Easier Said Than Done: Diagnosing Misconfiguration via Configuration Constraints Analysis

A Study of the Variance of Configuration Constraints in Source Code

Shulin Zhou
National University of Defense Technology
P. R. China
zhoushulin@nudt.edu.cn

Shanshan Li
National University of Defense Technology
P. R. China
shanshanli@nudt.edu.cn

Xiaodong Liu
National University of Defense Technology
P. R. China
liuxiaodong@nudt.edu.cn

Xiangyang Xu
National University of Defense Technology
P. R. China
xuxiangyang11@nudt.edu.cn

Si Zheng
National University of Defense Technology
P. R. China
si.zheng1009@gmail.com

Xiangke Liao
National University of Defense Technology
P. R. China
xkliao@nudt.edu.cn

Yun Xiong
Fudan University
P. R. China
yunx@fudan.edu.cn

ABSTRACT

Misconfigurations have drawn tremendous attention for their increasing prevalence and severity, and the main causes are the complexity of configurations as well as the lack of domain knowledge for software. To diagnose misconfigurations, one typical approach is to find out the conditions that configuration options should satisfy, which we refer to as configuration constraints. Current researches only handled part of the situations of configuration constraints in source code, which provide only limited help for misconfiguration diagnosis. To better extract configuration constraints, we conduct a comprehensive manual study on the existence and variance of the configuration constraints in source code from five pieces of popular open-source software. We summarized several findings from different aspects, including the general statistics about configuration constraints, the general features for specific configurations, and the obstacles in extraction of configuration constraints. Based on the findings, we propose several suggestions to maximize the automation of constraints extraction.

CCS CONCEPTS

• **General and reference** → **Surveys and overviews**; • **Software and its engineering** → **Maintaining software**;

EASE'17, Karlskrona, Sweden

DOI: http://dx.doi.org/10.1145/3084226.3084276

KEYWORDS

Misconfiguration, misconfiguration diagnosis, configuration constraints

ACM Reference format:
Shulin Zhou, Shanshan Li, Xiaodong Liu, Xiangyang Xu, Si Zheng, Xiangke Liao, and Yun Xiong. 2017. Easier Said Than Done: Diagnosing Misconfiguration via Configuration Constraints Analysis . In *Proceedings of EASE'17, Karlskrona, Sweden, June 15-16, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3084226.3084276

1 INTRODUCTION

In recent years, misconfigurations have drawn tremendous attention for their increasing prevalence and severity. As Barroso and Hoelzle [1] mentioned, misconfiguration were the second major cause of service outage in Google's main services, counting nearly 28 percentage. Furthermore, Rabkin and Katz [6] indicated that, considering both the reported clients' failure cases number and the total technique support time, misconfiguration was the leading cause of Hadoop cluster failures. Many other works [7, 8, 12] also indicate such misconfiguration problems, and many researches [2, 9, 10] have been focusing on it from different views.

The reasons for misconfigurations are mainly twofold. On one hand, users rarely have the knowledge of the software and its configuration options. For instance, there would be hundreds of configuration options in current database server [4] and web server [3], which would be a great challenge for rookies and even experienced users to correctly set them. On the other hand, the complexity in the procedure of configuration is also an obstacle. Every single configuration option's value should satisfy a valid range, and some structural module should be well formatted. In addition, there might be some complex relationship between different configuration options. For example, in PostgreSQL, the value of configuration option

"superuser_reserved_connections" must be less than the value of configuration option "max_connections", otherwise the daemon of PostgreSQL would exit.

When misconfiguration occurs, a typical diagnosis approach is to check whether the current settings of the configuration options are set in the correct value range. Thus, if we could extract the conditions that configuration options should satisfy, which is hereinafter referred to as configuration constraints, the users' configuration as well as the diagnosis and repair of misconfiguration will be much benefited.

As important references of the software, user manuals and official documentations may record the configuration constraints, but it is hard for common users to find them in thousands pages of documents. More seriously, some documentation of the software may not be updated timely or even not contain the constraints of the configuration options at all. So extra efforts are needed to obtain the configuration constraints.

