ABSTRACT

Title of thesis: SOFTWARE CRASH STUDY

Yantao Zhang, Master of Science, 2015

Thesis directed by: Professor Tudor Dumitras
Department of Electrical and Computer Engineering

With the development of personal computers, the user experience has become a vital part of every day work and life of the majority of people on the planet. Hardware components are usually preconfigured and most people tend not to tune them. However, the software environments change much more often because of the configuration by users, the upgrading by vendors and the attacks by hackers. All of those activities can be a factor in the stability of software. In this work, by analyzing a sample of 600,000 machine weeks and around 16,000 applications used on them, we try to derive the relationship between the software environment and the crashes of software. We mainly used association rule mining and analyzed our data on Spark. We also examined the predictability of crashes using the association rules and the difference of predictability between different versions of a same application.

SOFTWARE CRASH STUDY

by

Yantao Zhang

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2015

Advisory Committee:
Professor Tudor Dumitras, Chair/Advisor
Professor Edoardo Serra
Professor Joseph Jaja

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost I'd like to thank my advisor, professor Tudor Dumitras for giving me this opportunity to get hands on experience with valuable real world data and giving me generous guidance throughout the way. There are so many things I learned from him during this experience. Research is never easy and the most important thing is not the intelligence, but the patience and the determination. Sometimes I can easily be overwhelmed by the complexity of the problem and numerous failures coming from my seemingly naive ideas, but professor Dumitras always gives me confidence and encourages me to try.

Apart from research, I learned a lot by getting to know about professor Dumitras. He has great passion about everything he does, marathon, glass-icon painting, mountain climbing, etc. Life can sometimes have tough moments if we want to achieve something. The greatest lesson that I learned was that happiness depends mostly on your attitude for life, not what life brings to you.

I would also like to thank professor Edoardo Serra for discussing interesting ideas with me throughout the process. There are a lot of times I want to give up and you give me the courage to keep doing it. Grazie Mille!

I would also like to thank my parents who gave me encouragements throughout the whole process. Last year has been a very tough one for me. I experienced a lot of difficulties from both research and life. I had to learn all the fundamentals about

cloud computing in order to run all the algorithms on the clusters. I remembered the days when I read the tutorial online and tried to configure the pseudo environment on my own laptop. I read a lot of technical blogs and some of them have very crucial mistakes inside. There were times that I got stuck by an error the whole day and could not proceed. I was really worried because I wanted to produce meaning results. There were a lot of sleepless nights where I felt my work was dead in the water. My parents told me to take it easy and just try my best to do whatever I can. They told me that life was not about getting the results, but about the process. I used to believe that being successful was the most important thing for life. Now I realize there are so many things that are far more important, love, health, friendship,etc. Life is about experiencing different things and goals are not important at all.

I would also thank professor Joseph Jaja from Electrical and Computer Engineering department and Katherine C.McAdams from UMD ombuds office. Thanks for providing support during my study.

It is impossible to remember all, and I apologize to those I've inadvertently left out.

There are so many difficulties throughout the process, but I think I get the most important things in life from it. It helps me become a better person.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Overview

With the development of modern software technology, personal computer usage has experienced significant increase in the recent years. During the last twenty years, the average number of computer uses per 100 people has increased from almost none to 20 computers and this trend is still going up. And most of the computers sold each year are consumer PCs. The reliability issue of personal computers has become a more and more important factor as it impacts the day-to-day work and life of almost all of us.

Throughout the years, there have been extensive studies on personal computer reliability. The reliability of a personal computer in general depends on two factors, hardware and software. The reliability of software is a problem more complex than hardware reliability. Compared to hardware reliability, the reliability of software constantly changes. There are a lot of patching and updates after the initial release of the software. And the hardware environment of the hosts is evolving as the hardware technology improves. And the variance also comes from the different behaviors of users. Some users might be interested in configuring software more often than others. The knowledge about the machines and software technology also

varies a lot between different end-users. So factors like user behaviors and software environments might play a role as important as the code produced by vendors.

[1] gave a review of the a number of models that have been derived based on the pre and post behavior of the development of software products. The work pointed out that the reliability of complex software is still not well understood and remains challenging to quantify. This poses a problem for estimating the overall reliability that we can expect from the computer. This problem in practice is a very big issue for personal computer users as software crashes and even operating system instability will hinder the work efficiency for end-users. Thus understanding the possible causes for the crashes of software and operating systems are very important and it will allow users to assess how much they can rely on their own computers and what they can do in order to increase the stability of software. Good prediction models of the software reliability will allow the software vendors to allow resources for the development of software patches and to plan accordingly for future releases. The software vendors will also be able to address the problem directly if root cause of the crash can be discovered. It will also be helpful for software testing as it can provide a standard environment for testing the software.

In this thesis, we look at how the software environment influences reliability. We will discuss about examples of the coexistence of software causing instability of one of them. Besides, we will show to what extent we can predict the software crashes based on this factor. The goal of this study is to provide insight into the fact that software environment is an important factor that lead to the crash.

All the data is from Worldwide Intelligence Network Environment(WINE)

from Symantec, which provides us a wide range of end-hosts across the world. This makes the result very valuable since it considers the wide variability of different users and machines.

1.2 Related Work

There have been a lot of empirical studies focusing on the impact of hardware components to stability of computers. [2] presents the first large-scale analysis of hardware failure rates on a million consumer PCs. They find that hardware induced failures are recurrent and not independent. CPU speed and configurations of the desktop are related to hardware failures. They are also able to spot the spatial locality of DRAM failures. [3] provides a better understanding of what disk failures look like in the field. It is the first work to provide a large-scale study of disk failures in production systems. The work showed that MTTF(mean time to failure), which is a commonly used formula for disk replacement time estimation, was not practical according to the field data they have. [4] is the first work to study on the real DRAM failures in large production clusters. They analyzed measurements of memory errors in a large fleet of commodity servers over a period of 2.5 years. They mainly addressed the problem of frequency of memory errors in practice and the effect of external factors on the DRAM age. These works have proved that there is a gap between the lab measurements and data collected in the field.

Software reliability has been studied with respect to different viewpoints for a long time [5]. Most software reliability studies are focused on the software itself. [6]

tries to analyze a modular program as a composition of modules where the transfer of the control follows a semi-Markov process. The failure of each process is modeled as a poisson process. Littlewood shows the cost (induced by software failures) with different program running times. There has been some work focusing on the bugs for specific software. [7] uses an automatic, static compiler analysis to Linux and OpenBSD kernels to find the bugs from those operating systems and compared the bugs across different parts of the operating system and across different versions of the operating system. This is the first work to automatically find errors in operating systems based on the software they developed in [8]. Drivers directory is found to contain up to 7 times more of certain kinds of faults than other directory. After this, there have been multiple efforts done in improving the reliability of driver code. Ten years later, [9] transported the experiments to test on new versions of Linux released between 2003 and 2010. They showed that Linux kernel doubled in size during the period but the fault per line was decreasing. And the average fault rate of drivers directory is below the rate of other directories. These two work show that the research in this field has great impact on the development of software.

Throughout the years, there has been a trend of increasing software reliability discovered. [10] introduced the idea that the software reliability would increase as the user got more familiar with the software, which was captured by previous models. And they found that this kind of growth also comes from the fact that installation of some products involves installing some new drivers which would help the software to be more reliable. They included both factors in their model and achieved a better estimation of the crash of software.

However, the current literature of software reliability study faces two problems. First, the failure of software is defined very broadly and some of them are very different from user experience. This results in the fact that user-reported failures are often not compatible with data from elsewhere. Secondly, most field studies focus on enterprise systems that are professionally monitored and the hardware is consistent across machines.

In practice, users and vendors rely on certain ad-hoc proposals or popular beliefs in order to improve the reliability of PCs. To assist those problems better and considering the fact that consumers consist of a large portion of personal computer purchases, a study of machines involving wide range of users and software are needed in order to target the problems.

There is very few work currently that has been done on analyzing software behaviors and crashes on a large scale in the field. [11] presents the first in-depth analysis of I/O behavior of productivity applications among home-user applications in Apple desktop. An instrumentation framework based on the tracing system DTrace is used to build a benchmark for I/O workload comparison among those applications. The study reveals the organization of modern files and the lack of pure sequential accesses. It will help developers better understand the internal behavior of the applications on the platform and design better local and cloud-based storage systems. [12] proposed the idea that software reliability is affected by the presence of other software by a study of installation, usage and crash information on 200,000 machines with Windows in CEIP dataset [13]. The study focuses on 53 most used applications and explored the interaction between them. Both one to one interaction

and many to one interaction are explored. It also showed that mere installation of an application can affect the reliability of other applications. However, the work is focused only on a limited number of machines and did not check the consistency of the findings. Since it only considers the most commonly used applications, the impact of some uncommon applications is not studied. Since the application crash itself is rare, it is likely that the major cause can come from those uncommon applications.

Chapter 2: DATA AND METHODS

2.1 Overview

The research conducted in this thesis is based on the data collected by Worldwide Intelligence Network Environment(WINE), a platform for data intensive experiments in cyber security. WINE was developed by Symantec Research Labs for sharing comprehensive field data withe the research community. It provides a platform for repeatable experimental research. In order to fulfill the gap of insufficient data in computer security research, WINE provides a unique way by enabling external research on field data collected inside Symantec and by promoting rigorous experimental methods. Researchers are able to define the reference data sets and validate new algorithms or conduct empirical studies and try to establish the validity of the data sets for cyber security threat. Reproduction of previous experimental results are made possible by the existence of WINE and the comparison of different algorithms on the same dataset is thus able to be done. What's more, WINE also provides researchers a way to do the comparison and examine the results of algorithms across time. And the use of WINE dataset is not only limited to cyber security research, it can also be applied to researchers related to software engineering, software testing, etc.

WINE includes filed measured data, aggregated from 240,000 sensors from all over the world. The sensitive information in the data sets are protected so that researchers can only have access to the raw data onsite. All the snapshot of experiments performed are recorded for future reference.

WINE samples and aggregates multiple petabyte-size data sets, which Symantec uses in its day-to-day operations, with the aim of supporting experiments at a scale. The data we use in this study is archived on the WINE platform and is available to the research community for independent verification of our results and for follow-on studies. Because data on software failures is scarce, we believe that the public availability of the data that accompanies our findings will enable researchers to further investigate fundamental questions about software designs and testing environments.

2.2 Tables from WINE

In this section, we will describe the tables that we extract from WINE that is used to help us understand the software crash behaviors.

2.2.1 WINE_BINSTAB Schema

The WINE_BINSTAB Schema(the stability telemetry) is a collection of stability reports and the details of applications related to these stability reports.

The stability telemetry dataset provides us with a lot of information about the characteristics and configuration of monitored hosts as well as information about the

applications installed on them and any application crashes and hangs they experienced. Three reports from this telemetry are used.

- **Stability Reports** These reports are summaries of different kinds of errors that occur on hosts. Norton products collect this information on participating hosts by processing the Windows Event Log. Errors are reported in one of several categories including application error, application hang, msi installer error, miscellaneous error, service control manager error and system crash. Each report covers some period of time, on average a couple of hours.

- **Process Reports** For all participating hosts, daily summaries of the cumulative runtime of each process are submitted. Each process is identified uniquely by version and its application name. There is also additional information about the time of the sampling and other meta information like file version and file path. Each row contains information about the file as well as the cumulative runtime of the file during the sampling period of 24 hours.

- **FaultingApps Reports** These reports are the details of crashes corresponding to Stability Reports during the Stability Report sampling period. They are also collected by processing the Windows Event Log. This table tells us the exact number of faults for each application during the sampling period and the type of errors(application crash or application hang).

2.2.2 WINE_DIM Schema

The WINE_DIM Schema is a collection of mapping for all the ids in the other schemas to the real values.

- **dim.filesha2** This is a mapping from file_sha2_id to filesha2.
- **dim.filename** This is a mapping from file_name_id to filename.
- **dim.filedirectory** This is a mapping from file_directory_id to filedirectory.
- **dim.filemd5** This is a mapping from file_md5_id to filemd5.
- **dim.fileversion** This is a mapping from file_version_id to fileversion.
- **dim.url** This is a mapping from url_id to url.
- **dim.filesignerissuer** This is a mapping from file_signer_issuer_id to file_signer_issuer.
- **dim.filesignersubject** This is a mapping from file_signer_subject_id to file_signer_subject.

2.3 Data Preprocessing

The raw data from the tables listed above has two problems. The size of the data is too large and raw data can not be taken directly out of Symantec due to the policy. Thus we need to do some preprocessing so that the size is feasible and all the data we taken back is the aggregation results.

2.3.1 Extracting Data From WINE

The first step is to extract the data from WINE. And we mainly perform SQL operations on the following tables and we do not take the information of all the participating hosts, but only 25% of the total machines.

- **Application Usage Information** The ProcessRunTime table is used to get the application usage. We aggregate the total number of instances for each process on a specific machine on a weekly basis.

