Министерство общего и профессионального образования
Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ
УНИВЕРСИТЕТ “МИФИ”

Факультет Кибернетики и информационной безопасности
Кафедра Информационные технологии в социальных системах

К защите допустить:
Заведующий кафедрой
_____ М. В. Сергиевский

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему:

РАЗРАБОТКА АЛГОРИТМА ДЛЯ ВЫДЕЛЕНИЯ ЧАСТОВСТРЕЧАЮЩИХСЯ ШАБЛОНОВ ОШИБОК ИЗ ФАЙЛОВ ЖУРНАЛОВ ПРИЛОЖЕНИЙ

Студент: А. Г. Тропин

Научный руководитель:
к.т.н, доцент М. В. Сергиевский

Москва 2015

РЕФЕРАТ

Пояснительная записка к учебно-исследовательской работе содержит 20 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 в двух файлах на языке python и написаны две вспомогательные утилиты на языке bash.

СОДЕРЖАНИЕ

Реферат **1**

Нормативные ссылки **3**

Определения, обозначения и сокращения **4**

Введение **5**

1 Анализ предметной области **6**
1.1 Представление задачи в терминах MapReduce 6
1.2 Выбор языка программирования и средств разработки 7
1.3 Структура данных 8
1.4 Стадии выполнения задачи 8
1.4.1 Подготовка репозитория 8
1.4.2 Выбор хранилища для шаблонов 9
1.4.3 Написание программного кода 9

2 Имплементация задачи **17**
2.1 Диаграмма компонентов 17
2.2 Описание способов запуска 17
2.3 Анализ полученных результатов 17
2.4 Дальнейшее развитие 17

Заключение **18**

Список использованных источников **19**

Приложения **20**

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 выбражение 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 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Представление задачи в терминах MapReduce

Для параллельного запуска приложений была выбрана модель распределённых вычислений MapReduce по ряду причин. Основные понятия:

Как оказалось задача без особых сложностей выражается в терминах чистых функций flat map и flat reduce, так как основная структура, используемая в алгоритме — это пара(шаблон, количество совпадений) и благодаря этому однопоточный код легко запускается на множестве узлов MapReduce-кластера. Это обстоятельство освобождает от реализации сложного механизма сетевого взаимодействия.

В качестве реализации модели MapReduce была выбрана реализация Yandex Tables, обладающая рядом отличий и нововведений:

- Колоночное хранение и range-операции
- Дерево метаинформации
- Единая операция Map-Reduce
- Расширенная поддержка транзакций, включая вложенность
- Настраиваемый коэффициент репликации и алгоритмы сжатия данных
- Хранение файлов в системе и их раздача исполняемым задачам
- Разные форматы стриминга (yamr, dsv)
- Надёжность и производительность
  - Отказоустойчивый мультимастер
  - Отказоустойчивый реплицированный планировщик
  - Минорные обновления проходят без заметного эффекта для конечных пользователей
- Гибкие слоты (заказ CPU, RAM)
- Существенные обновления не требуют пересборки C++ клиентов (так как все работает через HTTP API и стриминг клиент)

Не смотря на существенные отличия от модели, описанной в документе от компании Google, код программы, реализующий алгоритм легко переносится на другие реализации данной модели распределённых вычислений.

1.2 Выбор языка программирования и средств разработки

В качестве среды разработки было решено использовать удалённую виртуальную машину с конфигурацией и окружением идентичными конфигурации и окружению MapReduce узла. Были выбраны следующие программные продукты:

- В качестве операционной системы использовался дистрибутив GNU/Linux Ubuntu 12.04 LTS с модифицированным ядром и дополнительными пакетами.

- Vim — один из немногих настоящих текстовых редакторов, обладающих массой встроенных возможностей и практически безграничным потенциалом к расширению за счёт встроенного интерпретируемого языка программирования VimL и поддержкой возможности написания расширений на таких языках, как python, ruby, perl.

- Сохранение состояние рабочего окружения осуществлялось с помощью терминального мультиплексора tmux, имеющего клиент-серверную архитектуру и позволяющего отсоединятся от текущей сессии, оставляя её работать в фоновом режиме с последующей возможностью переподключения. tmux — свободная консольная утилита-мультиплексор, предоставляющая пользователю доступ к нескольким терминалам в рамках одного экрана. tmux может быть отключён от экрана: в этом случае он продолжит исполняться в фоновом режиме; имеется возможность вновь подключиться к tmux, находящемуся в фоне. tmux является штатным мультиплексором терминалов операционной системы OpenBSD. Программа tmux задумывалась как замена программы GNU Screen.

