

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

به نام خدا

گزارش ۶۰ درصد پیشرفت پروژه کنترل کیفیت نرم افزار

نام سرباز : محمد خادم حسینی

نام استاد راهنما : دکتر مرتضی جوان

شهریور ۱۴۰۱

فهرست مطالب

مقدمه	۴
فصل اول : بررسی شاخص‌های کنترل کیفیت نرم‌افزار	۵
۱-۱- شاخص‌های کیفیت نرم‌افزار	۶
۱-۱-۱- قابلیت اطمینان	۶
۱-۱-۲- امنیت	۸
۱-۱-۳- قابلیت نگهداری	۹
۱-۱-۵- اندازه	۱۳
۱-۲- بررسی شاخص‌های نرم‌افزاری با استفاده از یادگیری ماشین	۱۷
۱-۲-۱- بررسی ویژگی‌های شاخص‌های نرم‌افزاری	۱۸
فصل دوم: بررسی ناهنجاری در سیستم سوئیفت با استفاده از لاگ‌های سیستم	۲۲
۲-۱- تشکیل گراف کنترل جریان برای لاگ‌ها	۲۳
۲-۱-۱- انواع لاگ	۲۳
۲-۱-۲- تشکیل پدر و فرزند برای هر لاگ	۲۴
۲-۱-۳- مرج کردن گراف‌ها	۲۵
۲-۱-۴- شناسایی ناهنجاری	۲۶
۲-۲- تشکیل گراف حالت برای لاگ‌ها	۲۶
۲-۳- بررسی انواع لاگ‌های سوئیفت	۲۸
۲-۳-۱- رسم گراف حالت برای داده‌های سوئیفت	۳۲
فصل سوم : مصورسازی	۳۸
۳-۱-۱- پایگاه داده گراف	۳۹

۳-۱-۲- استفاده از Neo4j.....	۳۹
۳-۱-۶- معایب استفاده از پایگاه داده گراف:.....	۴۱
۳-۱-۷- راه اندازی Neo4j:.....	۴۱
۳-۱-۹- کدهای پیاده سازی کلاس گراف Neo4j.....	۴۶
۳-۱-۱۰- اضافه کردن programname، method و user-agent به عنوان ویژگی های نود در Neo4j.....	۴۸
فصل چهارم: سیستم کشف ناهنجاری.....	۵۱
۴-۱- معماری سوئیفت و اهمیت کشف ناهنجاری.....	۵۲
۴-۱-۱- الگوریتم کشف ناهنجاری.....	۵۶
۴-۱-۲- بررسی چند نمونه لاگ ناهنجار.....	۶۱

مقدمه

کنترل کیفیت یک پروژه نرم‌افزاری از مهم‌ترین ابعاد آن است. کنترل کیفیت نرم‌افزار می‌تواند مفاهیم و جنبه‌های متفاوتی داشته باشد. در این پروژه ما از دو جهت این مفهوم را بررسی می‌کنیم. جهت اول، بررسی کیفیت یک پروژه از نظر میزان کیفیت برنامه‌نویسی است. پارامترهای مختلفی در این زمینه تعریف می‌شود که به طور کامل در این مطالعه بررسی خواهند شد. این پارامترها، میزان مشابَهت، وابستگی و شاخص‌های تست را زیر ذره‌بین قرار می‌دهند. شاخص تست به بررسی نتایج و میزان دربرگیرندگی تست‌های واحد می‌پردازد و به وسیله داده‌های به‌دست آمده از آن‌ها قضاوت می‌کند که نرم‌افزار تا چه اندازه مطابق با رویه‌های مورد انتظار که برای آن‌ها تست تهیه شده عمل می‌کند. همچنین مشخص می‌کند چه مقدار از کل کدهای موجود شامل خط‌های کد و شرط‌ها تحت پوشش تست قرار گرفته‌اند. در ادامه این بخش به بررسی شاخص‌های نرم‌افزاری با استفاده از یادگیری ماشین می‌پردازیم. هدف این قسمت پیش‌بینی میزان تلاش لازم جهت انجام پروژه است. دانستن این امر بسیار حایز اهمیت است چرا که کم یا زیاد تخمین زدن زمان لازم برای یک پروژه باعث تخمین غلط یک شرکت از میزان وسعت و پیچیدگی آن پروژه می‌شود و میزان هزینه‌ای که برای این پروژه در نظر می‌گیرد را تحت تاثیر قرار می‌دهد.

در قسمت دوم این مطالعه، ما سعی می‌کنیم از لاگ‌های یک نرم‌افزار، ناهنجاری را تشخیص دهیم. برای این منظور ابتدا سعی می‌کنیم گراف سلامت سیستم را رسم کنیم و پس از آن درصدد کشف ناهنجاری در یک سیستم نرم‌افزاری برآییم. به‌طور خاص در این قسمت ما نرم‌افزار سوییفت را مورد بررسی قرار می‌دهیم و کیفیت الگوریتم‌های خود را می‌سنجیم.

فصل 1- فصل اول : بررسی شاخص‌های کنترل کیفیت نرم‌افزار

۱-۱- شاخص‌های کیفیت نرم‌افزار

در این قسمت به بررسی شاخص‌های کیفیت نرم‌افزار می‌پردازیم. این شاخص‌ها شامل قابلیت اطمینان، امنیت، قابلیت نگهداری، تکرارها، اندازه، پیچیدگی و تست و پوشش هستند.

۱-۱-۱- قابلیت اطمینان

این شاخص بیان می‌کند که نرم‌افزار از لحاظ صحت عملکرد و پیروی از قوانین و الگوهای بهینه در چه جایگاهی قرار دارد. وجود باگ در نرم‌افزار، عدم پیروی از قوانین عمومی یا اختصاصی زبان، وجود آسیب‌پذیری و مواردی از این دست می‌توانند قابلیت اطمینان نرم‌افزار را کاهش دهند.

این شاخص به وسیله استخراج و شمارش باگ‌ها و مشکلات و شدت آن‌ها سنجیده می‌شود.

برای مثال سیستم سونار^۱ با شمارش خطاهای استخراج شد و در نظر گرفتن شدت آن‌ها برچسبی با عنوان رتبه‌بندی اطمینان به کد اختصاص می‌دهد و درجه قابلیت اطمینان آن را منطبق با آن‌چه در پایین لیست شده، تعیین می‌کند:

A. بدون خطا

B. حداقل یک مشکل فرعی

C. حداقل یک مشکل اصلی

D. حداقل یک مشکل بحرانی

E. حداقل یک مشکل مسدود کننده

علاوه بر این درجه‌بندی، شاخص هزینه اصلاح قابلیت اطمینان^۲ برای سورس کد ارائه می‌شود. این شاخص که در واحد زمان ارائه می‌شود بیان می‌کند چه میزان زمان برای برطرف کردن همه‌ی خطاهای موجود در سیستم و دستیابی به بیشینه قابلیت اطمینان نیاز است. این زمان بر مبنای ساعت و روز بیان می‌شود که در آن هر ۱ روز معادل با ۸ ساعت در نظر گرفته می‌شود. نرم‌افزار سویفت^۳ به دلیل وجود ۳۴ خطا که ۲ مورد از آن‌ها بلاکر هستند از این شاخص درجه اطمینان E دریافت کرده است. مابقی این موارد شامل ۳۱ خطای بحرانی و ۱ خطای فرعی هستند.

^۱sonar

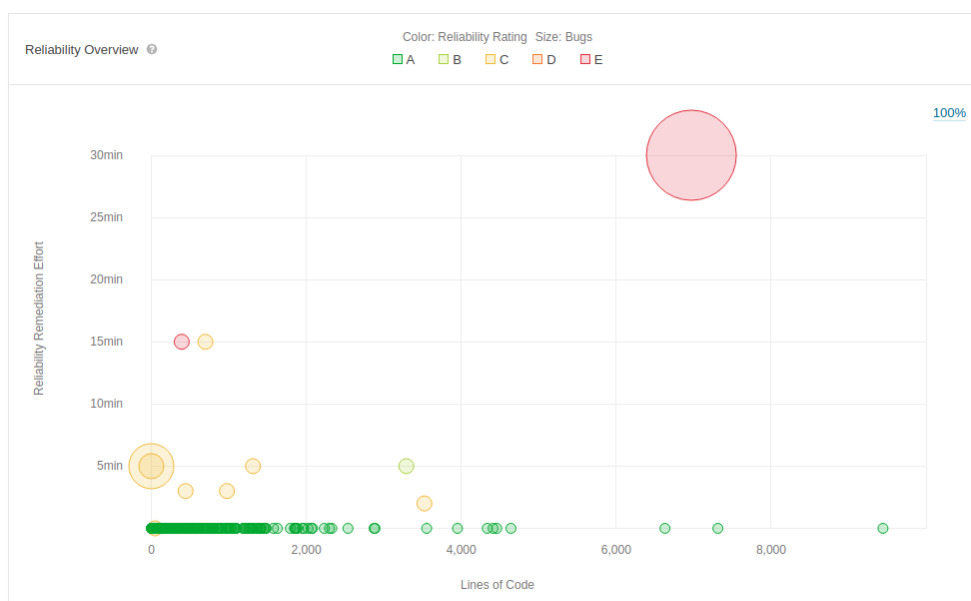
^۲Reliability Remediation Effort

^۳swift



شکل ۱: بررسی سوییفت در سیستم sonar

هزینه اصلاح قابلیت اطمینان برای سورس کد سوییفت ۱ ساعت و ۲۸ دقیقه محاسبه شده و نسبت آن به تعداد خط کدهای هر بخش در جدول زیر ترسیم شده است.



شکل ۲: تعداد خط کدهای سوییفت

برای مثال در پایین یکی از خطاهای بلاکر تشخیص داده شده را مشاهده می کنیم.

```
# See if the file is in the new format
magic = gz_file.read(4)
if magic == b'R1NG':
    format_version, = struct.unpack('!H', gz_file.read(2))
    if format_version == 1:
```

Remove this equality check between incompatible types; it will always return False. [Why is this an issue?](#) 5 years ago ▼ L180

Bug Blocker Open Not assigned 15min effort Comment unused

شکل ۳: بلاکر تشخیص داده شده در سوییفت

۲-۱-۱- امنیت

این شاخص بیان می‌کند که نرم‌افزار از نظر مقاومت در مواجهه با حملات در چه سطحی قرار دارد. نرم‌افزارها در مقابل لیستی از آسیب‌پذیری‌های عمومی (مانند SQL Injection و XSS ها) و آسیب‌پذیری‌های اختصاصی یک زبان (مثل استفاده از backtick در پایتون) بررسی می‌شوند و وجود هر یک از این موارد در آن‌ها گزارش می‌شود. شاخص امنیت با بررسی تعداد و شدت نقاط آسیب‌پذیری استخراج شده مورد سنجش قرار می‌گیرد.

برای مثال سیستم سونار با شمارش این آسیب‌پذیری‌ها و در نظر گرفتن شدت آن‌ها برچسبی با عنوان رتبه‌بندی امنیت به کد اختصاص می‌دهد و درجه امنیت آن را منطبق با آن‌چه در پایین لیست شده، تعیین می‌کند:

A. بدون آسیب‌پذیری

B. حداقل یک آسیب‌پذیری فرعی

C. حداقل یک آسیب‌پذیری اصلی

D. حداقل یک آسیب‌پذیری بحرانی

E. حداقل یک آسیب‌پذیری مسدود کننده

علاوه بر این درجه‌بندی، شاخص هزینه اصلاح امنیت^۴ برای سورس کد ارائه می‌شود. این شاخص که در واحد زمان ارائه می‌شود بیان می‌کند چه میزان زمان برای برطرف کردن همه‌ی خطاهای موجود در سیستم و دستیابی به بیشینه قابلیت اطمینان نیاز است. این زمان بر مبنای ساعت و روز بیان می‌شود که در آن هر ۱ روز معادل با ۸ ساعت در نظر گرفته می‌شود. نرم‌افزار سوئیفت به دلیل وجود ۲ آسیب‌پذیری امنیتی که هر ۲ مورد آن‌ها بلاکر هستند از این شاخص درجه اطمینان E دریافت کرده است.

هزینه اصلاح امنیت برای سورس کد سوئیفت صفر (با چشم پوشی از مقدار کوچک به‌دست آمده) محاسبه شده و نسبت آن به تعداد خط کدهای هر بخش در جدول زیر ترسیم شده است.

^۴ Security Remediation Effort



شکل ۴: نسبت هزینه اصلاح امنیت به تعداد خط کد

برای مثال در پایین یکی از آسیب‌پذیری‌هایی که بلاکر تشخیص داده شده را مشاهده می‌کنیم.

```
@unittest.skipIf(xprofile is None, "can't import xprofile")
def setUp(self):
    self.profile_file = tempfile.mktemp('profile', 'unittest')
```

'tempfile.mktemp' is insecure. Use 'tempfile.TemporaryFile' instead [Why is this an issue?](#)

6 years ago ▾ L471 🔗

🔒 Vulnerability ▾ ⚠️ Blocker ▾ 🔓 Open ▾ Not assigned ▾ Comment

🔍 cwe, owasp-a9 ▾

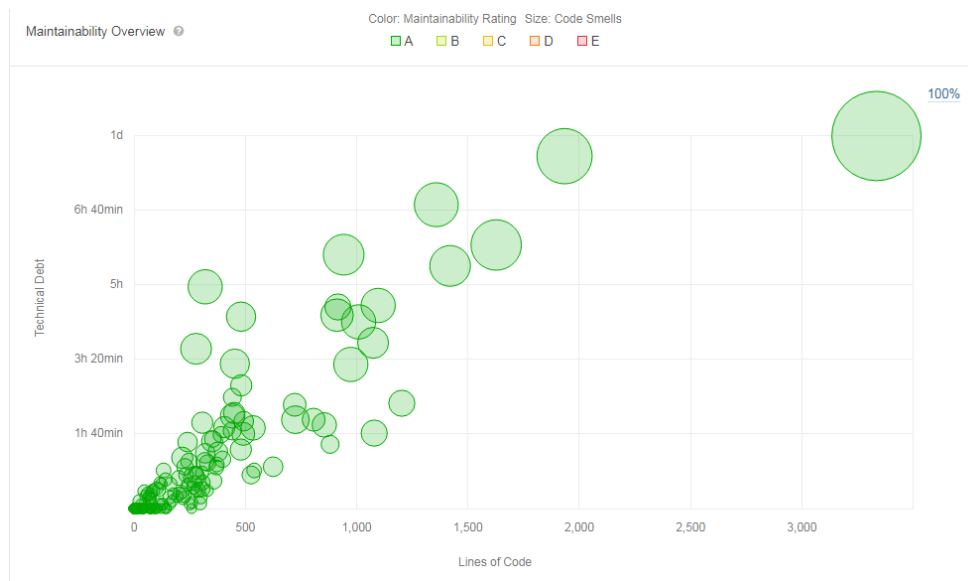
شکل ۵: آسیب‌پذیری بلاکر

۳-۱-۱- قابلیت نگهداری

این شاخص بیان می‌کند که نگهداری نرم‌افزار تا چه اندازه دشوار و زمان‌بر است و مثلاً برای برطرف کردن یک خطا به‌طور متوسط چه مقدار زمان باید صرف شود. کدهای مشکوک یکی از سنجه‌هایی است که برای اندازه‌گیری این شاخص استفاده می‌شوند. برای مثال وجود خط کدهای طولانی، استفاده از چندین if و else که می‌تواند با یک switch case جایگزین شود از این موارد هستند.

در همین ارتباط مفهوم بدهی فنی^۵ مطرح می‌شود. که زمان کلی برای برطرف کردن همه‌ی کدهای مشکوک را شامل می‌شود. این زمان‌ها معمولاً بسته به نوع مشکل به‌صورت ثابت‌هایی به دقیقه برای آن در نظر گرفته می‌شوند که مجموع آن‌ها بیان‌کننده بدهی فنی نرم‌افزار است. وضعیت کلی این شاخص در سوییفت بصورت زیر است.

technical debt°



شکل ۶: شاخص بدهی فنی برای سوییفت

بدهی فنی در این پروژه حدود ۱۰۸ روز تخمین زده شده است که در بخش های مختلف بصورت زیر است:

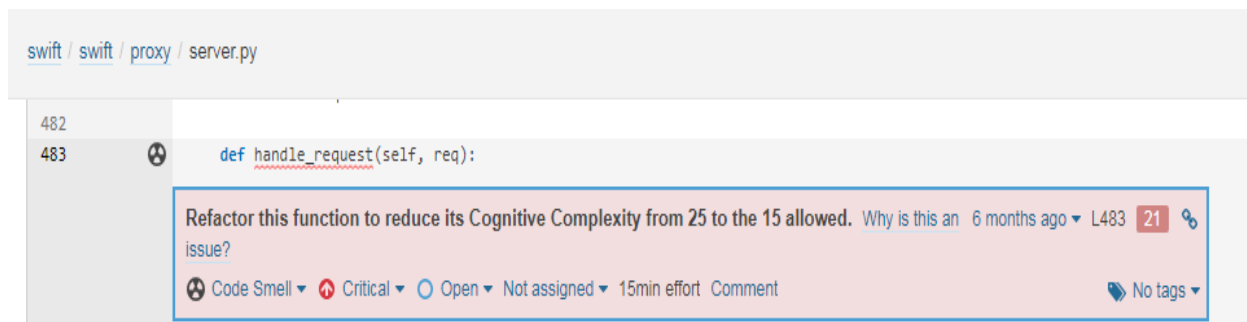
Technical Debt 19d

account	3h 58min
cli	1d 3h
common	11d
container	1d 5h
obj	2d 6h
proxy	1d 6h
__init__.py	0

7 of 7 shown

شکل ۷: بدهی فنی در قسمت های مختلف سوییفت

برای مثال در پایین یکی از کدهای مشکوک که بحرانی تشخیص داده شده را مشاهده می کنیم.



شکل ۸ : کدهای مشکوک

۴-۱-۱- تکرارها

این شاخص نشان‌دهنده میزان تکرار در سورس کد نرم‌افزار است. تکرار می‌تواند خط کد تکراری، بلاک کد تکراری یا فایل تکراری باشند. وجود تکرار در کد باعث افزایش اندازه کد، پیچیدگی نگهداری و افزایش زمان اعمال تغییرات می‌شود.

