same setting of factors.
- the larger blocksize is, the smaller o and t are.

We can also visualize the fitted o and t vs different factors other than m or conditional on different factors to get a more clear relationship among factor effects. However, the patters are very similar to the results in the exploratory analysis so we will not provide these plots for the purpose of paper length control.

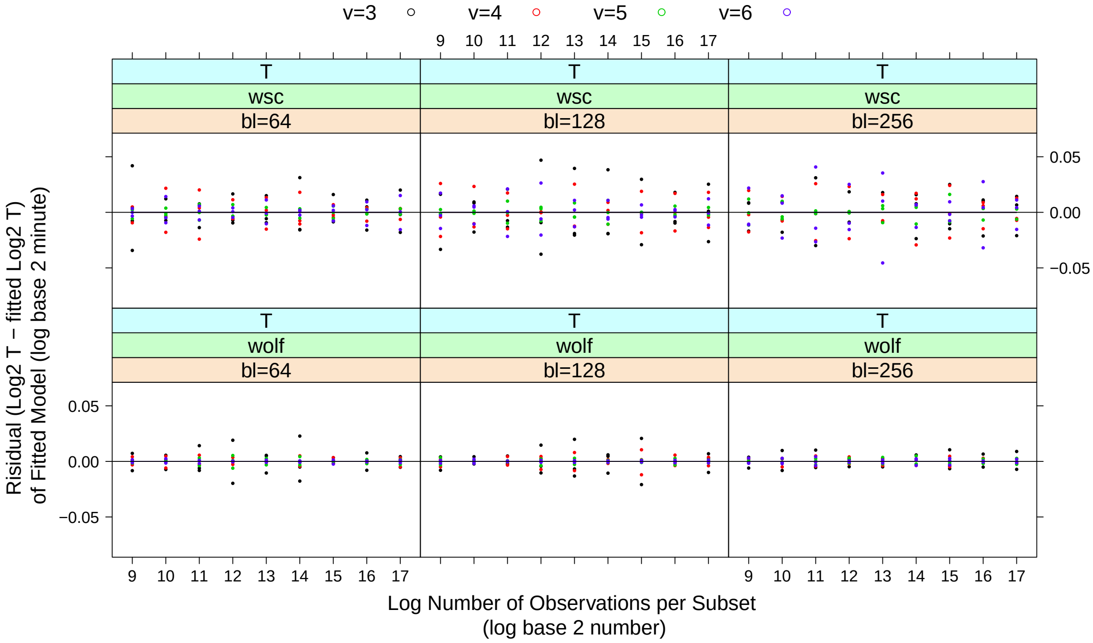

Fig. 1.13.: t fit: Residual vs m

l Fit in the Categorical Model

l fit is the focus of attention here. Figure 1.20 plots the fitted *l* vs *m* superpose on *v* conditional on *B* and *clusters*. As we can see from Figure 1.20,

- there is an interaction effect between *m* and *v* on *l*. For most values of *v*, *l* first decreases as m increases then increases, similarly to the pattern of *t*. But for wsc cluster, *l* increases slowly as *m* increases.

- there is some zig-zag pattern in *l* fits. This is probably because there are some noises when fitting the curve.

Figure 1.21 shows a more clear pattern of *l* vs *v*. The magnitude of *L* time is 1 min $\sim$ 16 min for wolf cluster and 0.25 min $\sim$ 4 min for wsc cluster. There is almost a linear relationship between *l* and *v* with some polynomial curve.

Figure 1.22 explores the relationship between *l* vs *B* where we have some very interesting findings. It has 18 panels, superpose on *v* and conditional on *m* and *clusters*.

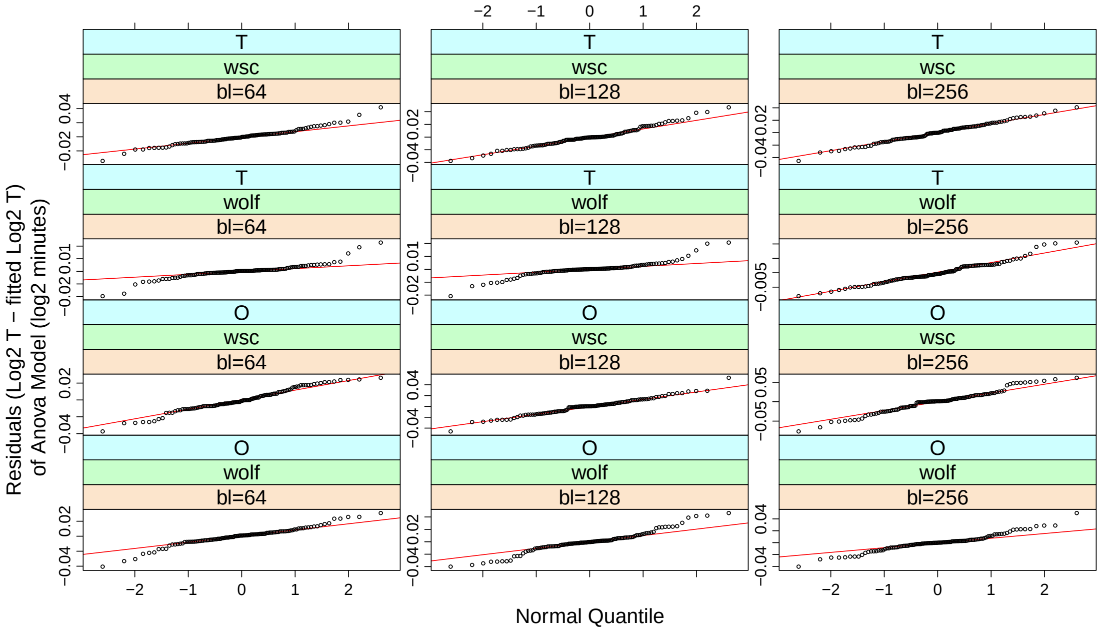

Fig. 1.14.: Categorical model Residual normal quantile

As we can see from Figure 1.22, the l fits stay as constant for three different block size values and this patterns hold across almost every panel and grouping variable v. This suggests that different B has little impact on l if other factors are unchanged. The B might plays a more import role in o, not l which means we can try to include B into functions g_o but not g_l.

1.4.4 Categorical Model on m, v and B

The similar model fitting is carried out by re-specifying the format of g_o and g_l as Equation 1.8 and Equation 1.9 below. It is worth to notice that now we have more data to fit the parameters since there is only two sets of fit for each of the cluster. That means, we need to fix (o, l) for each of wsc and wolf cluster. Hence there are $9 \cdot 4 \cdot 3 \cdot 3 \cdot 2 = 648$ observations in total to fit $9 \cdot 4 \cdot 3 + 9 \cdot 4 = 144$ parameters in the full categorical model 1.8 and 1.9.

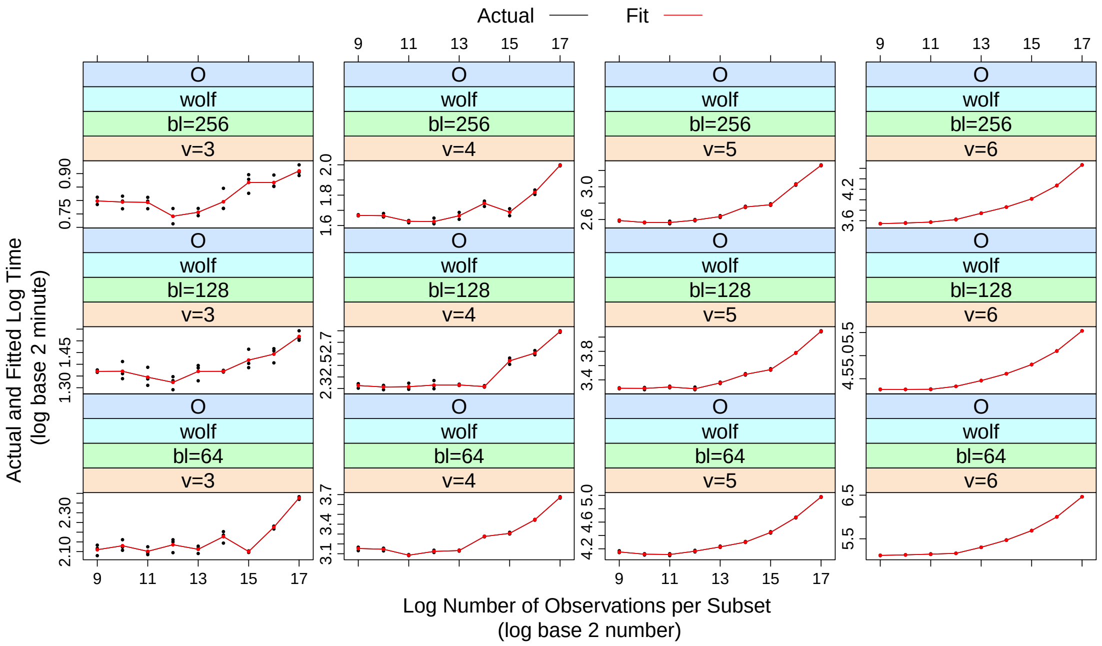

Fig. 1.15.: wolf o fits and actuals vs m

$$g_o(m_i, v_j, B_k) = \mu^o + \tau_i^o + \beta_j^o + \gamma_k^o + (\tau\beta)_{ij}^o + (\tau\gamma)_{ik}^o + (\beta\gamma)_{jk}^o + (\tau\beta\gamma)_{ijk}^o \tag{1.8}$$

$$g_l(m_i, v_j) = \mu^l + \tau_i^l + \beta_j^l + (\tau\beta)_{ij}^l \tag{1.9}$$

1.4.5 Model Diagnostics

The categorical models in Equation 1.8 and Equation 1.9 were fitted and we carry out the model diagnostics to evaluate the model assumptions and goodness of fit. Figure 1.23 shows the residual plots. There is no obvious patterns from the residual plots: residual plots are distributed symmetrically around horizontal line $y = 0$ and constant variance assumptions hold. Figure 1.24 displays the normal quantile distribution plot of residuals where we can conclude that Normality assumption is reasonable.

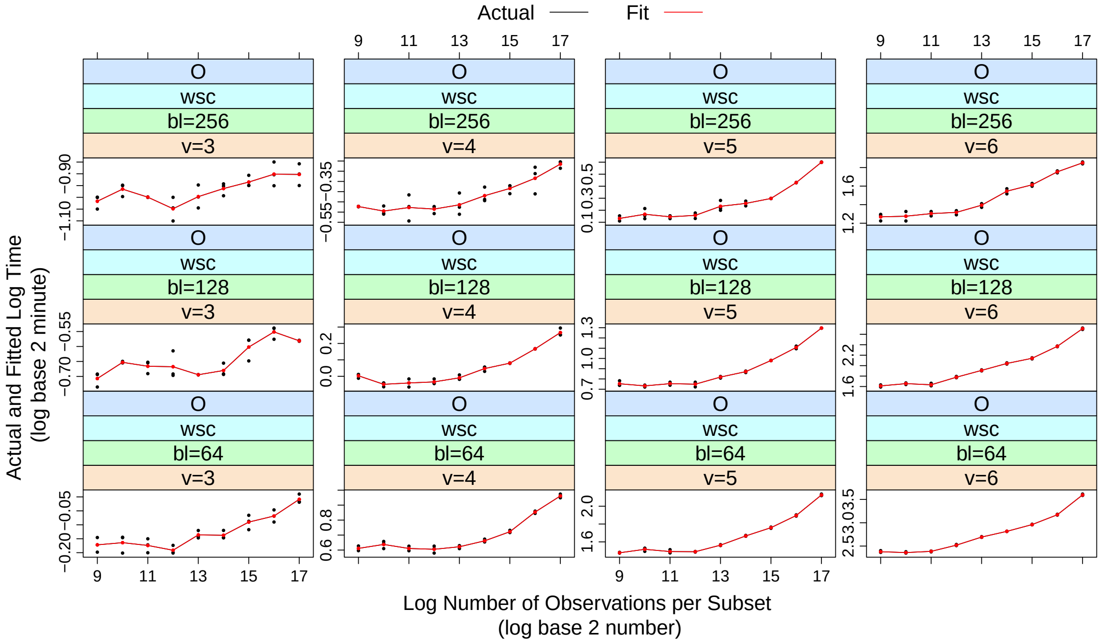

Fig. 1.16.: wsc o fits and actuals vs m

1.4.6 Model Fitting

The model diagnostics show no lack of fit for our fitted categorical model. The o and t fits are similar as before (Figure 1.25) so we will focus on l fit.

l Fit in the Categorical Model

We have more data to fit the l shape so we expect to see a smoother fit. Figure 1.26 plots the fitted l vs m superpose on v conditional *clusters*. And Figure 1.27 plots the fitted l vs v superpose on m conditional *clusters*. As we can see from Figure 1.26 and Figure 1.27:

- It seems there is a quadratic effect of m on l.
- There is almost linear relationship between l and v and there is a slight convex upward pattern.

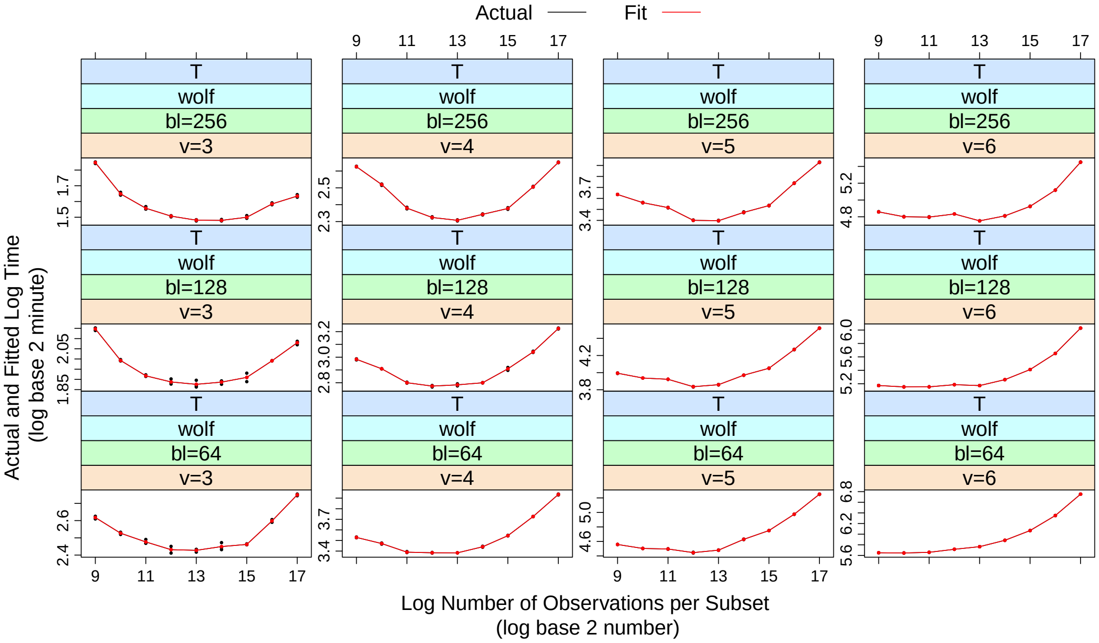

Fig. 1.17.: wolf t fits and actuals vs m

- There is an interaction between m and v on l.

1.5 Polynomial Model Building

The categorical model fitting in Section 1.4.1 and Section 1.4.4 are full categorical models and might be over fitted. Categorical models suggest a potential polynomial model between o, l and other variables. We specify a polynomial form of g_o and g_l in Equation 1.10 and Equation 1.11. Similarly, there are $9 \cdot 4 \cdot 3 \cdot 3 \cdot 2 = 648$ observations in total to fit $10 + 6 = 16$ parameters in model 1.10 and 1.11 so we have enough information for model fitting.

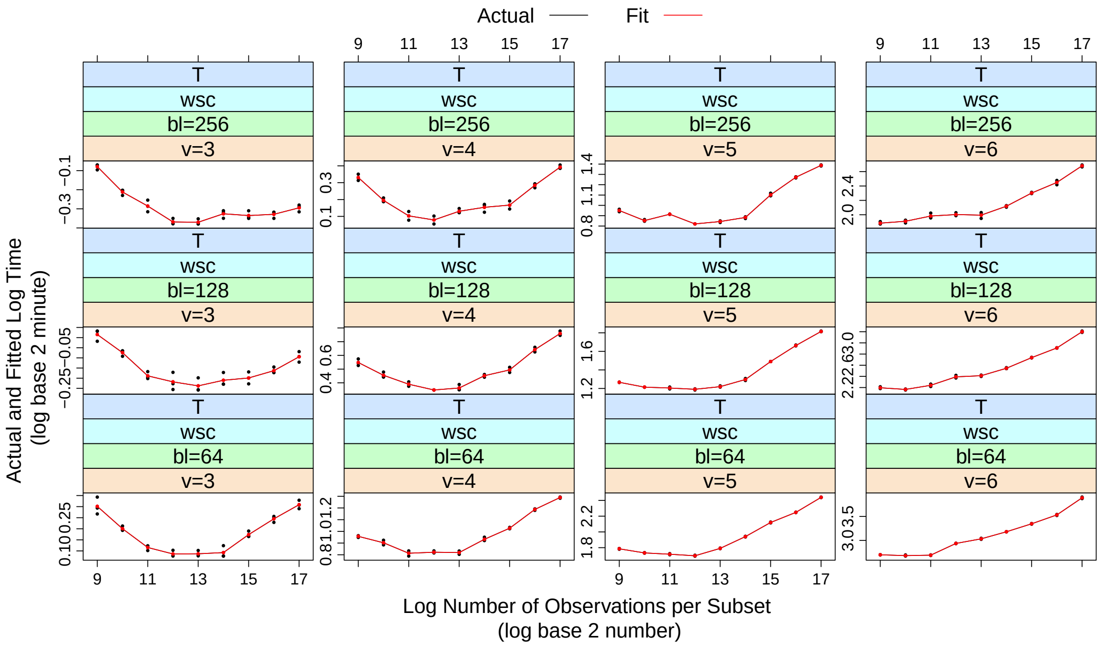

Fig. 1.18.: wsc t fits and actuals vs m

$$g_o(m, v, B) = \alpha_0 + \alpha_1 m + \alpha_2 v + \alpha_3 B + \alpha_{11} m^2 + \alpha_{22} v^2 + \alpha_{33} B^2$$
$$+ \alpha_{12} mv + \alpha_{13} mB + \alpha_{23} vB \qquad (1.10)$$
$$g_l(m, v) = \beta_0 + \beta_1 m + \beta_2 v + \beta_{11} m^2 + \beta_{22} v^2 + \beta_{12} mv \qquad (1.11)$$

where

- m, v, B are numeric.
- m, v, B are standardized to avoid the multicollinearity problem in the polynomial model fitting.

1.5.1 Polynomial Model Diagnostics

The polynomial models are fitted and model diagnostics are carried out to evaluate the model assumptions and goodness of fit. Figure 1.28 shows the residual plots.

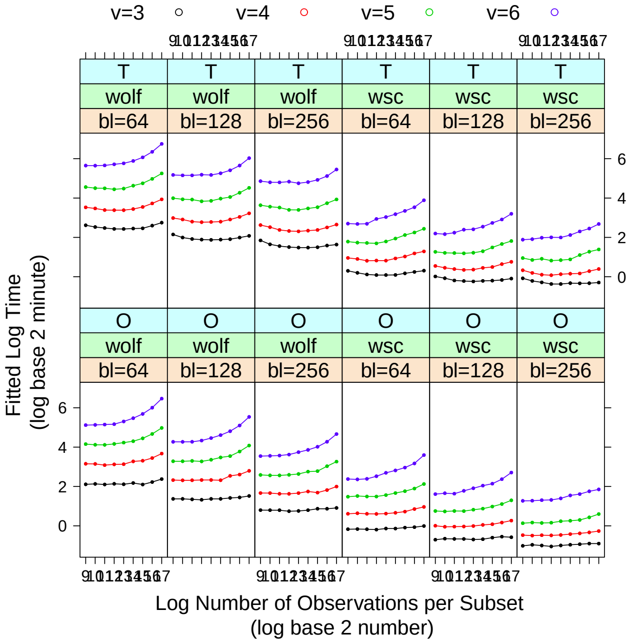

Fig. 1.19.: o,t fits vs m

There is no obvious patterns from the residual plots: residual plots are distributed symmetrically around horizontal line $y = 0$ and constant variance assumptions hold. There are several big residuals of o fits for $(m = 17, v = 6)$ and $(m = 9, v = 3)$. $(m = 17, v = 6)$ is corresponding to the largest subset in our experiment while $(m = 9, v = 3)$ defines the smallest subset. This explains these relative large residuals that the polynomial fit has some divergence of fitting for the edge cases. Figure 1.29 displays the normal quantile distribution plot of polynomial fit residuals: normality holds for most panels.

1.5.2 Polynomial Model Fitting

The model diagnostics show no lack of fit for our fitted polynomial model. We will check the o, t and l fit under different combinations of variables in the following sections.

