РЕФЕРАТ

Пояснительная записка к учебно-исследовательской работе содержит 18 страниц.

Ключевые слова: mapreduce, logs, распределенные вычисления, python, надежность, отказоустойчивость, функциональное программирование, регулярные выражения.

Цель работы — разработка алгоритма для выделения частовстречающихся шаблонов ошибок из файлов журналов приложений.

В процессе работы осуществлялась разработка алгоритма, реализация алгоритма на языке python. И внедрение в производство с использованием MapRedcue-системы Yandex Tables.

В результате работы был разработан алгоритм, реализован на языке python и написаны две вспомогательные утилиты на языке bash и две утилиты на языке python.

СОДЕРЖАНИЕ

НОРМАТИВНЫЕ ССЫЛКИ

В настоящей пояснительной записке к учебно-исследовательской работе использованы ссылки на следующие стандарты.

ГОСТ 7.32-2001 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления.

ГОСТ 7.9-95 Система стандартов по информации, библиотечному издательскому делу. Реферат и аннотация. Общие требования.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Определение 1 (MapReduce)**.** Модель распределённых вычислений, представленная компанией Google, используемая для параллельных вычислений над очень большими, несколько петабайт, наборами данных в компьютерных кластерах.

Определение 2 (Map)**.** map(f, list) — функция высшего порядка двух аргументов, применяет к каждому элементу списка list, функцию f(x), в качестве результата возвращает список полученных значений.

Пример. map(lambda x: x**3, [1, 2, 3]) вернёт список [1, 8, 27].

Определение 3 (Reduce)**.** reduce(f, list, init=None) — функция высшего порядка, последовательно применяет f(x, y) к элементу списка и значению от предыдущего выполнения функции.

Пример. map(lambda x, y: x + y, [1, 2, 3]) вернёт 6(сумма всех элементов). 6 = sum(sum(1, 2), 3).

ВВЕДЕНИЕ

Большинство программных систем, имеющих сложную структуру и состоящих из нескольких сотен различных компонент, обладают рядом схожих проблем. Например веб-поиск содержит следующие компоненты: балансеры, верхние, средние метапоиски, промежуточные и базовые поиски, колдунщики, антироботы, свежесть, региональные поиск, несколько десятков параллельных поисков.

Определение 4 (Экземпляр)**.** приложение, запущенное в контейнере и описываемое парой host:port.

Всего единовременно запущено несколько ******** тысяч экземпляров приложений. Каждый экземпляр генерирует множество ошибок и записывает каждую из них в файл журнала ошибок.

Некоторые приложения, близкие по функционалу, пишут в один и тот же файл. Файлы журналов ротируются согласно определённому алгоритму. Тем не менее объем файла журнала для одного экземпляра может достигать нескольких сотен мегабайт, что препятствует быстрому ручному анализу в случае инцидента и инженеры вынуждены тратить ценные секунды на просмотр сотен тысяч строк файла в поисках сообщения с описанием элемента, вызвавшего сбой работы системы.

Начальным требованием к системе для эффективного использования алгоритма является наличие сопоставимого с количеством поисковых экземпляров приложений количества узлов на которых может быть запущена программа, реализующая алгоритм.

В организации, в которой выполнялась учебно-исследовательская работа, развёрнута большая поисковая инфраструктура, которая не лишена недостатков и существует вероятность поломки некоторой её части. Существует множество средств мониторинга состояния веб-поиска и противодействия инцидентам, но в некоторых случаях инженерам их недостаточно и приходится вручную анализировать файлы журналов отдельных экземпляров приложений на отдельных серверах, что, в свою очередь, замедляет скорость реакции на непредвиденную ситуацию. Но даже автоматизация процесса анализа файла журнала одного экземпляра не решает проблему полностью, поэтому необходима возможность быстрого анализа файлов журналов сразу множества экземпляров.

Определение 5 (Шаблон)**.** Специально подготовленное регулярное выражение с экранированными спецсимволами.

Таким образом, целью этой учебно-исследовательской работы является разработка алгоритма, позволяющего собирать статистику по ошибкам, встречающимся в файлах журналов экземпляров поисковых приложений на основе существующих шаблонов и выделять новые шаблоны.