To obtain configuration constraints, current researches have made much efforts. After summarizing the three commonly used mapping structures, SPEX [11] extracts configuration constraints from source code based on the specific patterns, such as if-statement checking and lib-function calling. Then, based on those configuration constraints, some fault injections are made and some suggestions are provided for configuration design and implementation. Rabkin et al.[5] also try to extract some information from source code. Focusing on key-value model, they present a static analysis that extracts a list of configuration options for a program based on a few manual labeling. Besides, it [5] also can infer the type for most configuration options to reduce the burden of maintaining configuration documentations and configuration debugging. EnCore [13] uses machine learning algorithm to extract configuration constraints from thousands of user's configuration files, which may extract some hidden constraints between different configuration options or even environment variables, but it requires predefined constraints' templates to reduce the search space, limiting the usage of common users without domain knowledge.

Based on the conjectures that there must be some checking modules for configuration, and the results of existing work, we thought there must be plenty of explicit configuration constraints in source code. However, after comprehensive studies in real-world source code, we found it is often not the case. We manually studied five pieces of popular open-source software, and found that the formats of configuration constraints are varied in different kinds of software, not always in if-statement or switch-case-statement. Then, considering the fact that the configuration constraints in some software are context-sensitive with too much semantic information, it is barely impossible to extract by simple program analysis. What's more, as a consequence of bad configuration design or developers' faults, there might be no configuration constraints at all.

Faced with such situations, in order to fulfill the extraction of configuration constraints for misconfiguration diagnosis, we did a comprehensive manual study on the existence and variance of the configuration constraints in source code. Based on the overall results, several findings from different aspects are summarized, including the general statistics of configuration constraints, the general features of specific constraints and the obstacles in extracting configuration constraints. Furthermore, we also proposed several automation suggestions about constraints extraction.

The rest of this paper is organized as follows. Section 2 introduces the methodology we used. Section 3 describes the main findings about configuration constraints in source code. We propose several strategies to automatically extract configuration constraints from source code in Section 4. Finally, some conclusions are mentioned in Section 5.

2 METHODOLOGY

This section describes our methodology for analyzing the configuration constraints in source code. At present, there is no mature tools or approaches to extract configuration constraints from source code. So in this section, we read the source code by manual efforts to find the configuration constraints from source code. In detail, we first search the official documentations and manuals to collect all the configuration options of the software. Then, based on the observations in [14] and code searches, we got the configuration variables that related to configuration options. Through some sampling statistical analysis and summarization, we hold the opinion that configuration variables are usually directly used for the purpose of control the runtime flow. Based on that, we manually trace every use of the configuration variables to dig out the potential configuration constraints. The targetting configuration constraints will be introduced in the following subsetion.

2.1 Data Set

To cover possibility as far as we can, we choose five typical pieces of software to do the study, namely MySQL, Apache Httpd, Redis, Postfix, PostgreSQL, which are all widely used and ranking top in SourceForge or OSChina. For accuracy and effectiveness, all the pieces of studied software are the newest stable version.

2.2 Study Methodology

In order to figure out the existence of configuration constraints in source code, we should know what kinds of types that configuration options mainly are at first. To address this problem, we did a detailed survey on configuration types in official documentations and source codes, and found that configuration options may have various kinds of types and formations, but their basic types are commonly in range of numeric type, string type, enumeration type and complex type. The specific explanations of these configuration basic types are list in Table 1.

Based on the classification of configuration basis types mentioned above, as well as some reference in SPEX [11], we mainly focus on the configuration constraints listed in Table 2, considering the fact that those constraint types represent the mainly manifestations of configuration constraints in source code.