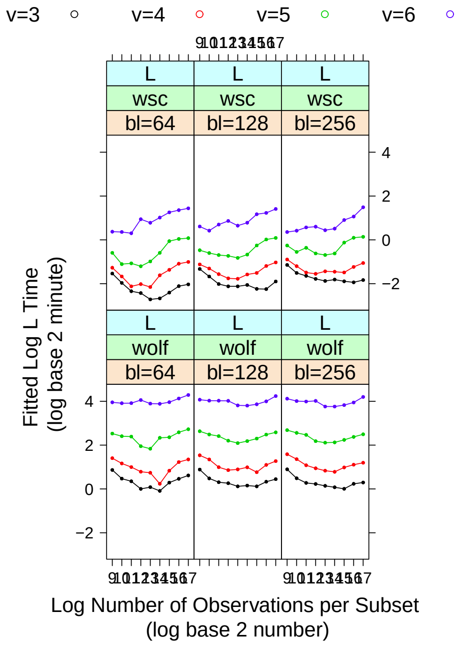

Fig. 1.20.: l fits vs m

o Fit and t Fit in the Polynomial Model

Figure 1.30 and 1.31 illustrate the fitted value with the actuals for wolf cluster. As we can see from Figure 1.30, we can see there are some small divergences of fittings with $(m = 17, v = 6)$ and $(m = 9, v = 3)$, which matches the residual plots 1.28. Figure 1.32 displays the fitted o and t vs m superpose on v conditional on B, *cluster* and *compute type*. Figure 1.33 displays the fitted o and t vs v superpose on m conditional on B, *cluster* and *compute type*. Figure 1.34 and 1.35 display the fitted o and t vs B superpose on v conditional on other variables. We can draw the similar conclusions as categorical models from figures above:

- There is a quadratic relationship between m and o. o is increasing slowly for smaller m values and increases faster when m gets larger.
- there is an up and down curve between t and m interacted with v.

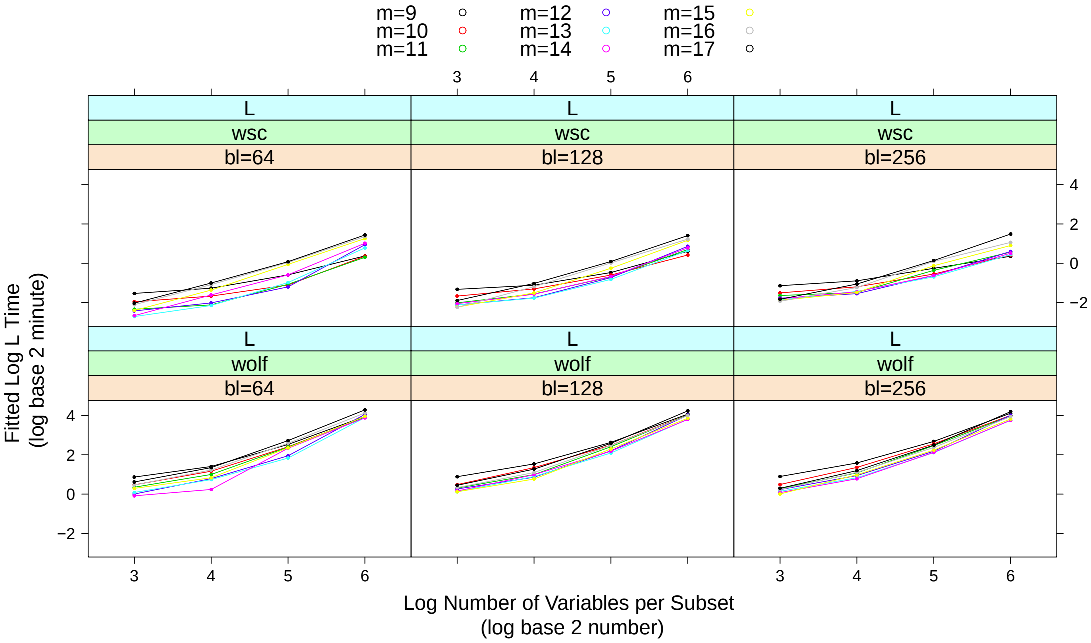

Fig. 1.21.: l fits vs v

- there is interaction effect between m and v.
- o and l increases almost linearly as v.
- the larger B is, the smaller o and t are.

l Fit in the Polynomial Model

Now we need to check the l fit from the polynomial model. Figure 1.36 plots the fitted l vs m superpose on v conditional on *clusters*. And Figure 1.37 plots the fitted l vs v superpose on m conditional on *clusters*. We have a much smoother fit and clear patters for l fit. As we can see from Figure 1.36 and 1.37:

- There is a quadratic relationship between m and l. l first decreases then increases as m increases.
- There is an interaction effect between m and v on l.

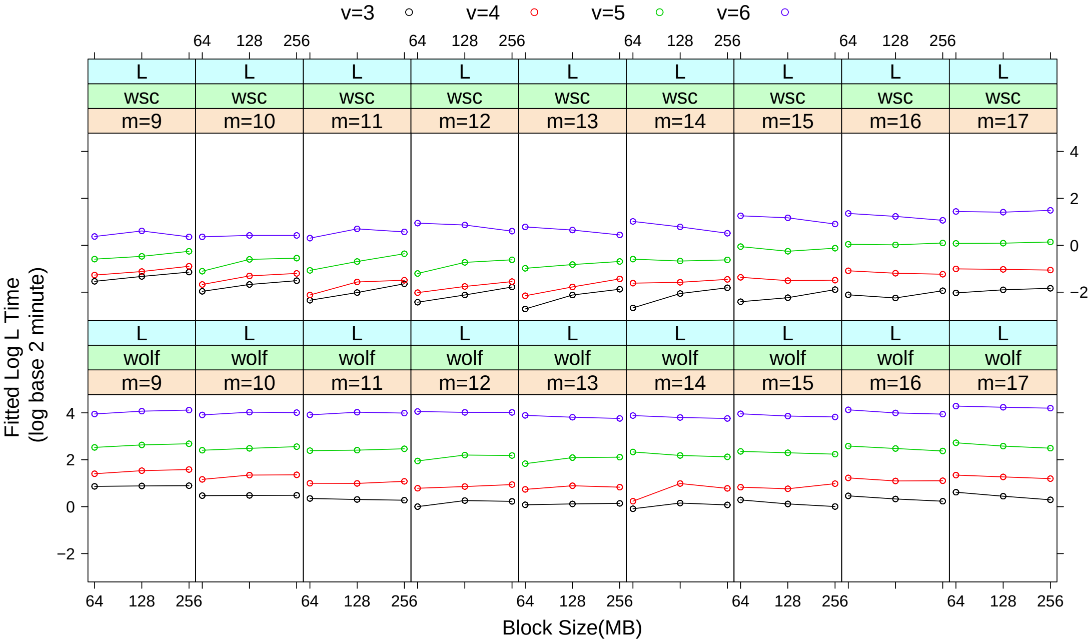

Fig. 1.22.: l fits vs B

- The relationship between v and l are not strict linear and there are a slight convex upward pattern of l value as v increases.

1.6 Experiment Factor Effects

The categorical model analysis in Section 1.4 and the polynomial model analysis in Section 1.5 provide good insight about the complicated main effects and interactions among factors with the experiment elapsed time. We will continue to use Trellis display to explore the factors effects in the following sections.

1.6.1 Block Size Effects

As we discuss before, B is one of the most important Hadoop configuration parameters and is the storage unit and computation unit. As a result, it can not only

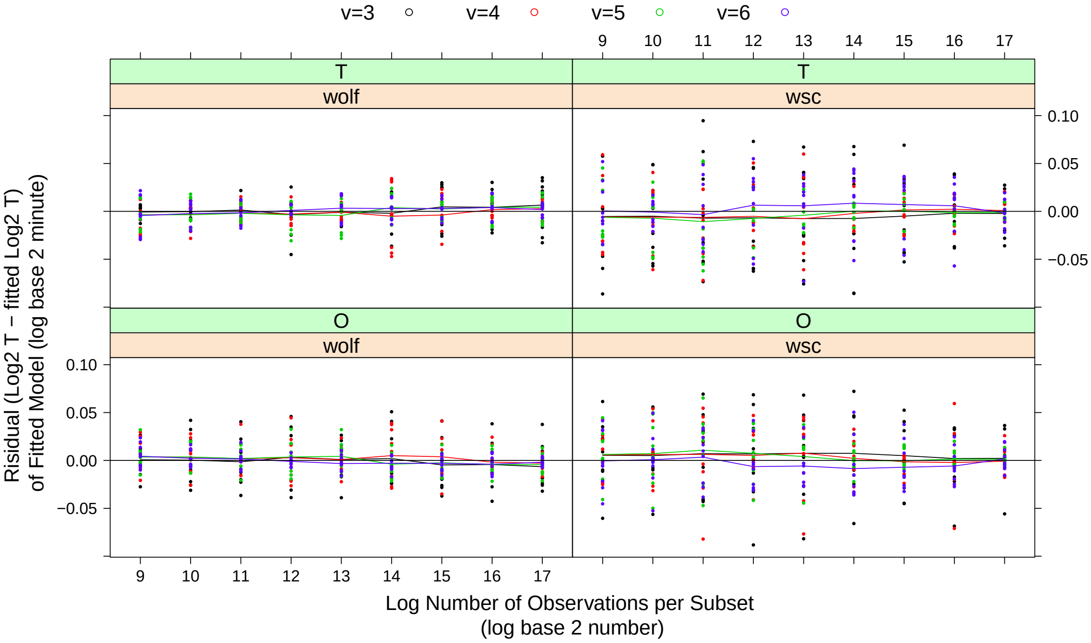

Fig. 1.23.: Residual vs m

impact how data is split and saved on HDFS but also impact the number of Map tasks computation. If we hold other factors unchanged, when B value is increased, each Mapper will read and operate on a larger piece of data. In the meanwhile, the number of mappers will decrease. Thus, B directly impacts the I/O of the HDFS and thus we would expect it to have an effect on o; And B also affects mapper numbers, so it could potentially impact l as well. However, as we discussed in Section 1.4, the B has immaterial impact on L in our experiment and the larger B will lead to a smaller O and T time. That means, the effect of B on total time T is all from its contributions to O. This is probably because our chosen analytic method in the experiment is logistic regression and the reduce step is to average the parameters from logistic regression fits. So L step is not a CPU-intensive computation so we might not be able to observe the B effects on L in the current settings.

In the experiment, we have three values of B: 64MB, 128 MB and 256 MB. We also observe a non-linear, convex decreasing curve between B and O (Figure 1.34). So we

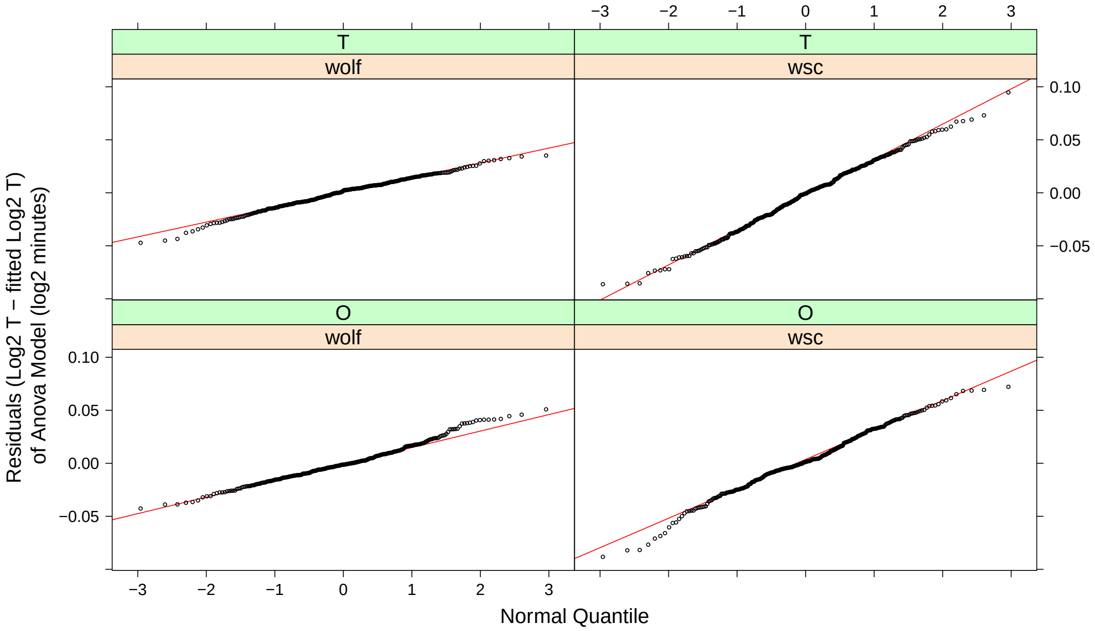

Fig. 1.24.: Residual normal quantile

will compare the effects of different values of B next. Figure 1.38 plots the difference of o between two different block size superpose on v, conditional on B and *clusters*. It has four panels and each row shares the same cluster information. This will help us to compare the different pairs of B impact: (64MB vs 128MB) and (128MB vs 256MB). It is worthy noticing that the difference between log Elapsed Time (O) is actually the log ratio of two values:

$$o_{B2} - o_{B1} = log_2(O_{B2}) - log_2(O_{B1}) = log_2(\frac{O_{B2}}{O_{B1}})$$

So if the difference $o_{B2} - o_{B1} = 1$, this means the ratio of object time $\frac{O_{B2}}{O_{B1}} = 2$, the object time O with B_1 is two times of that with B_2. As we can see from Figure 1.38:

- in each panel, the larger v is, the larger the difference is between two block size if holding m value unchanged. That means, larger value of block size has more time reduction benefit for large data set analysis than smaller ones.

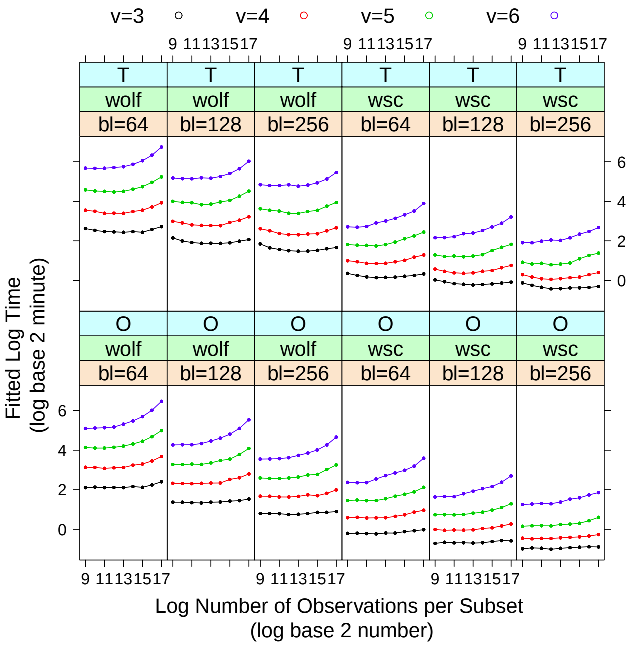

Fig. 1.25.: l fits vs m

- similarly, if we hold v value constant in each panel, there is a linear increasing relationship between the *log* difference and m. This also suggests that it is more recommended to set a large block size value if your key-value pair or subset size is large.

- if we compare the two panels in the same row, we can observe that the the line in the left panel is always above the line with the same color in the right panel. Take wsc cluster $v = 6, m = 17$ (the blue line) for example, the log ratio of time between 64MB and 128MB is close to 0.9 while the log ratio between 128MB and 256MB is close to 0.7. This means, the time reduction benefit is more significant when you change blocksize size from a smaller one to a big one. The elapsed time benefit by increasing blocksize might be limited.

Similarly, Figure 1.39 plots the difference of o between two different block size superpose on v, but with different conditional factor orders. It first conditional on

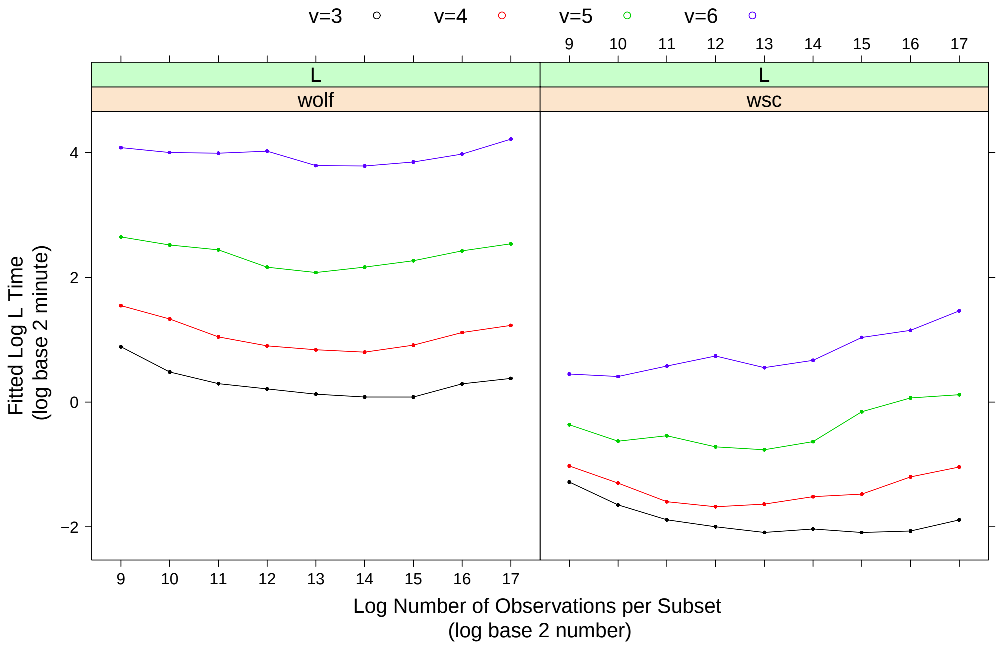

Fig. 1.26.: l fits vs m

clusters and then B. Hence it has four panels and each row shares the same information of blocksize comparison. Figure 1.39 gives us a better view to compare two clusters. Besides of the same conclusions we can draw from Figure 1.38, we can see from Figure 1.39:

- With the same values of v and m, the log ratio in the left panel is always larger than that in the right panel. This suggests that changing B has more impact on wolf cluster than wsc cluster.

Figure 1.40 plots the ratio of O and T against m superpose on B conditional on v and $cluster$. It has eight panels and each row shares the same information of cluster. It shows the information of the percentage of Object time O in the total elapsed time T. As we can see from Figure 1.40:

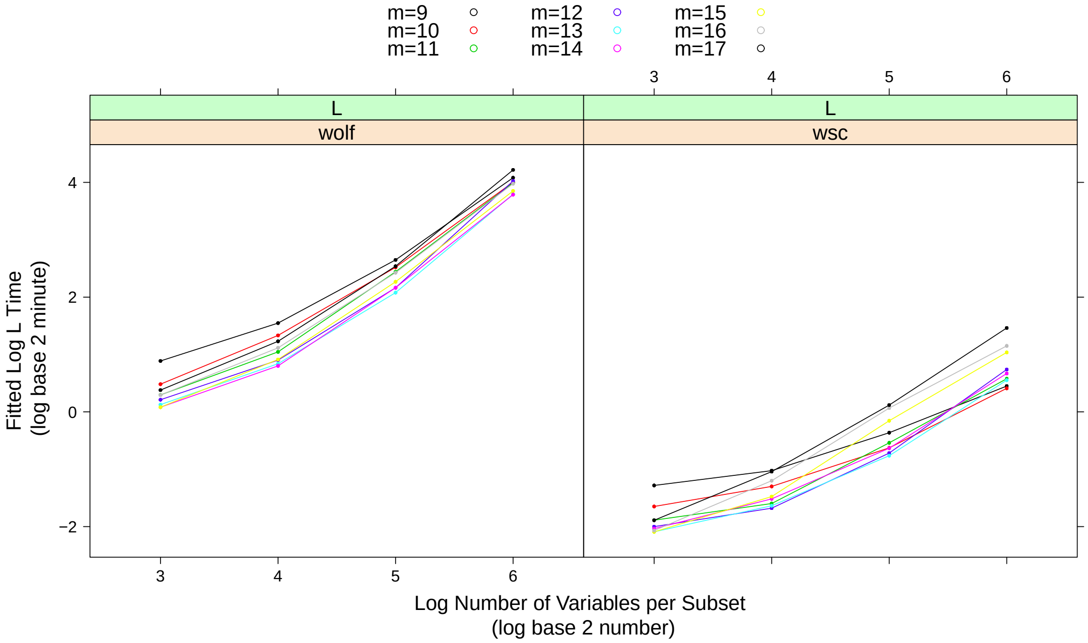

Fig. 1.27.: l fits vs v

- the larger the blocksize is, the smaller percentage of O in T is. That means, a larger blocksize will reduce the object time O and reduce the percentage in return.
- there are different change patters of percentage between two clusters. Assume we hold v constant, in the wsc cluster, the percentage has a up and down pattern as m increases. On the contrast, in the wolf cluster, there is a strictly increasing pattern of percentage as m increases. This suggests a structural difference between wsc cluster and wolf cluster.