- خط کد تکراری: تعداد خط کدهایی که در یک نرم‌افزار عیناً تکرار شده‌اند.
 - فایل تکراری: تعداد فایل‌هایی که در یک نرم‌افزار عیناً تکرار شده‌اند.
- بلاک کد تکراری: حداقل n دستور پشت‌سرهم و تکراری بدون در نظر گرفتن تعداد خطوط و توکن‌ها. که مقدار n برای زبان‌های برنامه‌نویسی مختلف متفاوت است. پارامتر n در زبان‌هایی مثل جاوا و پایتون برابر ۱۰ و در زبانی مانند Cobol برابر با ۱۰۰ است. علاوه بر این شاخص تراکم تکرار^۶ بیان می‌کند چند درصد از کدها در سورس کد یا به ازای هر فایل تکراری هستند. در سورس کد سوئیفت تراکم تکرار کل پروژه برابر ۱/۶ درصد است. در زیر لیست برخی از فایل‌های دارای بیش‌ترین تراکم تکرار ارائه شده است.

^۶density

Duplicated Lines (%)	Duplicated Lines
-------------------------	---------------------

شکل ۹: فایل‌های دارای بیشترین تراکم در سویفت

همچنین در مجموع ۱۷۹۵۴ خط کد تکراری، ۱۳۸۶ بلاک کد تکراری و ۹۳ فایل تکراری در پروژه وجود دارد.

```
body = [
  '-----WebKitFormBoundaryNcxTqxSlX7t4TDkR',
  'Content-Disposition: form-data; name="redirect"',
  '-----WebKitFormBoundaryNcxTqxSlX7t4TDkR',
  'Content-Disposition: form-data; name="max_file_size"',
  str(max_file_size),
  '-----WebKitFormBoundaryNcxTqxSlX7t4TDkR',
  'Content-Disposition: form-data; name="max_file_count"',
  str(max_file_count),
  '-----WebKitFormBoundaryNcxTqxSlX7t4TDkR',
  'Content-Disposition: form-data; name="expires"',
  str(expires),
  '-----WebKitFormBoundaryNcxTqxSlX7t4TDkR',
  'Content-Disposition: form-data; name="signature"',
  str(signature),
  '-----WebKitFormBoundaryNcxTqxSlX7t4TDkR'
]
```

شکل ۱۰: نمونه ارور در سوییفت

۵-۱-۱- اندازه

شاخص اندازه بیانگر بزرگی نرم افزار است که تعداد دایرکتوری ها، فایل ها، کلاس ها، توابع، خط کد و دستورات است. اعدادی که از شمارش این موارد به دست می آیند می توانند بینشی در مورد بزرگی یک نرم افزار ارائه کنند. برای شمارش خط کدها از معیار شمارش خط های کد و غیر کامنت استفاده می شود.

- خط^y: همه خطوط موجود در یک فایل.
 - خط کامنت[^]: همه ی خط های کامنت شده که حداقل یک کاراکتر غیر از space یا انواع tab یا کاراکترهای خاص مربوط به کامنت در آن وجود دارد.
 - خط کد[^]: تعداد خط های از فایل که حداقل یک کاراکتر غیر از space یا انواع tab در آن وجود دارد و جزو کامنت های آن زبان برنامه نویسی محسوب نمی شود.
- با استفاده از همین تعاریف مفهوم دیگری به نام تراکم کامنت نیز معرفی می شود که به شکل زیر محاسبه می گردد.

$$\text{Density of comment lines} = \text{Comment lines} / (\text{Lines of code} + \text{Comment lines})$$

* ۱۰۰

به وسیله این معادله برای مثال اگر تراکم کامنت یک فایل ۵۰ درصد باشد یعنی تعداد خط های کامنت با تعداد خط های کد برابر هستند و اگر ۱۰۰ درصد باشد به این معنی است که کل فایل را کامنت تشکیل می دهد. آمار استخراج شده برای سورس کد سوئیفت به شرح زیر است:

Lines of Code: ۲۲۲,۳۴۵

Lines: ۲۹۴,۳۸۵

Functions: ۱۳,۱۴۸

Classes: ۱,۴۰۹

Files: ۳۶۴

Comment Lines: ۳۷,۶۰۰

Comments (%): ۱۴,۵%

lines^y

comment lines[^]

NCLOC : Non Comment Lines Of Code[^]

۶-۱-۱- پیچیدگی

شاخص پیچیدگی بیان گر میزان پیچیدگی یک کد از لحاظ تو در تو بودن روال ها و قابل فهم بود کد است. این شاخص با دو سنجی اصلی پیچیدگی چرخه‌ای و پیچیدگی مفهومی اندازه‌گیری می‌شود.

پیچیدگی چرخه‌ای^{۱۰}: این پیچیدگی بر مبنای تعداد مسیرها موجود در کد محاسبه می‌شود. هر بار که رویه کنترلی کد انشعاب پیدا می‌کند (برای مثال استفاده از بلاک‌های شرطی) این پیچیدگی یک واحد افزایش پیدا می‌کند. نحوه محاسبه این پیچیدگی با توجه به وجود کلمات کلیدی متنوع در هر زبان بسته به زبان‌های برنامه‌نویسی متفاوت است. پیچیدگی مفهومی^{۱۱}: پیچیدگی مفهومی نشان می‌دهد که فهمیدن رویه کد چه مقدار دشوار است. برای محاسبه این فاکتور با سه قانون کلی در نظر گرفته می‌شود:

- در نظر نگرفتن ساختارهایی که اجازه می‌دهند چندین دستور در قالب یک دستور shorthand بازنویسی شوند.
- افزودن یک واحدی پیچیدگی برای هر شکستگی^{۱۲} در ساختار خطی رویه‌ها.
- افزودن پیچیدگی زمانی که ساختارهای رویه‌شکن تو در تو هستند.

Cyclomatic Complexity 27,388 	
 swift/common/utlis.py	994
 test/unit/proxy/test_server.py	842
 test/unit/obj/test_diskfile.py	812
 test/unit/common/test_utils.py	725
 test/unit/obj/test_server.py	484
 swift/obj/diskfile.py	450
 test/unit/proxy/controllers/test_obj.py	415
 test/functional/tests.py	415
 swift/proxy/controllers/obj.py	368
 swift/proxy/controllers/base.py	360

شکل ۱۱ : پیچیدگی چرخه‌ای کد سوییفت

^{۱۰} Cyclomatic Complexity

^{۱۱} Cognitive Complexity

^{۱۲} break

در سورس کد سوئیفت مجموع پیچیدگی چرخه‌ای برابر با ۲۷۳۸۸ است که در تصویر زیر تعداد از پیچیده‌ترین فایل‌های این سورس کد از نظر پیچیدگی چرخه‌ای مشاهده می‌شود:

Refactor this function to reduce its Cognitive Complexity from 16 to the 15 allowed. [Why is this an issue?](#) 2 years ago ▾ L238 10 🔗

🔗 Code Smell ▾ 🚨 Critical ▾ 🔓 Open ▾ Not assigned ▾ 6min effort Comment

🧠 brain-overload ▾

```
global HASH_PATH_SUFFIX
global HASH_PATH_PREFIX
if not HASH_PATH_SUFFIX and not HASH_PATH_PREFIX:
    hash_conf = ConfigParser()

    if six.PY3:
        # Use Latin1 to accept arbitrary bytes in the hash prefix/suffix
        with open(SWIFT_CONF_FILE, encoding='latin1') as swift_conf_file:
            hash_conf.readfp(swift_conf_file)
    else:
        with open(SWIFT_CONF_FILE) as swift_conf_file:
            hash_conf.readfp(swift_conf_file)

    try:
        HASH_PATH_SUFFIX = hash_conf.get('swift-hash',
                                         'swift_hash_path_suffix')

        if six.PY3:
            HASH_PATH_SUFFIX = HASH_PATH_SUFFIX.encode('latin1')
    except (NoSectionError, NoOptionError):
        pass

    try:
        HASH_PATH_PREFIX = hash_conf.get('swift-hash',
                                         'swift_hash_path_prefix')

        if six.PY3:
            HASH_PATH_PREFIX = HASH_PATH_PREFIX.encode('latin1')
    except (NoSectionError, NoOptionError):
        pass

    if not HASH_PATH_SUFFIX and not HASH_PATH_PREFIX:
        raise InvalidHashPathConfigError()
```

شکل ۱۲: نمونه‌ای از پیچیدگی چرخه‌ای

همین‌طور مجموع پیچیدگی مفهومی در سورس کد سوئیفت برابر با ۲۴۴۹۴ است که در تصویر زیر تعداد از پیچیده‌ترین فایل‌های این سورس کد از نظر پیچیدگی مفهومی مشاهده می‌شود:


```
def fallocate(fd, size, offset=0):
```

Refactor this function to reduce its Cognitive Complexity from 17 to the 15 allowed. Why is this an issue? 2 years ago L1021 14

Code Smell Critical Open Not assigned 7min effort Comment brain-overload

```
"""
Pre-allocate disk space for a file.

This function can be disabled by calling disable_fallocate(). If no
suitable C function is available in libc, this function is a no-op.

:param fd: file descriptor
:param size: size to allocate (in bytes)
"""
global _fallocate_enabled
if not _fallocate_enabled:
    return

if size < 0:
    size = 0 # Done historically; not really sure why
if size >= (1 << 63):
    raise ValueError('size must be less than 2 ** 63')
if offset < 0:
    raise ValueError('offset must be non-negative')
if offset >= (1 << 63):
    raise ValueError('offset must be less than 2 ** 63')

# Make sure there's some (configurable) amount of free space in
# addition to the number of bytes we're allocating.
if FALLOCATE_RESERVE:
    st = os.fstatvfs(fd)
    free = st.f_frsize * st.f_bavail - size
    if FALLOCATE_IS_PERCENT:
        free = (float(free) / float(st.f_frsize * st.f_blocks)) * 100
    if float(free) <= float(FALLOCATE_RESERVE):
        raise OSError(
            errno.ENOSPC,
            'FALLOCATE_RESERVE fail %g <= %g' %
            (free, FALLOCATE_RESERVE))

if _sys_fallocate.available:
```

شکل ۱۳: پیچیدگی مفهومی سویفت

Cognitive Complexity 24,494

swift/common/utlis.py	1,142
test/unit/obj/test_diskfile.py	642
swift/obj/diskfile.py	612
swift/proxy/controllers/base.py	560
test/unit/proxy/test_server.py	520
test/functional/tests.py	452
swift/common/ring/builder.py	450
test/functional/swift_test_client.py	435
swift/cli/recon.py	422
swift/proxy/controllers/obj.py	421

شکل ۱۴: پیچیدگی مفهومی در قسمت‌های مختلف سویفت

۷-۱-۱- تست و پوشش

این شاخص به بررسی نتایج و میزان دربرگیرندگی تست‌های واحد می‌پردازد و به وسیله داده‌های به دست آمده از آن‌ها قضاوت می‌کند که نرم‌افزار تا چه اندازه مطابق با رویه‌های مورد انتظار که برای آن‌ها تست تهیه شده عمل می‌کند. همچنین مشخص می‌کند چه مقدار از کل کدهای موجود شامل خط‌های کد و شرط‌ها تحت پوشش تست قرار گرفته‌اند. از جمله سنجه‌های مرتبط با تست واحد می‌توان از تعداد تست‌های واحد، تعداد تست‌های دارای خطا، تعداد تست‌های شکست خورده با خطای پیش‌بینی نشده و تراکم موفقیت تست‌ها نام برد. تعداد خط کد قابل پوشش، تعداد خط کدهای پوشش داده شده، تعداد شرط بر اساس خط کد، تعداد شرط‌های پوشش داده شده بر اساس خط کد و پوشش کلی از جمله سنجه‌های مرتبط با پوشش هستند.

۲-۱- بررسی شاخص‌های نرم‌افزاری با استفاده از یادگیری ماشین

در این بخش به بررسی شاخص‌های نرم‌افزاری با استفاده از یادگیری ماشین می‌پردازیم. هدف این قسمت پیش‌بینی میزان تلاش لازم^{۱۳} جهت انجام پروژه است. شکل زیر این مفهوم را نشان می‌دهد. دانستن این امر بسیار حایز اهمیت است چرا که کم یا زیاد تخمین زدن زمان لازم برای یک پروژه باعث تخمین غلط یک شرکت از میزان وسعت و پیچیدگی آن پروژه می‌شود و میزان هزینه‌ای که برای این پروژه در نظر می‌گیرد را تحت تاثیر قرار می‌دهد. در ابتدا کمی در مورد شاخص‌های نرم‌افزاری و ویژگی‌های آن‌ها بحث می‌کنیم.



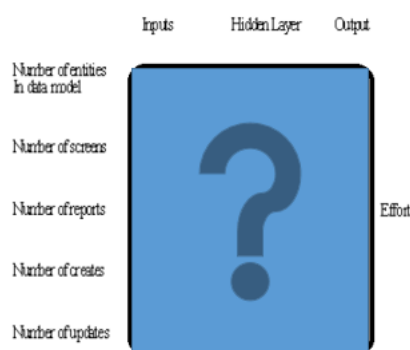
شکل ۱۵ : سیستم تخمین‌زننده تلاش لازم برای یک پروژه

^{۱۳} effort

۱-۲-۱- بررسی ویژگی‌های شاخص‌های نرم‌افزاری

برای آن که سیستم پیشگوی یادگیری ماشین بتواند تخمین درستی از میزان تلاش لازم برای انجام یک پروژه داشته باشد باید ویژگی‌های ورودی به سیستم ویژگی‌های جامع، کامل و مستقل خطی باشند. شاخص‌های نرم‌افزاری از یکدیگر مستقل خطی نیستند و این خود موجب کاهش دقت سیستم می‌شود. یک راهی که می‌توان برای حل این مشکل پیشنهاد داد استفاده از روش آنالیز فاکتور^{۱۴} است. مشکل دیگری که در شاخص‌های نرم‌افزاری به عنوان ورودی سیستم وجود دارد وجود دیتای گم‌شده^{۱۵} در این شاخص‌هاست. مشکل دیگر وجود دامنه‌های مختلف در این شاخص‌هاست که خود فرآیند یادگیری را دچار مشکل می‌کند. از طرف دیگر، دیتابیس شامل این شاخص‌ها پر از دیتای نویزی^{۱۶} هست. وجود این مشکلات حل مساله را دشوار می‌کند. به عبارت دیگر، ابتدا باید راهی برای حل این مسائل پیدا کرد و سپس مساله پیش‌بینی انجام شود.

در شکل زیر ویژگی‌های ورودی سیستم نمایش داده شده است. مساله یافتن ساختار مناسب به‌جای علامت سوال است.



شکل ۱۶ : ورودی سیستم تخمین‌زننده تلاش نرم‌افزاری

۱-۲-۲- بررسی روش‌های تخمین زدن تلاش نرم‌افزاری

در این جا ما دو روش را برای تخمین زدن میزان تلاش بررسی می‌کنیم:

- رگرسیون

^{۱۴} factor analysis

^{۱۵} missing data

^{۱۶} outlier

• شبکه‌های عصبی

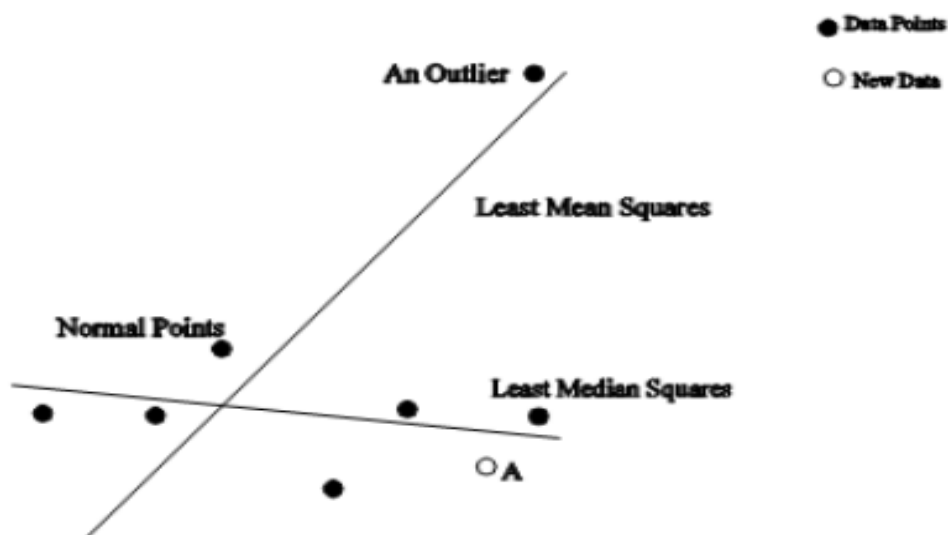
در مدل‌های آماری، تحلیل رگرسیون یک فرایند آماری برای تخمین روابط بین متغیرها می‌باشد. این روش شامل تکنیک‌های زیادی برای مدل‌سازی و تحلیل متغیرهای خاص و منحصر بفرد، با تمرکز بر رابطه بین متغیر وابسته و یک یا چند متغیر مستقل، می‌باشد. تحلیل رگرسیون خصوصاً کمک می‌کند در فهم اینکه چگونه مقدار متغیر وابسته با تغییر هر کدام از متغیرهای مستقل و با ثابت بودن دیگر متغیرهای مستقل تغییر می‌کند. بیشترین کاربرد تحلیل رگرسیون تخمین امید ریاضی شرطی متغیر وابسته از متغیرهای مستقل معین است که معادل مقدار متوسط متغیر وابسته است وقتی که متغیرهای مستقل ثابت هستند. کمترین کاربرد آن تمرکز روی چندک یا پارامتر مکانی توزیع شرطی متغیر وابسته از متغیر مستقل معین است. در همه موارد هدف تخمین یک تابع از متغیرهای مستقل است که تابع رگرسیون نامیده شده‌است. تحلیل رگرسیون به صورت گسترده برای پیش‌بینی استفاده شده‌است. تحلیل رگرسیون همچنین برای شناخت ارتباط میان متغیر مستقل و وابسته و شکل این روابط استفاده شده‌است. در شرایط خاصی این تحلیل برای استنتاج روابط عالی بین متغیرهای مستقل و وابسته می‌تواند استفاده شود. هر چند این می‌تواند موجب روابط اشتباه یا باطل شود بنابراین احتیاط قابل توصیه است. تکنیک‌های زیادی برای انجام تحلیل رگرسیون توسعه داده شده‌است. روش‌های آشنا همچون رگرسیون خطی و حداقل مربعات که پارامتری هستند، در واقع در آن تابع رگرسیون تحت یک تعداد محدودی از پارامترهای ناشناخته از داده‌ها تخمین زده شده‌است. رگرسیون غیر پارامتری به روش‌هایی اشاره می‌کند که به توابع رگرسیون اجازه می‌دهد تا در یک مجموعه مشخص از توابع با احتمال پارامترهای نامحدود قرار گیرند. رگرسیون با مینیمم کردن مجموع مجذورات خطا: این نوع از رگرسیون رایج‌ترین نوع رگرسیون در یادگیری ماشین است. رایج بودن آن به این معنی نیست که عملکرد بهتری نسبت به بقیه انواع رگرسیون دارد بلکه دلیل رایج بودن آن راحتی استفاده از آن به دلیل وجود پکیج‌های آماده پایتون است. در این نوع رگرسیون هدف یافتن بردار W جهت مینیم کردن تابع زیر است:

$$\sum (w^T x_i - y_i)^2 \quad (1)$$

که در آن y_i مقادیر هدف هستند. این نوع از رگرسیون بسیار به دیتای نویزی^{۱۷} حساس است. یعنی اگر دیتای پرتی در دیتابیس آموزش موجود باشد، سیستم سعی می‌کند خطی به دست بیاورد که این دیتای پرت را هم پوشش دهد. این امر باعث خطا در سایر دیتاها می‌شود. راه درست آن است که سیستم توجه کمی به دیتاهای پرت داشته باشد. از آنجایی که میانه داده‌ها به ندرت تحت تاثیر داده‌های پرت قرار می‌گیرد، روش زیر می‌تواند مشکل روش اول را حل کند. رگرسیون با مینیمم کردن میانه مجذورات خطا: در این نوع رگرسیون هدف یافتن مقادیر W جهت مینیمم کردن تابع زیر است:

$$\text{median}(w^T x_i - y_i)^2 \quad (2)$$

شکل زیر نشان می‌دهد که روش دوم تحت تاثیر داده‌های پرت قرار نمی‌گیرد.



