ISSN 2220-3397

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

ПрИн
2021
5
Том 12

Институт математики им. С.Л. Соболева

Новосибирский государственный университет
*НАСТОЯЩАЯ НАУКА

Российская Инженерная Академия
Институт Математики им. С. Л. Соболева СО РАН
Российская Ассоциация Искусственного Интеллекта
Новосибирский национальный исследовательский государственный университет
Institute of Electrical and Electronics Engineers (IEEE)

Знания Онтологии Теории

VIII Международная конференция

ЗНАНИЯ – ОНТОЛОГИИ – ТЕОРИИ

8 – 12 ноября 2021 г., Новосибирск

Целью конференции является ознакомление с новейшими научными достижениями, обмен знаниями и передовым опытом в области математических методов представления и анализа данных, извлечения знаний и построения теорий предметных областей, анализа формальных понятий и извлечения информации из текстов естественного языка. Сборник трудов конференции будет проиндексирован в РИНЦ, избранные статьи будут проиндексированы в Scopus.

Тематика конференции отражает основные стадии процесса познания:

- Обнаружение закономерностей и извлечение знаний, скрытых в структурированных и неструктурированных данных. Машинное обучение. Распознавание образов, анализ данных. Прогнозирование. Индуктивный вывод.
- Систематизация знаний. Инженерия знаний. Управление знаниями. Извлечение знаний из текстов на естественном языке. Разработка онтологий предметных областей, технологии создания и применения онтологий.
- Построение теорий предметных областей. Разработка семантических и онтологических моделей предметных областей. Анализ формальных понятий. Логическая семантика естественного языка. Нечёткие логики.

Работа конференции планируется в виде пленарных, секционных и стендовых докладов и круглых столов по тематике конференции. Рабочие языки конференции – русский и английский.

Подробности: http://www.math.nsc.ru/conference/zont/21/

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индекс по Объединенному каталогу "Пресса России" — **22765**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования и базу данных RSCI на платформе Web of Science.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 12
N 5
2021

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.4`233 DOI: 10.17587/prin.12.227-232

В. А. Галатенко, д-р физ.-мат. наук, зав. сектором, galat@niisi.ras.ru,
К. А. Костюхин, канд. физ.-мат. наук, ст. науч. сотр., kost@niisi.ras.ru, Федеральное государственное учреждение "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук", Москва

Посмертный анализ и его особенности при разработке приложений в среде ОСРВ Багет

Несмотря на все усилия программистов по созданию высококачественного программного обеспечения, некоторые ошибки неизбежно ускользают даже от самого строгого процесса тестирования и впервые встречаются уже у конечных пользователей программного обеспечения. Когда это происходит, разработчикам необходимо оперативно разобраться в причинах произошедших ошибок и устранить их. В качестве инструмента для поиска причин таких ошибок авторы предлагают средства так называемой посмертной отладки. В статье рассмотрены современное состояние методов отладки и роль инструментов посмертного анализа, а также сформулированы требования к средствам посмертной отладки для критически важных систем. Приведено описание реализованного авторами механизма посмертной отладки в отечественной операционной системе реального времени серии Багет и сформулированы задачи для дальнейшего развития.

Ключевые слова: *corefile, посмертный анализ, ОСРВ Багет, отладка, дамп памяти*

Введение

Еще в 1951 г., на заре современной вычислительной техники, Стэнли Гилл [1] писал о том, что следует уделять особое внимание к тем нештатным ситуациям, которые возникают уже после запуска программы и приводят к ее аварийному завершению. Гилл считается основоположником [2] так называемой посмертной отладки (*postmortem debugging*), когда в программе или системе реализуют функцию записи состояния в момент аварийного завершения, чтобы программист мог позже понять, что произошло и почему такая ситуация возникла.

С тех пор технология посмертной отладки используется в различных системах, включая все основные операционные системы (ОС) общего назначения, а также специализированные ОС, такие как встраиваемые системы и системы реального времени. Чтобы обеспечить высокий уровень надежности, ожидаемый от таких критически важных систем, необходимо, с одной стороны, реализовать возможность быстрого восстановления системы или ее части после произошедшего сбоя. С другой стороны, необходимо предусмотреть механизм сохранения как можно большего количества информации после каждого сбоя, чтобы впоследствии можно было выяснить причину его возникновения.

Чтобы понять ценность методов посмертной отладки, следует рассмотреть их классическую альтернативу. Практически все современные ОС, в том числе отечественная ОС реального времени (ОСРВ) Багет [3], в настоящее время предоставляют средства для так называемой отладки *in situ* (лат. "на месте"), заключающейся в поиске причин ошибок в ходе выполнения самой программы. Обычно процесс такой отладки включает в себя присоединение отдельной программы отладчика к неисправной программе, а затем управление выполнением неисправной программы в интерактивном режиме, инструкция за инструкцией или с использованием точек останова. Таким образом, пользователь останавливает программу в различных точках, чтобы проверить состояние программы и выяснить, где ее поведение начинает отличаться от ожидаемого. Зачастую этот процесс повторяется для проверки различных гипотез о причине возникновения ошибки.

Этот метод может быть очень эффективен для ошибок, которые воспроизводятся почти при каждом запуске программы, однако у него есть несколько недостатков. Во-первых, действие по остановке программы часто меняет ее поведение. Ошибки, возникающие в результате взаимодействия между параллельно выполняемыми потоками управления или процессами (например, состояние гонки — *race condition*), могут быть особенно сложными для такого анализа, поскольку интерактивная отладка влияет на время возникновения различных событий (так называемые ошибки Гейзенберга — *Heisenbugs*). Что еще более важно, отладка *in situ* часто оказывается несостоятельной в условиях реальной эксплуатации (*production environment*), поскольку многие отладчики эффективно работают только с неоптимизированной отладочной сборкой, которая слишком медленна для запуска в условиях реальной эксплуатации. Специалисты по отладке часто не имеют доступа к системам, в которых выполняется программа. Этот

случай характерный для большинства мобильных и настольных приложений и многих корпоративных систем. Следует также отметить, что необходимые инструментальные средства отладки зачастую в этих системах недоступны.

Даже в тех случаях, когда специалисты по отладке могут получить доступ к программному обеспечению (ПО), содержащему ошибки, с помощью необходимых им инструментальных средств, приостановка программы в отладчике обычно представляет собой неприемлемое нарушение производственного процесса и высокий риск того, что использование интерактивного отладчика может привести к сбою программы.

Самый серьезный проблемный вопрос с отладкой *in situ* заключается в том, что ее можно использовать только для обнаружения причин воспроизводимых ошибок. Многие ошибки в условиях реальной эксплуатации либо очень редки, либо связаны со сложными взаимодействиями различных систем, которые часто очень трудно воспроизвести в среде разработки. Редкий характер такого сорта ошибок не может заставить разработчиков закрыть на них глаза. Напротив, сбой ОС, который происходит только 1 раз в неделю, может быть чрезвычайно дорогостоящим с точки зрения простоя. Аналогично фатальная ошибка, которая возникает 1 раз в неделю в приложении, используемом тысячами людей, может привести к тому, что многие пользователи будут сталкиваться с ошибкой каждый день. Разработчики при этом по объективным причинам не смогут подключить отладочные средства к системе каждого пользователя.

Еще один популярный способ отладки — протоколирование (полное или частичное) событий, возникающих в ходе выполнения программы. Однако извлечение достаточного количества информации о фатальных сбоях из журнала событий зачастую является трудоемким процессом. Как следствие, разработчикам приходится выполнять несколько итераций модификации программы с добавлением новых событий, развертывания модифицированной программы и изучения выходных данных. Это обстоятельство также может оказаться неприемлемым в условиях реальной эксплуатации, поскольку специальные изменения кода часто нецелесообразны (в случае настольных и мобильных приложений) или запрещены политикой контроля изменений.

Решение состоит в том, чтобы создать средство, которое будет фиксировать состояние программы целиком в момент, когда программа выходит из строя. Отметим, что после того как система сохранит состояние программы, она может немедленно эту программу перезапустить, чтобы избежать простоя.

Подводя итог изложенному выше, можно утверждать, что для эффективной работы по выявлению первопричин возникающих сбоев средство посмертной отладки должно удовлетворять следующим требованиям.

1. Для обеспечения функционирования средства посмертной отладки прикладное ПО не должно требовать модификаций, которые не могут быть использованы в условиях реальной эксплуатации, например, запрет оптимизации кода или наличие дополнительной отладочной информации, значительно влияющих на производительность.

2. Средство посмертной отладки должно быть всегда включено: оно не должно требовать от администратора системы подключения отладчика или выполнения иных нештатных действий до возникновения проблемы.

3. Средство посмертной отладки должно быть полностью автоматическим: оно должно обнаружить сбой, сохранить состояние программы, а затем немедленно позволить системе перезапустить вышедший из строя компонент.

4. Дамп (сохраненное состояние) должен быть исчерпывающим: содержимое стека, будучи, вероятно, самой ценной частью информации, очень часто не предоставляет достаточной информации, чтобы выявить причину возникновения проблемы. Обычно разработчикам требуется анализировать такие параметры каждого потока, как стек, регистры и память. Кроме того, в зависимости от самой целевой системы в дамп должна попадать и специфическая информация, которая может оказаться полезной разработчикам, например, состояния системных объектов синхронизации, сети и подключенной аппаратуры. Чем больше информации может быть включено в дамп, тем больше вероятность того, что разработчики смогут определить первопричину, основываясь только на одном возникшем сбое.

5. Должна быть реализована возможность анализировать полученный дамп не только в условиях реальной эксплуатации, но и в отладочном окружении. Такой подход позволит разработчикам использовать дополнительные отладочные средства для анализа данных.

Посмертная отладка в ОС семейства Unix

Чтобы понять потенциальную ценность посмертной отладки для критически важных систем, полезно посмотреть те ОС, где методы посмертного анализа хорошо развиты и широко используются. По мнению авторов, такими ОС являются ОС семейства Unix.

В качестве примера рассмотрим программу из листинга 1.

```
1 long *func ()
2 {
3    /* Some action */
4    return (long *) 0;
5 }
6
7 void main ()
8 {
9    long *address = func();
10   /* Some code */
11   *address = 0xdeadbeaf;
12  return;
13 }
```

Листинг 1. Пример программы с ошибкой

После сборки и запуска этой программы пользователь увидит следующее сообщение:

```
$ gcc -g -o example1 example1.c
$ ./example1
Segmentation fault (core dumped)
```

Хотя этот пример надуман, он иллюстрирует основной метод посмертной отладки. Следует обратить внимание на тот факт, что в отличие от отладки *in situ*, этот метод хорошо масштабируется по мере роста отлаживаемой программы. Если бы вместо одного потока в одном процессе были тысячи потоков в десятках компонентов (как в случае ОС), полный дамп включал бы информацию обо всех. Безусловно, перед программистами в этом случае будет стоять вопрос, как разобраться в таком большом количестве информации. Однако по крайней мере можно будет выявить причину ошибки, поскольку вся информация доступна.

Следующим шагом является создание пользовательских инструментальных средств для извлечения и анализа конкретного состояния процесса, завершившегося аварийно. Комплексный подход с привлечением других отладочных средств здесь будет, безусловно, полезен. Например, gdb (GNU Debugger [4]) предоставляет механизм создания пользовательских команд (макросов) как на встроенном языке отладки, так и на языке Python. Эти макросы можно распространять вместе с исходным кодом, чтобы все разработчики могли использовать их как *in situ* (присоединяясь к уже запущенному процессу), так и посмертно (открывая основной файл с помощью gdb).

Если открыть файл `core`, полученный из листинга 1, с помощью отладчика `gdb` (листинг 2), то можно заметить, как отладчик обрабатывает этот файл. В представленном примере исполняемый файл был собран с отладочной информацией, поэтому пользователю доступен его исходный текст на языке Си.

```
$ gdb -c core.3603
...
[New LWP 3603]
Core was generated by `./example1'.
Program terminated with signal SIGSEGV,
Segmentation fault.
#0 0x0000000000400520 in ?? ()
(gdb) file example1
Reading symbols from example1...done.
(gdb) bt
#0  0x0000000000400520  in  main  ()  at
example1.c:11
(gdb)
```

Листинг 2. Пример сеанса работы `gdb` с файлом `core`

Ошибка, которая привела к сбою, произошла в строке 11, где, как видно на листинге 1, происходит попытка модификации памяти по нулевому указателю.

Задача, стоящая перед авторами, заключается в том, чтобы адаптировать механизм посмертной отладки для отечественной ОСРВ Багет 3.0 [3]. Решение этой задачи будет рассмотрено в следующем разделе.

Посмертная отладка в ОСРВ Багет 3.0

Особенность ОСРВ Багет 3.0 заключается в том, что в условиях реальной эксплуатации она должна эффективно выполнять свои функции, минимизируя задержки, связанные с обработкой ошибок. В частности, по стандарту ARINC 653 [5] процесс, в котором произошла исключительная ситуация, перезапускается. Разумеется, применение стандартного отладчика *in situ* в этой ситуации невозможно. Решением является интеграция в систему средств посмертного анализа с сохранением аварийного дампа процесса для его последующего изучения. Со стороны ОСРВ Багет 3.0 был реализован обработчик, осуществляющий запись аварийного дампа в файл, аналогичный файлам `core` для ОС типа Unix. При этом потребовалась некоторая доработка формата `core` для сохранения всей возможной информации, которая может понадобиться разработчикам при поиске причины возникновения сбоя.

Основными отечественными аппаратными платформами для запуска ОСРВ Багет 3.0 являются процессоры с архитектурой mips32 и mips64. В случае посмертного анализа в целях унификации формата данных всегда удобно оперировать 64-битными данными. Поэтому за основу формата файла `core` ОСРВ Багет для 32-битных архитектур был взят формат ELF32 файла `core` ОС Linux для архитектуры mips64 с ABI n32 или o64 [6], а для 64-битных архитетур — формат ELF64 файла `core` ОС Linux для архитектуры mips64.

Следует отметить, что основным форматом бинарных файлов, используемым как в ОСРВ Багет, так и в ОС Linux, является так называемый формат ELF (*Executable and Linkable Format* [7]).

На листинге 3 приведен заголовок `ELF`-файла `core`.

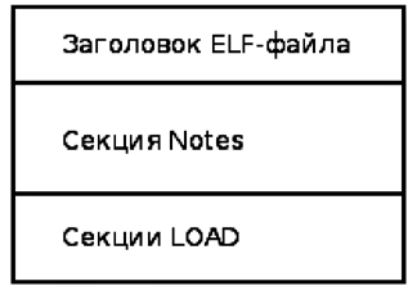
```
$ readelf -h core.1
ELF Header:
Magic:              7f 45 4c 46 01 02 01 00
                    00 00 00 00 00 00 00 00
Class:              ELF32
Data:               2's complement, big endian
Version:            1 (current)
OS/ABI:             UNIX — System V
ABI Version:                0
Type:               CORE (Core file)
Machine:            MIPS R3000
Version:            0x1
Entry point address: 0x0
Start of program headers:  52 (bytes into file)
Start of section headers:  0 (bytes into file)
Flags:              0x60002001, noreorder,
                    o64, mips64
Size of this header: 52 (bytes)
Size of program headers:  32 (bytes)
Number of program headers:  4
Size of section headers:  40 (bytes)
Number of section headers:  0
```

Листинг 3. Заголовок `ELF`-файла `core`

Следует обратить внимание на тип файла (указан как CORE), а также на флаги, подсказывающие отладчику, каким образом следует интерпретировать данные, содержащиеся в этом файле. В контексте настоящей статьи будут интересовать флаги, определяющие формат бинарного интерфейса уровня приложения (*Application Binary Interface* — ABI), в данном случае это o64.

В общих чертах структура файла `core` приведена на рисунке.

Заметим, что этим структура файла `core` не исчерпывается. Как и всякий ELF файл, он может содержать дополнительные секции с данными, специфическими для целевой программно-аппаратной архитектуры. Например, информацию о состоянии используемых системных объектов ОСРВ, таких как очереди сообщений, семафоры или мьютексы.



Структура файла `core`

В нашем примере программные заголовки ELF файла выглядят следующим образом (листинг 4).

```
Program Headers:
      Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flg    Align
      NOTE           0x0000b4   0xc001f078 0x00000000 0x006b4    0x00000           0x4
      LOAD           0x000768   0x00050d98 0x00000000 0x03c38    0x03c38    R      0x8
      LOAD           0x0043a0   0x10000000 0x00000000 0x0e940    0x0e940    RW     0x8
      LOAD           0x012ce0   0x1000e940 0x00000000 0x018ad    0x018ad    RW     0x8
```

Листинг 4. Программные заголовки файла `core`

Секции `LOAD` содержат дамп памяти пользовательского процесса ОСРВ Багет, породившего файл `core`. Если сравнить адреса сохраненных данных с адресами загружаемых секций исходного объектного файла, то видно, что они совпадают с адресами секций `.data` и `.bss` (листинг 5).

```
$ bt23j-objdump -h posix/userproc.o
...
Разделы:
 Idx Name          Size      VMA       LMA       File off  Algn
     0.text        0005508c  00000000  00000000  00001000  2**3
                             CONTENTS, ALLOC, LOAD, READONLY, CODE
     1.data        0000e940  10000000  10000000  00057000  2**3
                   CONTENTS,  ALLOC,   LOAD,     DATA
     2.bss         000018ad  1000e940  1000e940  00065940  2**3
                             ALLOC
 ...
```

Листинг 5. Программные заголовки файла `core`

В секции `Notes` содержится информация о процессе, который породил файл `core`: название файла; содержимое регистров; номер сигнала, заставивший процесс породить файл `core`.

Если взглянуть на эту секцию посредством утилиты `readelf`, то можно увидеть, что она состоит из нескольких записей (листинг 6).

```
$ readelf -n core.1
Notes at offset 0x000000b4 with length 0x000006b4:
  Owner  Data size  Description
   CORE  0x000001e0 NT_PRSTATUS (prstatus structure)
   CORE  0x00000108 NT_FPREGSET (floating point registers)
   CORE  0x000001e0 NT_PRSTATUS (prstatus structure)
   CORE  0x00000108 NT_FPREGSET (floating point registers)
   CORE  0x00000080 NT_PRPSINFO (prpsinfo structure)
```

Листинг 6. Содержимое секции `Notes`

В данном примере секция `Notes` содержит записи о двух потоках управления в структурах `prstatus` (идентификаторы, содержимое регистров общего назначения) и в структурах `fpregset` (плавающие регистры), а также общую информацию о процессе, породившем файл `core` в структуре `psinfo` (имя

файла-образа, аргументы). Размеры структур `prstatus` и `psinfo` фиксированы и различаются в зависимости от того, является ли целевая архитектура 32- или 64-битной, а также от того, какой тип mips ABI используется. Следует отметить, что современные версии библиотеки GNU BFD (*Binary File Description*) не поддерживают ABI o64. По этой причине авторами статьи среди прочих изменений отладчика ОСРВ Багет 3.0 (отладчик реального времени — ОРВ) для поддержки файлов `core` ОСРВ Багет была проведена модификация используемой отладчиком библиотеки BFD (*Binary File Description*). При этом следует упомянуть, что с ABI n32 библиотека BFD работает корректно.

Отдельно необходимо отметить реализованную возможность порождать файлы `core` отлаживаемого процесса ОСРВ Багет непосредственно из отладчика ОРВ посредством специальной команды. Такая возможность имеет смысл в тех случаях, когда отлаживаемый процесс не завершился аварийно, однако в ходе его выполнения возникли определенные проблемы, требующие дальнейшего детального изучения.

На листинге 7 приведен пример сеанса работы ОРВ с файлом `core` (комментарии авторов в листинге выделены полужирным шрифтом). Из этого листинга видно, что почти всю необходимую информацию ОРВ получает и отображает корректно.

```
Отладчику передается в качестве аргумента имя файла-образа и имя файла core

$ OPB40-gdb posix/userproc.o -c core.1
GNU gdb (GDB) 7.11
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3 +: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host = x86 _ 64-pc-linux-gnu --target = mips64-oc2000".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from posix/userproc.o...done.

Прочитана информация о двух потоках управления

[New LWP 1]
[New LWP 2]

Информации о бинарном файле-образе в разделе psinfo нет

Core was generated by `'.
#0 main () at main.c:14
14  main.c: Нет такого файла или каталога.
[Current thread is 1 (LWP 1)]
(gdb) info threads
  Id Target Id            Frame
* 1  LWP 1                main () at main.c:14
  2  LWP 2                0xc419bad8 in?? ()
(gdb) info frame
Stack level 0, frame at 0x101fff68:
 pc = 0xf8 in main (main.c:14); saved pc = 0x1c60
 called by frame at 0x101f9ce0
 source language c.
 Arglist at 0x101f9ca8, args:
 Locals at 0x101f9ca8, Previous frame's sp is 0x101f9ca8
 Saved registers:
  s8 at 0x101f9c98, ra at 0x101f9ca0, pc at 0x101f9ca0
(gdb) info registers
... Информация о регистрах...
```

Листинг 7. Сеанс работы ОРВ с файлом `core`

Заключение

Представлены новые возможности посмертной отладки пользовательских процессов ОСРВ Багет. Дальнейшие работы в этом направлении могут состоять в доработке формата файлов `core`, включающие больше данных о породившем файл `core` процессе, например, имя файла-образа, а также до-

полнительные программные секции, содержащие информацию о системных объектах ОСРВ Багет, относящихся к данному процессу.

Работа выполнена в рамках государственного задания по проведению фундаментальных научных исследований по теме (проекту) "38. Проблемы создания глобальных и интегрированных информационно-телекоммуникационных систем и сетей, развитие технологий и стандартов GRID. Исследование и реализация программной платформы для перспективных многоядерных процессоров (0065-2019-0002)".

Список литературы

1. **Gill S.** The diagnosis of mistakes in programmes on the EDSAC// Proceedings of the Royal Society. — 1951. — P. 538—554.

2. **Pacheco D.** Postmortem Debugging in Dynamic Environments// Communication of the ACM. — 2011. — Vol. 54, Iss. 12. — P. 44—51.

3. **Godunov A. N., Soldatov V. A.** Baget real-time operating system family (features, comparison, and future development) // Programming and Computer Software. — 2014. — Vol. 40, No. 5. — P. 259—264.

4. **GDB:** The GNU Project Debugger. URL: https://www.gnu.org/software/gdb (дата обращения 15.04.2021).

5. **Santos S., Rufino J., Schoofs T.** et al. A portable ARINC 653 standard interface// 2008 IEEE/AIAA 27th Digital Avionics Systems Conference. URL: https://ieeexplore.ieee.org/document/4702767 (дата обращения 15.04.2021).

6. **MIPS** calling conventions. URL: https://techpubs.jurassic.nl/manuals/-63/developer/Mpro_n32_ABI/sgi_html/ch02.html (дата обращения 15.04.2021).

7. **ELF** format specification. URL: http://www.skyfree.org/linux/references/ELF_Format.pdf (дата обращения 15.04.2021).