Block Size Summary

Blocks are the storage and computation unit in Hadoop. HDFS stores files by dividing them into equal size of blocks, and distribute it among HDFS. The current default value for block size is 64MB for Hadoop1 and 128MB for Hadoop2. Our

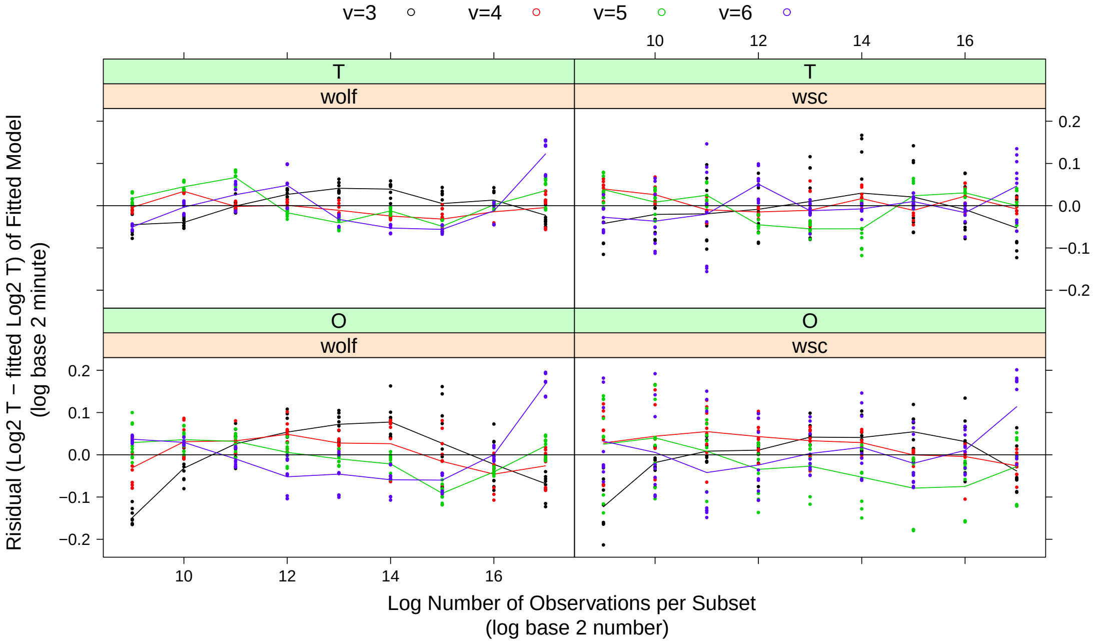

Fig. 1.28.: Poly fit: Residual vs m

experiment suggests a value of 256MB of the block size can achieve the best computation performance, especially when the raw data input size is large. Users can actively interact with the block size value by tuning Hadoop parameter *dfs.blocksize* in the task. Increasing block size brings a lot of computational benefits but it doesn't mean its value can be increased without limitations. Each mapper is running on a container and one mapper is reading one block of data. So the block size should not be larger than the RAM allocated to each container. Furthermore, the total RAM of a cluster is a fixed amount and it is configured to allocate to each container and other system operations. If the RAM per container is increased, the number of containers running in parallel at one time are reduced, which in turn might increase the total elapsed time.

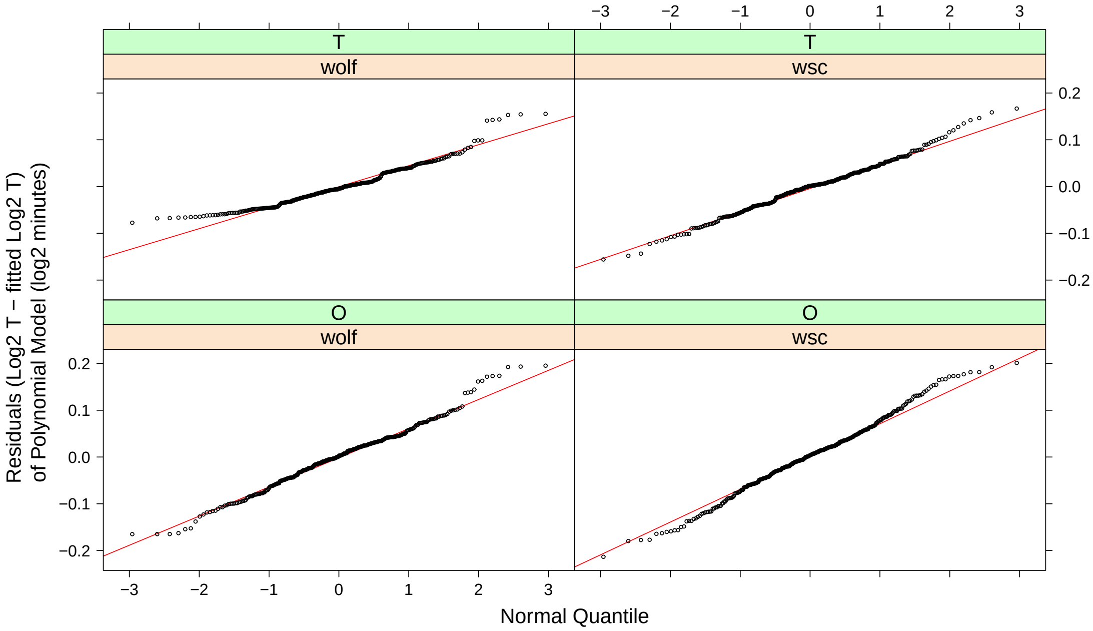

Fig. 1.29.: Residual normal quantile

1.6.2 Hardware factor Effects

We have seen the similarities and differences in the computational performance between wsc and wolf cluster in the previous sections. There are a lot of structural difference between wsc and wolf clusters (Table 1.5) where wolf is a small one-node sudo cluster compared with wsc cluster. The comparison between clusters will give good insights that one node cluster can takes care of very large data computation. Figure 1.41 and Figure 1.42 visualize and compare the two clusters effects.

Figure 1.41 plots the ratio of fitted time between wolf and wsc against m superpose on v and *compute type*. It has six panels and each row shares the same information of *compute type*. As we can see from Figure 1.41:

- the ratio range for T between wolf cluster and wsc cluster is between $4 \sim 8$. This means, wsc cluster has better performance than wolf cluster as expected

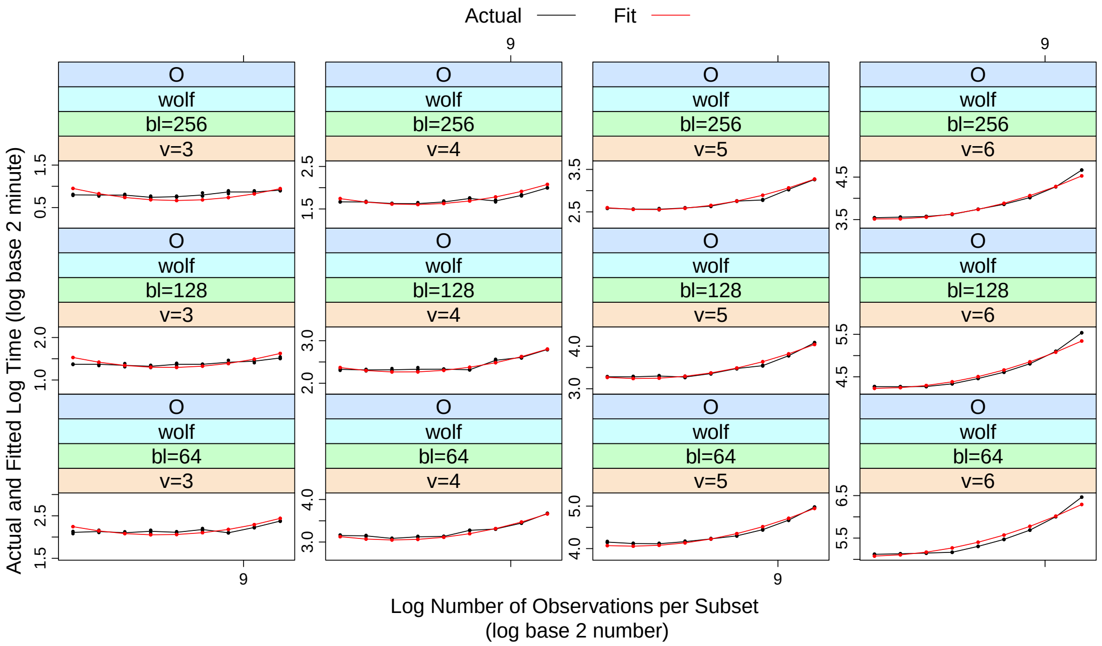

Fig. 1.30.: wolf o fits and actuals vs m

and the elapsed time T in wolf cluster is 4 times or 8 times the speed of that in *wsc cluster* among different combinations of other factors.

- the larger the v is, the faster of wsc cluster than wolf cluster is. This means, wsc cluster has more computation advantage than wsc cluster when analyzing large data set.

Figure 1.42 plots the ratio of fitted time between wolf and wsc against m superpose on B and *compute type*. It has six panels and each row shares the same information of *compute type*. As we can see from Figure 1.42:

- Similarly, the larger the v is, the faster of wsc cluster than wolf cluster is by comparing across panels.
- Within the most panels, the larger the B is, the smaller the ratio is. This suggests, increasing B will narrow the computation speed gap between wolf cluster

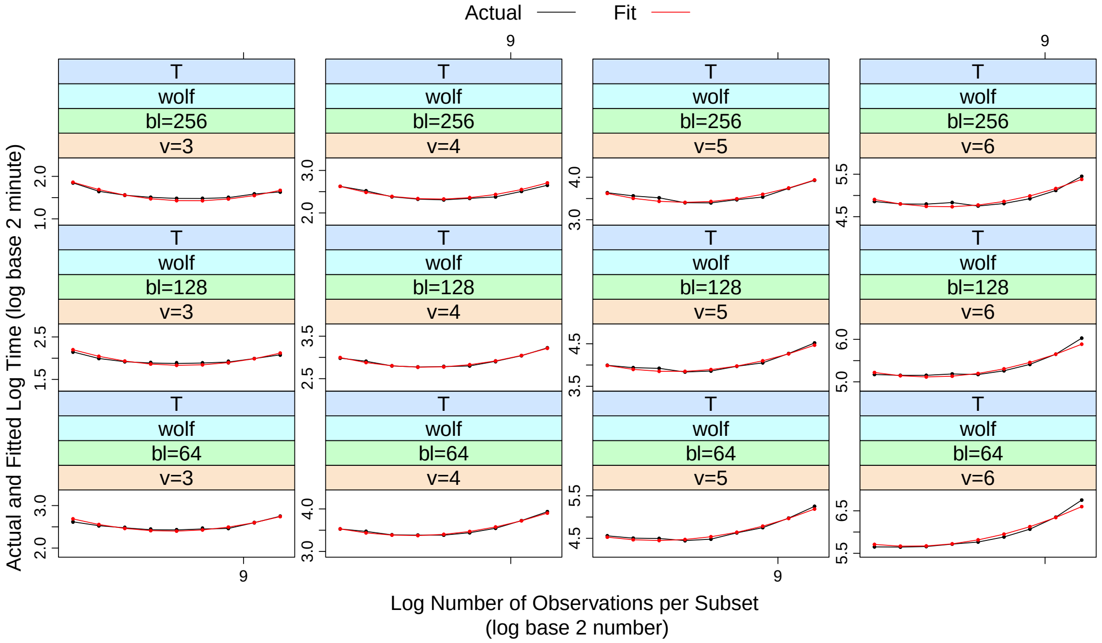

Fig. 1.31.: wolf t fits and actuals vs m

and wsc cluster, which suggests changing B has more impact on computations in the wolf cluster than that in the wsc cluster. This observation holds true for most of the panels except the last panel with compute type T and $v = 6$, where the ratio across three different blocksize almost overlap with each other. This suggests that when the subset size keeps increasing, the benefit of changing B on wolf cluster is smaller.

Cluster Computation Performance

The wsc cluster and wolf cluster are two very different clusters. As we can see from Table 1.5, there are at most 199 containers running in parallel in wsc cluster vs 23 containers in wolf cluster, this means the parallel number of containers in wsc is more than 8 times of that in wolf. The total disk storage of wsc is 40 times of that of the wolf cluster. The RAM of wsc is more than 6 times of wolf. Obviously, wsc cluster

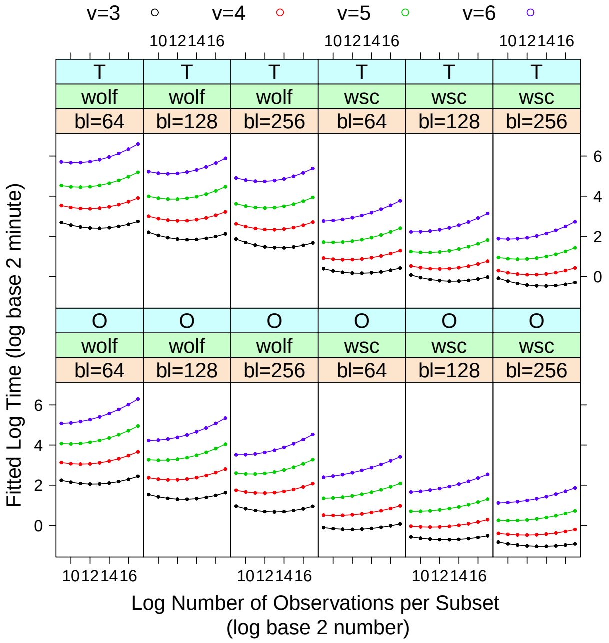

Fig. 1.32.: Polynomial model: o,t fits vs m

has considerably more compute power than wolf. It is also much more expensive to set up and maintain a 10 node cluster than a one node sudo-cluster. The cluster analysis in section 1.6.2 quantifies the cluster difference. As we can conclude, the wsc cluster is much faster than the wolf cluster as expected. However, the difference is not very large: the elapsed time T in wolf cluster is 4 times $\sim$ 8 times the speed of that in wsc cluster.

Cluster Computation Capacity

One another topic of high interest is how large data set wolf and wsc cluster can deal with. In our experiment, the largest data input size is 0.5TB. The total drive disk storage in wsc cluster is 160TB and 4TB in wolf cluster. The total elapsed time in wsc cluster can be as small as 3.7 minutes and 28.8 minutes in wolf cluster. This is impressive how efficient the cluster with Rhipe is on big data analysis. We also

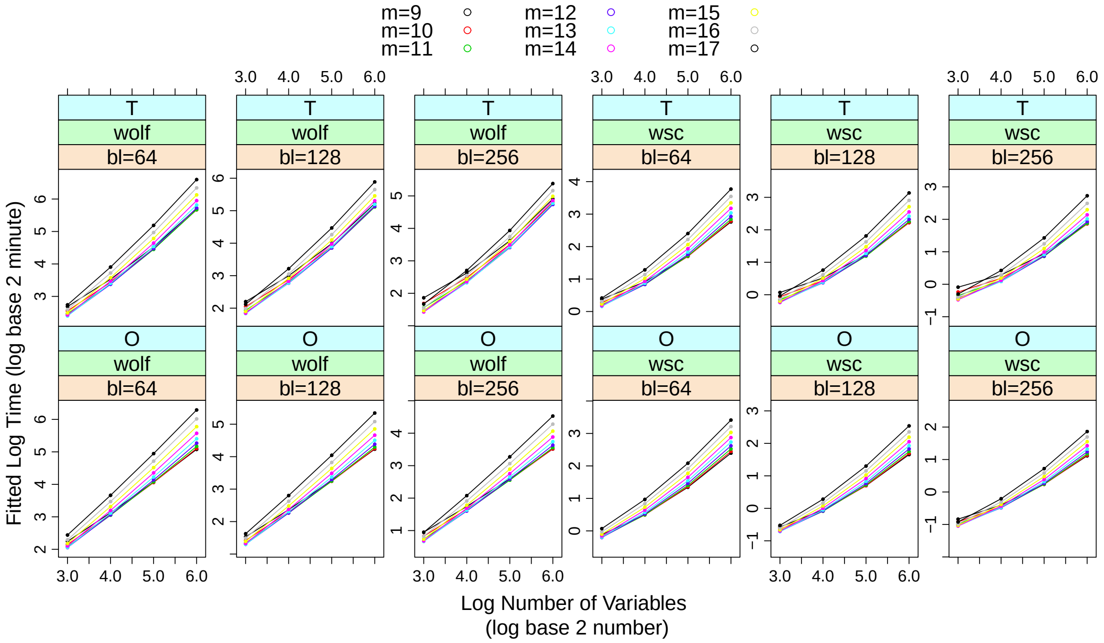

Fig. 1.33.: Polynomial model: o,t fits vs v

explore to expand the input data set size to 1TB and even 2TB on both clusters. Both clusters finish the logistic regression successfully. It is not surprising for wsc cluster on analyzing a 2TB data but the wolf cluster shows the surprising computation capacity to deal with very large data set. We can continue to increase the input data set size and the wolf cluster can finish the analysis with longer elapsed time.

1.7 Experiment Conclusion

With the fast development of new technologies, hardware updating and social network, the amount of data produced every day is growing very fast. This requires high storage capacity and computation efficiency. There are researchers and companies developing big data distributed platforms to improve the computation performance. The common practice is benchmarks to measure the CPM&A of big data distributed system, which almost always do not take account of the factors, nor do they measure

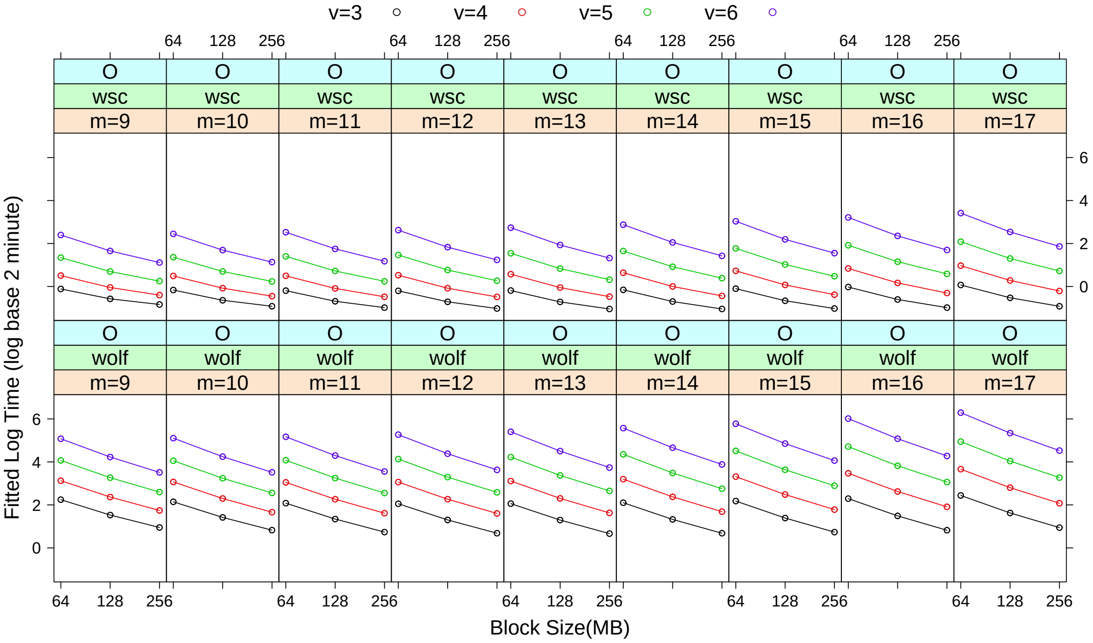

Fig. 1.34.: Polynomial model: o fits vs B

run time of analytic methods. For example, CPM&A is often done using benchmarks such as sort. For the data analyst, what matters is the elapsed times of analytic methods of big data. D&R with DeltaRho computation environment provides a feasible way to analyze big data. We propose and design a multi-factor experiments for CPM&A on analytic methods of big data illustrating using D&R with DeltaRho. The multi-factor experiment gives us insight of how different factors are interacting with each other and how to tune some parameters to improve the computational performance. Here are the conclusions we can draw from this experiment:

- O increases as v increases. The relationship between v and o is closed to linear with a slightly convex upward.
- O is a big portion of T, could be larger than 80%.
- O increases very slowly as m increases and when m is small. The slop of the increment gets larger and larger as m value gets larger.

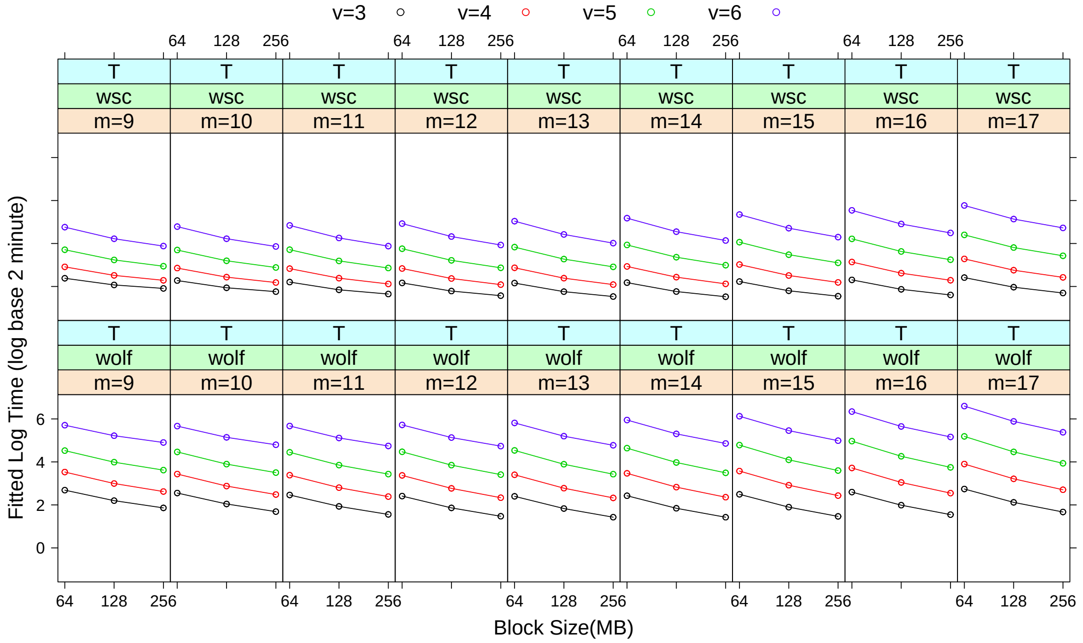

Fig. 1.35.: Polynomial model: t fits vs B

- O is a big portion of T, could be larger than 80%.