1 СОВРЕМЕННОЕ СОСТОЯНИЕ ПРОБЛЕМЫ АНАЛИЗА ФАЙЛОВ ЖУРНАЛОВ

1.1 Актуальность задачи

1.2 Сравнение с существующими аналогами

1.3 Постановка задачи

2 ПРОЕКТИРОВАНИЕ АЛГОРИТМА

2.1 Структура

2.2 Архитектура

2.3 Планируемая выгода

3 РЕАЛИЗАЦИЯ

3.1 Сравнение возможных решений

3.2 Особенности реализации, проблемы и способы их решения

про collet-logs.py и skynet

3.3 Алгоритм работы программы

4 ТЕСТИРОВАНИЕ И АНАЛИЗ РЕЗУЛЬТАТОВ

4.1 Методы тестирования

4.2 Анализ результатов тестирования

4.3 Дальнейшее развитие

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

```
#!/usr/bin/env python
# encoding: utf-8

import sys
import os
import re
import argparse
import fileinput
import copy
import itertools

from argparse import RawDescriptionHelpFormatter

from patterns import main_patterns


BUFFER_SIZE = 30

class Buffer(object):
    """Class for buffering input"""

    def __init__(self, output_filename=os.devnull):
        self.buf = [''] * BUFFER_SIZE
        self.line_scrolled = 0
        self.matched_lines = []
        self.unmatched_stream = open(output_filename, 'w')
        self.updated = True
        self.last_text = ''

    def add(self, line):
        self.buf.append(line)
        if 0 not in self.matched_lines:
            self.unmatched_stream.write(line)

        self.buf = self.buf[1:BUFFER_SIZE+1]
        self.matched_lines = filter(lambda x: x >= 0,
                                    map(lambda x: x - 1, self.matched_lines))
        self.line_scrolled += 1
        self.updated = True

    def text(self):
        """lazy function, returning joined lines
        :returns: str()

        """
        if self.updated:
            self.last_text = ''.join(self.buf)
            self.updated = False

        return self.last_text
```

```
    def try_to_match(self, pattern):
        matched = pattern.search(self.text())
        if matched:
            self.mark_matched(pattern)
        return matched

    def mark_matched(self, pattern):
        lines_count = len(pattern.split('\n'))
        marked_lines = range(BUFFER_SIZE - lines_count, BUFFER_SIZE)
        self.matched_lines += marked_lines

    def __enter__(self):
        return self

    def __exit__(self, type, value, traceback):
        for _ in xrange(BUFFER_SIZE):
            self.add('')
        self.unmatched_stream.close()


class SnippetsQueue(object):
    def __init__(self):
        self.new_snippets = []
        self.ready_snippets = {}
        self.pattern_used = {}

    def push(self, snippet):
        if not self.pattern_used.get(pattern, False):
            self.new_snippets.append(snippet)
        self.pattern_used[pattern] = True

    def add(self, line):
        for snippet in self.new_snippets:
            if snippet.full():
                self.make_ready(snippet)
            else:
                snippet.add(line)

    def make_ready(self, snippet):
        self.new_snippets.remove(snippet)
        pattern = snippet.pattern
        try:
            self.ready_snippets[pattern].append(snippet)
        except KeyError:
            self.ready_snippets[pattern] = [snippet]
        self.pattern_used[pattern] = False
        if len(self.ready_snippets[pattern]) == SNIPPETS_TO_SHOW:
            self.pattern_used[pattern] = True

    def make_all_ready(self):
        for snippet in self.new_snippets:
```

```
            self.make_ready(snippet)


class StatCollector(object):

    """Class for collecting statistics"""

    def __init__(self):
        self.match_count = {}
        self.lines_count = 0
        self.number_of_matched_lines = 0
        pass

    def add(self, pattern):
        """Add pattern matching event"""
        if pattern not in self.match_count:
            self.match_count[pattern] = 0
        self.match_count[pattern] += 1
        self.number_of_matched_lines += len(pattern.split('\n'))


def print_stat(stat_collector, snippets_queue=None, **kwargs):

    """Print statistics and snippets"""

    for pattern, count in stat_collector.match_count.iteritems():
        print """\
********************************************************************************
pattern: "{}"
--------------------------------------------------------------------------------
number of matches: {}
********************************************************************************\
""".format(pattern, count)
        if snippets_queue:
            if pattern in snippets_queue.ready_snippets:
                for snippet in snippets_queue.ready_snippets[pattern]:
                    snippet.show()
                    output_stream.write('-' * 80 + '\n')
            else:
                output_stream.write('|No snippets found : (\n')
                output_stream.write('-' * 80 + '\n')

    print 'patterns used: {}/{}'.format(
        len(stat_collector.match_count),
        len(kwargs['patterns']))
    print 'number of lines matched: {}/{}'.format(
        stat_collector.number_of_matched_lines,
        stat_collector.lines_count)



def make_escaped(string):
```

```
    """Make escaped pattern from string

    :string: string to be escaped
    :returns: pattern

    """

    return re.escape(string.replace('\n', 'NSLPH')).replace('NSLPH', '\n') + r'\n\Z'