The exact meanings of the constraints are the same as SPEX [11] declared except for enumeration and value control constraints types. In this paper, we take configuration options with type of enumerate, boolean as constraint *enumeration*. As for value control, we only take Boolean configuration as the controller configurations, because there are plenty of usages that different configuration options are used in one condition to control the different program

Table 1: Explanation of configuration types

Configuration Types		Explanations
Simple Type	Numeric Type	Includes integer and float types.
	Enumeration Type	Include enumerations, Boolean, and some macros used as enumeration.
	String Type	Other common strings, such as file path, user name, etc.
Complex Type		Structural configuration options and software-specific encapsulated configuration options.

Table 2: Configuration constraints and their explanations

Constraint Types	Explanations
Semantic Type	The semantic type of the configuration option
Value Range	The value range of a numeric configuration option should satisfy
Enumeration	The value space of a enumeration configuration option should satisfy
Value Control	The usage of a configuration option relies on another configuration option
Multi-value relationship	The value relationship of two different configuration options should satisfy

runtime routine in different situations, which obviously are not configuration constraints.

2.3 Threats to Validity and Limitations

Considering the limitation of manual analysis, our study is subject to a validity problem, which lies on the representativeness of software we choose and the constraints type we define.

To address the former, we selected diverse open-source software in terms of functionality, including web server, database server, mail transfer agent, all of which are widely used in their product categories. As for the latter, we have already found the common configuration types based on detailed survey, and thus defined our target configuration constraints sets considering the fact that those constraint types represent the mainly manifestations of configuration constraints in source code. Of course, there might be some kinds of configuration constraints not belonging to these lists, but those constraints might also be difficult to extract automatically, or even by manual analysis. Therefore, the configuration constraint types we investigated could represent the common situations.

3 FINDINGS IN REAL-WORLD SOURCE CODE

This section describes the main findings of configuration constraints in source code. Our findings mainly focus on the current existence of configuration constraints, the limitations of current researches and inspirations about configuration extracting automation.

Table 3: Different proportion of constraints in different software

Software	Proportion of configuration constraints	Proportion of configuration constraints that SPEX could extract
PostgreSQL	87.5%	44.3%
Redis	92.1%	84.3%
MySQL	53%	4.2%
Httpd	45.3%	33.8%
Postfix	49.5%	7.4%

3.1 General Statistics about Configuration Constraints

Finding 1: On average, 64% of the configuration constraints can be extracted through program analysis with various code shape, while current works mainly consider the if-statement situations.

According to our manual study of configuration constraints, it is obvious that different pieces of software have different level of configuration checking, representing as the proportion of configuration constraints existing in the source code. The detailed statistics are listed in Table 3. From the second column of Table 3, we can see that Redis has the highest proportion of configuration constraints as 92.1% while Httpd has the lowest proportion, 45.3%. With further analysis, we found that Redis has structured if-statement to check most of its configuration options, while Httpd uses a series of set-functions to assign the configuration value to variable, during which the assign procedures are too complex to extract our constraints.

More over, when compared with numbers of configuration constraints that current work could extract, we found that there are some gaps based on the last column of Table 3. The causes for these gaps are that the formations of configuration constraints are varied in different pieces of software, which will be detailed introduced in Finding 3 and 4. Take SPEX [11] as an example, it mainly considers the condition in if-statement as constraints. In implementation, SPEX first locates the if-statements where configuration options used as judging conditions and extract the configuration-related conditions as its ranges. Then, if in the branch block, the program exits, aborts, returns error code, or resets the parameter, SPEX treats the range as invalid. Otherwise, it is valid. Thus, SPEX could extract the configuration constraints. However, in our study, there are few configuration constraints could be extracted from if-statements, so the proportion is lower.

Finding 2: Numeric configuration options have highest proportion of constraints, while configuration options in string type is the lowest except for complex type.