- Block size mainly impact O time, not L. This might because the analytic method in the experiment is logistic regression and the reducer is averaging coefficients, which are not heavily CPU-intensive computations. Large Block size is recommended for large data computation.

- Parameter tuning could reduce O by more than 200%, T by more than 100%.

- Hadoop system is very robust to large data computation and it can handle large data set beyond the RAM.

- This experiment gives good insights that one node cluster can takes care of very large data. The wsc cluster is $3 \sim 7$ times faster than wolf. But it is much more expensive to set up and maintain a 10 nodes cluster than a single cluster.

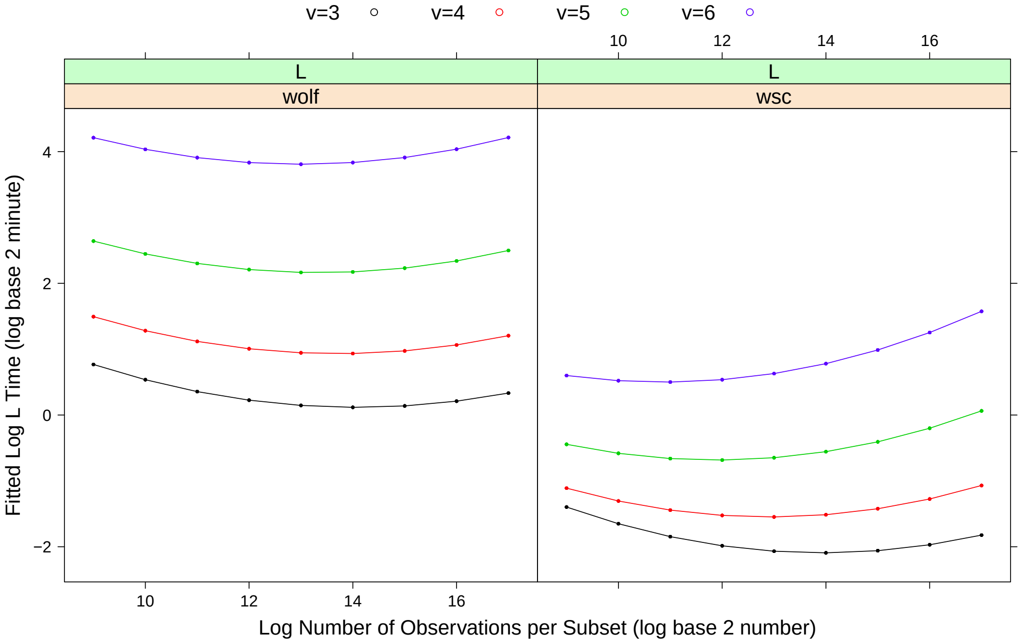

Fig. 1.36.: l fits vs m

There are other important tuning variables and other big data platform, such as Spark that are included in the experiment. And this project provides a statistically design multi-factor experiment framework for the CPM&A that can be easily generalized to other platforms.

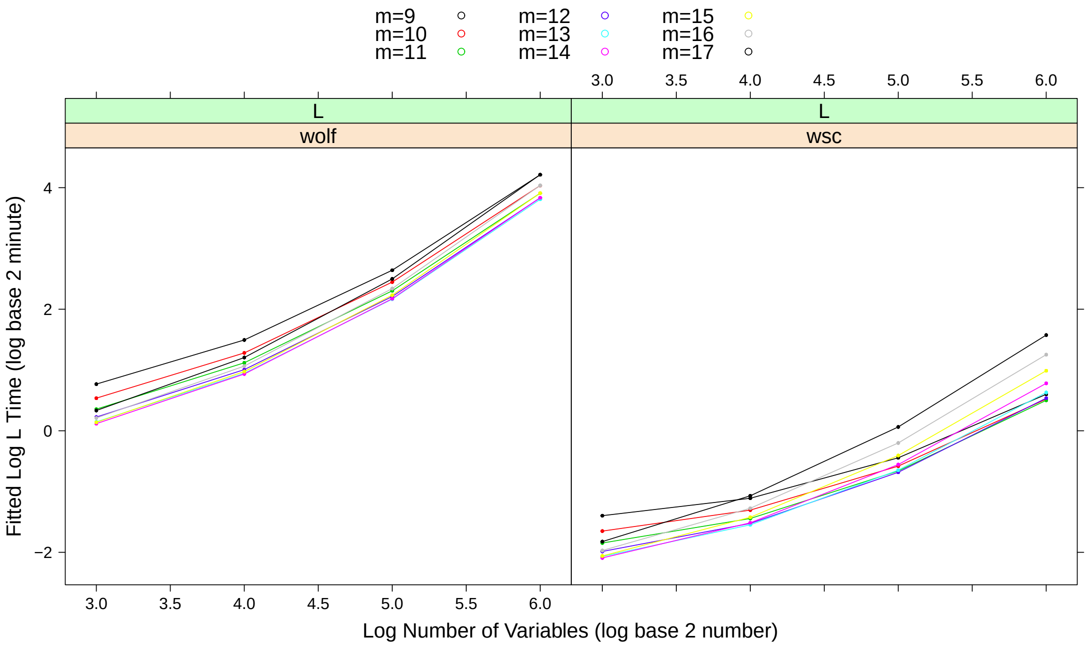

Fig. 1.37.: l fits vs v

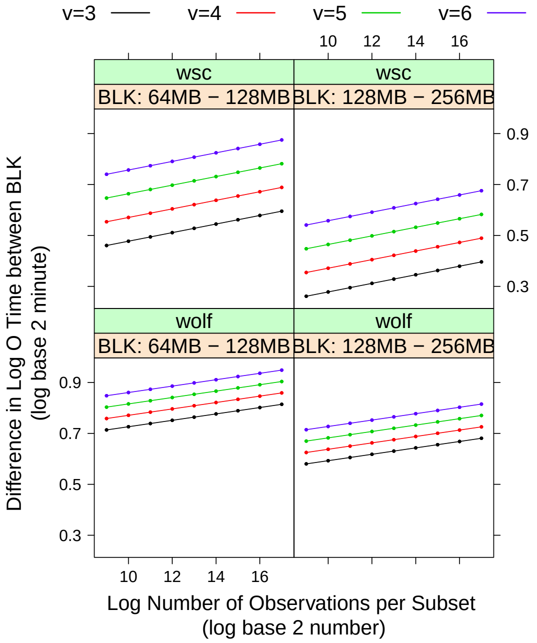

Fig. 1.38.: Effects of B on o conditional on *cluster* and B

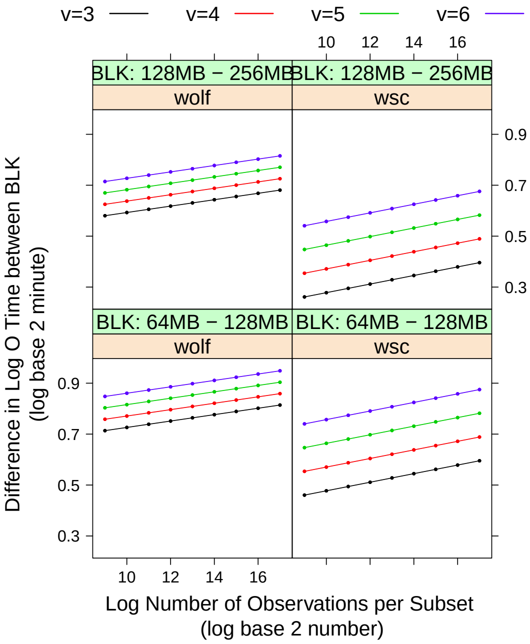

Fig. 1.39.: Effects of B on o conditional on B and *cluster*

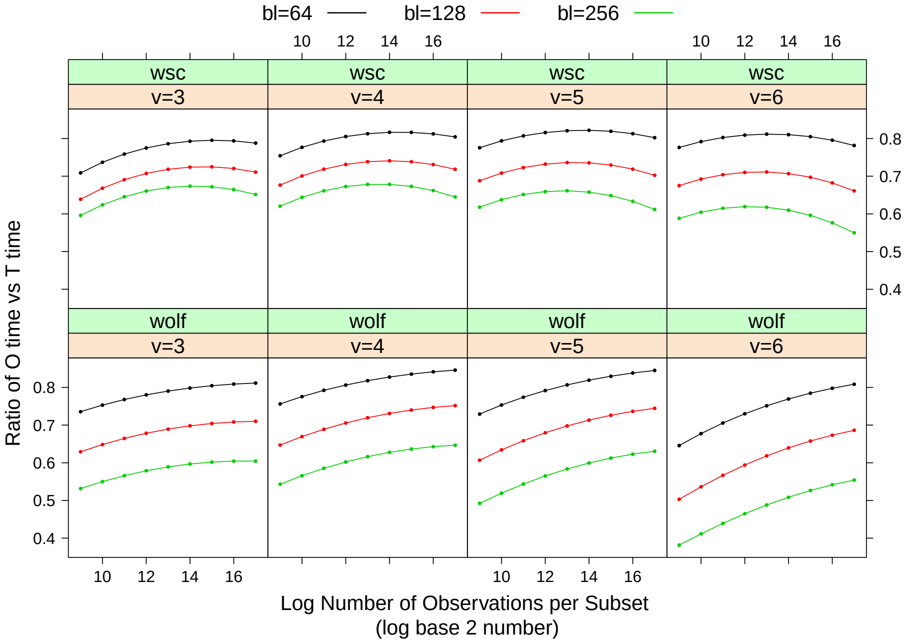

Fig. 1.40.: Effects of B on the ratio of O and T

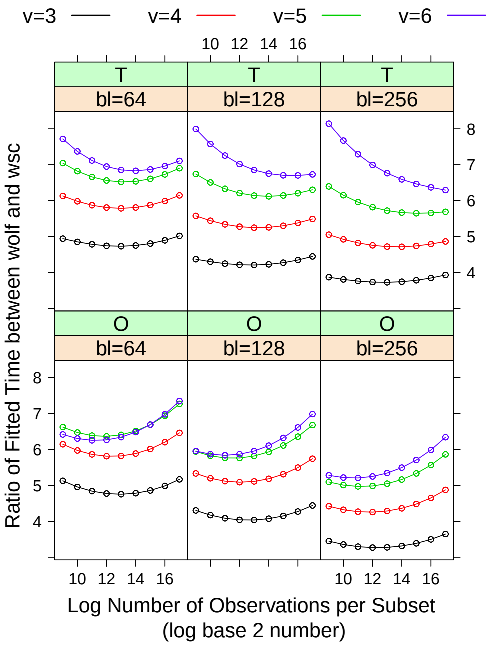

Fig. 1.41.: Ratio of fitted time between wolf and wsc cluster superpose on v

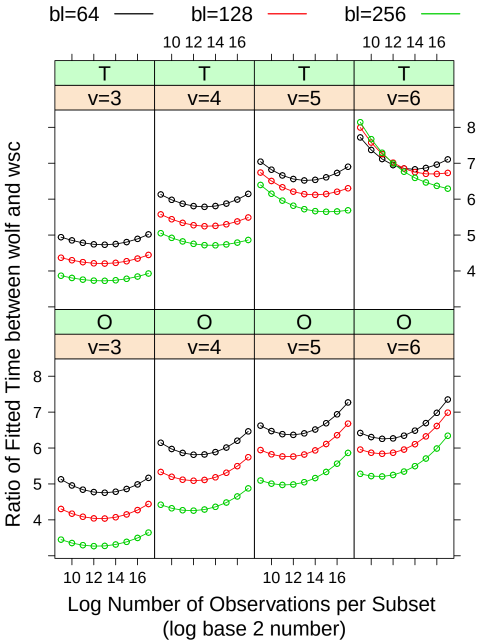

Fig. 1.42.: Ratio of fitted time between wolf and wsc cluster superpose on B

2. ANALYSIS OF TEN BILLION SPAMHAUS INTERNET BLACKLISTING QUERIES

2.1 Spamhaus Data

2.1.1 Spamhaus Service

An IP address referring to the internet protocol address, is a unique numerical label assigned to every device connected to a computer network that uses the Internet Protocol for communications [31]. It identifies the computer device and contains information of its location and access. Host computers on the Internet, e.g., laptops, web servers, mail servers, and routers, have IP addresses. These addresses have four sets of numbers and each number ranges from 0 to 255, separated by dots. For example, 10.1.60.9 is an IP address.

The Spamhaus project [32] is a global non-profit organization that provides service of classifying IP addresses as blacklisted or not based on many sources of information and many factors such as creating spam, being a major conduit for spam, or violating internet policies. It is queried in real time by internet mail servers and it is one of the largest database to provide blacklisted or non-blacklisted information for mail service and has been protecting more than 3 BN user mailboxes [33] till November 2020. Figure 2.1 describes how the mail service communicates with Spamhaus service: when a mail server (Receiver) receives an email from the Sender across the Internet, it sends a query to Spamhaus to determine if an IP address or domain name is blacklisted or not; The Spamhaus will response Yes or No depending on whether the queried IP is on the blacklists or not. If the sender's IP address meets the Spamhaus policy and is not blacklisted, the Receiver will deliver and process the mail normally. On the other side, if the sender's IP turns to be blacklisted, the receiver will makes its own policy decisions based on the its settings. The common practice is to reject the message and notifies the sender. The Spamhaus service is only responsible to check the IP's status and receivers will take different actions based on the system settings.

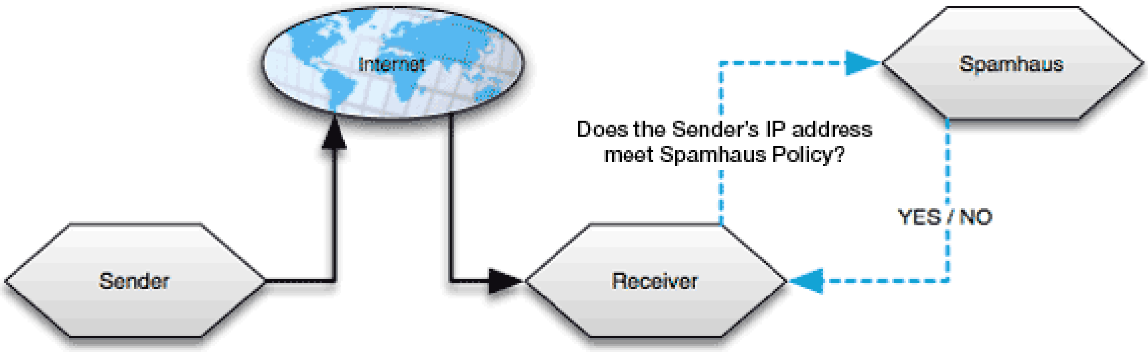

Fig. 2.1.: Basic DNSBL flow

DNSBL stands for *Domain Name System Block List* [34] and it provides a list of malicious hosts information. The Spamhaus provides several DNSBL zones for IP check, such as XBL zone, SBL zone, CSS zone and PBL zone. Different zones are various database of block lists with different focused study of policy violations. All the four zones are live database of IP addresses. XBL zone [35] is the Spamhaus Exloits Block List. It lists all the spam IPs compromised to third party devices exploits, such as worm viruses in spam engines and open proxies. SBL zone [36] provides the Spamhaus Block List with spam emails that are recommended to be rejected by Spamhaus service. It lists all the IPs with unsolicited bulk email (spam) and other security threats, such as Snowshoe spam, Spam hosting, Phish sites and Hacking attempts. CSS zone [37] is a component of SBL zone with focus on IPs sending out low reputation emails. PBL zone [38] provides a Policy Block List of IPs that sending unauthenticated SMTP emails to mail servers. SBL and XBL zones cover the majority of spam types of blacklisted IPs. The Spamhaus project provides the those block lists for mails servers to check. There are more than 80 public Spamhaus mirror servers distributed worldwide for the purpose of high redundancy and efficiency.

Our goal is to study the properties of the blacklisting process for blacklisted IPs. Spamhaus has a process by which a blacklisted IP can be unlisted from the blacklisted lists, but the time frame is on the order of hours and even days. One might expect then to see blacklisting persist for some time. A major task is to study for an IP address with blacklistings, the change through time back and forth of blacklisting and non-blacklisting. A pilot study [39] showed there was a faster back and forth blacklisting and non-blacklisting than expected. Our major goal of the project is to find the cause of this quick back and forth of blacklisting and non-blacklisting.

2.1.2 Spamhaus Data Collection

Receivers send queries and get responses from Spamhaus service all the time. Queries and their responses are collected by John Gerth in the Computer Graphics Laboratory from the Stanford mirror of the Spamhaus site in 2014. The data was collected at a raw data rate of 100 GB per week, over a period of eight calendar months. The Spamhaus Data packets were collected using *TCPDump* and were saved in *pcap* [40] format for files. Ashrith Barthur [39] from the Department of Statistics of Purdue University first transferred and processed the Spamhaus DNS packets using python [41] by which the raw data was converted from the binary format to text readable format. Approximately 1.0TB raw data was saved into 13,204 text files written to HDFS of wsc cluster. The wsc cluster is used in the analysis of Spamhaus data. It has 10 nodes, 200 cores, 1.25 TB RAM and 160 TB disk storage (Table 1.4). It is worthy noticing that the physical memory size of wsc cluster is only 25% greater than the data size. The large size of the raw data input is one of the largest challenges in the data analysis. D&R was used to analyze the data. Computations were carried out by utilizing R/RHIPE/Hadoop in wsc cluster.

2.1.3 Spamhaus Data Variables

Each query, along with its response is our data raw input. Each row in the raw text input is a query and response from Spamhaus Service. Each row has 13 variables to study the properties of blacklisting:

- **time**: numeric in second, query response collection time.
- **source**: character, the source IP address or querying IP address, such as 192.221.138.161.
- **qtype**: character, specifying the queried address type: IP address or Domain name or query error.
- **qname**: character, sender or queried address, such as 10.1.60.10.
- **rcode**: character, the response code identifying correct and incorrect query formatting.
- **state**: character, state of query and response: noinfo, blacklisted or query error.
- **ab.isp**: IP address ISP policy violation, 1 or 0.
- **ab.s**: IP address Spam, 1 or 0.
- **ab.sc**: IP address CSS spam, 1 or 0.
- **ab.e**: IP address Spamhaus exploit, 1 or 0.
- **ab.p**: IP address Spamhaus policy violation, 1 or 0.
- **db.s**: Domain address Spamhaus spam, 1 or 0.
- **db.sr**: Domain address Spamhaus spam redirection, 1 or 0.

The **time** variable is the timestamp when the response from Spamhaus were collected by the TCPDump package. This variable provides very important information of blacklisted properties of IPs.

The two variables **source** and **qname** are querying IPs and queried addresses respectively where **qname** can be either a IP address or Domain name. The variable **qtype** specifies whether the queried address is a IP address or Domain name. IP address is different from a Domain name. IP address is a unique numerical label assigned to every device. It is very useful and machine friendly for communications between computers but not human friendly to understand. Domain name is more user friendly to locate and understand, for example,"www.google.com" is a domain name for Google. Users use domain name to visit a website while computer will use the IP address to access information. Domain name and IP address are connected by the DNS system (Domain Name System). One domain name can correspond to multiple IPs while one IP can uniquely identify the device. As a result, our Spamhaus data analysis will focused on blacklisted IPs not on listed domain names. There are two blacklisted types for a blacklisted domain name: **db.s** and **db.sr** determined by DBL zone [42]. So the variables related to Domain name properties will be removed from the further data cleaning steps.

The variable **rcode** is the response code from DNS packet to identify whether the query output is valid or not and whether there is query error or format error [39]. And **rcode** variable partially defines **state** variable. The **state** variable is a derived variable that identify the status of the queried IP: blacklisted, noinfo or query error. When **rcode** returns code of no information, the value of **state** is set to noinfo as well, which suggests that there is no listing for the domain requested in DNS. When **rcode** returns query error, **state** is set to queryerror. If the **rcode** returns noerror, **state** value is determined by the five blacklisted response variables: **ab.isp**, **ab.s**, **ab.sc**, **ab.e**, **ab.p**. If there are blacklisted violations, the value of **state** is set as blacklisted.

There are five blacklisted types related to a blacklisted IP address: **ab.isp**, **ab.s**, **ab.sc**, **ab.e**, **ab.p**. Each is a binary variable with value 1 or 0 where value of 1 means the queried IP is on the blacklisted list with the violation of corresponding listed behaviours. One queried IP can have zero and up to five violations. Table 2.1 displays

Table 2.1.: Spamhaus Block Lists and Return Codes

Block List	**Return Code**	**Variable**	**Description**
SBL	127.0.0.2	as.s	Spamhaus Violation
	127.0.0.3	as.sc	CSS Spam
XBL	127.0.0.4	ab.e	Exploit
PBL	127.0.0.10	as.isp	ISP Violation
	127.0.0.11	as.p	Spamhaus Policy Violation

the relationship between Spamhaus block lists and the five blacklisted variables for IPs where CSS is included in SBL. There are common return codes specified by Spamhaus service connecting to different block lists and different malicious activities.