- **Application Crash Information** The FaultingApps table is joined with StabilityReport table to get the application crash information. We aggregate the total number of crashes for each process on a specific machine both on a a weekly basis.

2.3.2 Matching Uses with Crashes

After the aggregation of the two tables, we are going to match the application usage table and the application crash table. The process procedure is shown in figure 2.1.

During the application usage aggregation process, we originally proposed three ways to identify a specific application: name only, name plus sha256, and name plus version. For the name only way, it is easy to do but it will pose some difficulties for further analysis since the name itself is not enough to explore which exact distribution of the application that caused the problem. For the name plus sha256

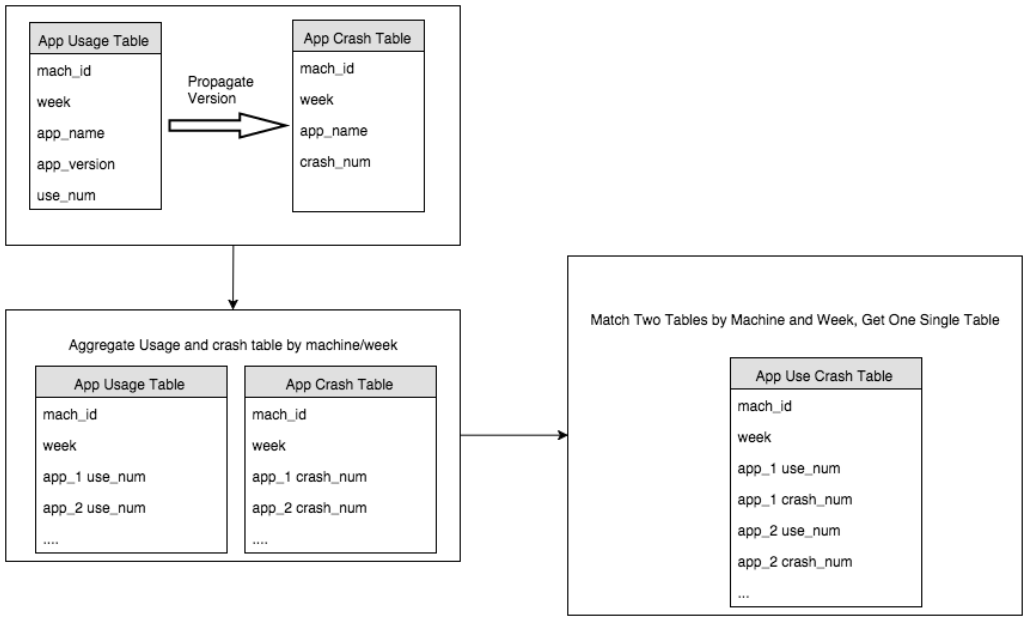

Figure 2.1: Matching Uses With Crashes

way, there are some issues with name and sha256 matching in the database and the matching between them is many to many, which is very weird. Finally, our choice is to use name plus version to make up for the defects of the previous two schemes.

The first thing we do is to propagate the version information in the usage table to the application crash table. This is because Symantec does not include version information for crash table for some reasons. And this propagation is feasible based on the following observation: different versions of the same application are not likely to be used in the same week. We look through the whole fault apps table and the whole process run time table in WINE and find that only 4% of machines have process runtime records for more than one version of the same application in a week.

The second step we do aggregation of the applications, which are identified by

name plus version, based on machine week. In this case, we will get a table such that each row corresponds to one machine in a sampling period. And also there will be a list of numbers of uses for each application on that machine in the specific sampling period(a week). We do the same for the crash table and we will get a table such that each row corresponds to one machine in a sampling period. And also there will be a list of numbers of crashes for each application on that machine in the specific sampling period(a week).

Then we match the machine id and the timestamp in both tables. Each machine id and timestamp pair serve as a unique identifier for our data and is the basis for further aggregation. After this matching process, we will have a table where each row corresponds to one machine in a sampling period. And also there will be a list of numbers of both uses and crashes for each application on that machine in the specific sampling period(a week).

2.3.3 Processing Numerical Data

The second step is to process the data in a meaningful way for the application of algorithms. At this stage, we have the use and crash aggregation data for each week during our five weeks total sampling period. The data is composed of several thousand features, the uses of different applications, and several thousand targets for our research, the crashes of those same number of applications.

In order to compare the features on a same scale, we try to dichotomize the features into binary variables. This can allow easier ways to apply our algorithms.

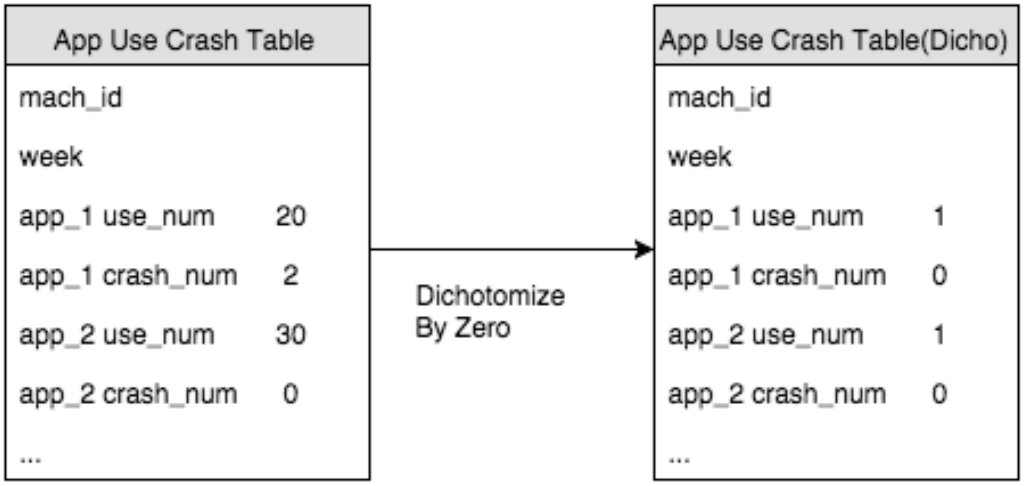

Figure 2.2: Processing Numerical Data

We try difference ways of dichotomizing the data and they represent different perspective of the problem. Following is the details about how we dichotomize the data.

- **Application Usage** We dichotomize the application usage in two ways. The first one is dichotomizing by whether the application is launched at least once in the sampling period on that machine. In this case, after processing, 0 means the application is not used and 1 means the application is used. The second one is dichotomized by median. In this case, after processing, 0 means the application is not heavily used and 1 means the application is heavily used.

- **Application Crashes** We dichotomize the application crashes based on whether it is zero or not. That means if the application crashed at least once in the period, we denote that it has recorded crash in that period.

Chapter 3: Data Analysis Methods

3.1 Overview

Data mining is about finding hidden, unknown and valuable knowledge and rules in big database or data warehouse. It is the combination of artificial intelligence and database, and it is one of the most valuable research directions for database and decision systems based on information. There are a lot of well-researched methods in data mining, such as pattern classification, association rules, decision tree, clustering pattern analysis, neural networks, etc.

Association rule mining is a very important research topic in data mining and it is widely used in almost every field. It is important for two reasons. First, it can be used to examine the knowledge formed by experience in the field. Secondly, it is an effective way to find new rules that are not obvious.

R.Agrawal [14] first proposed the problem of mining association rules in customer trading database. The key point of their work is to use recursion based on frequent item set theories. There are a large volume of work afterwards, including the optimization for Apriori algorithm [15] [16], Multi-level association rule mining algorithms [17], etc to make the efficiency of association rule mining algorithms better.

3.2 Problem Definition

Suppose $I = \{i_1, i_2, ..., i_k\}$ is a set composed of k different items. We have a transaction database, we call D, and every transaction T in D is composed of items from I, we can denote as $T \subset I$, T has a unique transaction identifier in the database. If $X \subset I$ and $X \subset T$, then the transaction T contains itemset X. Association rule has the form of $X \Rightarrow Y$, where $X \subset I$,$Y \subset I$,$X \cap Y = \emptyset$.

We have two criteria in order to form a valid rule $X \Rightarrow Y$. The first one is the support for the rule should be at least S. That means among all the transaction in the database, we should have at least $S\%$ of them containing both X and Y. The second one is confidence C. For all the transactions that contain X, there should be at least $C\%$ of them containing Y.

Association rule mining is to find association rules from the transaction database D that satisfies the minimum support *minsup* and minimum confidence *minconf*. Association rule mining can be divided into the following two sub-problems.

The first one is finding all the itemsets X such that the support of X is higher than the *minsup*, and those X are called large itemsets.

The second one is generating association rules by the large itemsets found in step one. For each large itemset A, if $B \subset A$,$B \neq \emptyset$, and $\frac{support(A)}{support(B)}) \geq minconf$, then we have an association rule $B \Rightarrow (A - B)$.

The second problem is easy to solve [14]. Most of the research is on the first sub-problem. In the next few sections, we will introduce several algorithms that we applied to our problem.

3.3 Apriori Algorithm

Apriori algorithm is an iterative algorithm for obtaining frequent itemsets that satisfy *minsup* level by level, that is, frequent itemsets with $k+1$ items are obtained using frequent itemsets with k items. First, we get frequent 1-itemset, denote as L_1. L_1 is used to get frequent 2-itemset L_2, and L_2 is used for finding L_3, so on and so forth, until there we reache k-itemset L_k where L_k is empty. The algorithm is not very efficient since we need to search the whole database once for each level in order to generate the frequent itemsets. This is necessary since we need to know the support for each of the candidate itemsets and filter the ones below *minsup*. In order to compress the search space, Apriori algorithm uses the following two important properties:

1. If X is a frequent itemset, then all the subsets of X are frequent itemsets.

2. If X is not a frequent itemset, then all the supersets of X are not frequent itemsets.

   There are a couple of bottlenecks for Apriori Algorithm:

1. The number of scans of the whole database is too big. When the transaction database has a huge amount of transaction data, it will pose a great load on I/O if the memory is limited. In each iteration, each element in candidate set must be scanned through the transaction database and filter using minimum support in order to add into frequent itemsets. If there is a candidate itemsets containing 10 items, we need to scan the transaction database 10 times. This

Algorithm 1 Apriori Algorithm

1: **procedure** AprioriAlgo($D, minsup$)
2: $L_1 = FindFrequentOneItemset(D)$
3: **for** $k = 1$; L_k not empty; $k{+}{+}$ **do**
4: $C_{k+1} = CandidateGen(L_k)$;
5: **for** each candidate t in D **do**
6: **for** each candidate c in C_{k+1} **do**
7: **if** c is in t **then**
8: $c.count = c.count + 1$
9: $L_{k+1} = \{c \in C_{k+1} | c.count \geq minsup\}$
10: Return $\cup_k L_k$
11: **procedure** CandidateGen(L_k, k)
12: $CandidateSet = \{\}$
13: **for** C_1, C_2 in L **do**
14: **if** $C_1 \neq C_2$ **then**
15: $c = C_1 \cup C_2$
16: **if** $c.size() == k + 1$ **then**
17: CandidateSet.add(c)
18: Return $CandidateSet$

will cause the scan step taking a lot of time and reducing the efficiency of Apriori.

2. It will cause the generation of candidate sets with large size. The step from L_{k-1} to C_k has exponential growth in size, for example, 10^3 one-item frequent itemset will have a possibility of generating 10^5 two-item candidate set. Thus if we want to get a rule with too many items, the intermediate elements generated in the process will be extremely large.

3. There are some rules that satisfy the minimum support and confidence but have no real significance. If we increase the minimum support, the information we get will be limited and we might be unable to find some meaningful rules.

3.4 Frequent Pattern Tree Growth

In order to solve the problem of Apriori's inefficiency in mining long rules, Han [18] proposed the Frequent Pattern Tree Growth algorithm(FP Growth). This algorithm only scans the whole database twice, which significantly boosts the efficiency of the algorithm. It does not need to do the iterative candidate generation like Apriori does. And due to some properties of the algorithm, it is easier to parallelize the algorithm.

3.4.1 Frequent Pattern Tree

Frequent Pattern Tree is a prefix tree which contains a root with Null value, children (the frequent prefix patterns) and a frequent pattern header which has

pointers to the nodes in the tree. Han [18] defines the FP-tree as follows:

1. One root labeled as "null" with a set of item-prefix subtrees as children, and a frequent-item-header table.

2. Each node in the item-prefix subtree consists of three fields:

   (a) Item-name: registers which item is represented by the node.

   (b) Count: the number of transactions represented by the portion of the path reaching the node.

   (c) Node-link: links to the next node in the FP-tree carrying the same item-name, or null if there is none.

3. Each entry in the frequent-item-header table consists of two fields:

   (a) Item-name: as the same to the node.

   (b) Head of node-link: a pointer to the first node in the FP-tree carrying the item-name.