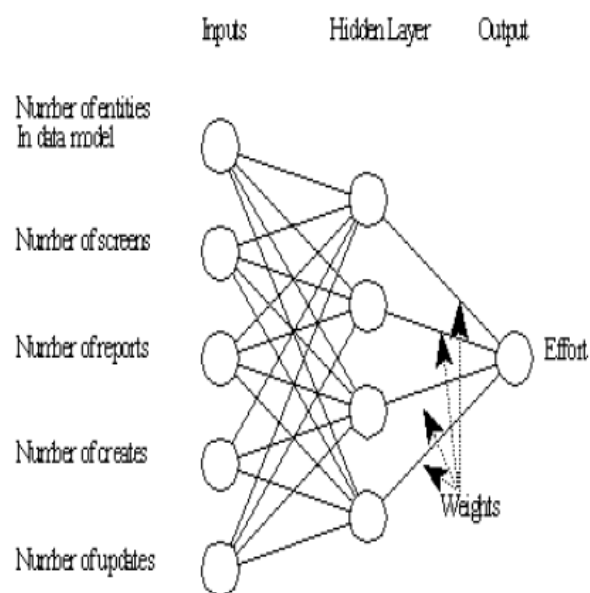
شکل ۱۶ : شرایطی که داده پرت ایجاد می‌کند.

شبکه‌های عصبی مصنوعی^{۱۸} یا به زبان ساده‌تر شبکه‌های عصبی سیستم‌ها و روش‌های محاسباتی نوین برای یادگیری ماشینی، نمایش دانش و در انتها اعمال دانش به دست آمده در جهت بیش‌بینی پاسخ‌های خروجی از سامانه‌های

^{۱۷} outlier

^{۱۸} Artificial Neural Networks - ANN

پیچیده هستند. ایده‌ی اصلی این گونه شبکه‌ها تا حدودی الهام‌گرفته از شیوه‌ی کارکرد سیستم عصبی زیستی برای پردازش داده‌ها و اطلاعات به منظور یادگیری و ایجاد دانش می‌باشد. این سیستم از شمار زیادی عناصر پردازشی فوق‌العاده بهم‌پیوسته با نام نورون تشکیل شده که برای حل یک مسئله با هم هماهنگ عمل می‌کنند و توسط سیناپس‌ها (ارتباطات الکترومغناطیسی) اطلاعات را منتقل می‌کنند. در این شبکه‌ها اگر یک سلول آسیب ببیند بقیه سلول‌ها می‌توانند نبود آن را جبران کرده، و نیز در بازسازی آن سهیم باشند. این شبکه‌ها قادر به یادگیری هستند. یادگیری در این سیستم‌ها به صورت تطبیقی صورت می‌گیرد، یعنی با استفاده از مثال‌ها وزن سیناپس‌ها به گونه‌ای تغییر می‌کند که در صورت دادن ورودی‌های جدید، سیستم پاسخ درستی تولید کند. به عنوان روش سوم، می‌توان از لایه آخر جهت تخمین زدن تلاش نرم‌افزاری استفاده کرد. شکل زیر این مساله را نشان می‌دهد. شبکه‌ی عصبی نشان داده شده در این شکل شامل یک لایه مخفی با ۴ نورون است.



شکل ۱۷ : شبکه عصبی تخمین تلاش نرم‌افزاری

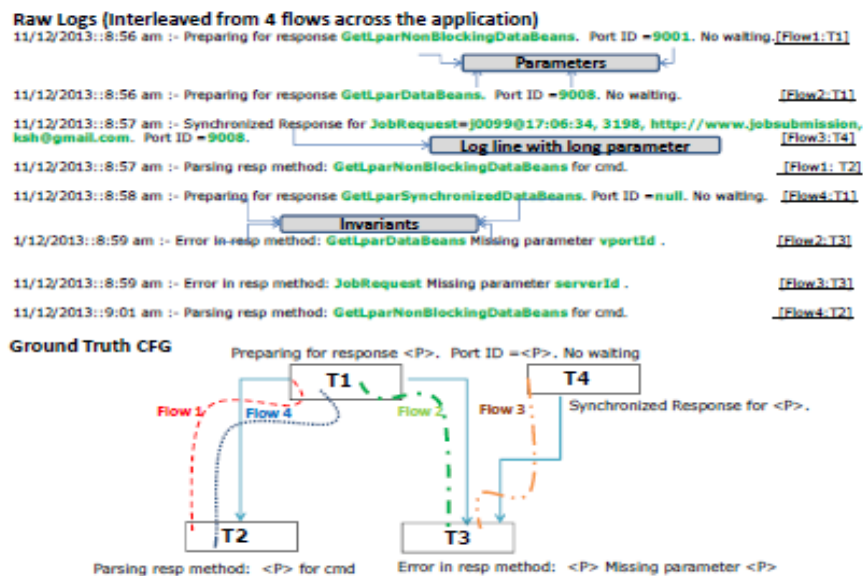
فصل ۲- فصل دوم: بررسی ناهنجاری در سیستم سوییچت با استفاده از لاگ‌های سیستم

۱-۲- تشکیل گراف کنترل جریان^{۱۹} برای لاگ‌ها

در این قسمت به بررسی نحوه تشکیل گراف کنترل جریان برای تشخیص ناهنجاری می‌پردازیم. برای این منظور ابتدا انواع و ساختار لاگ‌ها را بررسی می‌کنیم.

۱-۱-۲- انواع لاگ^{۲۰}

در شکل زیر انواع و دسته‌بندی لاگ‌های یک سیستم نرم‌افزاری را مشاهده می‌کنید.



شکل ۱۸ : ساختار لاگ‌ها

هر لاگ در سیستم نرم‌افزاری به دو دسته تقسیم می‌شود: قسمت تغییرپذیر^{۲۱} و قسمت ثابت^{۲۲}. قسمت ثابت شامل تمپلیت‌های ثابتی مانند موارد زیر است:

- Prepared for response
- Synchronized Response
- Parsing resp method

^{۱۹} Control flow graph (CFG)

^{۲۰} Log

^{۲۱} Variant

^{۲۲} Invariant

اما قسمت متغیر شامل موارد مانند آدرس پورت^{۲۳} و زمان انجام لاگ است. در شکل ۱، مقصود از T^1 و T^2 و.. همان تمپلیت‌های ثابت هستند. در پایین این شکل، یک گراف کنترل جریان از لاگ‌ها ساخته شده است. در این شکل، $Flow^1$ از تمپلیت‌های ۱ و ۲ استفاده میکند و همانطور هم که در شکل مشاهده می‌شود این فلو از T^1 و T^2 رد می‌شود. هدف در این قسمت ساختن چنین گرافی از لاگ‌ها به صورت کور است.

۲-۱-۲- تشکیل پدر^{۲۴} و فرزند^{۲۵} برای هر لاگ

جهت آن که به یک گراف کنترل جریان برسیم باید برای هر لاگ تعدادی پدر و فرزند تشکیل شود. برای تشکیل پدر و فرزند برای هر تمپلیت^{۲۶} باید توالی زمانی لاگ‌ها در نظر گرفته شود. به عنوان مثال فرض کنید که بخواهیم برای تمپلیت T^2 پدر و فرزند استخراج کنیم. در شکل ۲ این فرآیند نمایش داده شده است. در این شکل، از یک روش آماری برای استخراج پدر و فرزندهای T^2 استفاده شده است. برای آن که اساس این روش آماری توضیح داده شود ابتدا توالی لاگ‌های نمایش داده شده در شکل را در نظر بگیرید. تعدادی از این لاگ‌ها نویز هستند. با یک میانگین‌گیری ساده می‌توان متوجه شد که اکثر مواقع بعد از تمپلیت T^1 ، تمپلیت T^2 به وقوع می‌پیوندد. موارد نادر دیگری هم هستند که به عنوان نویز در نظر گرفته می‌شوند. با همین روش متوجه می‌شویم که بعد از T^2 در ۷۰ درصد موارد T^3 و در ۳۰ درصد موارد T^6 به وقوع می‌پیوندد. موارد نادر دیگری هم هستند که به عنوان نویز در نظر گرفته می‌شوند. به همین سادگی پدر و فرزند تمپلیت T^2 استخراج می‌شود. اما سوالی که یه وجود می‌آید این است: با چه حد آستانه‌ای^{۲۷} موارد نویزی را تشخیص دهیم؟ برای پاسخ به این سوال از یک قاعده سرانگشتی استفاده می‌کنیم و آن هم این نکته است: اگر درصد حضور زیر ۱۰ درصد باشد، آن مورد به عنوان نویز در نظر گرفته شود.

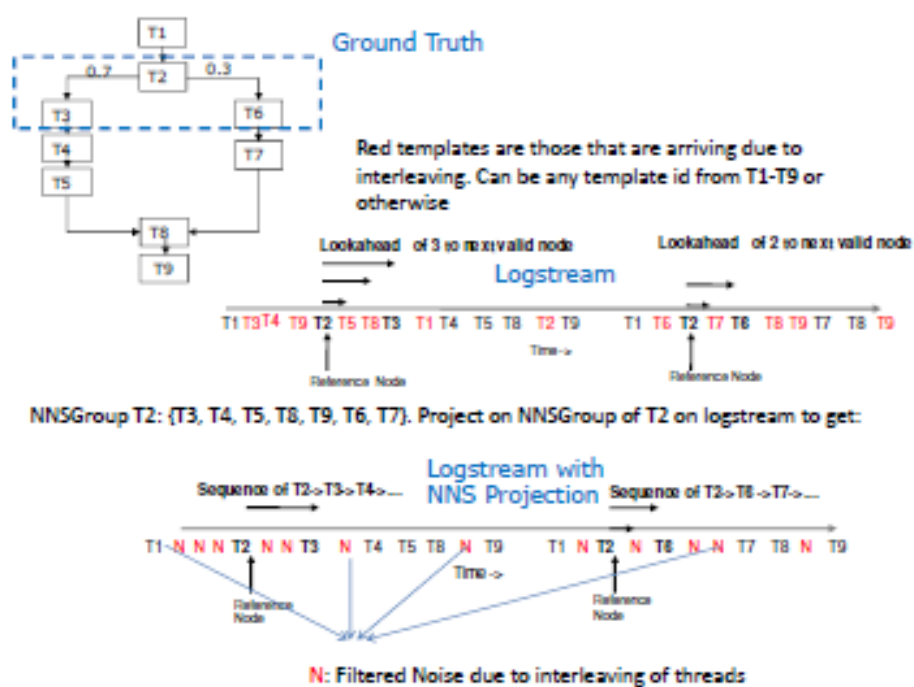
^{۲۳}Port address

^{۲۴}Predecessor

^{۲۵}successor

^{۲۶}Template

^{۲۷}Threshold



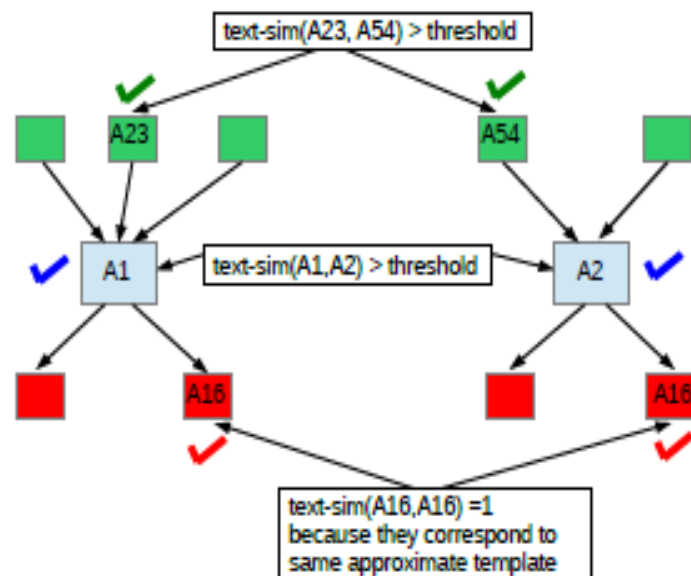
شکل ۱۹: تشکیل پدر و فرزند برای هر تمپلیت

۳-۱-۲- مرج کردن گرافها

در شکل ۳ نحوه مرج کردن گرافهای مربوط به هر تمپلیت نمایش داده شده است. در شکل زیر A^1 و A^2 دو تمپلیت هستند که همراه با پدرها و فرزندانهاون نمایش داده شده اند. اگر شباهت بین دو تمپلیت از نظر متنی^{۲۸} بالای یک حد آستانه باشد دو گراف با همدیگر مرج می شوند و بین پدرها و فرزندانهاون اجتماع گرفته می شود. اما اگر این شباهت فقط بین فرزندان و یا پدرها باشد، دو گراف از محل آن شباهت بایکدیگر مرج می شوند این مرج شدن به همین صورت بین هر دو گراف تکرار می شود تا در نهایت به یک گراف کلی برسیم. گراف به دست آمده همان گراف کنترل جریان برای لاگها است. معیار شباهت هم، شباهت $JACCARD^{29}$ است.

^{۲۸}Text similarity

^{۲۹}Jaccard similarity



شکل ۲۰: نحوه مرج کردن گراف‌ها

۴-۱-۲- شناسایی ناهنجاری^{۳۰}

در یک سیستم نرم‌افزاری پس از تشکیل گراف کنترل جریان و گراف هر تمپلیت، دو حالت می‌تواند نشانه ناهنجاری باشد:

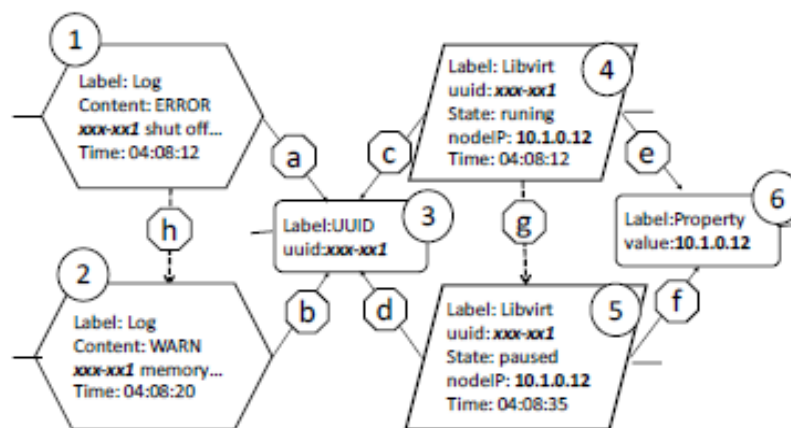
- الف) در یک بازه زمانی مشخص، فرزندان یک تمپلیت مشاهده نشوند. اگر همه فرزندان مشاهده نشوند احتمال ناهنجاری بیشتر از حالتی است که یک یا تعدادی از فرزندان مشاهده نشوند.
- ب) توزیع فرزندان یا پدران هر تمپلیت در یک بازه زمانی تغییر کند.

۴-۲- تشکیل گراف حالت^{۳۱} برای لاگ‌ها

در این قسمت به بررسی نحوه ساخت گراف حالت برای لاگ‌های نرم‌افزاری می‌پردازیم. شکل ۴ یک گراف حالت را نمایش می‌دهد.

^{۳۰}Anomaly detection

^{۳۱}State graph



شکل ۲۱: گراف حالت برای یک سیستم نرم افزاری

برای ساخت گراف حالت از لاگ‌های خام استفاده می‌شود. در ابتدای کار نیاز است که هر لاگ خام آنالیز شود و دو قسمت متغیر و اصلی از آن استخراج شود. پس از این مرحله، ساخت گراف حالت آغاز می‌شود. هر گراف حالت سه قسمت اصلی دارد: موجودیت^{۳۲}، حالت^{۳۳}، رخداد^{۳۴}. در شکل ۴، هر مستطیل یک موجودیت، هر متوازی الاضلاع یک حالت و هر چندضلعی یک رخداد را نمایش می‌دهد. عضو دیگر نمودار شکل ۴ یال‌ها هستند. یال‌ها به دو دسته فضایی^{۳۵} و زمانی^{۳۶} تقسیم می‌شوند. هر یال فضایی برای ایجاد یک رابطه بین یک موجودیت با یک رخداد و یا یک موجودیت با یک حالت استفاده می‌شود. هیچ‌گاه، یک رخداد با یک رخداد رابطه مستقیم ندارد و باید یک حالت و یا یک رخداد بین دو رخداد واسطه شود. حال به بررسی نحوه ساخت گراف فوق می‌پردازیم:

الف) حالت: این موارد در سیستم‌های نرم‌افزاری موارد مشخصی هستند و شامل مواردی مانند error, warning, run می‌شوند.