The spamhaus data analysis will focus on IP addresses and their blacklisted behaviors so that not all the 13 variables above will be analyzed in the data analysis. The domain name related variables will be dropped from the analysis later on.

2.1.4 Spamhaus Data Summary Statistics

The size of the raw input data is 1.0TB and there are 13,204 text files written to HDFS. The processed data consist of values of 13 variables for each of 10,621,808,809 queries during more than 8 months. The following pieces of information summarize the features of our input data.

- Duration of collection in months: 8+.
- Total number of queries: 10,621,808,809.
- Total number of querying IP addresses: 982,293.
- Total number of queried IP addresses: 206,952,971.
- Total number of queried IP addresses with at least one blacklist result: 59,432,518.

We are interested in the behaviour of Spamhaus blacklisted responses. And the raw data input is a very large dataset and we will apply D&R of big data analysis to the Spamhaus data in the following sections.

2.2 Spamhaus Data Divisions

The size of the raw input data is 1.0TB with 13,204 text files written to HDFS. Hadoop has a relatively poor performance to deal with large number of small key-value pairs [10], especially for text file input types. When the input data is text file, each query or line number is a key and the corresponding line is a value. This means there are 10,621,808,809 key-value pairs. It is always time consuming for Hadoop to access text inputs and the common practice is to divide the text inputs and save it to sequence files on HDFS. There are different ways to divide data and one of the most common method is by subject-matter divisions, that is, chunking data into subsets by conditioning on meaningful variables important to the analysis, such as time or location variables. All divisions in the Spamhaus analysis here are by subject-matter divisions. The raw input data will be broken up into numbers of key-value pairs and saved into sequence files in HDFS. The value is a R data frame for each key.

2.2.1 First Division by Variable Time

There are computation challenges to read and divide the Spamhaus data in practice due to its large input size and complex relations among variables. As we discussed, Hadoop is not suitable to deal with large number of small size key-value pairs but there is also performance problems when the key-value pair size is too large. There will be job failures if the key-value pair size is too large to fit into container memory. For this reason, we first divide the input to data frame with fix number of k rows by variable **time**. The value choice of k is set to be 6000 for a better computation performance [39]. That means the raw input data is chunked to multiple number of key-value pairs:

- Key: time stamp.
- Value: data frame with 6,000 rows and 13 columns, ordered by time stamp.

Here is one key-value pair example for the first division by **time**:

```
> head(key)
[1] 1390972922
> head(value,2)
        time         source  rcode   qtype         qname
1 1390972922 166.104.239.11 noinfo address 125.246.106.87.sbl.spamhaus.org
2 1390972922     8.0.35.239 noinfo address  147.66.112.74.zen.spamhaus.org
   state ab.isp ab.s ab.sc ab.e ab.p db.s db.sr
1 noinfo      0    0     0    0    0    0     0
2 noinfo      0    0     0    0    0    0     0
> dim(value)
[1] 6000    13
```

The first division is saved to sequence file on HDFS and becomes our new input for further analysis. It contains the same information as the raw text files but more efficient in computation performance than text inputs.

2.2.2 Second Division by Queried IPs

The objective of the Spamhaus analysis is to understand the properties and behaviours of the blacklisted IPs. The blacklisted IP is defined as IP that has at least one blacklisted queries across time. As a result, we need to collect all the queries over time related to the same queried IP and determine whether this IP is blacklisted or not. That suggests we need to form divisions by IP. Hence our second division is by queried IPs. The following steps describe how we form the second division.

- Firstly, form by-IP-by-week division: the key is (IP, week) and the value is a data frame including all the queries for the queried IP within a week.

- Secondly, form by-IP-by-month division: the key is (IP, month) and the value is a data frame including all the queries for the queried IP within a month.

- Lastly, form by-IP division: the key is IP and the value is a data frame including all the queries for the queried IP across 8+ months.

We are not directly forming the by-IP division due to the computation complexity. Instead, we first form the by by-IP-by-week division, then use that as input to form by-IP-by-month division and then form the final by-IP division. We take several interim steps before we got the by-IP division. The reason is we are dealing with big data analysis and some queried IPs might have very large number of queries and there will be memory problem and computational failures to include all the queries in one subset. In fact, there are Hadoop job failures when combining all months queries together for the same IP. It will simplify our analysis if we can include all the queries of queried IP in a subset so we need to optimize the data frame structure to reduce the subset size. There are two tricks to optimize the data storage. Firstly, we remove the variables that are not related to IP analysis, such as domain name related variables. Secondly, we reformat the variables types and summarize information by:

- converting the character variable **source** IP to numeric numbers. Character types takes more space to store than Numeric types.

- reducing five blacklisted variables **ab.isp**, **ab.s**, **ab.sc**, **ab.e**, **ab.p** into one variable **ab.sum**. As we discussed previously, each of the five variables is a binary variable with value 1 or 0 where value of 1 means the queried IP is on the blacklisted list with the violation of corresponding listed behaviours and one queried IP can have zero and up to five violations. So we can carry out the binary-to-decimal calculation to form one variable **ab.sum**. For example, assuming $(\mathbf{ab.isp}, \mathbf{ab.s}, \mathbf{ab.sc}, \mathbf{ab.e}, \mathbf{ab.p}) = (1, 0, 0, 0, 1)$, the corresponding decimal value of **ab.sum** for binary number 10001 is 17: $17 = (1 \cdot 2^4) + (0 \cdot 2^3) + (0 \cdot 2^2) + (0 \cdot 2^1) + (1 \cdot 2^0)$. The binary to decimal conversion are one to one mapping, that is, the five variables can be well represented by **ab.sum** and **ab.sum** can be easily transformed back to five binary variables.

The numeric transformation of **source** IP and binary calculation of **ab.sum** reduce one subset size by 33% to 60%. And we keep all the information we need and successfully reduce the subset size. Here is the key-value pair information for the second by-IP division:

- Key: Queried IP.

- Value: data frame with 3 columns.

And the one sample key-value pair is:

```
> head(key)
[1] "0.1.48.8"
> head(value,2)
        time     source ab.sum
1 1404479983 4446037760      0
2 1406261684   27787453      0
```

There are 206,952,971 queried IPs in total which means there are 206,952,971 subsets or key-value pairs. Each subset is a data frame including all the queries for the queried IP across 8+ months. Some IPs are very active to send emails and some are not. The number of queries among IPs varies from 1 to 16,777,216. About 56% queried IPs have only one query. And there is only 10% queried IPs with equal to or over 16 queries. Most queried IPs are not informative in statistical modelling.

2.2.3 Third Divison by Blacklisted Queried IPs

The focus subject of interest are blacklisted queried IPs (*BBQ-IP*) which are with at least one query that has a blacklist result. It is very straightforward to build a third division by *BBQ-IP* based on the input of the second by-IP division. We will only keep the IP and its corresponding queries if the IP is blacklisted. Here is the key-value pair for the third division:

- Key: *BBQ-IP*

- Value: data frame with 3 columns

And one key-value pair example is:

```
> head(key)
[1] "0.0.0.0"
> head(value,2)
        time     source ab.sum
1 1390953601 3213870602      0
2 1390953608  908939772      1
```

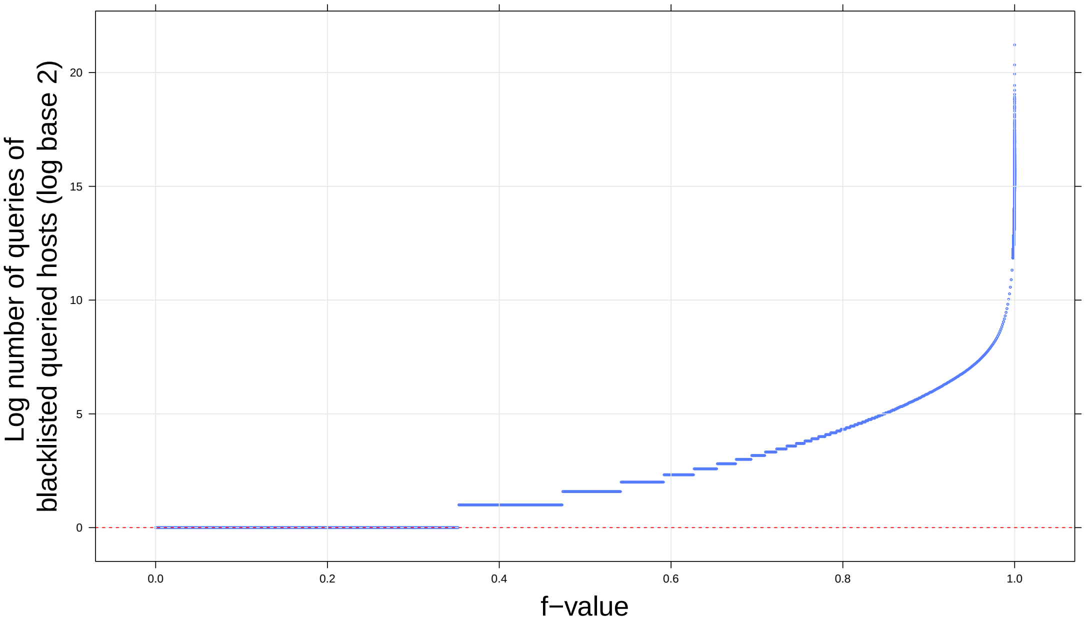

Fig. 2.2.: Distribution of log2 number of queries per IP

There are 59,432,518 number of subsets in the third division which means there are 59,432,518 *BBQ-IP*s in total and 28.7% of IPs are blacklisted. We calculate the global summary statistics across all the 59,432,518 *BBQ-IP*s to understand the data properties. Figure 2.2 displays the distribution of total queries number among *BBQ-IP*s in log2 base where the range of query numbers per IP varies from 1 to 2,439,536. There are 13,569,257 IPs, 22.8% of the total *BBQ-IP*s, with query counts more than or equal to 16. And 9,114,378 (15.3%) IPs have queries more than or equal to 32. This means, there are only 22.8% of

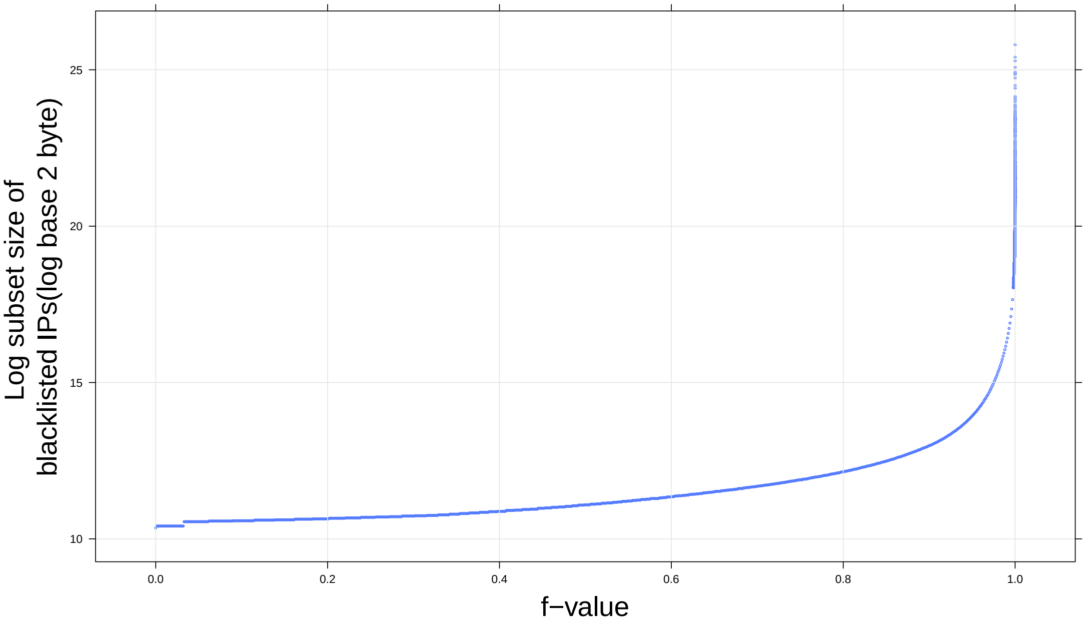

Fig. 2.3.: Distribution of subset size

*BBQ-IP*s queried more than or equal to 16 times across the 8+ months period while the majority of the *BBQ-IP*s stay in an inactive status. It is not statistically informative to study the blacklisted properties of inactive *BBQ-IP*s so we will firstly focus on the 22.8% of *BBQ-IP*s that have query numbers ≥ 16. Figure 2.3 shows the distribution of the subset size of the 13,569,257 IPs. The subset size change from 2^{10} Byte to more than 55 MB which further shows that there is a vast difference among IPs' activities. Figure 2.4 shows the distribution of log2 number of blacklisted queries per IP among the 22.8% of selected *BBQ-IP*s where 50% of IPs have less than 19 number of blacklisted queries across the 8+ months period. The range of the number of blacklisted queries varies from one to 1,997,428. Figure 2.5 shows the distribution of blacklisted frictions per IP. The blacklisted friction per IP is calculated by comparing the total number of blacklisted queries versus the total number of queries. The smallest blacklisted frictions is 0.0004% which happens for one specific IP with only one blacklisted IP and 232,559 total queries. There are 38,443 blacklisted IPs with blacklisted fraction value equal to 100%, which means all the queries for the IP are blacklisted. The total query numbers vary from 16 to 8033. And 50% (medium) of the blacklisted frictions are larger than 40.6%. We will carry out the exploratory analysis for the Spamhaus data in the following sections.

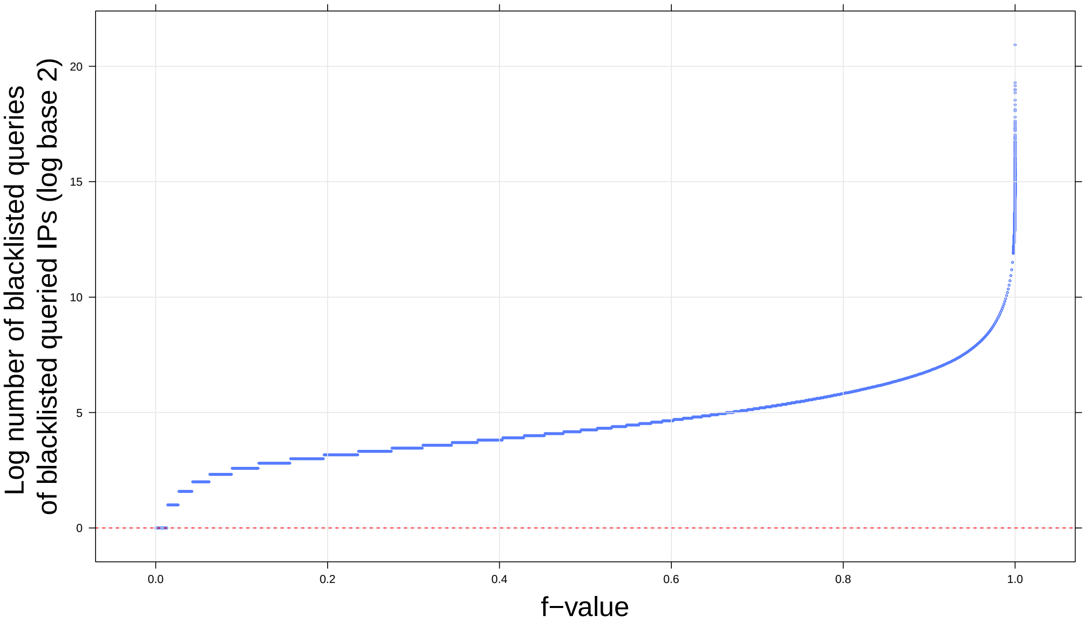

Fig. 2.4.: Distribution of log2 number of blacklisted queries per IP

2.3 First Sample Analysis of the Most Queried IPs

We selected 13,569,257 *BBQ-IP*s that have query counts more than or equal to 16 from the last section. This is still a large number of objects. Furthermore, Figure 2.3, 2.4 and 2.5 suggest there might be different blacklisted properties among different IPs, which might bring more challenges into data analysis. We are interested in the blacklisted properties and need to develop informative metrics or statistics to explore and understand it. For this purpose, we carry out the first sample analysis by selecting the top 100 blacklisted IPs with the most number of queries. The selected 100 *BBQ-IP*s are the most active ones which might convey more information of blacklisted property. Figure 2.6 describes the distribution of the queries numbers for the 100 selected IPs where the total query numbers per IP vary from 35,600 to 2,440,000. Figure 2.7 shows the distribution of IP's active time duration

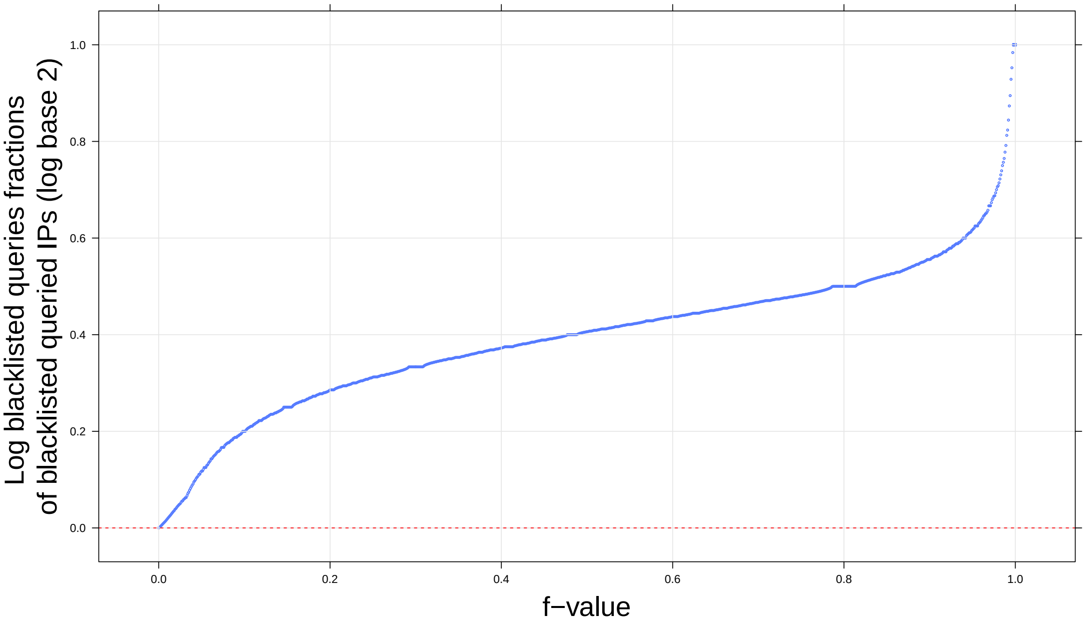

Fig. 2.5.: Distribution of blacklisted queries frictions

which is defined as the time difference between IP's first query to the last query. As we can see from Figure Figure 2.7, the duration lasts from 88 days to 215 days (8 months). Figure 2.8 gives the distribution of the blacklisted frictions among the 100 selected IPs where we can see more than 75% of the selected IPs with blacklisted friction less than 1%. This suggests that our selected most active IPs are lack of necessary blacklisted information. We need to have enough blacklisted data to study the blacklisted properties so we need to reform the criteria of sample selection.

2.4 Second Sample Analysis with Friction and Count Constrains

Now we have all the blacklisted IPs and we need to study those who have the most statistical information about the blacklisting properties. The first sample of most active IPs did not provide good representatives of blacklisted information. So we take the second sample of *BBQ-IP*s by the following two criteria:

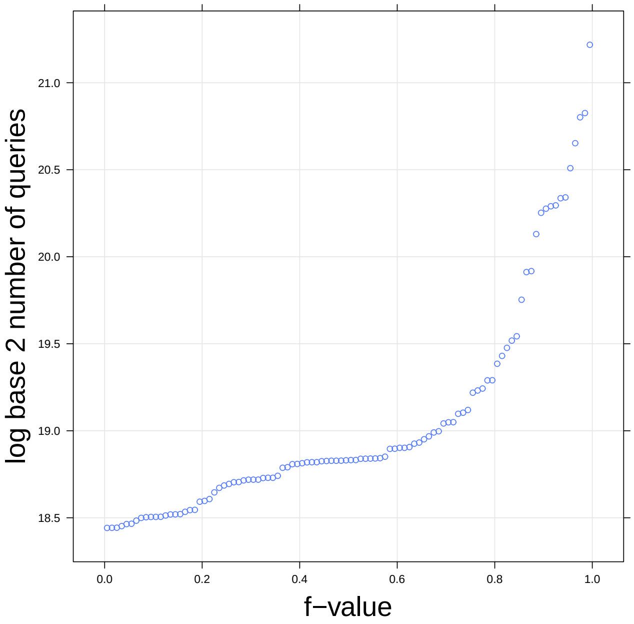

Fig. 2.6.: Distribution of queries numbers for 100 selected IPs

- The number of queries should be larger or equal to 2^{13}. We need to have active IPs to analyze the quick back-and-force blacklisting and non-blacklisting behaviours.

- The blacklisted fraction should be larger than or equal to 25%. We need to have enough blacklistings to understand the blacklilsting properties.

The two constrains will ensure that we have active IPs as well as good representatives of blacklisted activities. As a result, 30,088 *BBQ-IP*s are selected out of the 13,569,257 *BBQ-IP*s.