When considering with basic types that configuration options have, there are also some distinctions. Just as Table 4 shows, the configuration options in numeric type (i.e. integer and float) have a higher proportion of constraints in general. As for configuration options with string type, the proportion with constraints is generally lower. After manual analysis, we hold the view that the

Table 4: Constraint proportions of configuration options with different basic types

Software	Numeric	Enumeration	String	Complex
PostgreSQL	100%	100%	0%	-
Redis	100%	100%	50%	100%
MySQL	100%	100%	6.4%	0%
Httpd	64.3%	100%	22.2%	0%
Postfix	100%	100%	8.3%	-

configuration option in string type have flexible format. Except for few special types such as file path and directory, other kinds of configuration options' values are hard to check. When it comes to complex types configuration, due to the particularity of its meaning and formation in different pieces of software, it is barely possible to extract common configuration constraints with rich semantic context. Therefore, the constraint proportion of configuration options with complex type are extremely low.

3.2 General Features in the Existence of Configuration Constraints

Finding 3: Configuration constraints have various manifestations, rather than if-condition statement.

Current works mainly focus on if-condition statement to extract configuration constraints. In some circumstances it makes sense. However, after manual analysis, we found that it is only account for a small percentage. On the other hand, there are various manifestation of configuration constraints, especially for numeric configuration options. Considering the fact that those constraints are existing in the structures that configuration options map with relevant program variables, so we call them mapping constraints. In summary, we found that MySQL, PostgreSQL and Postfix have plenty of these kinds of constraints. For instance, in Fig. 1, MySQL use object declaration to announce the constraints, especially with some macros for readability, while PostgreSQL and Postfix use specific structure arrays to accomplish the complex jobs.

Finding 4: The enumeration constraints have various manifestations rather than simply switch-case-statement situations, but in uniform patterns.

In current works, researchers only focus on the switch-case situations to extract enumeration configuration constraints, while in manual study, we found that is often not the case. Through manually statistics shown in Table 5, it is obvious that switch-case-statement can be rarely used for configuration constraints in those pieces of software.

Furthermore, we found that there are some clustered code snippets for enumeration configuration values. As Fig. 2(a) shows, PostgreSQL uses a series of structure arrays to connect the value space and the relevant enumerated values, Redis and MySQL use the similar method to conduct this work. While in Fig. 2(a), Httpd uses Macro as Enumeration and fulfill the assignment from configuration value to program variable values.

Finding 5: The parameters of the functions that call configuration options contain the information of their semantic types.

```
/* mysql-5.7.16/sql/sys_vars.cc */
...
 static Sys_var_ulong Sys_connect_timeout(
      "connect_timeout",
      "The number of seconds the mysqld server is waiting for
          a connect "
      "packet before responding with 'Bad handshake'",
      GLOBAL_VAR(connect_timeout), CMD_LINE(REQUIRED_ARG),
      VALID_RANGE(2, LONG_TIMEOUT), DEFAULT(CONNECT_TIMEOUT),
          BLOCK_SIZE(1));
...
```

(a) MySQL-5.7.16

```
/* postgresql-9.5.6/src/backend/utils/misc/guc.c */
...
 {
   {"geqo_effort", PGC_USERSET, QUERY_TUNING_GEQO,
    gettext_noop("GEQO: effort is used to set the default
       for other GEQO parameters."),
    NULL
   },
   &Geqo_effort,
   DEFAULT_GEQO_EFFORT, MIN_GEQO_EFFORT, MAX_GEQO_EFFORT,
   NULL, NULL, NULL
 },
...
```