3.4.2 FP Growth Algorithm

The following table 3.1 is an example of mining the frequent item-sets using FP-tree growth. We use examples related to our data and each transaction corresponds to the list of applications used on that machine in the sampling period. We set the minimum support in the example to be 2. In this example, we have 7 machines and 5 applications in total: ie, word, ppt, googleupdate and chrome.

Table 3.1: Example of FP Growth Algorithm

Machine ID	Applications Used	Reordered Applications
1	{ie, word, chrome}	{word, ie, chrome}
2	{word, googleupdate}	{word, googleupdate}
3	{word, ppt}	{word, ppt}
4	{ie, word, googleupdate}	{word, ie, googleupdate}
5	{ie, ppt}	{ie, ppt}
6	{word, ppt}	{word, ppt}
7	{ie, ppt}	{ie, ppt}
8	{ie, word, ppt, chrome}	{word, ie, ppt, chrome}
9	{ie, word, ppt}	{word, ie, ppt}

- Step 1: Scan the whole database one to get the counts of 1-itemsets. In our case, we get ie:6, word:7, ppt:6, googleupdate:2, chrome:2. All of them satisfy the minimum support.
- Step 2: Sort the list of applications by their frequency. Then we have this ordering: word, ie, ppt, googleupdate, chrome.
- Step 3: Reorder the database based on the ordering.
- Step 4: Initialize the frequent-item-header table, all the links are initialized to NULL. Initialize the frequent pattern tree with a root node as null.
- Step 5: Read in the transactions and build the tree. The tree and the frequent-

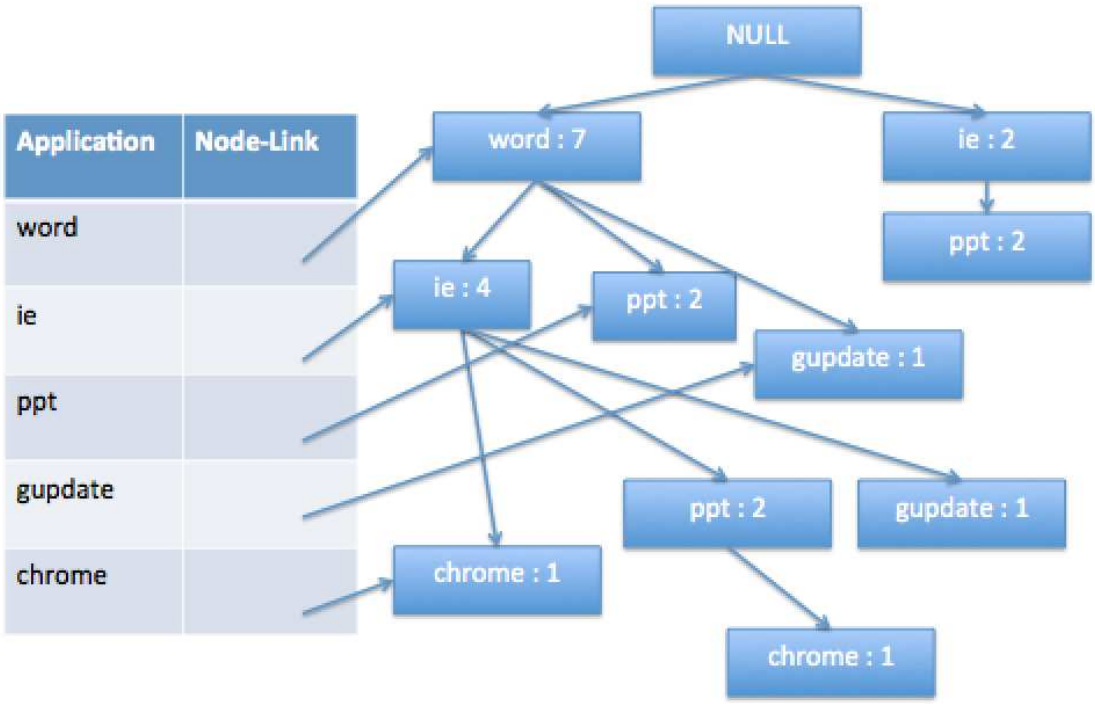

Figure 3.1: Frequent-Item-Header Table and Frequent Pattern Tree

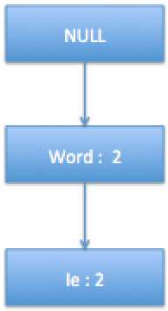

Figure 3.2: Conditional Pattern Tree for Chrome

item-header table is shown in figure 3.1. Increase the count of the node visited(if not visited, create the node and set the count to be 1) in the frequent

Table 3.2: Mining FP-Tree

Item	Conditional Pattern	Conditional FP-tree	Frequent Itemsets
chrome	(word, ie):1, (word, ie, ppt):1	(word : 2, ie : 2)	word chrome : 2, ie chrome : 2, word ie chrome : 2
gupdate	(word, ie):1, (word):1	(word : 2)	word gupdate : 2
ppt	(word):2, (ie):2, (ie, word):2	(word : 4, ie : 2), (ie : 2)	word ppt : 4, ie ppt : 2, word ie ppt : 2
ie	(word):4	(word:4)	word ie : 4

pattern tree and add a pointer to the node at the end of node-link.

- Step 6: Starting from the last item(the one with the smallest support) in the frequent-item-header table and find all the path that contains the item. In this case, we start from chrome and we have two path in the tree that leads to it. And we update the count of the nodes so that the count only involves the path considered. And the updated prefix paths will be: $word, ie, chrome$ having count 1 and $word, ie, ppt, chrome$ having count 1. And we can build a subtree based on these nodes, where word has count 2, ie has count 2 and ppt has count 1. We then delete the nodes having count less than the minimum support. We finally get the conditional pattern tree for chrome to be $word, ie$ having count 2. The conditional pattern tree for chrome is shown in figure 3.2.

- Step 7: Step 6 is done for each node and we are able to get all the frequent itemsets satisfying the minimum support.

Table 3.2 shows the final result from FP growth algorithm in our example.

Algorithm 2 FP Growth Algorithm

```
1: procedure FPGAlgo(Tree T, Pattern P)
2:     if T contains a single path then
3:         for each combination C of the nodes in the path do
4:             generate pattern C ∪ P with support to be MinSup of nodes in C
5:     else
6:         for each i in the header of T do
7:             generate pattern C = i ∪ P with support to be support of i
8:             construct the conditional pattern base of C and its conditional FP
   Tree T_C
9:             if T_C is not empty then
10:                FPGAlgo(T_C, C)
```

3.5 Cloud Computing Platforms

In this research, considering the number of machines and the number of applications that we are mining on, the dimensionality of the data makes it impossible for us to run algorithms locally. In order to efficiently get as much rules as possible out of the large amount of data, we used mainly two platforms: Apache Hadoop and Apache Spark.

3.5.1 Apache Hadoop

Apache Hadoop is an open-source cloud computing platform implemented in Java for distributed storage and processing of large datasets on computer clusters.

It is composed of two main parts: the distributed storage system(HDFS) and processing modules(MapReduce).

The Hadoop Distributed File System(HDFS) is a distributed file system that supports high fault tolerance and thus can be deployed to very low-cost hardware. The HDFS provides high throughput access to data and is thus suitable for problems with large datasets.

The files in HDFS are stored in blocks across machines. Each block is replicated several times on different machines in order to have good fault tolerance. By default, each block is 64MB. Each block is replicated three times by default. The block size and replication factor can be configured for each file.

HDFS employs a master/slave mechanism. There is one namenode(master) and a lot of datanodes(worker). The namenode is responsible for storage of the metadata about the storage place of the blocks of information(mapping information about which datanode stores a specific block). It also executes file system operations for opening, closing and renaming files and directories. The datanodes will serve read and write requests from the client and also perform file operation instructions from namenode.

3.5.2 Apache Mahout

Apache Mahout is an open source library for machine learning on a large scale implemented in Java and MapReduce. It supports most of the machine learning algorithms in literature and can be run both in sequential mode and in mapreduce

mode.

3.5.3 Parallel FP Growth

Frequent pattern tree growth algorithm is implemented in mahout versions before 0.8. The algorithm is based on the work in [19].

Figure 3.3 shows the procedure of the algorithm in Mahout. The implementation is based on the following steps, and in total there are three mapreduce procedures:

- Step 1: Split the database into several data shards and distribute to P nodes.
- Step 2: MapReduce1. Using the same approach as WordCount [20] and get the frequent 1-itemsets. Then sort them decreasingly based on the support and get a table, we call it F-List.
- Step 3: Separate the items in F-List into Q groups and assign a unique group id to each one. Then we have all the items along with its group id, we call this the G-List.
- Step 4: MapReduce2. Take the data shards from Step 1 as the input to mapper. For each transaction T_i, go through items in the transaction($\{a_1, ..., a_n\}$) from the last one. If we are currently processing a_L and it is the first time that its group id in G-List is scanned, then we output a key value pair. Key is the group id and value is the itemset $\{a_1, ...a_L\}$. Otherwise, we do not ouput anything. And the reducer takes the key value pair output from the mapper,

$[key = groupid, value = \{\{valuelist1\}, ...\{valuelistN\}\}]$. The reducer will do a recursive FP Growth mining on the value-list it receives. Instead of directly returning all the itemsets, it will put them into maximum heaps. Each maximum heap corresponds to one item. And the output key value pair of the reducer will be $[key = groupid, value = \{top\ K\ frequent\ patterns\ containing\ the\ item\}]$. Top K means the top k itemsets with the largest support.

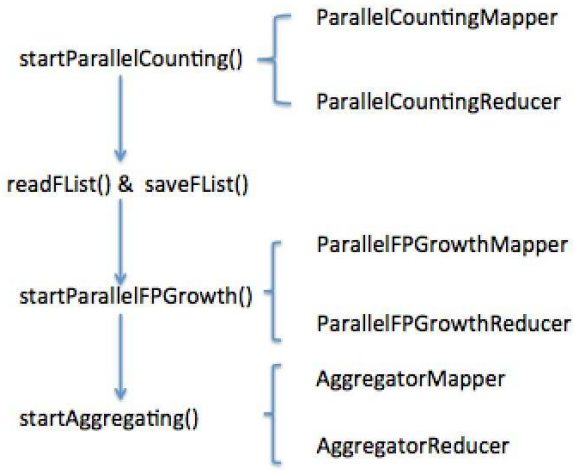

Figure 3.3: Parallel Frequent Growth Algorithm Mahout

- Step 5: MapReduce 3. Since the mapreduce procedure in step 4 used group ID to divide the data, we can expect that the same item may end in several different heaps. This is because each group of itemset list can possibly contain this item. That means the output of Step 4 will possibly have key value pairs with same item as key. And we need to merge those key value pairs to get one heap rather than a couple of heaps. The mapper in this step will take the output of step 4 as input. For each key value pair, we get all the items appeared

in the value. Then for each of these items, it outputs [$key = item\ t, value =$ *all the frequent itemsets containing item t on this node*]. The reducer will then merge the values(which is list of frequent itemsets) with the same key.

3.5.4 Independence Test for Rules

One of the problems brought by the association rule mining is the explosion of frequent itemsets. And it is very important to eliminate rules that are of low importance. It is both crucial for prediction reasons and for the understanding of the real causes of the problem we are researching on.

[22] discussed the fact that we should care about how likely the association of the left-hand side(LHS) of the rule and the right-hand side(RHS) is real. They can appear together on a random chance or due to a systematic effect. But neither support nor confidence tells us that LHS and RHS are independent or not. If they happen to be independent, then the rule is meaningless to us. So we can employ some statistical measures to tell us if a rule is important or not.

Statistical tests usually use the p-value to tell whether the result is significant or not. In the case of association rules, the p-value is the probability of the LHS and RHS are independent. If the p-value is high enough, then we can say that LHS and RHS have a high chance to co-occur even if they are independent. And these rules are the ones that should be excluded from analysis. For instance, if we only take the rules having p-value over 0.001, then that means we have 0.1% of the rules after this filtering can contain items that are completely independent. Adjusting

this cut-off threshold will allow us to reduce false-positives. On the other hand, if we only use support and confidence as thresholds for the rules, then we can either miss very important rules or get too much rules whose LHS and RHS happen to be together by chance.

In computational linguistics study [23], several important statistics have been used to check if two words in a bigram happens to be together at random. If they do not happen at random, then they are very likely to be collocations. For example, we tend to say "powerful computers" to describe computer with strong computing abilities. And statistical tests are able to reject the independence null hypothesis in these scenarios.

In general, Student's $t-$test, Pearson's $chi-$square test and Dunning's likelihood ratio test are used in testing the significance of independence.

The $t-$test computes the mean and variance of the samples of measurements. The defect of $t-$test is that it always assumes a normal distribution and it is not true in general cases. The χ^2 test is an alternative that doesn't assume normal distribution. [24] has examined the effectiveness of applying the test for independence test of association rules. The key idea is to do comparisons of the observed frequencies and the expected frequencies given the two items are independent. If we have a large discrepancy between the two, then we are able to reject the null hypothesis. However, χ^2 test is problematic when any of those frequencies are very small. And because of that, it is not very suitable for our case.