ب) رخداد: تشخیص موجودیت در یک سیستم نرم‌افزاری می‌تواند تاحدی چالشی باشد. معمولاً به عنوان یک قاعده سرانگشتی موارد با این ویژگی به عنوان یک موجودیت در نظر گرفته می‌شوند: ویژگی‌هایی که موارد متغیر بسیاری

^{۳۲} entity
^{۳۳} state
^{۳۴} event
^{۳۵} Spatial
^{۳۶} temporal

دارند و این موارد متغیر در چند حالت و یا رخداد تکرار شده‌اند. برای مثال در شکل ۴، ۱۰، ۱۱، ۱۲ در دو حالت و -XXX

^۱XX در دو حالت و دو رخداد تکرار شده است. بنابراین این موارد به عنوان یک موجودیت در نظر گرفته می‌شوند.

پ) لبه فضایی^{۳۷}: یک لبه فضایی، یک حالت یا رخداد را به یک موجودیت متصل می‌کند اگر آن حالت یا رخداد شامل آن موجودیت باشد. این نکته در شکل ۴ کاملاً مشخص است.

ت) لبه زمانی^{۳۸}: برای رسم لبه زمانی ابتدا از نظر هر موجودیت، حالت‌ها و رخدادها دسته‌بندی می‌شوند. به عبارت دیگر، حالت‌هایی که با رخداد مورد نظر در ارتباط هستند در یک گروه قرار می‌گیرند. همچنین، حالت‌هایی هم که با رخداد مورد نظر در ارتباط هستند هم در یک دسته قرار می‌گیرند. سپس براساس توالی زمان اتفاق افتادن حالت‌ها و یا رخدادها، بین هر دو رخداد یا هر دو حالت یک لبه زمانی رسم می‌شود. جهت این لبه زمانی براساس زمان است (از اتفاق زودتر به اتفاق دیرتر). این مورد در شکل ۴ کاملاً مشخص است.

۳-۲- بررسی انواع لاگ‌های سوئیفت

با توجه به آنکه هدف سیستم رسم گراف سلامت سیستم یا گراف جریان در سیستم است، ابتدا یک مطالع کامل بر روی انواع لاگ‌های موجود در سوئیفت برای سناریو آپلود و راه‌اندازی سرور انجام می‌دهیم. در این موارد سرورهای زیردخیل هستند:

- Object server
- Container server
- Proxy server
- Account server

در ابتدا سناریو آپلود را با هم بررسی می‌کنیم. فایل APT.TXT را در سوئیفت آپلود می‌کنیم. لاگ مربوط به پروکسی سرور به صورت زیر است :

^{۳۷} Spatial edge

^{۳۸} Temporal edge

> Sep 14, 2020 @ 18:39:57.793	sysloghost: hadi-VirtualBox port: 50,656 programname: proxy-server facility: local1 severity: err host: localhost type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.793 procid: - message: STDERR: 192.168.43.181 - - [14/Sep/2020 18:39:57] "GET /v1/AUTH_test/12.txt? limit=100&marker=test%2Fapt.txt&delimiter=%2F&prefix=test%2F&format=json HTTP/1.1" 200 699 0.069815 (txn: txa61ad0297dc14effa8b4d-005f5fb8fd) @version: 1 _id: 6TPqjXQ83w-t5q8mqzT _type: _doc _index: rstswift
> Sep 14, 2020 @ 18:39:57.744	sysloghost: hadi-VirtualBox port: 50,656 programname: proxy-server facility: local1 severity: err host: localhost type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.744 procid: - message: STDERR: (2974) accepted ('192.168.43.181', 50241) @version: 1 _id: qjPqjXQ83w-t5q8mqzT _type: _doc _index: rstswift _score: -
> Sep 14, 2020 @ 18:39:57.728	sysloghost: hadi-VirtualBox port: 50,656 programname: proxy-server facility: local1 severity: err host: localhost type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.728 procid: - message: STDERR: (2974) accepted ('192.168.43.181', 50240) @version: 1 _id: qTPqjXQ83w-t5q8mqzT _type: _doc _index: rstswift _score: -

شکل ۲۲ : نمونه اول لاگ پروکسی سرور در زمان آپلود

> Sep 14, 2020 @ 18:39:57.716	sysloghost: hadi-VirtualBox port: 50,656 programname: proxy-server facility: local1 severity: err host: localhost type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.716 procid: - message: STDERR: 192.168.43.181 - - [14/Sep/2020 18:39:57] "GET /v1/AUTH_test/12.txt? limit=100&delimiter=%2F&prefix=test%2F&format=json HTTP/1.1" 200 859 0.104513 (txn: tx9785cblcdfca4af9bf807-005f5fb8fd) @version: 1 _id: kTPqjXQ83w-t5q8mqzT _type: _doc _index: rstswift
> Sep 14, 2020 @ 18:39:57.716	sysloghost: hadi-VirtualBox port: 50,656 programname: proxy-server facility: local1 severity: err host: localhost type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.716 procid: - message: STDERR: 192.168.43.181 - - [14/Sep/2020 18:39:57] "GET /v1/AUTH_test/12.txt? limit=100&delimiter=%2F&prefix=test%2F&format=json HTTP/1.1" 200 859 0.110750 (txn: tx68570850957f437087db4-005f5fb8fd) @version: 1 _id: kjPqjXQ83w-t5q8mqzT _type: _doc _index: rstswift

شکل ۲۳ : نمونه دوم لاگ پروکسی سرور در زمان آپلود

لاگ مربوط به کانتینر سرور به صورت زیر است:

> Sep 14, 2020 @ 18:39:57.703	sysloghost: hadi-VirtualBox port: 50,656 programname: container-server facility: local5 severity: err host: localhost type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.703 procid: - message: STDERR: 127.0.0.1 - - [14/Sep/2020 18:39:57] "GET /sdb4/653/AUTH_test/12.txt? states=listing&delimiter=%2F&limit=100&prefix=test%2F&format=json HTTP/1.1" 200 1018 0.047543 (txn: tx68570850957f437087db4-005f5fb8fd) @version: 1 _id: ciPqjXQ83w-t5q8mqzT _type: _doc _index: rstswift
> Sep 14, 2020 @ 18:39:57.697	account_path: AUTH_test datetime: 14/Sep/2020:18:39:57 type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.697 partition_number: 653 request_method: GET message: 127.0.0.1 - - [14/Sep/2020:18:39:57 +0000] "GET /sdb4/653/AUTH_test/12.txt" 200 160 "GET http://192.168.43.149:8080/v1/AUTH_test/12.txt? states=listing&delimiter=%2F&limit=100&prefix=test%2F&format=json" "tx68570850957f437087db4-005f5fb8fd" "proxy-server 2974" 0.0387 "-" 2962 0 transaction_id: tx68570850957f437087db4-005f5fb8fd
> Sep 14, 2020 @ 18:39:57.689	account_path: AUTH_test datetime: 14/Sep/2020:18:39:57 type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.689 partition_number: 653 request_method: GET message: 127.0.0.1 - - [14/Sep/2020:18:39:57 +0000] "GET /sdb3/653/AUTH_test/12.txt" 200 160 "GET http://192.168.43.149:8080/v1/AUTH_test/12.txt? states=listing&delimiter=%2F&limit=100&prefix=test%2F&format=json" "tx9785cblcdfca4af9bf807-005f5fb8fd" "proxy-server 2974" 0.0487 "-" 2968 0 transaction_id: tx9785cblcdfca4af9bf807-005f5fb8fd

```

> Sep 14, 2020 @ 18:39:57.421 account_path: AUTH_test datetime: 14/Sep/2020:18:39:57 type: rsyslog @timestamp: Sep 14, 2020 @
18:39:57.421 partition_number: 653 request_method: HEAD message: 127.0.0.1 - - [14/Sep/2020:18:39:57
+0000] "HEAD /sdb4/653/AUTH_test/12.txt" 204 - "HEAD http://192.168.43.149:8080/v1/AUTH_test/12.txt?
states=listing&format=json" "txf42b270897734a50a4714-005f5fb8fd" "proxy-server 2974" 0.0010 "-" 2962 0
transaction_id: txf42b270897734a50a4714-005f5fb8fd user_agent: proxy-server 2974 sysloghost: hadi-

> Sep 14, 2020 @ 18:39:57.341 bytes_recvd_format: 0 auth_token: AUTH_tk148a6058e... bytes_sent: 0 bytes_per_second: 0
datetime: 14/Sep/2020/18/39/57 bytes_recvd: 0 type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.341
bytes_sum: 0 client_ip: 192.168.43.181 message: 192.168.43.181 192.168.43.181 14/Sep/2020/18/39/57
HEAD /v1/AUTH_test/12.txt/test/apt.txt HTTP/1.0 200 - SwiftStack-
Client/1.26.0%20%20Windows_NT%3B%20win32%2010.0.17763%20x64%29 AUTH_tk148a6058e... - -

> Sep 14, 2020 @ 18:39:57.100 sysloghost: hadi-VirtualBox port: 50,656 programname: proxy-server facility: local1 severity: err
host: localhost type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.100 procid: - message: STDERR:
192.168.43.181 - - [14/Sep/2020 18:39:57] "PUT /v1/AUTH_test/12.txt/test%2Fapt.txt HTTP/1.1" 201 333
1.727333 (txn: tx475338bcc4f7470da9233-005f5fb8fb) @version: 1 _id: vjPqjXQB3w-t5q8mp1x _type: _doc
_index: rstswift _score: -

> Sep 14, 2020 @ 18:39:57.098 bytes_recvd_format: 98 auth_token: AUTH_tk148a6058e... bytes_sent: 0
bytes_per_second: 56.81159420289855 datetime: 14/Sep/2020/18/39/57 bytes_recvd: 98 type: rsyslog
@timestamp: Sep 14, 2020 @ 18:39:57.098 bytes_sum: 98 client_ip: 192.168.43.181 message:
192.168.43.181 192.168.43.181 14/Sep/2020/18/39/57 PUT /v1/AUTH_test/12.txt/test/apt.txt HTTP/1.0 201 -
SwiftStack-Client/1.26.0%20%20Windows_NT%3B%20win32%2010.0.17763%20x64%29 AUTH_tk148a6058e... 98 - -

```

شکل ۲۴: لاگ مربوط به کانننر سرور در هنگام آپلود

لاگ مربوط به آجکت سرور به صورت زیر است:

```

> Sep 14, 2020 @ 18:39:57.340 sysloghost: hadi-VirtualBox port: 50,656 programname: object-server facility: local3 severity: err
host: localhost type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.340 procid: - message: STDERR:
127.0.0.1 - - [14/Sep/2020 18:39:57] "HEAD /sdb2/584/AUTH_test/12.txt/test/apt.txt HTTP/1.1" 200 487
0.002792 (txn: tx9598f70bc6ec41369f30d-005f5fb8fd) @version: 1 _id: -TPqjXQB3w-t5q8mpZkW _type: _doc
_index: rstswift _score: -

> Sep 14, 2020 @ 18:39:57.334 account_path: AUTH_test datetime: 14/Sep/2020:18:39:57 object_path: test/apt.txt type: rsyslog
@timestamp: Sep 14, 2020 @ 18:39:57.334 partition_number: 584 request_method: HEAD message: 127.0.0.1
- - [14/Sep/2020:18:39:57 +0000] "HEAD /sdb2/584/AUTH_test/12.txt/test/apt.txt" 200 98 "HEAD
http://192.168.43.149:8080/v1/AUTH_test/12.txt/test/apt.txt" "tx9598f70bc6ec41369f30d-005f5fb8fd" "proxy-
server 2974" 0.0017 "-" 2972 0 transaction_id: tx9598f70bc6ec41369f30d-005f5fb8fd content_length: 98

> Sep 14, 2020 @ 18:39:57.326 sysloghost: hadi-VirtualBox port: 50,656 programname: object-server facility: local3 severity: err
host: localhost type: rsyslog @timestamp: Sep 14, 2020 @ 18:39:57.326 procid: - message: STDERR:
(2972) accepted ("127.0.0.1", 38636) @version: 1 _id: -DPqjXQB3w-t5q8mpZkW _type: _doc
_index: rstswift _score: -

```

شکل ۲۵: لاگ آجکت سرور در هنگام آپلود

برای سناریو حذف فایل از سوییفت لاگ سه سرور فوق به صورت زیر است:

پروکسی سرور:

```

18:10:00.000 sysloghost: hadi-VirtualBox datetime: 15/Sep/2020/18/10/00 remote_addr: 192.168.43.181 client_etag: - request_path: /v1/AUTH_test/12.txt/test/fire%250wall.txt @timestamp: Sep 15, 2020 @
18:10:00.000 type: rsyslog port: 53,006 @version: 1 auth_token: AUTH_tk0e5cb3e52... message: 192.168.43.181 192.168.43.181 15/Sep/2020/18/10/00 DELETE
/v1/AUTH_test/12.txt/test/fire%250wall.txt HTTP/1.0 204 - SwiftStack-Client/1.26.0%20%20Windows_NT%3B%20win32%2010.0.17763%20x64%29 AUTH_tk0e5cb3e52... - - txfa0cb0befde4d39b1531-
005f610380 - 0.2711 - - 1600193408.602897882 1600193408.873960018 0 bytes_sent_format: 0 procid: - severity: info programname: proxy-server bytes_sent: 0 headers: - bytes_recvd: 0
referer: - log_info: - tags: swift, PROXY_SERVER_TRANSACTION policy_index: 0 status_int: 204 request_end_time: Sep 15, 2020 @ 18:10:00.873 bytes_recvd_format: 0 bytes_sum: 0

```

شکل ۲۶: لاگ پروکسی سرور در هنگام حذف

کانتینر سرور:

```

18:10:00.000 sysloghost: hadi-VirtualBox datetime: 15/Sep/2020:18:10:00 remote_addr: 127.0.0.1 request_path: /sdb4/653/AUTH_test/12.txt/test/fire20wall.txt server_pid: 7594 @timestamp: Sep 15, 2020
0 18:10:00.000 type: syslog port: 55,006 @version: 1 partition_number: 653 message: 127.0.0.1 - - [15/Sep/2020:18:10:00 +0000] "DELETE /sdb4/653/AUTH_test/12.txt/test/fire20wall.txt"
204 - "DELETE http://192.168.43.149:8080/sdb2/49/AUTH_test/12.txt/test/fire20wall.txt" "txfa6c0dbef0e4d39b1531-005f610380" "object-server 7601" 0.0201 "-" 7594 0 procid: - severity: info
programname: container-server account_path: AUTH_test container_path: 12.txt/test/fire20wall.txt tags: swift, CONT_SER_TRANSACTION referer: DELETE
http://192.168.43.149:8080/sdb2/49/AUTH_test/12.txt/test/fire20wall.txt policy_index: 0 status_int: 204 user_agent: object-server 7601 host: localhost

```

شکل ۲۷: لاگ کانتینر سرور در هنگام حذف

آبجکت سرور:

```

0 sysloghost: hadi-VirtualBox datetime: 15/Sep/2020:18:10:00 remote_addr: 127.0.0.1 request_path: /sdb2/49/AUTH_test/12.txt/test/fire20wall.txt server_pid: 7601 @timestamp: Sep 15, 2020
0 18:10:00.000 type: syslog port: 55,006 @version: 1 partition_number: 49 message: 127.0.0.1 - - [15/Sep/2020:18:10:00 +0000] "DELETE /sdb2/49/AUTH_test/12.txt/test/fire20wall.txt" 204
- "DELETE http://192.168.43.149:8080/v1/AUTH_test/12.txt/test/fire20wall.txt" "txfa6c0dbef0e4d39b1531-005f610380" "proxy-server 7602" 0.2244 "-" 7601 0 procid: - severity: info
programname: object-server account_path: AUTH_test container_path: 12.txt tags: swift, OBJ_SER_TRANSACTION referer: DELETE
http://192.168.43.149:8080/v1/AUTH_test/12.txt/test/fire20wall.txt policy_index: 0 status_int: 204 user_agent: proxy-server 7602 host: localhost transaction_id: txfac6c0dbef0e4d39b1531-

```

شکل ۲۸: لاگ آبجکت سرور در هنگام حذف

هر لاگ به صورت کلی در سامانه سوییفت به صورت زیر در syslog ثبت می شود :

{Datetime} {host} {programname} {message}

که به ترتیب از سمت چپ نشان دهنده: تاریخ و زمان ثبت لاگ، آدرس IP سرویس تولیدکننده لاگ ، نام

سرویس تولیدکننده لاگ و متن لاگ هستند.

```

Sep 5 15:42:57 172.19.0.1 account-server: 127.0.0.1 - - [05/Sep/2021:11:12:57 +0000] "GET /sdb3/802/AUTH_test"
404 - "GET http://127.0.0.1:8080/v1/AUTH_test?prefix=%00versions&format=json" "tx9f0261ad4e1c4366b0325-
006134a638" "proxy-server 29616" 0.0003 "-" 29603 -

```

شکل ۲۹: نمونه لاگ پروکسی سرور

همانگونه که ملاحظه میفرمایید، متن لاگ برای سرویسهای پروکسی و نودهای ذخیرهسازی ساختاریافته میباشد که در ادامه به تشریح ساختار آن پرداخته شده است؛ اما برای سایر سرویسها، متن لاگ از فرمت مشخصی تبعیت نمی کند. لاگ های پروکسی تمام درخواستهای API خارجی ارسال شده به سرور پروکسی را شامل میشوند. توضیح فیلدهای این ساختار لاگ در جدول ۱ آورده شده است. ساختار لاگ پروکسی سرور به صورت زیر است :

```

{client_ip} {remote_addr} {end_time.datetime} {method} {path} {protocol} {status_int} {referer} {user_agent}
{auth_token} {bytes_recv} {bytes_sent} {client_etag} {transaction_id} {headers} {request_time} {source}
{log_info} {start_time} {end_time} {policy_index}

```

شکل ۳۰: نمونه فرمت لاگ پروکسی

جدول یک - بررسی ساختار لاگ

فیلد	مقدار
Client ip	آدرس IP کلاینت نهایی با توجه به هدر درخواست
Remote addr	آدرس IP سرویس خارجی
End_time.datetime	زمان و تاریخ درخواست
method	نوع عملیات درخواستی
path	مسیر درخواست
protocol	پروتکل انتقال (HTTP یا HTTPS)
Status_int	کد پاسخ
Referrer	درخواستی که به واسطه دریافت آن، فرستنده اقدام به ارسال درخواست فعلی کرده است.
User_agent	نام سرویس طرف دیگر درخواست
Auth_token	توکن احراز هویت
Bytes_recvd	تعداد بایتهای خوانده شده از کلاینت به واسطه این درخواست
Bytes_sent	تعداد بایتهای فرستاده شده به کلاینت در بدنه پاسخ
Client_etag	مقدار Etag داده شده توسط کلاینت
Transaction_id	شناسه یکتای تراکنش که مقدار آن برای همه درخواستهای مربوط به یک تراکنش یکسان است
headers	هدرهای داده شده در درخواست
Request time	مدت زمان درخواست
source	منبع درخواست
Log_info	اطاعات اضافی
Start_time	زمان شروع درخواست با رزولوشن بالا
End_time	زمان خاتمه درخواست با رزولوشن بالا
Policy_index	مقدار اندیس سیاست ذخیرهسازی

۱-۳-۲- رسم گراف حالت برای داده‌های سوییفت

داده های سوییفت حالات مختلفی را در خود دارند. حالات مختلفی که می‌تواند برای این داده ها اتفاق بیفتد delete ، get ، auth و put است. برای آن که گراف حالت سوییفت را به دست بیاوریم ابتدا دیتاست کاملی از این داده‌ها و موارد و حالات ذکرشده در سوییفت را تولید می‌کنیم. دیتاست تولیدشده در نهایت شامل ۱۰۰۰۰ لاگ از حالات ذکرشده سوییفت است. بررسی یک نمونه از لاگ های سوییفت :

```
container-server: ۱۲۷,۰,۰,۱ - [۰۸/Dec/۲۰۲۰:۰۹:۲۳:۰۳ +۰۰۰۰] "PUT
/sdb۱/۶۶/AUTH_test/admin/۱۰.xlsx" ۲۰۱ - "PUT
http://localhost:۸۰۹۱/sdb۱/۱۳/AUTH_test/admin/۱۰.xlsx" "txcd۵۲f۸۴۵a۲۸۳۴۵۲۶b۴c۴d-۰۰۵fcf۴۵f۷"
```

"object-server ۷۵" ۰,۰۰۰,۷ "-" ۷۶ ۰
Dec ۸ ۰۹:۲۳:۰۳

ساختار لاگهای سوویفت ساختار مشخصی است. در ساختار این لاگ ها به طور معمول دو سرور پیامی را با هم رد و بدل میکنند. سرور های محدود هستند و شامل container server و proxy server و account server و object server هستند. پیام رد بدل شده بین این سرورها هم موارد put و delete و get و auth است. با توجه به ساختار مشخص این لاگ ها میتوان الگوریتم مطرح شده در مقاله را به در قالب مراحل زیر خلاصه سازی کرد:

الف) مرحله خواندن فایل شامل لاگ ها

```
f = open(fileName)
json_read = json.load(f)
```

در این مرحله فایل خوانده می شود و به یک دیکشنری تبدیل می شود.