- Удалённое подключение осуществлялось средствами защищённого протокола ssh
(беспарольная аутентификация с использованием ключа) и технологии cauth. SSH (англ. Secure Shell — “безопасная оболочка”]) — сетевой протокол прикладного уровня, позволяющий производить удалённое управление операционной системой и туннелирование TCP-соединений (например, для передачи файлов). Схож по функциональности с протоколами Telnet и rlogin, но, в отличие от них, шифрует весь трафик, включая и передаваемые пароли. SSH допускает выбор различных алгоритмов шифрования. SSH-клиенты и SSH-серверы доступны для большинства сетевых операционных систем.

- Для управления версиями исходных кодов использовалась децентрализованная система управления версиями git, в качестве сервиса для хранения репозитория был использован сервис github.

В качестве языка программирования был выбран python2.7. Так как:

– Он предустановлен в большинстве современных дистрибутивах операционных систем.

– Выразителен. Аналогичные программы на таких языках как Java, C++ имеют в разы большие объёмы исходных кодов.

– Обладает высокой производительностью.

– Имеет множество библиотек, в том числе библиотеки для работы с регулярными выражениями, обёртки для MapReduce.

1.3 Структура данных

В качестве входных данных использовался агрегированный файл журнала с нерегулярной структурой, содержащий сообщения об ошибках в различных форматах, как многострочные, так и однострочные.

Первая часть алгоритма, позволяет осуществить сбор статистики и возвращает список пар (шаблон, количество совпадений), так же существует возможность получить часть текста не удовлетворяющую известным шаблонам, для последующего анализа с помощью второй части алгоритма, позволяющей выделить новые предполагаемые шаблоны и с помощью первой части алгоритма получить статистику, подтверждающую или опровергающую предположение.

1.4 Стадии выполнения задачи

1.4.1 Подготовка репозитория

Был создан git-репозиторий на сервисе github, сделана его локальная копия. Была выбрана следующая структура проекта и правила именования файлов:

– В корневом каталоге лежат файлы README.md, LICENSE, .gitignore.

– В каталоге doc/ хранится документация. Исходные тексты в LaTeXи скомпилированная версия в PDF.

– В каталоге direlog/ хранятся исходные коды с расширением .py и тесты, имеющие префикс test_

– В каталоге direlog/example/ хранятся примеры файлов журналов и вспомогательные скрипты на языке bash.

1.4.2 Выбор хранилища для шаблонов

В качестве хранилища для паттернов было решено использовать обычный файл на языке python, названный patterns.py и содержащий в себе два списка паттернов prepare_patterns и main_patterns, используемых на подготовительном этапе и на этапе сбора статистики соответственно, и хранить его под контролем версий в этом же репозитории. Причин на это несколько: во-первых простота модификации файла с помощью скриптов, во-вторых возможность просмотра истории изменений и откат к предыдущим версиям, в-третьих возможность ручного редактирования.

1.4.3 Написание программного кода

Для разработки приложения, реализующего алгоритм была использована одна из гибких методологий разработки - экстремальное программирование.

Использовались следующие приёмы этой методологии разработки:

- Короткий цикл обратной связи
  - Разработка через тестирование
  - Заказчик всегда рядом
  - Парное программирование
- Непрерывный, а не пакетный процесс
  - Непрерывная интеграция
  - Рефакторинг
  - Частые небольшие релизы
- Понимание, разделяемое всеми
  - Простота
  - Коллективное владение кодом
  - Стандарт кодирования
- Социальная защищённость программиста
  - 40-часовая рабочая неделя

В результате можно выделить следующие этапы развития приложения:

а) Написание функциональных тестов для prepare.py.

б) Написание утилиты для предварительной обработки исходного файла журнала. Утилита получила название prepare.py и позволила подготовить сырой файл журнала для последующей обработки, путём замены уникальных токенов, таких как UUID, timestamp, версии, номера строк, пути, содержащие версии, на строковые константы.

в) Формирование prepare_patterns на основе ручного анализа файлов журналов.

г) Написание функциональных тестов для direlog.py.

д) Реализация алгоритма для сбора статистики с использованием main_patterns. Утилита получила название direlog.py. И позволяет на выходе получить список пар (шаблон, количество совпадений) или текст, не подходящий под известные шаблон.

е) Написание функциональных тестов для direlog.py.

ж) Добавление поддержки буферизации входного потока и поддержки многострочных шаблонов.

з) Добавление функции, позволяющей запускать алгоритм на MapRedcue-кластере.

2 ИМПЛЕМЕНТАЦИЯ ЗАДАЧИ

2.1 Диаграмма компонентов

2.2 Описание способов запуска

2.3 Анализ полученных результатов

2.4 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()
```