We will instead use a method called likelihood ratio test, which can deal with sparse data and can work with data that does not have the normal distribution

assumption.

We have the null hypothesis and alternative hypothesis as follows for the rule $W_1 \rightarrow W_2$.

$$H_0 : P(W_2|W_1) = P(W_2|\neg W_1) \tag{3.1}$$

$$H_1 : P(W_2|W_1) \neq P(W_2|\neg W_1) \tag{3.2}$$

We use W_{12} to denote the rule $W_1 \rightarrow W_2$. Let c_1, c_2 and c_{12} to be the frequencies of W_1, W_2 and W_{12}. We assume a binomial distribution for the probability(denote as P_x) of observing W_{12} out of all the cases when we observe W_1 and the probability(denote as P_y) of observing only W_2 out of the all the cases when W_1 is not observed . For binomial distribution in the following context, we will denote as $b(k; n, x)$, that is n choose k with probability x.

We use the multiplication of the two probabilities as the likelihood of observing the real data under the hypothesis. Then if the likelihood under H_0 is way larger than H_1, then we say the H_0 is far more likely to happen than H_1

$$L(H) = P_x P_y \tag{3.3}$$

Under H_0,

$$P_x = b(c_{12}; c_1, p), P_y = b(c_2 - c_{12}; N - c_1, p) \tag{3.4}$$

Under H_1

$$P_x = b(c_{12}; c_1, p_1), P_y = b(c_2 - c_{12}; N - c_1, p_2) \tag{3.5}$$

where p, p_1 and p_2 are the expected probabilities under the hypothesis. p is $P(W_2|W_1)$ and $P(W_2|\neg W_1)$ under H_0, p_1 is $P(W_2|W_1)$ under H_1 and p_2 is $P(W_2|\neg W_1)$ under H_1:

$$p = \frac{c_2}{N}, p_1 = \frac{c_{12}}{c_1}, p_2 = \frac{c_2 - c_{12}}{N - c_1} \tag{3.6}$$

Then the likelihood would be

$$L(H_0) = b(c_{12}; c_1, p)b(c_2 - c_{12}; N - c_1, p) \tag{3.7}$$

$$L(H_1) = b(c_{12}; c_1, p_1)b(c_2 - c_{12}; N - c_1, p_2) \tag{3.8}$$

And the final likelihood ratio is calculated as

$$Likelihood\ Ratio = -2\log\lambda = \log\frac{L(H_0)}{L(H_1)} \tag{3.9}$$

If we use the same notation to get the confidence, the confidence would be

$$Conf = \frac{P(W_{12})}{P(W_1)} = \frac{c_{12}}{c_1} \tag{3.10}$$

Chapter 4: Implementation Apriori on Spark

In our first few tries of rule mining, we mainly used the mahout package. However, one of the problems we find is that the frequent pattern tree growth algorithm is not able to get rules with low support very efficiently. If we feed it with a low minimum support threshold, the candidate set explodes and the algorithm never finishes. However, we think that rules with low support but high confidence might be interesting since there are so many factors that influences the software and none of them should be very prevalent and dominant.

So we turn to Apriori algorithm. As described in Chapter 3, Apriori goes level by level. And thus we can get fewer levels but lower minimum support for each level. That is the reason why we try to implement Apriori on Spark.

4.1 Apache Spark

Apache Spark is an open source cloud computing framework developed in scala by AMP Lab at University of California Berkeley. Spark uses a multi-stage in-memory computation model rather than Hadoop's two stage disk-based model. For some specific applications, Spark performs 100 times faster than Hadoop. Since all the data are loaded into the cluster's memory, users are able to do iterative

algorithms without having to load and save files into disks in each iteration. This makes a lot of fast implementation of machine learning possible.

In general, Spark uses the cluster management systems similar to Hadoop like YARN, Apache Mesos, etc. And Spark is able to be mounted on a couple of different distributed storage systems. And Spark is mainly composed of four components except Spark core, MLib(a machine learning library for Spark), GraphX(graph algorithms library for Spark), Spark SQL and Spark Streaming(streaming application library).

Spark uses Resilient Distributed Datasets (RDDs) for programming abstraction. In this way, it is able to do logical collections of data across machines. And it also allows coarse-grained transformations. All the process follows lazy execution, that is, Spark will do all the computation when it sees fine-grained transformations. This ensures high throughput of the whole program.

4.2 Apriori Implementation on Spark

Before switching to Spark, we have implemented a version of Apriori on Hadoop. The idea is a very simple extension of word count. However the algorithm is rather slow because in each iteration, we have to write the candidate itemsets to disk and load it in the next iteration. That means for each iteration, we have to perform a mapreduce task. This makes the computation extremely expensive since there are too much overhead in writing to disk.

Spark Mlib does not have Apriori algorithm. We use the same approach and

implement the algorithm in Spark. [21] has discussed about similar approaches. Since the result of each level are stored in RDD, we are able to read the result of last level directly from memory.

In this implementation, we mainly do two optimizations. One is to set the output of the previous level to be a broadcast variable. Another one is to use the data structure BitSet to store the original transactions. In this case, we are able to shrink the storage by a huge amount. And storing by BitSet makes it easier to look up by using bit manipulations. This is especially efficient when the candidate set is very huge, which is the case in our dataset. Figure 4.1 shows the diagram

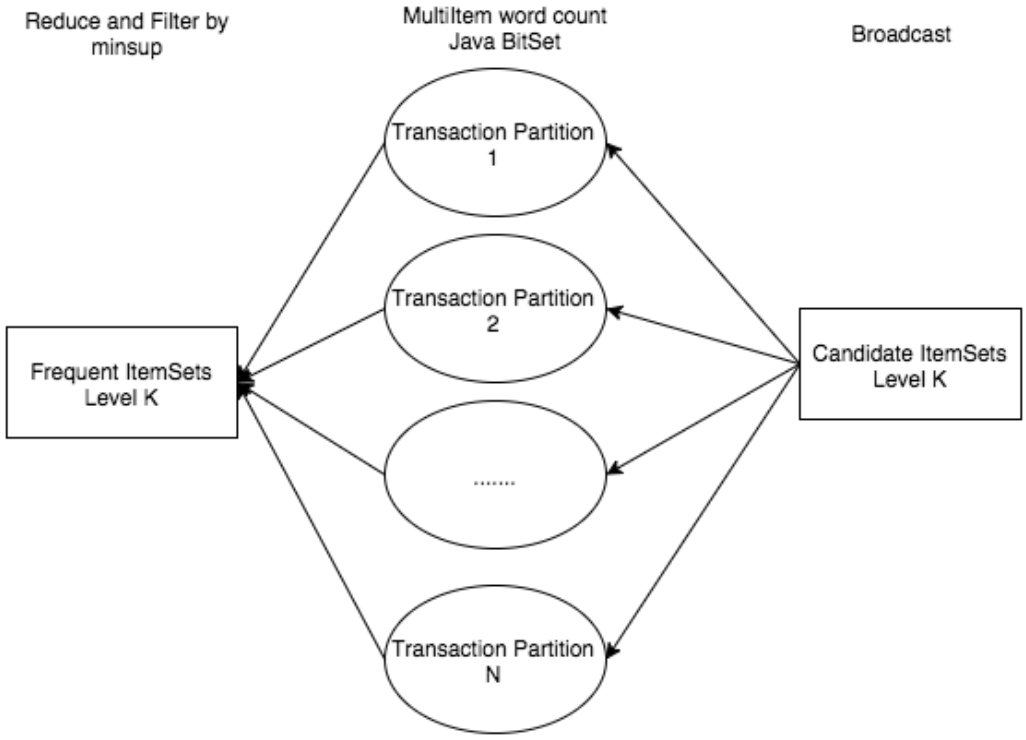

Figure 4.1: Spark Apriori Candidate Generation Diagram

for the candidate generation from the previous level to the current level. All the

optimizations we did are shown in the graph.

Chapter 5: Results and Analysis

5.1 Overview

In this chapter, we will apply the machine learning algorithms to derive knowledge about software crashes. We will analyze the results of the algorithms and do comparisons. There are two sampling period used: two weeks in the middle of July 2014 and the whole month of September 2012.

As described in Chapter 2, we have the following preprocessing.

- Each application is identified uniquely by the executable name plus executable version.
- The crash data and usage data are dichotomized by greater than zero or not.
- Each sampling unit is a specific machine on a specific week.

5.2 Data Preprocessing Result

Table 5.1 and 5.2 summarizes the general data statistics about the two months of data that we are working on. Table 5.1 shows the unique machine/weeks of usage and crashes in the use table and in the crash table. Table 5.2 shows the detailed statistics about the version propagation steps when we use the version information

Table 5.1: General Statistics From the Two Sampling Periods

Period	Weeks	Unique Mach/Weeks in Usage Table	Unique Mach/Weeks in Crash Table
July 2014	2	134248	37773
Sep 2012	5	641882	127991

Table 5.2: Crash Version Propagation Statistics From the Two Sampling Periods

Period	Total Number	No Match Number	Multiple Version Number	Match Number
July 2014	58438	11668	3133	43637
Sep 2012	190560	20574	22450	147536

in the usage table to get the version information in the crash table. The unit in table 5.2 is crash record number, that is the crash information of a specific application on a machine/week.

After we do a dichotomization of the uses and crashes based on greater than zero or not, we got the list of all used and crashed applications used on a specific machine/week. In order to reduce the number of applications to consider as a factor, we filter the applications such that they should have at least 100 machine/weeks presence in order to be considered. Before this filter, we have 268236 applications(name+version) used during July 2014 and 641882 applications(name+version) used during September 2012. After filtering, we have 9229 for July 2014 and 24672 for September 2012.

5.3 Application Crash Distribution

We first try to see how many distinct applications crash on each machine/week. For July 2014 data, we see that on average there are 0.273 distinct application crash for one machine/week. For September 2012 data, we see that on average there are 0.283 distinct application crash for one machine/week. Figure 5.1 shows the histogram of the number of machine/weeks having specific number of crashes, July 2014 and September 2012 respectively. The number of machine/weeks is normalized and represented in percentage. From the histogram, we can see that over 80% of the

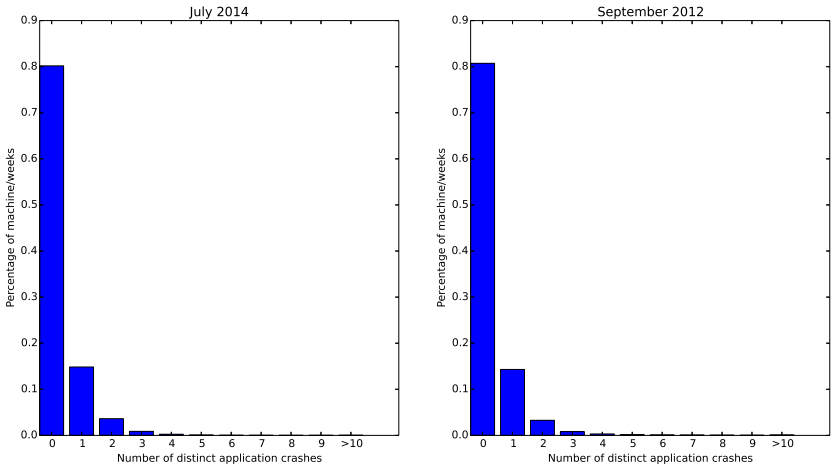

Figure 5.1: Distinct application crashes VS Number of Machine/Weeks

machine/weeks have zero crashes. And most of the machine/weeks that experience application crashes have less than or equal to 4 distinct application crashes. This means that if we are able to analyze the crash causes for most crashed applications(in terms of machine/weeks that it crashed on), we will be able to explain most of

the crashes in this period. And we removed the machines which have more than 4 distinct application crashes in the sampling period so that we are excluding machines that themselves are buggy(tons of application crashes).

Table 5.3: Internet Explorer Version With Most Machine/Week Presence

Version	Mach/Week Uses Count	Mach/Week Crashes Count	Mach/Week Crash Ratio
Internet Explorer July 2014			
11.00.9600.16428	41168	4417	10.73%
11.00.9600.16384	10238	1312	12.82%
9.00.8112.16421	9566	969	10.13%
8.00.6001.18702	6256	345	5.5%
10.00.9200.16384	5355	964	18.00%
10.00.9200.16521	1905	165	8.67%
8.00.7600.16385	1329	132	9.93%
Internet Explorer September 2012			
9.00.8112.16421	325584	29941	9.19%
8.00.6001.18702	92213	5873	6.36%
8.00.7600.16385	32687	3283	10.04%
7.00.6000.16386	10141	1287	12.69%
7.00.6000.17112	4027	135	3.35%
6.00.2900.2180	1944	228	11.72%
6.00.2900.5512	1460	78	5.34%
7.00.6000.17114	1553	49	3.15%
7.00.6000.17055	964	48	4.98%
9.00.8112.16443	308	32	10.38%

5.4 Internet Browser Statistics

We obtained the usage and crash information of three most commonly used internet browsers. From our observation, browsers are among the most commonly used applications on those Windows hosts.