(b) PostgreSQL-9.5.6

```
/* postfix-3.1.3/src/global/mail_params.c */
...
 static const CONFIG_TIME_TABLE time_defaults[] = {
   VAR_EVENT_DRAIN, DEF_EVENT_DRAIN, &var_event_drain, 1, 0,
   VAR_MAX_IDLE, DEF_MAX_IDLE, &var_idle_limit, 1, 0,
   VAR_IPC_TIMEOUT, DEF_IPC_TIMEOUT, &var_ipc_timeout, 1, 0,
   VAR_IPC_IDLE, DEF_IPC_IDLE, &var_ipc_idle_limit, 1, 0,
   VAR_IPC_TTL, DEF_IPC_TTL, &var_ipc_ttl_limit, 1, 0,
   VAR_TRIGGER_TIMEOUT, DEF_TRIGGER_TIMEOUT, &
       var_trigger_timeout, 1, 0,
   VAR_FORK_DELAY, DEF_FORK_DELAY, &var_fork_delay, 1, 0,
   VAR_FLOCK_DELAY, DEF_FLOCK_DELAY, &var_flock_delay, 1, 0,
   VAR_FLOCK_STALE, DEF_FLOCK_STALE, &var_flock_stale, 1, 0,
   VAR_DAEMON_TIMEOUT, DEF_DAEMON_TIMEOUT, &
       var_daemon_timeout, 1, 0,
   VAR_IN_FLOW_DELAY, DEF_IN_FLOW_DELAY, &var_in_flow_delay,
       0, 10,
   0,
 };
...
```

(c) Postfix-3.1.3

Figure 1: Examples of mapping constraints.

Table 5: Proportion of enumeration constraints that can be extract from switch-case-statement

Software	Total	Switch-case-statement
PostgreSQL	97	2
Redis	19	0
MySQL	86	3
Httpd	26	1
Postfix	19	0

To find out the semantic type of a configuration option, current work may need to use dataflow analysis to trace to known library functions. However, in our manual study, we found there are plenty of configuration options can't be traced to known library functions, or doesn't use the common library functions. On the other hand, many function parameters have meaningful identifiers rather than arbitrary strings. So we could mine much information from the function calling situation of the configuration options and the parameter identifiers. For example, in Postgresql, configuration option "ssl_key_file" cannot be traced to a known library function

```
/* postgresql-9.5.6/src/backend/utils/misc/guc.c */
...
 {
  {"backslash_quote", PGC_USERSET, COMPAT_OPTIONS_PREVIOUS,
   gettext_noop("Sets whether \"\\'\" is allowed in string
       literals."),
   NULL
  },
  &backslash_quote,
  BACKSLASH_QUOTE_SAFE_ENCODING, backslash_quote_options,
  NULL, NULL, NULL
 },
...
 static const struct config_enum_entry
    backslash_quote_options[] = {
  {"safe_encoding", BACKSLASH_QUOTE_SAFE_ENCODING, false},
  {"on", BACKSLASH_QUOTE_ON, false},
  {"off", BACKSLASH_QUOTE_OFF, false},
  {"true", BACKSLASH_QUOTE_ON, true},
  {"false", BACKSLASH_QUOTE_OFF, true},
  {"yes", BACKSLASH_QUOTE_ON, true},
  {"no", BACKSLASH_QUOTE_OFF, true},
  {"1", BACKSLASH_QUOTE_ON, true},
  {"0", BACKSLASH_QUOTE_OFF, true},
  {NULL, 0, false}
 };
...
```

(a) PostgreSQL-9.5.6

```
/* httpd-2.4.23/server/core.c */
...
 static const char *set_enable_sendfile(cmd_parms *cmd,
    void *d_, const char *arg)
 {
  core_dir_config *d = d_;

  if (strcasecmp(arg, "on") == 0) {
     d->enable_sendfile = ENABLE_SENDFILE_ON;
  }
  else if (strcasecmp(arg, "off") == 0) {
     d->enable_sendfile = ENABLE_SENDFILE_OFF;
  }
  else {
     return "parameter must be 'on' or 'off'";
  }

  return NULL;
 }
...
```

(b) Httpd-2.4.23

Figure 2: Examples of enumeration clustering in source code from different pieces of software.

Table 6: Proportion of enumeration constraints that can be extract from switch-case-statement

Software	Configuration options that could trace to known lib-functions	Configuration options that could trace to known function parameters
PostgreSQL	28.6%	42.9%
Redis	50%	66.7%
MySQL	40%	65%
Httpd	33.3%	88.9%
Postfix	33.3%	44.4%

call, but in its call trace, the function "pgwin32_safestat" is called, where the first parameter name is "path", just as shown in Fig. 3, so we could infer that the semantic type of configuration option "ssl_key_file" is "PATH".

