View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Purdue E-Pubs

Purdue University
Purdue e-Pubs

Open Access Dissertations | Theses and Dissertations

Fall 2014

The Tessera D&R computational environment: Designed experiments for R-Hadoop performance and Bitcoin analysis

Jianfu Li
Purdue University

Follow this and additional works at: https://docs.lib.purdue.edu/open_access_dissertations


Part of the Statistics and Probability Commons

Recommended Citation
Li, Jianfu, "The Tessera D&R computational environment: Designed experiments for R-Hadoop performance and Bitcoin analysis" (2014). *Open Access Dissertations*. 317.
https://docs.lib.purdue.edu/open_access_dissertations/317

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

Graduate School Form 30
(Revised 08/14)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By JIANFU LI

Entitled
THE TESSERA D&R COMPUTATIONAL ENVIRONMENT:
DESIGNED EXPERIMENTS FOR R-HADOOP PERFORMANCE
AND BITCOIN ANALYSIS

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

WILLIAM S. CLEVELAND

JUN XIE

LINGSONG ZHANG

BOWEI XI

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification/Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

WILLIAM S. CLEVELAND

Approved by Major Professor(s): _____

_____

Approved by: JUN XIE 10/15/2014

Head of the Department Graduate Program Date

THE TESSERA D&R COMPUTATIONAL ENVIRONMENT: DESIGNED EXPERIMENTS FOR R-HADOOP PERFORMANCE AND BITCOIN ANALYSIS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Jianfu Li

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2014

Purdue University

West Lafayette, Indiana

To those I love and those who love me.

ACKNOWLEDGMENTS

It has been a long journey, and I could not have been where I am now without the help and support of many people.

I have been very fortunate to have worked with an outstanding advisor, Dr. William S. Cleveland, from whom I have learned so much. His attitude towards conducting research, pursuit of detail perfection, and unique way of "looking" at data, have deeply influenced me during my time under his guidance and will continue to benefit me throughout my professional career.

I would like to thank the other committee members, Dr. Bowei Xi, Dr. Jun Xie, and Dr. Lingsong Zhang, for their help along the way, especially to Dr. Xi, for all the support and encouragement during the writing process of this dissertation.

I want to express my appreciation to my former and present colleagues, Ryan, Saptarshi, Jin, Xiang, Jeremiah, Ashrith, Yang, Philip, Barret, Xiaosu, Jeremy, Aritra, Yuying, and Jiasen, for all the head-scratching and laughters we have shared while computing and plotting with data.

I also want to thank all my dear friends at Purdue, who have made these years most enjoyable and memorable. Danni, thank you for being the best mentor from the first day I came to the department; Qiming, thank you for all the fun we have had as roommates and ever since; Chao, Cheng, Ritabrata, Chris, Qingling, and Han, thank you for being such an awesome class of 2008; Youran, Tim and Guy, thank you for all the beers and laughters during the fun nights; this list goes on and on, and I thank you all for spending your time with me.

I am very grateful to the Department of Statistics for providing an excellent environment, various opportunities, and a diverse group of people. The quality of my education here has been exceptional, and I have learned so much from many faculty members and fellow students. Sincere thanks to Darlene, Becca, Mary, Holly, and

Marian, for helping me with all the administrative procedures and paperwork. And a special thank goes to Doug, My, and Alex, for all the help and patience on my projects.

Last, but certainly not least, I want to thank my parents, my sister, and my beloved girlfriend, for their unconditional love, patience, and encouragement, and for making me a home that I can always return to. I owe you so much.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

API	Application Programming Interface
BTC	bitcoin, a unit of Bitcoin
CSV	Comma-separated Values
D&R	Divide-and-Recombine
GB	Gigabyte
HDFS	Hadoop Distributed File System
KB	Kilobyte
RHIPE	R and Hadoop Integrated Programming Environment
TCP/IP	Transmission Control Protocol and Internet Protocol
USD	United States Dollar

ABSTRACT

Li, Jianfu. Ph.D., Purdue University, December 2014. The Tessera D&R Computational Environment: Designed Experiments for R-Hadoop Performance and Bitcoin Analysis. Major Professor: William S. Cleveland.

D&R is a statistical framework for the analysis of large complex data that enables feasible and practical analysis of large complex data. The analyst selects a division method to divide the data into subsets, applies an analytic method of the analysis to each subset independently with no communication among subsets, selects a recombination method that is applied to the outputs across subsets to form a result of the analytic method for the entire data. The computational tasking of D&R is nearly embarrassingly parallel, so D&R can readily exploit distributed, parallel computational environments, such as our D&R computational environment, Tessera.

In the first part of this dissertation, I present a study of the performance of the Tessera D&R computational environment through designed experiments.

The base of the D&R computational environment is RHIPE, the R and Hadoop Integrated Programming Environment. R is a widely used interactive language for data analysis. Hadoop is a distributed, parallel computational environment consisting of a distributed file system (HDFS) and distributed compute engine (MapReduce). RHIPE is a merger of R and Hadoop.

The D&R framework enables a fast embarrassingly parallel computation on a cluster for large complex data that can lead to a small computational elapsed times for the applications analytic methods to all of the data. However, the time depends on many factors. The system we study is very complex and the effects of factors are complex. There are interactions, but not well understood. So we run a full factorial experiment with replicates to enable an understanding.

In the second part of this dissertation, I present an analysis of the Bitcoin transaction data utilizing the Tessera D&R computational environment.

Bitcoin is a de-centralized digital currency system. There is no central authority in the Bitcoin system to issue new money, or validate the transfer of money; both of these tasks are accomplished through the joint work of participants in the Bitcoin network. In the past two years, the Bitcoin system has become very popular, mostly due to its ease of use and embedded anonymity in the system.

The ease of use of Bitcoin is straightforward. The anonymity of the Bitcoin system, on the other hand, is rather debatable and has drawn much attention in its user community as well as the research community. We admit that a certain level of anonymity exists in the Bitcoin system, but it might not be as invulnerable as one would hope. For one thing, the entire history of Bitcoin transactions is publicly available, which provides an opportunity for passive analysis of Bitcoin usage such as ours.

I present here a study of the general statistical properties of the usage of Bitcoin transactions and the usage of Bitcoin addresses. We have also built profiles for a few groups of popular addresses among which the addresses share similar behavior. Furthermore, we provide a passive analysis of the anonymity of Bitcoin system by proposing a classification model to identify payment and change in majority of the Bitcoin transactions.

1. A MULTI-FACTOR DESIGNED EXPERIMENT FOR PERFORMANCE OF THE TESSERA D&R COMPUTATIONAL ENVIRONMENT FOR LARGE COMPLEX DATA

1.1 Divide and Recombine (D&R) for Large Complex Data

1.1.1 D&R Statistical Framework

D&R [1] [2] is a statistical framework for the analysis of large complex data that enables feasible and practical analysis of large complex data. The analyst selects a division method to divide the data into subsets, applies an analytic method of the analysis to each subset independently with no communication among subsets, selects a recombination method that is applied to the outputs across subsets to form a result of the analytic method for the entire data.

Analytic methods have two distinct categories, visualization methods whose outputs are visual displays, and number-category methods whose outputs are numeric and categorical values. In D&R, number-category analytic methods are typically applied to each of the subsets. Visualization methods are typically applied to each subset in a sample of subsets because often there are too many of them to look at plots of all [3]. The D&R result for an analytic method is almost always not the same as the result that would have occurred had it been feasible and practical to apply the method directly to all of the data. D&R research in statistical theory consists of developing division methods and recombination methods to maximize the statistical accuracy of the D&R result. Optimal choices can result in a statistical accuracy that is close to that of the direct all-data application of the method, were it feasible and practical. Statistically there are D&R efforts going on under different names, e.g., "Consensus" has been used for recombine [4].

The D&R statistical framework leads to simple, feasible, and practical computation on a cluster for the analysis of large complex data. The reason is that much of the computational tasking of D&R is nearly embarrassingly parallel, that is, no communication between the parallel processing, which is the simplest parallel processing. D&R can readily exploit computational environments that run on clusters, such as Hadoop [5] and Spark [6] [7], which enable distributing subsets across the cluster, and carrying out the parallel processing across the cluster. This also means the cluster nodes are both data nodes and compute nodes. Furthermore, for each analytic method, even iterative ones, the data in the form of subsets are read into memory only once.

1.1.2 Tessera Computational Environment

Tessera [8] is a computational environment to carry out D&R. The front end of Tessera, what the data analyst uses to program with the data, is R [9], the widely used and highly acclaimed interactive language for data analysis. The back end is the Hadoop distributed, parallel computational environment [5] consisting of a distributed file system (HDFS) [10] and distributed compute engine (MapReduce) [11]. RHIPE [12], the R and Hadoop Integrated Programming Environment, is a merger of R and Hadoop. The analyst carries out all analysis within R, using RHIPE R commands to communicate with Hadoop. This protects the analyst from having to manage the details of the Hadoop database management and parallel processing.

The analyst specifies R code for the three D&R tasks:

- divide the into subsets (D[dr] computations)
- apply the analytic method to each subset (A[dr] computations)
- recombine the outputs of the A computations and write results to the HDFS ((R[dr] computations).

In each case the analyst passes the R code to RHIPE R commands that manage the communication between R and Hadoop, and Hadoop carries out the R computations on the cluster nodes. A large part of these computations are embarrassingly parallel, which means no communication between the parallel computations, the simplest possible parallel processing.

Hadoop computes the subsets by executing the analyst D[dr] R commands, and distributes the subsets across the servers of the cluster into the Hadoop Distributed File System (HDFS). Hadoop runs the analyst A[dr] R commands on the subsets across the server cores using its Map computational procedure. Map is an embarrassingly parallel procedure, not allowing communication among the subset computations. Hadoop schedules these computations by assigning a core to a data block, or a collection of subsets. There are typically far more subsets than cores. As each subset computation ends, another collection of subsets is chosen and a core assigned to it. Hadoop attempts to optimize by assigning a core as close as possible to the subsets location, for example, on the same node as the subsets if possible. Hadoop runs the analyst R[dr] R commands on the outputs of the A[dr] computations using the Reduce computational procedure, which can compute across subsets. Aspects of the computation can be embarrassingly parallel. The D[dr] R commands can use both Map and Reduce, and much of this computation can be embarrassingly parallel.

1.1.3 Designed Experiments to Improve Performance of D&R Computations on a Cluster

The D&R framework enables a fast embarrassingly parallel computation on a cluster for large complex data that can lead to a small computational executions times for the applications of analytic methods to all of the data. However, the time depends on many factors. This presents an opportunity for optimizing the computation even further by making the best choice of the factors. Our approach to the optimization is to run statistically designed experiments. We report here on one such experiment.

The system we study is very complex and the effects of factors are complex. There are interactions, but not well understood. So we run a full factorial experiment with replicates to enable an understanding. Hadoop and the hardware architecture factors are a major part of the experimentation. There have been investigations of performance of these factors that are meant to lead to "best practices", such as [13] [14] and [15]; there have also been studies of Hadoop performance via simulation approach [16] [17], as well as through profiling and modelling [18] [19]; some studies also conducted experiments and benchmarking to explore optimal configuration settings and to seek improvements for Hadoop, such as [20] and [21]. However, we have not found comprehensive multi-factor experiments with a statistical design that can convincingly account for the effects of the factors including interactions.

1.2 Experimental Design

In our experiment, the analytic method is logistic regression. It is applied to each subset by the R function `glm.fit`. Elapsed time associated with this is the response. Quantitative results that guide optimization come from building and fitting a model that relates the response to the factors. In all runs of the experiment there are $N = 2^{30}$ observations of V variables. One variable is the dependent variable , randomly generated 0's and 1's in which the probability of a 1 is 0.5. The remaining are $V - 1$ explanatory variables, each a randomly generated normal with mean 0 and variance 1.

The data are generated, and written to the HDFS by a D[dr] computation. However, this computation is not part of the experiment. Rather, the total execution time is for the A[dr] and D[dr] computations together. The reason is that in a D&R analysis, a division resulting from a D&R division persists and is used for many analytic methods. The D[dr] computational cost is amortized across the analytic methods. This is a good thing because the D[dr] computation is often quite big. The recombination method in our experiment is to take the means of the subset estimates of the

coefficients; they are then written to the HDFS. This R[dr] computation is negligible compared with A[dr].

1.2.1 Factors, Replication, and Error Term Variability

There are 2 types of execution time that will be described shortly. Their sum is the total execution time. So this means that there is one *computation-type* categorical factor in the experiment with 2 levels. There are 2 *statistical factors* that measure characteristics of the dataset and the subsets. These factors also affect the statistical properties of the D&R estimates of the regression coefficients. One is the number of observations per subset. There are two *HDFS factors* that control aspects of the HDFS. One is the I/O buffer size. There are two *Hadoop MapReduce factors*. One is the maximum number of Map tasks allowed. There are two *hardware factors*, which are aspects of the cluster hardware architecture. One is the speed of the network connecting the nodes. Altogether there are 9 factors in the experiment. One statistical factor has 5 levels, the other statistical factor has 3, and the remaining factors each have 2.

Knowledge of 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. The reason is simply that while there has been some study of Hadoop performance one factor at time, we have been unable to find results on interactions. So we must rely on the empirical study of the experimental data to discover interactions, and a some of our knowledge of the functioning. For this reason the design is a full factorial. So there are 1920 combinations of the factors. There are 3 replicates for of each combination, so there are 5760 runs of the experiment.

Each run has full use of the cluster without other users and without other runs. However, there are known sources of variation that we cannot readily control, and leave as error variability. One source is from system processes that use a variable

amount of capacity on the cluster. We do not expect this to be large in magnitude compared with the effects of the controlled experimental factors of the experiment. Another source is the iterative fitting of *glm.fit* for each subset. The number of iterations is determined by a convergence criterion. The number varies, which means the execution times of individual fits vary. We could have fixed the number to a constant, but did not to maintain as much verisimilitude as possible. We do record the number of iterations. However, as we will see, the error variability is quite small compared with the effects of the factors. Part of the model building includes careful study of the variability.

1.2.2 Measurements and the Computation-Type Factor

There are two types of execution-time computation. The subsets are stored on the HDFS as R objects. The first computation type is O, the execution time to read the subsets from the HDFS and make them available to `glm.fit` in memory as an R objects. So O is a part of the A[dr] computation. The other type, L, starts when O ends. It consists of `glm.fit` computations on the subsets using Hadoop Map, plus gathering the subset estimates and computing the means using Hadoop Reduce. L is part of the A[dr] computation plus the R[dr] computation.

There are two types of execution-time measurements but they are not (O, L). The reason is that we cannot measure L directly by just running `glm.fit`. It needs data. So for each combination of all other factors, we measure O in one run and $T = O + L$ in another. We get O by doing just the reading into memory and forming the R object in memory. That is, we do not run `glm.fit`. We get T by running everything, reading and forming the object, applying `glm.fit`, and carrying out the recombination. There are 960 combinations of the factors other than computation type. With 3 replicates for combination, there are 2880 runs of measuring O runs. Similarly, there are 2880 T runs.

As part of our initial modeling, we hypothesize that for the measurement $T = O + L$, O (abusing notation here somewhat) has the same execution time distribution as O in the direct measurement. We check this in the model building as well as argue for it based on how we define O and L. Since we do not measure L directly, it is in effect a latent variable. To get the properties of L, we carry out a statistical model building approach that serves as a deconvolution method.

1.2.3 Statistical Factors

The number of observations of the variables, response and explanatory, is held constant to $N = 2^{30}$, so it is not a factor. There are two statistical factors. One is V, the number of variables. The second is M, the number of observations per subset, where $N/M = R$ is an integer, the number of subsets. The experimental values of M and V are shown in Table 1.1. For each $V = V_0$, we randomly generated one dataset,

Table 1.1.: Statistical factors.

Factor	Values	Description
V	2^4, 2^5, 2^6	Number of variables
M	2^{16},2^{14},$\cdots$,2^8	Number of observations per subset

response and explanatory variables, resulting in NV_0 values. This dataset is used in all runs with $V = V_0$.

There are three other variables derived from the values of N, M, and V that are important to our analysis. The first, R, is defined above. The next two, S and D, are sizes in memory, measured in bytes. Each value of the V variables in the experiment is stored in memory as double precision, which is 8 bytes. $S = 8MV$ bytes is the size of each subset. It changes across runs with M and V. $D = 8NV$ bytes is the total size of the dataset. It changes with V across runs. The values of R, S, and D taken in the experiment are shown in Table 1.2. s varies from 32 kilobytes to 32 megabytes.

S varies from 128 gigabytes to 0.5 terabytes. R varies from about 16,000 to about 4,000,000.

Table 1.2.: Derived statistical variables used in the analysis.

Factor	Values	Description
R	2^{14}, 2^{15}, $\cdots$, 2^{22}	Number of subsets.
S	2^{15}, 2^{16}, $\cdots$, 2^{25}	Size of subset in bytes.
D	2^{37}, 2^{38}, $\cdots$, 2^{39}	Size of dataset in bytes.

1.2.4 Hadoop HDFS Factors

Subsets in the experiment are put into blocks by the HDFS and stored as independent units. The size of a block, BLK, is a HDFS configuration parameter, and affects not only the storage of the dataset but also MapReduce scheduling of tasks. When a dataset is used as input to a MapReduce job, the Hadoop scheduler assigns a Map task to each block. So there are as many Map tasks as the number of blocks the dataset spans. Therefore, S, the size of the dataset, and BLK, the size of a block, determine the number of Map tasks. Given S, the smaller BLK and the larger the number of blocks, so consequently, the larger the number of Map tasks. Each Map task works with a lesser amount of data and in general this leads to better parallelization. However, more tasks increase the management workload for the NameNode and the JobTracker.

Another factor that has been shown to affect HDFS I/O performance is the Hadoop I/O buffer size, or IOB. Technically, IOB affects not only the HDFS, but also other aspects of Hadoop where I/O operations are present. IOB determines how much data is buffered in I/O pipes before transferring to other operations during read and write operations. Its default in our version of Hadoop is 4 kilobytes. This is a conservative setting. 64 kilobytes has been suggested as a good choice [15].

Table 1.3.: HDFS Factors.

Factor	Values	Description
BLK	128, 256	HDFS block size in megabytes.
IOB	4, 64	Hadoop I/O buffer size in kilobytes.

1.2.5 Hadoop MapReduce Factors

The Map task capacity, or MTC, controls the maximum number of Map tasks that can be run simultaneously on a given node. Of course, this is bounded by the number of cores on the node. Collectively on the experimental cluster which has the same number c of cores per node, the maximum that can be run is c times MTC. One might think a larger MTC should always be preferred over a smaller one. However, there is a limit on the the node hardware resources: core, memory, disk, and network. So too many tasks create bottlenecks in any one of a number of node resources. For Reduce tasks, there is a corresponding factor, RTC. We do not take this as a factor because the reduce execution time in this experiment is negligible. Table 1.4 gives the experimental values of MTC.

Map and Reduce tasks are each run on its own Java Virtual Machine (JVM) to isolate it from other running tasks. The overhead of starting new JVMs for all tasks can be significant in certain circumstances. However, MapReduce jobs that have a large number of very short-lived tasks can see performance gains when a JVM is reused for subsequent tasks. The maximum number of tasks to run for a given job for each JVM launched, $REUSE$, is a MapReduce configuration parameter. A values of 1 means JVM reuse is disabled. A value of -1 indicates no limit, so the same JVM may be used for as many tasks as possible. Table 1.4 shows the values of this factor.

Table 1.4.: MapReduce Factors.

Factor	Values	Description
MTC	12, 16	Map task capacity.
$REUSE$	True, False	Task JVM reuse.

1.2.6 Hardware Factors

The experiment was run on a cluster configured and maintained by system administrators for the Department of Statistics Purdue University. It consists of 6 Dell R515 nodes interconnected by 10 gigabit Ethernet. Collectively, the cluster has 96 cores, 384 gigabytes of RAM, and 144 terabytes of disk. Each node has dual 3.0 GHz 8-core AMD Opteron(tm) 4284 processors (16 cores); 64 GB RAM; and 12 2TB 7200 RPM nearline SAS (NL-SAS) physical disks. The cluster runs Cloudera Hadoop 0.20.2-cdh3u5; protobuf-2.4.1 protocol buffer software; RHIPE 0.73.1; R 2.15.1; and Java 1.7.0.07-b10.

There are two hardware factors, the number of physical disks per node and the speed of the node interactions. Normal operation of the cluster is described above. To change the 10 gigabit/sec speed to 1 gigabit/sec requires changing Ethernet cards in the nodes. To change the number of disks per node from 12 to 6 requires disconnects physical drives and redistributing the test data across the HDFS.

Table 1.5.: Hardware Factors.

Factor	Values	Description
$DISK$	12, 6	Number of disks per node.
$NETWORK$	10, 1	Network bandwidth in gigabits/sec.

1.3 The Model Building Strategy

Model building is the critical task in the statistical analysis of the experimental data. Once the model is identified, validated, and fitted, results come readily, although as we shall see they are complex, and need trellis display to be able to understand them.

In this section we describe the approach to model building we take to meet certain challenges in the process. The principal challenge is that we want to model the effects of factors on O and L, but we measure O and $T = O + L$. We do not measure L directly. We need a strategy of model building that includes a deconvolution method to provide surrogate data for L to guide its modeling.

1.3.1 The Model Core: Dependence on M and V

The two factors M and V have a special role in our investigation. They are the statistical part. Their values in practice are chosen by the data analyst based on both statistical considerations and computational considerations. The statistical accuracy of D&R results depend on them, and so does the computational efficiency. The analyst has to balance the two considerations in practice. The other 6 factors in the experiment — BLK, IOB, MTC, REUSE, DISK, and NETWORK — involve the computational system, hardware factors and Hadoop configuration. We expect interactions of some of them with M and V. We have a good bit of broad knowledge about the dependence of O and L on M and V based on Hadoop computational considerations and the nature of the computational tasking of O and L.

As V increases, the size of the dataset increases proportionally. So for other factors held fixed, this certainly means monotone increases in O and L with increases in V. O is mostly the task of reading all of the data into memory. As shown in Section 1.2, the size is $D = 8NV$; the values it takes are shown in Table 1.2 are 128, 256, and 512 gigabytes. We would expect O to increase proportionally with V for other factors

held fixed, or not be too far off. Because D does not change with M, we expect a small effect of M for other factors held fixed.

As M increases with other factors held fixed, we expect that L will first decrease and then increase. This is due to two expected behaviors due to the design of Hadoop. As M increases from the the smallest values, ongoing Map computations at any given moment are asking for more memory, which can cause bottlenecks; this tends of make L increase with as M increases. But as M decreases from the largest values, the number of subsets R increases, the number of Map tasks goes up, and task management costs increase. this tends of make L as M decreases. These two phenomena act against one another, effects can be extreme for both large and small M. So we expect L to first decrease and then increase with M.