The selected 30,088 *BBQ-IP*s are our focus of study for the following sections. As we mentioned in the Section 2.1.1: our goal is to study the properties of the blacklisting process for blacklisted IPs. Spamhaus has a process by which a blacklisted IP can be unlisted from blacklisted lists, but the time frame is on the order of hours and even days. One might expect then to see blacklisting persist for some time. A major task is to study for an IP address with blacklistings, the change through time back and forth of blacklisting and non-

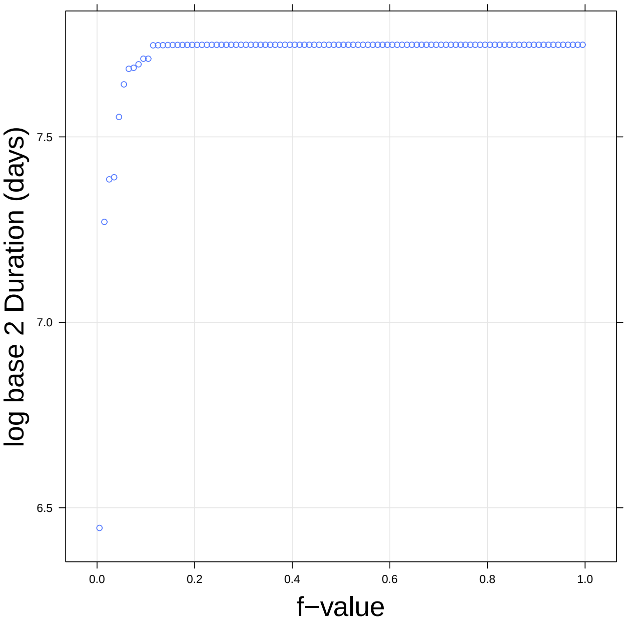

Fig. 2.7.: Distribution of IP active duration for 100 selected IPs

blacklisting. Our major goal of the project is to find the cause of this quick back and forth of blacklisting and non-blacklisting.

Figure 2.9 summarizes the distribution of the number of queries per IP which vary from 8,192 to 1,286,000. The total number of queries across all the 30,088 IPs are 526,732,379. Figure 2.10 shows the distribution of blacklisted fraction: the range of fractions are from 25.0% to 99.7%. 50.0% of *BBQ-IP*s has blacklisted frictions larger than 44.0%. The distribution is right skewed with more than 99.7% of fractions smaller than 70.0% and only less than 0.3% are larger than 70.0%.

We need to develop relevant metrics to understand the blacklisted activities. The selected 30,088 *BBQ-IP*s are still a large number of object so we select 100 IPs from the 30,088 ones to better understand the blacklisted pattern and to form meaningful statistics. 100 IPs are selected from the 30,088 IPs with the blacklisted frictions equally spaced between 0.25 and 1.0 so that we have a good representative of IPs across different blacklisted frictions. However, the fraction distributions are not uniformly distributed between 0.25 and 1.0 due to the right skewed distribution of fractions (Figure 2.10) and the right tail does not have

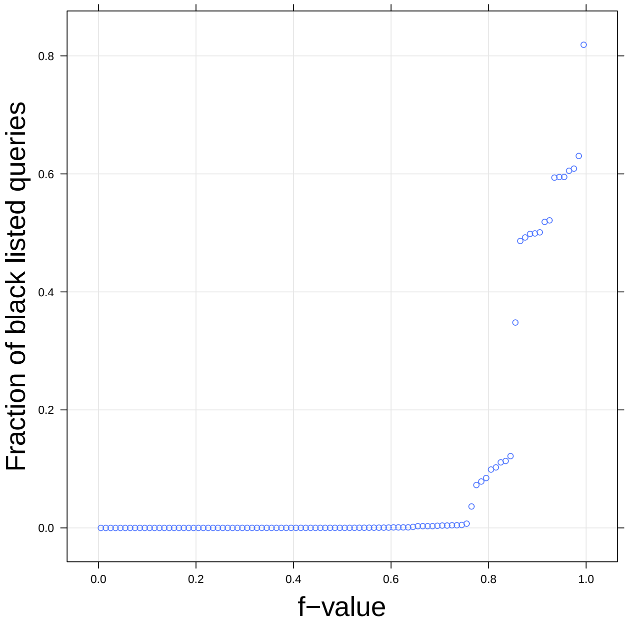

Fig. 2.8.: Distribution of blacklisted frictions for 100 selected IPs

enough data to sample from. There are only 21 IPs with fractions larger than 85.5% so we keep all the 21 IPs in the 100 samples. Figure 2.11 displays the index plot of the blacklisted fraction of the 100 IPs where we can see the 100 selected IPs's fractions linearly increase from 25.0% to 99.7% and there are some curvature off linear pattern in the tail.

We will first analyze the blacklisted properties on the 100 selected IPs and then generalize to the 30,088 *BBQ-IP*s.

2.4.1 Sample Analysis – Blacklisted Rate

We first study the blacklisted properties by analyzing the time series of blacklisted rate. For each of the 100 selected *BBQ-IP*s, their queries are sorted by time chronologically. We break down the queries per IP into groups of 64 consecutive queries. Let t_i be the time duration of the i^{th} group of a IP where t_i is the time difference between the first query and 64^{th} query in i^{th} group. Now we define the blacklisted query rate, non-blacklisted query rate and query rate as in Equation 2.1, Equation 2.2 and Equation 2.3 respectively. There are

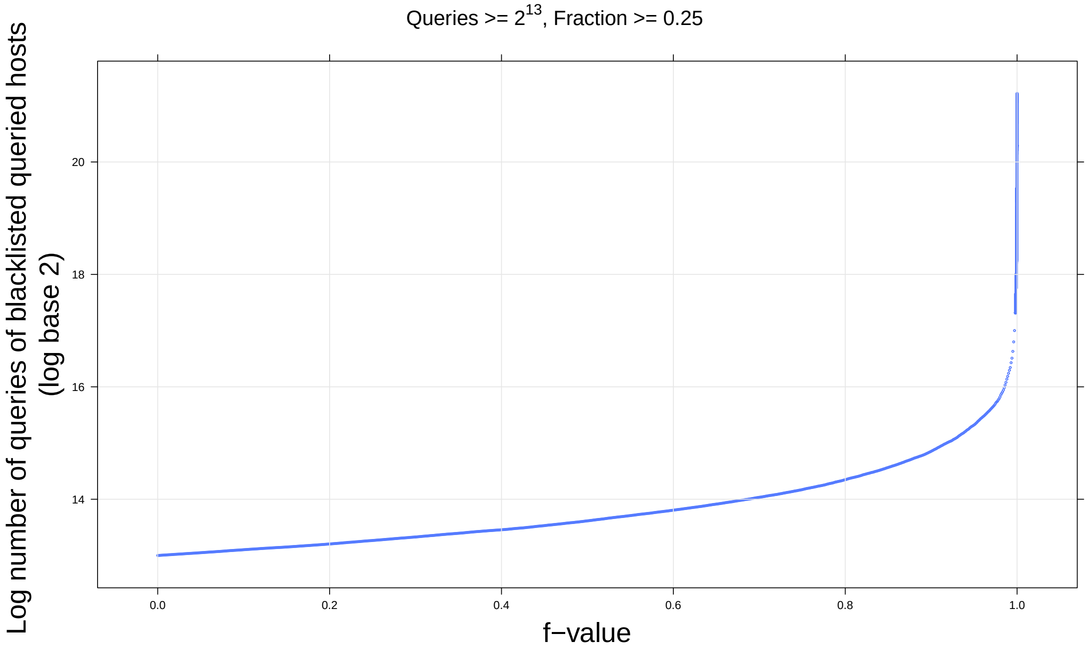

Fig. 2.9.: Distribution of queries count of 30,088 IPs

two different types of listing rate: the *blacklisted query rate* and *non-blacklisted query rate* where *blacklisted query rate* measures how fast the blacklisted is while the *non-blacklisted query rate* measures how fast the non-blacklisted is. Equation 2.4 and Equation 2.5 describe the mathematical relationship among the two listing rates and the query rates: we keep the total queries per group as a constant value 64 and measuring the duration.

$$\text{blacklisted query rate}_i = \frac{\text{number of blacklisted queries}_i}{t_i} = \frac{\text{BL}_i}{t_i} \tag{2.1}$$

$$\text{non-blacklisted query rate}_i = \frac{\text{number of non-blacklisted queries}_i}{t_i} = \frac{\text{NonBL}_i}{t_i} \tag{2.2}$$

$$\text{queried rate}_i = \frac{\text{number of queries}_i}{t_i} \tag{2.3}$$

where:

$$\text{blacklisted queried rate}_i + \text{non-blacklisted queried rate}_i = \text{query rate}_i \tag{2.4}$$

$$\text{BL}_i + \text{NonBL}_i = 64 \tag{2.5}$$

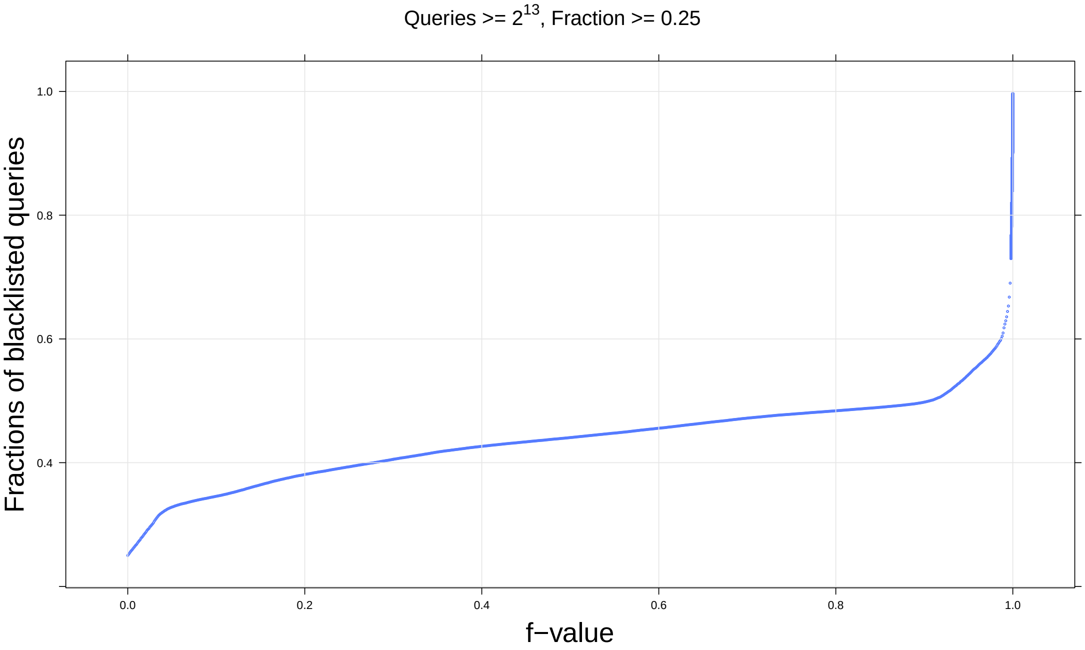

Fig. 2.10.: Distribution of blacklisted frictions of 30,088 IPs

We are interested in the relationship between the different listed rates versus query rates per IP so we fit a robust linear regression [43] of the *blacklisted query rate* versus *queried rate* (Equation 2.6) and *non-blacklisted query rate* versus *queried rate* in log format respectively (Equation 2.7). The robust regression is useful when there are outliers and it can gives better accuracy than ordinary linear regression by weighting the outliers impact. Robust regression is applied to each IP using the *rlm()* function in R package MASS.

$$log_2(\text{blacklisted queried rate}) \sim a_1 + k_1 \cdot log_2(\text{queried rate}) \tag{2.6}$$

$$log_2(\text{non-blacklisted queried rate}) \sim a_2 + k_2 \cdot log_2(\text{queried rate}) \tag{2.7}$$

Equation 2.6 and Equation 2.7 were fitted to each of the 100 IPs independently so there are 200 fittings in total. Trellis display is heavily used in the analysis of listing rates and queried rates. There are 100 pages of the queried rate scatter plot and each page is for a specific IP. The 100 pages plot is not included here for the thesis size purpose but we will illustrates the plot by showing several IPs. Figure 2.12 and Figure 2.13 illustrate the scatter plot between the listing rates and queried rates for two example IPs. Figure 2.12

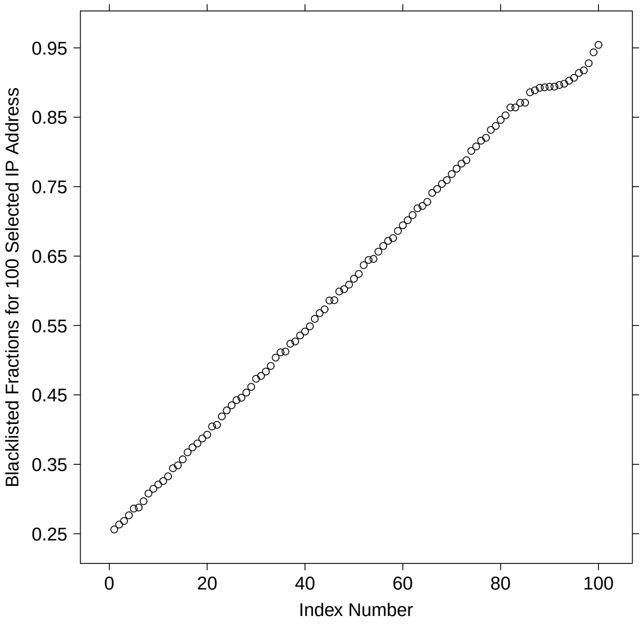

Fig. 2.11.: Distribution of blacklisted frictions of 100 selected IPs

is for IP 209.217.224.13 with overall blacklisted friction 52.4%. It has two panels: the left panel is log2 of *blacklisted query rate* versus log2 of *queried rate*; the right panel is log2 of *non-blacklisted query rate* versus log2 of *queried rate*; a line was fitted by applying the robust linear regression to each panel. Similarly Figure 2.13 is for IP 162.248.73.103 with overall blacklisted friction 61.7%. Both Figure 2.12 and Figure 2.13 show a strong linear relationship between the log2 of listing rate and log2 of queried rate for both IPs. And both the slope value of k_1 and k_2 are 1.0 for the two IPs. We summarize the slope information across the 100 IPs and Figure 2.14 displays the distribution of the slope values. It has two panels: the left panel is the fitted slope distribution of blacklisted rates k_1 and the right panel is for non-blacklisted fitting k_2. As we can see from Figure 2.14, both the value of k_1 and k_2 are concentrated at value 1.0.

We apply the same analysis to all the 30,088 IPs in Rhipe and summarize the $30088 \cdot 2 = 60,176$ fittings in Figure 2.15 where we can conclude that the same conclusion holds: the slopes of k_1 and k_2 are concentrated at value 1.0. Furthermore, the distribution plot of the two slopes difference $k_1 - k_2$ in Figure 2.16 shows that $k_1 - k_2$ is concentrated at value 0. All

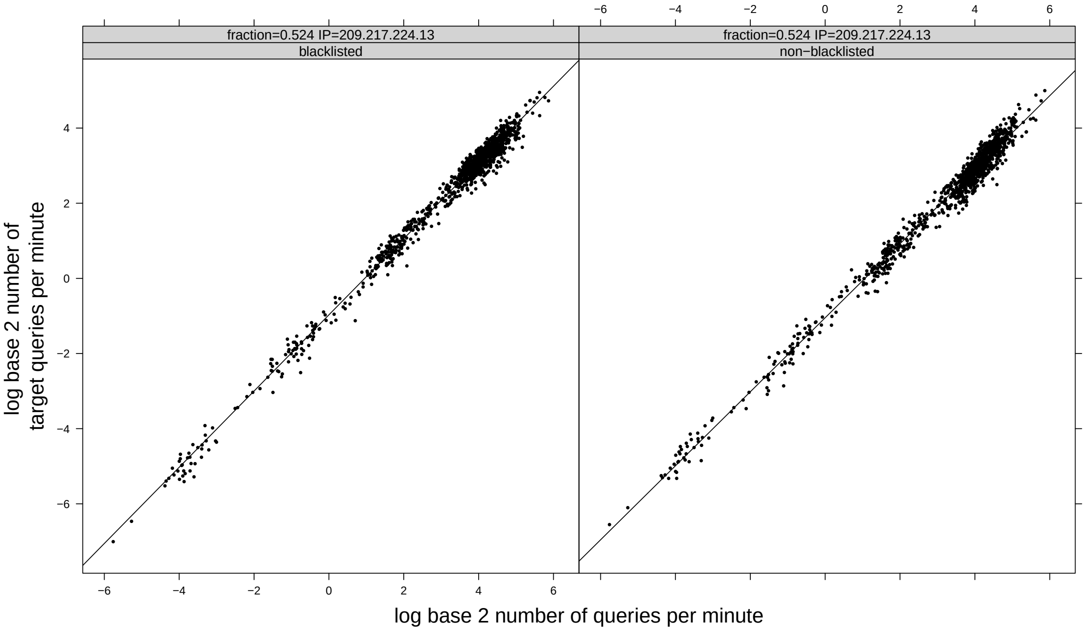

Fig. 2.12.: Log2 listing rate vs. log2 query rate example1

the discussions above have lead to the following important findings about the blacklisted properties of IPs:

- The blacklisting rate is proportional to the queried rate, which means the quicker the IP was queried, the faster the IP gets blacklisted.

- $k_1 = k_2$ for the most selected *BBQ-IP*s. For each group of 64 consecutive queries per IP, we can derive the following relationship between the blacklisted query numbers BL_i and non-blacklisted numbers $NonBL$ by applying Equation 2.1, 2.2 and 2.5 to model 2.6 and model 2.7:

  $$\frac{\mathrm{BL}_i}{\mathrm{NonBL}_i} \propto \text{queried rate}_i^{(k_1-k_2)}$$

  If $k_1 = k_2$, this suggests $\frac{\mathrm{BL}_i}{\mathrm{NonBL}_i}$ is constant across all the groups of 64 consecutive queries per IP. Furthermore, based on Equation 2.5, this suggests the blacklisted fraction $\frac{\mathrm{BL}_i}{64}$ is constant across every 64 queries within IP.

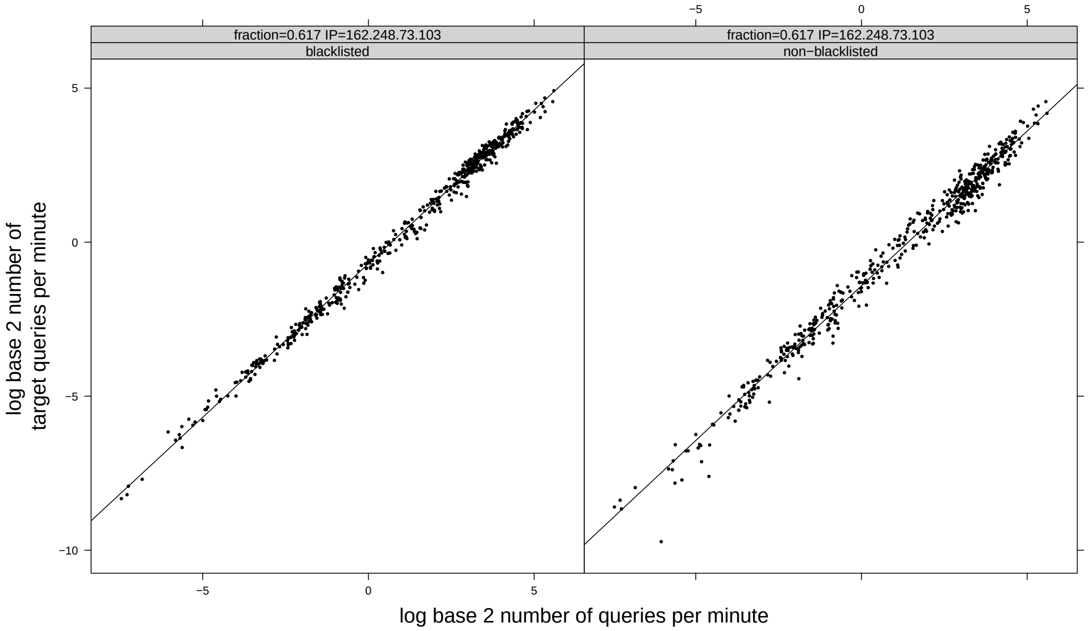

Fig. 2.13.: Log2 listing rate vs. log2 query rate example2

- These suggest a rapid back and forth of blacklisting and non-blacklisting activities: the blacklisting runs then the not-blacklisting runs, rather than long periods of each.

2.4.2 Sample Analysis – On-Off Process

Section 2.4.1 explores the blacklisted properties by calculating blacklisted frictions. Now we define a different metric to understand more blacklisted behaviours from time perspective.

A marked point process is a point process and associate marks [44], which can be expressed as:

$$\{(t_i, m_i) : i = 1, ..., n\},$$

where $t_1, ..., t_n$ are about locations information and $m_1, ..., m_n$ are associated marks. For the 30,088 subsets, each is a marked point process. The point process t_i is the query time and the marks m_i are relevant variables for each query time. Each row of the subset is a query to validate the status of IP. These blacklisted queried IPs time profiles are detailed data.