ب) مرحله تبدیل لاگ ها به فرمت استاندارد

در این قسمت هر لاگ به اجزای اصلی آن تبدیل می شود (نام سرورها و متودها). در این قسمت هر سرور به صورت زیر خلاصه میشود:

Proxy-server: p
Account-server: a
Object-server: o
Container server: c

یک نمونه از خلاصه سازی کد در این قسمت انجام میشود با مثال زیر قابل توضیح است :
نمونه لاگ :

```
container-server: ۱۲۷,۰,۰,۱ - [۰۸/Dec/۲۰۲۰:۰۹:۲۳:۰۳+۰۰۰۰] "PUT
/sdb۱/۶۶/AUTH_test/admin/۱۰.xlsx" ۲۰۱ - "PUT
http://localhost:۸۰۹۱/sdb۱/۱۱۳/AUTH_test/admin/۱۰.xlsx" "txcd۵۲f۸۴۵a۲۸۳۴۵۲۶b۴c۴d-۰۰۵fcf۴۵f۷"
"object-server ۷۵" ۰,۰۰۰,۷ "-" ۷۶ ۰
Dec ۸ ۰۹:۲۳:۰۳
```

خلاصه شده : o Auth C

```
def purifylogs(self, json_read):
    pure_logs = []
    transaction_ids = []
    datetime = []
    for i in range(len(json_read)):
        if "HEAD" in json_read[i]["request_method"]:
            try:
                str1 = json_read[i]["user_agent"]
                str2 = json_read[i]["programname"]
                pure_logs.append(self.define(str1) + "-" + "HEAD" + "-" +
self.define(str2))
                transaction_ids.append(json_read[i]['transaction_id'])
                datetime.append(json_read[i]['@timestamp'])
                continue
            except:
                pass
```

```

        if "GET" in json_read[i]["request_method"]:
            try:
                str1 = json_read[i]["user_agent"]
                str2 = json_read[i]["programname"]
                pure_logs.append(self.define(str1) + "-" + "GET" + "-" +
self.define(str2))
                transaction_ids.append(json_read[i]['transaction_id'])
                datetime.append(json_read[i]['@timestamp'])
                continue
            except:
                pass

        if "DELETE" in json_read[i]["request_method"]:
            try:
                str1 = json_read[i]["user_agent"]
                str2 = json_read[i]["programname"]
                pure_logs.append(self.define(str1) + "-" + "DELETE" + "-" +
+ self.define(str2))
                transaction_ids.append(json_read[i]['transaction_id'])
                datetime.append(json_read[i]['@timestamp'])
                continue
            except:
                pass

        if "PUT" in json_read[i]["request_method"]:
            try:
                str1 = json_read[i]["user_agent"]
                str2 = json_read[i]["programname"]
                pure_logs.append(self.define(str1) + "-" + "PUT" + "-" +
self.define(str2))
                transaction_ids.append(json_read[i]['transaction_id'])
                datetime.append(json_read[i]['@timestamp'])
                continue
            except:
                pass

    return np.array(pure_logs), np.array(transaction_ids),
np.array(datetime)

    @staticmethod
    def define(string):
        if "proxy" in string:
            return "p"
        if "account" in string:
            return "a"
        if "container" in string:

```

```

        return "c"
    if "object" in string:
        return "o"
    if "python" in string:
        return "s"

```

پ) مرحله دسته بندی اولیه با درنظر گرفتن transaction id

در این مرحله، لاگهایی که آیدی یکسان دارند به عنوان یک فرآیند شناخته میشوند. در حقیقت ما لاگ های با آیدی یکسان را با حفظ ترتیب به عنوان یک فرآیند درنظر میگیریم.

```

@staticmethod
def extractFlows(pure_logs, transaction_ids, datetime):

    final_logs = []
    final_date_time = []
    final_tx = []
    unique_transaction_ids = np.unique(transaction_ids)
    for i in range(len(unique_transaction_ids)):
        temp_logs = []
        temp_datetime = []
        temp_tx = []
        print(i)
        for j in range(len(transaction_ids)):
            if transaction_ids[j] == unique_transaction_ids[i]:
                temp_logs.append(pure_logs[j])
                temp_datetime.append(datetime[j])
                temp_tx.append(transaction_ids[j])

        final_logs.append(temp_logs)
        final_date_time.append(temp_datetime)
        final_tx.append(temp_tx)

    return final_logs, np.array(final_date_time), np.array(final_tx)

```

ت) سورت کردن لاگ های هر فرآیند

در این قسمت لاگ های هر فرایند با توجه به فیلد datetime سورت می شوند. این قسمت از آن نظر حائز اهمیت است که در فلو نهایی ترتیب درستی را نمایش دهیم.

```

@staticmethod
def sortlogs(logs_flows, datetime):

    for i in range(len(logs_flows)):
        temp_logs_flows = copy.deepcopy(logs_flows[i])
        temp_date_time = copy.deepcopy(datetime[i])
        for k in range(len(temp_date_time)):

```

```

        try:
            temp_date_time[k] =
int((temp_date_time[k].split("T")[1].split(":")[0])) * 3600 + (
                int(temp_date_time[k].split("T")[1].split(":")[1])) *
۶۰ + \
                    float(temp_date_time[k].split("T")[1].
split(":")[2].split("Z")[0])
            # int((temp_date_time[k].split("2022:")[1].split(":")[0]))
* 3600 + (
                # int(temp_date_time[k].split("2022:")[1].split(":")[1]))
* 60 + \
                    # int(temp_date_time[k].split("2022:")[1].split(":")[2])
        except:
            temp_date_time[k] = 0

        sorted_temp_date_time = np.array([y for y, x in
sorted(zip(temp_date_time, temp_logs_flows))])
        sorted_temp_logs_flows = np.array([x for y, x in
sorted(zip(temp_date_time, temp_logs_flows))])

        logs_flows[i] = sorted_temp_logs_flows

    return logs_flows

```

ث) مرحله کلاستر کردن فرآیندها

در این مرحله فرآیندهای یکسان با هم دیگر در یک کلاستر قرار میگیرند. در حقیقت یکی از آن ها به عنوان نماینده در ادامه الگوریتم حضور خواهد یافت. معیارهای کلاستر کردن فرآیندها به صورت است:

- برابر بودن طول فرآیند
- برابر بودن نظیر به نظیر لاگ ها

```

def clusterlogs(self, sorted_log_flows, tx):
    his = [sorted_log_flows[0]]
    his_tx = [tx[0]]
    for i in range(len(sorted_log_flows)):
        found = 0
        for j in range(len(his)):
            if len(his[j]) == len(sorted_log_flows[i]):
                count = 0
                for k in range(len(his[j])):
                    if his[j][k] == sorted_log_flows[i][k]:
                        count = count + 1
                    continue
                if count == len(his[j]):
                    found = 1

```

```

        if found == 0 and len(sorted_log_flows[i]) > 1:
            his.append(sorted_log_flows[i])
            his_tx.append(tx[i])
    return his, his_tx

```

ج) مرحله تولید فایل اکسل برای هر کلاستر

در این مرحله، سعی می‌شود که برای هر کلاستر یک فایل اکسل تولید شود که نام آن فایل اکسل شماره کلاستر یا فلو به اضافه transaction_id آن فلو است.

```

def generate_excel(self, clustered_sorted_log_flows, tx):
    for i in range(len(clustered_sorted_log_flows)):
        df = pd.DataFrame(clustered_sorted_log_flows[i])
        filename = str(i) + "_" + str(tx[i][0]) + '.csv'
        df.to_csv(filename)

```

فصل 3- فصل سوم : مصورسازی

۱-۱-۳- پایگاه داده گراف

گراف در واقع مجموعه ای موجودیت ها و ارتباط بین آن هاست، که به صورت منظم ذخیره سازی شده اند. معمولاً در پایگاه داده هایی که به صورت سطر و ستون ذخیره می شوند، سرعت دسترسی به اطلاعات خوب و منطقی به نظر میرسد. به خصوص زمانی که از Indexing استفاده شود، این در حالی است که در بسیاری از مواقع به دلیل نیاز به سرعت بالای دسترسی به اطلاعات، سرعت واکنشی و درج داده ها، در پایگاه داده های رابطه ای، مناسب به نظر نمی رسد. این اتفاق معمولاً زمانی رخ می دهد که میخواهید، یک یا چند Join مختلف، بر روی جداول مختلف یک پایگاه داده بزنید.

در پایگاه داده های رابطه ای بستگی به نوع سناریو که داریم بعضی اوقات نیاز هست که برای پیاده سازی روابط خیلی پیچیده، داده ها را در قالب جداول متعدد ذخیره کنیم. این جا ما مجبور به نوشتن Query ها و Join های زیادی هستیم که بتوانیم خروجی و گزارش دلخواه مان را بسازیم. همینطور که در جریان هستید هرچقدر کوئری های ما بزرگ و پیچیده بشود و روابط بین جداول زیاد شود، هزینه اجرای دستور ما (Cost of Query) بالا میرود و سرعت اجرای دستورات کمتر می شود.

۲-۱-۳- استفاده از Neo4j

Neo4j یکی از معروف ترین سیستم مدیریت دیتابیس است که از خانواده دیتابیس های NoSQL به شمار می رود. Neo4j با Mysql و یا MongoDB متفاوت و دارای ویژگی های خاص خود است که این ویژگی ها، آن را در مقایسه با سایر سیستم های مدیریت دیتابیس، خاص می کند. دیتابیس Neo4j داده ها را به صورت نمودار ذخیره و ارائه می کند و نه به صورت جدول یا JSON. در دیتابیس Neo4j کل داده ها توسط node ها نمایش داده می شوند و شما می توانید رابطه ای بین node ها ایجاد کنید. این بدان معناست که کل مجموعه دیتابیس مانند یک نمودار خواهد بود، به همین دلیل آن را از دیگر سیستم های دیگر منحصر به فرد می کند.

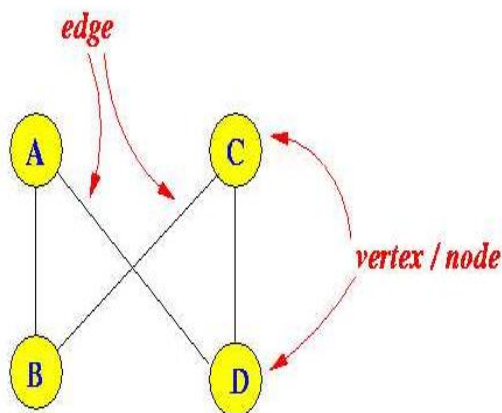
برخی از ویژگی های منحصربه فرد وجود دارد که باعث می شود Neo4j را به عنوان جایگزینی برای دیگر سیستم های مدیریتی دیتابیس در نظر گرفت. همچنین باید گفت، دیتابیس Neo4j برای ایجاد روابط بین داده ها، نیازی به Primary key و Foreign key ندارد. در اینجا می توان هر ارتباطی بین هر node مورد نظر خود اضافه کنید. این قابلیت در Neo4j، آن را برای داده های شبکه (Network) بسیار مناسب می کند.



شکل ۳۱: گراف Neo4j

۳-۱-۳- مفاهیم اصلی Graph Database ها :

- گره‌ها یا node نشان دهنده موجودیت یا entity های ما هستند .. مثل مردم، کسب و کار ، حساب‌های کاربری.
- یال‌ها یا edge به‌طور کلی یال‌ها وظیفه نمایش روابط را در دیتابیس بر عهده دارند. همچنین بسیاری از اطلاعات مهم در یال‌ها ذخیره می‌شوند.



شکل ۳۲: رابطه گرافی

۳-۱-۴- چه موقع از ساختار گراف باید استفاده کرد ؟

- مواقعی که برنامه شما دارای داده‌ها با ساختار سلسله مراتبی (hierarchy) است. مانند چارت سازمانی.
- زمانی موجودیت‌های شما دارای روابط پیچیده چند به چند باشد.

۳-۱-۵- مزیت استفاده از پایگاه داده گراف:

- برای داده های متصل بسیار مناسب هست.
- برای انسان قابل خواندن و یادگیری و آسان هست.
- داده های نیم ساخت یافته را به راحتی ارایه می کند.
- برای بازیابی و پیمایش داده های متصل ساده هست.

۳-۱-۶- معایب استفاده از پایگاه داده گراف:

- در تعداد استفاده از گره ها و روابط و خواص دارای محدودیت هست.
- از sharding پشتیبانی نمی کند.



شکل ۳۳: نمونه ای از گراف neo4j

۳-۱-۷- راه اندازی Neo4j:

در این پروژه، برای استفاده از این ابزار، از نسخه داکر Neo4j استفاده شده که به سهولت و بدون توجه به سیستم عامل و سایر مشخصات محیط توسعه و تنها با داشتن سرویس داکر قابل استفاده است.

داکر برنامه ای رایانه ای و ابزاری است که ایجاد، توسعه و اجرای اپلیکیشن ها را با استفاده از کانتینر (container) آسان می کند. کانتینر به توسعه دهندگان این امکان را می دهد که تمام پیش نیازها و نیازمندی های اپلیکیشن خود را برای استفاده و اجرا جمع آوری کنند؛ مانند کتابخانه ها (Libraries) و زیرساخت های لازم.

می‌توان گفت که داکر به توسعه‌دهندگان این اطمینان را می‌دهد که می‌توانند اپلیکیشن خود را بدون نگرانی از سیستم‌های میزبان برنامه خود در مرحله توسعه و تست و بر روی سیستم‌های مختلف، بدون هیچ‌گونه اشکال و به صورت کاملاً مشابه اجرا کنند و مشکلی از بابت تغییر سیستم‌عامل‌های اجراکننده برنامه خود نخواهند داشت.

داکر تا حدی شبیه به ماشین مجازی (Virtual Machine) است با این تفاوت که در ماشین مجازی، قسمتی از سخت‌افزار سیستم به ماشین مجازی اختصاص داده می‌شود و روی آن یک سیستم‌عامل کامل نظیر ویندوز یا لینوکس نصب می‌شود. در واقع می‌توان گفت در ماشین مجازی امکانات سخت‌افزاری سیستم تقسیم می‌شود و بر روی هر قسمت، سیستم‌عامل بخصوصی بالا می‌آید اما در داکر این طور نیست. در داکر امکانات سخت‌افزاری به تناسب نیاز هر کانتینر به صورت موقت اختصاص داده می‌شود و داکر این امکان را فراهم می‌آورد که اپلیکیشن‌ها برای مثال روی کرنل لینوکس اجرا شوند. در این حالت دیگر نیازی به نصب پیش‌نیازها و نیازمندی‌هایی که اپلیکیشن ما می‌خواهد و به طور پیش‌فرض روی سیستم وجود ندارد، نیست.

کانتینر، یک واحد استاندارد نرم‌افزار است که کدها و تمام نیازمندی‌های آن را جهت اجرا و توسعه اپلیکیشن‌ها، جمع می‌کند و باعث می‌شود که این اجرا به طور سریع و قابل اعتماد انجام شود. کانتینرها، نرم‌افزار را از محیط خودش ایزوله می‌کنند و اطمینان می‌دهند که علی‌رغم تفاوت‌های احتمالی در سیستم‌های میزبان، نرم‌افزار به طرز مشابه و یکسان اجرا خواهد شد.

کانتینرهای داکر قسمتی از نرم‌افزار را در یک سیستم فایل کامل تعبیه می‌کند. به صورتی که شامل هر آنچه جهت اجرا شدن (مانند کد زمان اجرا، ابزارهای سیستم و تنظیمات سیستم) لازم است و هر آنچه که می‌تواند بر روی یک سرور نصب شود می‌باشد. این امر اجرای برنامه را به صورت ثابت در هر نوع محیطی تضمین می‌کند.