def get_stat(input_files, snippets_count=0, context=3,
             patterns=main_patterns,
             output_stream=sys.stdout, snippets_file=None,
             unmatched_filename=os.devnull):
    """Show statistics for patterns

    :input_stream: input stream
    :snippets_count: maximum number of snippets to show for every pattern
    :context: number of lines in snippet around last line of pattern match
    :patterns: patterns that need to look
    :output_stream: stream for output
    :returns: None

    """

    input_stream = fileinput.input(input_files)

    LINES_ABOVE = context
    LINES_BELOW = context
    SNIPPETS_TO_SHOW = snippets_count
    SHOW_SNIPPETS = snippets_count > 0

    class Snippet(object):
        def __init__(self, lines_above, pattern, line_number):
            self.text = copy.copy(lines_above)
            self.pattern = pattern
            self.line_number = line_number

        def full(self):
            return len(self.text) >= LINES_ABOVE + LINES_BELOW + 1

        def add(self, line):
            self.text.append(line)

        def show(self):
            i = 0
            for line in self.text:
                if i == LINES_ABOVE:
                    output_stream.write('| {} ==>'.format(self.line_number))
                else:
                    if re.search(self.pattern, line):
                        output_stream.write('|-->')
```

```
                else:
                    output_stream.write('|>')
            output_stream.write(line)
            i += 1


    stat_collector = StatCollector()
    snippets_queue = SnippetsQueue()
    line_number = 1

    compiled_patterns = map(re.compile, patterns)

    with Buffer(unmatched_filename) as input_buffer:
        if snippets_file:
            snippets_buffer = Buffer()
        else:
            snippets_buffer = input_buffer

        for line in input_stream:
            input_buffer.add(line)
            if snippets_file:
                line = snippets_file.readline()
                snippets_buffer.add(line)

            for pattern in compiled_patterns:
                if input_buffer.try_to_match(pattern):
                    if SHOW_SNIPPETS:
                        if LINES_ABOVE > 0:
                            snippet_begining = snippets_buffer.buf[-LINES_ABOVE:]
                        else:
                            snippet_begining = []

                        snippet = Snippet(snippet_begining, pattern.pattern,
                                          line_number)
                        snippets_queue.push(snippet)
                    stat_collector.add(pattern.pattern)

            if SHOW_SNIPPETS:
                snippets_queue.add(line)
            line_number += 1

    stat_collector.lines_count = line_number - 1

    if not SHOW_SNIPPETS:
        snippets_queue = None
    else:
        snippets_queue.make_all_ready()

    return (stat_collector, snippets_queue)


def show_patterns():
```

```
    """Show patterns from patterns.py"""

    for pattern in main_patterns:
        print pattern
        print '-' * 80


def print_escaped(files):
    input_stream = fileinput.input(files)
    text = ''
    for line in input_stream:
        text += line
    escaped_text = make_escaped(text)
    print '\n'
    print escaped_text


def main():
    parser = argparse.ArgumentParser(description=\
    """
        Parse file[s]\n\n
        examples:
        ./%(prog)s file[s] -s 2 -C 3
        cat error_log | tail -n 1000 | ./prepare.py | ./%(prog)s
        ./%(prog)s -p
        echo "some \\ntext, that wants to be pattern" | ./%(prog)s -e
    """, formatter_class=RawDescriptionHelpFormatter)
    parser.add_argument('file', nargs='*', default=[],
                        help='file[s] to be parsed')
    parser.add_argument('-s', '--snippets', nargs='?', type=int, const=5,
                        help='show maximum SNIPPETS snippets (5 default)')
    parser.add_argument('-C', '--context', nargs='?', type=int,
                        help='show CONTEXT lines around pattern last line')
    parser.add_argument('-p', '--patterns', action='store_const', const=True,
                        help='show patterns')
    parser.add_argument('-e', '--escape', action='store_const', const=True,
                        help='escape given string')
    parser.add_argument('-o', '--original', nargs=1,
                        help='provide original file for better snippets')
    parser.add_argument('-u', '--unmatched', nargs=1,
                        help='output unmatched text to file')

    args = parser.parse_args()

    if args.patterns:
        show_patterns()
        return

    if args.escape:
        print_escaped(args.file)
        return
```

```
    kwargs = {}

    kwargs['input_files'] = args.file

    if args.snippets:
        kwargs['snippets_count'] = args.snippets

    if args.context is not None:
        kwargs['context'] = args.context

    if args.original:
        kwargs['snippets_file'] = fileinput.input(args.original)

    if args.unmatched is not None:
        kwargs['unmatched_filename'] = args.unmatched[0]

    kwargs['patterns'] = main_patterns

    result = get_stat(**kwargs)
    print_stat(*result, **kwargs)


if __name__ == '__main__':
    main()
```