Postmortem Analysis of Baget RTOS Processes

V. A. Galatenko, galat@niisi.ras.ru, **K. A. Kostyukhin**, kost@niisi.ras.ru, Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences", Moscow, 117218, Russian Federation

Corresponding author:

Kostyukhin Konstantin A., Senior Researcher, Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences", Moscow, 117218, Russian Federation
E-mail: kost@niisi.ras.ru

Received on April 15, 2021
Accessed on May 18, 2021

Despite the best efforts of programmers to create high-quality software, some errors inevitably escape even the most rigorous testing process and are first encountered by end users of the software. When this happens, developers need to quickly understand the reasons for the errors that occurred and eliminate them. Back in 1951, at the dawn of modern computing, Stanley Gill wrote that special attention should be paid to those errors that occur after the program is started, and lead to its termination. Gill is considered the founder of the so-called postmortem debugging, when a program or system is modified to record its state at the time of failure, so that the programmer can later understand what happened and why such a situation occurred.

Since then, postmortem debugging technology has been used in many different systems, including all major general-purpose operating systems (OS), as well as specialized OS such as embedded systems and real-time systems. To ensure the high level of reliability expected from such critical systems, it is necessary, on the one hand, to implement the possibility of rapid recovery of the system or its part after a failure. On the other hand, it is necessary to provide a mechanism for storing as much information as possible after each failure, so that the cause of its occurrence can be determined later.

To understand the real potential of postmortem debugging tools, we will first consider the current state of debugging methods and the role of postmortem analysis tools, as well as the requirements for postmortem debugging tools for critical systems. Next, we will describe the mechanism of postmortem debugging implemented by the authors in the RTOS Baget and formulate tasks for further development.

Keywords: *corefile, postmortem analysis, RTOS Baget, debugging, memory dump, real-time debugger*

For citation:

Galatenko V. A., Kostyukhin K. A. Postmortem Analysis of Baget RTOS Processes, *Programmnaya Ingeneria*, 2021, vol. 12, no. 5, pp. 227—232

DOI: 10.17587/prin.12.227-232

References

1. **Gill S.** The diagnosis of mistakes in programmes on the EDSAC, *Proceedings of the Royal Society*, 1951, pp. 538—554.

2. **Pacheco D.** Postmortem Debugging in Dynamic Environments, *Communication of the ACM,* 2011, vol. 54, iss. 12, pp. 44—51.

3. **Godunov A. N., Soldatov V. A.** Baget real-time operating system family (features, comparison, and future development), *Programming and Computer Software*, 2014, vol. 40, no. 5, pp. 259—264.

4. **GDB:** The GNU Project Debugger, available at: https://www.gnu.org/software/gdb

5. **Santos S., Rufino J., Schoofs T.** et al. A portable ARINC 653 standard interface, *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, available at: https://ieeexplore.ieee.org/document/4702767

6. **MIPS** calling conventions, available at: https://techpubs.jurassic.nl/manuals/0630/developer/Mpro_n32_ABI/sgi_html/ch02.html

7. **ELF** format specification, available at: http://www.skyfree.org/linux/references/ELF_Format.pdf

УДК 004.75 DOI: 10.17587/prin.12.233-245

Д. Ю. Булгаков, адъюнкт, dbulgakov7@yandex.ru, Федеральное государственное казенное образовательное учреждение высшего образования "Академия управления Министерства внутренних дел Российской Федерации", Москва

Использование распределенных облачных вычислений при решении ресурсоемких задач, активно использующих CPU

Предложен способ решения ресурсоемких задач, активно использующих CPU, когда вычислительных ресурсов одного сервера становится недостаточно. Потребность в решении данного класса задач возникает при использовании в продуктивной среде различных моделей машинного обучения, а также в научных исследованиях. Облачные вычисления позволяют организовать распределенную обработку задач на виртуальных серверах, которые легко создавать, поддерживать и тиражировать. Предложен и обоснован подход, основанный на использовании свободного программного обеспечения на базе языка программирования Python. Полученное решение рассмотрено с точки зрения теории массового обслуживания. Описан эффект от применения предложенного подхода при решении задач распознавания лиц и анализа биомедицинских сигналов.

Ключевые слова: *облачные вычисления, распределенные вычисления, распознавание лиц, теория массового обслуживания, биомедицинские сигналы, анализ ЭЭГ, Python, Celery, Flask, RabbitMQ*

Введение

Одной из перспективных задач информационного обеспечения правоохранительной деятельности в настоящее время является задача идентификации и верификации личности по биометрии лица [1—3], называемая *распознаванием лиц*. В современных алгоритмах распознавания лиц используются, как правило, глубокие сверточные нейронные сети. Начиная с 2013 г. именно алгоритмы, основанные на сверточных нейронных сетях, демонстрируют наиболее впечатляющие результаты [4, с. 6; 5, с. 6].

В то же время процедура извлечения биометрического шаблона из фотоизображения лица является задачей, требующей для своего выполнения значительных вычислительных ресурсов, а именно — ресурсов центрального процессора (CPU). Выполнение этой задачи занимает в зависимости от алгоритма от 0,01 до 1 с на одном ядре процессора типа Intel Xeon CPU E5-2630 v4 2.20 GHz [5, с. 10, 41—44]. Время построения биометрического шаблона для самых точных алгоритмов составляет в среднем 0,65 с [5, с. 39, 40]. Если требуется обработать очень большое число фотографий (более 100 000 000) и для всех этих фотографий создать биометрические шаблоны, то на это может потребоваться значительное время. Допустим, что в наличии имеется один сервер с 10 CPU-ядрами, который строит один биометрический шаблон за 0,65 с на одном ядре. В этом случае построение 100 млн биометрических шаблонов при задействовании всех ядер данного сервера займет не менее 75 сут[1]. При этом необходимо учитывать, что здесь не учтено время, затрачиваемое на извлечение информации из базы данных, передачу информации по сети и сохранение информации в базе данных.

Может возникнуть задача, когда нужно будет обработать все эти фотографии в течение одних суток. Эта задача не выглядит надуманно, если предположить, что ежесуточно такое число фотографий с лицами может поступать со всех камер видеонаблюдения крупного мегаполиса, подключенных к системе детектирования лиц. Данная система будет выявлять в видеопотоке лица людей, сохранять детектированные лица как статические фотографии и передавать их в систему распознавания лиц.

Данная задача требует решения не на стадии тренировки или обучения модели распознавания лиц, а именно при использовании обученной модели в продуктивной среде, когда время выполнения задачи имеет решающее значение.

Очевидно, что задача обработки 100 млн фотографий должна хорошо поддаваться распараллеливанию, так как обработка любой из этих фотографий не зависит от результата обработки другой фотографии. Соответственно, можно обрабатывать их одновременно независимо друг от друга.

[1] (100 млн фотографий × 0,65 с/фото/ядро CPU) / (86 400 с/сут × 10 ядер CPU) ≈ 75 сут.

Можно предположить, что если иметь в своем распоряжении распределенную вычислительную систему из 75 серверов с 10 CPU-ядрами, то теоретически возможно решить задачу обработки 100 млн фотографий за одни сутки.

Потребность в значительных вычислительных ресурсах также возникает при решении задачи анализа биомедицинских сигналов. Например, при исследовании восприятия времени [6] были записаны электроэнцефалограммы (ЭЭГ) 40 испытуемых длительностью 5 мин с частотой дискретизации 1000 Гц. При этом использовалось 10 каналов-отведений, подключенных по международной системе "10—20". Таким образом, получен набор из 400 сигналов (40 испытуемых, × 10 каналов) длительностью по 300 с каждый.

В целях получения информации о ритмах ЭЭГ в ходе исследования необходимо выполнить оконное дискретное преобразование Фурье с шириной окна в 3 с, начиная с 0-й с. Будем сдвигать каждое последующее окно вперед на 1 с. Анализируемый отрезок времени будет составлять 3 с. Под информацией о наличии ритмов ЭЭГ в заданный отрезок времени будем понимать сумму спектральной плотности мощности сигнала в определенном интервале частот, характеризующих данный ритм.

Каждый из 400 сигналов разбивается на отрезки времени длительностью 3 с. Каждый последующий отрезок отстоит от предыдущего на 1 с — таким образом, отрезки перекрывают друг друга на 1 или 2 с. В итоге получаем 120 000 сигналов (400 сигналов × 300 отрезков) длительностью 3 с каждый.

Если отказаться от использования алгоритма быстрого преобразования Фурье и использовать классическое дискретное преобразованию Фурье (DFT) в целях сохранения точности расчетов, то вычисление спектральной плотности мощности для всех 120 000 сигналов займет значительное время, измеряемое часами. Допустим, обработка каждого сигнала длительностью в 3 с занимает всего 1 с, тогда последовательная обработка всех сигналов займет 33,3 ч[1]. Если сдвигать окно не на 1 с, а на 0,1 с, то число сигналов вырастет в 10 раз (1 200 000) и время, требуемое на их обработку, также вырастет в 10 раз и составит уже 333 ч/24 ч ≈ 14 сут.

При этом возможно *распараллелить* обработку этого множества сигналов, так как их обработка не зависит друг от друга, и по итогам обработки свести полученные результаты в общую таблицу.

Таким образом, возникает цель — организовать эффективную обработку большого числа ресурсоемких задач в распределенной вычислительной системе. При этом желательно, чтобы было возможно *динамически* добавлять в вычислительную систему вычислительные узлы, когда в них возникает потребность, и исключать из системы эти вычислительные узлы, когда потребность в них пропадает.

В качестве примеров моделей для оптимизации выбраны:

- задача построения биометрических шаблонов по изображению лица по имеющимся базам данных фотоизображений общим объемом более 250 млн фотографий;
- анализ 120 000 сигналов, полученных из 40 ЭЭГ.

В данной статье не рассматриваются особенности используемых алгоритмов: распознавания лиц — построения биометрических шаблонов по изображению лица на основе сверточных нейронных сетей [7] и поиска по построенным биометрическим шаблонам на основе алгоритмов поиска *k*-ближайших соседей в пространствах высокой размерности [8], оконного дискретного преобразования Фурье для анализа биомедицинских сигналов [6]. Намеренно не делается акцент на обработке экспериментальных исходных данных. Данным вопросам посвящены отдельные публикации.

Целью статьи является демонстрация особенностей создания контролируемого, удобного в сопровождении и эффективного способа организации горизонтально масштабируемых, распределенных вычислений в облачной инфраструктуре — вычислений, потенциально произвольного характера.

Обзор возможных решений

CPU и GPU

Как показывает анализ литературных источников [9, с. 85—87], значительного ускорения вычислений можно добиться, если использовать для решения вычислительных задач не CPU, а GPU [10]. Это справедливо, в том числе для задачи построения биометрических шаблонов [10, с. 9, 61, 62]. В то же время это может быть сопряжено с рядом трудностей. Например, не все алгоритмы распознавания лиц могут быть адаптированы для работы на доступных GPU, или же их адаптация под целевые алгоритмы может вызвать затруднения инженерного характера.

В качестве примеров подобного рода затруднений пользователя можно привести следующие случаи. Пользователь может не располагать исходными кодами используемых моделей по причине лицензионных ограничений или утраты исходных кодов, чтобы адаптировать эти модели к работе на GPU. Пользователь может использовать скомпилированное проприетарное программное обеспечение, которое нельзя перекомпилировать для использования на GPU. Пользователь может располагать скомпилированными версиями программного обеспечения, совместимыми со старыми версиями GPU и их драйверами, но не совместимыми с новыми GPU и новыми версиями драйверов для GPU.

Число имеющихся в распоряжении пользователя серверов с GPU может быть гораздо меньше числа имеющихся серверов с CPU. Приобретаемые серверы с GPU, как правило, узко специализированы и устаревают быстрее, чем серверы с CPU. Серверы с GPU нельзя будет приспособить для решения любых вычислительных задач, если потребность в GPU пропадет. Поэтому не все организации стремятся наращивать парк серверов с GPU.

Наконец, всегда можно рассчитывать на выделение и быстрое тиражирование виртуальных серверов

[1] 120 000 сигналов / 60 с / 60 мин ≈ 33,3 ч.

с CPU в виде IaaS[1], если в организации есть собственный облачный центр обработки данных (ЦОД) или можно арендовать виртуальные вычислительные ресурсы с CPU в сети Интернет. В большинстве случаев аренда виртуальных серверов с CPU будет более простым, быстрым и дешевым решением, чем аренда виртуальных серверов с GPU.

Если все-таки будет принято решение организовать вычисления на нескольких серверах с GPU или будет использоваться смешанный вариант — серверы с GPU плюс серверы с CPU, то задача создания распределенной вычислительной системы по-прежнему остается актуальной.

Выбор языка программирования

Вычислительные модели, используемые в продуктивной среде, могут быть созданы с использованием различных технологий и языков программирования. Это относится и к вычислительным моделям, используемым в распознавании лиц.

Например, если разработчик алгоритма распознавания лиц хочет направить его алгоритм на тестирование в Национальный институт стандартов и технологий США[2], занимающийся тестированием алгоритмов распознавания лиц на постоянной основе, то он должен оформить свой биометрический алгоритм, реализующий функции построения биометрических шаблонов и поиска по ним, в виде бинарной программы, взаимодействующей по API с тестирующим программным обеспечением, разработанным NIST на языке программирования C++ [11].

В данном исследовании делается акцент на решение ресурсоемких задач, связанных с распознаванием лиц [7], обработкой биомедицинских сигналов [6, 12], задач, использующих различные модели машинного обучения в продуктивной среде, а также иных вычислительных задач, возникающих в ходе научных исследований и требующих значительных процессорных ресурсов. Необходимо разработать универсальное решение, которое можно было бы использовать для широкого круга задач, требующих параллельного выполнения большого числа вычислительных операций.

Как было обозначено выше, с 2013 г. решение задач, связанных с распознаванием лиц, основано на использовании сверточных нейронных сетей [4, с. 6; 5, с. 6] и, как следствие, — различных библиотек-фреймворков, предоставляющих возможность создания, обучения и использования моделей, основанных на нейронных сетях данного вида. Внутренняя реализация и языки программирования, с использованием которых созданы данные фреймворки машинного обучения, могут быть различными, но в подавляющем большинстве случаев все они предоставляют API для взаимодействия с ними из программ, написанных на языке программирования Python [9, с. 91, 92, 100, 101, 111]. Сказанное справедливо не только для фреймворков машинного обучения, но и для других библиотек, предназначенных для решения широкого класса задач — подавляющее их большинство имеют интерфейсы для взаимодействия с Python-программами [12].

Достаточно распространенным решением [9, с. 79] для параллельных вычислений и анализа больших данных является использование фреймворков Apache Hadoop [13] и входящего в данную экосистему Apache Spark [14]. Эти фреймворки написаны на языках программирования Java и Scala соответственно, и подходят в большей степени именно для обработки и анализа больших наборов данных с использованием парадигмы распределенных кластерных вычислений MapReduce, а не для научных вычислений.

Для научных вычислений, которые работают с изменяющимся набором данных, требований и алгоритмов, более подходящим решением является использование языка программирования Python и решений, созданных на его основе.

Язык программирования Python представляет собой высокоуровневый язык программирования общего назначения, используемый как для научных исследований, так и для веб-разработки. В настоящее время Python является одним из самых популярных языков программирования [15] и начиная с 2016 г. самым популярным языком программирования, используемым в научных исследования, задачах анализа данных и машинного обучения [9, с. 99; 16]. Поэтому при решении задачи распределенных вычислений рассматривались решения, разработанные на Python.

Следует особо обратить внимание на то, что в данном случае Python используется в качестве инфраструктурного языка, из-под которого запускаются высокоэффективные программы-библиотеки, написанные на языках программирования C и C++. Именно программы на C и C++ загружают все процессорные ядра сервера на 100 %, и поэтому для масштабирования приходится прибегать к распределенным вычислениям, так как получить еще большую производительность на одном сервере без значительных изменений в самих моделях-алгоритмах невозможно. Это касается и распознавания лиц и обработки биомедицинских сигналов.

Аналогичные исследования

Исследование по схожей проблеме выполнено учеными из Германии [17]. Решение задачи в их случае основано на Python-фреймворках Django [18] и Celery [19]. В то же время в публикации не раскрывается, каким образом в созданную ими вычислительную систему поступают задачи и имеет ли их система открытый API для взаимодействия с иными системами. Авторы упоминают, что распределенные вычисления возможны и в этом случае нужно быть готовыми к дополнительным накладным расходам на передачу данных между вычислительными узлами [17, с. 57, 58]. В исследовании используется не распределенная система виртуальных серверов малой или средней вычислительной мощности, а один

[1] От англ. *Infrastructure as a service* — инфраструктура как услуга. Один из вариантов предоставления услуг в облачных вычислениях, при которой заказчику предоставляются виртуальные серверы с заданной вычислительной мощностью и операционной системой, но без дополнительно установленного специального программного обеспечения.

[2] *National Institute of Standards and Technology (NIST).*

мощный физический сервер с 20 процессорными ядрами — NVidia DGX Station [17, с. 57]. Соответственно, в их исследовании не продемонстрированы возможности горизонтального масштабирования полученного решения. О возможности использования GPU авторы не упоминают.

Достаточно подробный анализ возможности масштабирования обучения глубоких нейронных сетях в распределенной инфраструктуре выполнен исследователями из Мюнхенского технического университета [20]. В исследовании рассмотрены 11 популярных фреймворков машинного обучения с открытым исходным кодом[1], на базе которых возможно создание и обучение глубоких нейронных сетей. Проанализирована готовность фреймворков к тому, чтобы стадию *обучения* моделей, созданных на их основе, можно было проводить в распределенной вычислительной инфраструктуре. Вопрос рассматривается в трех аспектах: сложность самих моделей, которая зависит от глубины используемых нейронных сетей; объем данных, на которых проводится обучение; вычислительная мощность используемой инфраструктуры, зависящая от числа доступных вычислительных узлов, числа вычислительных ядер (особенно GPU) и вычислительной мощности этих ядер. В статье делается упор именно на обучение нейронных сетей, но не на их использование. При обучении одной из ключевых проблем, по мнению авторов, является проблема синхронизации данных и обмен информацией между вычислительными узлами [20, с. 15], чем больше узлов в вычислительной системе, между которыми должен происходить обмен информацией, тем сложнее и медленнее будет работать вся система [20, с. 10]. По итогам исследования оказывается, что не все рассматриваемые фреймворки поддерживают возможность масштабирования обучения [20, с. 22—26] — некоторые в большей степени, некоторые в меньшей, некоторые не поддерживают.

Таким образом, можно констатировать, что в современной литературе имеются публикации на тему организации распределенных вычислений, в том числе в условиях, когда необходим быстрый обмен данными между вычислительными узлами по принципу "каждый с каждым" или близкий к этому.

В настоящем исследовании акцент делается не на обучение моделей в распределенной вычислительной инфраструктуре, а на их использование. С учетом текущего уровня развития технологий в области распределенных вычислений существуют различные способы решения обозначенной задачи. В статье демонстрируется подход, позволяющей создавать распределенные вычисления в условиях наличия облачной инфраструктуры, когда необходимо быстро вносить корректировки в выполняемые задачи и тиражировать вычислительные узлы.

[1] Рассматриваемые фреймворки: Caffe, Caffe2, Chainer, CNTK, Deeplearning4j, Keras, MXNet, PyTorch, SINGA, TensorFlow, Theano.

Обоснование выбора решения

Как отмечено во введении, планируется проводить обработку вычислительных задач в распределенной системе серверов, так как изначально предполагается, что ресурсов одного сервера для решения нашей задачи будет недостаточно. Если необходима параллельная обработка задач на одном сервере, то для этих целей возможно использовать такие традиционные для Python методы решения задачи одновременного выполнения нескольких программ на одном сервере, как асинхронное выполнение задачи с помощью стандартных модулей Python *asyncio* [21], *concurrent.future* [22], выполнение программ в несколько потоков с помощью модуля *threading* [23] или выполнение программы в виде нескольких процессов на разных ядрах CPU с помощью модуля *multiprocessing* [24].

Задачи, требовательные к вводу-выводу, и многопоточное программирование

Модули Python *asyncio, concurrent.future* и *threading* могут помочь в задачах, активно задействующих ресурсы ввода-вывода, когда задержки выполнения каждой задачи связаны с ожиданием получения (передачи) каких-либо данных из (в) дисковой подсистемы или от (по) сетевого соединения. Например, к таким задачам можно отнести считывание (сохранение) большого числа файлов из (на) дисковой подсистемы или через сеть. В решении данного класса задач может помочь разбиение задачи на подзадачи и выполнение подзадач в несколько потоков (*threads*) или асинхронно. Каждый из этих потоков будет незначительно задействовать ресурсы CPU, и одновременное выполнение подзадач в потоках может существенно сократить время выполнения всей задачи. Каждый из потоков не будет ждать выполнения другого потока и не будет зависеть от выполнения другого потока. И наоборот, выполнение подзадач в один поток — синхронно или последовательно — будет приводить к значительному замедлению всей задачи. Каждая последующая подзадача должна будет дождаться выполнения предыдущий подзадачи, даже если результат выполнения последующей подзадачи не зависит от результата выполнения предыдущей подзадачи.

В качестве примера таких задач можно привести работу веб-сервера, с которым одновременно работает большое число пользователей. Файлы веб-сервера, которые должны отправляться пользователю, хранятся на диске. Каждый раз, когда пользователь обращается к веб-серверу за получением файлов, из которых состоит веб-страница, веб-сервер считывает очередной файл с диска (задержка ввода-вывода) и пересылает этот файл пользователю (сетевая задержка). Если бы чтение и передача этих файлов происходили последовательно, то процесс передачи всех файлов занимал бы слишком много времени. На практике от пользователя сразу поступает запрос на получение нескольких файлов и веб-сервер распараллеливает эта работу за счет считывания файлов с диска и передачи их поль-

зователю по сети в несколько потоков. При этом узким местом на уровне сервера может быть дисковая подсистема, которая не позволяет считывать файлы с высокой скоростью (байт/секунду) или не позволяет выполнять большое число операций ввода-вывода за единицу времени (IOPS[1]). Также узким местом может стать сетевой канал связи, который не позволит передавать файлы с высокой скоростью (байт/с) или будет иметь значительные сетевые задержки. Устранив узкие места дисковой и сетевой подсистем и увеличив число потоков в веб-сервере можно сократить время выполнения данной задачи.

Задачи, требовательные к вычислительным ресурсам процессора

В исследовании рассматривается класс задач, требовательных к вычислительным ресурсам процессора, в которых задержки вызваны ожиданием того, когда CPU или GPU проведет необходимые вычисления. Если добиваться ускорения выполнения задачи только на одном сервере, то возможны три варианта: увеличивать вычислительную мощность каждого ядра CPU, увеличивать число ядер в каждом CPU, увеличивать число CPU в сервере. Очевидно, что все три варианта имеют определенные технологические ограничения, и в данном случае будет проведена попытка дорогостоящего вертикального масштабирования.

Глобальная блокировка интерпретатора (GIL)

При использовании многопоточного программирования в Python существует инструментальное средство, ограничивающие выполнение Python-программы в несколько потоков, известное как глобальная блокировка интерпретатора (GIL[2]) [25]. Механизмы GIL блокируют одновременное использование ядра CPU более чем одним потоком программы в рамках одного процесса. Необходимость наличия GIL обусловлена архитектурными ограничениями *CPython* — базовой реализации языка программирования Python, написанной на языке программирования C. Средство GIL необходимо в целях создания *потокобезопасной*[3] модели выполнения Python-программы, таким образом, чтобы одновременно выполняющиеся потоки не повредили используемые этими потоками общие структуры памяти.