به طور کلی می‌توان گفت داکر این قابلیت را به ما می‌دهد تا یک اپلیکیشن را در یک محیط ایزوله به نام کانتینر (Container) اجرا کنیم. ایزوله بودن و امنیت کانتینرها باعث می‌شود که بتوانیم تعداد زیادی کانتینر را روی ماشین میزبان اجرا کنیم. کانتینرها سبک و سریع هستند زیرا نیازی ندارند که مانند ماشین‌های مجازی سر بار Hypervisor را به ماشین تحمیل کنند و مستقیماً روی هسته (Kernel) کامپیوتر سرویس‌دهنده اجرا می‌شوند. این بدان معناست که می‌توان تعداد بیشتری کانتینر را روی یک ماشین نسبت به حالتی که از ماشین مجازی استفاده می‌کنیم اجرا کرد. البته حتی این قابلیت را نیز داریم که از داکر درون ماشین مجازی نیز استفاده کنیم.

Container یک نمونه قابل اجرای یک Image است. می‌توان یک کانتینر را به وسیله Docker API یا کلاینت ایجاد، اجرا، حذف و یا متوقف کرد.

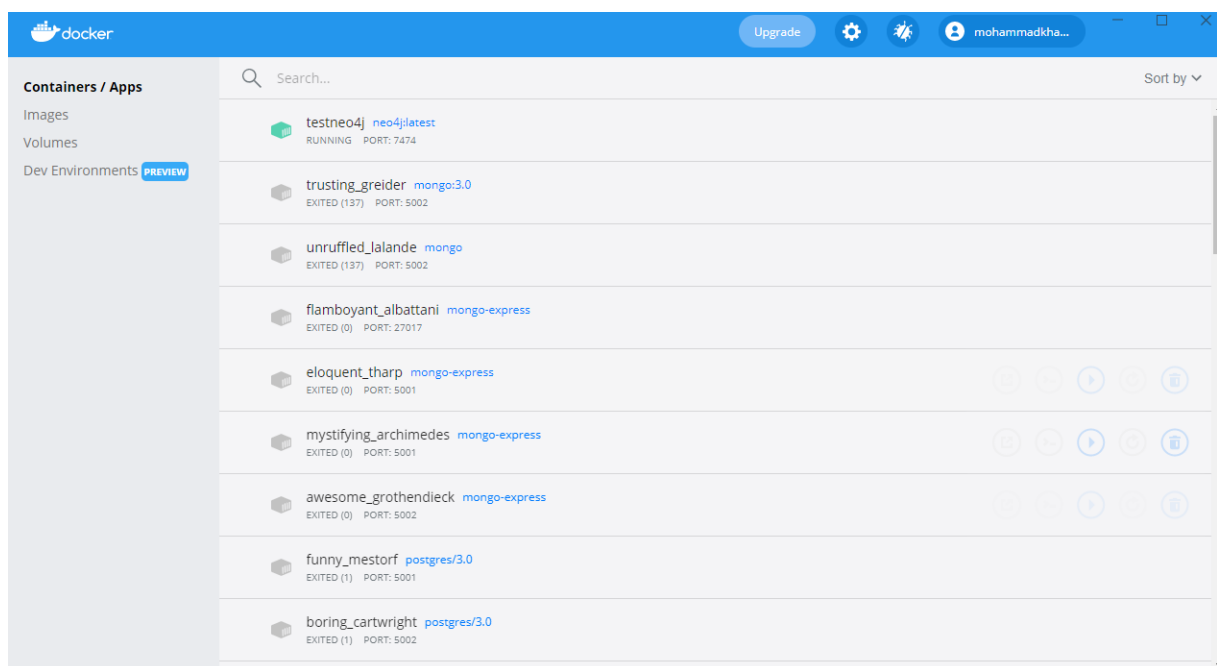
معمولاً کانتینرها از دیگر کانتینرهای در حال اجرا به خوبی ایزوله می‌شوند. شما می‌توانید میزان ایزوله بودن کانتینرها را در مواردی مانند دسترسی شبکه‌ای، حافظه مورد استفاده آن‌ها و سیستم‌های پایه‌ای مورد استفاده‌شان تنظیم کنید. یک کانتینر در واقع با Image آن و همه تنظیماتی که در زمان اجرای آن انجام می‌دهید تعریف می‌شود. هنگامی که کانتینر حذف می‌شود، هر تغییر فایلی که در آن به وجود آمده از بین می‌رود.

Docker Container Image یک پکیج نرم‌افزاری سبک، خوداتکا (Stand-alone) و قابل اجراست. ایمیج‌ها حاوی تمامی نیازمندی‌های اپلیکیشن برای اجرا مانند کدها، ابزارهای سیستمی، کتابخانه‌ها (Libraries) و تنظیمات

سیستمی هستند. Image ها در زمان اجرا تبدیل به کانتینر شده و روی Docker engine اجرا می‌شوند. ایمیج ها برای هر دو نوع اپلیکیشن‌های بر پایه لینوکس و ویندوز در دسترس هستند.

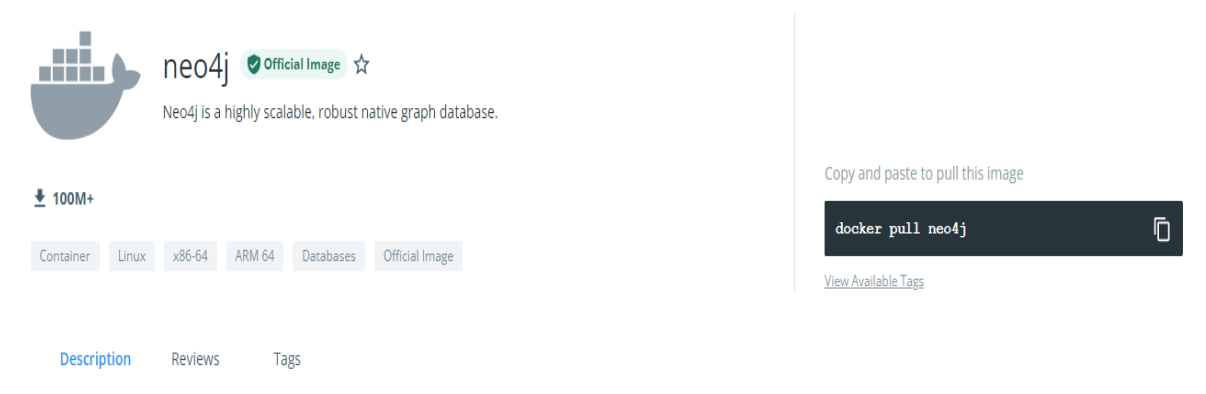
یک Image در واقع یک Read-Only Template حاوی دستوراتی برای ساخت یک کانتینر داکر است. معمولاً استفاده ما از Image ها به این صورت است که با تغییر یک Image، Image دیگری را که برای ما مناسب‌تر است می‌سازیم. مثلاً می‌توانیم Image ای بسازیم که برپایه Image اوبونتو است اما یک سرور Apache نیز روی آن نصب می‌شود. شما می‌توانید از Image های آماده استفاده کنید و یا Image مناسب خود را بسازید. برای ساختن Image جدید کافی است یک Dockerfile ساده بسازید که در آن دستوراتی برای نشان دادن چگونگی ساختن و اجرا کردن Image موجود است. وقتی که Dockerfile خود را تغییر می‌دهیم و Image جدیدی می‌سازیم، تنها بخش‌های جدید دوباره ساخته می‌شوند.

برای نصب داکر در ویندوز، بر روی بسته نرم افزاری Docker Desktop Installer.exe که قبلاً دانلود کرده اید دو بار کلیک کنید تا نصاب اجرا شود. در پنجره آغازین فرایند نصب مطمئن باشید که تیک گزینه Enable Hyper-V Windows Features خورده باشد. سپس با زدن دکمه OK مجوز نصب داکر را صادر کرده و فرایند نصب را تا انتها مطابق دستورالعمل‌های خواسته شده توسط نصاب، کامل کنید. بعد از نصب موفق و مشاهده پیغام Installation Succeeded، دکمه Close and Logout را انتخاب کنید تا فرایند نصب کامل شود. با انتخاب دکمه مذکور، شما به طور خودکار از حساب کاربری ویندوزتان خارج خواهید شد لذا لازم است مجدداً وارد حساب کاربری ویندوزتان شوید.



شکل ۳۴: تصویری از کانتینرها در داکر دسکتاپ

ایمیج Neo4j با تنظیمات پیش فرض عامی که معمولاً نیازی به تغییر ندارند در دسترس است^{۳۹}.



شکل ۳۵: ابزار neo4j در داکرهاب

برای اجرای آن، پس از پول کردن ایمیج از داکرهاب، کفایت دستور زیر را از ترمینال اجرا کنیم:

```
docker run \
  --name testneo4j \
  -p7474:7474 -p7687:7687 \
  -d \
  -v $HOME/neo4j/data:/data \
  -v $HOME/neo4j/logs:/logs \
  -v $HOME/neo4j/import:/var/lib/neo4j/import \
  -v $HOME/neo4j/plugins:/plugins \
  --env NEO4J_AUTH=neo4j/test \
  --env NEO4J_dbms_connector_https_advertised_address="localhost:7473" \
  --env NEO4J_dbms_connector_http_advertised_address="localhost:7474" \
  --env NEO4J_dbms_connector_bolt_advertised_address="localhost:7687" \
  neo4j:latest
```

شکل ۳۶: دستورات راه اندازی neo4j

پس از راه اندازی موفق کانترینر، در اپلیکیشن پایتونی خود، با استفاده از کتابخانه neo4j و پورت ۷۶۷۸ و پروتکل Bolt می‌توانیم با این ابزار کار کرده و پیام‌های مورد نظر خود را که شامل دستورات مربوط به ایجاد و پاک کردن گره‌ها و یال‌ها می‌شود، برای آن ارسال کنیم.

^{۳۹}https://hub.docker.com/_/neo4j/

همچنین با استفاده از مرورگر و روی پورت ۷۴۷۴ می‌توان به داشبورد Neo4j دسترسی داشت که به صورت بصری اطلاعات را نمایش می‌دهد، و همچنین می‌توان با استفاده از اسکریپت‌های به زبان cypher کوئری‌های مشابه SQL را در کنسول آن اجرا نمود و فیلترهای دلخواه را روی نودها و گره‌ها اعمال کرد.

شکل ۳۷: ورود به داشبرد neo4j

پس از وارد کردن یوزرنیم و پسورد وارد فضای زیر می‌شویم:

شکل ۳۸: داشبرد neo4j

۸-۱-۳- نکات مهم در استفاده از ابزار neo4j

- در استفاده از این ابزار حتما در ابتدا متودی با عنوان clear graph درست کنید و قبل از رسم هر گونه گراف، این متد را صدا بزنید. در غیر اینصورت، گراف قبلی و گراف جدید با هم نمایش داده می‌شوند و شکل بسیار شلوغی ایجاد می‌شود.

- در این پروژه هر خط از لاگ به منزله یک یال برای گراف هست و هر تمپلیت یک نود گراف را برای ما تشکیل می‌دهد.

۹-۱-۳- کدهای پیاده سازی کلاس گراف neo4j

(۱) ساختار و معماری فولدرهای پروژه به صورت زیر است:

```
C:\Users\mj\Desktop\sol_project>tree /A
Folder PATH listing
Volume serial number is E00E-5E81
C:..
|   .idea
|   |   inspectionProfiles
|   |   anomalies
|   |   model
|   |   |   flow_model
|   |   |   graph_model
|   |   __pycache__
|
|   LogDB_main.json
|   logfunction.py
|   logs_status.csv
|   neo4jhandler.py
|   nodes.csv
|   previous_test_run.txt
|   run_neo4j.txt
|   sol.json
|   test_run.py
|   train_run.py
```

روند اجرای برنامه، پردازش لاگ‌ها و تولید گراف به صورت زیر طی می‌شود:

```
from typing import Dict

class Neo4jHandler:
    graphDB_Driver = None

    def __init__(self, config: Dict) -> None:
        uri = config["uri"]
        userName = config["userName"]
        password = config["password"]
        self.graphDB_Driver = GraphDatabase.driver(
            uri, auth=(userName, password)
        )
```

```

def create_new_node(self, node):
    create_node = \
        "CREATE \n" + \
        f"({node}:node " + "{ name: " + f"\n{node}\n" + "})"
    with self.graphDB_Driver.session() as graphDB_Session:
        graphDB_Session.run(create_node)

    return create_node

def create_new_edge(self, node_1, node_2, edge, edge_name):
    create_edge = \
        "MATCH (u:node {name:" + f"'{node_1}'" + "}), (r: node {name: " + \
        f"'{node_2}'" + "}) " + \
        f"CREATE(u)-[:{edge}] " + \
        " {source_destination: " + f"'{edge_name}'" + "}]>(r)"

    with self.graphDB_Driver.session() as graphDB_Session:
        graphDB_Session.run(create_edge)

    return create_edge

def clear_graph(self):
    cqldelete1 = "match (a) -[r] -> () delete a, r"
    cqldelete2 = "match (a) delete a"
    with self.graphDB_Driver.session() as graphDB_Session:
        graphDB_Session.run(cqldelete1)
        graphDB_Session.run(cqldelete2)

```

در این قسمت در ادامه قسمت ۲-۳-۲، گراف سیستم را به دست می‌آوریم. در ابتدا کلاسترهای خود را به صورت یک رابطه درمی‌آوریم. بدین منظور ابتدا نودهای موجود در کلاسترها را استخراج می‌کنیم و سپس با پیمایش کلاسترها، مشخص می‌کنیم که هر نود با چه نود دیگری در ارتباط است. در قطعه کد زیر nodes مشخص کننده نودها و connections مشخص کننده ارتباطات هر نود است. Nodes, connections مستقیماً به neo4j داده می‌شوند.

```

@staticmethod
def extractconnections(logs_flows):
    nodes = []
    for i in range(len(logs_flows)):
        for j in range(len(logs_flows[i])):
            if logs_flows[i][j] not in nodes:
                nodes.append(logs_flows[i][j])
    nodes = np.array(nodes)
    connections = []
    flow_index = []
    for i in range(len(nodes)):
        temp_connections = []

```



```

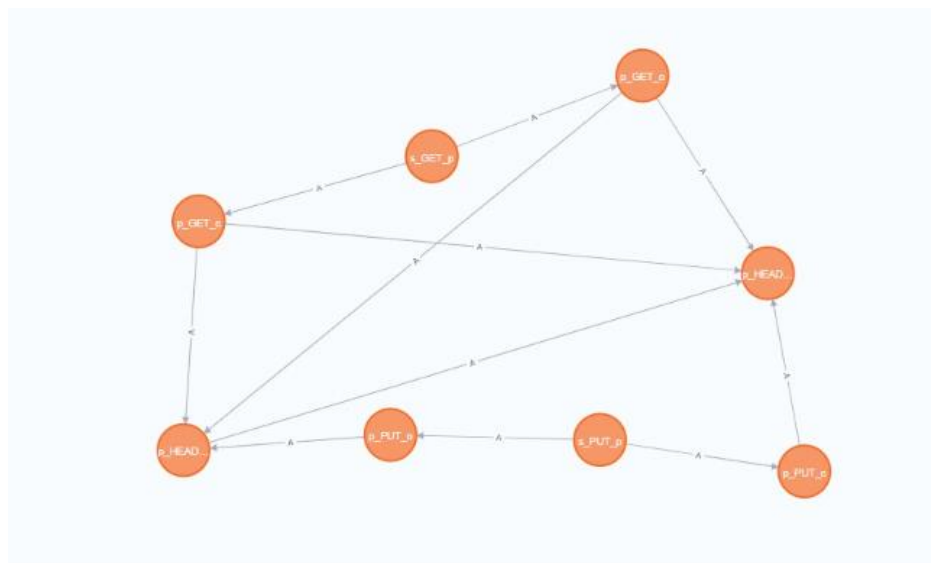
temp_flow_index = []
for k in range(len(logs_flows)):
    for p in range(len(logs_flows[k])):
        if nodes[i] == logs_flows[k][p] and p < len(logs_flows[k])
- 1:
            index_of_node = np.where(nodes == logs_flows[k][p +
\)\[.])

            if index_of_node != i:
                temp_connections.append(index_of_node[0])
                temp_flow_index.append(k)
            connections.append((np.array(temp_connections)))
            flow_index.append((np.array(temp_flow_index)))

return nodes, connections, flow_index

```

گراف سیستم در Neo4j به صورت زیر است:



شکل ۳۹: خروجی سیستم

۱۰-۱-۳- اضافه کردن **programname**, **method** و **user-agent** به عنوان ویژگی های نود در Neo4j

در این قسمت ویژگی های زیر را به یک نود در گراف خود اضافه میکنیم:

- Programname
- User-agent
- Method

به عنوان مثال در نمونه لاگ P-GET-a موارد فوق به صورت زیر مشخص می شوند :

- Programname:account-server

- User-agent: proxy-server
- Method: Get

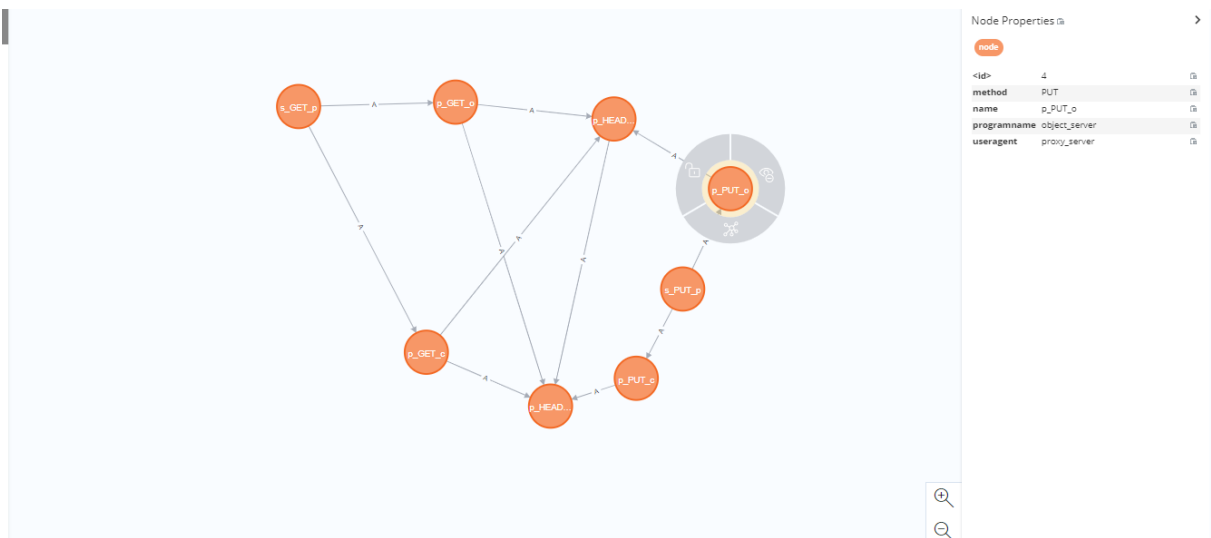
برای این کار تابع create_node را به صورت زیر اصلاح می‌کنیم :

```
def create_new_node(self, node):
    useragent = node.split("_")[0]
    method = node.split("_")[1]
    programname = node.split("_")[2]
    if useragent == "p":
        useragent = "proxy_server"
    elif useragent == "a":
        useragent = "account_server"
    elif useragent == "o":
        useragent = "object_server"
    elif useragent == "c":
        useragent = "container_server"
    else:
        useragent = "s"

    if programname == "p":
        programname = "proxy_server"
    elif programname == "a":
        programname = "account_server"
    elif programname == "o":
        programname = "object_server"
    elif programname == "c":
        programname = "container_server"
    else:
        programname = "s"

    create_node = \
        "CREATE \n" + \
        f"({node}:node " + "{ name: " + f"\"{node}\"" , programname: " +
        f"\"{programname}\"" , method: " + f"\"{method}\"" , useragent: " +
        f"\"{useragent}\"" + "}"
```