1.3.2 Experimental Values of O and T as Functions of M and V

We denote the experimental values of M and V for a specific one of the 2^6 combinations of the other factors by

$$M_i, i = 1, \ldots, 5 \text{ and } V_j, j = 1, \ldots, 3.$$

Their logs base 2 are, respectively, m_i and v_j. Letting k index the replicates, we denote the execution time response variables by

$$T_{ijk}\ L_{ijk} \text{ and } O_{ijk},\ i = 1, \ldots 9,\ j = 1, \ldots, 3,\ k = 1, \ldots, 3.$$

The log base 2 of these variables are, respectively, t_{ijk}, ℓ_{ijk} and o_{ijk}. Note we do not show dependence on the values of the other 6 factors for reasons to be discussed shortly. For each combination of the 6 factors and for each measurement type, T or O, there are $5 \times 3 \times 3 = 45$ measurements.

Each of the coming 4 plots has the following:

- 64 panels, one for each combination of the 6 Hadoop-hardware factors
- 4 pages, 16 panels per page

- on each panel t, o is plotted against m or v
- when the plot is against m, values of v are shown by plotting character, and a loess curve is fitted to t or o on m
- when the plot is against v, values of m are shown by plotting character, and connecting the the means of 3 log execution time replicates with lines

Figure 1.1 plots o against m, with different values of v superposed, conditioning on Hadoop-hardware factors. The strip labels of the panels describe the level of the Hadoop-hardware factors, and the factors are varying from fastest to slowest in the following order: IOB, NETWORK, MTC, DISK, BLK, REUSE. Pairs of panels when one factor is varying while others are held constant are separated vertically, horizontally, or by page, to make the comparison more effective. In most of panels, the loess curve of o on m is close to a flat line, and o does not seem to be varying much with the increase of m, although there are some panels where a slight increasing pattern is present for larger value of v. The relationship between o and m appears to be linear and the dependence is generally quite weak. On the other hand, as suggested by the clear separation of points and loess curves of different colors, o varies significantly with v, more specifically, it is monotone increasing with v in all panels. As we move across panels, the effects of Hadoop-hardware factors on o and their interactions with m come into play, and the values of o, as well as its relationship with m, vary from panel to panel to different degrees, depending on which factor(s) are varied.

Link to figure

Figure 1.1.: o is plotted against m superpose v, conditioning on Hadoop-hardware factors.

Figure 1.2 plots o against v, with different values of m superposed, conditioning on Hadoop-hardware factors. The relationship between o and v becomes more clear: o is increasing with v, and the increase is not always linear as the amount of increase in o from $v = 4$ to $v = 5$ can differ from the increase from $v = 5$ to $v = 6$ depending on the combination of Hadoop-hardware factors. Particularly, the increase in o with a unit increase of v is not exactly 1, meaning it does not take exactly twice as much time when the data size is doubled. Also there is a fair bit of overlapping for different values of m in the figure, and this is because of the weak dependence of o on m given the value of v we have already identified from the above figure.

Link to figure

Figure 1.2.: o is plotted against v superpose m, conditioning on Hadoop-hardware factors.

Figure 1.3 plots t against m, with different values of v superposed, conditioning on Hadoop-hardware factors. With the aid of the loess curves, it is easy to see that there is a non-linear relationship between t and m. And besides the difference in the values of t, the shape of the non-linear relationship between t and m also varies with the value of v, for example, the minima is achieved at a different value of m for each value of v. However, the shape of the curves seems to be generally preserved across panels for each value of v, despite the increase in the values of t with v, this will be further investigated with other displays.

Link to figure

Figure 1.3.: t is plotted against m superpose v, conditioning on Hadoop-hardware factors.

Figure 1.4 plots t against v, with different values of m superposed, conditioning on Hadoop-hardware factors. Similar to o, t is also increasing with v, and the increase is generally not linear, with the amount of increase in t from $v = 4$ to $v = 5$ being no larger than the increase from $v = 5$ to $v = 6$.

Link to figure

Figure 1.4.: t is plotted against v superpose m, conditioning on Hadoop-hardware factors.

1.3.3 Interactions and A General Model Specification to Account for Them

One property of the dependence of o and t on m and v is that there is more complexity in the t patterns than the o. The o patterns are either constant or increasing with m, and increasing with v. The t patterns decrease and then increase with v or m; because $T = O + L$, this suggests o decreases and then increase with v or m.

Another property of the dependence of o and t on m and v that is apparent in the plots is there is an interaction between m and v, and that the nature of the interaction changes quite substantially with the 2^6 combinations of the systems and hardware factors. This means there are pervasive interactions among all factors. It is important that we preserve them as much as possible, allowing them to enter in the modeling. So the modeling will be liberal, allowing for substantial interactions, then characterizing the statistical accuracy. The chief reason, as emphasized earlier, is that little is known about interactions.

We specify a general form for the dependence of o and ℓ on m and v, and then fit it independently for each of the 2^6 combinations of the systems and hardware factors. At the end, after the fitting, we will study the interactions, judging them in light of

both a characterization of the statistical variability, and qualitative knowledge about factors of the experiment. However, the model form is seen as tentative which, while quite general, needs validation with model building tools. They are based partly on the patterns seen in the plots of t and o shown earlier.

The tentative general model equations are

$$O_{ijk} = 2^{g_o(m_{ij},v_{ij})+\epsilon_{o;ijk}} \tag{1.1}$$

$$T_{ijk} = \{2^{g_o(m_{ij},v_{ij})} + 2^{g_\ell(m_{ij},v_{ij})}\}2^{\epsilon_{t;ijk}}. \tag{1.2}$$

g_o and g_ℓ are bivariate functions. $\epsilon_{o;ijk}$ and $\epsilon_{t;ijk}$ are jointly independent normal with mean 0, and variances σ_1^2 and σ_2^2 respectively. Note that g_o and g_ℓ are linear in the unknown parameters, but the overall form of the model is nonlinear in those parameters.

1.3.4 Discussion of Assumptions

In Equation 1.2, the term $2^{(g_\ell(m_{ij},v_{ij}))}$ is in effect a latent variable. However, the parameters are identifiable because of the assumption that the dependence of O on the factors is the same in both equations. We will verify this empirically, but is very much aided by our definition of O. This will be discussed in more detail below.

We have 3 replicates for each of the 1920 combinations of the factors, including the measurement type factor, so altogether there are 5760 runs, 2880 yielding O measurements, and 2880 yielding T measurements. In Section 1.2 we have identified two sources of error variability: the number of iterations of the logistic regression, and system processes that use a variable amount of capacity on the cluster. They are the basis of the error terms $\epsilon_{o;ijk}$ and $\epsilon_{t;ijk}$ in the model. We expect both to be small in magnitude compared with the effects of the factors.

1.3.5 Deconvolution, A Categorical Model, and Model Building

The lack of direct of observations of L interferes with the model building. So we take the following approach. The first model is categorical. The values of m and v are taken to be categorical. g_o and g_ℓ each have main effects and full interaction. The model is very general, allowing for a very wide range of dependencies of L and O on m and v. Now this is surely an over-specified model allowing far more than what we would expect and what is seen in the above plots, which suggest substantial smoothness in g_o and g_ℓ. However, we fit the categorical model, and then use the fitted values of o and ℓ in the model building as if they are the data, to find smooth g_o and g_ℓ that fit these *cat-fit data*. So this approach provides is a deconvolution method that yields the cat-fit values of ℓ for model building.

It is vital to emphasize that a resulting smooth modeling is then subjected to diagnostic checking, not just by how well it fitted the cat-fit data, but on its own, studying its fit to the data based on study of residuals corresponding to the error terms $\epsilon_{o;ijk}$ and $\epsilon_{t;ijk}$ in the model. Given that the data for each of the 2^6 combinations have 45 observations, we might question whether there is enough information to support reliable results. As we will see, the residuals are quite small in magnitude compared with the magnitude of most of the effects of the factors of the model. Uncertainty of whether observed effects are valid occurs only for those quite small in magnitude; we will use the bootstrap to help just these cases.

1.4 Identifying A Tentative Polynomial Model From the Categorical

We fitted the categorical model to the data for each of the 2^6 combinations of the systems and hardware factors, and then carried out model diagnostics to detect lack of fit if it exists by plotting residuals, and check for normally distributed errors, $\epsilon_{o;ijk}$ and $\epsilon_{t;ijk}$. In the interest of space we do not report details of this, just conclusions, because the we describe model checking details for the model arising from this categorical fitting which is the model to be used to characterize the effects of the factors on o

and ℓ. This is the model checking that determines the adequacy of the assumptions for the model used to make inferences about the dependence of o and ℓ on the factors.

We found no lack of fit of the categorical model that was more than very minor. This is not surprising because the number of observations for each combination is 90, and the number of parameters to fit the data is 30. We checked the assumption of normality of the errors, $\epsilon_{o;ijk}$ and $\epsilon_{t;ijk}$, by normal quantile plots of the residuals, one per combination. We found the empirical distributed to be well approximated by the normal, a very welcome finding.

Figure 1.5 plots the o cat-fit values against m. The general layout is like that of the plots in Section 1.3 with one panel for each of the 64 combinations of the Hadoop-hardware factors. Each panel has the values of v encoded by color as shown in the key. The change in o as a function of m is clearly small compared with that of v, as expected. The dependence on m does show, in some cases consistent small changes that also interact with the value of v; that is, there is an interaction of m and v.

Link to figure

Figure 1.5.: o cat-fits are plotted against m.

Figure 1.6 plots the o cat-fit values against v with values of m in each panel encoded as shown in the key. The patterns are very close to linear, but show in many cases a slight convex upward pattern, indicating that going from $v = 4$ to $v = 5$ has less increase than from $v = 5$ to $v = 6$.

Link to figure

Figure 1.6.: o cat-fit values are plotted against v.

To describe the dependence seen in the plots, we take as as a tentative model for the o dependence on m and v to be

$$g_o(m, v) = \mu_o + \alpha_{o1} \cdot v + \alpha_{o2} \cdot v^2 + \beta_{o1} \cdot m.$$

This is part of the overall model for O and L in Equations 1.1 and 1.2. In specifying $g_o(m, v)$ we provide for all observed patterns across the 2^6 combinations, which means some specifications, such as convexity in v, are needed in some cases but not all. We can of course prune unneeded model features for some the combinations.

Figure 1.7 plots the ℓ cat-fit values against m in the same manner as the o cat-fit values against m above.

Link to figure

Figure 1.7.: ℓ cat-fit values are plotted against m.

There is a quadratic effect for $v = 4$ and $v = 5$ but with a suggestion of a cubic for $v = 6$, because the slope drops somewhat for the last two values of v.

Figure 1.8 plots the ℓ cat-fit values against v in the same manner as the o cat-fit values against v above.

Link to figure

Figure 1.8.: ℓ cat-fit value are are plotted against v.

There is a pronounced quadratic effect, but with a suggestion that the effect changes with m.

To describe the dependence seen in the plots, we take as as a tentative model for the ℓ dependence on m and v to be

$$g_\ell(m, v) = \mu_\ell + \alpha_{\ell 1} \cdot v + \alpha_{\ell 2} \cdot v^2 + \beta_{\ell 1} \cdot m + \beta_{\ell 2} \cdot m^2 + \gamma_\ell \cdot v \cdot m$$

These polynomial models, $g_\ell(m, v)$ and $g_o(m, v)$, are part of the tentative overall model for O and L in Equations 1.1 and 1.2. So the tentative model for O and L is

$$O_{ijk} = 2^{g_o(m_{ij},v_{ij})+\epsilon_{o;ijk}}$$
$$T_{ijk} = \{2^{g_o(m_{ij},v_{ij})} + 2^{g_\ell(m_{ij},v_{ij})}\}2^{\epsilon_{t;ijk}}.$$

We have

$$g_o(m, v) = \mu_o + \alpha_{o1} \cdot v + \alpha_{o2} \cdot v^2 + \beta_{o1} \cdot m.$$

and

$$g_\ell(m, v) = \mu_\ell + \alpha_{\ell 1} \cdot v + \alpha_{\ell 2} \cdot v^2 + \beta_{\ell 1} \cdot m + \beta_{\ell 2} \cdot m^2 + \gamma_\ell \cdot v \cdot m.$$

In addition the two sets of mutually independent error terms, $\epsilon_{o;ijk}$ and $\epsilon_{t;ijk}$, are each taken to be i.i.d normal with variances σ_o^2 and σ_t^2, respectively. This tentative specification is based on the residuals for the categorical model being well approximated by the normal. This tentative model applies to each of the 64 combinations of the Hadoop-hardware factors, has different parameter values across the combinations, and has independent error terms across the combinations. This is done because we need to provide empirically as much latitude as possible to see interactions. So each combination is fitted independently, maximum likelihood.

1.5 Polynomial Model: Diagnostics and Statistical Variability

In this section we describe diagnostics to check the assumptions made in the tentative polynomial model, and provide information about the statistical variability of the estimates of the response surface.

1.5.1 Diagnostics

To check for lack of fit of the polynomial model, we compare o and ℓ poly-fits, the fitted values for o and ℓ from the polynomial model, with the o and ℓ cat-fits, the fitted values for o and ℓ from the categorical model, emphasizing that the cat-fits

serve as surrogates for the data. The poly-cat residuals for the polynomial model are the cat-fits minus the poly-fits. For these residuals, instead of the log base 2 scale we have been using, we use log base e, to provide much better quantitative interpretation. Natural logs between about ± 0.25 can be interpreted as a fractional change because $\log(1 + r) \approx r$. The poly-cat residuals are well within ± 0.25.

We also need to check the assumption that the error terms $\epsilon_{o;ijk}$ and $\epsilon_{t;ijk}$ are well approximated by the normal. We do this by studying the two sets of error residuals: values of o minus the o poly-fits, and values of t minus the o and ℓ poly-fits.

Figure 1.9 plots o poly-cat residuals against m for each of the 2^6 combinations of the Hadoop-hardware factors. The layout of the panels of the display is the same as that for other plots against m for the combinations. Variability is mostly contained within ± 0.05 which is very small compared with effects of the factors. There are a few values for $v = 6$ and $m = 16$ whose departures are larger, but these too remain small, not much larger than the general variability. We have observed in Section 1.4 that the categorical fits for for $v = 6$ show a departure from quadratic for $m = 16$. We could account for this by adding a cubic term to the model, but because of the relative small size of the departures, and our need for parsimony in the modeling, we do not alter the model.

Link to figure

Figure 1.9.: o poly-cat residuals for the polynomial model are plotted against m.

Figure 1.10 is the plot of ℓ poly-cat residuals against m. General variability is somewhat greater than for o. Again we see outliers, caused by the same issue discussed above, but occurring now at $m = 14$. As for o, these larger departures are not enough larger than the general variability, and are also small compared with magnitudes of major effects, so we take no action.

Link to figure

Figure 1.10.: ℓ poly-cat residuals for the polynomial model are plotted against m.

Figure 1.11 is a normal quantile plot of the o error residuals. The line on the plot is drawn through the lower and upper quartiles. The magnitudes are overall very small. The approximation is quite good, certainly justifying lease-squares fitting, and sufficient for using the parametric bootstrap with a normal assumption to characterize variability.

Link to figure

Figure 1.11.: Normal quantile plot of o error residuals for the polynomial model.

Figure 1.12 is a normal quantile plot of the t error residuals. The line on the plot is drawn through the lower and upper quartiles. The magnitudes are overall very small. Departures from normality are somewhat greater than for the o error residuals, but are not great enough threaten use of least-squares or ensuing inferences based on the parametric bootstrap.

Link to figure

Figure 1.12.: Normal quantile plot of t error residuals for the polynomial model,

The polynomial model has survived diagnostic testing. It has a few blemishes that make it wrong, but not by an amount that will appreciably affect conclusions drawn

from it. The error terms are quite small. Almost all O errors are contained within $\pm 10\%$ of the O measurements, and the T errors within $\pm 5\%$ of the T measurements.

1.5.2 Statistical Variability of Response Surface Estimates

For each of the 2^6 combinations of the Hadoop-hardware factors, we used the bootstrap to characterize the statistical variability of the o and ℓ poly-fits: o and ℓ fitted values of the polynomial model at the 45 values of m and v of the experiment. We study the response surface because it carries the informative information of the experiment, magnitudes of execution times and how they change with the factors.

There were 1000 draws for each combination. Each draw consists of 45 values of o and 45 values of ℓ. The process begins with 45 i.i.d. draws from each of the fitted distributions for the error terms $\epsilon_{o;ijk}$ and $\epsilon_{t;ijk}$. The fitted distributions are normal with mean 0 and a variance equal to the sum of squares of the error residuals of each, divided by 45. The 45 o draws are then the sum of the o error term draws plus the o poly-fits at the 45 values of m and v. Similarly, the 45 t draws are the 45 t error term draws plus the sum of the o poly-fits and the ℓ poly-fits at the 45 values of m and v. Then the polynomial model was fitted to each bootstrap sample of t and o values, and the fit evaluated the 45 values of m and v, providing one bootstrap sample for the 45 o poly-fits and the 45 t poly-fits. To study the variability we subtract the 45 o poly-fits and the 45 t poly-fits from the corresponding 45 values of the bootstrap sample to get the bootstrap deviations.

Figure 1.13 plots, for each of the 2^6 combinations of the Hadoop-hardware factors, the 0.025 and 0.975 quantiles of the 45 o poly-fit bootstrap deviations. The layout of the panels of the display is the same as that for previous plots against m for the combinations. For these bootstrap deviations, as they are well within ± 0.25, we again use log base e for better quantitative interpretation. The bootstrap variability of the o poly-fits is very small. Almost all of the 0.025 and 0.975 quantiles of bootstrap deviations are contained within $\pm 5\%$ of the original o poly-fits.

Link to figure

Figure 1.13.: 0.025 and 0.975 quantiles of bootstrap deviations for o poly-fits.

Figure 1.14 plots, for each of the 2^6 combinations of the Hadoop-hardware factors, the 0.025 and 0.975 quantiles of the 45 ℓ poly-fit bootstrap deviations. General variability is great than that for o, but not much greater. The majority of the 0.025 and 0.975 quantiles of the bootstrap deviations are contained within $\pm 10\%$ of the original ℓ poly-fits.

Link to figure

Figure 1.14.: 0.025 and 0.975 quantiles of bootstrap deviations for ℓ poly-fits.

1.6 Effects and Interactions of Hadoop-hardware Factors

In this section we present our findings of the impact of the Hadoop-hardware factors on the O computation and the L computation, including the main effects of these factors and the interactions among these factors we have discovered in the experiments. The impact of a particular Hadoop-hardware factor on the response surface of o and ℓ, is quantified by the improvement, or in this case the decrease, in the fitted values of o or ℓ from the polynomial model when varying this factor from one level to the other level, while holding all other factors constant. We also rely on bootstrap to characterize the statistical variability of the impact of the factors on corresponding estimates of response surface.

We relied very heavily on trellis displays to effectively study the effects and interactions of the Hadoop-hardware factors, however, in the interest of space, we will use

the factor BLK as a demonstration. Other details are not reported here and instead a summary of our findings is given.

1.6.1 Effects of BLK and Interactions with Other Factors

BLK is a HDFS configuration parameter that affects the storage of the dataset on the HDFS and the scheduling of Map tasks. Given all other factors remained the same, when BLK becomes larger, each Map task reads and computes with a larger amount of data while the number Map tasks becomes smaller accordingly. Thus, BLK directly impacts the I/O of the HDFS and thus we would expect it to have an effect on o; and BLK also affects the parallelization of Map tasks, thus it could also impact ℓ as well.

Main Effects of BLK on o

In this experiment, we have doubled BLK from 128 MB to 256 MB. Figure 1.15 plots the impact of BLK on o poly-fits against m for each of the remaining 2^5 combinations of the other Hadoop-hardware factors. The impact of BLK is represented as the change of o poly-fits when BLK is varied from 256 MB to 128 MB, while all other factors are held constant. A positive value on the vertical scale says the o time is larger when $BLK = 128$ MB than that when $BLK = 256$ MB, i.e., increasing BLK improves the o time. The layout of the panels of the display is similar to that for previous plots against m for the combinations, however, due to the collapse of the two levels of BLK into the representation of the difference, there are only 32 panels across two pages.

Link to figure

Figure 1.15.: Impact of BLK on o poly-fits is plotted against m.

To study the variability of the impact, we again used bootstrap and computed the bootstrap deviations for the impact of BLK on o poly-fits, i.e., the difference between the impact computed from each bootstrap sample and the impact computed from the original fit of the polynomial model. Figure 1.16 plots, for each of the remaining 2^5 combinations of the other Hadoop-hardware factors, the 0.025 and 0.975 quantiles of the bootstrap deviations of the impact of BLK on o poly-fits. The layout of the panels of the display is the same as that for the previous plot against m for the combinations. The bootstrap variability is very small, most of 0.025 and 0.975 quantiles of bootstrap deviations are contained within around $\pm 5\%$ of the impact computed from the original fit.

Link to figure

Figure 1.16.: 0.025 and 0.975 quantiles of bootstrap deviations for impact of BLK on o poly-fits.

The main effects of BLK on o are obvious, as shown in the majority of the panels in Figure 1.15, the connected points lie above zero, meaning that o is larger when BLK is 128 MB than that when BLK is 256 MB, i.e., when BLK is increased, the O computations are generally faster. And the amount of improvement, when there is any, can be up to 40%. There are a few panels where some of the points lie slightly below 0, however, the deterioration is usually less than 10%, which is comparable to the bootstrap variability.

The positive effects on o time by increasing BLK agree with our understanding of how BLK functions. As BLK is doubled from 128MB to 256MB while all other factors are held constant, the amount of data/subsets in each block/Map task is doubled and the number of blocks/Map tasks is halved accordingly.

First of all, with fewer number of Map tasks, there are fewer task overheads, which are are part of the O computation, so increasing BLK would improve the o timing in this aspect.

Furthermore, with fewer number of blocks, there are fewer subsets crossing block boundaries, resulting in smaller amount of extra data reads due to subsets crossing block boundaries, thus could further improving o time. Notice that the amount of extra data reads also depends on the size of subsets, so the amount of improvement made by increasing BLK in this aspect is likely affected by factors that determines the size of subsets, i.e., m and v in this experiment.

Lastly, BLK affects the level of parallelization, i.e., larger number of smaller tasks versus smaller number of larger tasks, and the effects on timing could go either way, while the combined effect of all the above aspects turned out to be positive in this experiment when BLK is doubled.

Interaction between BLK and m on o

The interaction between BLK and m is reflected on the slopes on the lines connecting the points in each panel of Figure 1.15. In some of the panels, the lines have a slope very close to zero, suggesting that there is no or very weak interaction between BLK and m in the O computation, and the amount of improvement in o via increasing BLK is about the same across different values of m; however, under certain combinations of other Hadoop-hardware factors, especially in the top panels on the second page of Figure 1.15, there is a non-zero slope of the lines. In fact, the lines tend to go up in the top panels ($DISK = 12$) on both pages, suggesting that the amount of improvement via increasing BLK is larger when the subsets are larger; and they tend to stay flat or go down very slightly in the bottom panels ($DISK = 6$). So there could be a weak 2-way interaction between BLK and m, and a 3-way interactions between BLK, m and $DISK$.