Программные механизмы GIL особенно сильно проявляют себя при попытке выполнения ресурсоемких вычислительных задач в несколько потоков. Они не позволяют одному процессу Python, независимо от числа используемых им потоков, задействовать более одного ядра CPU.

Ограничение аналогичное GIL имеет место не только в Python, но и в других популярных интерпретируемых языках программирования. Например, в реализации языка программирования Ruby на языке C — Ruby MRI, также известному как CRuby, имеется Global VM Lock [26] — аналог Python GIL.

В языке программирования JavaScript ситуация схожая — скрипты должны выполняться в один поток, но при этом задача *многопоточного программирования* эффективно решается через *конкурентное программирование*[4] с помощью *асинхронной модели* программирования. В то же время в некоторых реализациях JavaScript, например в Node.js, построенного на движке V8[5], начиная с Node.js версии 10, существует возможность запускать вспомогательные потоки, которые запускают код JavaScript в параллельном режиме для выполнения ресурсоемких вычислительных JavaScript-операций [27].

Необходимость наличия механизмов GIL в Python — это не особенность языка программирования Python, а особенность его реализации на языке C — CPython. Так, GIL-механизмы отсутствуют в Jython [28] — реализации Python, которая выполняется под управлением виртуальной машины Java[6] и IronPython [29] — реализации Python для платформы .NET Framework.

Следует отметить, что непосредственно на "чистом" Python крайне редко требуется программировать ресурсоемкие вычислительные задачи. Язык Python для этого не предназначен. Обычно для этого используются вспомогательные библиотеки и фреймворки, написанные на компилируемых языках программирования, таких как C, C++, FORTRAN. При вызове из Python-программы сторонних библиотек, GIL не оказывает влияния на их работу. Вызываемые сторонние библиотеки при этом могут задействовать для своей работы одновременно столько ядер процессора, сколько им потребуется для их эффективной работы. Примером такой библиотеки является NumPy [30] — базовый пакет для выполнения научных вычислений на Python. При вызове NumPy в различных потоках одного процесса будут задействованы различные ядра CPU [31].

Многопроцессорные и параллельные вычисления

Если в наличии имеется только один сервер и необходимо максимально использовать его производительность, задействовав все его CPU-ядра на 100 %, то эффективным средством является использование модуля *multiprocessing* [24]. Данный модуль позволяет запустить программу в несколько параллельных *процессов*, таким образом, что каждый процесс при выполнении будет использовать свое ядро CPU и не будет конкурировать за ресурсы CPU с другими процессами.

При решении задачи с помощью модулей *threading* и *multiprocessing* в рамках выполнения одной программы возникает вопрос совместного использования общих данных различными потоками и процессами этой программы. Этот вопрос разрешается с помощью Python-модулей *queue* [32], *heapq* [33] и вспомогательных структур, таких как *multiprocessing.Queue*.

---

[1] От англ. *input/output operations per second* — число операций ввода-вывода в секунду.

[2] Сокр. от англ. *global interpreter lock* — глобальная блокировка интерпретатора.

[3] Англ. *threadingsafe*.

[4] Англ. *concurrent programming*.

[5] Движок JavaScript с открытым исходным кодом, разработанный компанией Google.

[6] Англ. Java Virtual Machine (JVM).

Распределенные вычисления

При наличии несколько серверов естественным образом возникает желание организовать распределенные вычисления на них. При этом необходимо решить задачи согласованного, быстрого и надежного обмена данными по сети между клиентом и вычислительными узлами, а также задачи упорядочивания поступающих заданий в очереди и возвращения клиенту результатов выполненных заданий. Здесь на помощь приходят *брокеры сообщений* и *системы управления базами данных* (СУБД), при этом сама распределенная система начинает представлять собой *систему массового обслуживания*.

Как упоминалось во введении, задания по распознаванию лиц, поступающие на обработку, могут представлять собой не только статичный набор большого числа файлов с фотоизображениями. Это могут быть фотографии лиц, извлекаемые из видеопотоков, которые поступают с десятков и даже сотен тысяч видеокамер. Такие фотографии поступают неоднородно, через заранее неизвестные промежутки времени и требуют значительных вычислительных ресурсов для их обработки.

В связи с тем, что основным языком программирования, который использовался для решения обозначенной выше задачи, является Python рассматривались системы, позволяющие организовать распределенные вычисления именно на этом языке программирования. В качестве основных критериев, повлиявших на такой выбор, стали: широта распространения; наличие подробной документации; поддержка различных брокеров сообщений; устойчивость к сбоям; корректная обработка сбоев при передаче данных и обработке некорректных запросов; контролируемый характер выполнения; возможность масштабирования; возможность использования различных СУБД для сохранения результатов *заданий*; гибкость при адаптации под решение конкретных задач — изменение исходного кода *заданий*, возможность тиражирования *исполнителей*[1] в условиях облачной инфраструктуры, наличие средства мониторинга и контроля "в комплекте".

Распределенные вычисления с помощью Python

В экосистеме Python существует несколько проектов, связанных с организацией распределенной обработки заданий, которые можно рассматривать как основу для решения рассматриваемой задачи, а именно — Celery [19], RQ (Redis Queue) [34], Dramatiq [35] и др. [36]. Результатами выполнения этих проектов являются фреймворки с открытым исходным кодом, разработанные на языке программирования Python.

Популярность — число "звезд"[2] на веб-хостинге совместной разработки GitHub и число разработчиков[3] GitHub, внесших непосредственный вклад в разработку данных библиотек[4], выглядит следующим образом (звезд/разработчиков): Celery — 17,098/931; RQ — 7,654/202; Dramatiq — 2,540/60.

Фреймворк Celery — самое популярное и мощное решение для создания систем асинхронной, распределенной обработки заданий, имеющее богатые функциональные возможности и поддерживающее большое число СУБД для сохранения результатов выполнения заданий. В то же время некоторым разработчикам Celery представляется слишком сложным решением с функциональными возможностями, которые часто бывают не востребованы в простых проектах. Это обстоятельство заставляет разработчиков создавать иные проекты со схожими функциями.

Так, например, проект RQ (Redis Queue), по словам автора, основывается на лучшем, что есть в проектах Celery и Resque[5], и является легковесной альтернативой тяжеловесному Celery и другим аналогичным проектам [34]. При этом RQ поддерживает работу только с одной резидентной СУБД типа "ключ-значение"[6] Redis [37], которая используется и в качестве брокера сообщений, и как СУБД для сохранения результатов выполнения заданий.

Проект Dramatiq: "...это библиотека обработки фоновых заданий для Python с акцентом на простоту, надежность и производительность" [35]. В то же время Dramatiq в сравнении с Celery — это относительно молодой, но амбициозный проект. В перспективе он может составить серьезную конкуренцию Celery. Автор Dramatiq сам приводит сравнение своего решения с функциями, которые предоставляют другие аналогичные проекты [38].

Каждая из упомянутых выше библиотек позволяет тем или иным способом реализовать распределенные вычисления на Python. Самым популярным и богатым по своим возможностям решением является Celery, на основе которого и были реализованы распределенные вычисления.

Реализация выбранного решения

Архитектура распределенных вычислений

После анализа доступных вариантов решения задачи распределенных вычислений в качестве основного компонента, позволяющего организовать распределенные вычисления в Python, был выбран Celery [19]. В качестве брокера сообщений — RabbitMQ [39], в качестве СУБД для кратковременного сохранения результатов выполнения заданий — резидентная СУБД типа "ключ-значение" Redis [37], в качестве СУБД для долговременного сохранения результатов выполнения заданий — реляционная СУБД PostgreSQL [40]. В качестве операционной системы используется Linux CentOS 7.6 x86-64. Все предложенное программное обеспечение является свободным с открытым исходным кодом.

В качестве обоснования предлагаемого выбора можно привести следующие аргументы.

---

[1] От англ. *worker* — рабочий, исполнитель — программа, которая выполняет отдельное задание.

[2] Число пользователей GitHub, добавивших данный проект в свои избранные проекты.

[3] Англ. — *contributors*.

[4] По состоянию на 19 апреля 2021 г.

[5] Система управления заданиями для Ruby на СУБД Redis.

[6] Англ. — *in-memory key-value DBMS*.

Рис. 1. Предлагаемая архитектура распределенной вычислительной системы

Брокер сообщений RabbitMQ выбран в связи с тем, что данный брокер сообщений является рекомендуемым для Celery решением, позволяющим раскрыть весь потенциал Celery и гарантирующим надежное сохранение поступивших на обработку сообщений в очередях типа durable[1]. В этом случае все поступившие в такие очереди сообщения-задания сохраняются на диск и переживают возможные сбои системы.

Redis используется как *промежуточная* СУБД — называемая Backend[2] в терминологии Celery, для сохранения результата работы *исполнителей* и возврата этих результатов *клиенту.* После получения *клиентом* результатов работы *исполнителей* эти данные удаляются из промежуточной СУБД через определенное время. Выбор Redis обусловлен тем, что это резидентная СУБД, хранящая все свои данные в оперативной памяти и позволяющая быстро запомнить и отдать клиенту результат без задержек, связанных с дисковым вводом-выводом.

В задаче распознавания лиц в Redis сохраняются построенные биометрические шаблоны, в задаче анализа ЭЭГ — информация о спектральной плотности мощности временного окна в анализируемом сигнале.

Система управления базами данных PostgreSQL используется, если требуется долговременное сохранение полученных результатов. Конкретное решение зависит от типа сохраняемых данных — здесь PostgreSQL предоставляет разработчику богатый выбор.

В задаче распознавания лиц в СУБД PostgreSQL сохраняются: исходные фотографии, поступившие на обработку; вырезанные фрагменты исходных фотографий с лицами людей, если они были детектированы на исходных фотографиях; построенные по этим лицам биометрические шаблоны. Таблицы в PostgreSQL, хранящие фотографии в blob-полях типа *bytea*, были секционированы на 1024 партиций для эффективного сохранения большого количества файлов-фотографий. В качестве первичного ключа в таблицах с фотографиями используется хеш-сумма от содержимого файла — это позволяет избежать повторного сохранения одних и тех же фотографий и решает проблему быстрого доступа к нужному файлу через соответствующий SQL-запрос.

[1] Англ. *durable* – надежный, стойкий.
[2] Англ. *backend* – внутренний, фоновый, серверный.

В качестве основы для разработки REST API интерфейса был выбран python-фреймворк Flask [41], работающий в связке с Gunicorn [42], — Python WSGI http-сервером, реализующим стандарт WSGI взаимодействия между python-программой, http-сервером и http прокси-сервером Nginx [43] с возможностями балансировщика нагрузки.

В соответствии с проведенным компанией JetBrains в 2018 г. опросом python-разработчиков [44], Django и Flask являются самыми используемыми фреймворками для разработки веб-приложений. Выбор Flask, а не Django в качестве основы для создания веб-приложения для распределенной вычислительной системы был продиктован следующими соображениями. Фреймворк Django предоставляет разработчику богатые функциональные возможности "из коробки", однако он навязывает определенные рамки проектирования веб-приложения, выйти за которые достаточно трудно. Фреймворк Flask, наоборот, является достаточно минималистичным средством, обладающим достаточными функциональными возможностями для разработки масштабируемых веб-приложений. При необходимости базовые функциональные возможности Flask расширяются за счет подключения дополнительных модулей.

Принцип работы

Предлагаемая модель архитектуры распределенной вычислительной системы, как показано на рис. 1, работает следующим образом.

Взаимодействие клиента и вычислительной системы происходит посредством REST API через отправку *клиентом* (1) запросов *диспетчеру* (2) по протоколу http. Возможны два способа взаимодействия *клиента* и *диспетчера* — *синхронный* и *асинхронный.*

При *синхронной* отправке задания *клиент* дожидается его выполнения заранее определяемый *период времени* и после этого завершает выполнение http-запроса с ошибкой, если в течение этого периода времени он не получил ответа от *диспетчера.*

В случае *асинхронной* отправки задания *клиент* в ответ от *диспетчера* получает *идентификатор*, который Celery присваивает данному *заданию.* После получения *идентификатора клиент* завершает выполнение http-запроса. Для получения результата клиент должен обратиться за результатом выполне-

ния задания позже, предъявив при запросе *идентификатор задания*.

Рассмотрим подробнее синхронный и асинхронный режимы работы.

Синхронный режим

Клиент (1) направляет запрос, содержащий задание с исходными данными, диспетчеру (2) и *ожидает ответ* с результатом выполнения данного задания — соединение с диспетчером не закрывается. При этом время ожидания клиента (1) ответа от диспетчера (2) регулируется соответствующим таймаутом, по истечении которого клиенту (1) будет сообщено, что выполнение его задания закончилось неудачей. Диспетчер (2) помещает полученное задание вместе с его исходными данными в очередь брокера сообщений (3). Свободный исполнитель (4) из множества доступных исполнителей забирает задачу из очереди брокера сообщений (3) для ее выполнения. Назначенный исполнитель (4), после выполнения полученного задания сохраняет результат его выполнения в промежуточной[1] базе данных (5), при необходимости сохраняет результат выполнения задания в долговременной базе данных[2] (6) и сообщает о том, что он выполнил полученное задание путем передачи соответствующего сообщения брокеру сообщений (3). После этого назначенный исполнитель становится свободным исполнителем и может выполнять следующую задачу. Диспетчер (2) оповещается о том, что направленное им задание выполнено через получение соответствующего сообщения от брокера сообщений (3). Диспетчер (2) получает результат выполнения задания из промежуточной базы данных (5) и сообщает этот результат клиенту (1). Клиент (1) получает ответ от диспетчера (2) и завершает выполнение запроса.

Синхронный способ взаимодействия проще для клиента, так как нет необходимости каким-либо специальным образом адаптировать программное обеспечение на клиенте, чего нельзя сказать об асинхронном способе работы.

К недостаткам синхронного способа можно отнести то обстоятельство, что необходимо экспериментальным путем подбирать скорость поступления заданий от клиента, чтобы исполнители успевали их обрабатывать. Это возможно лишь при нагрузке на исполнителей, не превышающей 100 % их допустимой производительности. Соответственно, часть исполнителей может использоваться не полностью и часть вычислительных ресурсов будет простаивать.

При превышении предельно допустимой нагрузки на исполнителей задания, поступающие от клиента, будут не успевать обрабатываться исполнителями и их выполнение будет завершаться неудачей.

Асинхронный режим

В случае асинхронного режима работы взаимодействие клиента и вычислительной системы изменится следующим образом.

Как показано на рис. 1, клиент (1) направляет запрос, содержащий задание с исходными данными, диспетчеру (2). Диспетчер (2) *сообщает* клиенту (1) *идентификатор* полученного от него задания. Клиент (1) получает идентификатор, завершает выполнение запроса и закрывает соединение с диспетчером. Диспетчер (2) помещает полученное задание вместе с его исходными данными[3] в очередь брокера сообщений (3) и "забывает" про это задание. Задание — в порядке живой очереди или в соответствии с назначенными приоритетами, ожидает в очереди (3), когда его извлечет для выполнения освободившийся исполнитель. Дальнейшая обработка освободившимся исполнителем (4) задания из очереди (3) происходит как в случае синхронного режима.

Клиент (1) через какое-то время может обратиться к диспетчеру (2) с запросом о состоянии отправленного им на обработку задания, сообщив при этом *идентификатор задания*. В качестве ответа клиент (1) получит от диспетчера (2) информацию о текущем состоянии задания и, если задание выполнилось успешно, получит результат выполнения задания.

Асинхронный режим работы сложнее для клиента, так как необходимо на клиенте адаптировать программное обеспечение под асинхронный способ работы — добавить режим ожидания выполнения заданий, предусмотреть хранение идентификаторов заданий, реализовать возможность неоднократных запросов статуса выполнения заданий.

К преимуществам асинхронного способа можно отнести следующее обстоятельство — все поступающие задания упорядочиваются в очередь, теоретически — очередь неограниченного размера. Соответственно, пока есть задания в очереди — все доступные исполнители будут их выполнять. Таким образом, возможно эффективно использовать все выделенные вычислительные ресурсы, и никакие исполнители при постоянно заполненной очереди заданий не будут простаивать.

Программные компоненты диспетчера

Рассмотрим более подробно принцип работы программных компонентов, из которых состоит диспетчер.

На рис. 2 изображена схема взаимодействия программных компонентов диспетчера (2). Запрос от клиента (1) поступает на http-прокси-сервер Nginx [43], который, в свою очередь перенаправляет его на WSGI http-сервер Gunicorn [42]. Gunicorn перенаправляет поступивший запрос на один из экземпляров своих процессов-исполнителей — веб-сервисы, созданные на основе фреймворков Flask [41] и Celery [19]. Экземпляры веб-сервисов взаимодействуют с брокером сообщений (3) и резидентной базой данных (5).

Для запуска Gunicorn используется Supervisor[4] — модуль для управления сервисами, написанный на Python, который позволяет следить за состоянием

[1] Англ. Backend.
[2] Англ. Persistent Database.

[3] Когда исходные данные достаточно большого размера, то рекомендуется, чтобы они сохранялись не в брокере сообщений, а во вспомогательной СУБД, в качестве которой может выступать Redis или PostgreSQL.
[4] http://supervisord.org/ (дата обращения: 15.05.2021).

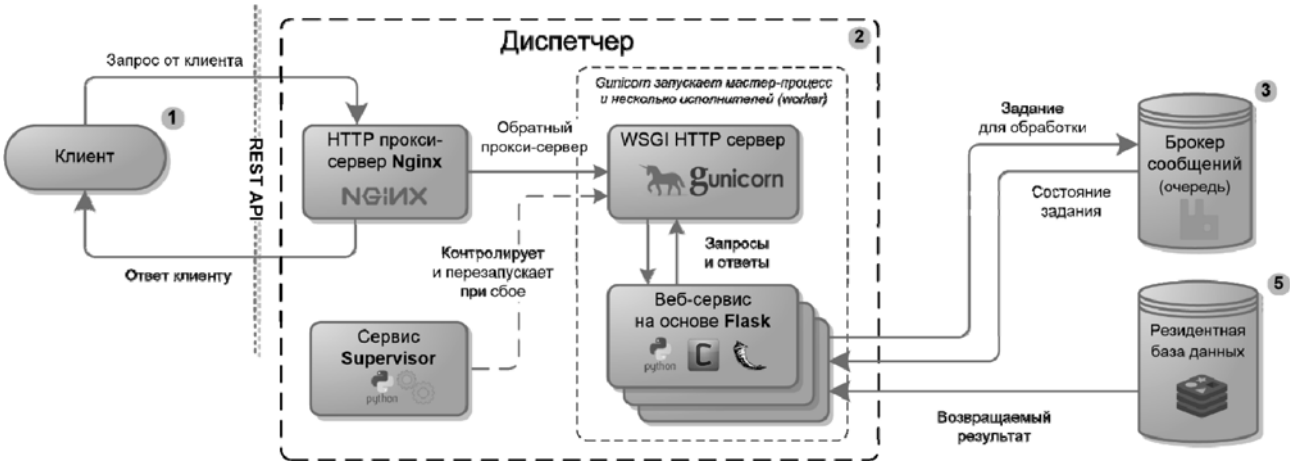


Рис. 2. Программные компоненты диспетчера

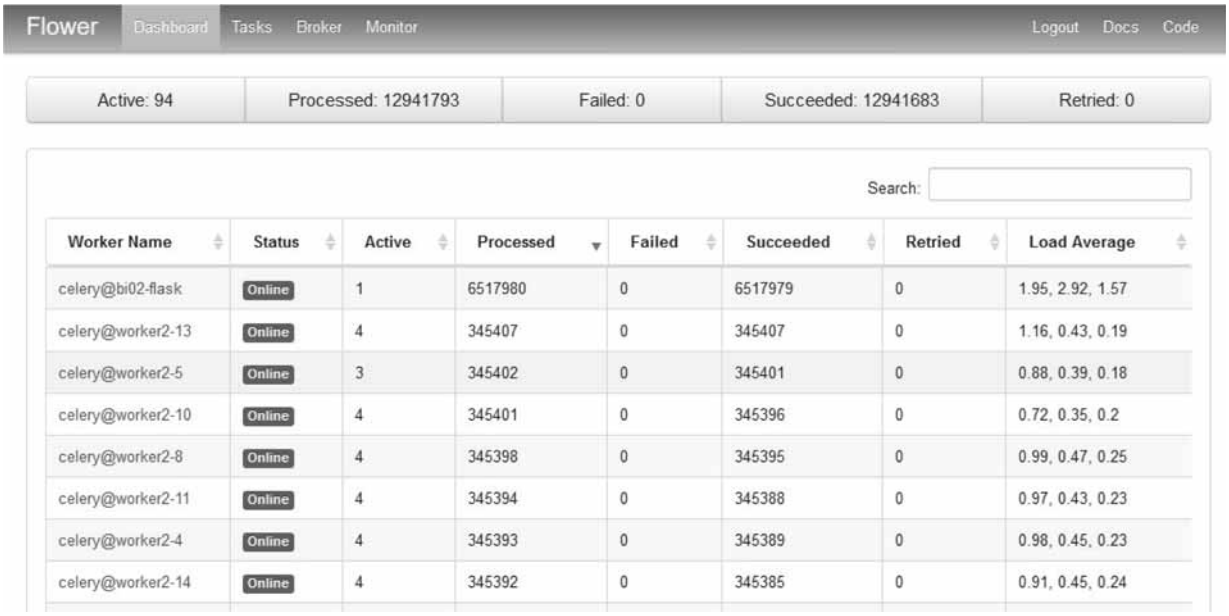


Рис. 3. Информационная панель веб-инструмента Flower на основе продуктивной вычислительной системы автора

запущенных под его управлением сервисов и автоматически перезапускать их в случае сбоя.

При запуске Gunicorn создает мастер-процесс и определенное число экземпляров процессов-исполнителей. Число создаваемых процессов-исполнителей рекомендуется[1] делать равным (2 × число CPU-ядер) + 1. Например, для сервера с 8 ядрами число процессов-исполнителей будет равно 17.

Мониторинг и управление

При решении задачи распределенных вычислений возникают следующие вопросы, связанные с мониторингом и контролем за выполнением заданий:

- формирование общей очереди *заданий*;
- выполнение *заданий* каждым из *исполнителей*;
- получение и сохранение результатов выполнения *заданий*, получаемых от каждого *исполнителя*;
- синхронизация выполнения *заданий*, зависимых друг от друга;
- обмен данными между зависимыми *заданиями.*

Необходимо следить:

- за равномерной нагрузкой на *исполнителей*;
- чтобы каждый из *исполнителей* не простаивал и был нагружен по возможности на 100 % своей вычислительной мощности;
- чтобы каждый *исполнитель* не получал *заданий* больше, чем он может выполнить.

Фреймворк Celery позволяет эффективно решать все эти вопросы. Он имеет встроенный интерактивный веб-инструментарий, называемый Flower [45], для мониторинга и управлением задачами Celery в реальном времени. Он запускается и работает как отдельное веб-приложение, позволяет следить и управлять поступлением и выполнением задач.