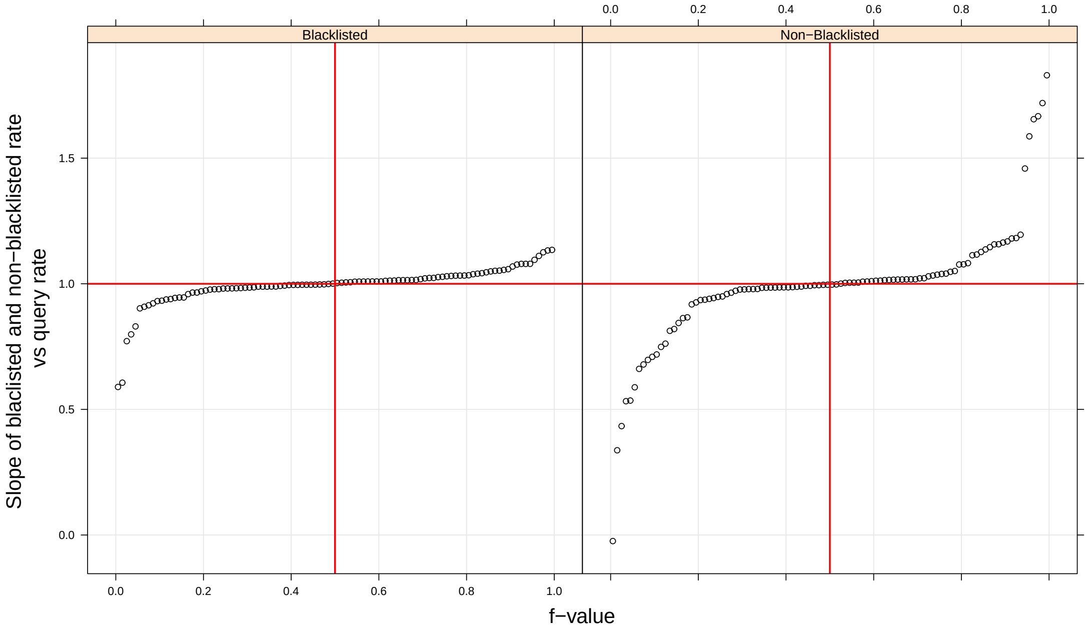

Fig. 2.14.: Distribution of slopes of robust linear regression for 100 IPs.

Now we define the On-Off run Process to understand how long a IP can stay as blacklisted or non-blacklisted:

- *On* interval is the consecutive blacklistings.
- *Off* interval is the consecutive non-blacklisting.
- Blacklisting counts: consecutive blacklisting count.
- Non-Blacklisting counts: consecutive non-blacklisting count.

For example, suppose we have an queried IP with consecutive queries status: 0, 1, 1, 1, 0, 0, 0, where value 1 stands for a blacklisted query and 0 stands for a non-blacklisted query, we can see there is a *On* and *Off* runs switch: off, on, off. This is alternating time process. An important question is what the distributions of the lengths of the *On* and *Off* intervals are. Our measuring device is the queries themselves. The *On* process ends when there is a 1 followed by 0. But the *On* process could end anytime between the 1 and 0. Similarly, the *On* process begins when there is a 0 followed by 1. However, its start time

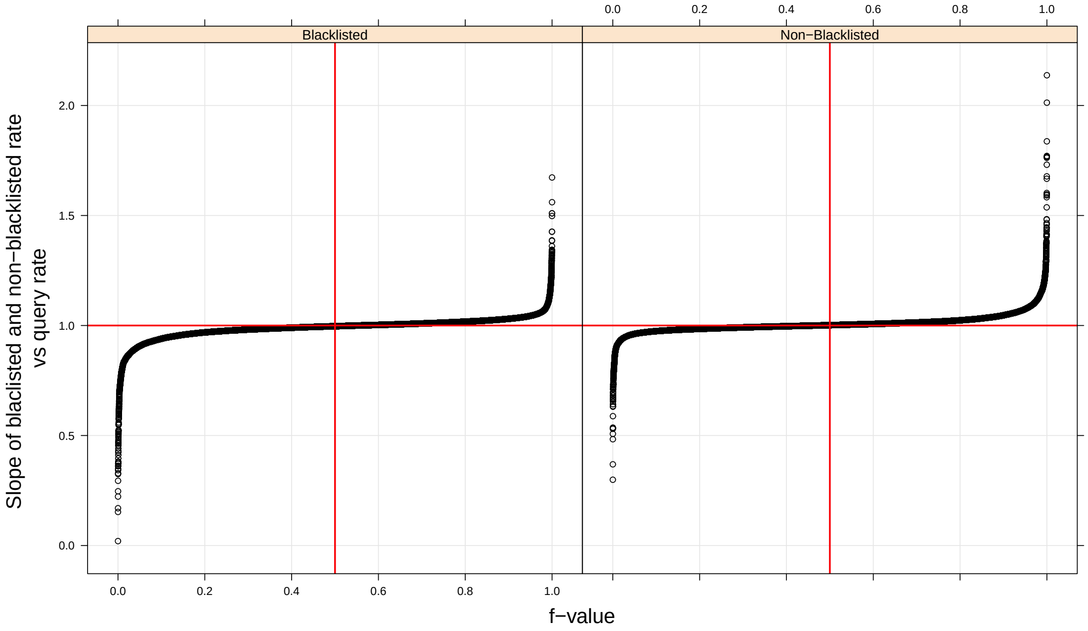

Fig. 2.15.: Distribution of slopes of all 30,088 IPs.

could begins anytime between 0 and 1. The best we can do is the upper bound and lower bound for the length of the interval. Similarly is for the *Off* process. Here we define the *On* and *Off* lower bound and upper bound:

- On Time Lower Bound $L_{1,i}$: the duration between the first blacklisted query and the last blacklisted query in one run i.

- On Time Upper Bound $U_{1,i}$: the duration between the last non-blacklisting of one run and the next non-blacklisting i.

- Off Time Lower Bound $L_{0,j}$: the duration between the first non-blacklisted query and the last non-blacklisted query in one run j.

- Off Time Upper Bound $U_{0,j}$: the duration between the last blacklisting of one run j and the next blacklisting.

We illustrate the lower bound and upper bound calculation by using the last example query lists: 0, 1, 1, 1, 0, 0, 0 with corresponding time t_1, t_2, t_3, t_4, t_5, t_6, t_7. The *On* time

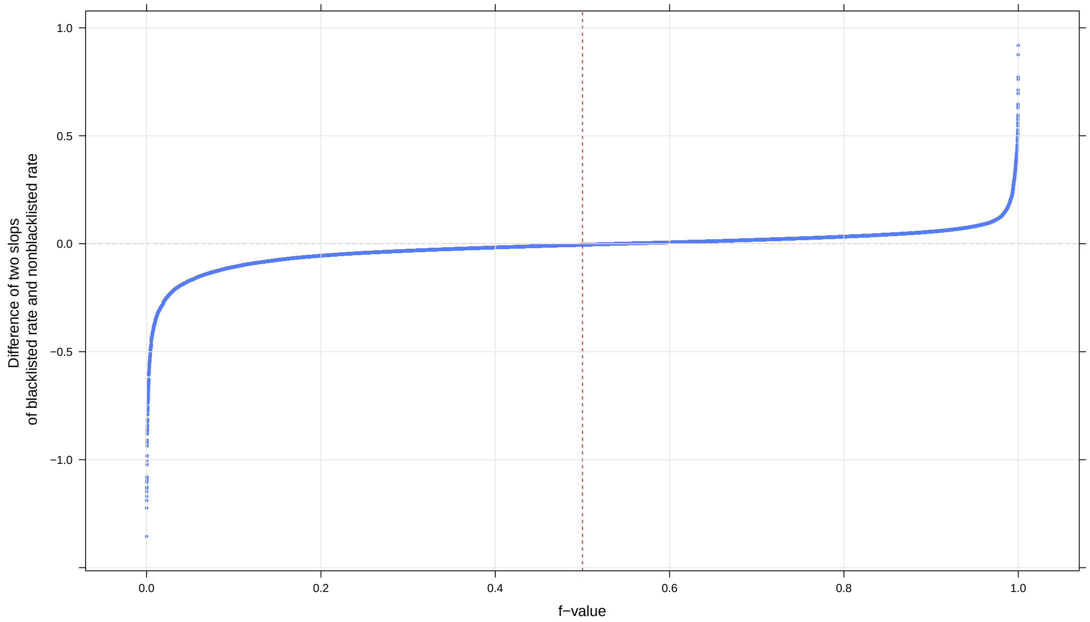

Fig. 2.16.: Distribution of slope difference k1 - k2 of 30,088 IPs.

lower bound for the first *On* run is $L_{1,1} = t_5 - t_1$; The *On* time upper bound for the first *On* run is $U_{1,1} = t_4 - t_2$. Similar calculation holds for *Off* process. For each IP, we can calculate a sequence of lower bound and upper bound for On and Off process: *On* lower bound list $\{L_{1,1}, L_{1,2}, ..., L_{1,n}\}$; *On* upper bound list $\{U_{1,1}, U_{1,2}, ..., U_{1,n}\}$; *Off* lower bound list $\{L_{0,1}, L_{0,2}, ..., L_{0,m}\}$; and *Off* upper bound list $\{U_{0,1}, U_{0,2}, ..., U_{0,m}\}$. We use Trellis display to analyze the distributions of the lower and upper bound list for On-Off process for each IP. So the distribution plot for *On* process contains 100 pages for the selected 100 IPs: each page is the upper bound and lower bound distribution for a specific IP. Similar holds for *Off* analysis. Figure 2.17 illustrate one IP's *On* upper bound and lower bound distributions: the red dot is for upper bound; the black dot is for lower bound; a horizontal line with value $log_2(10minutes)$ was plotted. As we can tell from Figure 2.17, more than 99% of the runs have both upper bound and lower bound less than 10 minutes, which suggests that the time duration for a IP to stay as blacklisted is less than 10 minutes. This means there is very fast non-blacklisted process which violates our understanding about Spamhaus service. We would expect that once a IP address is listed on the Spamhaus blacklisted list zones, it will persist a blacklisted status in a time order of hours and days. The administrator for a blacklisted IP needs to contact the Blocklist Removal Center [32] to apply for the removal from the detected lists.

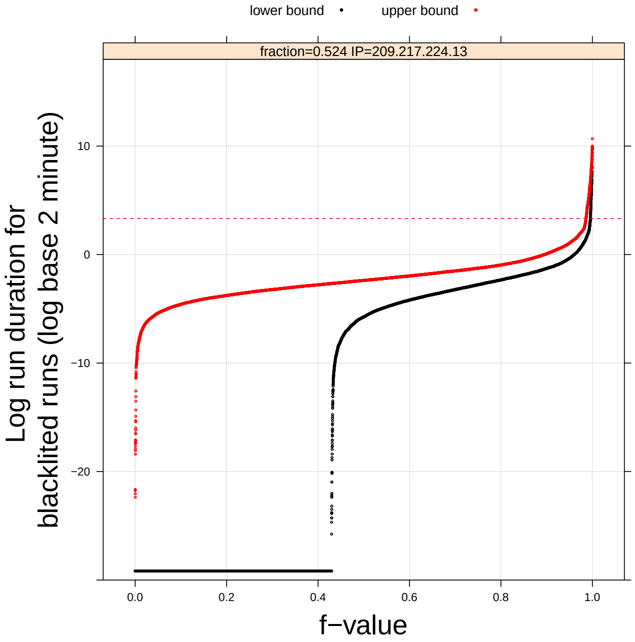

Fig. 2.17.: One upper bound and lower bound distributions example

We calculated the medium of the lower bound and upper bound for each IP *On* and *Off* process and summarize their distributions in Figure 2.18 and Figure 2.19. Figure 2.18 describes the distribution of mediums of upper bound with two panels: the left panel is the upper bound for the *On* process and the right panel is for *Off* process; a red horizontal line specifying 10 minutes was put to each panel. As we can see from this plot, the mediums of both *On* and *Off* panels are between 7 minutes to 10 minutes. Figure 2.19 describes the distribution of mediums of lower bound with two panels: the left panel is the lower bound for the *On* process and the right panel is for *Off* process; a red horizontal line specifying 1 minutes was put to each panel. As we can see from this plot, both panels has a large percentage of zero lower bound which happens when there is a quick blacklisting and non-blacklisting alternating: {0, 1, 0 } or { 1, 0, 1 }. There is only one blacklisting following the non-blacklisting in the sequence of {0, 1, 0} so the lower bound for blacklisting persisting is 0.

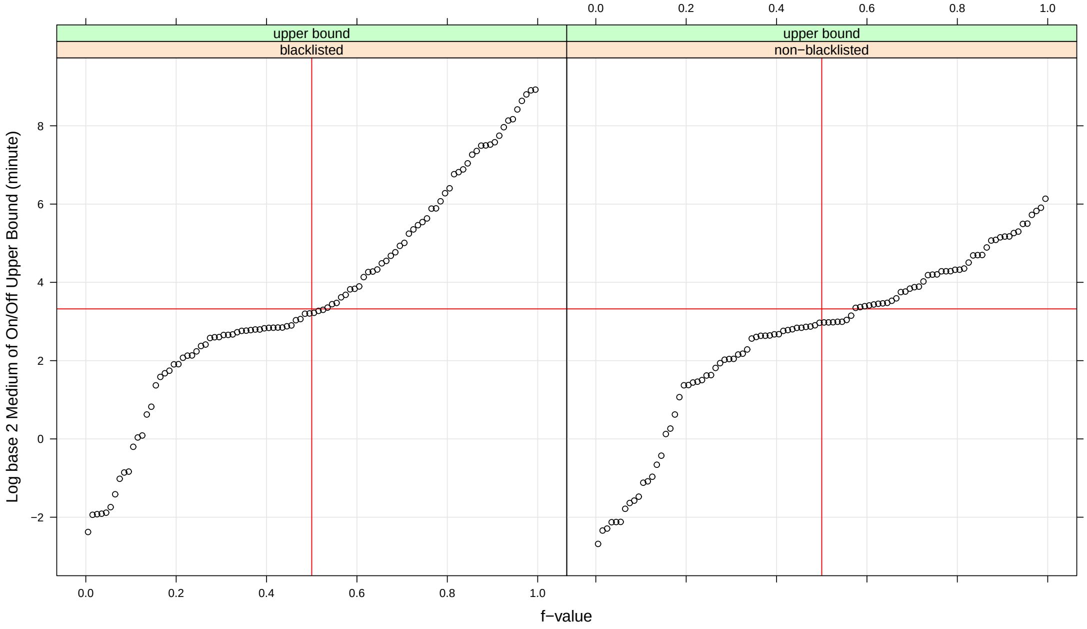

Fig. 2.18.: On-Off medium distribution of upper bound across 100 IPs

We apply the same analysis of the 100 IPs to all the 30,088 hosts and get the similar conclusions. Figure 2.20 and 2.21 describe the medium distribution of upper and lower bound of On-Off process across all the 30,088 IPs. The 2.21 suggests there are a large percentage of zero lower bound and quick switch of one blacklisted followed by non-blacklisted or the other way around. And as we can see from Figure 2.20, the medium of upper bound is between 7 minutes to 10 minutes for both *On* and *Off* process.

The Table 2.2 provides a reasonable explain for the fast back and forth blacklisting and non-blacklisting activities. Table 2.2 shows the three list of blacklisted zones for IP Spamhaus service, their corresponding blacklisted types and updating time. XBL [35] and SBL [36] are the top two largest spam zones in the Spamhaus service covering the most of blacklisted types of queries. Both XBL and SBL have over 80 public DNSBL mirror servers located around the world. Each mirror server is running independently to provide IP check service. And query checks are sent and saved locally for the most efficiencies and

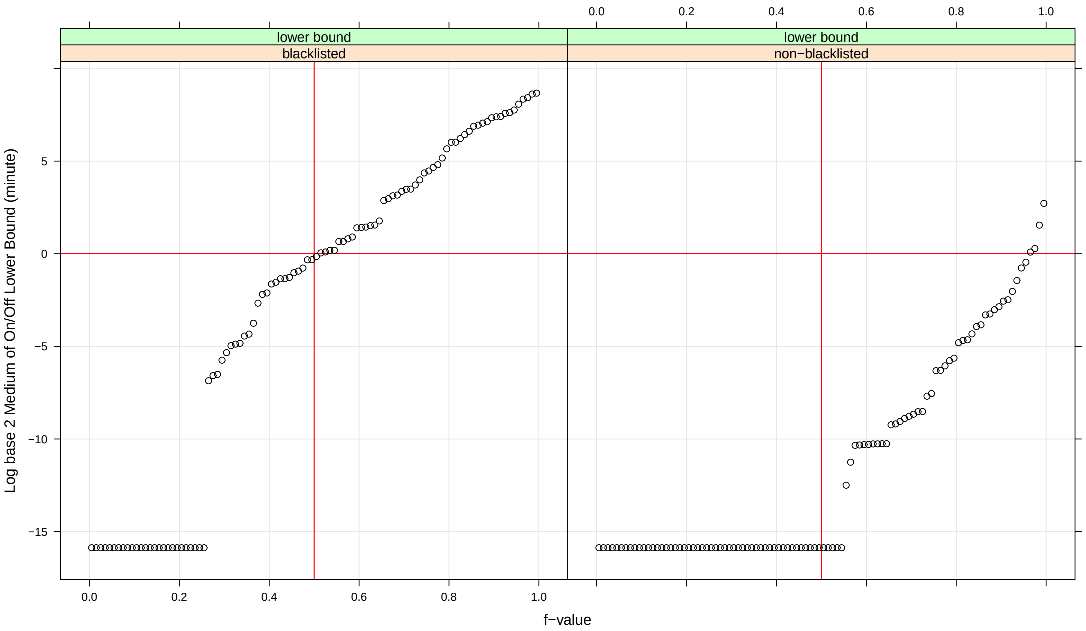

Fig. 2.19.: On-Off medium distribution of lower bound across 100 IPs

speed. The check and respond speed is in milliseconds. Both the XBL and SBL DNS zone are rebuilt and reloaded every 10 minutes, 24 hours and 7 days a week, "to ensure that new 'no Unauthenticated SMTP allowed' IP addresses are blocked and that any mistaken listings are swiftly removed" [35] [36]. The updating time for PBL zone [38] is 15 minutes as shown in Table 2.2. The fast blacklisting and non-blacklisting back and forth might be related to the Blacklisted lists Zone rebuilt frequency. A IP can send emails to different locations and different queries are sent to different Spamhaus servers. The local lists zone are updated every 10 or 15 minutes so they may make different responses globally. Figure 2.22 displays the dotplot of the distribution of blacklisted types of all queries where we can see *Exploit* from XBL zone is the most frequent cause of spamhaus violations with more than 55% of the total violations. Spam and CSS Spam from SBL zone are one of the most frequent cause too. Actually, violations related to XBL and SBL take more than 99% in our total violation queries. The medium of upper bound of *On* interval is 7 minutes to 10 minutes from Figure 2.20 and it is a reasonable conclusion that the fast back and forth blacklisting and non-blacklisting is related to the 10 minutes of SBL and XBL updating

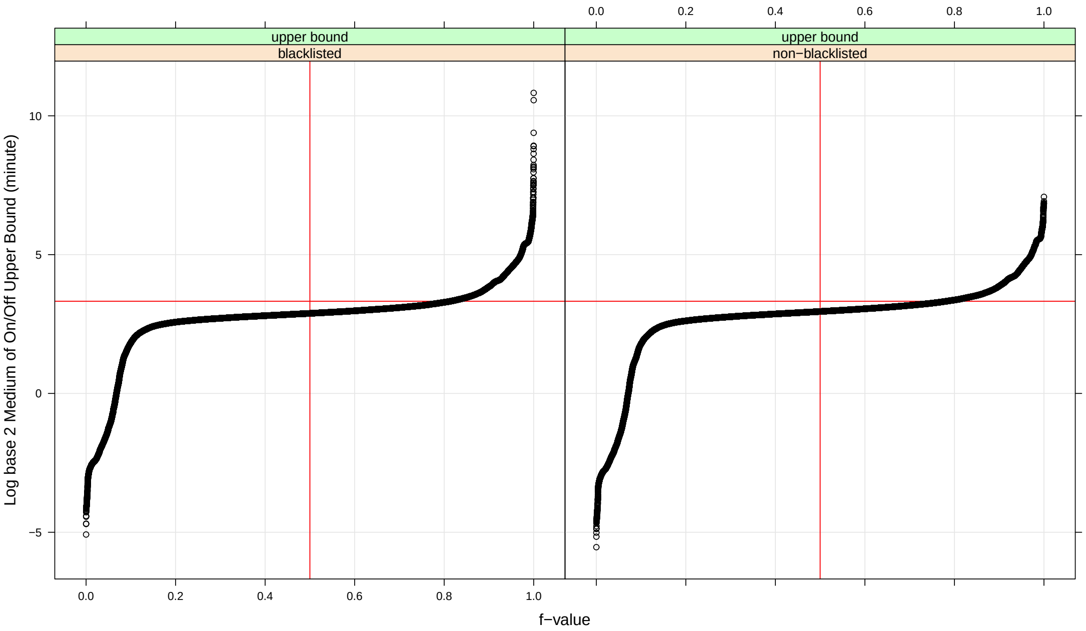

Fig. 2.20.: On-Off medium distribution of upper bound across 30,088 IPs

frequency. It is worth noticing that the updating time was 10 minutes for both XBL and SBL, 15 minutes for PBL when we check the Spamhaus website in February 2019. And the updating time has reduced to 5 minutes for XBL and SBL, 10 minutes for PBL when we recheck on November 2020. So if we recollect the most recent data, we would expect the upper bound for the consistent *On* interval will be reduced from 10 minutes to close 5 minutes.