The interaction between BLK and m can be attributed to their joint impact on the extra data reads caused by subsets crossing block boundaries. As m increases while holding other factors constant, the size of each subset increases, and thus the amount of extra data reads due to subsets crossing block boundaries would increase. So the amount of improvement through increasing BLK by having few subsets crossing block boundaries should be larger when m is larger than that when m is smaller.

As for the 3-way interaction among BLK, m, and $DISK$, we suspect that when $DISK$=6, there is too much congestion in disk reads and this has become the bottleneck for the O computation, and thus suppresses the effects of BLK or m.

Interaction between BLK and v on o

The interaction between BLK and v is reflected on the vertical separation of the lines in each panel of Figure 1.15. In most of the panels in the top rows ($DISK = 12$) on both pages, the lines shift up as v increases, and the amount of improvement through increasing BLK is larger when v is larger. This again suggests that the improvement is larger when the subsets are larger, and the cause of interaction between BLK and v is similar to that between BLK and m, i.e., extra data reads due to subsets crossing block boundaries.

In most of the panels in the bottom rows ($DISK = 6$), on the other hand, there is little separation among the lines for different values of v. This can also be explained by a 3-way interaction among BLK, v, and $DISK$ similar to the one above among BLK, m, and $DISK$.

Interactions between BLK and Other Hadoop-hardware Factors on o

The interactions between BLK and the other Hadoop-hardware factors can be visually assessed by comparing pairs of panels in Figure 1.15, either horizontally, vertically, or across pages.

For example, by comparing every horizontally adjacent pair of panels in the same row, we can assess the interaction between BLK and IOB. Except for the top left pair of panels on the first page of Figure 1.15, there does not seem to be much visual difference between the pairs of panels when varying IOB, thus suggesting no interaction between BLK and IOB. In fact, although not shown here, we have found that IOB does not seem to have any main effects on either O time or L time. Similarly, there does not appears to be any strong indication of interactions between BLK and $NETWORK$ or MTC either.

However, when comparing vertically adjacent pairs of panels in Figure 1.15, a noticeable difference emerges, suggesting an interaction between BLK and $DISK$. This is hardly surprising, as we have previously identified possible 3-way interactions among these two factors along with m or v.

Furthermore, an even more obvious interaction between BLK and $REUSE$ can be identified by comparing panels across pages. While in general, increasing BLK from 128 MB to 256 MB improves the O time, $REUSE = True$ clearly decreases the gain compared with $REUSE = False$. We shouldn't be surprised to see their interaction either. Recall that one reason increasing BLK improves the O time is the reduced task overhead by having less number of Map tasks, this reduction is partially offset by turning on $REUSE$, which already reduces the overhead of each individual task via reusing JVM.

Effects of BLK on ℓ

Similarly, the main effects of BLK on the ℓ time and its interactions with other factors can also be assessed via a pair of similar displays as shown in Figure 1.17 and Figure 1.18. In summary, BLK does not have as strong (positive or negative) impact on ℓ time as it does on o time, which in in accordance with our intuition as BLK affects the I/O of HDFS more directly than it does with the actual computations; and the statistical variability obtained from bootstrap is also larger for the impact of

BLK on ℓ time, a $\pm 10\%$ range versus a $\pm 5\%$, similar to the difference of bootstrap variability between o poly-fits and ℓ poly-fits in Section 1.5.

Link to figure

Figure 1.17.: Impact of BLK on ℓ poly-fits is plotted against m.

Link to figure

Figure 1.18.: 0.025 and 0.975 quantiles of bootstrap deviations for impact of BLK on ℓ poly-fits.

1.6.2 Summary of Effects and Interactions of Hadoop-hardware Factors

We performed similar analysis and assessment to study the effects of other Hadoop-hardware factors on o time and ℓ time, as well as their interactions. We will skip the details but only report a summary of our findings.

For the o time, BLK, $REUSE$, $NETWORK$, and $DISK$ appear to have a strong impact while the other factors do not seem to affect the timing much; and due to the presence of interactions among these factors along with m and v, the improvement that can be achieved varies with different combinations of factors, with the largest improve of around 70%. The superior setting of levels of these factors are $BLK = 256$ MB, $REUSE = True$, $NETWORK = 10$ gigabit/sec, and $DISK = 12$.

For the ℓ time, on the other hand, fewer factors appears to have a strong impact, namely $REUSE$, MTC, and $DISK$, and the impact is also generally smaller compared with that on the o time, with up to a 30% improvement. Recall that the improvement is quantified by a percentage/ratio instead of absolute time because

of the use of log scale, the smaller improvement could be attributed to the relative larger base value of the ℓ time compared with the o time. As for the superior levels of factors, while $REUSE = True$ and $MTC = 16$ improves the ℓ time compared with the other levels of these factors, setting $DISK = 12$ actually deteriorates the ℓ time compared with $DISK = 6$, which is the superior level of $DISK$ for the o time. We do not have a convincing explanation to offer for this phenomenon, but simply to note that the improvement in o time surpasses the loss in ℓ time, thus resulting in an improvement in the combined t time.

2. BITCOIN DATA ANALYSIS AND MODELING

2.1 Introduction

Bitcoin is a de-centralized digital currency system that was implemented and established in January 2009. Unlike fiat money such as U.S. Dollars, there is no central authority in the Bitcoin system to issue new money, or validate the transfer of money; both of these tasks are accomplished through the joint work of participants in the Bitcoin network. In the past two years, especially after the establishment of various Bitcoin exchange sites, where Bitcoins can be exchanged from or to local currencies such as U.S. Dollars and Euros, Bitcoin has become very popular, mostly due to its ease of use and embedded anonymity in the system.

The ease of use of Bitcoin is straightforward. The Bitcoin system is an online system and runs on various operating systems and devices, and it is also free to use. To join the Bitcoin network, one could either download one of many implementations of the Bitcoin client program, or simply make use of one of many online services. Making payments or accepting payments with Bitcoins are also free and as easy as, if not easier than, any other existing online payment system.

The anonymity of the Bitcoin system, on the other hand, is rather debatable and has drawn much attention in its user community as well as the research community. We admit that a certain level of anonymity exists in the Bitcoin system, but it might not be as invulnerable as one would hope. For one thing, the entire history of Bitcoin transactions is publicly available, which provides an opportunity for passive analysis of Bitcoin usage such as ours.

We downloaded the Bitcoin transaction data for the first time in May 2013 and we have been continuing the collection and analysis of the transactions. On May 3rd 2014, we concluded our data collection and obtained the complete Bitcoin transaction

history data up to that date. We present here a study of the general statistical properties of the usage of Bitcoin transactions and the usage of Bitcoin addresses. We have also built profiles for a few groups of popular addresses among which the addresses share similar behavior. Furthermore, we provide a passive analysis of the anonymity of Bitcoin system by proposing a classification model to identify payment and change in majority of the Bitcoin transactions.

2.2 Background

The design of the Bitcoin system was introduced in 2008 by Satoshi Nakamoto [22], whose true identity remains unknown and it is not even known whether the name is real or a pseudonym, or whether the name represents one person or a group of people. On January 3rd 2009, the very first Bitcoins were minted, quickly followed by the release of the first Bitcoin client program Bitcoin v0.1 [23], which mark the establishment of the current Bitcoin network. The Bitcoin system is designed to be de-centralized and the system is powered by its users with no central authority, such as a bank; and it also provides a level of anonymity that allows its users to transfer Bitcoins without giving away personal financial information.

In this section, we describe three of the most fundamental and important aspects of how the Bitcoin system works: how are Bitcoins transferred, how are Bitcoins minted, and how are the transfer verified. Chronologically, Bitcoins need to be minted before they are transferred and verified, however, for simplicity, we start with the transfer of Bitcoins by introducing Bitcoin transactions and Bitcoin addresses, followed by the minting of Bitcoins and the verification of Bitcoin transactions. We also discuss the implication of these design aspects on the anonymity of the Bitcoin system.

2.2.1 Bitcoin Transactions and Bitcoin Addresses

The transfer of Bitcoins, or making a payment in the Bitcoin system, between the sending entity, or the sender, and the receiving entity, or the receiver, are carried out

via Bitcoin transactions. We will refer to these transactions as payment transactions. The sender and receiver in a payment transaction are identified by the corresponding Bitcoin addresses. The Bitcoin address(es) that identifies the sender and serves as the the source of the transaction are referred to as sending address(es); and the Bitcoin address(es) that identifies the receiver and serves as the destination of the transaction are referred to as the receiving address(es). In these payment transactions, the ownership of some amount of Bitcoins is transferred from the sending address(es) to the receiving address(es).

A Bitcoin address, also known as a public key, is a string of 27–34 alphanumeric characters, starting with the number 1 or 3. An example of a Bitcoin address is 1KfqhLeiBSaPteVK4RUKgCxBhysr3qLdm1. To obtain a Bitcoin address, a user can make use one of many Bitcoin client programs or online wallet services, and generate a pair of public key and private key. The public key is the Bitcoin address, and it could be made public if the user would like to this Bitcoin address to accept payments from others. The matching private key proves the ownership of the Bitcoin address, and allows the user to spend the Bitcoins received in this Bitcoin address. If somehow the private key is lost or stolen, the user will no longer be able to spend the Bitcoins in this Bitcoin address, therefore the private keys are usually kept secret.

Once an address is generated by a user, the user can use it to make transactions, i.e., receiving and sending Bitcoins. It is often helpful to draw the analogy between a Bitcoin address and an email address. While an email is sent from the sender's email address to the receiver's email address(es), Bitcoins are transferred from sending address(es) to receiving address(es); just like one person can possess many email addresses, a user can generate and own as many Bitcoin addresses as wanted. However, unlike a new email address can be used to send emails immediately after generation, a newly generated Bitcoin address can not be used to send Bitcoins unless it has received some amount of Bitcoins from other address(es). So the life of a Bitcoin address always begins with a receiving transaction, where it receives some amount of Bitcoin from other address(es). The same Bitcoin address can be used repeatedly

to receive in multiple transactions, and consequently it can be used to send multiple times. However, it is often suggested that one should generate a new address for every transaction in order to achieve better privacy.

In a payment transaction, a single sending address could be used to provide funding, or many sending addresses can be pooled together. Each sending address, accompanied with its digital signature and the reference to a previous receiving transaction, in which this address has received some amount of Bitcoins, provides (partially) the funding for the current transaction. The digital signature is produced by the private key and proves the sender's ownership of the sending address; and the reference to the previous receiving transaction identifies the source of the funding. Notice that since there is no central authority, such as a bank, to keep track of the account balance of any sending address, the address alone is not enough to identify the source of the funding. However, when combined with the reference to a previous receiving transaction, the source of funding is uniquely identified, because what is being spent is exactly what has been received previously. Also when there are multiple sending addresses in a transaction, the addresses might not be unique. This happens when the same sending address have received Bitcoins in multiple transactions previously occurred, and these (address, signature, previous receiving transaction) tuples could be combined as the source in a single transaction.

As for receiving, Bitcoin transactions also support multiple receivers, i.e., there could be one or many receiving addresses in the same transaction. In fact, as we shall see in Section 2.4, the majority of transactions have more than one receiving addresses. In a single transaction, the receiving addresses might not be unique either, and they can even be the same as the sending addresses, meaning the sender could be sending Bitcoins to himself/herself.

With the help of Bitcoin client programs and online wallet services, it is fairly easy to make a Bitcoin transaction. The sender needs to know the address(es), as string(s) of characters, from the receiver(s), and specify it as the receiving address(es). It is also possible for the sender to select the source of funding, i.e., the tuple(s) of (send-

ing address, signature, previous receiving transaction), but usually the program will automate the selection and complete the transaction for the sender. The transaction is then broadcasted to the Bitcoin network by the sender's program, later relayed by other peers in the network, and eventually received by the receiver(s).

Obviously, a payment transaction needs to be verified before the receiver(s) are certain that there are sufficient funds, and the sender is not double-spending them. However, the receiver(s) of the transaction usually do not make the effort to verify the transaction themselves. Instead, a particular group of entities in the Bitcoin network, referred to as the miners, would verify the transactions on behalf of the whole Bitcoin network. This process is called mining.

2.2.2 Mining and Verification of Transactions

The verification of payment transactions is carried out by the miners in the Bitcoin network through the mining process. Each of the miners verifies a certain number of payment transactions and shares the verification with the entire Bitcoin network. Miners also verify the verification produced by other miners so that no miner is able to trick the whole system. Besides verification of payment transactions, the mining process also issues/mints new Bitcoins into circulation and it is the only way of generating new Bitcoins.

During the mining process, each miner works independently and picks up the payment transactions occurred in the Bitcoin network during the past few minutes' interval (usually 10 minutes or so), and verifies those transactions by digging into the history of transactions to ensure sufficient funds and to prevent from double-spending. Besides the verification of transactions, which can be completed very quickly with moderate computation power, the miners also need to work on a much more computation-intensive task named proof-of-work [24]. While working independently, the first miner to finish the verification and the proof-of-work task will create a block of transactions, including those payment transactions that have just been

verified and a special generation transaction, and inform the whole Bitcoin network by broadcasting the block into the network. Upon receiving a newly generated block, other miners will give up the unfinished work, instead verify the block they have just received, and then start generating the next block by verifying the subsequent transactions occurred in the next few minutes' interval, as well as working on the next proof-of-work task. Because the miners verify the previous blocks produced by other miners, no miner would be able to smuggle invalid transactions in the blocks unless it can consistently out-computes all other miners, which is very unlikely given the large number of existing miners in the Bitcoin network.

All the blocks generated by the miners are indexed by a block height, an integer starting from 0 and incremented by 1 for each subsequent block. The very first block created in the Bitcoin network is referred to as the genesis block. Because the miners would verify the previous blocks when generating any new blocks, each subsequent block generated is said to have confirmed its previous blocks. This chronological chain of confirmed blocks, or more formally the blockchain, consists of all the verified Bitcoin transactions, and it is shared in the Bitcoin network to serve as the honest history of the Bitcoin transactions. In practice, the number of confirmations of a transaction, i.e., the number of subsequent blocks since the block that included this particular transaction had been generated, is often used by the receiver(s) of this transaction as a measure of confidence that this transaction is valid and the sender can not change the transaction anymore [22].

One might wonder why the miners are making the effort to verify transactions for the entire Bitcoin network. They are certainly not doing it for free. There are two incentives provided by the Bitcoin system: (1) the miner who generated each block will be rewarded a certain amount of Bitcoins, referred to as the block rewards; (2) in each Bitcoin transaction, the sender has an option to specify a transaction fee, and the fee is also collected by the miner who generated the block, to provide an incentive for the miners to give higher priority of verification to this transaction over others during the same time period.

When a miner successfully generated a block, the block rewards and combined transaction fees are sent to the receiving address(es) specified by the miner in a special transaction named the generation transaction. The generation transactions differ from payment transactions in that there are no sending addresses and the source is referred to as the Coinbase. Notice that the block rewards are newly minted Bitcoins and this is how new Bitcoins are issued in the Bitcoin system. The number of Bitcoins in the block rewards was 50 BTC at the inception of Bitcoin and is halved every 4 years or so. To prevent miners, especially those with high computation power, from generating blocks too fast and causing inflation in the currency, the difficulty of proof-of-work is automatically adjusted by the Bitcoin system to ensure a projected growth of Bitcoins in circulation.

With the increasing difficulty of proof-of-work, mining with low computation power can take a very long time and it becomes very unlikely for those miners to generate blocks. In order to take advantage of the embarrassingly parallel nature of proof-of-work and generate blocks in a reasonable amount of time, pooled mining has become very popular, where multiple individual miners contribute to the generation of a block, and then split the block rewards according their contributions [25].

2.2.3 Anonymity in the Bitcoin System

One advantage of the Bitcoin system over most of the other digital payment systems is its anonymity and many users adopt the Bitcoin system because of the anonymity. The anonymity of the Bitcoin system has also drawn a lot of attention from the research community, such as [26] [27] [28] and [29]. It is true that there is a certain level of anonymity imbedded in the Bitcoin system, mostly due to the non-centrality and the usage of addresses instead of registered account names, however, as we shall see, using Bitcoins might not be as anonymous as one would have hoped.

First of all, an entity, including both individual users and organizations, does not need to give away any personal information in order to acquire and use Bitcoin ad-

dresses through the Bitcoin client programs. Thus, although the complete history of Bitcoin transactions between addresses is publicly available, the entities behind these transactions and addresses are anonymous, unless they voluntarily give away identification information, e.g., when publicly asking for donations to certain addresses. However, if the entities choose to use one of many online wallet services instead of the client programs, they usually would have to go through a registration process and thus give away certain identification information, such as email addresses, to those services. Furthermore, when entities "cash out" Bitcoins they own into local currency, they would usually go through the Bitcoin exchange sites, where more identification information, such as bank information, would have to be exposed to these exchange sites.

Secondly, one could, or as generally suggested one should, generate a new address for each transaction, and consequently each entity could potentially own many different addresses and this leads to a many-to-one mapping from addresses to entities. Furthermore, whether or not two addresses belong the same entity is unknown to the public. However, due to the design of the Bitcoin system, certain grouping of addresses into entities is possible. As introduced in [22] and [26], and further studied in [30] [31] and [28], the sending addresses in the same transaction are owned by the same entity. For example, if address A_1 and A_2 are both sending addresses in a transaction T_1, then A_1 and A_2 are owned by the same entity; furthermore, if A_2 is used as sending address in another transaction T_2 with another sending address A_3, then A_1, A_2, and A_3 are all owned by the same entity. We will refer to these addresses as having co-sending relations, and the unions of addresses with co-sending relations form a conservative estimate of entities in the Bitcoin system. The co-sending relations based grouping of addresses into entities is conservative because it is entirely possible that the same entity could own multiple such unions of addresses.

Thirdly, another level of anonymity in the Bitcoin system is that the intention of a transaction is generally not explicitly specified and thus often unknown to others outside of the entities involved in the transaction. This is largely due to the existence

of change address in the Bitcoin transactions, which is ubiquitous in Bitcoin transactions because of an important aspect of the design of the Bitcoin system: Bitcoins can not be spent as a fraction. What this means is that, once an (sending address, signature, previous receiving transaction) tuple is used in a transaction, all the Bitcoins associated with this tuple are considered as spent, and this tuple can no longer be used to send in another transaction. This may sound bizarre at first, but recall that there is no central authority to keep track of the balance of addresses and to validate transactions, this spend-all-or-none design greatly simplifies the verification of transactions: the miner only needs to search the history for the previous transaction(s) referenced in this transaction, and check if there is sufficient fund and if there are other transactions referring to the same tuple to prevent from double-spending. Obviously, it is impractical to assume that, in a transaction, the amount in the (address, previous receiving transaction) tuple(s) happens to be exactly the same as the desired receiving amount, so there is usually a change in a transaction. And assuming that a rational user does not simply give up Bitcoins, the change needs to be sent back to an address of the sender's, and more importantly, this has to be done in the same transaction as if the change is just another receiving instance. Note that the users do not need to take care of the change manually because the Bitcoin client programs or the online wallet services have made this process automatic. Nonetheless, this has a significant impact on how a Bitcoin transaction would look like from an outsider's point of view. Because of the change, there are usually more than one receiving address in a Bitcoin transaction, including both the actual receivers' address(es), or the payment address(es), and the sender's change address. However, the transaction itself does not explicitly reveal which one of the receiving addresses is the change address, so the actual payment(s), or the intention of the transaction, is unknown to an outsider. In [27] and [28], researchers have demonstrated methods/heuristics to identify those change addresses automatically generated by Bitcoin client programs and online wallet services; and in this study, we also propose a model that effectively identifies

payment and change in majority of the transactions, revealing the true intentions of Bitcoin transaction hidden behind seemingly unrelated addresses.

Lastly, to further obfuscate the source and destination of a Bitcoin transaction and improve privacy, users can make use of one of many mixing services [32] when making transactions. A mixing service is a third-party service, and it mixes the transactions between multiple senders and multiple receivers by having the senders send Bitcoins to address(es) owned by the service, and then the service sends specified amount of Bitcoins, not necessarily the same Bitcoins it has received from senders, to the targeted receivers specified by the senders. In this way, the senders and receivers are not directly associated in a transaction, and thus privacy is improved. However, due to the centrality introduced by the mixing services and the security of these services themselves, mixing services have received limited popularity among Bitcoin users. Also, as suggested in [29], because of the limited user base, many of these mixing services are not able to effectively obfuscate the transactions, making it possible, sometimes easy, to track the real source and destination of transactions.

2.3 Initial Analysis: Data and Statistics

The Bitcoin transaction history data, namely the blockchain, is available to the whole Bitcoin network via the Bitcoin client program. Many of the online wallet services also provide access of the blockchain and allow queries for individual or batch transaction level information or address level information. However, in order to perform a comprehensive analysis such as ours, the blockchain needs to be downloaded and processed into suitable formats and structures. In this section, we describe the procedures of our data collection and initial processing, followed by summary statistics of the obtained blockchain. Further processing of the data into multiple databases of various structures are discussed in details in the following sections along with the corresponding analyses enabled and facilitated by each database.

2.3.1 Data Collection and Initial Processing

The first step of our data collection is to run the Bitcoin client program, Bitcoin Core version 0.9.1 [33], on our computers in order to sync to the to Bitcoin network and download the whole blockchain. During the sync process, the Bitcoin client program connects to the Bitcoin network, and starts to download the blocks from other peers in the network until it catches up with the up to date blockchain by finish downloading the most recent blocks. On May 3rd 2014, we stopped the sync process and concluded our data collection, giving us the raw blockchain from the very first block up to block 298,851 generated on May 3rd 2014. The raw blockchain data was in binary format of around 24 GB stored in a database system named LevelDB [34], and the transaction level data needed to be extracted and converted into text files for further processing.

Secondly, the extraction of transaction level data and the conversion from binary to text format were done via one of the APIs [35] supported by the Bitcoin client program. We extracted all the transactions in the blockchain and they were converted into a plain text file of around 29 GB. Each line of the text file corresponded to the information of a sending address or a receiving address in a transaction, and the following information was included for each transaction:

1. The transaction ID, which uniquely identifies each transaction.
2. The receiving address(es).
3. The index of each receiving address in this transaction, i.e., the order in which this receiving address is specified in this transaction, starting from 0 and incremented by 1.
4. The amount of Bitcoins, in the unit of BTC, received by each receiving address.
5. The sending address(es).