На рис. 3 приведена информационная панель Flower, на которой выведена информация о состоянии текущей нагрузки на исполнителей. В левом верхнем углу указано Active: 94 — число активных

[3] https://docs.gunicorn.org/en/stable/design.html#how-many-workers (дата обращения: 15.05.2021).

исполнителей, т. е. число задействованных *исполнителей*, выполняющих задания в настоящий момент времени. Processed: 12 941 793 — общее число задач, выполненных исполнителями с момента запуска сервера, в том числе успешно, — показатель Succeeded и со сбоем — показатель Failed.

В графе Worker Name указано имя сервера, на котором может выполняться несколько исполнителей, в графе Active — число исполнителей, выполняющихся в настоящий момент времени на этом сервере из графы Worker Name, в графе Processed — число заданий, выполненных всеми исполнителями данного сервера.

Celery позволяет упорядочивать задания *в цепочки заданий — chained tasks* в том случае, когда выполняемые задания зависят друг от друга. Поступающим заданиям можно назначать *приоритеты*, связывать эти задания с определенными очередями и за этими очередями закреплять определенных исполнителей. Нагрузка на доступных исполнителей может распределяться равномерно или в соответствии с задаваемой конфигурацией — так называемым *task routing.*

Таким образом, в качестве исполнителей можно использовать серверы с различной вычислительной мощностью, имеющие или не имеющие GPU. Задачи, эффект от выполнения которых на GPU будет максимальным, можно помещать в *специализированные очереди* и направлять на исполнение конкретным исполнителям, задействующим в своей работе GPU.

Celery как система массового обслуживания

Фактически, полученная вычислительная система представляет собой *систему массового обслуживания.* В этом случае в терминологии систем массового обслуживания, *исполнители* являются *каналами обслуживания*, а поступающие на обработку *задания — требованиями.*

Напомним, что системы массового обслуживания можно классифицировать на основе следующих признаков:

- с очередями (с ожиданием) / без очередей (без ожидания);
- без потерь / с частичными потерями / с потерями;
- с ограниченной длинной очереди / с ограниченным временем требования / с ограниченным временем ожидания;
- с одной очередью / со многими очередями;
- с одним каналом обслуживания / с конечным числом каналов обслуживания / с бесконечным числом каналов обслуживания;
- с конечным числом требований / с бесконечным числом требований.

Celery позволяет задавать *предельное время выполнения* заданий, *предельное время ожидания* задания в очереди. Изменяя число *исполнителей,* можно изменять число каналов обслуживания. Возможно задавать *предельный размер очереди* при использовании RabbitMQ в качестве брокера сообщений.

Архитектурные варианты

При необходимости возможно внесение некоторых изменений в предложенную архитектуру.

Вернемся к рис. 1. Возможно отказаться от использования долговременной базы данных (6) в случае, если долговременное сохранение полученных данных не требуется. В этом случае на *клиента* (1) возлагается задача по сохранению результатов выполненных заданий.

Возможно отказаться от использования промежуточной базы данных (5) в случае, если задачи требуют от исполнителей (4) только выполнения каких-то действий, например, отправки электронных писем, и клиенту (1) важен статус их выполнения (успешно/не успешно), но не важны данные, образовавшиеся в результате их выполнения, данные в результате их выполнения не образуются или же данные, полученные в результате их выполнения, не возвращаются клиенту (1), а сохраняются в долговременной базе данных (6).

В качестве брокера сообщений (3) можно использовать Redis вместо RabbitMQ. В этом случае сокращается число компонентов, из которых состоит вычислительная система, но теряются некоторые преимущества, связанные с использованием RabbitMQ.

Полученные результаты

В результате организации распределенной вычислительной системы в соответствии с предлагаемой архитектурой в целях распределенного *построения биометрических шаблонов* были получены следующие результаты.

Количество обрабатываемых изображений выросло до 10 млн фотоизображений в сутки при задействовании 20 серверов с 8 vCPU[1] каждый. За счет сохранения результатов обработки в транзакционной реляционной СУБД PostgreSQL и брокера сообщений RabbitMQ с очередями типа durable вычислительная система стала устойчива к сбоям оборудования и потерям связи между компонентами системы.

Время обработки данных биомедицинских сигналов ЭЭГ от 40 испытуемых сократилось до 30 мин, стал возможным быстрый пересчет показателей в случае необходимости внесения изменений в алгоритмы их обработки.

Вычислительные ресурсы "частного облака" на базе инфраструктуры OpenStack [46] стали расходоваться более экономно — при необходимости увеличения числа обрабатываемых фотографий или сигналов необходимо лишь добавление новых исполнителей к вычислительной системе.

В инфраструктуре OpenStack добавление новых сервера-исполнителей происходит через копирование имеющихся виртуальных серверов с исполнителями в несколько щелчков мышью. За одну операцию можно сразу создать необходимое число виртуальных серверов-исполнителей. Новые серверы-исполнители не требуют никакой дополнительной настройки — после создания запускаются, получают по DHCP внутренние IP-адреса, подключаются к инфраструк-

[1] Сокр. от англ. *virtual CPU* – виртуальные CPU, ядра CPU физических серверов, выделяемые для виртуальных серверов.

type Celery и сразу начинают выполнять поступающие задания.

Если нагрузка на вычислительную систему снижается, то можно остановить или удалить неиспользуемые серверы-исполнители — в этом случае высвобожденные облачные вычислительные ресурсы можно использовать для других нужд.

В качестве основной конфигурации для одного виртуального сервера, используемого в качестве исполнителя для построения биометрических шаблонов, используется сервер с 8 vCPU, 8 ГБ ОЗУ, 20 ГБ для системного диска. Несмотря на то, что OpenStack позволяет создавать виртуальные сервера с большим числом vCPU и ОЗУ, данная конфигурация позволяет выполнять автоматическую миграцию виртуальных серверов между физическими серверами в случае сбоев.

Фреймворк машинного обучения, используемый для построения биометрических шаблонов, вызывается из Python-программы, но написан на языке программирования C++ и может задействовать для построения одного биометрического шаблона одновременно несколько ядер CPU. Механизмы GIL Python [25] на эффективное использование нескольких ядер CPU данным фреймворком не оказывают никакого влияния.

Для эффективного задействования всех vCPU, доступных на одном сервере-исполнителе, экспериментальным образом выбран вариант, когда на одном сервере-исполнителе одновременно запускаются четыре процесса-исполнителя, занимающихся построением биометрических шаблонов. Каждый из четырех процессов-исполнителей в своей работе задействует две выделенных для него vCPU на 100 %. Таким образом, вычислительные ресурсы одного сервера-исполнителя используются на 100 %.

Заключение

Для достижения поставленной цели — организации эффективной обработки большого числа ресурсоемких задач в распределенной вычислительной системе, в ходе исследования была спроектирована и реализована распределенная вычислительная система, основанная на фреймоворке для распределенной обработки заданий Celery, брокера сообщений и систем управления базами данных с открытым исходным кодом.

Полученное решение рассмотрено с точки зрения теории массового обслуживания, проанализированы особенности его использования, связанные с мониторингом и управлением.

Для созданного решения необходимо развертывание системы виртуальных и (или) физических серверов, которая будет решать вычислительные задачи. После решения вычислительных задач или снижения требований к числу вычислительных ресурсов, необходимых для их выполнения, серверы из этой системы могут быть удалены и высвобожденные вычислительные ресурсы могут быть использованы в других целях.

Эффективность предлагаемого решения подтверждена его практическим применением в продуктивной среде в задаче распознавания лиц и анализа биомедицинских сигналов.

Созданное решение может быть рекомендовано к использованию как в частной, так и в публичной облачной инфраструктурах.

Исследование выполнено при поддержке гранта РФФИ № 19-29-01156 МК.

Список литературы

1. **Bradford B., Yesberg J. A., Jackson J.** et al. Live Facial Recognition: Trust and Legitimacy as Predictors of Public Support for Police Use of New Technology // Br. J. Criminol. — 2020. — Vol. 60, No. 6. — P. 1502—1522.

2. **Kotsoglou K. N., Oswald M.** The long arm of the algorithm? Automated Facial Recognition as evidence and trigger for police intervention // Forensic Sci. Int. Synerg. Elsevier B. V. — 2020. — Vol. 2. — P. 86—89.

3. **Fussey P., Murray D.** Independet Report on the London Metropolitan Police Service's Trial of Live Facial Recognition Technology. The Human Rights, Big Data and Technology Project. Colchester: Human Rights Centre, University of Essex, 2019. — 128 p.

4. **Grother P. J., Ngan M., Hanaoka K. K.** Ongoing Face Recognition Vendor Test (FRVT). Part 2: Identification. Gaithersburg, MD, 2018. — 145 p.

5. **Grother P. J., Ngan M., Hanaoka K. K.** Face Recognition Vendor Test (FRVT). Part 2: Identification. National Institute of Standards and Technology — NISTIR 8238. 2019. 183 p.

6. **Булгакова Я. В., Булгаков Д. Ю., Туровский Я. А.** и др. Исследование ЭЭГ-коррелятов субъективных временных шкал // Нейронаука для медицины и психологии: XVI Междунар. междисципл. конгресс. Труды Конгресса. Судак, 2020. — С. 119—120.

7. **Булгаков Д. Ю.** Об особенностях создания биометрических систем распознавания лиц на основе нейросетевых моделей // Стратегическое развитие системы МВД России: состояние, тенденции, перспективы: Сборник статей международной научно-практической конференции / под ред. В.О Лапина. — М.: Академия управления МВД России, 2019. — С. 67—68.

8. **Булгаков Д. Ю., Нарушев И. Р.** О некоторых особенностях решения задачи поиска k-ближайших соседей в пространствах высокой размерности // Стратегическое развитие системы МВД России: состояние, тенденции, перспективы: Сборник статей международной научно-практической конференции / под ред. И. Г. Чистобородова, А. Л. Ситковского, В. О. Лапина. — М.: Академия управления МВД России, 2020. — С. 133—137.

9. **Nguyen G., Dlugolinsky S., Bobák M.** et al. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey // Artif. Intell. Rev. Springer Netherlands. — 2019. — Vol. 52, No. 1. — P. 77—124.

10. **NVIDIA** Corporation. Popular GPU-accelerated Applications Catalog. 2020. 73 с.

11. **Булгаков Д. Ю.** Современные подходы к тестированию систем биометрической идентификации по изображению лица // Искусственный интеллект (большие данные) на службе полиции: Сборник статей международной научно-практической конференции. — М.: Академия управления МВД России, 2020. — С. 45—51.

12. **Булгаков Д. Ю., Булгакова Я. В., Каратыгин Н. А.** Современное свободное программное обеспечение для анализа и обработки электроэнцефалограмм: возможности и выбор // Программная инженерия. — 2020. — Т. 11, № 4. — С. 205—212.

13. **Apache** Software Foundation. Apache Hadoop: Project develops open-source software for reliable, scalable, distributed computing. 2020. URL: https://hadoop.apache.org/ (дата обращения: 30.07.2020).

14. **Apache** Software Foundation. Apache Spark: Unified Analytics Engine for Big Data. 2020. URL: https://spark.apache.org/ (дата обращения: 30.07.2020).

15. **JetBrains s.r.o.** The State of Developer Ecosystem in 2020 Infographic | JetBrains: Developer Tools for Professionals and Teams. 2020. URL: https://www.jetbrains.com/lp/devecosystem-2020/ (дата обращения: 14.07.2020).
16. **Crawford C., Montoya A., O'Connell M. M. P.** 2018 Kaggle ML & DS Survey. The most comprehensive dataset available on the state of ML and Data Science. 2018. URL: https://www.kaggle.com/kaggle/kaggle-survey-2018 (дата обращения: 14.11.2020).
17. **Stigler S., Burdack M.** A Practical Approach of Different Programming Techniques to Implement a Real-time Application using Django // ATHENS J. Sci. — 2020. — Vol. 7, No 1. — P. 43—66.
18. **Django** Software Foundation. Django: The Web framework for perfectionists with deadlines URL: https://www.djangoproject.com/ (дата обращения: 05.11.2020).
19. **Solem A.** Celery: Distributed Task Queue. 2020. URL: https://docs.celeryproject.org/en/stable/ (дата обращения: 15.07.2020).
20. **Mayer R., Jacobsen H.-A.** Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools. Munich, Germany, 2019. — 35 p.
21. **Python** Software Foundation. asyncio: Asynchronous I/O — Python 3.8 documentation. 2020. URL: https://docs.python.org/3/library/asyncio.html (дата обращения: 15.07.2020).
22. **Python** Software Foundation. concurrent.futures: Launching parallel tasks — Python 3.8 documentation. 2020. URL: https://docs.python.org/3/library/concurrent.futures.html (дата обращения: 15.07.2020).
23. **Python** Software Foundation. threading: Thread-based parallelism — Python 3.8 documentation. 2020. URL: https://docs.python.org/3/library/threading.html (дата обращения: 15.07.2020).
24. **Python** Software Foundation. multiprocessing: Process-based parallelism — Python 3.8 documentation. 2020. URL: https://docs.python.org/3/library/multiprocessing.html (дата обращения: 15.07.2020).
25. **Python** Software Foundation. Global interpreter lock (GIL) — Python 3.8 documentation. 2020. URL: https://docs.python.org/3/glossary.html#term-global-interpreter-lock (дата обращения: 15.07.2020).
26. **Matsumoto Y.** Global VM Lock — Documentation for Ruby 2.7.0 — Threading. 2020. URL: https://docs.ruby-lang.org/en/2.7.0/doc/extension_rdoc.html#label-Threading (дата обращения: 08.10.2020).
27. **OpenJS** Foundation. Worker threads: Node.js v14.13.1 Documentation. 2020. URL: https://nodejs.org/docs/latest-v14.x/api/worker_threads.html (дата обращения: 08.10.2020).
28. **Python** Software Foundation. Jython: Java implementation of Python. 2020. URL: https://www.jython.org/ (дата обращения: 30.07.2020).
29. **NET** Foundation. IronPython: the Python programming language for the.NET Framework. 2020. URL: https://ironpython.net/ (дата обращения: 30.07.2020).
30. **NumPy** project. NumPy: The fundamental package for scientific computing with Python. 2020. URL: https://numpy.org/ (дата обращения: 31.07.2020).
31. **Nikolic B.** Python Global Interpreter Lock (GIL) + numerical data processing. 2019. URL: http://www.bnikolic.co.uk/blog/python-numerical-gil.html (дата обращения: 27.07.2020).
32. **Python** Software Foundation. queue: A synchronized queue class — Python 3.8 documentation. 2020. URL: https://docs.python.org/3/library/queue.html (дата обращения: 27.07.2020).
33. **Python** Software Foundation. heapq: Heapq queue algorithm — Python 3.8 documentation. 2020. URL: https://docs.python.org/3/library/heapq.html (дата обращения: 27.07.2020).
34. **Driessen V.** RQ: Simple job queues for Python. 2020. URL: http://python-rq.org/ (дата обращения: 15.07.2020).
35. **Popa B.** Dramatiq: Background tasks. 2020. URL: https://dramatiq.io/ (дата обращения: 15.07.2020).
36. **Makai M.** Full Stack Python: Task Queues. 2020. URL: https://www.fullstackpython.com/task-queues.html (дата обращения: 15.07.2020).
37. **Redis Labs.** Redis: open source, in-memory data structure store, used as a database, cache and message broker. 2020. URL: https://redis.io/ (дата обращения: 16.07.2020).
38. **CLEARTYPE** SRL. Dramatiq 1.9.0 documentation. Motivation. 2020. URL: https://dramatiq.io/motivation.html (дата обращения: 15.07.2020).
39. **VMware Inc.** RabbitMQ: open source message broker. 2020. URL: https://www.rabbitmq.com/ (дата обращения: 16.07.2020).
40. **The PostgreSQL** Global Development Group. PostgreSQL: The world's most advanced open source database. 2020. URL: https://www.postgresql.org/ (дата обращения: 16.07.2020).
41. **Flask:** Lightweight WSGI web application framework — The Pallets Projects. 2020. URL: https://palletsprojects.com/p/flask/ (дата обращения: 31.07.2020).
42. **Gunicorn:** Python WSGI HTTP Server for UNIX. URL: https://gunicorn.org/ (дата обращения: 05.11.2020).
43. **F5 Inc.** NGINX: High Performance Load Balancer, Web Server, & Reverse Proxy. 2020. URL: https://www.nginx.com/ (дата обращения: 05.11.2020).
44. **JetBrains s.r.o.** Python Developers Survey 2018 Results. 2018. URL: https://www.jetbrains.com/research/python-developers-survey-2018/ (дата обращения: 31.07.2020).
45. **Movsisyan M.** Flower: Celery monitoring tool — Flower 1.0.0 documentation. 2016. URL: https://flower.readthedocs.io/en/latest/ (дата обращения: 08.10.2020).
46. **Open** Infrastructure Foundation. OpenStack: Open Source Cloud Computing Infrastructure. 2020. URL: https://www.openstack.org/ (дата обращения: 07.11.2020).

Using Distributed Cloud Computing to Solve Resource-Intensive Tasks

D. Yu. Bulgakov, dbulgakov7@yandex.ru, Management Academy of the Ministry of the Interior of Russia, Moscow, 125993, Russian Federation

Corresponding author:

Bulgakov Dmitry Yu., Adjunct, Management Academy of the Ministry of the Interior of Russia, Moscow, 125993, Russian Federation
E-mail: dbulgakov7@yandex.ru

Received on November 08, 2020
Accepted on May 19, 2021

A method for solving resource-intensive tasks that actively use the CPU, when the computing resources of one server become insufficient, is proposed. The need to solve this class of problems arises when using various machine learning models in a production environment, as well as in scientific research. Cloud computing allows you to organize distributed task processing on virtual servers that are easy to create, maintain, and replicate. An approach based on the use of free software implemented in the Python programming language is justified and proposed. The resulting solution is considered from the point of view of the theory of queuing. The effect of the proposed approach in solving problems of face recognition and analysis of biomedical signals is described.

Keywords: *cloud computing, distributed computing, face recognition, Queuing theory, biomedical signals, EEG analysis, Python, Celery, Flask, RabbitMQ*

For citation:

Bulgakov D. Yu. Using Distributed Cloud Computing to Solve Resource-Intensive Tasks, *Programmnaya Ingeneria,* 2021, vol. 12, no. 5, pp. 233—245.

DOI: 10.17587/prin.12.233-245

References

1. **Bradford B., Yesberg J. A., Jackson J.** et al. Live Facial Recognition: Trust and Legitimacy as Predictors of Public Support for Police Use of New Technology, *Br. J. Criminol*, 2020, vol. 60, no. 6, pp. 1502—1522.

2. **Kotsoglou K. N., Oswald M.** The long arm of the algorithm? Automated Facial Recognition as evidence and trigger for police intervention, *Forensic Sci. Int. Synerg,* Elsevier B. V., 2020, vol. 2, pp. 86—89.

3. **Fussey P., Murray D.** Independet Report on the London Metropolitan Police Service's Trial of Live Facial Recognition Technology. *The Human Rights, Big Data and Technology Project*, Colchester: Human Rights Centre, University of Essex, 2019, 128 p.

4. **Grother P. J., Ngan M., Hanaoka K. K.** Ongoing Face Recognition Vendor Test (FRVT). Part 2: Identification, NIST, 2018, 145 p.

5. **Grother P. J., Ngan M., Hanaoka K. K.** Face Recognition Vendor Test (FRVT). Part 2: Identification, NIST, 2019, 183 p.

6. **Bulgakova Y. V., Bulgakov D. Yu., Turovsky Ya. A.** et al. Investigation of EEG Correlates Of Subjective Timescales, *Neuroscience for Medicine and Psychology*, Sudak, LLC MAKS Press, 2020, pp. 119—120 (in Russian).

7. **Bulgakov D. Y.** Features of creating biometric facial recognition systems based on neural network models, *Strategic development of the Russian interior Ministry system: status, trends, and prospects,* Moscow, Management Academy of the Ministry of the Interior of Russia, 2019, pp. 67—68 (in Russian).

8. **Bulgakov D. Y., Narushev I. R.** On Some Features of Solving the Problem of Finding k-nearest Neighbors in High-Dimensional Spaces, *Strategic development of the Russian interior Ministry system: status, trends, and prospects*, Moscow, Management Academy of the Ministry of the Interior of Russia, 2020, pp. 133—137 (in Russian).

9. **Nguyen G., Dlugolinsky S., Bobák M.** et al. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey, *Artif. Intell. Rev.*, Springer Netherlands, 2019, vol. 52, no. 1, pp. 77—124.

10. **NVIDIA Corporation.** Popular GPU-accelerated Applications Catalog, 2020, 73 p.

11. **Bulgakov D. Y.** Modern Approaches to Testing Biometric Identification Systems Based on Facial Images, *Artificial Intelligence (Big Data) in The Service Of The Police*. Moscow, Management Academy of the Ministry of the Interior of Russia, 2020. pp. 45—51 (in Russian).

12. **Bulgakov D. Y., Bulgakova Y. V., Karatygin N. A.** Up-to-date Open-Source Software for the Analysis and Processing of Electroencephalograms: Opportunities and the Choices, *Programmnaya Ingeneria*, 2020, vol. 11, no. 4, pp. 205—212 (in Russian).

13. **Apache** Software Foundation. Apache Hadoop: Project develops open-source software for reliable, scalable, distributed computing, 2020, available at: https://hadoop.apache.org/

14. **Apache** Software Foundation. Apache Spark: Unified Analytics Engine for Big Data, 2020, available at: https://spark.apache.org/

15. **JetBrains s.r.o**. The State of Developer Ecosystem in 2020 Infographic | JetBrains: Developer Tools for Professionals and Teams, 2020, available at: https://www.jetbrains.com/lp/devecosystem-2020/

16. **Crawford C., Montoya A., O'Connell M M. P.** 2018 Kaggle ML & DS Survey. The most comprehensive dataset available on the state of ML and Data Science, 2018, available at: https://www.kaggle.com/kaggle/kaggle-survey-2018

17. **Stigler S., Burdack M.** A Practical Approach of Different Programming Techniques to Implement a Real-time Application using Django, *ATHENS J. Sci.,* 2020, vol. 7, no. 1. pp. 43—66.

18. **Django** Software Foundation. Django: The Web framework for perfectionists with deadlines, available at: https://www.djangoproject.com/

19. **Solem A.** Celery: Distributed Task Queue, 2020, available at: https://docs.celeryproject.org/en/stable/

20. **Mayer R., Jacobsen H.-A.** Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools. Munich, Germany, 2019. ArXiv ID: 1903.11314.