خروجی کد فوق در j‌neo به صورت شکل زیر مشخص می‌شود:



شکل ۴۰: نمایش ویژگی ها در نود

فصل 4- فصل چهارم: سیستم کشف ناهنجاری

۱-۴- معماری سویفت و اهمیت کشف ناهنجاری

در ابتدا، یک بحث کوتاهی روی معماری سویفت انجام می‌دهیم و یک مروری بر روی سرویس‌ها داریم. این قسمت مشخص می‌کند که مقصود ما از ناهنجاری چه مفهومی است. اجزای سویفت همانطور که در فصل‌های پیشین بحث شد به صورت زیر است:

سرور پروکسی: بخشی که مسئول اتصال همه اجزای سویفت به همدیگر است. با هر درخواست، پروکسی به دنبال محل حساب، کانتینر و یا شی جستجو کرده و درخواست را مطابق آن در مسیر درست خود هدایت می‌کند. در برخی مواقع پروکسی مسئول رمزگذاری و رمزگشایی اشیاء نیز هست. همچنین تعداد زیادی از خطاها و عدم موفقیت‌ها در پروکسی مدیریت می‌شوند، مثلاً اگر هنگام قرار دادن شی، سرور در دسترس نباشد، پروکسی از رینگ درخواست سرور جایگزین کرده و درخواست را به آن مسیر هدایت می‌کند. اشیایی که از یا به سرور شی منتشر می‌شوند، مستقیماً از طریق پروکسی به یا از سمت کاربر هدایت می‌شوند.

رینگ: نماینده نگاشتی بین نام موجودیت‌های ذخیره شده بر روی دیسک و محل فیزیکی آن‌هاست. در مورد هر سیاست ذخیره سازی، رینگ‌های مختلفی برای حساب، کانتینر و اشیاء وجود دارند. هر مولفه‌ای که قصد انجام عملیاتی روی شی، کانتینر یا حساب داشته باشد، باید با رینگ متناسب آن تعامل کند تا محل آن در کلاستر را پیدا کند.

سرور شی: سرور ذخیره سازی ساده که می‌تواند برای ذخیره سازی، بازیابی و حذف اشیاء ذخیره شده روی دستگاه‌ها استفاده شود. اشیاء با استفاده از مسیری که با نام هش شی و زمان عملیات تولید می‌شود، ذخیره می‌شوند.

سرور کانتینر: وظیفه اصلی سرور کانتینر این است که لیست بندی اشیاء را مدیریت کند. این سرور اطلاعاتی از محل اشیاء ندارد، بلکه فقط این که چه اشیایی داخل یک کانتینر خاص هستند. لیست‌های مذکور به صورت فایل دیتابیس sqlite ذخیره شده و مشابه خود اشیاء در سراسر کلاستر تکرار می‌شوند.

سرور حساب: مشابه سرور کانتینر، با این تفاوت که وظیفه آن نگهداری و مدیریت لیست بندی کانتینر هاست.

با توجه به سرویس‌های معرفی شده، هر آن چه که گراف تولید شده در فصل پیش را پیروی نکند به عنوان ناهنجاری در نظر گرفته می‌شود. منبع لاگ‌های ما در این قسمت، لاگ‌های تولید شده توسط syslog است.

ابزار سوئیفت لاگ نسبتاً مفصلی تولید می‌کند که می‌تواند برای موارد متعددی استفاده شود، از جمله: نظارت بر کارکرد کلاسترها، محاسبات مربوط به بهره‌وری، و ثبت سوابق حسابرسی^{۴۰}. لاگ‌های سوئیفت برای syslog ارسال می‌شوند و با توجه به سطح لاگ و امکانات syslog سازماندهی می‌شوند. تمام لاگ‌های مربوط به یک درخواست شناسه تراکنش یکسانی دارند. لاگ‌های موجود در syslog را ابتدا پردازش می‌کنیم و به فرمت زیر که یک دیکشنری است تبدیل می‌کنیم. در زیر چند نمونه از لاگ‌های مرتب شده آمده است:

```
[
{
  "": 0,
  "log_info": "-",
  "@version": 1,
  "facility": "local1",
  "bytes_sent": 0,
  "bytes_recvd": 3131,

  "bytes_recvd_format": 3131,
  "object_path": "folder8/file173.txt",
  "tags": "['swift', 'PROXY_SER_TRANSACTION']",
  "datetime": "31/May/2022/09/45/51",
  "procid": "-",
  "container_path": "Mycontainer17",
  "protocol": 1,
  "@timestamp": "2022-05-31T09:45:51.317Z",
  "bytes_per_second": 14349.220898258478,
  "request_method": "PUT",
  "port": 40082,
  "sysloghost": "m2-r1z1s1",
  "severity": "info",
  "account_path": "AUTH_test",
  "transaction_id": "tx0f782efac508406091fbd-006295e3cf",
  "programname": "proxy-server",
  "user_agent": "python-requests/2.18.4",
  "message": "172.29.0.1 172.29.0.1 31/May/2022/09/45/51 PUT
/v1/AUTH_test/Mycontainer17/folder8/file173.txt HTTP/1.0 201 - python-
۱۶۵۳۹۹۰۳۵۱,۰۹۸۶۸۴۳۱۱ ۱۶۵۳۹۹۰۳۵۱,۳۱۶۹۰۷۸۸۳۰۰,
"
:
"
۲
۰
۲
۲
-
۰
۵
-
۳
```

```

"source": "-",
"request_start_time": "2022-05-31T09:45:51.098Z",
"ver": "v1",
"remote_addr": "172.29.0.1",
"auth_token": "AUTH_tk56362d58f...",
"client_ip": "172.29.0.1",
"referer": "-",
"client_etag": "-",
"policy_index": 0,
"host": "r1z1s1",
"type": "rsyslog",
"status_int": 201,
"bytes_sent_format": 0,
"headers": "-",
"request_time": 0.2182,
"bytes_sum": 3131,
"device_name": "",
"request_path": "",
"partition_number": "",
"server_pid": ""
},
{
  "": 1,
  "log_info": "",
  "@version": 1,
  "facility": "local0",
  "bytes_sent": "",
  "bytes_recvd": "",
  "request_end_time": "",
  "bytes_recvd_format": "",
  "object_path": "folder8/file173.txt",
  "tags": "['swift', 'OBJ_SER_TRANSACTION']",
  "datetime": "31/May/2022:09:45:51",
  "procid": "-",
  "container_path": "Mycontainer17",
  "protocol": "",
  "@timestamp": "2022-05-31T09:45:51.282Z",
  "bytes_per_second": "",
  "request_method": "PUT",
  "port": 41688,
  "sysloghost": "m6-r1z1s1",
  "severity": "info",
  "account_path": "AUTH_test",
  "transaction_id": "tx0f782efac508406091fbd-006295e3cf",
  "programname": "object-server",
  "user_agent": "proxy-server 264",

```

```
"message": "172.29.0.2 - - [31/May/2022:09:45:51 +0000] \"PUT
/sdc6/881/AUTH_test/Mycontainer17/folder8/file173.txt\" 201 - \"PUT
\\\"tx0f782efac508406091fbd-006295e3cf\\\" \\\"proxy-server 264\\\" 0.0838 \\\"-\\\" 145
.0,
```

```
"source": "",
"request_start_time": "",
"ver": "",
"remote_addr": "172.29.0.2",
"auth_token": "",
"client_ip": "",
"referer": "PUT
http://127.0.0.1:8082/v1/AUTH_test/Mycontainer17/folder8/file173.txt",
"client_etag": "",
"policy_index": 0,
"host": "r1z1s1",
"type": "rsyslog",
"status_int": 201,
"bytes_sent_format": "",
"headers": "",
"request_time": 0.0838,
"bytes_sum": "",
"device_name": "sdc6",
"request_path": "/sdc6/881/AUTH_test/Mycontainer17/folder8/file173.txt",
"partition_number": 881,
"server_pid": 145
},
{
  "": 2,
  "log_info": "",
  "@version": 1,
  "facility": "local0",
  "bytes_sent": "",
  "bytes_recvd": "",
  "request_end_time": "",
  "bytes_recvd_format": "",
  "object_path": "",
  "tags": "['swift', 'CONT_SER_TRANSACTION']",
  "datetime": "31/May/2022:09:45:51",
  "procid": "-",
  "container_path": "Mycontainer17",
  "protocol": "",
  "@timestamp": "2022-05-31T09:45:51.129Z",
  "bytes_per_second": "",
  "request_method": "HEAD",
  "port": 40452,
```



```

"sysloghost": "m8-r1z1s1",
"severity": "info",
"account_path": "AUTH_test",
"transaction_id": "tx0f782efac508406091fbd-006295e3cf",
"programname": "container-server",
"user_agent": "proxy-server 264",
"message": "172.29.0.2 - - [31/May/2022:09:45:51 +0000] \"HEAD
/
s
d
c
^
C:..
├──.idea
│   └──inspectionProfiles
├──anomalies
├──model
│   ├──flow_model
│   └──graph_model
└──__pycache__

C:\Users\mj\Desktop\sol_project>tree /A
Folder PATH listing
Volume serial number is E00E-5E81
C:..
+---.idea
|   \---inspectionProfiles
+---anomalies
+---model
|   +---flow_model
|   \---graph_model
\---__pycache__

C:\Users\mj\Desktop\sol_project>tree /F
Folder PATH listing
Volume serial number is E00E-5E81
C:..
├──LogDB_main.json
├──logfunction.py
├──logs_status.csv
├──neo4jhandler.py
├──nodes.csv
├──previous_test_run.txt
├──run_neo4j.txt
├──sol.json
├──test_run.py
└──train_run.py

```

۱-۴- الگوریتم کشف ناهنجاری

ساختار و معماری فولدرهای پروژه به صورت زیر است:

الگوریتم ناهنجاری پیشنهادی، موارد زیر را به عنوان لاگ ناهنجار در نظر می گیرد:

- توالی لاگ های یک تراکنش از نود استارت شروع نشود و یا در نود پایانی خاتمه نیابد.
- دو لاگ متوالی از یک تراکنش، فاصله زمانی بیشتر از ۵ ثانیه داشته باشند.

لاگ آتی در توالی تراکنش متناظر با نود آتی در فلو گراف نباشد. قطعه کد زیر در ابتدا آرایه های connections، start و node را load می کند و سپس با پیروی از قوانین فوق لاگ های ناهنجار را تشخیص می دهد.

```

import pandas as pd

import os

```

```

import glob

from logfunction import LogsFunction

import numpy as np

import json

def run(fileName):

    """ check and detect anomalies

        :param filename: name of the json file

        :return: an excel file with (logstatus.csv), in the excel
        file anomalies are determined with \.

    """

    # load model

    connections = pd.read_csv("./model/graph_model/" +
                              "connections.csv")

    nodes = pd.read_csv("./model/graph_model/" + "nodes.csv")

    start = pd.read_csv("./model/graph_model/" + "start.csv")

    extension = fileName.split(".")[1]

    if extension == 'json':

        f = open(fileName)

        json_read = json.load(f)

        print("start processing ...")

        files = glob.glob('./anomalies/*')

        for f in files:

            print(f)

```

```

os.remove(f)

new_json = []

for i in range(len(json_read)):
    if json_read[i]["user_agent"] != "Swift":
        new_json.append(json_read[i])

# make standard logs (for example p-get-a)

pure_logs, transaction_ids, datetime =
LogsFunction().purifylogs(new_json)

# extract flows

logs_flows, datetime, tx =
LogsFunction().extractFlows(pure_logs, transaction_ids, datetime)

# sort logs

sorted_log_flows = LogsFunction().sortlogs(logs_flows,
datetime)

# detect anomalies

count = np.zeros(len(sorted_log_flows), dtype=object)

temp_tx = []

for i in range(len(sorted_log_flows)):
    count[i] = .

    priority = []

    for u in range(len(sorted_log_flows[i])):
        for j in range(len(nodes)):
            if sorted_log_flows[i][u] == nodes["."] [j]:
                priority.append(j)

```

```

        sorted_priority = []

        for b in range(len(priority)):
            sorted_priority.append(priority[len(priority) - \
- b])

        if start["."][sorted_priority[.]] == .:
            count[i] = \
            temp_tx.append(tx[i][.])

            continue

        counter = np.zeros(len(sorted_priority) - \)

        for k in range(len(sorted_priority) - \):
            for m in
range(len(connections[f"{priority[k]}"])):
                if str(connections[f"{m}"][priority[k]]) ==
'nan':

                    continue

                if int(connections[f"{m}"][priority[k]]) ==
int(priority[k + \]):

                    if counter[k] != \:

                        counter[k] = counter[k] + \

            if (sum(counter) != len(sorted_priority) - \) or
(sum(counter) == .):

                count[i] = \

                temp_tx.append(tx[i][.])

                continue

        else:

```

```

        count[i] = .

        temp_tx.append(tx[i][.])

        continue

    temp_tx = np.array(temp_tx)

    logs_status = np.concatenate((count.reshape(-1, 1),
temp_tx.reshape(-1, 1)), axis=1)

    df = pd.DataFrame(logs_status)

    # save logs status in logs_status.csv, 1 for anomalies

    filename = 'logs_status.csv'

    df.to_csv(filename)


    # save anomalies in anomalies folder

    for i in range(len(tx)):

        exact_logs = []

        if count[i]:

            for j in range(len(json_read)):

                if json_read[j]["transaction_id"] ==
tx[i][.]:

                    exact_logs.append(json_read[j])

            df = pd.DataFrame(exact_logs)

            filename = "./anomalies/" + str(i) +
"_____ " + str(tx[i][.]) + '.csv'

            df.to_csv(filename)


    print("processing finished")


print('Enter the filename:')

```

```

filename = input()

try:

    extension = filename.split(".")[1]

    if extension != "json":

        print("not a json file")

        exit(0)

    run(filename)

except:

    print(f"There is an error, please check the following
items:\n")

    print("\ - The file must be exist in this directory\n")

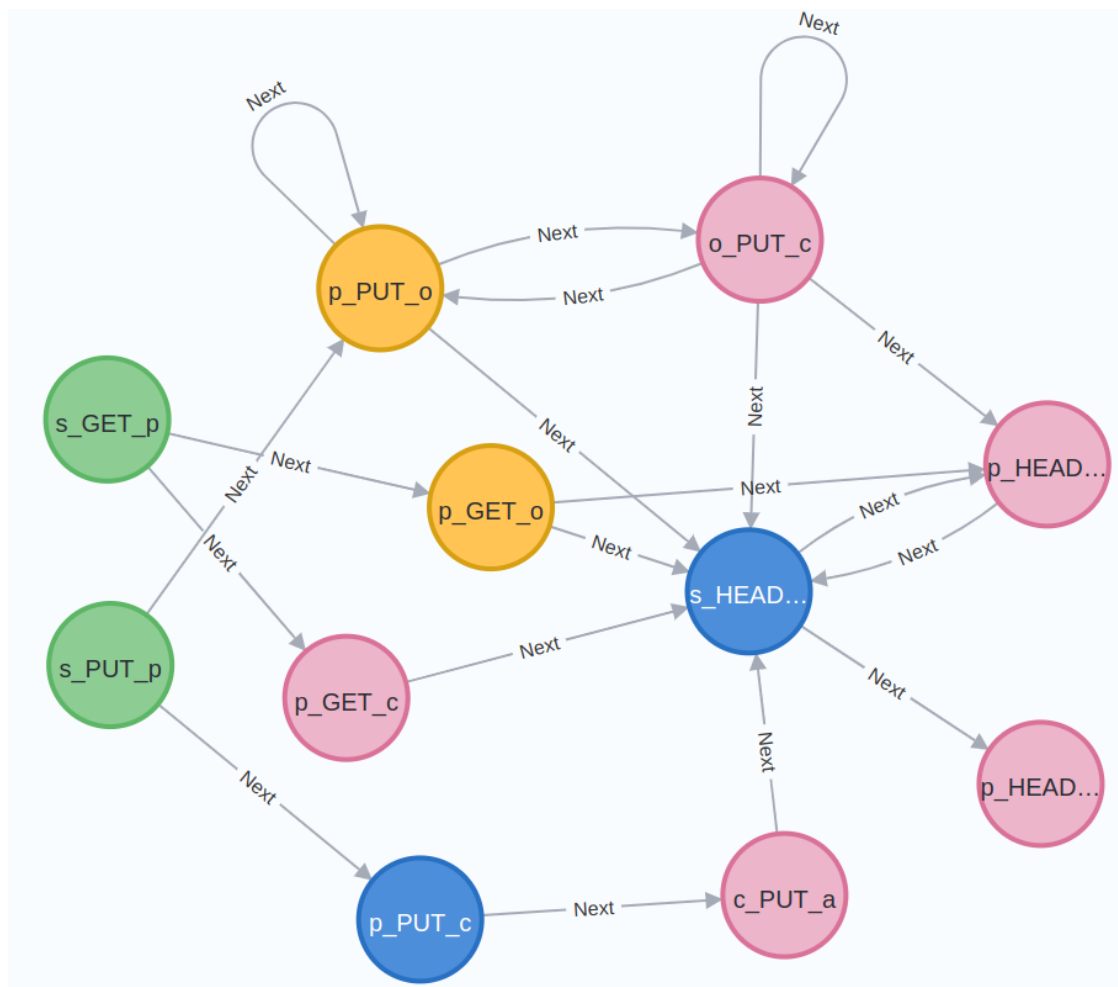
    print("\ - The input file must be a json file. please check
the extension of the file and make sure that the data "

        "structure of file is json.")