6. The transaction ID of each referenced receiving transaction for each sending address.

7. The index of each receiving address in its referenced receiving transaction.

8. The height of the block that included this transaction.

9. The time that the block was generated.

Thirdly, in order to incorporate the co-sending relations based grouping of addresses with the transaction level data, we used a modified version [36] of Ivan Brugere's Bitcoin-Transaction-Network-Extraction tools [37] and re-organized the transaction level data. We have made numerous improvements through our modification, most importantly, our modifications recovered various important information that was missing in the data produced by the unmodified tools, including the sending/receiving addresses and their indices in corresponding transactions, which are essential for tracking referenced transactions, and we also corrected the handling of a few problematic addresses, where the unmodified tools incorrectly treated them as the same address, etc. During this step of processing, the following information was added to the transaction level data in addition to the above list for each transaction:

10. The entity ID of each receiving address, while the entities are formed based on co-sending relations.

11. The entity ID of the sending addresses.

Additionally, besides the raw blockchain, we also downloaded external data for the daily exchange rate from bitcoin (abbreviated as BTC) to United States Dollar (USD) from the Bitcoin Block Chain Explorer website [38]. The data was in CSV format of 50 KB.

Finally, due to the relatively large size of the data, it calls for a distributed computation system for the comprehensive analysis. We made heavy use of the Tessera D&R

Computational Environment [8], particularly the R and Hadoop Integrated Programming Environment [12], or RHIPE, to carry out the computations and analyses. The resulting text files from the previous steps were processed, and various databases of R objects were created to facilitate different analysis threads, including the transaction-based database for analyzing properties of transactions, the address-based databases for analyzing properties of addresses, etc. The structures of these databases and the procedures to create them are described in Section 2.4.1 and Section 2.5.1.

2.3.2 Summary Statistics and Growth of Bitcoin

Overall, the data we have collected consists of all the verified Bitcoin transactions over the period of 5 years, from Jan 3rd 2009, when the Bitcoin system was created, up to May 3rd 2014, when we completed the data collection. There are a total of 298,851 blocks in the blockchain, consisting of 37,993,792 transactions between 34,999,937 addresses. As of May 3rd 2014, there are a total of 12,721,275 Bitcoins that have been minted. With an exchange rate of 434.5 USD for 1 BTC on May 3rd 2014, these Bitcoins are worth a total of $5,527,393,988.

On Jan 3rd 2009, the first block, or the genesis block, was created including a single generation transaction, which is the first transaction ever. This marks the inception of the Bitcoin system and the blockchain. Since then, the Bitcoin system has drawn more and more attention, especially in the recent couple of years. Figure 2.1 shows the growth of daily number of transactions in the Bitcoin system.

Link to figure

Figure 2.1.: Daily number of Bitcoin transactions.

From Jan 2009 to Jan 2010, the daily transaction count stayed at a relatively low and constant level, with around 100–130 transactions per day. As shown in Figure 2.2,

the majority of the transactions during this time period were generation transactions, and there were very few payment transactions made. At that time, the users of the Bitcoin system were mostly those people who were interested in this new technology and were capable of setting up the mining system to obtain Bitcoins. The technology barrier for acquiring Bitcoins and using them had limited the popularity of the Bitcoin system among general users and the system was more of a plaything for tech-gurus than a method of payment.

Link to figure

Figure 2.2.: Daily fraction of generation transactions.

From Jan 2010 to June 2011, there had been a very rapid growth in the number of transactions, due to the increased publicity of the Bitcoin system and the establishment of various Bitcoin exchange sites, including the very first exchange site the Bitcoin Market [39] established on Feb 06th 2010 and a later but much more popular exchange Mt. Gox [40] established on July 17th 2010. These exchange sites had enabled a much wider range of users to participate in the Bitcoin network, by offering both conversion service between Bitcoins and the local currencies, which made the acquisition of Bitcoins much easier for users than mining their own blocks, and the online wallet service, which also made the transaction of Bitcoins easier. The fraction of generation transactions among all transactions also started to decrease rapidly as more users started to transfer Bitcoins between their addresses, and more payment transactions were made during this period.

From June 2011 to April 2012 and from April 2012 to May 2014, the Bitcoin system has become very popular and the daily transaction count remained at a very high level, around 4,000 and 32,000 transactions per day, respectively, with a steady growth during each of these two time periods. The payment transactions started to dominate the daily transactions while the fraction of generation transactions de-

creased to around 2% and then to less than 1%. During this time, the ecosystem around Bitcoin has been spawned, with many more individuals and organizations starting to accept Bitcoins as a method of payment, and a variety of new services being established for various purposes, such as online gambling services. Notably, on April 2012 the SatoshiDice [41], a Bitcoin-based online gambling service, was started and it was solely responsible for the jump of daily number of transactions between the two time periods. The SatoshiDice service is studied in details in Section 2.5.3.

Due to the tremendous amount of popularity the Bitcoin system has gained in the recent years, the value of Bitcoins, in terms of local currencies, has increased significantly, albeit the fluctuation of this immature market. Figure 2.3 shows the exchange rate from BTC to USD, and the value of a Bitcoin has increased from just a few cents, when Bitcoin exchange sites were established, to several hundreds of dollars in recent weeks, with the value peaking at about $1,150 in December 2013.

Link to figure

Figure 2.3.: Daily exchange rate from BTC to USD.

2.4 Transaction Based Database

The first database we have created is the transaction based database, in which each transaction is a unit of data record and the transactions can be accessed individually, in a batch, or as a whole. The transaction based database has enabled and greatly facilitated our analysis of the general properties of all 37,993,792 transactions in our data, including 298,851 (less than 1%) generation transactions and 37,694,941 payment transactions.

In this section, we describe the design and structure of the transaction based database, as well as the procedures to create the database, followed by the analysis of

various properties of the transactions, including the usage of addresses, values transacted, etc. Due to the difference in functionalities between generation transactions and payment transactions, their properties are analyzed separately in the following sections.

2.4.1 Design and Construction of the Transaction Based Database

In order to analyze the properties of transactions, we often need to compute various variables from the transactions, such as the number sending/receiving addresses, the values transferred, transaction fees, etc. Since these computations are all performed on a per transaction basis, it is necessary and convenient to construct a database in which each transaction is a unit of data record consisting of all the information we have about this transaction. However, the text file of the transaction level data generated through the initial processing described in Section 2.3.1 does not organize the data in such a way, instead, each line of the text file corresponds to the information of a sending address or the information of a receiving address in a transaction. Thus, a mergence of the sending information and receiving information of the same transaction into a single object is needed.

One reason behind the inconvenient structure of the text file, inconvenient in a data analyst's point of view, is the difficulty to represent a transaction as a single line in the text file, because transactions could have different numbers of sending addresses and different numbers of receiving addresses, which results in a variable and potentially large number of fields in the text file if we were to represent a transaction in a single line. Therefore, suitable data structures are also needed for the representation of a transaction in the database.

Furthermore, due to the design of the Bitcoin system that transactions reference to previous transactions and are referenced by following transactions in order to determine the source of funding, certain information regarding a transaction are not explicitly available in the current transaction without the knowledge of refer-

enced transactions and referencing transactions. For example, the amount of Bitcoins brought into the current transaction by each sending address is not available until we trace back to the referenced receiving transaction; whether or not the received Bitcoins of each receiving address were spent, and if so, when were they spent, are also unknown based on the current transaction alone. Therefore, this derived information from related transactions needs to be computed and merged into each transaction in the database.

Fortunately, with the help of RHIPE, which integrates R and Hadoop, the mergence of sending information and receiving information into transaction objects, the computations of derived information from related transactions, and the suitable data structures to represent transaction objects, are readily available: while R provides rich and generic types of data structures that suit the needs, Hadoop enables the efficient processing of GBs of text file and computations over millions of records. Here we describe in details the procedures to construct the transaction based database via RHIPE on a cluster of 11 servers and 242 processors.

Transfer Text File into HDFS.

The text file of the transaction level data is transferred from the local file system of the cluster to the HDFS, where the file is broken into smaller pieces for efficient processing and distributed to the servers on the cluster. This step is achieved through RHIPE function `rhput()`.

Input data A plain text file consisting of 37,993,792 transactions scattered into 185,229,749 lines. Each line corresponds to the sending information for each sending address or the receiving information for each receiving address in a transaction. Lines for sending information contains 8 tab-delimited columns: (1) an indicator that says this is sending information, (2) the current transaction ID, (3) the referenced transaction ID, (4) the index of the sending address in the referenced transaction, (5) the sending address, (6) the entity ID of the

sending address, (7) the time of the block that included this transaction, and (8) the block height of the block. Lines for receiving information also contains 8 tab-delimited columns, but the contents are different: (1) an indicator that says this is receiving information, (2) the current transaction ID, (3) the index of the receiving address in the current transaction, (4) the receiving address, (5) the entity ID of the receiving address, (6) the amount of Bitcoins received, (7) the time of the block that included this transaction, and (8) the block height of the block.

Output data The same text file on HDFS.

Create Initial Database of Transaction R Objects

The transaction objects are constructed by merging the sending information and receiving information in the text file, as well as the exchange rate data. Transactions are identified by the transaction ID and each transaction is stored in an R list object, consisting of 5 or 6 named elements depending on the type of the transaction. For a payment transaction, the list contains 6 named elements: the names and contents of the first 4 are (1) `time`, the time of the block that included this transaction, (2) `block`, the block height of the block, (3) `sender`, the entity ID of the sender entity, and (4) `btc2usd`, the exchange rate from BTC to USD on the date of transaction, and these four elements are all in the form of a numeric vector of length 1; the last 2 are data frames, one data frame named `sending` contains sending information with variable number of rows and 3 columns, (1) `address`, each sending address, (2) `transaction`, the referenced transaction ID of each sending address, (3) `index`, the index of each sending address in its referenced transaction, and the other data frame named `receiving` contains receiving information with variable number of rows and 4 columns, (1) `address`, each receiving address, (2) `receiver`, the entity ID of each receiving address, (3) `index`, the index of each receiving address in the current transaction, (4) `value`, the amount of Bitcoins received by each receiving address.

For generation transactions, because there are no sending addresses or referenced transactions, the element `sending` is removed and the rest of the list is the same as payment transactions.

We make the choice to use an R list to represent a transaction because we needed various structures, in this case numeric vectors and data frames, to represent different pieces of transaction information, and R lists provide the flexibility of incorporating various data structures. The list structure also enables fast access to each piece of information by name, e.g., if `trx` is the R list object of a transaction, the time of the transaction can be accessed by `trx$time` and the receiving addresses can be accessed by `trx$receiving$address` in R, respectively.

The initial database of transaction R objects is constructed through a MapReduce job described below.

Input data The text file of transaction level data on HDFS.

Output data 37,993,792 transactions, each in the form of a key-value pair, with the transaction ID as the key, and an R list containing transaction information as the value.

Map Parse each line of the text file as a character string. For each line, or each piece of sending information or receiving information, the current transaction ID is extracted and used as the key of the intermediate data. An R vector containing the rest of the fields of the line is used as the value of the intermediate data. Intermediate data containing such key-value pairs are transferred to Reduce.

Reduce All the pieces of sending information and receiving information corresponding to the same transaction are assembled, re-organized and stored as an R list described as above. During this step, the exchange rate data is also made available across the whole cluster as an R data frame, and the exchange rate on the date of transaction is looked up and appended to the R list object. With the transaction ID as the key, and the R list as the value, a key-value pair is outputted for each transaction.

Compute Derived Information from Related Transactions

The transaction R objects in the initial database do not contain the information such as how much Bitcoins were brought in by each sending address, when were they received, when were the Bitcoins received in the current transaction spent. This information can be derived after the current transaction is correlated with its referenced and referencing transactions. In order to obtain the derived information, the transaction objects are disassembled into one piece of receiving information, consisting of the element `receiving` along with other information regarding the current transaction, and many pieces of sending information each consisting one row of the element `sending` along with other information regarding the current transaction. The receiving information in the current transaction are matched with the corresponding pieces of sending information from its referencing transactions, and similarly, each piece of sending information is matched with the corresponding receiving information from its referenced transaction, and then the derived information is computed and exchanged.

This process is accomplished in a MapReduce job described below.

Input data 37,993,792 transactions from the previous step.

Output data 125,120,532 pieces of sending/receiving information, each in the form of a key-value pair, with the current transaction ID as the key and an R list containing sending/receiving information and the derived information as the value.

Map For each transaction object, one key-value pair is generated as intermediate data for the element `receiving` and one key-value pair is generation for each row of the element `sending`. For `receiving`, the current transaction ID is used as the key, and a subset of the elements in the transaction object, including `time`, `block`, `sender`, `btc2usd`, and `receiving` are extracted and used as the value. For each row of `sending`, the referenced transaction ID is extracted as the key, and rest of the row, along with the current transaction ID, `time`, and

`block`, are extracted and used as the value. Intermediate data containing such key-value pairs are transferred to Reduce.

Reduce Receiving information and its pieces of referencing sending information are assembled and the derived information is exchanged. More specifically, after matching receiving and its referencing sending information by address and index, the receiving values, time and block of the receiving transaction for each address are copied from receiving information to each of the referencing sending information; the sending transaction ID, time and block of the sending transaction for each address, are copied from each sending information to the receiving information. After the exchange of information, the receiving information are outputted as the value with the current transaction ID as the key; each sending information are outputted as the value with the referencing transaction ID as the key.

Construct the Final Transaction Based Database

After the computation and exchange of the derived information between each transaction and its referencing and referenced transactions, a final transaction object with the derived information is formed by re-assembling its sending information and receiving information. Similar to the initial database, transactions are identified by the transaction ID and each transaction is stored in an R list object, with the same 5 or 6 named elements as the initial database. However, the derived information is now contained in the elements `sending` and `receiving`.

More specifically, in addition to the first 4 named elements of numeric vectors of length 1: (1) `time`, the time of the block that included this transaction, (2) `block`, the block height of the block, (3) `sender`, the entity ID of the sender entity, and (4) `btc2usd`, the exchange rate from BTC to USD on the date of transaction, the transaction R object also includes a data frame `sending` containing a variable number of rows and 6 columns: (1) `address`, each sending address, (2) `transaction`, the

referenced transaction ID of each sending address, (3) `index`, the index of each sending address in its referenced transaction, (4) `value`, the value brought in by each sending address, (5) `time`, the time when each sending address received these Bitcoins, (6) `block`, the block in which each sending address received these Bitcoins; and a data frame `receiving` containing a variable number of rows and 7 columns, (1) `address`, each receiving address, (2) `receiver`, the entity ID of each receiving address, (3) `index`, the index of each receiving address in the current transaction, (4) `value`, the amount of Bitcoins received by each receiving address, (5) `transaction`, the transaction ID in which the received Bitcoins were spent, (6) `time`, the time when the received Bitcoins were spent, and (7) `block`, the block in which the received Bitcoins were spent. If the received Bitcoins of a receiving address in a transaction are not yet spent, then the corresponding entries of `transaction`, `time`, and `block` in the data frame `receiving` is assigned with `NA`.

The final transaction based database is constructed in the following MapReduce job.

Input data 125,120,532 pieces of sending/receiving information from the previous step.

Output data 37,993,792 transactions, each in the form of a key-value pair, with the transaction ID as the key, and an R list containing transaction information as the value.

Map For each piece of sending/receiving information, the key-value pair is outputted as it is.

Reduce All pieces of the sending information and the receiving information corresponding to the same transaction are assembled. Duplicated information between pieces of sending information and receiving information is removed and the rest of the information are stored in an R list object. With the current transaction ID as the key, and the R list as the value, a key-value pair is outputted for each transaction.

With the construction of the transaction based database, individual queries of specific transactions can be made directly to the database, and batch or entire database scan can also be easily accomplished through MapReduce jobs with the database provided as the input. Thus the computations of variables on a per transaction basis have been made very easy and efficient.

In the following subsections, we make heavy use of the transaction based database and analyze various properties of generation transactions and payment transaction, such as the growth of daily transaction, the number of addresses used, values of Bitcoins involved, and how fast Bitcoins are spent, etc.

2.4.2 Analysis of Properties of Generation Transactions

The generation transactions are created by miners that generated each block of the blockchain. The miners that created each block had verified the payment transactions included in the block on behalf of the Bitcoin network, and as a reward for their service, the block rewards and the combined transaction fees from the included payment transactions were sent to the miners via the generation transaction in the block.

The generation transactions can be extracted from the transaction based database by simply filtering those transaction objects where the element sending is missing. There are a total of 298,851 blocks in the blockchain and each block consists of a single generation transaction and zero or many payment transactions. The block rewards in the generation transactions are newly minted Bitcoins, and the number of Bitcoins minted per block/generation transaction was 50 BTC when the Bitcoin system was started, and is halved every 210,000 blocks or about 4 years, with the current block rewards being 25 BTC. This results in a controlled supply of the Bitcoin as a currency and the number of Bitcoins in existence will never exceed 21 million. As of May 3rd 2014, there are a total of 12,721,275 Bitcoins minted.

The Growth of Generation Transactions

To prevent the miners from creating too many blocks too fast and thus causing inflation in the Bitcoin currency, the difficulty in creating blocks, which is largely the computation effort to complete the proof-of-work, is adjusted by the Bitcoin system so that the blocks are created at a relatively constant rate, about one block every 10 minutes. As shown in Figure 2.4, the number of generation transactions has remained at around 2^7–2^8 per day since January 2010 despite the rapid growth of usage of Bitcoins during this time period. Before January 2010, however, the daily number of generation transactions was significantly lower, and this is simply because the lack of popularity of the Bitcoin system and the low value of Bitcoins, and thus there were too few miners and those miners were not working hard enough to create blocks.

Link to figure

Figure 2.4.: Daily number of generation transactions.

Usage of Addresses

In generation transactions, the miners can specify one or many receiving addresses. On September 14th 2010, the block rewards and combined transaction fees in block 79764 were sent to two receiving addresses, and this is the first occurrence of split allocation in generation transactions. Overall, the distribution of number of receiving addresses in generation transactions is very skewed as shown in Figure 2.5. There is only one receiving address in over 95% of the generation transactions, and we will refer to these as single-receiving generation transactions; however, there are a large number, in hundreds, of receiving addresses in very few generation transactions, with the largest being 919 addresses. Recall that miners can work as individual miners or

participate in mining pools in the Bitcoin network, the dominance of single-receiving generation transactions does not necessarily imply that most of the blocks are mined by individual miners. In fact, as we shall see in Section 2.5.4, mining pools, especially a few major ones, are also specifying a single address to receive the block rewards and combined transaction fees in generation transactions and then re-distribute the earnings to its participants in a series of following transactions.

Link to figure

Figure 2.5.: Quantiles of number of receiving addresses in generation transactions.

How Fast Are Minted Bitcoins Spent

The Bitcoins received by the miners in generation transactions are usually spent fairly quickly, either transferred to other addresses owned by the miners, or re-distributed to the participants of the mining pools. We quantify how fast these Bitcoins are spent by the number of confirmations waited before they are used as funding sources in following transactions. As shown in Figure 2.6, in over 50% of the generation transactions, the Bitcoins are spent with less than 2^7 confirmations, or within 24 hours; and in about 70% of the generation transactions, the Bitcoins are spent with less than 2^{10} confirmations, or within a week.

Link to figure

Figure 2.6.: Quantiles of number of confirmations before the Bitcoins in generation transactions are spent.

It is also worth noting in Figure 2.6 that there are 39,460 generation transactions, or 13% of all generation transactions, in which the Bitcoins have not yet been spent as of May 3rd 2014, with a total value of 1,899,484 BTC, or 15% of all minted Bitcoins. These Bitcoins were never in circulation after being minted and remain dormant ever since. (The term "dormant Bitcoin" were introduced in [31].) In fact, most of these generation transactions occurred in the earlier days of the Bitcoin system. As shown in Figure 2.17, by the end of 2010, the number of such generation transactions had reached about 35,000, and the value of dormant Bitcoins had already reached 1,757,705 BTC; since 2011, the number of such generation transactions had only increased slightly and the Bitcoins minted during this time period had been in circulation; in the most recent months, there was a small jump in the number of such generation transactions, likely due to the fact that this is close to the end of our data collection and the Bitcoins minted in those generation transactions were too new to be spent.

Link to figure

Figure 2.7.: Cumulative number of generation transactions in which the Bitcoins minted were not spent as of May 3rd 2014.

The reason that these Bitcoins were never spent after minted is unknown, except that those newly minted ones which might be spent soon. It could simply be that the earlier miners/owners of these Bitcoins have lost their private key, thus effectively the control/ownership, to the receiving addresses in those generation transactions and were not able to spend those Bitcoins anymore. Since the earlier miners in the Bitcoin network were mostly playing with this new technology and might not foresee the value of Bitcoins nowadays, it is possible that they did not make any effort to preserve the Bitcoins they have obtained through mining. Nevertheless, this large amount of dormant Bitcoins still pose a threat to this immature Bitcoin market as

one can imagine the impact they would have to the value of Bitcoins if they were somehow recovered and entered into circulation.

Notice that in Figure 2.6, the largest number of confirmations for the Bitcoins minted in a generation transaction to be spent is 282,031, and it turns out that the Bitcoins in this particular generation transaction were minted on January 30th 2009 and were spent 5 years later on February 2nd 2014. So even the earliest minted Bitcoins can still be and are being recovered. In fact, there are 2,298 generation transactions with a total value of 114,902 BTC that were spent 3 or more years after they were minted. Given the current value of Bitcoins in local currencies, it is no surprise that the earlier miners are doing whatever they can to recover those Bitcoins they have minted.

2.4.3 Analysis of Properties of Payment Transactions

Payment transactions are used to transfer Bitcoins from the address(es) of the sender to the address(es) of the receiver(s). They can be extracted from the transaction based database by simply filtering those transaction objects where the element `sending` is not missing. There are a total of 37,694,941 payment transactions, that is over 99% of all transactions, and payment transactions have dominated the daily transactions in the Bitcoin network since June 2011. In this section, we break down the steps of making a payment transaction, and explore the properties of each in the data.

Usage of Addresses

To make a payment in Bitcoin via a payment transaction, the payer/sender first must possess one or many addresses that have previously received some amount of Bitcoins in previous generation or payment transactions. If the sender owns multiple such addresses or a single address that have previously received Bitcoins multiple times in a single or multiple transactions, the sender could customize which of them

will be used as sending addresses in this payment transaction. However, for convenience, most of the available Bitcoin client programs and online wallet services can automate the selection of sending addresses and complete the transaction for the sender.

The overall distribution of the number of sending addresses in payment transactions is shown in Figure 2.8. Overall, in about 62% of the payment transactions, there is a single sending address; and in all other transactions, two or more sending addresses are pooled together to provide funding for a single transaction. The number of sending addresses in a single transaction can be as large as 2,585, however, the fraction of transactions with 20 or more sending addresses is fairly small, being less than 1%.

Link to figure

Figure 2.8.: Quantiles of number sending addresses in payment transactions.