21. **Python** Software Foundation. Asyncio: Asynchronous I/O — Python 3.8 documentation, 2020, available at: https://docs.python.org/3/library/asyncio.html

22. **Python** Software Foundation. concurrent.futures: Launching parallel tasks — Python 3.8 documentation, 2020, available at: https://docs.python.org/3/library/concurrent.futures.html

23. **Python** Software Foundation. threading: Thread-based parallelism — Python 3.8 documentation, 2020, available at: https://docs.python.org/3/library/threading.html

24. **Python** Software Foundation. multiprocessing: Process-based parallelism — Python 3.8 documentation, 2020, available at: https://docs.python.org/3/library/multiprocessing.html

25. **Python** Software Foundation. Global interpreter lock (GIL) — Python 3.8 documentation, 2020, available at: https://docs.python.org/3/glossary.html#term-global-interpreter-lock

26. **Matsumoto Y**. Global VM Lock — Documentation for Ruby 2.7.0 — Threading, 2020, available at: https://docs.ruby-lang.org/en/2.7.0/doc/extension_rdoc.html#label-Threading

27. **OpenJS** Foundation. Worker threads: Node.js v14.13.1 Documentation, 2020, available at: https://nodejs.org/docs/latest-v14.x/api/worker_threads.html

28. **Python** Software Foundation. Jython: Java implementation of Python, 2020, available at: https://www.jython.org/

29. **.NET** Foundation. IronPython: the Python programming language for the.NET Framework, 2020, available at: https://ironpython.net/

30. **NumPy project.** NumPy: The fundamental package for scientific computing with Python, 2020, available at: https://numpy.org/

31. **Nikolic B.** Python Global Interpreter Lock (GIL) + numerical data processing, 2019, available at: http://www.bnikolic.co.uk/blog/python-numerical-gil.html

32. **Python** Software Foundation. queue: A synchronized queue class — Python 3.8 documentation, 2020, available at: https://docs.python.org/3/library/queue.html

33. **Python** Software Foundation. heapq: Heapq queue algorithm — Python 3.8 documentation, 2020, available at: https://docs.python.org/3/library/heapq.html

34. **Driessen V.** RQ: Simple job queues for Python, 2020, available at: http://python-rq.org/

35. **Popa B.** Dramatiq: Background tasks, 2020, available at: https://dramatiq.io/

36. **Makai M**. Full Stack Python: Task Queues, 2020, available at: https://www.fullstackpython.com/task-queues.html

37. **Redis Labs**. Redis: open source, in-memory data structure store, used as a database, cache and message broker, 2020, available at: https://redis.io/

38. **CLEARTYPE SRL**. Dramatiq 1.9.0 documentation. Motivation, 2019, available at: https://dramatiq.io/motivation.html

39. **VMware Inc.** RabbitMQ: open source message broker, 2020, available at: https://www.rabbitmq.com/

40. **The PostgreSQL** Global Development Group. PostgreSQL: The world's most advanced open source database, 2020, available at: https://www.postgresql.org/

41. **Flask:** Lightweight WSGI web application framework — The Pallets Projects, 2020, available at: https://palletsprojects.com/p/flask/

42. **Chesneau B.** Gunicorn: Python WSGI HTTP Server for UNIX, 2020, available at: https://gunicorn.org/

43. **F5 Inc**. NGINX: High Performance Load Balancer, Web Server, & Reverse Proxy, 2020, available at: https://www.nginx.com/

44. **JetBrains s.r.o**. Python Developers Survey 2018 Results, 2018, available at: https://www.jetbrains.com/research/python-developers-survey-2018/

45. **Movsisyan M**. Flower: Celery monitoring tool — Flower 1.0.0 documentation, 2016, available at: https://flower.readthedocs.io/en/latest/

46. **Open Infrastructure Foundation.** OpenStack: Open Source Cloud Computing Infrastructure. 2020, available at: https://www.openstack.org/

УДК 004.421-423 DOI: 10.17587/prin.12.246-259

С. Е. Попов, канд. техн. наук, ст. науч. сотр., popov@ict.sbras.ru,
Р. Ю. Замараев, канд. техн. наук, ст. науч. сотр., **Н. И. Юкина,** канд. техн. наук, науч. сотр.,
О. Л. Гиниятуллина, канд. техн. наук, науч. сотр., **Л. С. Миков,** мл. наук, науч. сотр.,
И. Е. Харлампенков, канд. техн. наук, науч. сотр.,
Е. Л. Счастливцев, д-р техн. наук, проф., зав. лаб., Федеральное государственное бюджетное научное учреждение "Федеральный исследовательский центр информационных и вычислительных технологий", Новосибирск

Программный комплекс для расчета деформаций земной поверхности с использованием спутниковых радарных данных*

Представлено описание программного комплекса для расчетов скоростей смещений и выявления сдвигов земной поверхности над районами интенсивной угледобычи. Комплекс построен на базе микросервисной архитектуры Docker Swarm в интеграции с системой массово-параллельного исполнения заданий Apache Spark как высокоуровневый инструмент для организации вычислений контейнерного типа с оркестрацией аппаратных ресурсов. За счет использования контейнеризации объектов-исполнителей достигнута независимость расчетов как внутри одного пула заданий, так и между различными пулами, инициализированными в многопользовательском режиме.

Применение операторов-обработчиков позволило сохранять промежуточные результаты работ процедур в схемах расчета скоростей смещений и проводить расчеты с различными параметрами.

Объединение технологий Docker Swarm и Apache Spark в одном программном комплексе позволило реализовать идею высокопроизводительной вычислительной системы с использованием малобюджетных аппаратных ресурсов.

Ключевые слова: *программный комплекс, дифференциальная интерферометрия, скорости смещения земной поверхности, метод постоянных отражателей, метод малых базовых линий*

Введение

Активное развитие методов дифференциальной интерферометрии и средств дистанционного зондирования требует создания проблемно-ориентированных программных комплексов для обработки большого объема получаемых данных. При этом зачастую основная ценность космической информации, поступающей при мониторинге земной поверхности, заключается в возможности ее оперативной пред- и постобработки и анализа результатов. Для получения точных и непротиворечивых результатов требуется исходный массив радарных наблюдений, состоящий в среднем из 30 съемок за 30 разных дат. Причем обработка может включать в себя повторные этапы (формирование интерферограмм, расчет когерентности и оценка ее значений сигнал/шум и т. п.) для составления корректного временного стека сцен съемки с последующим расчетом методами *Small Baseline Subset* (SBaS) или *Persistent Scatterers* (PS) [1, 2]. Таким образом, на отдельных стадиях расчетов может возникать резкий спад производительности. Экспериментальные вычисления показывают время от 3 до 5 ч для 12 пар снимков небольшого разрешения (3000×1000 пикселей) для выявления динамики вертикальных смещений с погрешностью разности высот ЦМР (цифровая модель рельефа) не более чем ±3 мм/пиксел.

Вопросам сокращения времени выполнения рабочих процессов и алгоритмов при расчете смещений на базе радарной интерферометрии посвящено большое число работ [3—27]. В настоящее время реализованы различные системы мониторинга, основанные на данных дистанционного зондирования (ДДЗ) Земли [18—27], использующих различные типы и форматы ДДЗ, как мульти- и гиперспектральные, так и радарные данные. Многие из них носят преимущественно информационный характер с набором ретроспективных данных и отчетов и по факту не обеспечивают интерактивной расчетной части для обработки ДДЗ.

В ряде работ [4—10, 12, 16] авторы используют технологию CUDA для улучшения производительности процедуры развертки фазы, построенные на базе различных математических моделей. В частности, рассматриваются: уравнение Пуассона с весовыми

* Исследование выполнено при финансовой поддержке РФФИ и Кемеровской области в рамках научного проекта № 20-47-420002 р_а.

коэффициентами, метод сопряженных градиентов [6, 7]; метод роста регионов и отсечения ветвей [8, 9]; метод компенсации фазового набега [10] и совмещения (корегистрации) радарных снимков. В работе [10] представлен алгоритм генерации нативных (синтетических) радарных данных, симулирующих различные ошибки/шумы принимающей/передающей части космических аппаратов в зависимости от наблюдаемой земной поверхности, в целях выработки и ускорения программных методов коррекции.

Наряду с GPU имплементацией существуют программные реализации алгоритмов обработки радарных данных на базе параллельных вычислений на CPU. В работе [13] предложена интеграция технологии параллельных вычислений MPI с системой комплексной обработки радарных данных Doris (*Delft object-oriented radar interferometric software*) [14]. Рассмотрены основные этапы обработки InSAR/PS-InSAR-данных, включая наиболее ресурсоемкие (корегистрация, формирование интерферограммы и развертка фазы). Предложена стратегия распараллеливания декомпозиции главного/подчиненного изображений с большим числом сегментов и частичным перекрытием границ, обеспечивающих минимальное межпроцессорное взаимодействие.

Преимуществом подходов, показанных в работах [4—19], является алгоритмизация решения задач пре- и постобработки, которые легко переносятся на параллельные вычисления, с применением уже широко известных процедур, реализованных в библиотеках для GPU и CPU. Так, в упомянутых выше работах за основу взяты элементы пакетов CuFFT, CuBLASS, CuSPARSE, PBLASS, FFTW, ScaLAPACK [15, 17].

В последнее время для стандартного алгоритмического аппарата DInSAR получил широкое распространение метод интерферометрии малых базовых линий SBaS. Метод SBaS основан на совместном использовании длинных временных серий радарных изображений одной территории, полученных в повторяющейся геометрии съемки и хронологически упорядоченных по времени съемки. Метод SbaS относят к одной из наиболее ресурсоемких технологий обработки радарных данных [18, 19]. Так, например, на начальном этапе формируются интерферограммы на базе комплексного перемножения всех возможных пар главного/подчиненного изображений. Затем для каждой парной интерферограммы проводится развертка фазы в целях получения полного смещения в направлении на спутник. При проведении развертки чаще всего сначала применяют алгоритм потока минимальной стоимости, а затем алгоритм роста регионов.

В области комплексной обработки радарных данных наиболее развитым в плане программного обеспечения (ПО), набора функциональных возможностей и доступа к базам данных космоснимков является веб-портал Geohazard Tep [20]. Портал построен на базе облачной архитектуры *Amazon Web Service* (AWS), он содержит широкий пул процессинговых сервисов, ориентированных на решение различных прикладных задач радарной интерферометрии (построение смещений, определение влажности почв, высоты лесного покрова и т. п.). Портал поддерживает платформу облачных вычислений PaaS (*Platform as a Service*). Однако представленные веб-службы портала не дают имплементации именно realtime-обработки в потоковом представлении предметных данных. Сервисы функционируют по модели доступа *On-demand Processing Service*, большая часть из них использует коммерческое ПО (ENVI, SARScape и т. п.).

Реализация и широкое внедрение большого числа программных алгоритмов технологических этапов обработки радарных данных показывают целесообразность применения их совместно в инфраструктуре, предоставляющей массово-параллельное исполнение расчетных заданий, где программный каркас (фреймворк) такой инфраструктуры выступает как интегратор распределенного исполнения программного кода на данных, получаемых в потоковом режиме.

Целью исследования, результаты которого представлены в статье, является разработка программного комплекса расчета смещений земной поверхности с использованием спутниковых радарных данных на базе микросервисной архитектуры с поддержкой массово-параллельного исполнения расчетных заданий.

Для достижения поставленной цели авторы предлагают решение перечисленных далее задач.

1. Разработка информационной модели взаимодействия процессов-идентификаторов в кластерной среде микросервисной архитектуры на основе метаописаний схемы взаимодействия операторов-обработчиков радарных данных.

2. Адаптация методов дифференциальной интерферометрии расчета смещений за счет программных алгоритмов независимого исполнения в средах с массово-параллельными процессами и разделением аппаратных ресурсов в сетевой среде.

3. Создание программных компонентов системы с функциональными возможностям взаимодействия и релевантного обмена информацией в микросервисной среде Docker Swarm и интеграция их с кластерной архитектурой Apache Spark.

Методы исследования

К настоящему времени в основе всех методов расчета скоростей смещений лежат два фундаментальных метода: метод постоянных отражателей (*Persistent Scatterers,* PS) и метод малых базовых линий (SBaS, *Small Baseline Subset*).

Метод PS [2] предназначен для выявления объектов на поверхности, неизменных во времени и по географическим координатам (постоянные отражатели, PS-кандидаты). Постоянные отражатели рассматриваются как точечные объекты типа квазиуголковых отражателей, которые обеспечивают достаточно сильный и устойчивый во времени отраженный сигнал. Выбор пикселей изображения в кандидаты в отражатели осуществляется на основе амплитудной дисперсии, и для каждого PS-кандидата выполняется анализ фазовой стабильности путем вычисления временной когерентности. Проводится развертка фазы для

PS-кандидата, фазовая информация пересчитывается в высоты и вычисляется средняя скорость их изменения.

Метод SBaS [1] позволяет регистрировать деформации земной поверхности и анализировать их пространственно-временные характеристики. Идея метода заключается в поиске таких отражающих площадок, в которых влияние шумов минимально. Для таких пикселей проводится разделение фазы, связанной с деформациями и фазовыми смещениями, выражаемыми атмосферными и ЦМР-артефактами. В частности, погрешности в ЦМР создают фазовые смещения, прямо пропорциональные перпендикулярной базовой линии. Для атмосферных помех принимается, что они плавно меняются в пространстве, но быстро во времени. На этой основе вклад атмосферы и ошибок в ЦМР устанавливается и устраняется, что позволяет выделить составляющую, связанную со смещением площадок во времени.

Представленный в настоящей работе программный комплекс использует в обработке длинные временные серии изображений одной территории. Среднее число изображение выбирается не менее 25. Преимущественно в теплый период времени с целью снизить временную декорреляцию и фазовые помехи, накладываемые плотным снежным покровом.

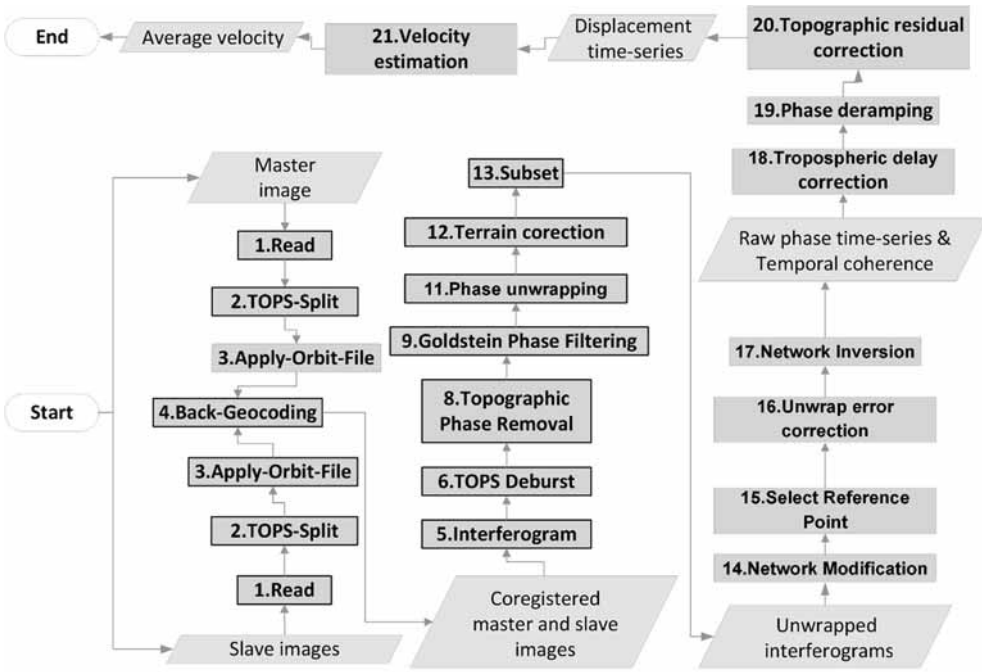


Рис. 1. Алгоритм расчета смещений методом малых базовых линий SBaS

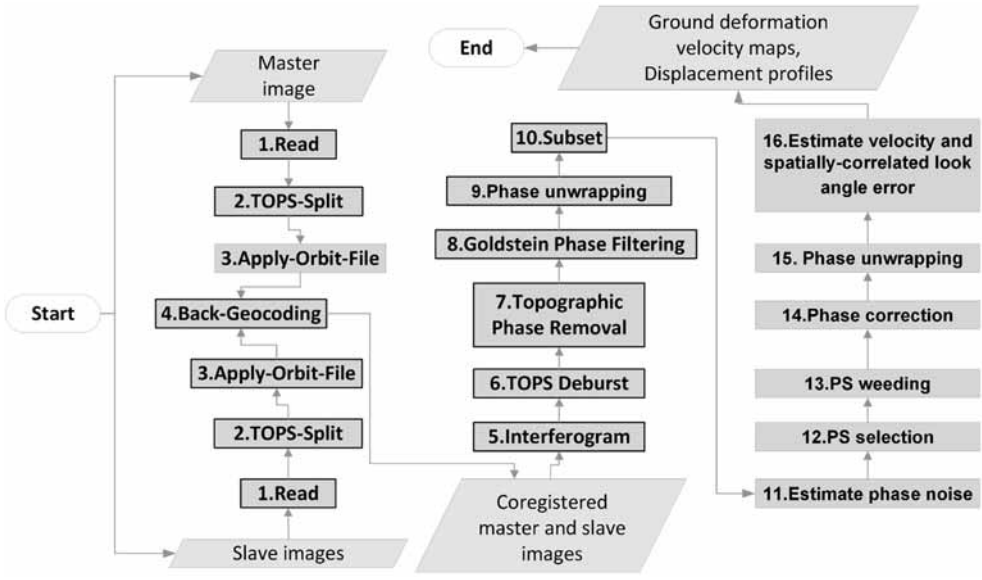


Рис. 2. Алгоритм расчета смещений методом постоянных отражателей PS

В программном комплексе реализованы изображенные на рис. 1 и 2 алгоритмы пред- и постобработки для двух представленных методов.

Прямоугольником выделены процедуры метода, ромбом — основные промежуточные и финальные данные, получаемые на выходе данных процедур.

Подробное описание каждого из этапов пред- и постобработки радарных данных методами PS и SBaS представлено в работе [30]. В каждом из алгоритмов используется главное (Master) и подчиненные (Slave) изображения. Таким образом, формируются стек из N совмещенных с субпиксельной точностью радарных изображений длинной временной серии (в среднем от 60 пар). Здесь субпиксельная точность означает, что любая произвольно взятая точка на Master-изображении будет иметь абсолютно одинаковые географические координаты и на $N-1$ подчиненных изображениях. Именно совмещенные пары используются в дальнейших расчетах всех этапов представленных выше алгоритмов.

Программная реализация вычислительного ядра

Ядро программного комплекса базируется на интеграции двух платформ Apache Spark [28] и Docker [29].

Основная часть алгоритмов в представленных математических моделях интерферометрической обработки преимущественно, использует попиксельную трансляцию в направлениях азимута и дальности на единичных, либо попарно сформированных изображениях (например, совмещение изображений и расчет интерферограмм, развертка фазы, методы фильтрации скользящим окнам) [30]. Более того, представленные итерационные методы в качестве начальных принимают независимые наборы данных. Все это дает возможность применения парадигмы MapReduce, когда для вычисления наборов распределенных задач задействуется большое количество компьютеров (называемых "нодами"), образующих кластер.

Работа MapReduce состоит из двух шагов: Map и Reduce. На Map-шаге происходит предварительная обработка входных данных. Для этого один из компьютеров (называемый главным узлом — *master node*) получает входные данные задачи, делит их на части и передает другим компьютерам (рабочим узлам — *worker node*) для предварительной обработки. На Reduce-шаге происходит свертка предварительно обработанных данных. Главный узел получает ответы от рабочих узлов и на их основе формирует результат — решение задачи, которая изначально формулировалась [30]. Преимущество MapReduce заключается в том, что он позволяет распределенно проводить операции предварительной обработки и свертки. Операции предварительной обработки работают независимо друг от друга и могут проводиться параллельно. Аналогично, множество рабочих узлов может осуществлять свертку, когда все результаты предварительной обработки с одним конкретным значением ключа обрабатываются одним рабочим узлом в один момент времени.

Для реализации массово-параллельного исполнения задач Map и Reduce будет использован фреймворк Apache Spark [28, 30] с открытым исходным кодом для реализации распределенной стратегии вычислений. Это позволит расширить двухуровневую концепцию MapReduce с дисковым хранилищем, используя специализированные примитивы для рекуррентной обработки в оперативной памяти на основе распределенных наборов данных (RDD — *resilient distributed dataset*). RDD представляет собой отказоустойчивый набор элементов, с которыми можно работать параллельно. Инициализация RDD происходит путем логического распараллеливания существующей коллекции данных (в нашем случае, независимых снимков или данных каждого из пикселов внутри снимка) в управляющей программе-драйвере. При этом на получившихся RDD-наборах возможно использование классических функций Map-трансформаций, которые как раз и будут представлять алгоритмизацию математических моделей интерферометрической обработки [28, 30].

Для подключения, управления и инициализации вычислений на кластере будет имплементирована технология контекстного запуска независимых разделяемых исполнителей Spark Context. Spark Context предоставляет методы соединения с кластером Spark и может использоваться для создания RDD, аккумуляторов и широковещательных переменных в этом кластере. Spark Context может быть инициализирован для виртуальных машин языка Java, Python и Scala. Настройка кластера будет осуществляться путем встроенных объектов Spark Config, с передачей параметров распределенной памяти, сетевой адресации узлов кластера, назначения портов, настройки схем выполнения заданий, в том числе определения зависимых библиотек исполняемого кода, контроля интенсивности доступа к дискам распределенной файловой системы HDFS и т. п. [28, 30].

Преимущество выполнения заданий на основе системы массово-параллельного исполнения Apache Spark + YARN по сравнению с Java Multi-Threading или Unix Multiple Commands Run заключается в автоматизированном управлении и распределении ресурсов с помощью нескольких компонентов (рис. 3).

Система микросервисов-контейнеров строится на базе открытого ПО Docker Swarm. Система используется для автоматизации развертывания и управления приложениями в средах с поддержкой контейнеризации, она позволяет "упаковать" приложение со всем его окружением (программными модулями операторов-обработчиков) и зависимостями в микроконтейнер, который обеспечивает полную изоляцию и разделение ресурсов на уровне файловой системы (у каждого контейнера собственная корневая файловая система), на уровне процессорного времени, на уровне сети (каждый контейнер имеет доступ только к привязанному к нему сетевому пространству имен и соответствующим виртуальным сетевым интерфейсам) [29, 30].