Table 5.4: Chrome Version With Most Machine/Week Presence

Version	Mach/Week Uses Count	Mach/Week Crashes Count	Mach/Week Crash Ratio
Chrome July 2014			
31.0.1650.63	594	31	5.22%
32.0.1700.107	241	11	4.56%
36.0.1985.125	39033	624	1.6%
34.0.1847.131	259	3	1.16%
34.0.1847.116	234	2	0.85%
35.0.1916.153	22018	175	0.79%
Chrome September 2012			
21.0.1180.89	144116	2159	1.5%
20.0.1132.57	544	4	0.74%
21.0.1180.79	690	5	0.72%
21.0.1180.83	25800	161	0.62%
22.0.1229.79	30575	116	0.38%

Internet Explorer is the most often used applications among all the applications in our dataset. And its number of mach/week crashes is the largest among all applications. Table 5.3 shows the statistics about the number of mach/weeks of usage and number of mach/weeks of crash for each version of Internet Explorer in

Table 5.5: Firefox Version With Most Machine/Week Presence

Version	Mach/Week Uses Count	Mach/Week Crashes Count	Mach/Week Crash Ratio
Firefox July 2014			
26.0	190	5	2.63%
28.0	320	5	1.56%
27.0.1	137	2	1.46%
30.0	20592	38	0.18%
29.0.1	743	1	0.13%
31.0	2476	1	0.04%
Firefox September 2012			
15.0.1	48771	1223	2.51%
10.0.2	1924	26	1.35%
11.0	3280	35	1.07%
12.0	6776	71	1.05%
15.0	16421	108	0.66%
9.0.1	2097	12	0.57%
13.0.1	4318	15	0.35%
14.0.1	30020	60	0.2%
8.0.1	1491	2	0.13%

July 2014 and September 2012 respectively. The average crashing rate of Internet Explorer in terms of mach/week is around 10%.

As a comparison, we also list the same statistics about Chrome in Table 5.4 and Firefox in Table 5.5. We can see from the statistics that although the crash ratio between different versions many vary a little bit, the crash ratio of Chrome and Firefox are much less than Internet Explorer.

5.5 Googleupdate Statistics

Googleupdate is another software that is commonly used and has a fair amount of crashes. The table 5.6 below shows the crash statistics in July 2014 and September 2012. It is very interesting that the same version do not crash a lot in September 2012 starts to crash more often in July 2014.

Table 5.6: Googleupdate Crash Ratio

Version	Mach/Week Uses Count	Mach/Week Crashes Count	Mach/Week Crash Ratio
July 2014			
1.2.183.21	38800	1757	4.53%
1.2.183.9	13534	399	2.95%
1.3.21.103	49001	1437	2.93%
1.2.131.7	3170	91	2.87%
1.3.25.0	893	2	0.22%
September 2012			
1.2.183.9	133568	28	0.02%
1.3.21.103	58903	5	0.01%
1.2.183.21	290478	24	0.01%
1.2.131.7	37115	3	0.01%

5.6 Microsoft Office Statistics

Microsoft office is one of the production softwares that are both commonly used and crucial to people's work and life. We calculated the same statistics and we find that the average rate of office applications are actually very low compared with

Table 5.7: July 2014 Office Crash Ratio

Version	Mach/Week Uses Count	Mach/Week Crashes Count	Mach/Week Crash Ratio
Microsoft Word			
14.0.7125.5000	12644	70	0.5%
12.0.6700.5000	10417	75	0.7%
15.0.4631.1000	4949	60	1.2%
12.0.4518.1014	2198	13	0.59%
Microsoft Excel			
14.0.7125.5000	8950	64	0.73%
12.0.6683.5002	7126	81	1.1%
15.0.4631.1000	3447	51	1.5%
12.0.4518.1014	1696	32	1.9%
11.0.8404	1317	3	0.2%
15.0.4623.1000	831	11	1.3%
Microsoft Powerpoint			
14.0.6009.1000	2215	9	0.4%
12.0.6600.1000	1521	5	0.4%
15.0.4627.1000	768	4	0.5%
15.0.4454.1000	421	4	0.95%
12.0.4518.1014	346	4	1.2%
14.0.4754.1000	113	1	0.88%
Microsoft Outlook			
14.0.7113.5000	7302	105	1.4%
12.0.6691.5000	4894	64	1.3%
15.0.4631.1000	2420	50	2.1%
11.0.8326	1229	14	1.14%
12.0.4518.1014	815	15	1.8%
15.0.4623.1000	746	19	2.5%

Table 5.8: Sep 2012 Office Crash Ratio

Version	Mach/Week Uses Count	Mach/Week Crashes Count	Mach/Week Crash Ratio
Microsoft Word			
12.0.6661.5000	89618	463	0.52%
14.0.6024.1000	70496	358	0.51%
12.0.4518.1014	11821	53	0.45%
11.0.8345	20707	54	0.26%
Microsoft Excel			
12.0.4518.1014	8125	67	0.82%
14.0.4756.1000	1859	15	0.81%
14.0.6117.5003	40984	292	0.71%
12.0.6661.5000	49809	297	0.6%
9.0.2719	1941	10	0.52%
11.0.8346	11376	12	0.11%
10.0.6871	2375	2	0.08%
10.0.2614	1242	1	0.08%
11.0.5612	3715	2	0.05%
Microsoft Powerpoint			
14.0.6009.1000	18022	121	0.67%
12.0.4518.1014	2910	13	0.45%
12.0.6600.1000	21010	85	0.4%
11.0.5529	911	3	0.33%
11.0.8335	3210	2	0.06%
Microsoft Outlook			
11.0.8326	9948	129	1.3%
14.0.6117.5001	30162	391	1.3%
12.0.4518.1014	4006	50	1.25%
12.0.6661.5003	30286	373	1.23%
14.0.4760.1000	1265	13	1.03%
11.0.5510	2251	15	0.67%

browser applications, most of the versions have less than or close to 1% of crash rate. Table 5.7 and Table 5.8 lists the statistics for Microsoft Office applications in July 2014 and in September 2012. In general, outlook has the largest crash ratio among office applications.

5.7 Rule Mining Result

The dataset size poses a great problem for rule mining. In terms of total number of transactions(mach/weeks) in our dataset, we have 134248 unique mach/weeks and 9229 unique items that can appear in our transactions in July 2014 and 641882 unique mach/weeks and 20574 unique items.

For Apriori Rule Mining, we want to keep the minimum support as low as possible so we are able to discover the rules that do not have a large amount of support. This is very meaningful for two reasons. First of all, the crash of Internet Explorer itself is not a frequent event with around 10% probability. Secondly, users have different using behaviors and thus the reason for software to crash may be compound effects. Uncommon use cases that lead to high crash rates can be significant in this sense.

For Apriori algorithm, the amount of data and the low support we wish to achieve poses a big problem in computing. Even if we tried to implement algorithms on Spark, this issue is still very big. The large number of items can easily cause the candidates set to explode very easily. Thus we decide to get the rules only limited to two levels, that is we are trying to find rules with one items and two items on

the left hand side.

In order to mine rules as meaningful as possible and mine as efficient as possible, we do two optimizations to reduce the size of the data. First, when we are targeting the crash of a specific application, say application A, we only extract those transactions containing usage of A. Second, when we mine the frequent itemsets, we only use transactions that contain the crash of A. And when we got all the frequent itemsets, we turn back to the transactions that contain A to calculate the confidence for the specific rule.

5.7.1 Association Rules

We present the one item and two item rule mining results obtained by our implementation of Apriori algorithm. We only list the rules with the highest confidence. They are in the tables in attachments.

One important observation from the Internet Explorer results is that the rules with high confidence are composed of mostly adwares and browser extensions. The adwares and browser extensions are written in bold for one item rules. Compared with the average crash rate of those Internet versions, the rate is brought up by several times. This suggests that they are a very important factor in the crash of those Internet Explorer versions. And the results hold if we take a look at the rules for the chrome version.

We also list the rules for one of the most commonly used googleupdate versions in July 2014. As can be seen from the rules, the rules are mainly composed of driver

programs or software service programs. For instance, evteng.exe is supporting software for Intel wireless LAN adapters, riconman.exe is a card reader driver to enable various Realtek PCIE Card Readers and ravcpl64.exe is Realtek High Definition Audio Driver.

5.7.2 September 2012 Data Result Train and Test

For September 2012 result, we have five weeks of data and we are interested in if we are able to use some of the data for training and testing on the rest. In this way, we are able to use the data to answer the following questions.

The first question we want to answer is how consistent the rules are over time. In order to do this, we divide the data into training and testing datasets based on time. We separate the five weeks of data into two groups. The first three weeks of data is used for training rules and the next two weeks of data are used for testing. We use Apriori to extract one item and two item rules from the first three weeks and tested using these rules to predict the next two weeks of data. We tried different thresholds for minimum confidence when we select the rules. As we decrease the threshold, we are able to cover more crashes, but the false positive number also increases.

For Internet Explorer, we plot Receiver Operating Characteristic(ROC) curve for four most frequently used versions in Figure 5.2. Minimum confidence threshold is tuned to get different true positive rate and false positive rate.

As we can see in the graph, our model predicts better than the random clas-

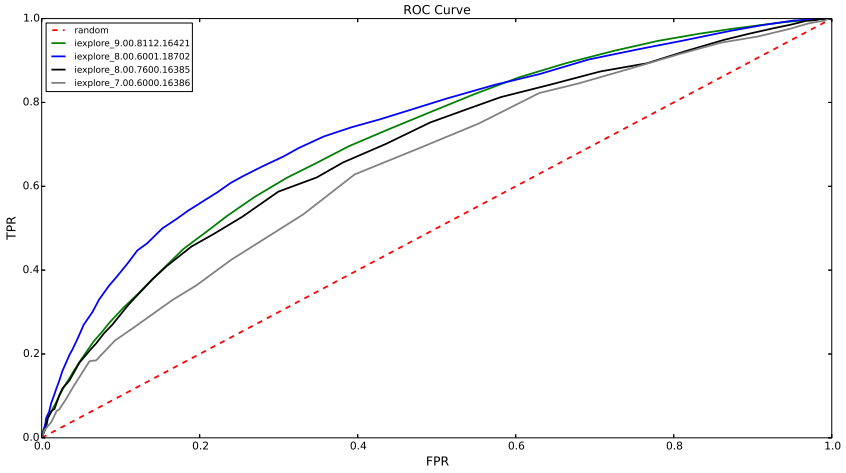

Figure 5.2: IE Roc Curve Sep 2012 (Train/Test Separate By Time)

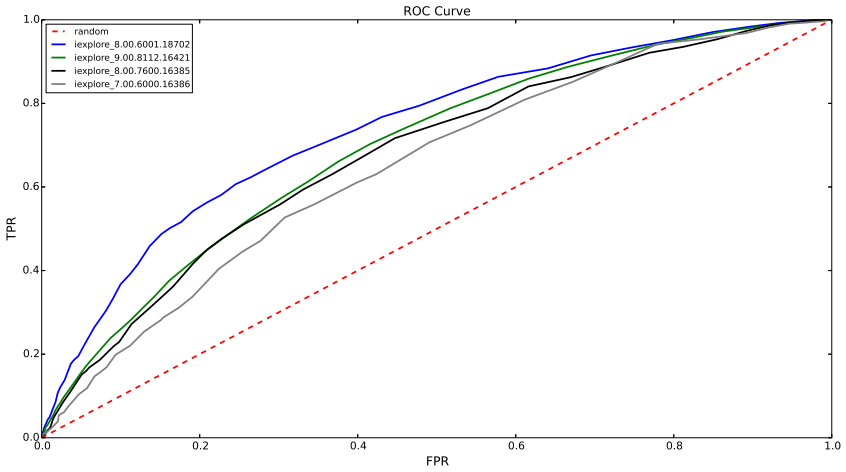

Figure 5.3: IE Roc Curve Sep 2012 (Train/Test Separate By MachineID)

sifier, especially for the case of version 8.00.6001.18702 and 9.00.8112.16421. As for version 8.00.6001.18702, we can achieve around 80% of true positive ratio with false positive rate of 40%.

The second question we want to answer is how consistent the rules are over machines. In order to do this, we divide the data into training and testing datasets based on machine ID. We also separate the data into the same portion of 60% training and 40% testing as the experiment with time. We tried different threshold for minimum confidence when we select the rules. As we decrease the threshold, we are able to cover more crashes, but the false positive number also increases.

For Internet Explorer, we plot Receiver Operating Characteristic(ROC) curve for four most recently used versions in Figure 5.3. Minimum confidence threshold is tuned to get different true positive rate and false positive rate.

The curves look very similar to the experiment in Figure 5.2.

With the richness of machines, we are able to perform a cross validation in terms of machine IDs.