We refer to those payment transactions with a single sending address as one-sending transactions and all other transactions with two or more sending addresses as multi-sending transactions. The relative volume of these transactions changed along time and is suggesting that the usage of sending addresses in payment transaction is evolving, as shown in Figure 2.9. Prior to March 2011, there were not sufficient payment transactions and thus the fraction of one-sending transactions varied quite a lot; however, since payment transactions began to dominate daily transactions in March 2011, about 80% of the payment transactions were one-sending transactions, and the fraction has gradually decreased to about 50% since then. The increasing usage of multi-sending transactions is suggesting that, as more and more payment transactions are made, Bitcoins are more often broken into smaller bills and then merged when spent. This also increases the occurrence of co-sending relations between addresses and will impact the anonymity as discussed in Section 2.6.

Link to figure

Figure 2.9.: Monthly fraction of one-sending transactions.

After taking care of the sending addresses, the next step of making a payment transaction is for the sender to specify the receiving address(es) of the receiver(s) of the payment. The sender can specify any number of receiving addresses to transfer Bitcoins to, and the receiving addresses need not to be unique, which means the sender can even transfer Bitcoins to the same address multiple times in a single transaction.

The distribution of the number of receiving addresses in payment transactions is quite skewed as shown in Figure 2.10. In majority of the payment transactions there are a very small number of receiving addresses, with about 6% of the payment transactions having a single receiving address, about 87% having two receiving addresses, and about 4% having three receiving addresses; the remaining 3% of payment transactions have four up to hundreds or thousands of receiving addresses, with the largest number of receiving addresses being 3,075.

Link to figure

Figure 2.10.: Quantiles of number receiving addresses in payment transactions.

We refer to those transactions with a single receiving address as one-receiving transactions, those with two receiving addresses as two-receiving transactions, and the remaining transactions as three-or-more-receiving transactions. The change in the relative volume of these transactions along time is shown in Figure 2.11. Before March 2011, none of the payment transactions had three or more receiving addresses, and actually there were not many other payment transactions in the Bitcoin network

either, which explains the fluctuation of the fraction of one-receiving transactions and two-receiving transactions; after March 2011, payment transactions started to dominate daily transactions and nearly 90% of them were two-receiving transactions; however, in the most recent months, the fraction of two-receiving transactions had seen a slight decrease to about 80%, while the other two taking about 10% each.

Link to figure

Figure 2.11.: Monthly fraction of one-receiving, two-receiving, and three-or-more-receiving transactions.

Certainly users in the Bitcoin network could be, and in fact they have been, taking advantage of the ability to transfer Bitcoins to as many as thousands of receiving addresses in a single transaction, however, recall the existence of change addresses discussed in Section 2.2.3, while there are two receiving addresses in a two-receiving transaction, one of the two is the change address to collect the change for the sender, and the other is the payment address that belongs to the true receiver of the transaction. So effectively, there is only one receiver in a two-receiving transaction and the dominance of two-receiving transactions is simply reflecting the fact that most of the payment transactions are from the sender to a single receiver. Similarly, there is at least one change address, in some cases there could be more than one, in three-and-more-receiving transactions, making the effective number of receivers less than the number of receiving addresses in those transactions.

Transaction Values

After specifying the sending addresses and receiving addresses, the sender then needs to specify the receiving values, i.e., the amount of Bitcoins to be sent to each of the receiving addresses. The smallest unit of Bitcoin is a "satoshi", which is 10^{-8} of

one bitcoin (BTC), so the receiving values are always multiples of a satoshi. However, it is possible to transfer 0 BTC to a receiving address in a transaction, and there have been multiple occurrences of such zero value transactions.

In a one-receiving transaction, the payment value being transacted is the receiving value of the single receiving address. In two-receiving transactions and three-or-more-receiving transactions, however, the actual payment values are unknown due to the existence of the change addresses. So instead, we will study the overall values of Bitcoins being involved in each transaction.

We define the transaction value of a payment transaction to be the value being involved, i.e., it is the combined values in the sending addresses in this transaction. Figure 2.12 shows the overall distribution of transaction values. The transaction values can be as small as 0 satoshi and as large as 550,000 BTC; the median of transaction values is close to 1 BTC and over 80% of the values are within the range of 2^{-5} BTC to 2^{5} BTC.

Link to figure

Figure 2.12.: Quantiles of transaction values in payment transactions.

As Bitcoins become more valuable in local currencies, the transaction values have become smaller in recent years, meaning more small bills of Bitcoins are being transferred. As shown in Figure 2.13, there was a decreasing trend in the monthly percentiles of transaction values. Since May 2012, the monthly median values of transaction values had remained less than 1 BTC except in a couple of months, which means over 50% of payment transactions involved less than 1 BTC; by the end of April 2014, the 3rd quartile had dropped to 1 BTC and 75% of payment transactions involved less than or equal to 1 BTC. The big drop in percentiles between April 2012 to June 2012 can be attributed to SatoshiDice, which was established April 2012 and began to

contributed to a significant proportion of payment transactions in the Bitcoin network since then. More details about SatoshiDice are discussed in Section 2.5.3.

Link to figure

Figure 2.13.: Monthly selected percentiles of transaction values in payment transactions.

Transaction Fees

Before the sender completes a payment transaction, the last step, which is optional, is to specify a transaction fee to be collected by the miner who would later include and verify this transaction when generating a block. Effectively, the transaction fee is computed as the (positive) difference between the combined sending values and the combined receiving values, and any amount of Bitcoins that is not sent to receiving addresses will be considered as the transaction fee. It is often suggested that the transaction fee should be a fraction of the payment value, e.g., 5% of the payment, but transaction fees are voluntary on the part of the senders, as they can include any fee or none at all in a payment transaction. On the other hand, the miners need not necessarily to process this transaction and include it in the new block being generated, which means the verification/confirmation of this payment transaction could be delayed. Therefore, the transaction fee is an incentive from the sender to make sure that a particular transaction will quickly get included into one of the following blocks being generated. Furthermore, in a lot of popular implementations of the Bitcoin client programs and online wallet services, there is a default transaction fee imposed when making a payment transaction [42], especially on those with a very small amount of transaction values involved.

It is shown in Figure 2.14 that with the increased usage of the Bitcoin system in the past few years, more and more payment transactions were made with a transaction fee. Prior to January 2011, there were very few payment transactions made and a very small fraction of them were made with a transaction fee; however, since then the fraction had increased significantly, especially from April 2012 to June 2012, when the fraction had doubled and kept increasing to over 95% in the recent months.

Link to figure

Figure 2.14.: Monthly fraction of payment transactions with a transaction fee.

It seems that the users in the Bitcoin network are being good citizens and paying the transaction fees as a good gesture, however, considering the fact that a default transaction fee is usually imposed by the Bitcoin client programs and online wallet services the users are using, the users are actually being forced to pay the fees, unless they have sufficient technical knowledge to either utilize the low level Bitcoin raw transaction [43] or take advantage of other non-standard implementations of the Bitcoin client program without enforced transaction fee [44]. This is also partially reflected in Figure 2.15 of the quantiles of overall transactions fees, where a few popular values dominates in the transactions, as opposed to a less concentrated distribution one would expect if the transaction fees are paid as the suggested fraction of payment values. A few popular values are 0.001 BTC, 0.0005 BTC, 0.0002 BTC, and 0.0001 BTC, and their combined fraction of payment transactions is over 91%. These values happen to be the default amount of transaction fees imposed by those popular implementations of Bitcoin client programs and online wallet services . Therefore, while the most technical users are able to avoid the transaction fees, less technical users, which likely are the majority in the user community, have to pay a fee for the transactions they are making. And of course, the default fee has declined by a factor

of 100 from 0.01 BTC in the earliest days of Bitcoins to the recent 0.0001 BTC as the exchange rate has risen.

Link to figure

Figure 2.15.: Quantiles of transaction fees in payment transactions.

How Fast Are Bitcoins Spent

After a payment transaction has been created and broadcasted by the sender, it will be relayed by the participants of the Bitcoin network, and eventually be picked up by the targeted receiver(s) and the miners. Before getting a sufficient number of confirmations from the miners, this transaction can still be voided and excluded from the blockchain. However, the received Bitcoins in each receiving address are available to be spent in another payment transaction immediately after the original transaction is created. Of course, if the original transaction is voided and excluded from the blockchain, all following transactions will be voided as well.

It is suggested in [22] that a minimum of 6 confirmations, which is about 1 hour, is needed to ensure that a transaction is valid, and typically merchants accepting Bitcoin as a method of payment require much more confirmations before they ship out their products. However, after tracking all receivings instances that have been spent on or before May 3rd 2014 in all payment transactions, we found out that Bitcoins received are often spent faster than that. As shown in Figure 2.16, in about 45% of all receiving instances the received Bitcoins were spent with less than 6 confirmations, and about 16% were even spent without any confirmation at all.

Link to figure

Figure 2.16.: Quantiles of number of confirmations before received Bitcoins are spent.

One typical situation that the received Bitcoins could be safely spent with few or even none confirmations is the existence of change addresses: if the receiving instance in a payment transaction is the receiving of the change, then the sender is certain that the original transaction is valid and thus continue to spend the change received without taking any risk. Furthermore, there are various other situations where the receiver deliberately spend the received Bitcoins very fast, including SatoshiDice, which is discussed in Section 2.5.3.

2.4.4 Analysis of Dormant Bitcoins

As we have seen in Section 2.4.2, there are a noticeable amount of dormant Bitcoins that have not yet been in circulation as of May 3rd 2014 since they had been minted. In fact, there are also Bitcoins that are transferred in payment transactions for a period of time after minted in generation transactions, and then became inactive ever since, further reducing the amount of Bitcoins in circulation.

Combining both generation transactions and payment transactions, Figure 2.17 shows that, for any given date, the cumulative amount of Bitcoins that have become inactive since then, i.e., those Bitcoins were never spent from that date up to May 3rd 2014. By the end of 2010, there were about 2 million dormant Bitcoins, among which 1.76 million were contributed by generation transactions. By the end of 2013, the amount of dormant Bitcoins had reached 4 million BTC and the increase was mostly due to inactive Bitcoins received in payment transactions. As it gets closer to the end of our data collection, the amount of "dormant" Bitcoins quickly climbed up, simply because there had not been enough time for them to be transferred again. And lastly, the value on the top right corner in the figure, which reads 12.72 million

BTC on May 3rd 2014, corresponds to the total amount of Bitcoins that have been minted so far.

Link to figure

Figure 2.17.: Cumulative amount of Bitcoins that were not spent from any given date to May 3rd 2014.

Therefore, of all the Bitcoins that have minted so far, over 30% have been out of circulation for over one and a half years, suggesting that the remaining Bitcoins have been transferred very frequently considering the large number of Bitcoin transactions we have seen in the recent years. Also, as we have discussed before, the large amount of dormant Bitcoins are posing a threat to the immature Bitcoin market as they could enter or re-enter into circulation and shock the market. In fact, similar to what we have seen for the dormant Bitcoins in generation transactions, the ancient Bitcoins in payment transactions could also be discovered and become active again. In Figure 2.16, the largest number of confirmations for the Bitcoins received in a payment transaction to be spent is 267,338, corresponding to a payment transaction occurred in February 2009, where the received Bitcoins were spent about five years later.

2.5 Address Based Databases

Addresses identify the sender and receiver entities of a transaction. Studying the profile or usage pattern of addresses is very important, if not more important than studying transactions themselves, to understand how the Bitcoin system is being used and to study various entities in the Bitcoin network. Through our analysis, we have found that the usage pattern of an address can often be characterized by 3 groups of properties of this address, and address based databases that contain these properties

for each address are needed to enable efficient access and analysis of these properties, as well as the creation of visualizations for these properties on a per address basis.

The first group of properties aim to summarize the activity level of the addresses. An activity of an address is essentially a transaction this address is used in, and we define 4 types of activities for an address based on how this address is being used in each transaction: (1) if this is a generation transaction, this address is receiving the block rewards and we say that this address is conducting a generation activity; (2) in a payment transaction, if this address is a receiving address but not a sending address, we say that this address is conducting a receiving activity; (3) if this address is a sending address but not a receiving address, we say that this address is conducting a sending activity; (4) and if this address is both a sending address and a receiving address in the same transaction, then we say that this address is conducting a self-sending activity. Generation, receiving, and self-sending activities are receiving related activities as the address is receiving some amount of Bitcoins, and sending and self-sending activities are sending related activities as the is spending some amount of Bitcoins. The activity level of an address can then be summarized by a few variables: first of all, the overall level of activity of an address, i.e., the number of transactions of an address, and the number of activities of each type tell a lot about how actively this address is being used; furthermore, the time and block of the beginning and ending of sending/receiving related activities, and the number of active blocks, i.e., the number of blocks this address is conducting activities, provide information about this address's lifetime; lastly, the ending account balance of an address, i.e., how much Bitcoins this address has received but not yet spent as of May 3rd 2014, often gives a reasonable indication on whether or not this address will remain active. These variables are aggregated summaries and they take one value for each address, and they should be included in the address based database.

The second group of properties summarize the usage of addresses in those activities of a particular address. These include the distributions of the following variables across all or some types of the activities of this address: (1) the number of sending ad-

dresses and number of receiving addresses in the activities of each type; (2) in sending related activities, where this address is a sending address, the number of replications of this address as the sending address, i.e., the number of times this address is specified as the sending address in the same transaction; (3) in receiving related activities where this address is a receiving address, the number of confirmations this address waited before the Bitcoins received are spent; (4) the number of receiving activities of this address contributed by each sender entity. These variables quantify how many as well as how fast addresses are used, and a representation of their distributions should be included in the address based database.

The third group of properties carry the detailed information of each individual activity, including the activity type, time and block of this activity, and the amount of Bitcoins of this address that are involved in this activity. These properties are no longer aggregated or summarized information as the previous two groups are, and the address based database should nonetheless include them to provide comprehensive information regarding activities.

In the following subsections, we first describe our design and structure of various address based databases that enable query and analysis of those properties for all 34,999,937 addresses, along with the construction of these databases, then we present our analysis of various properties of these addresses, and we conclude this section by the studies of a few representative groups of addresses with typical usage patterns.

2.5.1 Design and Construction of the Address Based Databases

The address based databases are created from the transaction based database as described in Section 2.4.1, where activities/transactions are conveniently stored as stand-alone objects. However, for each transaction object in the transaction based database, the information contained needs to be propagated for each address used in this transaction and then all pieces of transaction information of the same address need to be assembled to produce those properties mentioned above.

One naive design of an address based database that contains all the properties for each address is simply to form a list object consisting of all the activities/transactions for each address after they are assembled. Certainly, thus constructed database contains all the information we would need since we would be able to compute all the properties for each address given the list of all its transactions. However, there are two issues with this design. Firstly, because of the propagation of transactions for every address in a transaction, the size of the resulting database becomes quite large as there are usually at least 3 or more addresses in a transaction, and this would hurt the efficiency of accessing the database and consequent analyses. Secondly, as we shall see in Section 2.5.2, there are a few addresses that have been used in a huge number of transactions, in the order of millions, thus the size of the list objects of all transactions for these addresses would exceed the upper limit on the size of a single object in R and these list objects will have to be broken into smaller pieces, which quickly complicates the computation of those properties since the information for a single address could potentially span across multiple list objects in the database and these objects need to be assembled before the computation and analysis of those properties.

Therefore, instead of storing all activities/transactions of an address as a list object in the address based database, we decide to extract and compute the variables of those properties and store them with suitable data structures for each address in our address based databases. We have created three independent address based databases, one for each group of properties, where individual queries for a particular address and batch or entire database scan can be made to any one or any combination of these databases. Note that these databases could be easily merged into a single database, but we choose not to do so simply because we constructed these databases sequentially during different stages of our analysis and having three independent databases provide the same level of accessibility and efficiency with slightly more flexibility compared to a single database. Also, as new property variables are being discovered during the interactive analysis, it is usually easier and more efficient to

create additional address based databases for these properties than to re-compute all the old properties and merge them with the new ones.

Here we describe in details the procedures to construct these address based databases via RHIPE on a cluster of 11 servers and 242 processors.

Construct the Activity Level Summary Database

The activity level summary object consists of the following variables for each address: the total number of activities, the number of activities of each type, the time and block of the beginning and ending of sending/receiving related activities, the number of active blocks, and the ending account balance. For each transaction object and each address in the transaction, these variables are computed and then aggregated by address to produce the activity level summary object for each address. Each activity level summary object is stored as an R named vector of numeric values, each of which corresponds to a variable.

The activity level summary database is constructed in the following MapReduce job.

Input data The transaction based database, consisting of 37,993,792 transactions, each in the form of a key-value pair, with the transaction ID as the key, and an R list containing transaction information as the value.

Output data 34,999,937 activity level summary objects, each in the form of a key-value pair, with the address as the key, and an R vector containing activity level summary information of the address as the value.

Map For each transaction and each address in the transaction, indicator variables that indicate whether or not this activity is of a particular type for this address are formed, the time and block of this transaction are extracted, and if this address is a receiving address and the Bitcoins received are not yet spent, the balance is assigned with the receiving value, otherwise it is assigned with 0.

With the address as the key, and an R vector consisting of those variables with proper names as the value, the key-value pairs are transferred to Reduce.

Reduce All activities of the same address, in the form of R vectors, are assembled. The summary variables are computed: each of the indicator variables is aggregated across all activities, the max and min of time and block for sending/receiving related activities are computed, the number of unique blocks and the balances are also aggregated. The summary variables are again stored in an R named vector. With the address as the key, and the R vector as the value, a key-value pair is outputted for each address.

Construct the Address Usage Summary Database

The address usage summary object consists of information about the distributions of the following variables for each address: the number of sending addresses and number of receiving addresses in the activities of each type, the number of replications of this address as the sending address in sending related activities, the number of confirmations this address waited before the Bitcoins received are spent in receiving related activities, the number of receiving activities of this address contributed by each sender entity. Due to the skewness and extreme values in these variables, the mean and standard deviation are not suitable summaries for their distributions, instead, we compute up to 5,000 quantiles with probabilities equally spaced between 0 and 1 for each variable. The address usage summary object is formed as an R list, each element of which corresponds to a variable and is formed as an R data frame of up to 5,000 rows and two columns: the f-value and corresponding quantiles.

The address usage summary database is constructed in the following MapReduce job.

Input data The transaction based database, consisting of 37,993,792 transactions, each in the form of a key-value pair, with the transaction ID as the key, and an R list containing transaction information as the value.

Output data 34,999,937 address usage summary objects, each in the form of a key-value pair, with the address as the key, and an R list containing address usage summary information of the address as the value.

Map For each transaction and each address in the transaction, the type of activity is determined, the number of sending/receiving addresses, the number of replications and the number of confirmations are computed, and the sender entity ID is extracted. Notice that unlike the other variables, the number of values for the number of confirmations could exceed 1 if the same address is specified as receiving address multiple times. So each of these variables are stored as an R vector, and an R list consisting of these vectors is formed. With the address as the key, and the R list as the value, the key-value pairs are transferred to Reduce.

Reduce All activities of the same address, in the form of R lists, are assembled. The variables are re-organized by type of activities, and the quantiles are computed and formed as an R data frame. For the sender entity variable, the number of activities by sender entity is computed and the quantiles are obtained. With the address as the key, and an R list of those data frames as the value, a key-value pair is outputted for each address.

Construct the Activity Detail Database

The activity detail object consists of detailed information about each individual activity: the type of this activity, the time and block of this activity, and the amount of Bitcoins of this address that are involved in this activity. Each activity detail object is constructed as an R data frame with each row being an activity.

The activity detail database is constructed in the following MapReduce job.

Input data The transaction based database, consisting of 37,993,792 transactions, each in the form of a key-value pair, with the transaction ID as the key, and an R list containing transaction information as the value.

Output data 34,999,937 activity detail objects, each in the form of a key-value pair, with the address as the key, and an R data frame containing activity detail information of the address as the value.

Map For each transaction and each address in the transaction, the type of activity is determined, the time and block of this activity, as well as the amount of Bitcoins involved are extracted, and these variables are formed into an R vector. With the address as the key, and the vector as the value, the key-value pairs are transferred to Reduce.

Reduce All activities of the same address, in the form of R vectors, are assembled. The vectors are binded into an R data frame with each vector/activity being a row in the data frame, and the data frame is then ordered by time of activity. With the address as the key, and the data frame as the value, a key-value pair is outputted for each address.

With the construction of the address based databases, queries of individual address or groups of addresses can be made directly to the databases and their properties can be obtained, and entire database scan for all addresses can also be accomplished through MapReduce jobs. In the following subsections, we first present an overview of properties of all addresses by utilizing the activity level summary database, then we focus on a few specific groups of addresses with interesting and representative usage patterns by making heavy use of all three databases.

2.5.2 Analysis of Properties of Addresses

As of May 3rd 2014, we have observed a total of 34,999,937 addresses that have ever been used in the Bitcoin network. While there are groups of addresses shar-

ing similar usage patterns, the properties could vary drastically from addresses to addresses.

Activity Level of Addresses

The distribution of number of activities/transactions of all types for each address is shown in Figure 2.18. Clearly, the distribution is very concentrated and heavy-tailed: close to 5% of addresses have conducted only 1 activity, in which the address is receiving and the Bitcoins received are never spent; over 80% of addresses have 2 activities, most likely 1 receiving activity followed by 1 sending activity draining the address; less than 1% of addresses have conducted 28 or more activities, however, there are a small number of popular addresses that account for a significant proportion of all the activities, with the largest one conducting 3,153,571 activities, in other words, this address has occurred in 8.3% of all transactions. We should also notice that when the the number of activities is small, the address count/fraction for odd numbers of activities is always smaller than that of the subsequent even numbers, and this is hardly surprising, as it is simply suggesting that many addresses are used in as many sending activities as receiving activities. Therefore, the majority of addresses are used only a small number of times and are never used again, while very few address have been used extraordinarily frequently. It is generally suggested that a new address should be generated for every receiving related activity, in order to achieve better privacy, and our observation does suggest that many addresses are generated and used in this way.

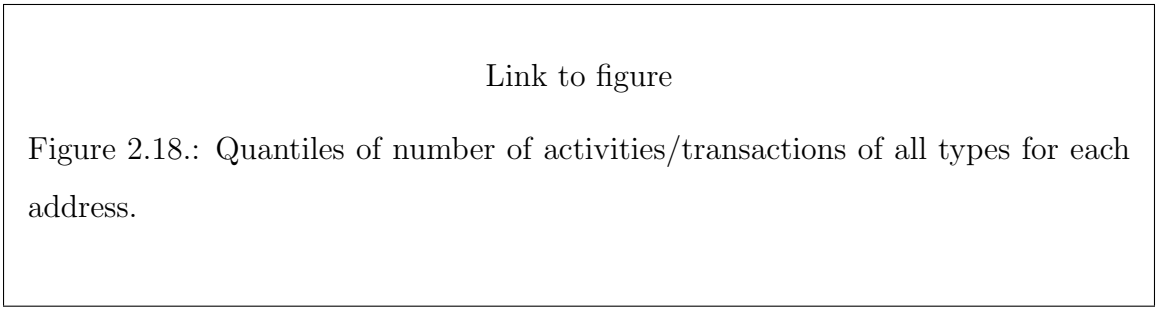

Figure 2.18.: Quantiles of number of activities/transactions of all types for each address.