Применение системы Docker Swarm позволяет автоматизировать запуск расчетных заданий на нескольких узлах кластера с автоматическим распре-

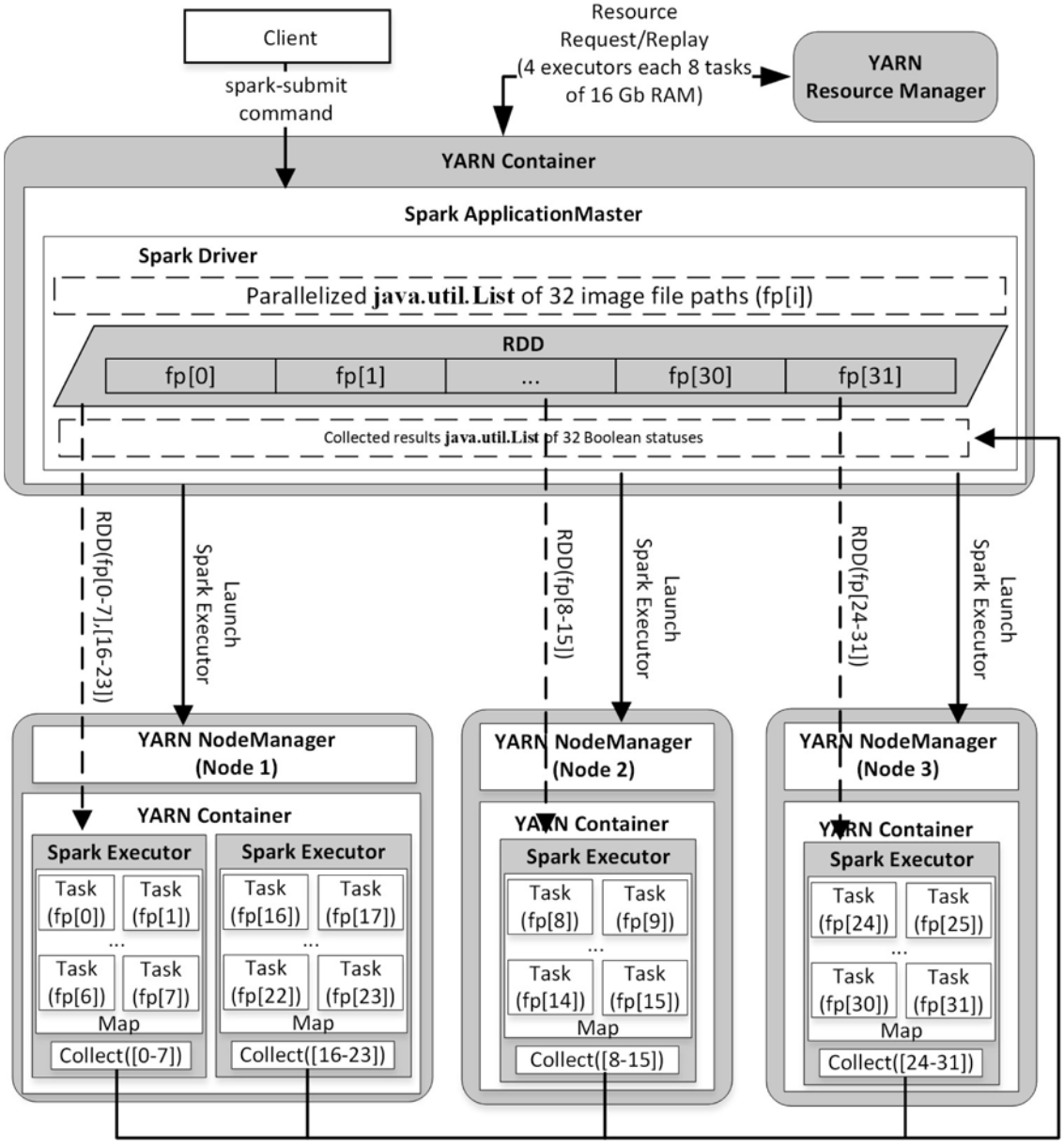


Рис. 3. Диаграмма передачи данных во время запуска алгоритмов на платформе Apache Spark + YARN [30]

делением нагрузки и контролем аппаратных ресурсов, требуемых для задания в системе Apache Spark. Контейнеризация расчетных заданий позволяет организовать многопользовательскую среду за счет независимой работы контейнеров в Docker Swarm. Программный комплекс выполняет каждый расчетный запрос в разделенных виртуальных средах, используя распределенное дисковое пространство Apache HDFS-системы [28—30].

Метаописание операторов-обработчиков

Программный комплекс интегрирует информационную модель метаописания операторов-обработчиков радарных данных.

Модель построена на базе формата данных BEAM-DIMAP (рис. 4), содержит параметры (элементы) и переменные величины (атрибуты), используемые в программных объектах системы при подаче на них информации об изменениях входных значений методов обработки радарных снимков. Модель описывает возможные состояния метаданных, показывает взаимосвязь между вызовом метода программного объекта и элементами/атрибутами схемы метаописаний, имплементируемыми в программном коде.

Элемент <Metadata_Id> содержит информацию о версии формата DIMAP, <Dataset_Id> — уникальный идентификатор данных, включающий название миссии, режима съемки, типа продукта, уровень обработки, поляризации и др., с добавлением префикса оператора-обработчика; <Production> содержит время съемки; <Coordinate_Reference_System> и <Geoposition> — информацию о системе координат и геопозиционировании спутника; <Raster_Dimensions> — размеры изо-

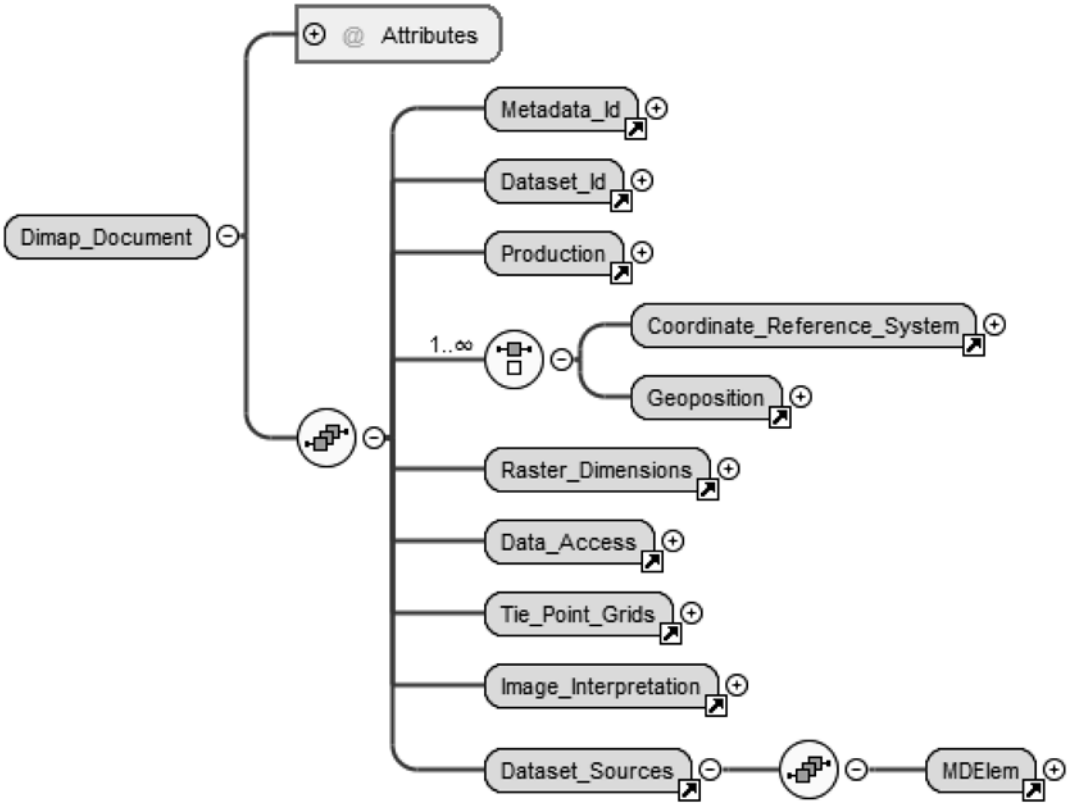


Рис. 4. Общая схема представления метаописания радарных данных

бражения и число полос; <Data_Access> — информацию к служебным файлам изображения (например, данные географических координат, данные углов падения по направлению съемки, значения опорных точек и др.); <Tie_Point_Grids> — точки геопривязки; <Image_Interpretation> — служебную информацию о спектральных полосах изображений.

Ключевым элементом в данной иерархии является элемент <Dataset_Sources>. На его основе строится и описывается цепочка вызовов параметризованных операторов со значениями атрибутов в формате XML (рис 5, 6). Данный элемент содержит атрибуты, изменяемые операторами-обработчиками, на каждом шаге исполнения.

```
<MDElem name="Processing_Graph">
    <MDElem name="node.0">
        <MDATTR name="operator" type="ascii" mode="rw">TOPSAR-Split</MDATTR>
        <MDElem name="sources">
            <MDATTR name="source" ...>hdfs://data/S1A_IW_SLC__1SDV__2AA5.zip</MDATTR>
        </MDElem>
        <MDElem name="parameters">
            <MDATTR name="selectedPolarisations">VV</MDATTR>
            <MDATTR name="subswath" ...>IW2</MDATTR>
            <MDATTR name="firstBurstIndex"...>3</MDATTR>
            <MDATTR name="lastBurstIndex"...>3</MDATTR>
        </MDElem>
    </MDElem>
    <MDElem name="node.1">
        <MDATTR name="operator" ...>Write</MDATTR>
        ...
        <MDElem name="parameters">
            <MDATTR name="file" ...>hdfs://s1_tops_split/S1A_IW_SLC__1SDV__2AA5_split.dim</MDATTR>
            ...
        </MDElem>
    </MDElem>
</MDElem>
```

Рис. 5. Фрагмент метаописания файла радарного снимка после выполнения операции S1 TOPS Split

```
<MDElem name="Processing_Graph">
    <MDElem name="node.0"></MDElem>
    <MDElem name="node.1"></MDElem>
    <MDElem name="node.2">
        <MDATTR name="operator">Apply-Orbit-File</MDATTR>
        <MDElem name="sources">
            <MDATTR name="source" ...>hdfs://s1_tops_split/S1A_IW_SLC__split.dim</MDATTR>
        </MDElem>
        <MDElem name="parameters">
            <MDATTR name="orbitType" ...>Sentinel Precise (Auto Download)</MDATTR>
            <MDATTR name="continueOnFail" ...>false</MDATTR>
            <MDATTR name="polyDegree" ...>3</MDATTR>
        </MDElem>
    </MDElem>
    <MDElem name="node.3">
        <MDATTR name="operator" ...>Write</MDATTR>
        ...
        <MDElem name="parameters">
            <MDATTR name="file" ...>hdfs://apply_orbit_file/S1A_IW_SLC__1SDV__split_Orb.dim</MDATTR>
            ...
        </MDElem>
    </MDElem>
</MDElem>
```

Рис. 6. Фрагмент метаописания файла радарного снимка после выполнения операции Apply Orbit File, следующего за S1 TOPS Split

Каждый новый метод вносит в модель метаописания изменения (выделены прямоугольниками на рис. 5, 6), идентифицирующие его в цепочке обработчиков (операций) радарного снимка, и добавляет целевой файл, который передается как параметр следующему обработчику.

Модель позволяет, не модифицируя иерархии родительских элементов, добавлять дочерние элементы, содержащие структурированное описание оператора-обработчика (название модуля, входные/выходные параметры и т. п.), выполненного предыдущим вызовом. При этом структура общих элементов модели не изменяется, за исключением значений их атрибутов. Оригинальность такого подхода заключается в том, что он позволяет отслеживать целостность и корректность последовательности применения схемы полного цикл обработки радарных снимков и повторно выполнять запуск метода (процедуры в схеме) на нужном шаге.

Программный модуль

Разработанный программный комплекс разделен на три уровня (модули): BACKEND, MIDDLEWARE, FRONTEND. Использовался следующий технологический стэк. Вычислительное ядро (BACKEND), реализовано на базе Apache Spark API (Java, Python), менеджера ресурсов Apache YARN, система менеджера микросистем-контейнеров Docker Swarm. Система обработки http-запросов пользователя (MIDDLEWARE) функционирует под управлением NodeJS, Chart.js (ECMAScript 6 (ES6)). Графический интерфейс комплекса построен с применением библиотек React + Redux, Plotly, Ant UI (ES6) под управлением сервера Nginx (FRONTEND). Операции ввода/вывода BACKEND- и MIDDLEWARE-компонентов реализуются на базе распределенной файловой системы HDFS (ОС Ubuntu 18.04).

BACKEND

BACKEND-уровень представлен python-скриптами, реализующими непосредственно расчетную часть математических алгоритмов методов расчета смещений PS и SBaS и java-классами, имплементирующими методы предварительной обработки радарных данных и методов взаимодействия с объектами Apache Spark API. На рис. 7 показан фрагмент диаграммы классов модуля SBaS. Каждый из классов отвечает за этап схемы расчета (см. рис. 1) и реализует его программную логику. Например, классы Stage1-Stage2 содержат программный код [30] и соответствуют этапам 1—3 алгоритма SBaS (см. рис. 1).

Результатом выполнения методов каждого из классов является файл промежуточных результатов в специальном формате DIMAP (.dim), имеющий структуру представленную на рис. 4—6. Файлы такого формата могут быть переданы по цепочке каждому из классов на обработку.

MIDDLEWARE

MIDDLEWARE-уровень (рис. 8) реализован на базе объектов языка ES6. Имплементирует методы программного каркаса библиотеки NodeJS API. Функционирует как прокси-уровень между FRONTEND и BACKEND, выступает в качестве обработчика пользовательских http-запросов для вызовов их методов.

Для запуска задания на стороне BACKEND-компонента MIDDLEWARE-компонент использует объекты router и request, предоставляемые стандартной библиотекой NodeJS Express API. Данные объекты транслируют POST-запросы со стороны FRONTEND-компонента в команды запуска микросистем-контейнеров Docker Swarm (рис. 9).

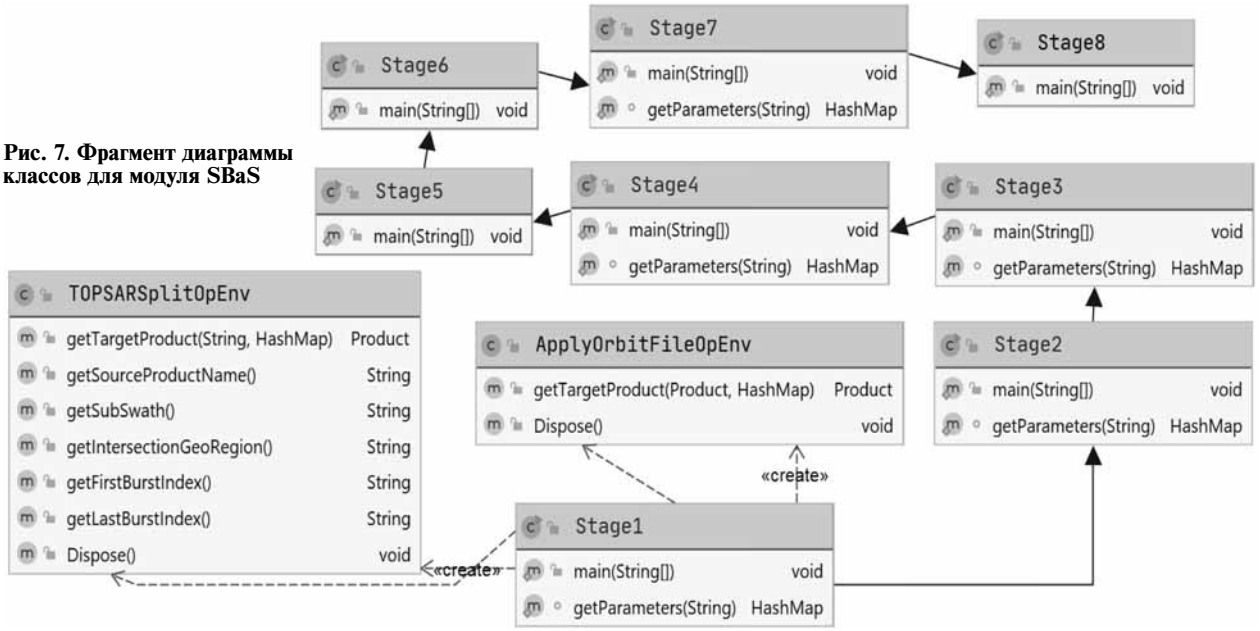


Рис. 7. Фрагмент диаграммы классов для модуля SBaS

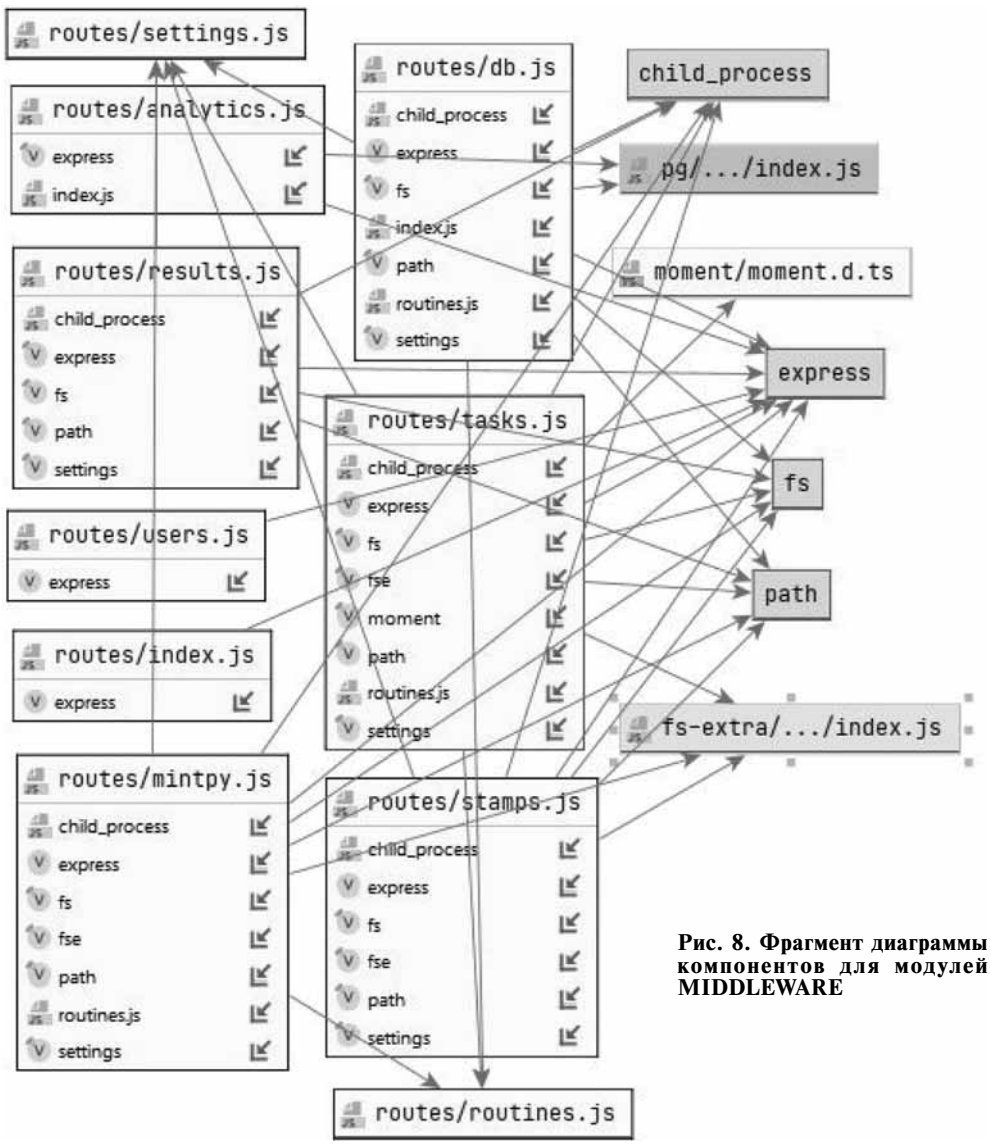


Рис. 8. Фрагмент диаграммы компонентов для модулей MIDDLEWARE

```
docker service create --entrypoint /opt/stamps_prep/bin/start.sh
        --restart-condition=none
        --restart-max-attempts=0
        --mount type=bind,source=/mnt/satimg/Satellites/Sentinel-1A,destination=/mnt/satimg
        --mount type=bind,source=/mnt/disk/public/monitor_radar_usr/id,destination=/mnt/public
        --name stamps_prep -d stamps_prep:1.0 /mnt/output /mnt/public/config /opt/stamps_prep/graphs
            /mnt/satimg/S1B_IW_SLC__1SDV_20190106T002757_20190106T002825_014366_01ABBB_71C4.zip ....
```

Рис. 9. Фрагмент команды запуска контейнера в системе Docker Swarm, запускаемой на стороне MIDDLEWARE-компонента

POST-запрос содержит настройки параметров среды Apache Spark и Docker, менеджера ресурсов Apache YARN, параметров расчетного модуля и управляющей программы-драйвера [30].

Команда содержит названия файловых директорий, которые отображаются в контейнер в качестве локальных, тем самым связывая файловые системы хост-системы с операционной системой контейнера. Это позволяет контейнеру обращаться к любым смонтированным директориям как к своим собственным локальным.

Каждый компонент на диаграмме (см. рис. 8) отвечает за свой тип запросов. Например, mintpy.js отвечает за запуск команд расчета скоростей смещений методом SBaS, stamps.js — методом PS; компонент tasks.js отвечает за создание, обработку параметров расчета, управление пользовательскими заданиями, отслеживание хода выполнения и информирование о завершении; компонент db.js содержит методы взаимодействия с базой данных спутниковых снимков Sentinel-1, методы получения информации о них, ее преобразования и отправки на сторону клиента для отображения в графическом элементе (рис. 10, см. третью сторону обложки). База данных функционирует в СУБД PostgreSQL 12. Таблица содержит название снимка, путь к файлу, типы орбит, дату съемки, поляризацию, названия полос и другую вспомогательную информацию.

FRONTEND

FRONTEND-уровень представлен программными объектами языка ES6, наследующими и расширяющими функциональные возможности типизированных классов библиотеки React. Данные классы отвечают за графический интерфейс и взаимодействие с пользователями веб-приложения (рис. 10, 11).

Каждый элемент графического интерфейса поддерживает модель управления состоянием приложения (Redux) в виде JSON-объектов, содержащих пары ключ/значение свойств данного элемента. Изменение состояния осуществляется за счет функций-действий (*action*), привязанных к тому или иному объекту. FRONTEND-компонент имеет древовидную структуру объектов (родительский/дочерний элемент). Состояние каждого объекта может быть изменено на любом из уровней. Файлы index.js включают код размещения (функция *render*()) JS-элементов более низкого уровня, согласно древовидной структуре, вплоть до последнего дочернего элемента. Такая структура позволяет легко масштабировать систему в целом и выявлять ошибки кода, локализуя их на более низких уровнях.

Основной функцией FRONTEND является подготовка входных данных (рис. 11), передача их на сторону MIDDLEWARE и инициализация удаленного запуска расчета скоростей смещений в системе Apache Spark посредством Docker Swarm. А также визуализация результатов в виде карты смещений. Программные объекты FRONTEND взаимодействуют с компонентами MIDDLEWARE через протокол http, с помощью библиотеки axios-js.

Модуль "База данных" отвечает за создание наборов-списков (DataSet, помечен красным прямоугольником на рис. 10, см. третью сторону обложки) спутниковых снимков на выбранную территорию (помечен синим прямоугольником, рис. 10, см. третью сторону обложки). Основные поля таблицы (рис. 10, см. третью сторону обложки) отображаются в данном графическом элементе и поддерживают гибкую систему фильтрации и выбора. Модуль позволяет управлять ранее созданными списками DataSet (переименовывать, удалять отображать на карте и модифицировать). Созданные в данном модуле списки DataSet используются в модуле "Расчетные задания" (рис. 11).