Based on this finding, we get the statistics of the existence of this phenonmen, and the results is listed in Table 6.

```
/* postfix-3.1.3/src/global/mail_dict.c */
...
 dymap_init(path, var_shlib_dir);
 myfree(path);
...

/* postfix-3.1.3/src/global/dynamicmaps.c */
...
 void dymap_init(const char *conf_path, const char *
    plugin_dir)
 {
  static const char myname[] = "dymap_init";
  ...
 }
...
```

Figure 3: Example of semantic information in parameter identifiers from function call in PostgreSQL-9.5.6.

Table 7: Proportion of file resource-related configuration that has been checked

Software	File resource-related configurations	Proportion of accessibility checking
PostgreSQL	14	0%
Redis	4	66.7%
MySQL	20	25%
Httpd	9	44.4%
Postfix	9	33.3%

3.3 Obstacles in Extraction of Configuration Constraints

Finding 6: Structural configuration options are hard to analyze and the effect scope of configuration options are not considered yet.

There are some structural configuration options in Httpd and MySQL to organize the configuration files well, but those kinds of configuration options and its effect on constraints extraction are not considered yet. For instance, in Httpd, plenty of structural configuration options, such as "<VirtualHost", "<Directory", are used to limit the effect scope of specific configuration options. The similar measures are used in MySQL configuration files. Moreover, in PostgreSQL, there are parameters limiting the effect scope of a configuration option, such as the keyword "PGC_USERSET" in Fig. 1(b) and Fig. 2(a). These kinds of implementation have great effect on the extraction of configuration constraints, but it is complex to handled them well yet.

Finding 7: Resource-related configuration are not checked well in majority of studied software.

In runtime of software, it might need various of resources, such as file resources, network resources, hardware resourced and so on. Most of these resources might be declared in configuration files considering the difference in users' system environment. So it is necessary to check the accessibility of those resources before the program uses them. However, after study, we found that the checking for software-needed resources are relatively lacked. In term of file resources, the proportion of accessibility checking is shown in Table 7.

With these results, we thought that developers may hold the opinion that users who configure those resource-related configuration are experienced users, so they can make it correct. On the

```
/* postgresql-9.5.6/src/include/utils/guc_tables.h */
...
 struct config_int
 {
  struct config_generic gen;
  /* constant fields, must be set correctly in initial
     value: */
  int    *variable;
  int   boot_val;
  int   min;
  int   max;
  GucIntCheckHook check_hook;
  GucIntAssignHook assign_hook;
  GucShowHook show_hook;
  /* variable fields, initialized at runtime: */
  int    reset_val;
  void   *reset_extra;
 };
...
```

Figure 4: Definition of mapping structure in PostgreSQL-9.5.6.

other hand, in the view of good configuration design, it may be potential misconfiguration causes.

4 AUTOMATIC EXTRACTION OF CONFIGURATION CONSTRAINTS

Based on the findings we summarized from real-world software, as well as the configuration constraints type we defined in Table 2, we want to extract common situations and patterns to maximize the automation of configuration constraints extraction from various source codes. In summary, the main configuration constraints contain the followings: 1) numeric value ranges; 2) enumeration constraints; 3) semantic type of configuration options.

Strategy 1: Extracting numeric value ranges from mapping code snippets.

As mentioned in Finding 3, the mapping structures of configuration options and relevant variables contain mainly value range constraints, i.e. mapping constraints. Although there are obvious features to locate the mapping code snippets [14], it is hard to extract the constraints from those snippets without any domain knowledge. Aiming to this challenge, we are inspired to use some semantic information to help the extractions. For instance, in PostgreSQL, the definition of the mapping structure arrays is shown in Fig. 4, where the value range of integer type configuration are declared by structure members "int min" and "int max". Therefore, if we could use this kind of semantic information during the automatic analysis, the accuracy of constraints extraction will be much improved. The similar situations also occur in MySQL and Postfix.