Figure 2.19 compares the distribution of the number of activities/transactions for each address with the Pareto distribution, and this gives a better visualization of the tails of the distribution. Notice that in the top right corner, there are two groups of adjacent points having similar values on the horizontal axis within groups, suggesting that there are two clusters of addresses with similar number of activities. These turned out to be addresses belong to a popular Bitcoin based gambling service, the SatoshiDice, and in fact, many of the top addresses, including the largest one with 3,153,571 activities, also belong to SatoshiDice. The SatoshiDice addresses are analyzed in details in Section 2.5.3.

Link to figure

Figure 2.19.: Pareto quantile-quantile plot of number of activities/transactions of all types for each address.

For each address, the number of sending related activities, including sending and self-sending, is plotted against the number of receiving related activities, including generation, receiving, and self-sending, in Figure 2.20. This figure is produced via R package "hexbin" [45], and it is a variation of the usual scatter plot, but tailored for large datasets. The values of (x, y) pairs to be plotted are grouped into hexagon bins and the count of values in each bin is plotted/colored instead of the actual values. 28,078,037 addresses, or over 80% of all, have exactly 1 sending activity and 1 receiving activity; 2,017,879 addresses, or 5.77% of all, have exactly 2 sending activities and 2 receiving activities; 2,032,908 addresses, 5.81% of all, have only been used in receiving related activities but never been used in sending related activities, among which, 1,599,702 addresses, or 4.57% of all, have received once and the Bitcoins received were never spent. For most addresses, the number of sending related activities is no more than that of receiving related activities. This is because the Bitcoins received in multiple receiving related activities are often merged to provide

funding in a subsequent sending related activity. However, there are a few addresses that have conducted more sending related activities than receiving related activities, suggesting that they are receiving multiple times in the same receiving activity and these receivings are spent separately in subsequent sending activities.

Link to figure

Figure 2.20.: Scatterplot (based on hexagon binning) of number of sending related activities and number of receiving related activities of each address.

The number of activities/transactions and number of active blocks for each address are plotted in Figure 2.21. While majority of the points fall fairly close to the $y = x$ reference line, suggesting these addresses' activities are well spread across blocks, there are addresses whose activities are quite concentrated and producing hundreds of activities per active block, including those addresses with largest number of activities/transactions.

Link to figure

Figure 2.21.: Scatterplot (based on hexagon binning) of number of activities/transactions and number of active blocks for each address.

Inactive Addresses

The account balance of an address at a given time is the sum of received Bitcoins that are not yet spent at that time. As of May 3rd 2014, 32,098,717 addresses, or 92% of all addresses, have a balance of 0, and only 2,901,220 addresses hold a positive balance. Notice that it is not possible for an address to have a negative balance, since

addresses could only spent the Bitcoins they have previously received. Figure 2.22 shows the distribution of balance for addresses with a non-zero balance. Most of these addresses have a small balance, with about 15% having a balance of only 1 satoshi and over 90% of addresses having a balance of 1 BTC or less, but a few addresses have very large balance. In fact, the 1,000 "wealthiest" addresses hold nearly 40% of all the Bitcoins that have ever been minted. The address with the largest balance is 1FfmbHfnpaZjKFvyi1okTjJJusN455paPH and its balance is 144,341.53 BTC, worth of over 60 million USD as of May 3rd 2014. This address belongs to the FBI and it was used to seize the Bitcoins from the alleged owner and operator of "Silk Road", a website designed to enable its users to buy and sell illegal drugs and other unlawful goods and services using the Bitcoin system [46] [47] [48].

Link to figure

Figure 2.22.: Quantiles of account balance as of May 3rd 2014 for addresses with a non-zero balance.

An ending balance of zero, or in other words, the address has been drained, is usually an indicator that this address becomes inactive and is likely not going to be used again. However, addresses with a positive balance could have become inactive as well, especially when the balance is small. We have observed addresses, such as a group of addresses belong to SatoshiDice discussed in Section 2.5.3, that had been used very frequently in some periods of time, and after they had been drained at the end of these very involved time periods, they received small amounts of Bitcoins, such as 1 satoshi, during a couple of receiving activities spanned in a much longer time. Taking into account the fact that a receiving activity is somewhat passive from the address owner's point of view compared to other types of activities, because anyone can just randomly send some amount of Bitcoins to any address, thus the receiving activities of an address are in less control of its owner, and they could even take place

with no intentions from its owner involved. We want to note that one of our speculations about the motives of such behavior is that the sender entity is trying to hide his intention or target of his/her transaction by including some unrelated addresses as receiving addresses in addition to the targeted receiving address. Verification of this speculation is very challenging and it is beyond the scope of this study.

With these considerations in mind, we believe that the last sending related activity of an address is a more reasonable and accurate representation of the address's last usage. Thus we define the life time of an address as the time from its first (receiving related) activity to its last sending related activity, if there is any.

Figure 2.23 shows the distribution of life time (in minutes) of addresses with at least one sending related activity. About 5.81% of all addresses are excluded because they have not conducted any sending related activities. Among the remaining of the addresses, about 15% of them have lifetime as 0 minute, i.e., their sending activities occurred immediately after their receiving activities and both transactions were included in the same block; besides those, another 60% of addresses "lived" shorter than a day and another 20% of addresses were in the range of a day to a month; however, there are also a few long-lived addresses with a lifetime ranges from a few months to several years, and the longest lifetime of an address is over 5 years and it turned out to be one of oldest addresses in a generation transaction being recovered in February 2014.

Link to figure

Figure 2.23.: Quantiles of lifetime of addresses with at least one sending related activity.

2.5.3 The SatoshiDice Addresses and Gambling Services

SatoshiDice [41] is the most popular Bitcoin-based gambling services in the Bitcoin network, and it utilizes the Bitcoin system to place bets, evaluate the bets, and return the payouts of the bets. The SatoshiDice service was launched in April 2012 by Erik Voorhees [49] and remained active as of May 3rd 2014. It is the largest contributor to the transactions in the Bitcoin network, and its addresses are among the most actively used addresses in the entire Bitcoin history.

How dose SatoshiDice work

There are 27 different betting games hosted by SatoshiDice, as listed in Table 2.1. Each of the betting games has a different winning condition, and thus provides a different odds of winning as well as a correspondingly different return rate. Notice that in Table 2.1, although it says "Win Odds", it is actually the probability of winning a bet: the most difficult-to-win game has merely a 0.0015% chance to win, but the return is 64,000 times the bet if ever wins; the least difficult-to-win game has a 97.6563% chance to win, and the return is only 1.004 times the bet if wins; other games have probabilities of winning between these two extremes, and the corresponding return rates also lie in between.

Table 2.1.: SatoshiDice bet options table.

Link to table

To play a betting game on SatoshiDice, or to place a bet, a player simply needs to make a payment transaction using the Bitcoin client programs or online wallet services, and send some amount of Bitcoins, or the bet, to one of the bet addresses listed in Table 2.1. We will refer to these transaction that are used to place bets as bet transactions.

When SatoshiDice receives a bet transaction, it evaluates win or lose and generates a return transaction, which is also a payment transaction and the return of the bet is sent back to the player who places the bet. If the bet wins, the return is the bet multiplied by the prize multiplier in Table 2.1 and that amount is sent back. If the bet loses, the return is the bet times 0.005, resulting in a number much smaller than the bet.

If desired, a player can place multiple bets in a single bet transaction by specifying multiple bet addresses as the receiving addresses. However, each bet will be evaluated and return independently. For each bet, there will be a return transaction and thus a bet transaction including multiple bets will spawn multiple return transactions. In each return transaction, SatoshiDice always includes the corresponding bet it received as one of the (sending address, signature, previous receiving transaction) tuples, and this identifies which bet this return transaction is responding to.

Normally, the return of a bet is sent back to the same address that the player used to place the bet, i.e., the sending address in the bet transaction if there is only one, or one of the sending addresses if the bet transaction includes multiple sending addresses. However, SatoshiDice provides a method to have the return sent to a custom return address specified by the player: if the player adds another receiving address to the bet transaction in addition to the bet address(es), and send exactly 0.00543210 BTC to it, then the return for the bet will be sent to that address instead.

In order to have the return transactions verified quickly by miners, SatoshiDice always includes a transaction fee of 0.0005 BTC in the return transactions. When SatoshiDice sends a return transaction to a player, the transaction fee is subtracted from the return amount. If this makes the return amount zero or less, the return is set to 0.00005430 BTC. Notice that the transaction fee in a return transaction also accelerate the verification of the corresponding bet transaction, because the bet transaction is referenced in the return transaction, and the miners will need to verify the bet transaction before they can verify the return transaction.

The Evolution of SatoshiDice

In April 2012, SatoshiDice launched 24 of its betting games, and later in July 2012, 3 additional betting games were launched. Quickly after its launch, SatoshiDice had become very popular and started to generate a large number of transactions. Overall, there are a total of 11,470,008 transactions involving SatoshiDice, either sent to or from SatoshiDice. That is over 30% of all time transactions in the entire Bitcoin network!

As shown in Figure 2.24 and Figure 2.25, within weeks after being launched, SatoshiDice was responsible for around 16,000 transactions daily, or around 50% of the daily transactions in the Bitcoin network. Up until July 2013, its daily number of transactions and daily fraction of transactions had remained at a very high level during that time, with 8,000–32,000 daily transactions and 40%–60% as a fraction of all Bitcoin transactions.

Link to figure

Figure 2.24.: Daily number of transactions involving SatoshiDice.

Link to figure

Figure 2.25.: Daily fraction of SatoshiDice transactions out of all Bitcoin transactions.

Due to the vast amount of transactions it had generated, SatoshiDice had drawn a lot attention and debate inside and outside of the Bitcoin user community. Many Bitcoin users believed that SatoshiDice was spamming the Bitcoin network, and therefore should be banned; however, others thought of it as a stress test to the Bitcoin

system, and as long as it would pay the transaction fees for its transactions, its usage should be considered valid [50]. Outside the Bitcoin user community, SatoshiDice also faced various issues because of the gambling nature of the service. In May 2013, SatoshiDice decided to block all US-based IP addresses attempting to visit its official website, due to potential legal issues. However, this did not seem to affect its popularity, as the daily number of transactions shown in Figure 2.24 remained at about the same level during that time. After all, once the players who wanted to play the betting games had learned those 27 bet addresses, they could place bets and receive returns, and bypass the website completely.

On July 18th 2013, Erik Voorhees, the founder of SatoshiDice, announced that SatoshiDice had been sold for 126,315 BTC, or US$12.4 million at the time of the announcement [51]. This had raised concerns and caused some insecurity in its user base and the daily number of transactions plunged from over 16,000 to about 8,000. Since then, SatoshiDice had become less popular and the daily number of transactions had gradually decreased to 1,000–2,000 recently. Notice that there was a sizable drop in the daily number of transactions around October 2013 shown in Figure 2.24, and this was because On October 31st 2013 SatoshiDice launched the SatoshiDice Tribute game [52], which claimed to conduct its transactions off the blockchain and thus those transactions were not included in our data. Unfortunately, the new game was shut down a month later after being plagued by slow performance, distributed denial of service attacks and other user related issues.

Bets and Returns

Considering different functionalities, there are three types of transactions involving SatoshiDice, and they make up the total of 11,470,008 SatoshiDice transactions.

The first type are bet transactions, in which players placed bets by sending Bitcoins to one or many of the 27 bet addresses listed in Table 2.1, and there are a total of 5,196,942 bet transactions. Recall that players could place multiple bets in

the same bet transaction by sending Bitcoins to one or multiple bet addresses one or multiple times, there are a total of 6,285,179 bets placed in these bet transactions, and 394,506 bet transactions have multiple bets.

The second type are return transactions, in which the returns of the bets were sent from SatoshiDice to the players that placed the bets, and there are a total of 6,270,147 return transactions. One thing to note is that the number of return transactions is larger than the number of bet transactions, and this is due to the existence of multiple bets in a single bet transaction and each bet resulting in a separate return transaction. Another thing is that the number of return transactions is smaller than the number of bets, and in fact, there are no corresponding return transactions for 15,032 bets. Except for a small number of those bets that were placed towards the end of our data collection and thus the return transactions might have not been included in our data, the majority of such bets with no returns had a value less than 0.0001 BTC. We believe that SatoshiDice simply ignored these bets as the values are much smaller than the the minimum bet they have specified in Table 2.1. Furthermore, these Bitcoins received by SatoshiDice in these bets were not spent as of May 3rd 2014, partially suggesting that SatoshiDice was not trying to trick the players and steal away the Bets.

The remaining 2,917 transaction are the third type, and we will refer them as operational transactions. The usage of these transactions are unknown as they are neither bet transactions or return transactions. Based on manually inspection, most of these transactions occurred around the time SatoshiDice was just launched, and we believe that they were mainly used to to initially set up the pool of funds to back the bets, and some were used to collect and transfer the earnings of the SatoshiDice service.

Overall, combining all 27 bet addresses, the total value of bets that have been placed on SatoshiDice is 4,407,347 BTC, and the total value of returns sent back to players is 4,321,181 BTC. The difference of these two numbers, subtracted by the total value of transaction fees 5,597 BTC paid out by SatoshiDice, gives the earnings

of SatoshiDice a total value of 80,569 BTC as of May 3rd 2014, or 1.83% of total value of bets.

Figure 2.26 shows the number of bets to each of 27 bet addresses. The most popular bet game is the one with winning probability 48.83%, and there are close to 1.5 million bets, or around 25% of all the bets, to this bet address; it is closely followed by 17% of all bets placed on the bet address with winning probability 50.00% and 9.3% on the bet address with winning probability 73.24%. Interestingly, there are far more bets placed on the bet address with the lowest winning probability than any other bet addresses with lower winning probabilities. Gambling, indeed!

Link to figure

Figure 2.26.: Number of bets to each bet address.

According to Table 2.1, there is a minimum on the value of bets for all 27 bet addresses and the minimum is set to be 0.01 BTC; there is also a maximum on the value of bets and it varies across bet addresses. Bets that fall outside of the range are not supposed to be evaluated, and the bets would simply be returned back to the player. However, in practice, we have observed that bets smaller than the specified minimum are often accepted and evaluated by SatoshiDice, suggesting that SatoshiDice did not always respect the specified minimum. The overall quantiles of values of the bets are shown in Figure 2.27, over 90% of values are fairly small, within the range from 0.001 BTC to 1 BTC, and around 26% of bets have exactly the specified minimum value of 0.01 BTC. The largest bet ever placed on SatoshiDice has a value of 717 BTC, which exceeded the specified maximum, and the bet was not evaluated and directly returned to the player that placed the bet.

Link to figure

Figure 2.27.: Quantiles of values of the bets.

As one would have expected, the distributions of values of the bets are different across bet addresses because of the different odds of winning. This is summarized in Figure 2.28, where 10 selected percentiles of the values of the bets are plotted against the odds of winning for each bet address. The lower percentiles across bet addresses are fairly similar and the values are concentrated around 0.01 BTC; the upper percentiles, on the other hand, are generally increasing with the odds of winning, suggesting players are placing larger bets on easier-to-win games and smaller bets on harder-to-win games, which is a reasonable betting strategy.

Link to figure

Figure 2.28.: Selected percentiles of values of bets to each bet address.

Besides its ease of use, another important reason that SatoshiDice is able to generate a large number of transactions is that the betting games are operating with zero confirmation, meaning the bets are evaluated as soon as SatoshiDice picks them up from the Bitcoin network, and the return transactions are usually sent back instantly. Figure 2.29 shows the quantiles of number of confirmations SatoshiDice waited before sending back the returns. Around 65% of return transactions are within the same block as the bet transactions, and over 95% of return transactions are made with less than 3 confirmations.

Link to figure

Figure 2.29.: Quantiles of number of confirmations between bet and return.

Finally, in order to verify that SatoshiDice is hosting the betting games at the odds of winning claimed in Table 2.1, we investigated every bet and return pair and compared the observed odds of winning with the claimed ones for each bet address. As shown in Figure 2.30, for majority of the bet addresses, the observed odds are practically the same as the the claimed ones; for bet addresses with the smallest winning odds, there are deviations between observed odds and claimed ones. However, the deviations are likely due to the low odds of winning and thus a small variation in the number of winning bets to these bet addresses could cause a non-trivial difference in the observed odds of winning.

Link to figure

Figure 2.30.: Observed odds of winning and claimed odds of winning for each bet address.

Usage of Bet Addresses and Non-bet Addresses

We have found that SatoshiDice was in control of at least a total of 1,090 addresses. Besides those 27 bet addresses that SatoshiDice had publicly announced on their official website as listed in Table 2.1, another set of 1,063 addresses are also identified as SatoshiDice addresses, and they will be referred to as non-bet addresses to distinguish them from bet addresses. These non-bet addresses were not publicly announced by SatoshiDice as their addresses, but they are nevertheless identified as

such because they have co-sending relations with those bet addresses and with each other.

SatoshiDice, or more specifically the computer program behind the SatoshiDice services, makes heavy use of both the bet addresses and non-bet addresses to support the operation of the betting games. Because they were programmed to do so, the bet addresses and non-bet addresses are used in very systematic yet distinct ways.

The 27 bet addresses are among the most actively used Bitcoin addresses of all time, and in fact, 9 of them make it to the top 10 list of addresses by frequency of occurrences, and the rest of them are also on the top 130 list. In terms of activity types, the bet addresses have conducted only receiving activities and sending activities, and they have never been used in self-sending activities. Furthermore, in terms of functionality, they are only used in bet transactions and return transactions, but never used in operational transactions. In bet transactions, they are specified as receiving addresses by the players of SatoshiDice in order to place bets; and in bet transactions with multiple bets, multiple bet addresses would be specified as receiving addresses and furthermore, they could even be replicated if the player decided to place multiple bets to the same bet addresses in a single bet transaction. For every bet sent to a bet address, the bet address is programmed to quickly reacts to the bet and generate a return transaction. In return transactions, since SatoshiDice is in control of making the transactions, the bet addresses are used in a more deterministic fashion: exactly one bet address will be used as a sending address in each return transaction so that the return transaction could refer to the corresponding bet that was placed.

The 1,063 non-bet addresses are only used in operational transactions and return transactions, and their functionalities are quite different from those of bet addresses. Besides being used in operational transactions either to set up the pool of funds to get the SatoshiDice service started or to collect and transfer the earnings of SatoshiDice, the non-bet addresses are mainly used in return transactions for two purposes: (1) to provide additional funding, and (2) to collect the change.

The first one is a direct consequence of the fact that SatoshiDice always includes the particular bet address as a sending address in the corresponding return transaction. More specifically, in each return transaction, depending on the result of the bet, none or some non-bet addresses would need to be included as sending addresses in addition to the referenced bet address: if the bet wins, the return value would be larger than the bet, so the tuple of (bet address, signature, bet transaction) alone does not have sufficient funds to return to the player, therefore one or many non-bet addresses would be pooled to fund the return transaction; in fact, even if the bet loses, non-bet addresses were sometimes still pooled in the return transaction to cover the transaction fees. Figure 2.31 shows the quantiles of number of sending addresses in return transactions for each bet address. As the winning probability decreases, the fraction of return transactions that include a single-sending address, i.e., only the bet address as sending address, increases from less than 10% to more than 60%. This is because higher winning probabilities lead to higher fraction of winning bets, and consequently result in higher fraction of return transactions in need of additional funding. Another interesting thing in Figure 2.31 is that, besides the obvious favor of one bet address and one non-bet address combination as the sending addresses to provide funding in return transactions for winning bets, the combination of one bet address and three non-bet addresses also occurred very frequently, taking up around 20% of return transactions for each bet address.

Link to figure

Figure 2.31.: Quantiles of number of sending addresses in return transactions for each bet address. Bet addresses are ordered by winning probability.

The second usage of non-bet addresses can be attributed to the existence of change in any Bitcoin transactions, in this case the return transactions. In each return transaction, regardless of win or lose, some amount of Bitcoins is returned to the player's

address, and the change of this transaction, if there is any, is then collected by non-bet addresses. As shown in Figure 2.32, there are only three possible values of the number of receiving addresses in return transactions: 1, 2, and 3. The value 2 dominates in return transactions across all bet addresses, meaning in most of the return transactions, one non-bet address was specified as a receiving address in addition to the player's address to collect the change; there are very small number of return transactions where the player's address was the only receiving address, suggesting there was no change to be collected; there are also a small number of return transactions where two non-bet addresses were specified as receiving addresses, and it turns out that the values of change in these return transactions were quite large, and SatoshiDice were breaking the change into two small pieces for further usage.

Link to figure

Figure 2.32.: Quantiles of number of receiving addresses in return transactions for each bet address. Bet addresses are ordered by winning probability.

These non-bet addresses, despite not publicly announced, were also used very frequently as 50 of them make it to the top 80 list of Bitcoin addresses by frequency of occurrences. Figure 2.33 shows the quantiles of number of activities/transactions of the non-bet addresses, and three groups of non-bet addresses clearly stand out and separate themselves from each other and the rest of the non-bet addresses: group 1 consists of 25 addresses and each was involved in 395,000 activities; group 2 consists of another 25 addresses and each was involved in 84,000 activities; group 3 consists of 1,001 addresses and each was involved in a range of 100–200 activities; and the rest of 12 addresses all have less than 60 activities.

Link to figure

Figure 2.33.: Quantiles of number of transactions of non-bet addresses.

The first three groups of non-bet addresses were mainly used in return transactions to provide additional funding and collect the change, but each group were active in a different period of time. Addresses in group 3 came out first during the first few weeks when SatoshiDice was launched, and by the end of May 2012, these addresses were drained, i.e., all Bitcoins ever received were transferred away, and they had become inactive ever since. Addresses in group 2 became active in May 2012 and they replaced group 3 to support the SatoshiDice betting games. In September 2012, addresses of group 2 were also drained and they were replaced by the addresses in group 1, which remained active as of May 3rd 2014. In fact, the addresses in group 2 did not disappear completed from the Bitcoin network. We have observed a few transactions where addresses in group 2 are specified as one of many receiving addresses, but the receiving values are quite small and they have never been spent by SatoshiDice, therefore, we believe that SatoshiDice is not aware of these transactions and these addresses had indeed become inactive.

Finally, after investigating the properties, we found out that these non-bet addresses were used in a very similar fashion, if not identical, and we believe that their migration is simply the result of SatoshiDice upgrading its underlying program that operates the betting games.

Address/Entity Grouping in Bet Transactions

Thanks to the unique functionalities of the bet transactions to SatoshiDice, we have identified two facts that could be exploited to further the grouping of addresses

into entities beyond the grouping based on co-sending relations as described in Section 2.2.3.