```

۲-۱-۴- بررسی چند نمونه لاگ ناهنجار

شکل نهایی گراف لاگ که ملاک پردازش و کشف ناهنجاری قرار میگیرد در شکل ۴۲ آمده است:



شکل ۴۰ : گراف نهایی

نمونه لاگ مرتب شده و اصلاح شده طبق شکل ۴۲ است.

```

array = (NdArrayItemsContainer) <pydevd_plugins.extensions.types.pydevd_plugin_numpy_types.NdArrayItemsContainer object at 0x000001A6E9309D08>
01 0 = (str; 0) s-HEAD-p
01 1 = (str; 0) s-HEAD-p
01 2 = (str; 0) p-HEAD-c
01 3 = (str; 0) s-HEAD-p
01 4 = (str; 0) p-PUT-o
01 5 = (str; 0) p-PUT-o
01 6 = (str; 0) p-PUT-o
01 7 = (str; 0) s-PUT-p
  
```

شکل ۴۱ : نمونه ناهنجاری

این مورد ناهنجاری اعلام شده است چون s-Head-p اتصال مجدد به خودش ندارد. نمونه دیگری هم از لاگ مرتب شده طبق شده ۴۳ است.

```

array = (NdArrayItemsContainer) <pydevd_plugins.extensions.types.pydevd_plugin_numpy_types.NdArrayItemsContainer object at 0x000001A6E9309D08>
01 0 = (str: ()) s-HEAD-p
01 1 = (str: ()) s-HEAD-p
01 2 = (str: ()) p-HEAD-c
01 3 = (str: ()) s-HEAD-p
01 4 = (str: ()) p-PUT-o
01 5 = (str: ()) p-PUT-o
01 6 = (str: ()) p-PUT-o
01 7 = (str: ()) s-PUT-p

```

شکل ۴۲: نمونه ناهنجاری

این مورد ناهنجاری اعلام شده است چون s-Head-p اتصال مجدد به خودش ندارد.

فصل 5- فصل پنجم: مرور کلی و کارهای آینده

۱-۵- مقدمه

در این قسمت یک مرور کلی بر روی روش‌های مطالعه شده بر روی شناسایی گراف سلامت سیستم و کشف ناهنجاری در این سیستم‌ها می‌پردازیم. این قسمت از این نظر قابل اهمیت است که زمینه را برای کارهای آینده فراهم می‌کند.

۲-۵- روش‌های مطالعه شده در استخراج گراف سلامت سیستم

در یک سیستم ابری، همواره چالش سنجش عملکرد صحیح سیستم وجود داشته است. بررسی این مساله در نوع خود بسیار سخت و دشوار است. ابزارهای مختلفی برای سنجش عملکرد سیستم در یک سیستم ابری وجود دارد که یکی از ابزارهای معروف grafana است. ابزار دیگری که این کار را انجام می‌دهد portainer است. در اینجا به طور خلاصه این دو ابزار را معرفی می‌کنیم.

۱-۲-۵- ابزار grafana

Grafana یک راهکار متن باز برای اجرای تحلیل و بررسی داده‌ها و رسم معیارهای منطقی روی انبوه داده‌ها است. همچنین از این راهکار به منظور مانیتور کردن اپلیکیشن‌ها نیز استفاده می‌شود که دارای داشبوردهای قابل شخصی سازی برای کمک به شما است. این ابزار به تمامی منابع داده‌ای ممکن متصل می‌شود و به طور معمول به دیتابیس‌هایی مانند Graphite، Prometheus، Influx DB، ElasticSearch، MySQL، PostgreSQL و ... قابل ارجاع است. متن باز بودن گرافانا به ما این امکان را می‌دهد که برای ارتباط و همکاری با منابع داده مختلف بتوانیم پلاگین نویسی کنیم.

در سازمان‌ها طبیعتاً از Grafana برای اهداف نظارتی استفاده می‌کنند و در محیط پیش تولید برای بررسی خطاهای ظاهر شده و زمان آپ تایم سرورها، در محیط‌های توسعه که به عنوان مثال از داکر کانتینرها استفاده می‌کنند ممکن است نمونه‌ها به دلایل و مشکلاتی از کار بیافتند که باعث از کار افتادن کلی سیستم شود. در این سناریوها از طریق داشبورد گرافانا می‌توانیم این مشکلات را به راحتی ردیابی کنیم که کار ما را بسیار راحت تر خواهد کرد.

۲-۲-۵- ابزار portainer

این ابزار به خوبی یک GUI کامل در اختیار ما قرار می‌دهد. اما با این تفاوت که دیگر نیاز نیست حتماً بر روی همان کامپیوتر سرویس دهنده داکر نصب شود. بلکه می‌تواند به سرویس دهنده‌های مختلف داکر متصل شود. این ابزار این امکان را دارد که به چند تا سرویس دهنده داکر متصل شود. با استفاده از این ابزار می‌توان به خوبی سرویس‌های بزرگ را پیاده سازی و از آنها استفاده کرد. در ضمن Portainer بر روی وب در دسترس می‌باشد از این رو می‌توان آن را بر روی سرورهایی که دارای UI و Desktop نیستند به خوبی کاربرد دارد.

۳-۲-۵- سایر روش‌های مانیتورینگ

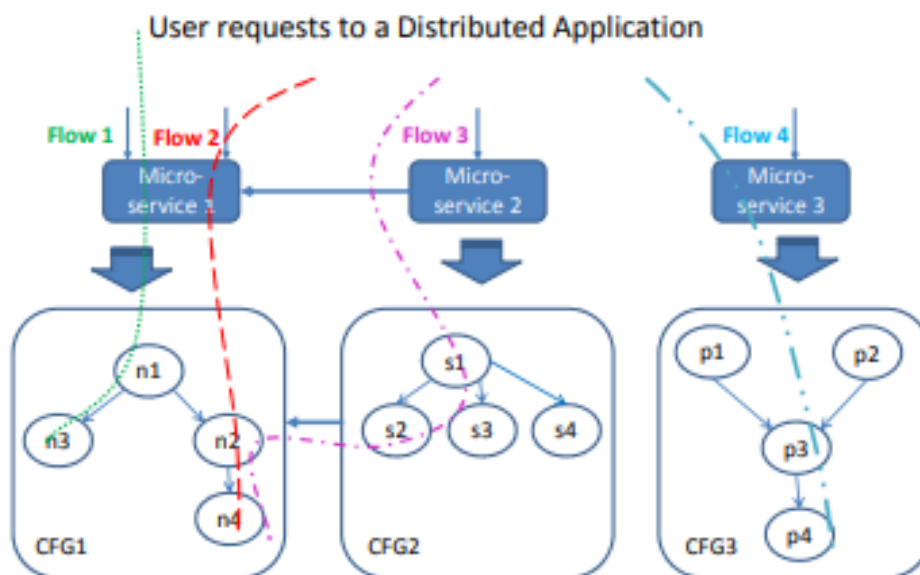
بعضی از مطالعات همواره بدنبال کشف رفتارسیستم و بررسی ناهنجاری از طریق یادگیری ماشین و با استفاده از دیتای جمع شده در سرور ابری هستند. یکی از ابزار مهم در این زمینه لاگ‌ها هستند. بسیاری از روش‌ها این کار را در دو حالت انجام می‌دهند: ۱- رسم گراف سلامت سیستم، ۲- استخراج ناهنجاری براساس گراف سلامت سیستم. ناهنجاری در حقیقت هر آنچه که مسیری در گراف پیدا نکند، تعریف می‌شود. [۱] و [۲] از این روش برای تشخیص ناهنجاری بهره می‌برند.

استخراج گراف سلامت سیستم را می‌توان مشابه مساله network inference problem دانست. [۳] و [۴] برپایه این تعریف به حل این مساله پرداختند.

با توجه به پیچیده بودن این مساله، در بعضی از مطالعه‌ها فرض‌های ساده‌تر شدن این مساله انجام شده است. برای مثال در [۵] و [۶] فرض شده که نودهای گراف از قبل مشخص هستند و مشارکت اصلی این مطالعه‌ها در تشخیص یال‌های بین این نودهاست.

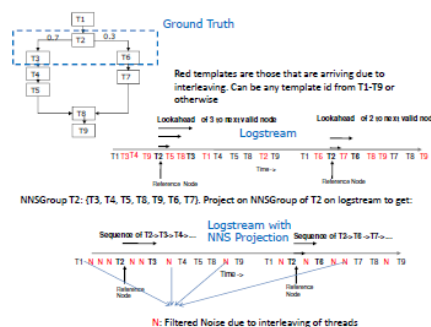
در بعضی مطالعه‌ها فرض شده است که قسمت متغیر یک لاگ در ابتدای لاگ است و طول این قسمت هم ناچیز است. مراجع [۷] و [۸] از این فرض استفاده کردند.

در دسته دیگری از مطالعه‌ها فرض شده است که لاگ‌های ما دارای transaction id هستند. این امر در شناسایی یک فرایند بسیار کمک کننده است. این مورد در مراجع [۹] و [۱۰] وجود دارد. به طور کلی یک سیستم توزیع شده را می‌توان به صورت زیر در نظر گرفت:



شکل ۴۳: سیستم توزیع شده

در مقاله [۱۰] که یک مطالعه بزرگ در گروه تحقیقات شرکت IBM است، فرض شده که هیچ گونه دانش اولیه ای نسبت به لاگ ها وجود ندارد. این رویکرد تشخیص گراف سلامت سیستم را بسیار دشوار می کند. خلاصه این روش به شرح زیر است : هر لاگ در سیستم نرم افزاری به دو دسته تقسیم می شود: قسمت تغییرپذیر و قسمت ثابت. قسمت متغیر شامل موارد مانند آدرس پورت و زمان انجام لاگ است. جهت آن که به یک گراف کنترل جریان برسیم باید برای هر لاگ تعدادی پدر و فرزند تشکیل شود. برای تشکیل پدر و فرزند برای هر تمپلیت باید توالی زمانی لاگ ها در نظر گرفته شود. به عنوان مثال فرض کنید که بخواهیم برای یک تمپلیت پدر و فرزند استخراج کنیم. در شکل ۴۴ این فرآیند نمایش داده شده است. در این شکل، از یک روش آماری برای استخراج پدر و فرزندهای T^2 استفاده شده است. برای آن که اساس این روش آماری توضیح داده شود ابتدا توالی لاگ های نمایش داده شده در شکل را در نظر بگیرید. تعدادی از این لاگ ها نویز هستند. با یک میانگین گیری ساده می توان متوجه شد که اکثر مواقع بعد از تمپلیت T^1 ، تمپلیت T^2 به وقوع می پیوندد. موارد نادر دیگری هم هستند که به عنوان نویز در نظر گرفته می شوند. با همین روش متوجه می شویم که بعد از T^2 در ۷۰ درصد موارد T^3 و در ۳۰ درصد موارد T^6 به وقوع می پیوندد. موارد نادر دیگری هم هستند که به عنوان نویز در نظر گرفته می شوند. به همین سادگی پدر و فرزند تمپلیت T^2 استخراج می شود. اما سوالی که به وجود می آید این است: با چه حد آستانه ای موارد نویزی را تشخیص دهیم؟ برای پاسخ به این سوال از یک قاعده سرانگشتی استفاده می کنیم و آن هم این نکته است: اگر درصد حضور زیر ۱۰ درصد باشد، آن مورد به عنوان نویز در نظر گرفته شود.



شکل ۴۴ : تشکیل پدر و فرزند برای هر تمپلیت

پس از این مرحله گراف های مربوط به هر مرحله با هم مرج می شوند. اگر شباهت بین دو تمپلیت از نظر متنی بالای یک حد آستانه باشد دو گراف با همدیگر مرج می شوند و بین پدرها و فرزندان اجتماع گرفته می شود. اما اگر این شباهت فقط بین فرزندان و یا پدرها باشد، دو گراف از محل آن شباهت بایکدیگر مرج می شوند این مرج شدن به همین

صورت بین هر دو گراف تکرار می شود تا در نهایت به یک گراف کلی برسیم. گراف به دست آمده همان گراف کنترل جریان برای لاگ ها است. معیار شباهت هم جاکارد است. در یک سیستم نرم افزاری پس از تشکیل گراف کنترل جریان و گراف هر تمپلیت، دو حالت می تواند نشانه ناهنجاری باشد:

الف) در یک بازه زمانی مشخص، فرزندان یک تمپلیت مشاهده نشوند. اگر همه فرزندان مشاهده نشوند احتمال ناهنجاری بیشتر از حالتی است که یک یا تعدادی از فرزندان مشاهده نشوند.

ب) توزیع فرزندان یا پدران هر تمپلیت در یک بازه زمانی تغییر کند.

در [۱۱]، گراف حالت با مدل متفاوتی ساخته می شود و روی لاگ های هدوپ این پروسه تست می شود. برای ساخت گراف حالت از لاگ های خام استفاده می شود. در ابتدای کار نیاز است که هر لاگ خام آنالیز شود و دو قسمت متغیر و اصلی از آن استخراج شود. پس از این مرحله، ساخت گراف حالت آغاز می شود. هر گراف حالت سه قسمت اصلی دارد: موجودیت، حالت، رخداد. یال ها به دو دسته فضایی و زمانی تقسیم می شوند. هر یال فضایی برای ایجاد یک رابطه بین یک موجودیت با یک رخداد و یا یک موجودیت با یک حالت استفاده می شود. هیچ گاه، یک رخداد با یک رخداد رابطه مستقیم ندارد و باید یک حالت و یا یک رخداد بین دو رخداد واسطه شود.

الف) حالت: این موارد در سیستم های نرم افزاری موارد مشخصی هستند و شامل مواردی مانند error, warning, run می شوند.

خلاصه ای از رویکردهای مختلف در مساله پردازش گراف در جدول زیر جمع اوری شده است:

مرجع	رویکرد
[۱] و [۲]	روش هیبرید در تشخیص گراف سلامت سیستم
[۳] و [۴]	حل مساله با دید network inference problem
[۵] و [۶] و [۷] [۸] ، [۹]	فرض های ساده کننده در حل مساله
[۱۰]	حل مساله بدون فرض ساده کننده

شماره مقاله	تکنیک	دیتاست	متودارزیایی
مقاله شماره ۱	رسم گراف + کشف ناهنجاری با در نظر گرفتن فرض وجود ترنزشن آیدی بر روی گراف های با فرمت مشخص و ساده سازی شده	لاگ های هدوپ	FALSE POSITIVE
مقاله شماره ۲	رسم گراف + کشف ناهنجاری با در نظر گرفتن فرض کوتاه بودن طول قسمت متغیر لاگ	لاگ های هدوپ	FALSE POSITIVE
مقاله شماره ۳	کشف ناهنجاری در لاگ ها با استفاده از تکنیک های بیژین در network inference problem	SILK + هدوپ	FALSE POSITIVE
مقاله شماره ۴	کشف ناهنجاری در لاگ ها با استفاده از فرض بولین در network inference problem	SILK + اسپارک	FALSE POSITIVE
مقاله شماره ۵	کشف ناهنجاری در لاگ ها با استفاده از فرض مشخص بودن نود ها از قبل و فرض وجود ترنزشکن آیدی و استخراج کانکشن ها با متود ساخت گراف کامل از زیرگراف ها	لاگ های ویندوز + لاگ های APACHE ERROR	تعداد ناهنجاری درست به نسبت کل
مقاله شماره ۶	بهبود مثال شماره پنج در کلاسترینگ گروه لاگ های مشابه با تغییر متود شباهت از فازی به جاکارد	لاگ های هدوپ	تعداد ناهنجاری درست به نسبت کل
مقاله شماره ۷	استخراج ویژگی + PCA جهت کشف ناهنجاری	لاگ های هدوپ	FALSE POSITIVE
مقاله شماره ۸	کلاستر کردن لاگ ها براساس معیار EDIT DISTANCE BASED به این صورت که شباهت هر دو لاگ اگر بالای ترشلد مشخصی باشد در یک گروه قرار می گیرند	لاگ های هدوپ	تعداد ناهنجاری درست به نسبت کل
مقاله شماره ۹	استفاده از متود PARTIATION SEARCH BIJECTION BY در استخراج کلاسترها	لاگ های هدوپ	تعداد ناهنجاری درست به نسبت کل
مقاله شماره ۱۰	استخراج گراف سلامت + ناهنجاری بدون هیچ گونه فرض اولیه با استفاده از متود پدر و فرزند در استخراج توالی و کلاسترینگ با معیار جاکارد و کشف ناهنجاری با پیشمایش آرایه ای کانکشن های گراف	لاگ های هدوپ	تعداد ناهنجاری درست به نسبت کل

در هر کدام از این مقاله ها، اعدادی به عنوان دقت این الگوریتم ها براساس معیار استفاده شده در خود مقاله ذکر شده است. اما نکته قابل تامل این است که این مقاله ها نتایج خود را با هم مقایسه نکرده اند.

فصل 6- فصل ششم: مراجع

- [1] J.-G. Lou, Q. Fu, Y. Wang, and J. Li. Mining dependency in distributed systems through unstructured logs analysis. In SIGOPS Operation Systems Review, 2010.
- [2] J.-G. Lou, Q. Fu, Y. Wang, and J. Li. Mining dependency in distributed systems through unstructured logs analysis. In SIGOPS Operation Systems Review, 2010.
- [3] B. Abrahao, F. Chierichetti, R. Kleinberg, and A. Panconesi. Trace complexity of network inference. In KDD, 2013.
- [4] M. Gomez-Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. In KDD, 2010.
- [5] W. V. der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. In TKDE, 2004.
- [6] C. W. Gunther and W. M. van der Aalst. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In BPM, 2007.
- [7] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. Detecting large scale system problems by mining console logs. In ICML, 2010.
- [8] Q. Fu, J.-G. Lou, Y. Wang, and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis. In ICDM, 2009.
- [9] A. Makanju, A. Z. Heywood, and E. E. Milios. Clustering event logs using iterative partitioning. In KDD, 2009.
- [10] J.-G. Lou, Q. Fu, S. Yang, J. Li, and B. Wu. Mining program workflow from interleaved traces. In KDD, 2010.