Модуль "Расчетные задания" позволяет задать входящие. параметры расчета для основных этапов алгоритма расчета скоростей смещений методами SBaS и PS (см. рис. 1, 2). Параметры могут быть выбраны вручную, либо установлены по умолчанию. Также задается название задания и выбирается метод расчета. Указанные в данном модуле параметры сохраняются на стороне MIDDLEWARE в виде JSON-файлов (рис. 12) в файловой системе HDFS и впоследствии считываются в соответствующем контейнер-исполнителе Docker. Каждый JSON-файл сохраняется под своим именем, соответствующем разделу параметров (например, Apply Orbit File или Topo Phase Removal и т. п.). Данный файл является основой для создания метаописаний для операторов-обработчиков, представленных на рис. 5, 6, за счет считывания каждого значения в JSON-файле.

Для каждого вновь созданного задания создается уникальный идентификатор на основе временной метки. На стороне MIDDLEWARE каждый конфигурационный файл содержится в соответствующей файловой папке задания по номеру идентификатора. Согласно команде запуска Docker-контейнера (см. рис. 9) данный идентификатор указывается при монтировании локальных папок в систему Docker Swarm.

На рис. 13 представлен фрагмент кода сохранения JSON-файла параметров на стороне MIDDLEWARE-уровня с применением библиотеки axios-js.

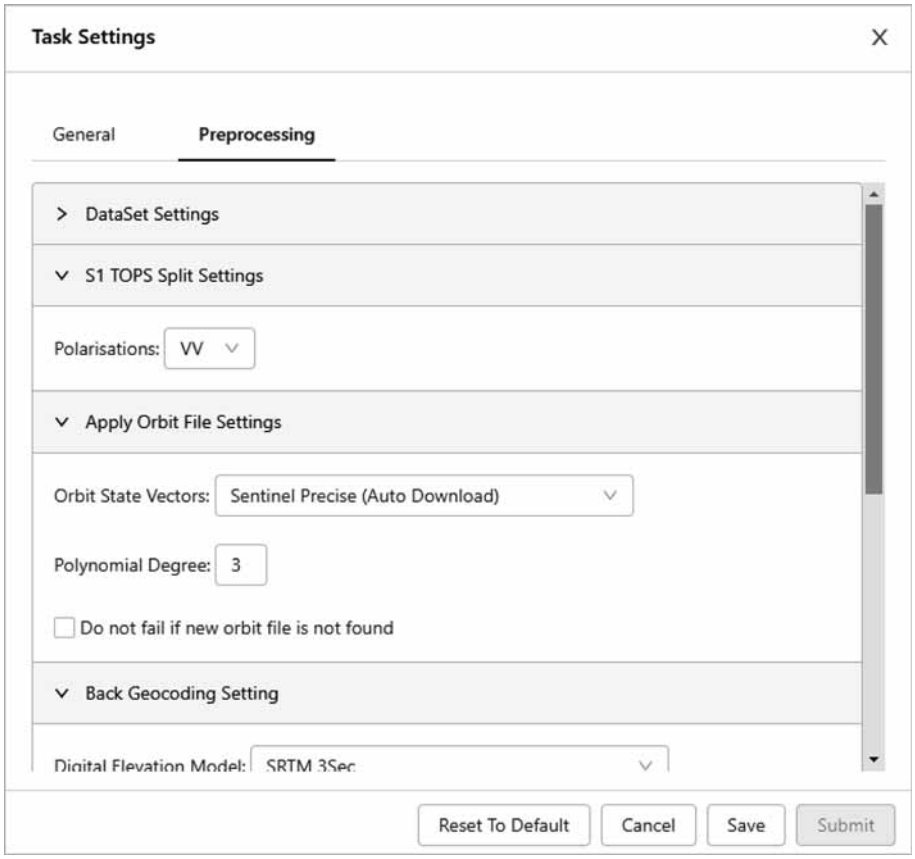


Рис. 11. Графический интерфейс модуля "Расчетные задания"

```
"label": "Back Geocoding",
"name": "back_geocoding",
"description": "This operator co-registers SLC split products (master and slaves)
"parameters": {
  "demName": {
    "index": 0,
    "name": "demName",
    "label": "Digital Elevation Model",
    "type": "inputTypes.select",
    "value": "SRTM 3Sec",
    "defaultvalue": "SRTM 3Sec",
    "options": [
      "ACE2_5min",
      "ACE30",
      "ASTER 1sec GDEM",
      "GETASSE30",
      "SRTM 1Sec Grid",
      "SRTM 3Sec",
      "Kompsat5 Precise"
    ]
  },
  "demResamplingMethod": {"name": "demResamplingMethod"...},
  "resamplingType": {"name": "resamplingType"...},
  "maskOutAreaWithoutElevation": {"name": "maskOutAreaWithoutElevation"...},
```

Рис. 12. Фрагмент JSON-файла параметров (оператор-обработчик Back-Geocoding)

FRONTEND-модуль поддерживает отображение результатов расчетов скоростей смещения методами PS и SBaS на электронной карте с отображением градуированного облака точек (рис. 14, см. третью сторону обложки), соответствующих постоянным отражателям (PS) и выявленным изменениям высот (SBaS). Поддерживается временная шкала выбранных снимков и распределение скоростей на гистограмме.

Методы построения графического интерфейса веб-приложения основаны на подходах, заложенных в технологии Redux. Оригинальный графический интерфейс, предлагающий пользователю возможность сформировать параметры для расчетных методов PS/SBaS в виде запросов, возвращающих данные в виде JSON-файлов, далее обрабатываемых программными функциями отображения графических элементов.

Оригинальностью данного подхода к разработке ПО стоит считать использование технология ReactJS, которая позволяет организовать взаимодействие клиент-серверной части посредством асинхронных вызовов удаленных методов. Данный подход снимет сложности, связанные с сохранением состояния переменных как на стороне сервера, так и на стороне клиента, что позволяет инкапсулировать функциональные возможности и сделать модель поведения веб-приложения приближенной к стационарному приложению.

Апробация программного комплекса

Данные, используемые в тестах, предоставлены открытым порталом Европейского космического агентства Copernicus Open Access Hub (https://scihub.copernicus.eu/dhus).

Тест проводился для зоны покрытия, равной десяти подполосам (*burst*) одной интерферометрической полосы (IW1) с вертикальной поляризацией (VV) для снимков космического аппарата Sentinel-1A.

Аппаратные характеристики кластера: 3 сервера (AMD Ryzen 1700 (8 + 8 cores (Simultaneous

```
export function saveConfig(taskId, userName, method) {
    let methodName = method.name;
    return axios.post(
        url: 'http://' + settings.MIDDLEWARE_IP + ':' + settings.MIDDLEWARE_PORT
                    + '/' + settings.TASK_PROCESSOR + '/saveConfig',
        data: {
            username: userName,
            taskId: taskId, // уникальный идентификатор задания
            config: JSON.stringify(method.config),
            method: methodName
        }
    ).then(response => {
        let savedTask = {
            key: taskId.toString(),
            date: moment(taskId,  format: 'x').format( format: "YYYY-MM-DD hh:mm:ss").toString(),
            method: method.config.common.parameters.method.value,
            description: method.config.common.parameters.description.value,
            filesList: method.config.step1.dataSet.parameters.filesList.value,
            masterName: method.config.step1.dataSet.parameters.masterName.value,
            status: '',
            execTime: null
        }
        return {status: response.data, task: savedTask};
    }).catch(e => {
        return false;
    });
```

Рис. 13. Функция сохранения параметров в JSON-файл

Multi-Threading)) 3.2 GHz, 32Gb RAM, 1Gb/s — скорость передачи данных между серверами). Программное обеспечение ESA SNAP запускалось на одном узле, без поддержки Apache Spark API.

В таблице приведен тест для 32 пар снимков для процедур предобработки, представленных на рис. 1, 2.

На рис. 15 приведен тест процедуры Interferogram полной схемы. Начальные значения для процедуры — 2 пары изображений, конечное — 32 пары (6101×1091). На рис. 15 приведено сравнительное время расчетов.

Результаты тестирования системы в сравнении с программным комплексом ESA SNAP для некоторых этапов расчетной схемы

Метод	Время расчета, мин	
	Apache Spark API	ESA SNAP Toolbox
Размер снимка 24027×9117		
Apply-Orbit-File	2,27	24,85
Back-Geocoding	3,5	62,21
TOPS Deburst	1,2	13,61
Subset	0,41	8,21
Размер снимка 6101×1091		
Interferogram	4,81	125,9
Topo Phase Removal	1,06	5,51

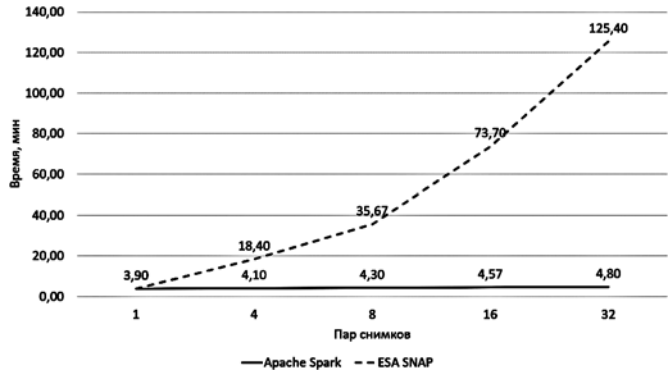


Рис. 15. Зависимость времени расчета процедуры Interferogram на базе системы Apache Spark в представленном программном комплексе и без нее в программном комплексе ESA SNAP

Сравнение финальных результатов расчета скоростей смещений (рис. 16, см. четвертую сторону обложки) проводилось для схемы метода SBaS (см. рис. 2) в представленном программном комплексе и в ПО ENVI с модулем SARScape. Рассматривалась территория г. Кемерово Кемеровской области, Россия. Для наглядности результаты экспортированы и визуализированы в ПО QGIS. Попиксельное сравнение значений скоростей смещений показало 90 %-ное совпадение результатов с допустимым отклонением в 10^{-5} мм/год. Различия некоторых значений допустимы в виду отличия алгоритмов пред- и постобработки в сравниваемых программных комплексах.

Заключение

Предложенный в работе способ обработки радарных данных в системе массово-параллельного исполнения заданий Apache Spark позволяет существенно сократить время исполнения как отдельных процедур, так и полного цикла расчета скоростей смещений. За счет использования контейнеризации объектов-исполнителей достигнута независимость расчетов как внутри одного пула заданий, так и между различными пулами, инициализированными в многопользовательском режиме. Применение системы управления ресурсами кластера и планирования заданий YARN позволило абстрагировать все вычислительные ресурсы кластера от конкретного запуска заданий и обеспечить диспетчеризацию приложений распределенной обработки.

Использование в программном коде на основе Sentinel-1 Toolbox возможности сохранения промежуточных результатов работы процедур в схемах расчета скоростей смещений позволяет проводить расчеты с различными параметрами, а распараллеливание обеспечивает сокращение времени расчетов по сравнению с коммерческими программными продуктами.

Использование открытого программного кода и эргономичность запуска расчетных заданий в разработанном способе обработки радарных данных обеспечивает возможность его реализации в виде веб-приложения.

Результаты тестирования программного комплекса показали высокие характеристики в плане производительности с сохранением требуемой точности результатов. В частности, адаптированные и интегрированные в систему Apache Spark + Docker Swarm.

Список литературы

1. **SBaS** Tutorial. URL: http://sarmap.ch/tutorials/sbas_tutorial_V_2_0.pdf (дата обращения 12.02.2021).
2. **Sousaa J. J., Hooperc J. A., Hanssenc R. F.** et al. Persistent Scatterer InSAR: A comparison of methodologies based on a model of temporal deformation vs. spatial correlation selection criteria // Remote Sensing of Environment. — 2011. — Vol. 115, No. 10. — P. 2652—2663.
3. **Massonnet D., Feigl K. L.** Radar interferometry and its application to changes in the earth's surface // Reviews of Geophysics. — 1998. — Vol. 36. — P. 441—500.
4. **Costantini M., Farina A., Zirilli F.** A fast phase unwrapping algorithm for SAR interferometry // IEEE Trans. GARS. — 1999. — Vol. 37, No. 1. — P. 452—460.
5. **Mistry P., Braganza S., Kaeli D., Leeser M.** Accelerating phase unwrapping and affine transformations for optical quadrature microscopy using CUDA // Proc. of the 2nd Workshop on General Purpose Processing on Graphics Processing Units, GPGPU. Washington, DC, USA. — 2009. — P. 28—37.
6. **Karasev P., Campbell D., Richards M.** Obtaining a 35x Speedup in 2D Phase Unwrapping Using Commodity Graphics Processors // Proc. of the Radar Conference. Waltham, MA, USA. IEEE. — 2007. — P. 574—578.
7. **Zhenhua Wu, Wenjing Ma, Guoping Long** et al. High Performance Two-Dimensional Phase Unwrapping on GPUs // Proc. of the 11th ACM Conference on Computing Frontiers. Cagliari, Italy. — 2014. — P. 1—10.
8. **Shi Xin-Liang, Xie Xiao-Chun.** GPU acceleration of range alignment based on minimum entropy criterion // Proc. of Radar Conference. Xi'an. — 2013. — P. 1—4.
9. **Guerriero A., Anelli V., Pagliara A.** et al. High performance GPU implementation of InSAR time-consuming algorithm kernels // Proc. of the 1st WORKSHOP on the State of the art and Challenges Of Research Efforts at POLIBA. Milan, Italy. — 2015. — P. 4264—4267.
10. **Fan Zhang, Bing-nan Wang, Mao-sheng Xiang.** Accelerating InSAR raw data simulation on GPU using CUDA / Fan Zhang, Bing-nan Wang, Mao-sheng Xiang // Proc. of the Geoscience and Remote Sensing Symposium (IGARSS). Honolulu, HI, USA. — 2010. — P. 2932—2935.
11. **Marinkovic P. S., Hanssen R. F., Kampes B. M.** Utilization of Parallelization Algorithms in InSAR/PS-InSAR Processing // Proc. of the 2004 ENVISAT ERS Symposium (ESA SP-572). Salzburg, Austria. — 2004. — P. 1—7.
12. **Gao Sheng, Zeng Qi-ming, Jiao Jian, Liang Cun-ren, Tong Qing-xi.** Parallel processing of InSAR interferogram filtering with CUDA programming // Science of Surveying and Mapping. — 2015. — No. 1. — P. 54—68.
13. **A High** Performance Message Passing Library. URL: https://www.open-mpi.org/ (дата обращения 30.03.2021).
14. **Introduction** to Doris. URL: http://doris.tudelft.nl/ (дата обращения 30.03.2021).
15. **Frigo M., Johnson S. G.** FFTW: An Adaptive Software Architecture for the FFT // Proc. of 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98. Seattle, WA, USA. — 1998. — Vol. 3. — P. 1381—1384.
16. **GPU-Accelerated** Libraries. URL: https://developer.nvidia.com/gpu-accelerated-libraries (дата обращения 30.03.2021).
17. **Introduction** to ScaLAPAC. URL: http://www.netlib.org/scalapack/ (дата обращения 30.03.2021).
18. **Феоктистов А. А., Захаров А. И., Гусев М. А., Денисов П. В.** Исследование возможностей метода малых базовых линий на примере модуля SBaS программного пакета SARScape и данных PCA ASAR/ENVISat и PALSAR/ALOS. Часть 1. Ключевые моменты метода // Журнал Радиоэлектроники. — 2015. — № 9. — С. 1—26.
19. **Riccardo L., Francesco C., Mariarosaria M.** et al. An overview of the small baseline subset algorithm: A DInSAR technique for surface deformation analysis // Pure and applied geophysics. — 2007. — Vol. 164. — P. 637-661.
20. **Geohazard** Tep. URL: https://geohazards-tep.eo.esa.int/ (дата обращения: 15.02.2018).
21. **Zinno I., Mossucca L., Elefante S.** et al. Cloud Computing for Earth Surface Deformation Analysis via Spaceborne Radar Imaging: a case study // IEEE Trans. Cloud Computing. — 2015. — Vol. 4, No. 1. — P. 104—118.
22. **Zinno I., Stefano E., Lorenzo M.** et al. First Assessment of the P-SBaS DInSAR Algorithm Performances Within a Cloud Computing Environment // IEEE Journal of Selected Topics in Applied Earth Observation and Remote Sensing. — 2015. — Vol. 8, Iss. 10. — P. 4675—4686.
23. **Simmons A. D., Kerekes J. P., Raqueno N. G.** Hyperspectral monitoring of chemically sensitive plant sentinels // Proc. of the SPIE 7457. San Diego, CA, USA. — 2003. — P. 45—51.
24. **Лупян Е. А., Савин И. Ю., Барталев С. А.** и др. Спутниковый сервис мониторинга состояния растительности ("Вега") // Современные проблемы дистанционного зондирования Земли из космоса. — 2011. — Т. 8, № 1. — С. 190—198.
25. **Lavrova O. Yu., Mityagina M. I., Uvarov I. A.** et al. Multi-User Information System Ocean Processes Investigations Based on Satellite Remote Sensing Data // An International Journal of Earth Sciences. — 2013. — Vol. 54. — P. 146—147.
26. **Гордеев Е. И., Гирина О. А., Лупян Е. А.** и др. Изучение продуктов извержений вулканов Камчатки с помощью гиперспектральных спутниковых данных в информационной системе VolSatView // Современные проблемы дистанционного зондирования Земли из космоса. — 2015. — Т. 12, № 1. — С. 113—128.
27. **Лупян Е. А. Барталев С. А., Ершов Д. В.** и др. Организация работы со спутниковыми данными в информационной системе дистанционного мониторинга лесных пожаров Федерального агентства лесного хозяйства (ИСДМ-Рослесхоз) // Современные проблемы дистанционного зондирования Земли из космоса. — 2015. — Т. 12, № 5. — С. 222—250.
28. **Spark** Documentation. URL: https://spark.apache.org/docs/latest/ (дата обращения 28.04.2021).
29. **Docker** Engine Overview. URL: https://docs.docker.com/engine/ (дата обращения 28.04.2021).
30. **Попов С. Е., Замараев Р. Ю., Миков Л. С.** Массово-параллельный подход к обработке радарных данных // Современные проблемы дистанционного зондирования Земли из космоса. — 2020. — Т. 17, № 2. — С. 49—61.

Software for Calculating Deformations of the Earth's Surface using Satellite Radar Data

S. E. Popov, popov@ict.sbras.ru, **R. Yu. Zamaraev,** zamaraev@ict.sbras.ru,
N. I. Yukina, yukina@ict.sbras.ru, **O. L. Giniyatullina,** skiporol@mail.ru,
L. S. Mikov, mikov@ict.sbras.ru, **I. E. Kharlampenkov,** kharlampenkov@ict.sbras.ru,
E. L. Schastlivtsev, schastlivtsev@ict.sbras.ru, Federal Research Center for Information and Computational Technologies, Novosibirsk, 630090, Russian Federation

Corresponding author:

Popov Sergey E., Senior Researcher, Federal Research Center for Information and Computational Technologies, Novosibirsk, 630090, Russian Federation
E-mail: popov@ict.sbras.ru

Received on May 05, 2021
Accessed on May 25, 2021

The article presents a description of a software package for calculating displacement rates and detecting displacements of the earth's surface over areas of intensive coal mining. The complex is built on the basis of the microservice architecture Docker Swarm in integration with the system of massively parallel execution of tasks Apache Spark, as a high-level tool for organizing container-type computations with orchestration of hardware resources. In the software package, the container is used as an element of the sequence of calculation stages of the mathematical model of interferometric processing, presented in the form of a managed service. The service itself is built on the basis of a microkernel of the specified operating system, with support for multitasking of process identifiers and network protocols. Due to the use of containerization of executor objects, the independence of calculations is achieved both within one pool of jobs and between different pools initialized in multi-user mode. The use of the cluster resource management system and YARN job scheduling made it possible to abstract all the computing resources of the cluster from the specific launch of jobs and to provide dispatching of distributed processing applications.

The use in the program code based on the Sentinel-1 Toolbox of the possibility of storing the intermediate results of the operation of procedures in the schemes for calculating the displacement rates makes it possible to carry out calculations with various parameters, and parallelization provides a reduction in the calculation time in comparison with commercial software products.

The combination of Docker Swarm and Apache Spark technologies in one software package made it possible to implement the idea of a high-performance computing system based on open source software and cross-platform programming languages Java and Python using low-budget hardware blocks, including those made in Russia.

***Keywords*:** *Software package, differential interferometry, ground displacement velocities, constant reflector method, small baseline method*

Acknowledgements:

The reported study was funded by RFBR and Kemerovo region, project number 20-47-420002.

For citation:

Popov S. E., Zamaraev R. Yu., Yukina N. I., Giniyatullina O. L., Mikov L. S., Kharlampenkov I. E., Schastlivtsev E. L. Software for Calculating Deformations of the Earth's Surface using Satellite Radar Data, *Programmnaya Ingeneria,* 2021, vol. 12, no. 5, pp. 246—259.

DOI: 10.17587/prin.12.246-259

References

1. **SBaS** Tutorial, available at: http://sarmap.ch/tutorials/sbas_tutorial_V_2_0

2. **Sousaa J. J., Hooperc J. A., Hanssenc R. F.** et al. Persistent Scatterer InSAR: A comparison of methodologies based on a model of temporal deformation vs. spatial correlation selection criteria, *Remote Sensing of Environment,* 2011, vol. 115, no. 10, pp. 2652—2663.

3. **Massonnet D., Feigl K. L.** Radar interferometry and its application to changes in the earth's surface, *Reviews of Geophysics*, 1998, vol. 36, pp. 441—500.

4. **Costantini M., Farina A., Zirilli F.** A fast phase unwrapping algorithm for SAR interferometry, *IEEE Trans. GARS,* 1999, vol. 37, no. 1, pp. 452—460.

5. **Mistry P., Braganza S., Kaeli D., Leeser M.** Accelerating phase unwrapping and affine transformations for optical quadrature microscopy using CUDA, *Proc. of the 2nd Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU, Washington, DC, USA, 2009, pp. 28—37.

6. **Karasev P., Campbell D., Richards M.** Obtaining a 35x Speedup in 2D Phase Unwrapping Using Commodity Graphics Processors, *Proc. of the Radar Conference*, Waltham, MA, USA, IEEE, 2007, pp. 574—578.

7. **Zhenhua Wu, Wenjing Ma, Guoping Long** et al. High Performance Two-Dimensional Phase Unwrapping on GPUs, *Proc. of the 11th ACM Conference on Computing Frontiers*, Cagliari, Italy, 2014, pp. 1—10.