Table 2.2.: Blacklisted Lists and its Blacklisted types

Blacklisted Lists	**Blacklisted Type**	**Updating Time (minute)**
XBL	Exploit	10
SBL	Spam, CSS Spam	10
PBL	ISP Violation, Spamhaus Policy Violation	15

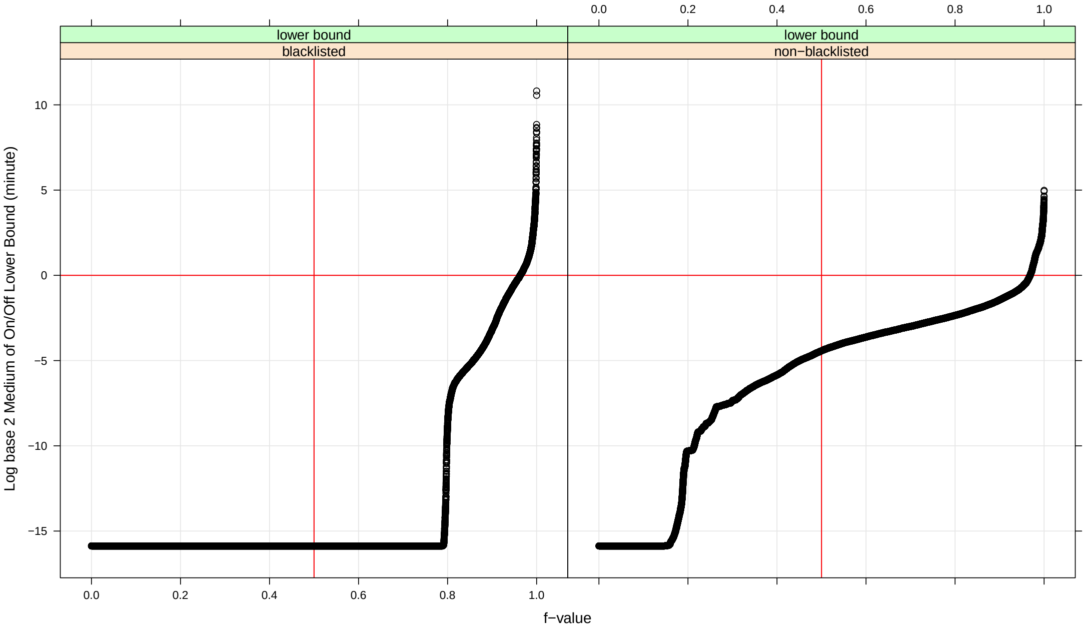

Fig. 2.21.: On-Off medium distribution of lower bound across 30,088 IPs

2.5 Conclusion

We provide a reasonable explain for the quick back and forth blacklisting and non-blacklisting phenomenon by analyzing the blacklisted frictions and On-Off process. Our conclusion is this is driven by the Spamhaus list zones updating and rebuilding frequency. There are two major challenges of the Spamhaus data analysis: one is the large input of the 1.0 TB raw data; second is to form some useful data metrics to understand and explain the blacklisted properties. The former was accomplished by applying D&R to the data analysis so that we form meaningful data divisions and are able to carry out the big data analysis. The latter was done by analyzing the universal statistical distributions and carrying out the sample analysis to recover the blacklisted behaviours. There are still other works to be done to better understand the Spamhaus data. Our current work is based on the queried IPs and we can form a more complicated data structure by forming Queried IP-Query IP division so there might be some interesting findings among interactions of queried IPs and querying IPs.

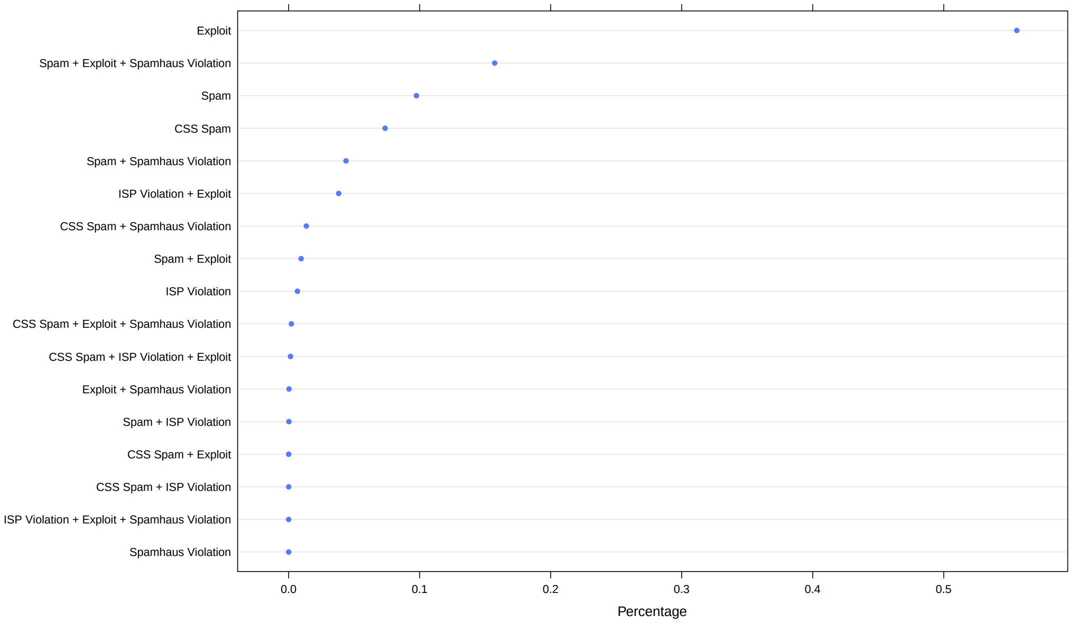

Fig. 2.22.: Distribution of blacklisted types of all queries

REFERENCES

REFERENCES

[1] S. Guha, R. Hafen, J. Rounds, J. Xia, J. Li, B. Xi, and W. S. Cleveland, "Large complex data: divide and recombine (d&r) with rhipe," *Stat*, vol. 1, no. 1, pp. 53–67, 2012.

[2] W. S. Cleveland and R. Hafen, "Divide and recombine (d&r): Data science for large complex data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 7, no. 6, pp. 425–433, 2014.

[3] "Deltharho website," http://deltarho.org/.

[4] W.-w. Tung, A. Barthur, M. C. Bowers, Y. Song, J. Gerth, and W. S. Cleveland, "Divide and recombine (d&r) data science projects for deep analysis of big data and high computational complexity," *Japanese Journal of Statistics and Data Science*, vol. 1, no. 1, pp. 139–156, 2018.

[5] R. Hafen, "Divide and recombine: Approach for detailed analysis and visualization of large complex data." 2016.

[6] X. Tong, "Divide and recombined for large complex data: Nonparametric-regression modelling of spatial and seasonal-temporal time series," https://docs.lib.purdue.edu/open_access_dissertations/1018, 2016, open Access Dissertations.

[7] Q. Liu, "Divide and recombine for large and complex data: Model likelihood functions using mcmc and trmm big data analysis," https://docs.lib.purdue.edu/dissertations/AAI10822153/, 2018, open Access Dissertations.

[8] "Trelliscope tutorial," http://deltarho.org/docs-trelliscope/.

[9] R. C. Team *et al.*, "R: A language and environment for statistical computing," 2013.

[10] T. White, *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[11] "Hadoop website," http://hadoop.apache.org/.

[12] S. Guha, "Rhipe-r and hadoop integrated processing environment," *Retrieved on August*, vol. 20, p. 2016, 2010.

[13] "Rhipe tutorial," http://deltarho.org/docs-RHIPE/.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST).* Ieee, 2010, pp. 1–10.

[15] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning spark: lightning-fast big data analysis.* " O'Reilly Media, Inc.", 2015.

[16] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[17] "Sequence file," https://cwiki.apache.org/confluence/display/HADOOP2/SequenceFile.

[18] "R contributed packages," https://cran.r-project.org/web/packages/.

[19] J. Li, "The tessera d&r computational environment: Designed experiments for r-hadoop performance and bitcoin analysis," https://docs.lib.purdue.edu/open_access_dissertations/317, 2014, open Access Dissertations.

[20] D. Heger, "Hadoop performance tuning-a pragmatic & iterative approach," *CMG Journal*, vol. 4, pp. 97–113, 2013.

[21] C.-O. Chen, Y.-Q. Zhuo, C.-C. Yeh, C.-M. Lin, and S.-W. Liao, "Machine learning-based configuration parameter tuning on hadoop system," in *2015 IEEE International Congress on Big Data*. IEEE, 2015, pp. 386–392.

[22] "Mapreduce performance tuning," https://data-flair.training/blogs/mapreduce-performance-tuning/.

[23] "Spark officially sets a new record in large-scale sorting," https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html.

[24] D. G. Kleinbaum, K. Dietz, M. Gail, M. Klein, and M. Klein, *Logistic regression*. Springer, 2002.

[25] "Simon davies and r core team. fitting generalized linear models." https://stat.ethz.ch/R-manual/R-devel/library/stats/html/glm.html.

[26] S. B. Joshi, "Apache hadoop performance-tuning methodologies and best practices," in *Proceedings of the 3rd acm/spec international conference on performance engineering*, 2012, pp. 241–242.

[27] R. Huang and W. Xu, "Performance evaluation of enabling logistic regression for big data with r," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015, pp. 2517–2524.

[28] "Disk block size," https://docs.oracle.com/cd/E19253-01/817-5093/fsfilesysappx-9/index.html.

[29] "Itap research computing," https://www.rcac.purdue.edu/services/communityclusters/.

[30] "Single cluster configuration," https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html.

[31] J. Postel, "Dod standard internet protocol," *ACM SIGCOMM Computer Communication Review*, vol. 10, no. 4, pp. 12–51, 1980.

[32] "The spamhaus website," https://www.spamhaus.org.

[33] "The spamhaus project," https://www.spamhaus.org/organization/.

[34] "The spamhaus dnsbl website," https://www.spamhaus.org/faq/section/DNSBL%20Usage.

[35] "Spamhaus xbl zone," https://www.spamhaus.org/xbl.

[36] "Spamhaus sbl zone," https://www.spamhaus.org/sbl/policy/.

[37] "Spamhaus css zone," https://www.spamhaus.org/css/policy/.

[38] "Spamhaus pbl zone," https://www.spamhaus.org/pbl.

[39] A. Barthur, "Computational environment for modeling and analysing network traffic behaviour using the divide and recombine framework," Ph.D. dissertation, Purdue University, 2016.

[40] TCPDump Working Group, "TCPDump - Packet Collection Tool," 2014. [Online]. Available: http://www.tcpdump.org

[41] "Python dpkt package," https://github.com/ashrith/dpkt.

[42] "Spamhaus dbl zone," https://www.spamhaus.org/dbl/.

[43] "Robust linear regression in r," https://www.rdocumentation.org/packages/MASS/versions/7.3-53/topics/rlm.

[44] D. J. Daley and D. Vere-Jones, "An introduction to the theory of point processes, volume 1: Elementary theory and methods," *Verlag New York Berlin Heidelberg: Springer*, 2003.

APPENDICES

A. BASH FILE SAMPLE

Here is the bash command sample to carry out experiment of $v = 3$ and $blocksize = 64MB$: `run_experiment.v3.64.sh` .

```
#!/bin/bash
n=30
v=3
m=9
mapper=23
blocksize=67108864
run=1
while (( $m <= 17 ))
do
  nohup Rscript --vanilla 1.divide.R
                 $n $v $m $mapper $blocksize > 1.$m.rout 2>&1
  rm 1.$m.rout
  while (( $run <= 3 ))
  do
    nohup Rscript --vanilla 2.experiment.object.time.R
                 $n $v $m $blocksize $run > 2.m$m.rout 2>&1
    rm 2.m$m.rout
    nohup Rscript --vanilla 3.experiment.total.time.R
                 $n $v $m $blocksize $run > 3.m$m.rout 2>&1
    rm 3.m$m.rout
    let run+=1
  done
  let m+=1
  run=1
done
```

B. PERFORMANCE FUNCTIONS R SCRIPT

Here is the Performance Functions R script `0.performance.function.R`.

```
library(Rhipe)
rhinit()
rhoptions(zips = "/wsc/song273/bin/R.Pkg.tar.gz")
rhoptions(runner = "sh ./R.Pkg/library/Rhipe/bin/RhipeMapReduce.sh")
#n=30
#r.vec = 10:23
#m.vec = n - r.vec

divide_data_old_way = function(n, v, m, mappers,
                               blocksize=134217728 ){
  print( paste( "blocksize is:", blocksize ) )
  dm = list()
  p = 2^v-1
  dir.exp = paste( "/wsc/song273/pf/n",n,"/v",v,"/", sep="")
  dm$map = expression({
    for (r in map.values){
      set.seed(r)
      value = matrix(c(rnorm(m*p),
                       sample(c(0,1), m, replace=TRUE)),
                     ncol=p+1)
      rhcollect(r, value) # key is subset id
    }
  })
  dm$input = c(2^(n-m),mappers)
  dm$output = paste(dir.exp,”divide/m”,m, sep=””)
  dm$jobname = dm$output
  dm$mapred = list(
    mapreduce.task.timeout=0,
    mapreduce.job.reduces=0,## no reduce
    mapreduce.job.maps=mappers, ## number of mappers
    dfs.blocksize = blocksize
    #     mapreduce.map.memory.mb = 6000,
    #     mapreduce.map.java.opts =”–Xmx4800m”
  )
  dm$parameters = list(m=2^m, p=p)
  dm$readback = FALSE
  dm$jobname = dm$output
  dm$noeval = TRUE
  dm.mr = do.call(”rhwatch”, dm)
  rhex(dm.mr, async=FALSE)
  print(m)
  rhclean()
  return()
}


object_time = function(n, v, m, run,
                              path=”performance/rst/”,
                              blocksize=134217728
){
  # m = 10
  # n = 30
  # v = 4
  # run = 1
  # path is the local path to save the result
# path = ”/home/song273/performance/results/”
# dir.exp is the hdfs path
dir.exp = paste( ”/wsc/song273/pf/n”,n,”/v”,v,”/”, sep=””)
bl = 2^(log2(blocksize)- 20)
name1 = paste(’n’,n, ’v’, v, ’bl’, bl,sep=””)
dir.local = path
timing = data.frame()
## timing for O
p = 2^v-1
type = ”O”
dir.dm = paste(dir.exp,”divide/m”,m, sep=””)
dir.nf = paste(dir.exp,”nf/m”,m,’/run’,run, sep=””)
nf = list()
nf$map = expression({})
nf$mapred = list(
  mapreduce.task.timeout=0,
  mapreduce.job.reduces=1,
  #rhipe_map_buff_size=2^15,
  dfs.blocksize = blocksize
)
nf$parameters = list(p=p)
nf$input = dir.dm
nf$output = dir.nf
nf$jobname = nf$output
nf$noeval = TRUE
nf.mr = do.call(’rhwatch’, nf)
t = as.numeric(system.time({rhex(nf.mr, async=FALSE)})[3])
## division size in gb
input_tmp = rhls(dir.dm)[-1,]
division_size = 2^(log2(sum(input_tmp$size))-30)
num_file = nrow(rhls(dir.dm))
```

```
  t = data.frame(type=type,n=n,p=p,m=m,run=run,t=t,blocksize=bl,
                   division_size=division_size,num_file=num_file)
  timing = rbind(timing,t)
  #save(timing, file=paste(dir.local, name1, ”.RData”, sep=””))
  write.table(timing,
               file=paste(dir.local, name1, ”.csv”, sep=””),
               append=TRUE,
               sep = ”,”,
               row.names = FALSE,
               col.names = FALSE
  )
  print(timing)
  Sys.sleep(time=120)
}

total_time = function(n, v, m, run,
                          path=”performance/rst/”,
                          blocksize=134217728
){
  # m = 10
  # n = 30
  # v = 4
  # run = 1
  # path = ”/home/song273/performance/results/”
  bl = 2^(log2(blocksize)- 20)
  name1 = paste(’n’,n, ’v’, v, ’bl’, bl,sep=””)
  dir.exp = paste( ”/wsc/song273/pf/n”,n,”/v”,v,”/”, sep=””)
  dir.local = path
  timing = data.frame()
  p = 2^v-1
  type = ”T”
```

```
dir.dm = paste(dir.exp,”divide/m”,m, sep=””)
dir.gf = paste(dir.exp,”nf/m”,m,’/run’,run, sep=””)
gf = list()
gf$map = expression({
  for (v in map.values) {
    value = glm.fit(v[,1:p],v[,p+1],family=binomial())$coef
    rhcollect(1, value)
  }
})
gf$reduce = expression(
  pre = {
    v = rep(0,p)
    nsub = 0
  },
  reduce = {
    v = v +
        colSums(matrix(unlist(reduce.values), ncol=p, byrow=TRUE))
    nsub = nsub + length(reduce.values)
  },
  post = { rhcollect(reduce.key, v/nsub) }
)
gf$mapred = list(
  mapreduce.task.timeout=0,
  mapreduce.job.reduces=1,
  #rhipe_map_buff_size=2^15,
  dfs.blocksize = blocksize
)
gf$parameters = list(p=p)
gf$input = dir.dm
gf$output = dir.gf
gf$jobname = gf$output
```

```
    gf$noeval = TRUE
    gf.mr = do.call(’rhwatch’, gf)
    t = as.numeric(system.time({rhex(gf.mr, async=FALSE)})[3])
    ## division size in gb
    input_tmp = rhls(dir.dm)[-1,]
    division_size = 2^(log2(sum(input_tmp$size))-30)
    num_file = nrow(rhls(dir.dm))
    t = data.frame(type=type,n=n,p=p,m=m,run=run,t=t,blocksize=bl,
                   division_size=division_size,num_file=num_file
    )
    timing = rbind(timing,t)
    write.table(timing,
                file=paste(dir.local, name1, ”.csv”, sep=””),
                append=TRUE,
                sep = ”,”,
                row.names = FALSE,
                col.names = FALSE
    )
    print(timing)
    Sys.sleep(time=120)
}
```

C. DIVIDE STEP R SCRIPT

Here is Divide Step R script `1.divide.R`.

```
#!/usr/bin/env Rscript
args = commandArgs(trailingOnly=TRUE)
### load function
source(”0.performance.function.R”)

n = as.numeric( args[1] )
v = as.numeric( args[2] )
#r.vec = 10:23
m.vec = as.numeric( args[3] )
mappers = as.numeric( args[4] )
blocksize = as.numeric( args[5] )

#### 1. divide
for( m in m.vec ){
  start_t = proc.time()[3]
  divide_data_old_way(n, v, m, mappers, blocksize=blocksize)
  divide_t =  proc.time()[3]
  long = round( ( divide_t - start_t )/60, 2)
  print(paste(”Divide: it takes ”, long, ” minutes for m”, m, sep=””) )
}
```

D. OBJECT TIME MEASUREMENT R SCRIPT

Here is the R script `2.experiment.object.time.R` to measure the object time.

```
#!/usr/bin/env Rscript
args = commandArgs(trailingOnly=TRUE)
### load function
source(”0.performance.function.R”)

n = as.numeric( args[1] )
v = as.numeric( args[2] )
#r.vec = 10:23
m.vec = as.numeric( args[3] )
blocksize = as.numeric( args[4] )

run = as.numeric( args[5] )

#### 2. object time
for( m in m.vec ){
  start_t = proc.time()[3]

  object_time( n, v, m, run,blocksize=blocksize)

  object_t =  proc.time()[3]
  long = round( ( object_t - start_t )/60, 2)
  print(paste(”Object time: it takes ”, long,
             ” minutes for m”, m, ” in the run”, run, sep=””) )
}
```

E. TOTAL TIME MEASUREMENT R SCRIPT

Here is the R script `3.experiment.total.time.R` to measure the total time.

```
#!/usr/bin/env Rscript
args = commandArgs(trailingOnly=TRUE)
### load function
source(”0.performance.function.R”)

n = as.numeric( args[1] )
v = as.numeric( args[2] )
#r.vec = 10:23
m.vec = as.numeric( args[3] )
blocksize = as.numeric( args[4] )
run = as.numeric( args[5] )

#### 3. total time
for( m in m.vec){
  start_t = proc.time()[3]

  #### 3. total time
  total_time(n, v, m, run,blocksize=blocksize)

  total_t =  proc.time()[3]
  long = round( ( total_t - start_t )/60, 2)
  print(paste(”Total time: it takes ”, long, ” minutes for m”, m,
              ” in the run”, run, sep=””) )
}
```

```
#### 4. delete division
 if( run == 3){
   for( m in m.vec){
     path = paste("/wsc/song273/pf/n",n,"/v",v,"/divide/m", m, sep="")
     rhdel( path )
     rhdel(paste( "/wsc/song273/pf/n",n,"/v",v,"/nf", sep="" ))
     print( paste("Delete division for n", n, "v", v, "m", m,
                  "in the run", run, sep = "") )
   }
 }
```

VITA

VITA

Yuying Song was born in Laizhou, Shandong, China. She entered University of Science and Technology of China in August 2008 and received the degree of Bachelor of Science in Statistics in July 2012. She got admitted by the Statistics Department of Purdue University in 2013 and earned her Master of Science degree in Mathematical Statistics in May 2019. Her academic interests include statistical modeling, divide and recombine in the big data analysis, data visualization and experiment design.