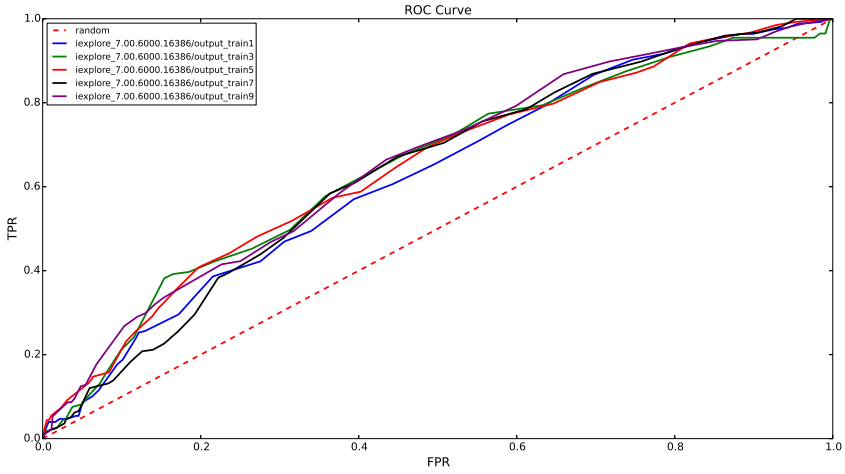

Figure 5.4: IE_7.00.6000.16386 Roc Curve Cross Validation

We do a five fold cross validation. We separate the data into five groups, each

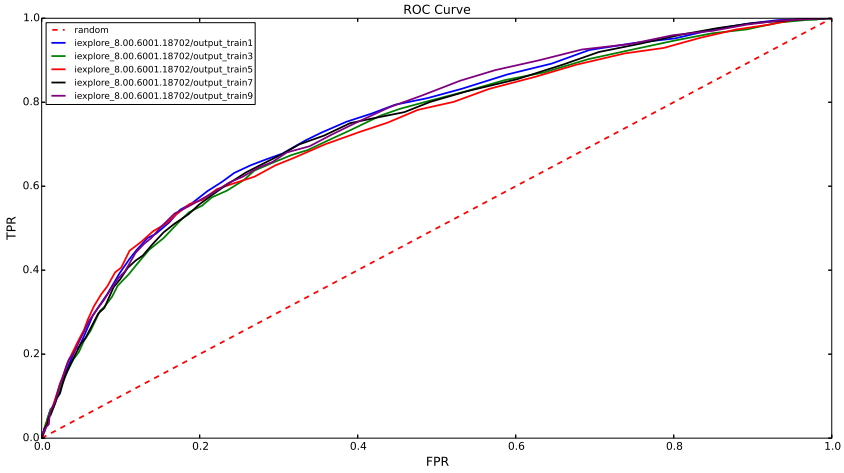

Figure 5.5: IE_8.00.6001.18702 Roc Curve Cross Validation

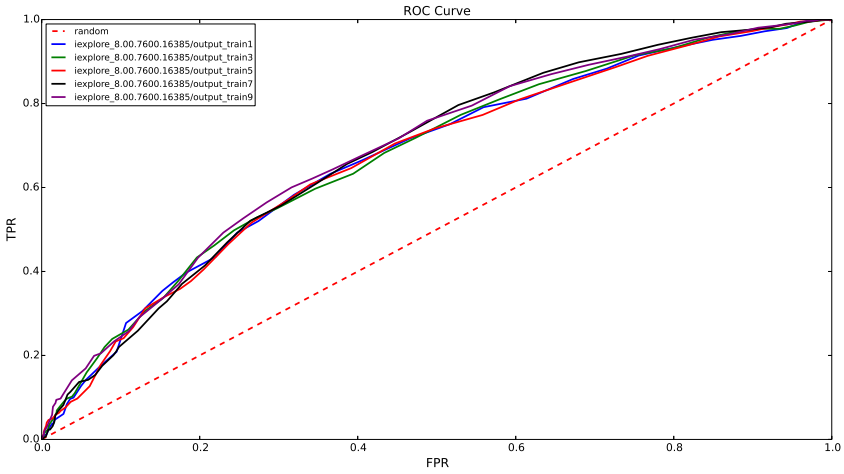

Figure 5.6: IE_8.00.7600.16385 Roc Curve Cross Validation

cover 20% of the data. And the five groups have no machines IDs in common. This is done having the machine ID mod 10. For one specific Internet Explorer version, we take one of the groups as testing dataset each time and four others as training

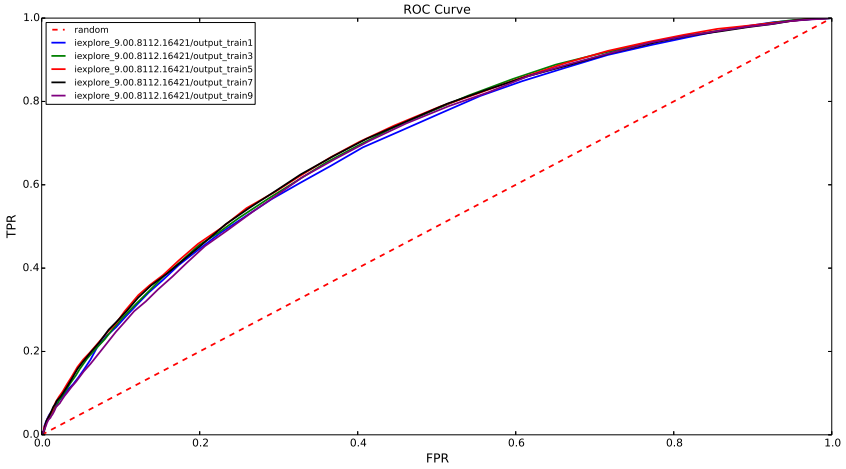

Figure 5.7: IE_9.00.8112.16421 Roc Curve Cross Validation

dataset. Thus we will have five roc curves for one single Internet Explorer version. They are presented in Figure 5.4, 5.5, 5.6, 5.7.

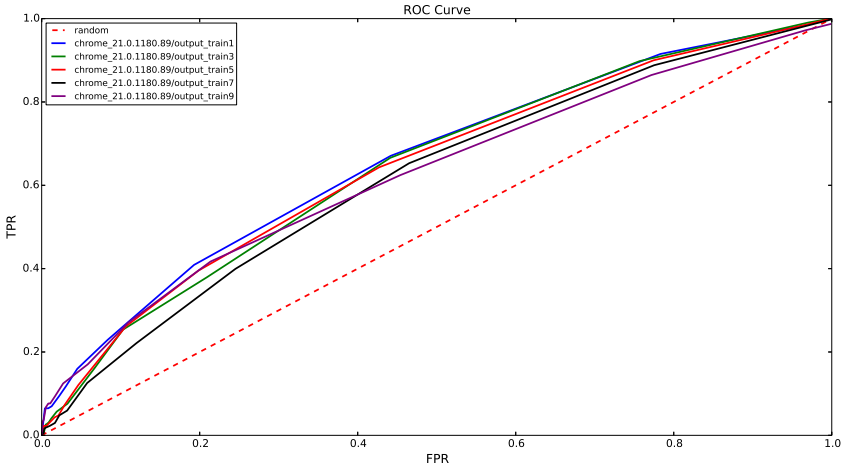

Figure 5.8: Chrome21 Roc Curve Cross Validation

We also did the same cross validation for chrome 21.0.1180.89 and firefox

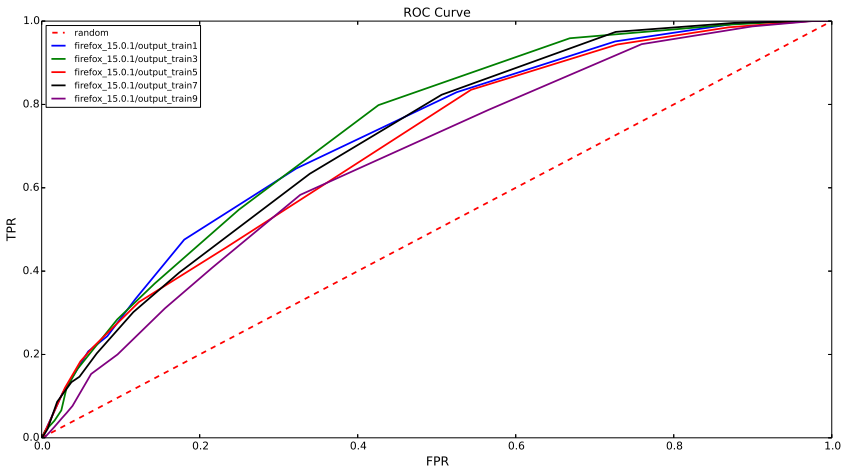

Figure 5.9: Firefox 15 Roc Curve Cross Validation

15.0.1, which are the most commonly used chrome and firefox version in our dataset. They are shown in figure 5.8, 5.9. The result shows that we are able to achieve very similar prediction results and better than the random classifier.

The dataset for the training and testing is very unbalanced and we have far more cases with no crash than cases with crash. We do another cross validation on a balanced dataset by sampling the cases with no crash. Figure 5.10 shows the results for the Internet Explorer Versions. And it is very similar to that of unbalanced cases. To illustrate this, we calculate the average area under the ROC curve for each version in both the unbalanced dataset and balanced dataset. The results are shown in table 5.9.

In this set of training and testing as well as the cross validation, we can see that adwares and browser plugins play an important role in the crash of internet browsers. And the prediction result is consistent across time and machines.

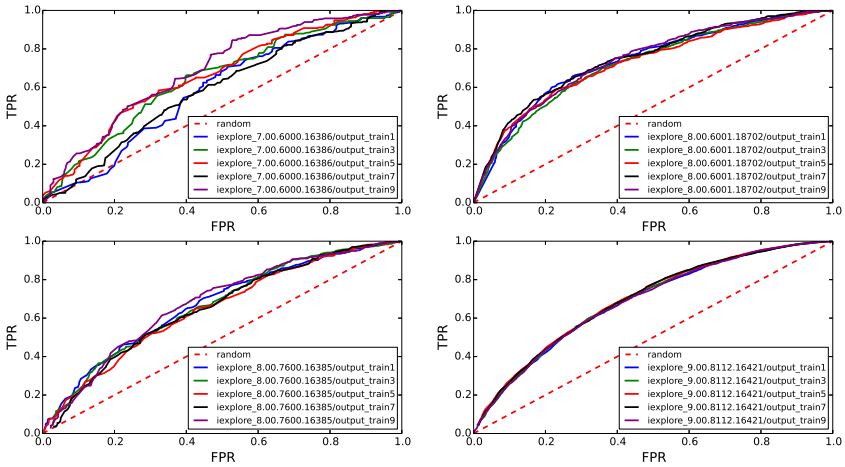

Figure 5.10: Internet Explorer Versions Cross Validation(Balanced)

Table 5.9: Area Under ROC Curve(Internet Explorer)

Version	Area
Unbalanced	
7.00.6000.16386	0.64
8.00.6001.18702	0.74
8.00.7600.16385	0.68
9.00.8112.16421	0.69
Balanced	
7.00.6000.16386	0.63
8.00.6001.18702	0.73
8.00.7600.16385	0.66
9.00.8112.16421	0.69

5.7.3 Likelihood Ratio for Rules

We calculate the likelihood ratio for the rules from each of the Internet Explorer versions. So what is the meaning of the likelihood ratio in terms of the rules we find out? For example, in the case of Internet Explorer version 9.00.8112.16421 crash in September 2012, we have a rule $mwssvc**1,0,0,5 => crash$ with confidence 20% and support 0.87% which tops likelihood ratio, but only ranked 523^{th} if we use confidence. The confidence says in every 100 cases where we spot the application $mwssvc$_1.0.0.5 together with Internet Explorer 9.00.8112.16421, we will see 20 cases that lead to the crash. And the likelihood ratio is 1348. This likelihood ratio value means that two two events, use of $mwssvc$1.0.0.5 and crash of Internet Explorer 9.00.8112.16421 are $e^{0.5*1348} = 5*10^{292}$ times more likely to be together than random occurrence. According to virusTotal [25], this is a very notorious adware popup with 16/57 detection rate from vendors.

We also find that rules are more consistent between the training and testing dataset. For the one item rules obtained from training set and testing set in September 2012 data, we calculate the Jaccard index between them for the top 100 rules. Jaccard index is a statistic used for comparing the similarity and diversity of sample sets. And it is defined as the size of the intersection divided by the size of the union of the sample sets.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{5.1}$$

We compare the one-item rules ranked by confidence and the case ranked by

likelihood ratio. The result is shown in table 5.10. We can find that likelihood ratio gives a more consistent results between the training and testing dataset as the Jaccard index value is much larger.

This shows that the ranking by the rules provide a better explanation of the crashes.

We then use the likelihood ratio value to perform tests to see if any rules are rejected by the test. The likelihood ratio follows a χ^2 distribution. We set the confidence level to be $\alpha = 0.005$, and the critical value is 7.88.