Strategy 2: Extracting enumeration constraints from clustered code snippets.

As for enumeration constraints, we could use texture analysis to find the clustered code snippets for enumeration constraints, as Fig. 2 shows, without considering the complex and varied context as well as its implement. Then based on the features of these clustered snippets, the value spaces of enumerations could be extracted. Besides, some verifying strategies could be used to ensure the correctness based on the structured formations.

Strategy 3: Extracting semantic type with function parameters' information.

Aiming to the semantic type extraction of configuration options, we could optimize the dataflow analysis by using function parameter informations. To extract the semantic type of a configuration option, we find the function calls that take this configuration option as argument at first. Then, we could either analyze the parameter information to extract the possible semantic type of this configuration option, or trace into the implementation of this function to further analysis. Thus, we could get the semantic type of this configuration option.

5 CONCLUSION

Misconfigurations have become major causes of software failure. Configuration constraints play a vital role in misconfiguration diagnosis. In order to better obtain configuration constraints, we did a comprehensive study about the existence and variance of configuration constraints in source code. Based on the study results, we summarized several findings about configuration constraints and configuration designs. Finally, with features reflected in findings, we proposed several strategies to automatically extracting configuration constraints from source code.

ACKNOWLEDGMENTS

This work was partially supported by National Natural Science Foundation (61690203, 61532007) and National 973 Program (2014CB 340703) of China.

REFERENCES

[1] L Barroso, J Clidaras, and U Hoelzle. 2009. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. 8, 3 (2009), 154.
[2] Zhen Dong, Mohammadreza Ghanavati, and Artur Andrzejak. 2013. Automated diagnosis of software misconfigurations based on static analysis. In *IEEE International Symposium on Software Reliability Engineering Workshops*. 162–168.
[3] Httpd. 2017. http://httpd.apache.org/. (2017).
[4] MySQL. 2017. http://www.mysql.com/. (2017).
[5] Ariel Rabkin and Randy Katz. 2011. Static extraction of program configuration options. In *International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , Hi, Usa, May*. 131–140.
[6] Ariel Rabkin and Randy Howard Katz. 2013. How Hadoop Clusters Break. *IEEE Software* 30, 4 (2013), 88–94.
[7] Y Sverdlik. 2012. Microsoft: misconfigured network device led to azure outage. Retrieved January 29, 2017 from http://www.datacenterdynamics.com/focus/archive/2012/07/microsoft-misconfigured-network-device-led-azure-outage. (2012).
[8] A Team. 2011. Summary of the amazon ec2 and amazon rds service disruption in the us east region. Amazon Web Services,[online] http://aws.amazon.com/message/65648. (2011).
[9] Yingfei Xiong, Arnaud Hubaux, Steven She, and Krzysztof Czarnecki. 2012. Generating range fixes for software configuration. In *International Conference on Software Engineering*. 58–68.
[10] Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. 2016. Early Detection of Configuration Errors to Reduce Failure Damage. (2016).
[11] Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. 2013. Do not blame users for misconfigurations. In *Twenty-Fourth ACM Symposium on Operating Systems Principles*. 244–259.
[12] Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N. Bairavasundaram, and Shankar Pasupathy. 2011. An empirical study on configuration errors in commercial and open source systems. In *ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October*. 159–172.
[13] Jiaqi Zhang, Lakshminarayanan Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, Tianyin Xu, and Yuanyuan Zhou. 2014. EnCore: exploiting system environment and correlation information for misconfiguration detection. In *International Conference on Architectural Support for Programming Languages and Operating Systems*. 687–700.
[14] Shulin Zhou, Xiaodong Liu, Shanshan Li, Wei Dong, Xiangke Liao, and Yun Xiong. 2016. ConfMapper: Automated Variable Finding for Configuration Items in Source Code. In *2016 IEEE International Conference on Software Quality, Reliability and Security, QRS 2016, Companion, Vienna, Austria, August 1-3, 2016*. 228–235. DOI : https://doi.org/10.1109/QRS-C.2016.35