Firstly, since a player can specify a custom return address to receive the return of a bet, it is reasonable to believe that this custom return address belongs to this player. In order to identify the custom return addresses, we search for receiving addresses in bet transaction that satisfies both of the following two conditions: (1) this address must not be one of the bet addresses; (2) this address is receiving exactly 0.00543210 BTC. Note that we have found a small number of bet transactions where multiple receiving addresses satisfy both conditions, while SatoshiDice had treated whichever address with the largest index, i.e., specified the last among receiving addresses, as the custom return address to send the return to, we will treat them all as the custom return address, assuming that the players are simply experimenting to find out the behavior of SatoshiDice with respect to custom return addresses. We have thus identified a total of 56,125 bet transactions where at least one custom return address is present.

Secondly, recall the widely existence of change addresses in payment transactions, there should also be change addresses in bet transactions, and the change addresses should belong to the players as well. We argue that that in bet transactions, the bet addresses are not receiving the change, because they are naturally receiving the payments, in this case the bets, and they are programmed by SatoshiDice to react to the bets by sending the returns in return transactions. Thus this leaves one of the rest of receiving addresses, if there is any, to be the change address. Therefore, in a bet transaction, if there is only one receiving address that is neither a bet address nor a custom return address, then we treat this receiving address as the change address. We have found a total of 5,146,202 bet transactions where the change address can be identified as such.

Each one of the above two strategies would lead to link up a pair of co-sending relation based entities, one corresponding to the player entity that placed the bet, and the other corresponding to the entity that the change/custom return address

belongs to. Overall, combining both strategies, we have identified 441,736 unique pairs of entities that we believe could be further grouped into same entities. These pairs consist of a total of 344,070 entities, and a union-finding algorithm merges them into merely 14,557 unique entities, a 24:1 ratio of further grouping.

Notice that while the first strategy is somewhat unique in the context of bet transactions to SatoshiDice, the second strategy, on the other hand, is quite general and can be extended and applied to many other transactions involving other services. This is further discussed in Section 2.6.

Other Bitcoin-based Gambling Services

The early success of SatoshiDice has inspired a few other Bitcoin-based gambling services. These services are not as popular as SatoshiDice, nevertheless, they have contributed a noticeable amount of transactions in the Bitcoin network, and many of their addresses have easily made to the top 100 list of addresses by frequency of occurrences.

A few of the notable gambling services are: the BetCoin Dice [53], established in July 2013 and responsible for 1,137,443 transactions; the SatoshiBones [54], established in September 2013 and responsible for 565,606 transactions; and the Lucky-Bit [55], established in October 2013 and responsible for 308,620 transactions. Jointly, they have contributed 5.29% of all time transactions, in addition to SatoshiDice's 30.19%. And all of these services remained active as of May 3rd 2014 and continued to generate a non-trivial number of transactions on a daily basis.

These gambling services provide variations of betting games similar to the ones provided by SatoshiDice and they also operate in a similar way as SatoshiDice operates in terms of handling bets and returns, more specifically, they also make use of bet addresses to receive and react to bets and non-bet addresses to provide additional funding and to collect change, etc. The same analysis we have performed for

SatoshiDice could readily be applied to these services as well and we will skip the detailed investigation here.

2.5.4 The DeepBit Addresses and Similar Addresses

DeepBit [56] is one of the largest and oldest mining pools in the Bitcoin network, and it was established on February 24th 2011 and remained active, although barely, as of May 3rd 2014.

DeepBit had not revealed its addresses to the public, but one particular address had been identified by the Bitcoin user community as one of DeepBit's addresses: 1VayNert3x1KzbpzMGt2qdqrAThiRovi8. We will refer to this address as the DeepBit distribution address, because it was mainly used, or more specifically it was programmed, to distribute the block rewards DeepBit had earned via the pooled mining to its participants. This address have been used very heavily, and its usage presents a typical usage pattern shared by many other addresses.

Transactions involving the DeepBit Distribution Address

The DeepBit distribution address is involved in a total of 786,501 transactions, or 2.07% of all time Bitcoin transactions, and it is ranked the 5th among all Bitcoin addresses by frequency of occurrences, the only one on the top 10 list that does not belong to SatoshiDice. Figure 2.34 shows the daily number of transactions involving the DeepBit distribution address. In November 2011, the DeepBit distribution address became active and started to generate 1,000–2,000 transactions on a daily basis; starting from late 2012, however, due to the growing competition from newly joined miners and mining pools equipped with various advanced mining technology, DeepBit was not able to generate as many blocks as it was able to in the previous year, and the daily number of transactions involving the distribution address had dropped to 500–1,000 in the following six months; notably, after the establishment of GHash.IO [57] in July 2013, which later became the largest mining pool in the

Bitcoin network, the market share of DeepBit continued to shrink and many of its participants left DeepBit for other mining pools, and the DeepBit distribution address was generating merely a few transactions a day in recent months.

Link to figure

Figure 2.34.: Daily number of transaction involving the DeepBit distribution address.

Interestingly, despite being the address of one of most popular mining pools, the DeepBit distribution address has never conducted any generation activity, i.e., it has never directly received block rewards from the Coinbase in generation transactions. Instead, some other temporary addresses, which we believe should also belong to DeepBit, were used in generation transactions to receive the block rewards; these addresses were later used in payment transactions, alone or jointly, to send or merge the rewards to the DeepBit distribution address, and thereafter these addresses were never used again; finally, the DeepBit distribution address distributed the rewards in a series of following transactions to many different addresses, which we believe belong to the participants of the DeepBit mining pool.

We have identified a total of 17,171 (out of 298,851) generation transactions, where a single temporary address other than the DeepBit distribution address was receiving the block rewards, and then sent all of them to the DeepBit distribution address in a later transaction. Furthermore, we have also found a small number of generation transactions, where the temporary addresses receiving the block rewards sent a fraction of the rewards to the DeepBit distribution address in later transactions. Overall, the total value of block rewards received indirectly by the DeepBit distribution address is 810,723.7 BTC. Notice that the DeepBit distribution address had not been used by DeepBit until November 2011, about 8 months after the establishment of the

DeepBit mining pool, the above amount only provides a lower bound of how much block rewards DeepBit and its participants has earned.

Properties of the DeepBit Distribution Address

Because of the particular functionality as a method of distributing relatively large amount of Bitcoins to many receiving entities, the usage of the DeepBit distribution address presents a typical pattern, characterized by its types of activities, the specifications of sending and receiving addresses when conducting sending related activities, and the confirmations needed for it to spend received Bitcoins when conducting receiving related activities.

First of all, the DeepBit distribution address was very frequently used in self-sending activities, where this address appeared both as a sending address and a receiving address in the same transaction. There are 768,089 self-sending activities, taking up to 97.66% of all its activities, 14,293 receiving activities, or 1.82%, and 4,119 sending activities, or 0.52%. The activities of the DeepBit distribution address, including sending, receiving, and self-sending activities, are shown in Figure 2.35, where different types of activities are colored differently and plotted as vertical lines in the order of time of occurrence; the vertical lines originate from 0, and the length of the lines corresponds to the amount of Bitcoins of this address that are involved during these activities, and the direction of the lines corresponds to the direction of the change of account balance of this address, i.e., lines pointing up when this address's balance is going up and pointing down when balance is going down.

Link to figure

Figure 2.35.: Activities of the DeepBit distribution address.

It is easy to see in Figure 2.35 that the activities of the DeepBit distribution address consist of many cycles of similar series of activities: a series usually starts with a few receiving activities, where this address received some relatively large amount of Bitcoins (usually 25 BTC or larger), followed by a much larger number of self-sending activities and a couple of sending activities, where relatively smaller pieces of Bitcoins (usually within the range from 0.01 BTC to 1 BTC) were sent away from this address.

Secondly, certain specifications of sending and receiving addresses dominate the self-sending and sending activities of the DeepBit distribution address. In fact, there are 759,585 among all 768,089 self-sending activities where one sending address and two receiving addresses were specified in a transaction; and there are 4,054 among all 4,119 sending activities where one sending address and one receiving address were specified. Therefore, in most cases, when the DeepBit distribution address was self-sending, it was the only sending address and it was also receiving the change while sending some amount of Bitcoins to another receiving address; when the DeepBit distribution address was sending, which usually happened after a series of self-sending activities, it simply sent all the left-over Bitcoins to a single receiving address.

Thirdly, as shown in Figure 2.36, the number of confirmations the DeepBit distribution address waited before spending received Bitcoins are generally small, yet the distribution varies in different types of activities. When the DeepBit distribution address was self-sending, because the Bitcoins it received were the change of this transaction and the validity was implied, they were usually spent very quickly: over 60% of them were spent without any confirmation, and over 90% were spent with 6 or fewer confirmations; on the other hand, when this address was receiving, most likely receiving block rewards from those temporary addresses, the Bitcoins received were spent not as fast, with the corresponding 60% percentile being 5 confirmations and 90% percentile being 9 confirmations. Given the fast response of the DeepBit distribution address in receiving related activities, it is likely that this address is handled by a computer program to operate in such a way, especially during self-sending activities.

Link to figure

Figure 2.36.: Quantiles of number of confirmations the DeepBit distribution address waited before spending received Bitcoins.

In summary, the majority of activities of the DeepBit distribution address consist of many similar series of activities: one receiving activity triggers a long chain of self-sending activities, and the chain is then concluded with a sending activity. During each series of activities, a large bulk of Bitcoins received in the receiving activity is gradually and quickly distributed to a number of receiving addresses, one receiving address at a time. Therefore, the DeepBit distribution address presents the following properties: (1) self-sending activities dominate the activities of this address, accounting for 97.66% of all activities; (2) in most of its self-sending activities, 98.89% of all self-sending activities, it is the sole sending address and it is also collecting the change, while another receiving address is receiving the payment; (3) between consecutive self-sendings activities, the number of confirmations is usually small.

Other Distribution Addresses

Powered by the address based databases, we are able to identify many other addresses that share similar properties as the DeepBit distribution address: these addresses are mainly conducting self-sending activities, and during these chains of self-sending activities, these addresses send some amount of Bitcoins to many other addresses, with one address at a time. All these addresses are, in some sense, "distributing" relatively larger bulk of Bitcoins into many other addresses.

A couple of notable examples of addresses sharing similar properties are: address 1BTC24yVKQdQNAa4vX71xLUC5A8Za7Rr71 that belongs to Bitcoin-24 [58], one of the largest Bitcoin exchange site in Europe, and it is mainly used to send to its

users' addresses whenever they purchase Bitcoins with local currencies; and address 13x9weHkPTFL2TogQJz7LbpEsvpQJ1dxfa that belongs to Reddit Bitcoin Faucet [59], a faucet service started by a user of www.reddit.com [60], and the address is used to give away tiny fractions of Bitcoins for free to many other users of www.reddit.com for the purpose of promoting the Bitcoins system.

Note that this is certainly not the only way of distributing Bitcoins, and we have found other addresses with variations of the distribution behavior: these addresses are also mainly conducting self-sending activities, however, instead of sending to one other address at a time, many receiving addresses could be specified and thus reduce the length of the chains of the self-sending activities. A couple of examples of such addresses are: address 15ArtCgi3wmpQAAfYx4riaFmo4prJA4VsK that belongs to Bitcoin Faucet [61], another faucet service that used to give away free Bitcoins for the same purpose of promoting the Bitcoin system from early 2012 to early 2013; address 1cointQVgw2EwnJx3EFVPvD65gSsD9nJ7 and related addresses that belong to CoinBox.me [62], a service that enables its users to make micro Bitcoin transactions while avoiding the transaction fees by merging multiple micro transactions into larger transactions.

2.5.5 The WikiLeaks Addresses and Similar Addresses

On June 14th 2011, WikiLeaks announced on Twitter that they started accepting donations in Bitcoin at the address 1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v [63], which we will refer to as the WikiLeaks static address.

The WikiLeaks static address has been involved in a total of 2,394 transactions, several orders of magnitude smaller than the SatoshiDice addresses and the DeepBit distribution address. Nevertheless, the WikiLeaks static address presents another typical usage pattern, which is shared by many other addresses. More importantly, rather than some computer programs behind the SatoshiDice addresses and the DeepBit distribution address, our analysis suggests that the WikiLeaks static address is

handled by actual human users, which make the analysis more applicable to general users of the Bitcoin system.

First of all, the activities of the WikiLeaks static address are dominated by receiving activities, which accounts for 2,286 out of 2,394 transactions, and there are only a few sending activities along with very few self-sending activities. As shown in Figure 2.37, a pattern of a long series of receiving activities followed by a few sending activities emerges during the course of activities of this address. And the receiving values are usually relatively small compared with the sending values.

Link to figure

Figure 2.37.: Activities of the WikiLeaks static address.

Secondly, these receiving activities of the WikiLeaks static address are contributed by a relatively large number of sender entities, i.e., this address has received Bitcoins from many different entities. In fact, there are a total of 2,018 entities sending to this address in those 2,286 receiving activities. As shown in Figure 2.38, the majority of the sender entities have sent to the WikiLeaks static address only once and no particular sender entity stands out.

Link to figure

Figure 2.38.: Quantiles of number of transactions from each sender entity to the WikiLeaks static address.

Thirdly, when the WikiLeaks static address is conducting sending activities, it usually pools multiple funding sources in the same transaction. Besides including other sending addresses, the WikiLeaks static address is often specified multiple times

as sending addresses in the same transaction. This is demonstrated in the left panel of Figure 2.39, as the WikiLeaks static address is replicated at least once in the majority of its sending activities, and sometime it is replicated 10s of times.

Link to figure

Figure 2.39.: Quantiles of number of replications the WikiLeaks static address is specified as a sending address in each transaction.

In summary, we have learned that the WikiLeaks static address has received small amounts of Bitcoins from many different entities in many receiving activities, and these Bitcoins received are often merged into larger pieces and sent away in following sending activities. Considering the functionality of the WikiLeaks static address, which is accepting donations from donors, it is very natural that this usage pattern emerges from this address: various donors make donations by sending Bitcoins to the WikiLeaks static address, and these donations are then merged and transferred to other addresses, likely to those exchange services where the Bitcoins are converted into local currencies.

As of May 3rd 2014, the WikiLeaks static address has received a total of 3,857 BTC as donations. However, these do not include every donation they have received via the Bitcoin system. WikiLeaks have introduced a mechanism for donors to make donations to newly generated addresses on their website [64] instead of the static address, in order to better protect the donors' privacy. These temporary addresses are usually used twice, one receiving activity and then one sending activity, and many of them have been used in the same transactions along with the WikiLeaks static address to merge the donations.

Finally, we have found that there are many other addresses being used in a similar way as the WikiLeaks static address: they have been used to receive small amounts of Bitcoins from many different entities in many receiving activities, and these Bitcoins

received have been merged into larger pieces and spent in following sending activities. We will refer to these addresses as static receiving addresses, and we believe that they are likely the "face" addresses of some individuals or organizations, and they remain static so that the sender entities could make payments, in the context of WikiLeaks, make donations, to those individuals or organizations on an ad hoc basis.

It is important to note that these static receiving addresses are unlikely to be used as the change address in a transaction because they are usually the natural payment addresses. In Section 2.6.2, we will exploit this fact for the purpose of the identification of payment and change in a transaction, and an empirical definition of static receiving address is also provided.

2.6 Modeling: Identification of Payment and Change

As discussed in Section 2.2.3, a change address almost always exists in a payment transaction to collect the change for the sender, however, it does not distinguish itself from other receiving address(es), or the payment address(es). Thus, in a payment transaction with more than one receiving addresses, the addresses receiving the payments are unknown to the outsider of the transaction, neither are the payment values being transferred. This greatly complicates the flow of Bitcoins and makes it very difficult to understand the transactions between entities.

Thus, the identification of payment address and change address become very important to understand the actual usage of the Bitcoin system. For one thing, being able to determine which receivings are the payments and which is the change answers two questions that are trivial in other currency systems, yet quite difficult in the context of the Bitcoin system, to whom the payment is sent and how much is being transferred. Furthermore, in the sense of anonymity, the change address identified can be associated with the sender entity, and thus further improve the understanding of ownership and relationship of addresses beyond the co-sending relations based grouping into entities discussed in Section 2.2.3.

The identification of payment address and change address is certainly very important and rewarding, yet it comes with great challenges as well. The challenges lie not only in coming up with heuristics or models for identification, but also the validation of such heuristics/methods, simply because the lack of truth known to the outsiders of payment transactions.

In certain transactions, the notion of change does not apply and thus it is trivial to identify the payments in such transactions. These include 298,851 generation transactions, in which miner's addresses receive the block rewards, and 2,371,902 one-receiving transactions, which are payment transactions with a single receiving address: in generation transactions, there is no change address and all receiving addresses are receiving the payments, or more specifically, the block rewards; in one-receiving transactions, obviously the single receiving address is the payment address and there is no change address involved either. It is entirely possible that a sender could simply be sending some amount of Bitcoins to his/her own address in a one-receiving transaction, nevertheless, we can still think of it as a payment, only that the payment is made to the sender himself/herself.

For all other payment transactions, there are two or more receiving addresses. We argue that at least one of these receiving addresses is receiving the payment and at least one other is collecting the change. In theory, it is possible that all of the receiving addresses are receiving payments, but in reality, the probability of such events should be very very low for the following reasons. First, it is very unlikely that the combined sending value in the sending addresses happens to be exactly the same as the targeted payments, so the probability of not having a change in a transaction is very low; second, given that there exists the change, a rational sender is very unlikely to give away the change by not specifying a change address.

For simplicity, we will only target on two-receiving transactions, i.e., payment transactions with exactly two receiving addresses, and we exclude all three-or-more-receiving payment transactions in this study. This simplifies the task to be the identification of which one of the two receiving addresses is the payment address, or

equivalently, which one is the change address. Recall that around 87% of all payment transactions are two-receiving transactions and less than 7% of payment transactions are three-or-more-receiving transactions, the majority of the transactions are well covered.

In the following subsections, we first identify a collection of two-receiving transactions where the payment and change can be determined easily with a high confidence, we further identify features that could be used to characterize the properties of payments or changes in this collection of transactions, and finally we propose a classification model for the identification of payment and change in two-receiving transactions. We will see that, for three-or-more-receiving transactions, the proposed model can also be applied with some modifications.

2.6.1 Partition of Two-receiving Transactions

In general, the payment and change of a two-receiving transaction is unknown to an outsider of the transaction, however, there are certain scenarios where this information is implied. Recall that the change address is used to collect the change for the sender entity, thus intuitively the change address should belong to the sender. Therefore, in a two-receiving transaction, if one receiving address is known to be owned by the sender while the other is not, then the former address is more likely to be the change address than the later. We have identified two scenarios in which we could determine whether or not a receiving address belongs to the sender, and thus we could identify two corresponding subsets of two-receiving transactions where the payment and change are known with a high confidence.

The first set are those transactions where one and only one receiving address is the same as the sending address, or is the same as one of the sending addresses if there are multiple sending addresses in this transaction. It is obvious this receiving address belongs to the sender and thus we consider it to be collecting the change. We will refer to these transactions as same-address transactions.

The second set are those transactions where one and only one receiving address is known to belong to the same entity as the sender via the co-sending relations based grouping, and thus it is also owned by the sender and collecting the change. We will refer to these transactions as same-entity transactions.

Besides the same-address transactions and same-user transactions, there are a relatively small number of two-receiving transactions, or 112,274 transactions to be exact, where both the receiving addresses are known to be owned by the sender, either both are identical to the sending addresses, or both can be grouped into the sender entity based on co-sending relations. In such cases, the sender is simply breaking larger bulk of Bitcoins into two smaller pieces for future use in these transactions. We could still reasonably impose a notation of payment and change in such transactions by claiming that the receiving that is spent first to be the payment and the other to be the change. However, since these transaction make up a very small fraction of all the transactions, we decide to simply exclude them from this model.

Furthermore, we also excluded a few notable groups of transactions: the bet transactions to SatoshiDice's bet addresses, the return transactions from SatoshiDice's bet addresses, and the distribution transactions from DeepBit's distribution address. There are mainly two reasons for the exclusion of these transactions: (1) as demonstrated in Section 2.5.3 and Section 2.5.4, it is fairly easy to determine the payment and change in these transactions and thus there is no need to include them in the model; (2) these sets of transactions make up a significant proportion of all the two-receiving transaction, and we do not want to build a model that is tailored to these transactions.

Finally, we have identified 4,801,793 same-address transactions and 3,169,716 same-entity transactions, where the payment and change are implied. Combining these two sets of transactions, we have a total of 7,971,509 two-receiving transactions, and these transactions will serve as the model building set from which we would identify features that characterize the properties of payment and change, and build and validate a model to classify payment and change. In the remaining 12,915,420

two-receiving transactions, neither of the receiving addresses is known to be owned by the sender, and the payment and change are to be determined with the help of the model. It is important to note that, despite the lack of truth unknown to outsiders of payment transactions, we were able to identify those transactions where the truth is implied, and they could be used to validate or evaluate other heuristics or methods attempting to identify payment and change such as ours.

2.6.2 Feature Identification

After a detailed investigation of the model building set of transactions defined above, we identified a selection of features that characterize the properties of payment and change. These features are very intuitive and they include features derived from the receiving values, features derived from the profile of receiving addresses, and features describing usage of the receivings, etc.

Complexity of Receiving Values

The complexity of a receiving value is computed in the following way: the receiving value, in the unit of BTC, is converted to the unit of satoshi by multiplying the value by 10^8, this effectively converts the receiving value into integers because satoshi is the smallest unit of Bitcoin; then the leading and trailing 0's in the string of the value are removed; and finally, the complexity of the value is the number of digits in the resulting string. For example, a receiving value of 0.099 BTC will be converted to 9900000 satoshi, then to "99" after removing the trailing 0's, resulting in a complexity of 2 as there are two digits in the resulting string.

The complexity of a receiving value measures how "random" the value string is, and we argue that in a payment transaction, the complexity of the payment value is likely not higher than that of the change value. The payment value, which usually reflects the price of goods or services, are often determined in a systematic way and tend to be less complex values such as 10 BTC or 0.099 BTC, with a complexity of

1 and 2, respectively. On the other hand, the change value is the combined sending values subtracted by the payment value and the transaction fee, which is, while not zero, usually a small fix amount such as 0.0005 BTC or a fix proportion of the payment value such as 5%. Thus the complexity of the change value depends on not only the complexity of the combined sending values, but also the complexity of the payment value, and possibly the transaction fee. If the combined sending values have a low complexity and there is no transaction fee, then the complexity of the change value should be similar to that of the payment value; if the combined sending value has a high complexity or there is some amount of transaction fee, then the complexity of the change value is likely to be higher than that of the payment value. For example, given two receiving values being 0.99 BTC and 0.3757798 BTC in the same payment transaction, the former is more likely to be the payment and the later is more likely to be the change.