We are able to eliminate unnecessary rules whose likelihood ratio is below the critical value. And fortunately, the rules with high confidence all pass the test. For example, for Internet Explorer version 9.00.8112.16421, we have 6445 one-item rules and 255840 two-item rules. Among the one-item rules, confidence level ranges from 0.01 to 0.473. Among the two-item rules, confidence level ranges from 0.01 to 0.733. And rules with confidence higher than 0.18 all pass the test.

Thus with rules ranked by likelihood ratio gives a better balance between the support and confidence and enables us to achieve the same prediction results with much less rules. That is to say, it helps us to find the most important rules.

Table 5.10: Jaccard Index for One Item Rules in Train and Test Sep 2012

Fold	J-Index(Confidence)	J-Index(Likelihood)
Internet Explorer 9.00.8112.16421		
1	0.117	0.563
2	0.117	0.471
3	0.190	0.527
4	0.176	0.515
5	0.130	0.613
Internet Explorer 8.00.7600.16385		
1	0.058	0.235
2	0.047	0.220
3	0.026	0.220
4	0.026	0.212
5	0.031	0.136
Internet Explorer 8.00.6001.18702		
1	0.015	0.410
2	0.020	0.418
3	0.058	0.449
4	0.042	0.449
5	0.042	0.429
Internet Explorer 7.00.6000.16386		
1	0.051	0.187
2	0.027	0.100
3	0.034	0.158
4	0.045	0.147
5	0.030	0.131

Chapter 6: Conclusion and Future Work

In this work, we used association rule mining to study the cause of software crashes and our focus is on the software environment. The data we work on is a sample of 600,000 machines and 16,000 applications from Worldwide Intelligence Network Environment(WINE) from Symantec. Our main goal is to use the association rules to explain the crashes of certain commonly used softwares.

To obtain the data that we need, we mainly used the process runtime table and faulting applications table from WINE and do the aggregation and propagation to match the instances. After dichotomizing the numerical data, we feed the data to both Apriori(implemented on Spark by us) and frequent pattern tree growth(implemented by Mahout on Hadoop). For the frequent pattern tree growth, we find out that we are not able to push the minimum support low enough because of candidate set explosion. Thus, we put our emphasis on Apriori and mined out the rules for a couple of commonly used applications. We mainly worked on browsers since in our dataset they are the ones that crashed the most often and have the most number of crashes. We also take a look at other categories of applications like Microsoft Office, etc. But in general, their crash ratio is very low, implying that they are fairly stable.

We find out that the adwares and browser plugins are a main cause of Internet browsers, especially Internet Explorer. And we try to do a five fold cross validation by separating the machines by their IDs into 5 groups. We used roc curves to show the results. The figures suggest that we are able to predict the crashes to some extent, but not very impressive. The reason behind this would be the complexity of stability of softwares. Some other reasons besides environment might be self-bugginess, resource exhaustion, compatibility of platforms, etc.

This work provides a lot of valuable future directions. First, it is very interesting to see whether the plugin architecture itself is problematic. We can take at other softwares like IDEs which have the same plugin architecture. Secondly, we can examine at a deeper level for the cause of crashes like dynamic linked libraries(dlls). Thirdly, the adwares and browser plugins itself is very interesting to look at for browsers since most people must have experienced those annoying software. It will significantly enhance the user experience of browsers.

Appendix A: Association Rules

Table A.1: Internet Explorer 11.00.9600.16428 One Item Rules(Jul 2014)

Antecedent	Confidence	Support
idmsq	49.18%	0.073%
smu_2.1.0.92	47.92%	0.056%
datamngrcoordinator_5.0.0.13251	47.31%	0.107%
datamngrui_5.0.0.13251	47.31%	0.107%
gentray	44.64%	0.061%
severeweatheralertsapp_1.0.9.0	43.04%	0.083%
datamngrcoordinator_5.0.0.13277	42.86%	0.102%
datamngrui_5.0.0.13277	42.86%	0.102%
shopathomehelper_7.0.3.15	42.65%	0.07%
shopathomewatcher_7.0.3.15	42.03%	0.07%
adblock_1.0.0.0	42.03%	0.07%
smi64	41.51%	0.053%
smi32	40.68%	0.058%
pctechhotline_3.0.0.4	40%	0.049%

Table A.2: Internet Explorer 11.00.9600.16428 Two Item Rule(Jul 2014)

Antecedent	Confidence	Support
flashutil64_activex_14.0.0.145 severeweatheralertsapp_1.0.9.0	62.86%	0.053%
versioncheck hpsaobjutil7_1.0.0.30	61.77%	0.051%
flashutil64_activex_14.0.0.145 datamngrcoordinator_5.0.0.13251	61.54%	0.078%
flashutil64_activex_14.0.0.145 datamngrui_5.0.0.13251	61.54%	0.078%
googletoolbaruser32_7.5.5111.1712 datamngrcoordinator_5.0.0.13277	60.61%	0.049%
googletoolbaruser32_7.5.5111.1712 datamngrui_5.0.0.13277	60.61%	0.049%
flashutil64_activex_14.0.0.145 severeweatheralerts_1.21.0.0	57.90%	0.053%
adblock_1.0.0.0 monitor_7.0.0.373	56.76%	0.051%
flashutil64_activex_14.0.0.145 datamngrcoordinator_5.0.0.13277	55.10%	0.066%
flashutil64_activex_14.0.0.145 datamngrui_5.0.0.13277	55.10%	0.066%
flashutil64_activex_14.0.0.145 shopathomehelper_7.0.3.15	53.85%	0.051%
monitor_7.0.0.373 autocare_7.0.0.19	53.49%	0.056%

Table A.3: Internet Explorer 11.00.9600.16384 One Item Rules(Jul 2014)

Antecedent	Confidence	Support
plsapp_2.2.7.6	54.05%	0.195%
pureleadstray_2.0.17	52.63%	0.195%
servicelocator_21.8.0.261	51.90%	0.4%
toolbar_21.8.0.261	51.90%	0.4%
pureleadssvc_2.0.17	51.28%	0.195%
pureleads.service_2.0.6.0	51.28%	0.195%
couponprinterservice_6.0.1.0	36.84%	0.273%
yahoomessenger_11.5.0.0228	36.54%	0.371%
ymsgr_tray_11.5.0.0228	36.364%	0.273%
reminderhelper_2.0.1.1	35.71%	0.195%
ytbb_2014.6.3.01	35.51%	0.371%
versioncheck	33.784%	0.244%
solutolauncherservice_1.3.1067.1	32.91%	0.254%
yahooauservice_1.0.0.53	32.215%	1.407%
soluto_1.3.1193.1	31.65%	0.244%
solutoservice_1.3.1193.1	30.67%	0.225%

Table A.4: Internet Explorer 11.00.9600.16384 Two Item Rules(Jul 2014)

Antecedent	Confidence	Support
mdnsresponder_3.0.0.10 plsapp_ 2.2.7.6	68.97%	0.195%
pureleadssvc_2.0.17 mdnsresponder_3.0.0.10	64.52%	0.195%
pureleadstray_2.0.17 mdnsresponder_3.0.0.10	64.52%	0.195%
pureleads.service_2.0.6.0 mdnsresponder_3.0.0.10	64.52%	0.195%
googlecrashhandler64_1.3.24.15 servicelocator_21.8.0.261	59.46%	0.215%
googlecrashhandler64_1.3.24.15 toolbar_21.8.0.261	59.46%	0.215%
msfeedssync_11.00.9600.16384 plsapp_2.2.7.6	58.82%	0.195%
cleanmgr_6.3.9600.17031 toolbar_21.8.0.261	58.49%	0.303%
cleanmgr_6.3.9600.17031 servicelocator_21.8.0.261	58.49%	0.303%

Table A.5: Internet Explorer 10.00.9200.16384 One Item Rules(Jul 2014)

Antecedent	Confidence	Support
makecert_6.1.7600.16385	49.18%	0.56%
hotkeyutility_3.0.3001.0	42.424%	0.784%
stacsv64_1.0.6417.0	41.063%	1.587%
virtualdrive_8.0.1.1926	40.146%	1.027%
hpservice_4.2.7.1	40.132%	1.139%
couponprinterservice_6.0.1.0	40.0%	0.373%
sttray64_1.0.6417.0	38.5%	1.438%
yahooauservice_1.0.0.53	36.429%	1.905%
gamesappintegrationservice_4.0.31.21	36.364%	0.373%
hpmsgsvc_3.0.3.0	35.843%	2.222%
power2goexpress8_8.0.3.2527	34.783%	0.448%
communicator	34.118%	0.542%
nasvc_11.0.0028	34.0%	1.587%
atieclxx_6.14.11.1126	33.454%	5.191%
fuel.service_1.0.0.0	33.381%	4.388%
atiesrxx_6.14.11.1126	33.373%	5.229%
ytbb_2014.6.3.01	32.787%	0.373%
appintegrator64_1.0.6.21	32.394%	0.859%
39barsvc_1.0.1.0	32.353%	0.411%

Table A.6: Internet Explorer 10.00.9200.16384 Two Item Rules(Jul 2014)

Antecedent	Confidence	Support
yahooauservice_1.0.0.53 hpmsgsvc_3.0.3.0	67.742%	0.392%
yahooauservice_1.0.0.53 rndlresolversvc	67.742%	0.392%
fuel.service_1.0.0.0 appintegrator64_1.0.7.183	62.0%	0.579%
glcnd_6.2.9200.20780 fuel.service_1.0.0.0	57.143%	0.373%
fuel.service_1.0.0.0 virtualdrive_8.0.1.1926	56.757%	0.784%
hotkeyutility_3.0.3001.0 ccc_3.5.0.0	56.41%	0.411%
yahooauservice_1.0.0.53 riconman_1.5.0.0	56.25%	0.672%
mom_2.0.0.0 hotkeyutility_3.0.3001.0	55.0%	0.411%
stacsv64_1.0.6417.0 fuel.service_1.0.0.0	54.962%	1.345%
fuel.service_1.0.0.0 epowersvc_7.0.3006.0	54.717%	0.542%
epowertray_7.0.3006.0 fuel.service_1.0.0.0	54.717%	0.542%
sttray64_1.0.6417.0 fuel.service_1.0.0.0	54.622%	1.214%
yahooauservice_1.0.0.53 hpwmisvc_3.0.1.0	53.521%	0.71%
makecert_6.1.7600.16385 flashutil_activex_14.0.0.145	53.191%	0.467%
yahooauservice_1.0.0.53 atiesrxx_6.14.11.1126	52.874%	0.859%
yahooauservice_1.0.0.53 atieclxx_6.14.11.1126	52.874%	0.859%
fuel.service_1.0.0.0 hotkeyutility_3.0.3001.0	52.727%	0.542%

Table A.7: Internet Explorer 9.00.8112.16421 One Item Rules(Jul 2014)

Antecedent	Confidence	Support
mahostservice_1.0.0.1	28.767%	0.439%
39srchmn_1.0.1.0	26.667%	0.251%
node_0.10.4	26.316%	0.209%
appintegrator64_1.0.7.183	25.362%	0.366%
14barsvc_1.0.0.9	25.0%	0.24%
iaanotif_8.7.0.1007	24.8%	0.324%
iaantmon_8.7.0.1007	23.944%	0.355%
pccmservice_7.3.0.11	23.913%	0.23%
14brmon_1.0.0.1	23.81%	0.209%
searchprotection_2009.1.30.1	23.009%	0.272%
39barsvc_1.0.1.0	22.485%	0.397%
39brmon_1.0.2.0	22.0%	0.345%
hpwucli_4.0.14.1	21.739%	0.261%
datamngrui	21.56%	0.491%
bingbar_7.3.132.0	20.812%	0.429%
acrobroker_9.5.5.316	20.712%	0.669%
pcmagent_5.0.0.0	20.588%	0.22%
bingapp_7.3.132.0	20.398%	0.429%
inbox_2.0.1.90	20.093%	0.45%

Table A.8: Internet Explorer 9.00.8112.16421 Two Item Rules(Jul 2014)

Antecedent	Confidence	Support
yahooauservice_1.0.0.53 inbox_2.0.1.90	48.889%	0.23%
acrobroker_9.5.5.316 agcp_5.1.30214.0	38.889%	0.22%
mahostservice_1.0.0.1 msfeedssync_9.00.8112.16561	36.765%	0.261%
yahooauservice_1.0.0.53 agcp_5.1.30214.0	33.088%	0.47%
wincal_6.0.6000.16386 rndlresolversvc_	30.38%	0.251%
googlecrashhandler_1.3.24.15 39srchmn_1, 0, 1, 0	30.0%	0.22%
mwssvc_1, 0, 0, 5 msfeedssync_9.00.8112.16561	30.0%	0.22%
googlecrashhandler_1.3.24.15 iaanotif_8.7.0.1007	29.703%	0.314%
hphc_service_3.1.10.1 msfeedssync_9.00.8112.16561	29.63%	0.251%
flashutil32_activex_14,0,0,145 searchprotection_2009, 1, 30, 1	29.268%	0.251%
hphc_service_3.1.9.1 agcp_5.1.30214.0	29.213%	0.272%
flashutil32_activex_14.0.0.145 mahostservice_1.0.0.1	29.126%	0.314%
logon_6.0.6000.16386 datamngrui	29.114%	0.24%