Therefore, a simple rule to determine the payment and change is to compute the complexities of both receiving values in the same transaction, and claim the receiving value with smaller complexity to be the payment and the other to be the change. Figure 2.40 shows the distribution, in terms of fraction of transactions, of the absolute value of difference between the complexities of two receiving values in the model building set of transactions. In around 13% of the transactions in the model building set, the two receiving values have the same complexity; in the rest 87% of transactions, there is a non-zero difference and the absolute value of the difference varies from 1 to 11, with more transactions having smaller differences and less transactions having larger differences.

Link to figure

Figure 2.40.: Fraction of transactions in the model building set against absolute value of difference between the complexities of two receiving values.

We apply the above rule to those transactions in the model building set when there is a non-zero difference between the complexities of two receiving values, and the results are shown in Figure 2.41. The fraction of transactions in which the receiving value with a lower complexity is the payment is plotted with the absolute value of difference between the complexities of two receiving values. The fractions are all greater than 0.5, suggesting that the payment values are more likely to be less complex one; and as the absolute value of difference of complexities increases, the fraction increases from around 0.6 to over 0.95, meaning the less complex value are more and more likely to be the payment value when the difference between complexities becomes larger.

Link to figure

Figure 2.41.: Fraction of transactions where the receiving value with smaller complexity is the payment.

Therefore, to incorporate the information about complexities of receiving values when determine payment and change, we consider the difference between the complexities of two receiving values in the same transaction as a feature and include this variable in the upcoming model.

Multi-sending Transactions

Multi-sending transactions are those payment transactions in which there are two or more sending addresses. In these transactions, the Bitcoins in the sending addresses are pooled together to provide funding for the payment. Intuitively, a multi-sending transaction should only occur when the payment value is relatively large and the sending values in multiple sending addresses have to be combined in order to provide sufficient funding. Thus, strictly speaking, in these multi-sending transactions, the

payment value should be larger than the combined values of every combination of the sending addresses except the one combination which includes all of the sending addresses; and the change value, on the other hand, should be smaller than each of the sending values, and consequently smaller than the payment value.

In practice, however, this is not always the case. Figure 2.42 gives a hexagon binning version of the scatter plot of the payment value against the change value for each transaction in the model building set, conditioning on the number of sending addresses in this transaction. On the first page which corresponds to single-sending transactions, the points are scattered on both sides of the $y = x$ reference line and no pattern stands out. On all other pages, which correspond to multi-sending transactions, there are points on the top left side of the $y = x$ reference line as well and the payment values are not always greater than the change values. However, we should note that the the payment values are generally more likely to be larger than the change values in multi-sending transactions, especially as the number of sending addresses increases.

Link to figure

Figure 2.42.: Payment value against change value in model building set of transactions, conditioning on number of sending addresses.

Furthermore, we have observed that senders would sometimes include sending addresses bringing various small amount of Bitcoins, such as 1 satoshi, into multi-sending transactions, and we suspect that these senders are merging their (small) funds while making a payment at the same time. Therefore, in order to take advantage of the general properties of multi-sending transactions while taking into account these unusual behaviors, we decide to loose the restriction by simply checking if a receiving values is larger than each of the sending values. Another simple rule to determine payment and change is to see if one of the receiving values is larger than each of the

sending values and the other is not, and we would claim the former is the payment and the later is the change.

In the model building set of transactions, there are a total of 3,415,525 multi-sending transactions, and in 2,458,966 of them the above rule applies, i.e., one of the receiving values is larger than all individual sending values while the other is not. After applying the above rule, we have correctly identified the payment in 92.82% of those 2,458,966 transactions.

Therefore, to make use of the information regarding the receiving values in multi-sending transactions, we also include variables that indicate whether or not each receiving value is larger than all individual sending values in the upcoming model.

Shadow Address

As introduced in [27] and further exploited in [28], there are addresses, referred to as shadow addresses, that are automatically generated by the Bitcoin client programs or the online wallet services to collect the change for the sender each time the sender is making a payment.

The identification of shadow address would directly lead to the identification of change address, however, the truth about which address is a shadow address is unknown. Empirically, an informative guess can be made based on the properties of these shadow addresses: because a new shadow address is generated every time there is a change to be collected and the sender, or effectively the owner of the shadow address, is usually not informed by the client program or online wallet services of this address and thus is not aware of its existence, the sender is likely not going to use this address to receive any Bitcoins again. Therefore, these shadow addresses are usually used to receive once and will likely never be used to receive again.

Based on these properties, various definitions of shadow address have been proposed in [27] and [28]. Similarly in this study, we define a shadow address to be an address that satisfies the following conditions. It is important to note that these

definitions are usually more aggressive than we would want them to be, because these properties are shared by addresses other than shadow addresses. For example, a very careful Bitcoin user would generate a new address for each receiving activity, and these addresses would have been label as shadow address based on these definitions as they are also used to receive only once, but not for collecting the change. Nevertheless, these aggressive definitions have proved to be useful in other studies and is also useful in ours to determine payment and change, and we decide to take advantage of them anyway.

- This address has conducted only one receiving activity.
- This address was not receiving in a generation transaction.
- This address was not the only receiving address.
- This address was not receiving in a same-address transaction.
- This address was not receiving multiple times in the same transaction.
- If there are more than one address satisfying the above conditions, none of them are shadow addresses.

Through this definition, we have identified a total of 12,249,619 shadow addresses. In a two-receiving transaction, if one of the two receiving addresses is a shadow address and the other is not, then the shadow address is likely to be the change address and the other address is like to be the payment address. In the model building set, there are 1,485,994 transactions where this condition holds, and in 95% of them, the shadow address is indeed receiving the change.

Thus, we include additional variables that indicate whether or not each receiving address is a shadow address in the upcoming model.

Static Receiving Address

As introduced in Section 2.5.5, static receiving addresses, such as the WikiLeaks static address, are those addresses that have been used to receive Bitcoins from many different entities in many receiving activities, and then those Bitcoins received are often merged into larger pieces and are spent in following sending activities of those addresses.

We believe that many of these static receiving addresses are the "face" addresses of certain individuals or organizations, and they are used to receive payments, or donations in cases similar to WikiLeaks, from various sender entities. Some of these addresses are often publicly announced in some way, either on their websites or in various online forums, and they remain static and active to conduct receiving activities because it is more convenient for different sender entities to make payments, sometimes repeatedly, to the same address; other static receiving addresses are often the "favorite" addresses of individual users, i.e., many users did not bother generating new addresses when they receive Bitcoins from others and kept using the same addresses again and again for simplicity and their own convenience.

Loosely speaking, addresses hosting specific services, such as the 27 SatoshiDice bet addresses and the bet addresses of other gambling services mentioned in Section 2.5.3, are also static receiving addresses. These addresses are receiving Bitcoins from many other sender entities, and they have to remain static so that they can continue to provide particular services, betting games in the case of SatoshiDice and other gambling services, for its users. The only difference between these service addresses and the "face" addresses mentioned above is how the received Bitcoins are spent: while "face" addresses tend to merge small funds into larger pieces when spending, the service addresses would behave in a more deterministic and systematic way as they are programmed to do so, e.g., those bet addresses would react to each bet and spend the received Bitcoins in the return transaction rather than merging funds.

Naturally, these static receiving addresses are likely to be used to receive payments instead of changes due to its particular functionalities.

In this study, we define a static receiving address as following:

- This address has no less than 10 receiving activities.
- This address has no less than 10 sending related activities, including sending activities and self-sending activities.
- This address rarely conducts self-sending activities, more specifically less than 20% of sending related activities are self-sending activities
- When this address conducts self-sending activities, it is the only receiving address in the transactions, i.e., this address is merging funds into itself rather than collecting the change in its self-sending activities.
- At least 20% of the sender entities has sent to this address only once.

We have identified a total of 167,103 static receiving addresses by this definition. In a two-receiving transaction, if one of the two receiving addresses is a static receiving address and the other is not, then the static receiving address is likely to be the payment address and the other is the change address. In the model building set, there are 1,958,085 transactions where this condition holds, and in almost all of them, 99.78% to be exact, the static receiving address is receiving the payment.

Therefore, we include additional variables that indicate whether or not each receiving address is a static receiving address in the upcoming model.

2.6.3 The Classification Model

From the previous subsection, we have identified 4 features that characterize various properties of payment and change in two-receiving transactions. In the model, we decide to represent the features in the form of the following predictor variables describ-

ing the two receiving values/addresses, as a trade-off between simplicity/efficiency of storage and representability of the features.

complexity: The difference between the complexities of two receiving values in a two-receiving transaction. The difference is calculated as the complexity of the second receiving value subtracted by that of the first receiving value. This variable takes integer values from -11 to 11.

multi.sending: Whether or not neither, either, or both of the two receiving values satisfy the condition that the value is greater than each of the sending values in a two-receiving transaction. This is a categorical variable and it is represented as an integer value from 0 to 3, with 0 says neither of the two receiving values satisfy the condition, 1 says the first one satisfies, 2 says the second one satisfies, and 3 says both of the two satisfy. Notice that in a single-sending transaction, neither of the two receiving values could be greater than the single sending value, so this variable would take the value of 0, which is the same value this variable would take when neither of the two receiving values are greater than each of the sending values in multi-sending transactions. In both cases, the information regarding the relative size of receiving values compared with sending values is not going to be of much help in determining the payment and change anyway, so we decide to merge these two cases into the same category of this variable, instead of introducing an additional indicator variable for multi-sending/single-sending transactions.

shadow: Whether or not neither, or either of the two receiving addresses is a shadow address. This is also a categorical variable and it is represented as an integer value from 0 to 2, with 0 says neither of the two receiving addresses are shadow addresses, 1 says the first one is, and 2 says the second one is. Based on our definition of shadow addresses, there is at most one shadow address in a transaction, so it is impossible to have both receiving addresses as shadow addresses.

static: Whether or not neither, either, or both of the two receiving addresses is a static receiving address. This is another categorical variable and it is represented as an integer value from 0 to 3, with 0 says neither of the two receiving addresses are static receiving addresses, 1 says the first one is, and 2 says the second one is.

For each two-receiving transaction, the response variable y, or the class label, takes one of two values, "Payment" and "Change", depending on the status of the first receiving address: y = Payment if the first receiving address is receiving the payment, and y = Change if it is collecting the change.

With the help of the transaction based database described in Section 2.4.1, the extraction of the model building set of two-receiving transactions from all the transactions has become very easy to program and efficient to run, so are the computations of predictor variables and response variables in those transactions. This is achieved in a single MapReduce job and the database is scanned once. In this MapReduce job, we computed the 4 predictor variables and the response variable for each transaction in the model building set, and there are a total of 7,971,509 observations/transactions. Now that the data are small enough, we could read them into memory and store them in an R data frame of 7,971,509 rows and 5 columns for more interactive analysis and modelling.

Notice that most of our predictor variables are categorical variables, we decide to fit a classification tree [65] to the data using the R package "tree" [66]. The classification tree model is built in the following steps.

First of all, we randomly split the 7,971,509 observations into two parts, one part consists of 4,971,509 observations and serves as the training set in order to build and validate the model, and the other part consists of 3,000,000 observations and serves as the testing set in order to obtain a better estimate of the testing misclassification error of the model.

Secondly, we fit a classification tree to the training set. A large tree is grown with 29 terminal nodes, and the training misclassification error rate is 0.04749. We apply

the tree to the testing set and the testing misclassification error rate is 0.04740. The results are very promising as the misclassification error rate is less than 5% in both the training set and the testing set.

Next, we consider whether pruning the tree might lead to improved results. We perform a 10-fold cross-validation in order to determine the optimal level of tree complexity, and the smallest tree with the smallest cross-validation error rate is selected. The selected tree has 9 terminal nodes, and it is fitted to the training set and then applied to the testing set. The misclassification error rate is 0.05135 in the training set and 0.05125 in the testing set. Compared with the above large tree, the misclassification error rates has only increased slightly while the selected tree has become much simpler.

Figure 2.43 displays the result of the selected tree. It consists of a series of splitting rules, starting at the top of the tree. The splitting rules are based on the value of predictor variables and all 4 predictor variables are being used in this tree. For example, the top split assigns observations/transactions having variable shadow = 1, i.e., the first receiving address as a shadow address, to the left branch and predict the first receiving address as the change address. Observations having variable shadow = 0 or 2, i.e., the second receiving address is a shadow address or neither of the two are shadow addresses, are assigned to the right branch and further splitting are performed based on other variables. There are a total of 9 terminal nodes in the tree, corresponding to 9 different conditions under which the first receiving address in a two-receiving transaction is predicted to be a change address or a payment address.

Link to figure

Figure 2.43.: Classification tree for identifying payment and change.

Finally, we want to note that the training error and testing error of the tree are both fairly small (around 5%), suggesting the model is doing a decent job identifying

payment and change; and also these two are close to each other, suggesting that there is unlikely to be any over-fitting. Indeed, given the vast number of observations and small number of features, over-fitting is unlikely to occur. On the other hand, this provides opportunities to further improve the model, by introducing more and more features. Potential features could be discovered from either within the transaction history data as we have obtained, or from external data sources, including lower level network information such as TCP/IP layer information, off-network information such as voluntary disclosure of ownership of addresses, etc. With the additional features added to the model building set of two-receiving transactions we have identified, similar tree based models or other classification models could be readily built and validated with the enhanced data similar to the analysis we have presented here. Furthermore, while the classification model does not apply directly to three-or-more-receiving transactions, these features we have identified are nonetheless informative to identify the payment address(es) and change address(es) in those transactions. And we will leave these as future work.

2.7 Conclusion

We have described our procedures for the collection and processing of the Bitcoin transaction history data, and we have designed and constructed various databases of transaction level and address level information that enable efficient individual random queries and entire database scans, and facilitate comprehensive analysis of Bitcoin transactions and Bitcoin addresses. The transaction based database consists of information per transaction basis, and various properties of the transactions are analyzed while extensively utilizing the database. The address based databases consist of summarized or detailed information per address basis, and powered by these databases, we have performed an analysis of overall properties of all addresses, as well as the construction and analysis of the profiles of a few representative groups of addresses. Furthermore, we have proposed and validated a classification model

for the identification of payment and change in Bitcoin transactions: the model itself effectively identifies payment and change in two-receiving transactions, and along with the model building set of transactions we have identified and the features characterizing the properties of payment and change we have proposed, they provide a framework for building and validating similar models to identify payment and change and a novel method of studying the anonymity of the Bitcoin system.

REFERENCES

REFERENCES

[1] Guha, Saptarshi and Hafen, Ryan and Rounds, Jeremiah and Xia, Jin and Li, Jianfu and Xi, Bowei and Cleveland, William S. Large Complex Data: Divide and Recombine (D&R) with RHIPE. *Stat*, 1(1):53–67, 2012.

[2] W. S. Cleveland and R. P. Hafen. Divide and Recombine (D&R): Data Science for Large Complex Data. *Statistical Analysis and Data Mining*, to appear.

[3] Ryan Hafen, Luke Gosink, Jason McDermott, Karin Rodland, Kerstin Kleese-Van Dam, and William S Cleveland. Trelliscope: A System for Detailed Visualization in the Deep Analysis of Large Complex Data. In *Proceedings, IEEE Symposium on Large-Scale Data Analysis and Visualization*, pages 105–112. IEEE, 2013.

[4] Steven L Scott, Alexander W Blocker, Fernando V Bonassi, H Chipman, E George, and R McCulloch. Bayes and Big Data: the Consensus Monte Carlo Algorithm. In *EFaBBayes 250" Conference*, volume 16, 2013.

[5] Apache Hadoop. `http://hadoop.apache.org/`.

[6] Apache Spark. `https://spark.apache.org/`.

[7] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, pages 10–10, 2010.

[8] Tessera: Open Source Environment for Deep Analysis of Large Complex Data. `http://tesseradata.org/`.

[9] R Core Team et al. R: A Language and Environment for Statistical Computing. 2012.

[10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[11] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[12] Saptarshi Guha. *Computing Environment for the Statistical Analysis of Large and Complex Data*. PhD thesis, Purdue University Department of Statistics, 2010.

[13] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[14] Shrinivas B Joshi. Apache Hadoop Performance-tuning Methodologies and Best Practices. In *Proceedings of the third joint WOSP/SIPEW International Conference on Performance Engineering*, pages 241–242. ACM, 2012.

[15] Kimball, Aaron. Configuration Parameters: What Can You Just Ignore? http://blog.cloudera.com/blog/2009/03/configuration-parameters-what-can-you-just-ignore/, 2009.

[16] Guanying Wang, Ali Raza Butt, Prashant Pandey, and Karan Gupta. A Simulation Approach to Evaluating Design Decisions in MapReduce Setups. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on*, pages 1–11. IEEE, 2009.

[17] Guanying Wang, Ali R Butt, Prashant Pandey, and Karan Gupta. Using Realistic Simulation for Performance Analysis of MapReduce Setups. In *Proceedings of the 1st ACM Workshop on Large-Scale System and Application Performance*, pages 19–26. ACM, 2009.

[18] Herodotos Herodotou and Shivnath Babu. Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. *Proc. of the VLDB Endowment*, 4(11):1111–1122, 2011.

[19] Shivnath Babu. Towards Automatic Optimization of MapReduce Programs. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 137–142. ACM, 2010.

[20] Dawei Jiang, Beng Chin Ooi, Lei Shi, and Sai Wu. The Performance of MapReduce: An in-depth Study. *Proceedings of the VLDB Endowment*, 3(1-2):472–483, 2010.

[21] Jeff Buell. A Benchmarking Case Study of Virtualized Hadoop Performance on VMware vSphere 5. *technical white paper. VMware, Inc*, 2011.

[22] Satoshi Nakamoto. Bitcoin: A Peer-to-peer Electronic Cash System. *Consulted*, 1:2012, 2008.

[23] Bitcoin v0.1 Released. http://www.mail-archive.com/cryptography@metzdowd.com/msg10152.html.

[24] Bitcoin Proof of Work. https://en.bitcoin.it/wiki/Proof_of_work.

[25] Bitcoin Pooled Mining. https://en.bitcoin.it/wiki/Pooled_mining.

[26] Fergal Reid and Martin Harrigan. An Analysis of Anonymity in the Bitcoin System. In *Security and Privacy in Social Networks*, pages 197–223. Springer, 2013.

[27] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating User Privacy in Bitcoin. In *Financial Cryptography and Data Security*, pages 34–51. Springer, 2013.

[28] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A Fistful of Bitcoins: Characterizing Payments Among Men with No Names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[29] Malte Möser. Anonymity of Bitcoin Transactions: An Analysis of Mixing Services. In *Proceedings of Muenster Bitcoin Conference 2013*, 2013.

[30] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and Anonymity of the Bitcoin Transaction Graph. *Future Internet*, 5(2):237–250, 2013.

[31] Dorit Ron and Adi Shamir. Quantitative Analysis of the Full Bitcoin Transaction Graph. In *Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.

[32] Bitcoin Mixing Service. https://en.bitcoin.it/wiki/Mixing_service.

[33] Bitcoin Client Program. https://bitcoin.org/en/download.

[34] LevelDB. https://code.google.com/p/leveldb/.

[35] Bitcoin APIs. https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_Calls_list.

[36] Barret Schloerke. Modified Bitcoin Transaction Network Extraction Tools. https://github.com/schloerke/Bitcoin-Transaction-Network-Extraction.

[37] Ivan Brugere. Bitcoin Transaction Network Extraction Tools. https://github.com/ivan-brugere/Bitcoin-Transaction-Network-Extraction.

[38] Bitcoin Block Chain Explorer. http://blockchain.info/.

[39] Bitcoin Market. https://en.bitcoin.it/wiki/Bitcoin_Market.

[40] Mt. Gox. https://mtgox.com.

[41] SatoshiDice. http://www.satoshidice.com.

[42] Rules for Calculating Minimum Fees. https://en.bitcoin.it/wiki/Transaction_fees.

[43] Bitcoin Raw Transaction. https://en.bitcoin.it/wiki/Raw_Transactions.

[44] Bitcoin Client Soft-fork "No Forced TX Fee". https://bitcointalk.org/index.php?topic=22434.0.

[45] Dan Carr, Nicholas Lewin-Koh, and Martin Maechler. hexbin: Hexagonal Binning Routines. *R package version*, 2014.

[46] Manhattan U.S. Attorney Announces Seizure Of Additional $28 Million Worth Of Bitcoins Belonging To Ross William Ulbricht, Alleged Owner And Operator Of "Silk Road" Website. http://www.justice.gov/usao/nys/pressreleases/October13/SilkRoadSeizurePR.php.

[47] Nicolas Christin. Traveling the Silk Road: A Measurement Analysis of a Large Anonymous Online Marketplace. In *Proceedings of the 22nd international conference on World Wide Web*, pages 213–224. International World Wide Web Conferences Steering Committee, 2013.

[48] Dorit Ron and Adi Shamir. How Did Dread Pirate Roberts Acquire and Protect His Bitcoin Wealth?

[49] Erik Voorhees, the Founder of SatoshiDice. http://en.wikipedia.org/wiki/Erik_Voorhees.

[50] Bitcoin Forum. https://bitcointalk.org/.

[51] SatoshiDice Sold for $12.4 Million. https://bitcointalk.org/index.php?topic=101902.msg2751536#msg2751536.

[52] SatoshiDice Tribute. https://bitcointalk.org/index.php?topic=322063.0.

[53] BetCoin Dice. https://www.betcoin.tm/dice/.

[54] SatoshiBones. http://bitzillions.com/satoshibones.

[55] LuckyBit. http://luckyb.it/.

[56] DeepBit. https://deepbit.net/.

[57] GHash.IO. https://ghash.io/.

[58] Bitcoin-24. https://bitcoin-24.com/en/.

[59] Reddit Bitcoin Faucet. http://www.reddit.com/r/Bitcoin/comments/zqocl/exchange_your_karma_for_bitcoin_reddit_bitcoin.

[60] Reddit: The Front Page of the Internet. http://www.reddit.com.

[61] Bitcoin Faucet. http://web.archive.org/web/20120407230101/http://freebitcoins.appspot.com/.

[62] CoinBox.me. http://www.coinbox.me/.

[63] WikiLeaks Accepting Bitcoin Donations. https://twitter.com/wikileaks/status/80774521350668288.

[64] WikiLeaks Generating New Addresses to Accept Donations. http://shop.wikileaks.org/donate#dbitcoin.

[65] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and Regression Trees*. CRC press, 1984.

[66] Brian Ripley. tree: Classification and Regression Trees. *R package version*, 2014.

VITA

VITA

Jianfu Li was born in Tongshan, Hubei, China in 1986. He earned his B.S. in Statistics and Probability at Yuanpei College, Peking University, China in 2008, and his M.S. in Mathematical Statistics at Purdue University, Indiana in 2011. His academic interests include statistical model building, exploratory data analysis, data visualization, and statistical computation with massive data.