Table A.9: Chrome 36.0.1985.125 Rules(Jul 2014)

Antecedent	Confidence	Support
dsrlte_1.3.0.0	15.493%	0.056%
dnkt	7.843%	0.072%
wrtc_1.0.0.1	7.643%	0.092%
dmwu	7.348%	0.105%
silverlight.configuration_5.1.30514.0	5.871%	0.077%
agcp_5.1.30514.0	4.439%	0.179%
skypec2cpnrsvc_7.3.16540.9015	4.375%	0.072%
updatus.18722395_runasuser	3.945%	0.051%
gameoverlayui_02.32.45.01	3.68%	0.09%
wrtc_1.0.0.1 mdnsresponder_3,0,0,10	8.661%	0.056%
mdnsresponder_3,0,0,10 dmwu	8.224%	0.064%
dnkt dmwu	7.977%	0.072%
dnkt wrtc_1.0.0.1	7.843%	0.072%
wrtc_1.0.0.1 dmwu	7.759%	0.092%
armsvc_1.701.3.3014 dmwu	7.746%	0.056%
nacl64_36.0.1985.125 apsdaemon_2.3.4.36	7.092%	0.051%
silverlight.configuration_5.1.30514.0 mdnsresponder_3,0,0,10	6.604%	0.054%
silverlight.configuration_5.1.30514.0 agcp_5.1.30514.0	6.048%	0.077%
nacl64_36.0.1985.125 flashutil_activex_14,0,0,145	5.804%	0.067%

Table A.10: Googleupdate_1.2.183.21 One Item Rules(Jul 2014)

Antecedent	Confidence	Support
evteng_15, 1, 0, 0	10.135%	0.077%
riconman_1.3.4.1	10.12%	0.152%
hpmsgsvc_2.6.3.0	10.096%	0.054%
adobecollabsync_10.1.10.18	9.955%	0.057%
ravcpl64_1, 0, 0, 639	9.881%	0.064%
igfxsrvc_8.15.10.2476	9.739%	0.106%
pdfsvc_4.0.35.2001	9.73%	0.093%
tcrdkbb_1, 0, 1, 64	9.719%	0.098%
hpmsgsvc_2.5.2.0	9.677%	0.077%
atiesrxx_6.14.11.1088	9.677%	0.108%
stacsv64_1.0.6292.0	9.653%	0.064%
syntpenh_15.2.4.4 15Dec10	9.589%	0.072%
atieclxx_6.14.11.1088	9.557%	0.106%
kenotify_2, 0, 50, 8	9.524%	0.108%
sttray64_1.0.6292.0	9.524%	0.057%
syntphelper_15.2.4.4	9.524%	0.072%
atiesrxx_6.14.11.1102	9.465%	0.059%

Table A.11: Googleupdate_1.2.183.21 Two Item Rules(Jul 2014)

Antecedent	Confidence	Support
riconman_1.3.4.1 stacsv64_1.0.6381.0	20.863%	0.075%
flashutil64_activex_14,0,0,145 biomonitor_5.1.0.495	20.721%	0.059%
hpqwmiex_6, 1, 16, 1 evteng_15, 1, 0, 0	20.588%	0.054%
hpqwmiex_6, 1, 16, 1 bthsamppalservice_15, 1, 0, 3	20.588%	0.054%
biomonitor_5.3.1.7 msfeedssync_11.00.9600.16428	20.561%	0.057%
truesuiteservice_5.3.1.7 msfeedssync_11.00.9600.16428	20.561%	0.057%
hpqwmiex_6, 1, 16, 1 zeroconfigservice_15.1.0.2	20.388%	0.054%
hpqwmiex_6, 1, 16, 1 bthssecuritymgr_15.1.0.8	20.388%	0.054%
googlecrashhandler_1.3.24.15 stacsv64_1.0.6381.0	20.313%	0.067%
googlecrashhandler64_1.3.24.15 stacsv64_1.0.6381.0	20.313%	0.067%
evteng_15, 1, 0, 0 hpsa_service_7.2.45.3	20.192%	0.054%
hpsa_service_7.2.45.3 bthsamppalservice_15, 1, 0, 3	20.192%	0.054%
googlecrashhandler_1.3.24.15 sttray64_1.0.6381.0	20.161%	0.064%
googlecrashhandler64_1.3.24.15 sttray64_1.0.6381.0	20.161%	0.064%
hpsa_service_7.2.45.3 bthssecuritymgr_15.1.0.8	20.0%	0.054%
googlecrashhandler_1.3.24.15 biomonitor_5.3.1.7	20.0%	0.054%
googlecrashhandler64_1.3.24.15 truesuiteservice_5.3.1.7	20.0%	0.054%

Table A.12: Internet Explorer_7.00.6000.16386 One Item Rules(Sep 2012)

Antecedent	Confidence	Support
gcbarsvc_1.0.0.9	46.809%	0.217%
swhelper_10.2.23	40.506%	0.316%
2jsrchmn_1.0.0.5	40.0%	0.217%
yspservice_2010.6.14.01	39.063%	0.247%
2jbarsvc_1.0.0.9	36.667%	0.217%
2jbrmon_1.0.0.1	36.667%	0.217%
mwssvc_1.0.0.1	35.593%	0.207%
selectrebatesdownload_1.0.0.3	35.366%	0.286%
searchprotection_2009.1.30.1	34.286%	0.237%
yspservice_2010.4.1.01	31.937%	0.602%
14srchmn_1.0.0.12	30.556%	0.217%
2pbarsvc_1.0.0.9	30.303%	0.197%
selectrebates_5.2.0.0	30.137%	0.434%
m3srchmn_1.0.0.5	28.358%	0.937%

Table A.13: Internet Explorer_8.00.6001.18702 One Item Rules(Sep 2012)

Antecedent	Confidence	Support
cmhelper_1.0.0.10	56.716%	0.041%
alotsettings_1.1.3.0	48.649%	0.039%
genieutils	42.647%	0.031%
gentray	42.574%	0.047%
2zbarsvc_1.0.0.9	41.935%	0.028%
v4barsvc_1.0.0.9	41.772%	0.036%
firsttime_setup	40.449%	0.039%
pmvservice_1.00.2818	39.286%	0.024%
4jbarsvc_1.0.0.9	38.462%	0.033%
jp2launcher_6.0.110.3	35.714%	0.022%
mwssvc_1.0.0.1	35.165%	0.035%
rthdvcpl_1.0.0.105	35.088%	0.022%
alotsettings_1.0.0.4	34.884%	0.065%
gtbarsvc_1.0.0.9	34.524%	0.031%

Table A.14: Internet Explorer_8.00.7600.16385 One Item Rules(Sep 2012)

Antecedent	Confidence	Support
fightersuiteservice_3.1.186.0	53.191%	0.076%
fighterstray_4.0.64.0	48.077%	0.076%
ezprint_3.15.0.0	47.619%	0.061%
flv_runnertoolbarhelper_1.0.1.0	46.939%	0.07%
firsttime_setup	46.512%	0.122%
gentray	45.556%	0.125%
coupons.comtoolbarhelper_1.0.1.0	44.444%	0.061%
optprolauncher_3.0.1.0	42.553%	0.061%
genieutils	42.466%	0.095%
mwsoemon_1.2.2.5	41.818%	0.07%
64srchmn_1.0.0.5	41.667%	0.061%
starterw3i_1.00.0001	41.176%	0.064%
winzipbartoolbarhelper_1.0.1.0	40.984%	0.076%
mwssvc_1.0.0.4	40.351%	0.07%

Table A.15: Internet Explorer_9.00.8112.16421 One Item Rules(Sep 2012)

Antecedent	Confidence	Support
wiseconverttoolbarhelper1_1.0.1.0	48.936%	0.007%
dca-ua_1.7.0.8467	41.81%	0.03%
crextp2p_1.0.2.10	40.559%	0.018%
productivity_3.1toolbarhelper_1.0.1.0	36.667%	0.03%
a_free_ride_games_bartoolbarhelper_1.0.1.0	36.486%	0.025%
produtools_mapstoolbarhelper_1.0.1.0	33.739%	0.034%
coupons.comtoolbarhelper1_1.0.1.0	32.857%	0.007%
codec	32.843%	0.021%
aoltbserver_5.74.1.8383	32.716%	0.016%
road_runnertoolbarhelper_1.0.1.0	32.571%	0.018%
whitesmoke_us_newtoolbarhelper_1.0.1.0	32.422%	0.025%
produtools_manuals_2.1toolbarhelper_1.0.1.0	32.397%	0.06%
default_tab_search_results	32.372%	0.031%
startw3i_3.0.1.0	32.37%	0.017%
defaulttabsetup2	32.323%	0.029%

Bibliography

[1] Murphy, B. The Difficulties Of Building Generic Reliability Models for Software In *Empirical Software Engineering*, 2012

[2] Edmund B. Nightingale and John R Douceur and Vince Orgovan Cycles, Cells and Platters: An Empirical Analysis of Hardware Failures on a Million Consumer PCs In *Proceedings of EuroSys 2011, Awarded "Best Paper"*, April 2011.

[3] Schroeder, Bianca and Gibson, Garth A. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, San Jose, CA, 2007

[4] Schroeder, B., Pinehiro, E., and Weber, W.-D. Dram errors in the wild: a large-scale field study. In *SIGMETRICS*, 2009

[5] Marathe, M. and Cukier, M In *Proceedings of the IEEE 21st International Symposium on Software Reliability*, 2011

[6] Littlewood, Bev Software Reliability Model for Modular Program Structure In *IEEE Transactions on Reliability*, 1979

[7] Chou, Andy and Yang, Junfeng and Chelf, Benjamin and Hallem, Seth and Engler, Dawson An Empirical Study of Operating Systems Errors In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, 2001

[8] D.R. Engler, B. Chelf, A. Chou, and S. Hallem. Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions. In Proceedings of Operating Systems In *Design and Implementation (OSDI)*, September 2000

[9] Palix, Nicolas and Thomas, Gaël and Saha, Suman and Calvès, Christophe and Lawall, Julia and Muller, Gilles Faults in Linux: Ten Years Later In

Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2011

[10] Reliability Growth Of Software Products Reliability Growth Of Software Products In *Institute of Electrical and Electronics Engineers, Inc.*, 2004

[11] Tyler Harter, Chris Dragga, Michael Vaughn, Andrea C. Arpaci Dusseau, Remzi H. Arpaci Dusseau A File is Not a File: Understanding the I/O Behavior of Apple Desktop Applications In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011

[12] Christian Bird, Venkatesh-Prasad Ranganath, Thomas Zimmermann, Nachiappan Nagappan, and Andreas Zeller Extrinsic Influence Factors in Software Reliability: A Study of 200,000 Windows Machines In *Proceedings of the 36th International Conference on Software Engineering* , 2014

[13] Microsoft Inc. Windows Customer Experience Improvement Program In *http://technet.microsoft.com/en-us/library/ee126127(WS.10).aspx.*, 2011

[14] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington D.C., May 1993.

[15] Rakesh Agrawal, Ramakrishnan Srikant Fast algorithms for mining association rules In *Proc 20th Int'l Conf Very Large Database*, Santiago, Chile, Sep 1994

[16] Ashoka Savasere,Edward Omiecinski,Shamkant B. Navathe An efficient algorithm for mining association rules in large databases In *Proc of the 21th International Conference on Very Large Database*, Zurich, Switzerland, May 1995

[17] Ramakrishnan Srikant, Rakesh Agrawal Mining Generalized Association Rules In *VLDB '95 Proceedings of the 21th International Conference on Very Large Data Bases*, San Francisco, CA, 1995

[18] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach In *Data Mining and Knowledge Discovery, Volume 8 Issue 1*, January 2004

[19] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, Edward Y. Chan Pfp: parallel fp-growth for query recommendation In *Proceedings of the 2008 ACM conference on Recommender systems*,2008

[20] Hadoop Tutorial https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[21] Hongjian Qiu, Rong Gu, Chunfeng Yuan, Yihua Huang YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark In *International Parallel & Distributed Processing Symposium Workshops*, 2014

[22] Liu, Guimei and Zhang, Haojun and Wong, Limsoon Controlling False Positives in Association Rule Mining In *Proc. VLDB Endow.*, 2011

[23] Christopher D. Manning and Hinrich Schtze Foundations of Statistical Natural Language Processing

[24] Alvarez, Sergio A. Chi-squared computation for association rules: preliminary results. Technical Report BC-CS-2003-01, Boston College.

[25] www.virustotal.com