8. **Shi Xin-Liang, Xie Xiao-Chun.** GPU acceleration of range alignment based on minimum entropy criterion, *Proc. of Radar Conference*, Xi'an, 2013, pp. 1—4.
9. **Guerriero A., Anelli V., Pagliara A., Nutricato R., Nitti D.** High performance GPU implementation of InSAR time-consuming algorithm kernels, *Proc. of the Ist WORKSHOP on the State of the art and Challenges Of Research Efforts at POLIBA*, Milan, Italy, 2015, pp. 4264—4267.
10. **Fan Zhang, Bing-nan Wang, Mao-sheng Xiang.** Accelerating InSAR raw data simulation on GPU using CUDA / Fan Zhang, Bing-nan Wang, Mao-sheng Xiang, *Proc. of the Geoscience and Remote Sensing Symposium (IGARSS),* Honolulu, HI, USA, 2010, pp. 2932—2935.
11. **Marinkovic P. S., Hanssen R. F., Kampes B. M.** Utilization of Parallelization Algorithms in InSAR/PS-InSAR Processing, *Proc. of the 2004 ENVISAT ERS Symposium (ESA SP-572),* Salzburg, Austria, 2004, pp. 1—7.
12. **Gao Sheng, Zeng Qi-ming, Jiao Jian, Liang Cun-ren, Tong Qing-xi.** Parallel processing of InSAR interferogram filtering with CUDA programming, *Science of Surveying and Mapping*, 2015, no. 1, pp. 54—68.
13. **A High** Performance Message Passing Library, available at: https://www.open-mpi.org/
14. **Introduction** to Doris, available at: http://doris.tudelft.nl/
15. **Frigo M., Johnson S. G.** FFTW: An Adaptive Software Architecture for the FFT, *Proc. of 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98,* Seattle, WA, USA, 1998, vol. 3, pp. 1381—1384.
16. **GPU-Accelerated** Libraries, available at: https://developer.nvidia.com/gpu-accelerated-libraries
17. **Introduction** to ScaLAPAC, available at: http://www.netlib.org/scalapack
18. **Feoktistov A. A., Zakharov A. I., Gusev M. A., Denisov P. V.** Investigation of the capabilities of the small baseline method using the example of the SBaS module of the SARScape software package and the SAR / ENVISat and PALSAR / ALOS SAR data. Part 1. Key points of the method, *Zhurnal Radioelektroniki,* 2015, no. 9, pp. 1—26 (in Russian).
19. **Riccardo L., Francesco C., Mariarosaria M.** et al. An overview of the small baseline subset algorithm: A DInSAR technique for surface deformation analysis, *Pure and applied geophysics,* 2007, vol. 164, pp. 637—661.
20. **Geohazard** Tep, available at: https://geohazards-tep.eo.esa.int/
21. **Zinno I., Mossucca L., Elefante S.** et al. Cloud Computing for Earth Surface Deformation Analysis via Spaceborne Radar Imaging: a Case Study, *IEEE Trans. Cloud Computing*, 2015, vol. 4, no. 1, pp. 104—118.
22. **Zinno I., Stefano E., Lorenzo M. et al.** First Assessment of the P-SBaS DInSAR Algorithm Performances Within a Cloud Computing Environment, *IEEE Journal of Selected Topics in Applied Earth Observation and Remote Sensing,* 2015, vol. 8, iss. 10, pp. 4675—4686.
23. **Simmons A. D., Kerekes J. P., Raqueno N. G.** Hyperspectral monitoring of chemically sensitive plant sentinels, *Proc. of the SPIE 7457,* San Diego, CA, USA, 2003, pp. 45—51.
24. **Lupyan E. A., Savin I. Yu., Bartalev S. A. et al.** Satellite service for monitoring the state of vegetation ("Vega"), *Sovremennyye problemy distantsionnogo zondirovaniya Zemli iz kosmosa*, 2011, vol. 8, no. 1, pp. 190—198 (in Russian).
25. **Lavrova O. Yu., Mityagina M. I., Uvarov I. A.** et al. Multi-User Information System Ocean Processes Investigations Based on Satellite Remote Sensing Data, *An International Journal of Earth Sciences,* 2013, vol. 54, pp. 146—147.
26. **Gordeev E. I., Girina O. A., Lupyan E. A.** et al. Study of volcanic eruption products in Kamchatka using hyperspectral satellite data in the VolSatView information system, *Sovremennyye problemy distantsionnogo zondirovaniya Zemli iz kosmosa,* 2015, vol. 12, no. 1, pp. 113—128 (in Russian).
27. **Lupyan E. A., Bartalev S. A., Ershov D. V.** et al. Organization of work with satellite data in the information system for remote monitoring of forest fires of the Federal Forestry Agency (ISDM-Rosleskhoz), *Sovremennyye problemy distantsionnogo zondirovaniya Zemli iz kosmosa*, 2015, vol. 12, no. 5, pp. 222—250 (in Russian).
28. **Spark** Documentation, available at: https://spark.apache.org/docs/latest/
29. **Docker** Engine Overview, available at: https://docs.docker.com/engine/
30. **Popov S. E., Zamaraev R. Yu., Mikov L. S.** Massively Parallel Approach to Radar Data Processing, *Sovremennyye problemy distantsionnogo zondirovaniya Zemli iz kosmosa*, 2020, vol. 17, no. 2, pp. 49—61 (in Russian).

ИНФОРМАЦИЯ

Searching for Activity Results and Experts in a Given Subject Area, Taking Results Significance into Account

D. A. Shachnev, mitya57@mitya57.me, Software Engineer, Lomonosov Moscow State University, Institute of Mechanics Moscow, 119192, Russian Federation

Corresponding author:

Shachnev Dmitry A., Software Engineer, Lomonosov Moscow State University, Institute of Mechanics, Moscow, 119192, Russian Federation
E-mail: mitya57@mitya57.me

Received on April 12, 2021
Accepted on May 12, 2021

The paper describes the mathematical models, algorithms and software tools developed by the author for searching for objects in information analysis systems. A search query is a subject area, which is specified by keywords or rubrics of various classifiers. The search takes into account the relevance of the objects to the query, which is calculated by constructing thematic portraits of the objects and comparing them to the query, as well as the significance of the objects, the rules for calculating which can be set dynamically. Thematic portraits of objects are built by analyzing their relations with other objects in the system, for which a topic is specified.

When applied to current research information systems, such an approach can be used to search for scientific results in a given subject area and for experts in this area.

Keywords: *CRIS systems, information analysis systems, information search, subject area, keywords, similarity coefficient*

УДК 004.424.4:004.891 DOI: 10.17587/prin.12.260-266

Д. А. Шачнев, программист, mitya57@mitya57.me, НИИ механики МГУ им. М. В. Ломоносова

Поиск результатов деятельности и экспертов в заданной предметной области с учетом значимости результатов

Описаны разработанные автором математические модели, алгоритмы и программные средства для поиска объектов в информационно-аналитических системах. Поисковым запросом является предметная область, которая задается ключевыми словами или рубриками различных классификаторов. При поиске учитывается релевантность объекта запросу, которая рассчитывается на основе построения тематического портрета объекта и его сравнения с запросом, а также коэффициенты значимости результатов, правила для вычисления которых могут быть заданы динамически. Тематические портреты объектов строятся на основе анализа их связей с другими объектами в системе, для которых есть информация о тематике.

В приложении к системам, хранящим и анализирующим информацию о научной деятельности (CRIS-системам), подобный подход может быть использован для поиска научных результатов в заданной предметной области и для поиска экспертов по этой области.

Ключевые слова: *CRIS-системы, информационно-аналитические системы, информационный поиск, предметная область, ключевые слова, коэффициент схожести*

Introduction

Modern information analysis systems store objects of various types, and these objects are related to each other with various types of relations. The natural metaphor for such a system is a graph, even if the actual database management system uses a different data model such as relational one.

We will illustrate our algorithms on the example of current research information systems (CRIS). These systems store information about researchers, their works and activity results, such as publications, conference talks,

participation in projects, and courses taught. But they also store information about all related objects, such as journals, theses collections, conferences, organizations and their departments. All these objects are closely related to each other: a publication can be made in a journal or in a collection, a researcher works in a department, and so on.

These relations can be used for building thematic portraits of objects: for example, if articles have keywords associated with them, it is possible to construct a weighted set of keywords appearing in a journal or in a collection where the articles are published. But these relations, along with the numeric indicators that are part of the objects' metadata, can also be used to assess the significance of activity results. We propose a model that allows the users to dynamically specify the rules for calculating significance coefficients. For example, a significance rule for an article can be based on its citations count, presence in international citation indices, journal impact factor, number of pages and other indicators. The model allows one to combine multiple indicators using numeric or logical operations. It should be noted that the use of impact factors for assessing quality of scientific works is sometimes criticized [1], however we do not have a goal of developing new scientometric indicators, instead we provide users with an ability to use the indicators that are most suitable for their subject area.

To sum up, the goal of this paper is developing algorithms for searching for activity results and experts in a given subject area, using the data from CRIS systems and taking into account rules for filtering the results and assigning significance coefficients to them. The algorithms should be capable of finding the objects by their own keywords and rubrics, by synonyms, and — if the object does not have its own keywords — using the data from related objects in the system.

Review of existing works

Expert finding systems were already known in the 1980s. Early research in that direction, such as [2], assumed that there is a finite set of experts, and for each of them their area of interest or concern is known, and this information is used for search. Later works investigated the ability of using additional information such as email communications [3], documents on an organization's intranet [4], or personal home pages and data on projects and events [5].

Two modern systems that aggregate information about researchers and their activity results are AMiner [6] and Microsoft Academic Graph [7]. The former uses data from the internet, mostly from the researchers' personal pages. The latter also uses the data of Bing search service. For building the list of subject areas and classifying the data, machine learning algorithms are used.

These works are mostly focused on the analysis of heterogeneous data, such as web pages and texts of publications. In contrary, this work assumes use of structured data from CRIS systems, which allows taking more information about activity results and their related objects into account.

Thematic portraits

By thematic portraits in a general sense we will denote a data structure that describes the subject matter of an object. Such portraits can be constructed for almost any type of objects that are present in the information system. There are several known ways to build such structures.

First of all, we should mention a model that represents the subject in the form of a vector in $\mathbb{R}^n$ space. In particular, such models are used in neural networks based algorithms, for example, in the word2vec [8] algorithm. The main advantage of this model is the simplicity of assessing the similarity of various objects: it is enough to take the dot product of two vectors. However, the results of operations on vectors are difficult to visualize. At the same time, our algorithms can be a part of business processes where it is needed to illustrate the degree of correspondence of a particular portrait to a given subject area. Another limitation is that when composing vectors from texts, all possible keywords are usually taken as the basis of the space. This implies independence of words from each other and does not allow taking into account relationships between words, such as synonymy. To eliminate this shortcoming, algorithms such as latent semantic analysis (LSA) [9] have been developed, but they have a rather large computational complexity and make the presentation even less intuitive.

Another direction of thematic analysis is associated with the construction of ontologies of subject areas, i.e. determining the classes, objects in each class and relationships between objects. However, at the moment, the methods for automated construction of ontologies based on texts are far from perfect and allow identifying only some of the relationships. So building a full-fledged ontology requires a lot of manual work [10].

In this work, we will use a model based on keywords and rubrics. The main reason for this choice is that both keywords and rubrics are an essential part of scientific publications and other objects appearing in CRIS systems. Also, such portraits are very intuitive even if they are created algorithmically.

A keyword is one or several words in a natural language describing a subject area. A rubric is an element of some tree classification. Examples of them are Universal Decimal Classification (UDC) [11] or the Field of Science and Technology Classification of Organisation for Economic Co-operation and Development (OECD).

We will use the term *feature* to refer to either a keyword or a rubric. Features in a portrait have weights associated with them. The weight of a feature is a number between 0 and 1 that shows the degree to which it characterizes the object.

Based on the above, the notion of a thematic portrait is formalized as follows:

Definition 1. A thematic portrait is a set of features (keywords or rubrics), in which every feature has a weight associated with it:

$$p=\{(f_1, c_1), ..., (f_n, c_n)\},$$

where $0 < c_i \leqslant 1$. By F_p we denote the set of features that form a portrait.

Our goal in this section is developing functions of similarity between two features and between two portraits, i.e. functions that determine how close the corresponding subject areas are to each other. The first function is based on the context similarity function described in [12], with modifications that take weights c_i into account and that

make sure that the similarity of a feature with itself is equal to 1. This approach is itself based on the distributional hypothesis, that implies that two features are more similar to each other if they occur in a larger number of common contexts (portraits). In other words, if we build an auxiliary graph in which nodes correspond to features, and there is an edge between two nodes if there is a portrait containing two corresponding features, then the value of the similarity function should be higher when two nodes have more common neighbors in that graph. To get this information, we need to use some subset of portraits from the information analysis system as a training set.

First of all, let us define the function that can be interpreted as the weight of an edge connecting two features in the described graph:

$$\omega(f_k, f_l) = \sum_{p \in P(f_k) \cap P(f_l)} \frac{c_l}{\sum_{f_i \in F_p} c_i},$$

where $P(f)$ is the subset of portraits in the training set that contain feature f. Note that this function is not symmetrical because it takes into account weights of f_l and not f_k in each portrait p.

Then, let us denote by $F_{i,j}$ the set of features that occur together with both f_i and f_j in the training set:

$$F_{i,j} = \left\{ f : \left|P(f_i) \cap P(f)\right| \geqslant 1, \left|P(f_j) \cap P(f)\right| \geqslant 1 \right\}.$$

As the initial version of the similarity function, we take the following:

$$s_0(f_i, f_j) = \sum_{f_k \in F_{i,j}} \omega(f_i, f_k) + \sum_{f_k \in F_{i,j}} \omega(f_j, f_k).$$

The disadvantage of this formula is that it does not take into account the frequency of occurrence of the feature in the entire training set. Because of this, the value of similarity function of two features may be large only because these features are quite general and often occur in the sample. To correct this, we will divide it by sum of frequencies of the two features. So the final similarity function is:

$$s(f_i, f_j) = \frac{s_0(f_i, f_j)}{\left|P(f_i)\right| + \left|P(f_j)\right|}.$$

This function is symmetrical because both f_i and f_j appear as left arguments of function ω. Let us prove that it takes values in range [0, 1].

Theorem 1. $s(f_i, f_j) = 1$ if and only if all features that are neighbors of f_i are also neighbors of f_j, and vice versa. In all other cases $s(f_i, f_j) < 1$.

Proof. Let us first consider the case when the sets of neighboring features coincide. Then the set $F_{i,j}$ also coincides with them, and each feature of every portrait containing f_i or f_j is included in $F_{i,j}$. Some features can be present in several such portraits, so their contribution to $\omega(f_i, f_k)$ or $\omega(f_j, f_k)$ consists of several components corresponding to each portrait. Thus, we can talk about the contribution of the pair (portrait p, feature $f \in F_p$).

The numerator consists of two sums: $\sum_{f_k \in F_{i,j}} \omega(f_i, f_k)$ and $\sum_{f_k \in F_{i,j}} \omega(f_j, f_k)$. Note that for each portrait $p \in P(f_i)$, the sum of the contributions of the pairs (p, f) corresponding to this portrait to the first sum is equal to

$$\sum_{f_k \in F_p} \frac{c_k}{\sum_{f_l \in F_p} c_l} = 1.$$

Therefore, the sum of the contributions of all pairs to the first sum is $\sum_{p \in P(f_i)} 1 = \left|P(f_i)\right|$. Similarly, the sum of the contributions of all pairs to the second sum is equal to $\left|P(f_j)\right|$. Hence, the numerator is equal to $\left|P(f_i)\right| + \left|P(f_j)\right|$. But this is exactly the denominator, so $s(f_i, f_j) = 1$.

If the sets of neighboring features do not coincide, either the first sum in the numerator is less than the left side of the denominator, or the second sum is less than the right side, or both. Thus, we get $s(f_i, f_j) < 1$. Q.E.D.

For rubrics of a common classification, there is another natural similarity metric: length of the path between two rubrics in the classification tree. We can use this metric to verify the adequacy of the developed similarity function. Such a comparison has been performed for the GRNTI (Russian State Rubricator of Scientific and Technical Information). This classifier has three levels, so with the root node added, the maximum path length between two rubrics can be six. All pairs of rubrics were split into seven groups based on the length of path between them, and within each group the average value of similarity function has been calculated, based on research projects data from a CRIS system. The results of such comparison are shown in Fig. 1. As the figure shows, the average value decreases almost uniformly as the path length grows.

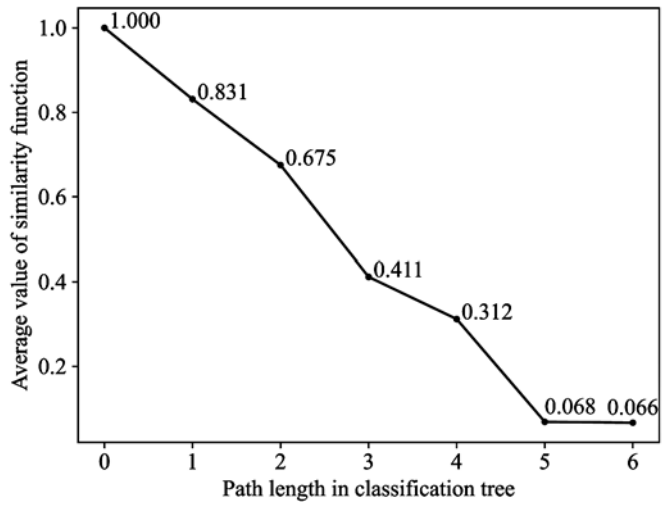


Fig. 1. Comparison of path length in classification tree with the similarity function

Now let us define the similarity function for two thematic portraits. Portraits are often represented as vectors in $\mathbb{R}^n$, and similarity is measured as the cosine of angle between them. However, if there is a non-zero similarity between some of the features forming the basis, such a basis cannot be treated as orthonormal. There is a known generalization of cosine measure that takes similarity between features into account, namely the *soft cosine similarity* [13]. We will use the second variant of soft cosine function because it is known to take values in range [0, 1]. It is defined as follows.

Let p and p' be two thematic portraits. Let us extend each portrait with features from the other one, with zero weights. Then the portraits can be represented as $\{(f_i, c_i) \mid 1 \leqslant i \leqslant n\}$ and $\{(f_i, c'_i) \mid 1 \leqslant i \leqslant n\}$. Let us introduce the following auxiliary values:

$$c_{ij} = \sqrt{s(f_i, f_j)}\frac{c_i + c_j}{2},\ c'_{ij} = \sqrt{s(f_i, f_j)}\frac{c'_i + c'_j}{2}.$$

Now the similarity function of p and p' is:

$$s(p, p') = \frac{\sum_{i,j} c_{ij} c'_{ij}}{\sqrt{\sum_{i,j} c_{ij}^2}\sqrt{\sum_{i,j} c_{ij}'^2}}.$$

It follows from the Cauchy—Schwarz inequality that this function takes values from 0 to 1, and its value is 1 if and only if c_{ij} and c'_{ij} are proportional, which is equivalent to c_i and c'_i being proportional. The value also does not change if a feature is replaced with another one having the same similarity with all the other f_i.

Building thematic portraits using relations between objects

Data in information analysis systems can often be represented as a directed graph, with nodes corresponding to objects and edges corresponding to properties of objects and relations between them. Properties can be numbers, strings or other values that are not tied to the specifics of the system. We will call these *literal values* and define L as the set of all possible literal values. Features are also members of L. Now let us formalize the notion of information analysis system:

Definition 2. Information analysis system is the following pair of sets:

- Set N of graph nodes.
- Set E of graph edges, in which each edge is a semantic triple (subj, pred, obj), where subj, pred $\in N$, obj $\in N \cup L$. Elements of triples are called subject, predicate and object.

A predicate determines the type of relation between the subject and the object. For example, there can be the following relations between a human and a book: author, editor, translator.

To build a portrait for a node (which we will call target node), we need the information about features which are connected to that node via one or multiple edges. The most useful features are those that are connected using a single edge, but these may not be always available.

As edges are directed, not all edges in the path between a feature and a node will necessarily have the same direction. To overcome this complication, let us add for each predicate pred a complementary predicate pred^{-1} to N, and for each triple (subj, pred, obj) such as obj $\in N$, add (obj, pred^{-1}, subj) to E.

Now we can transform any such path into a chain of edges directed from the target node to the feature. This can be achieved by replacing every edge having the opposite direction with the newly added reverse triple. Then each such path can be represented in the following form:

$$\text{subj} \xrightarrow{\text{pred}_1} \text{obj}_1 \xrightarrow{\text{pred}_2} \ldots \xrightarrow{\text{pred}_m} f,$$

where subj is the target node, f is the feature.

For each type of the nodes, different relations should be taken into account. For example, a portrait of a researcher can be deduced from their publications, and a portrait of a dissertation council can be deduced from the dissertations defended there or from the portraits of its members.

The input of the algorithm for building the thematic portraits for nodes of a particular type is a list of tuples of predicates, where each tuple has some positive coefficient associated with it, and the sum of all coefficients is equal to 1. Let us denote the tuples as $(\text{pred}_1, \ldots, \text{pred}_m)$ and the coefficient as $\sigma(\text{pred}_1, \ldots, \text{pred}_m)$.

The first step of the algorithm is building for each tuple of predicates $(\text{pred}_1, \ldots, \text{pred}_m)$ all possible chains of nodes $\text{obj}_1, \ldots, \text{obj}_m$, where $\text{obj}_i \in \text{pred}_i(\text{obj}_{i-1})$ for $1 \leqslant i \leqslant m$, $\text{obj}_0 = \text{subj}$, $\text{obj}_m \in L$ is a feature.

We will iterate each such chain from the last element to the first one, and on each step we will assign some weight to the current node. Initially the weight of obj_m is 1. The weight of obj_i should depend on how many are there outgoing edges from obj_i and incoming edges to obj_{i+1} with the same predicate pred_{i+1}.

Definition 3. Image of node subj under predicate pred is the set $\text{pred}(\text{subj}) = \{\text{obj} \in N \cup L : (\text{subj}, \text{pred}, \text{obj}) \in E\}$.

Preimage of node obj under predicate pred is the set $\text{pred}^{-1}(\text{obj}) = \{\text{subj} \in N : (\text{subj}, \text{pred}, \text{obj}) \in E\}$ (which is the same as image under predicate pred^{-1}).

Based on this definition, we can define the weight of an edge $\text{subj} \xrightarrow{\text{pred}} \text{obj}$ as follows:

$$\Omega(\text{subj}, \text{pred}, \text{obj}) = \frac{1}{|\text{pred}(\text{subj})| \cdot |\text{pred}^{-1}(\text{obj})|}.$$

We consider the second factor to be 1 if obj $\in L$.

Now, if $l : \text{obj}_0 \xrightarrow{\text{pred}_1} \text{obj}_1 \xrightarrow{\text{pred}_2} \ldots \xrightarrow{\text{pred}_m} \text{obj}_m$ is a chain, let us define its contribution to the weight of feature obj_m as

$$\Omega(l) = \sigma(\text{pred}_1, \ldots, \text{pred}_m)\prod_{i=1}^{m}\Omega(\text{obj}_{i-1}, \text{pred}_i, \text{obj}_i).$$

For each feature f_j such that there exists at least one chain ending with it, we define its weight as sum of contributions of all such chains:

$$c_j = \sum_{l:\ldots\to f_j} \Omega(l).$$

Finally, the built thematic portrait of subj is

$$p(\text{subj}) = \{(f_1, c_1), \ldots, (f_n, c_n)\}.$$

Let us show that such a portrait satisfies Definition 1, i.e. all c_j do not exceed 1. We can prove an even stronger statement.

Theorem 2. If $p(\text{subj})$ is built with the algorithm described above, then $\sum_{j=1}^{n} c_j \leqslant 1$.

Proof. Let us consider the total contribution of all chains with specific predicates $\text{pred}_1, \ldots, \text{pred}_m$ and show that this contribution is not larger than $\sigma(\text{pred}_1, \ldots, \text{pred}_m)$. This contribution is equal to

$$\sigma(\text{pred}_1, \ldots, \text{pred}_m) \sum_l \prod_{i=1}^{m} \frac{1}{\left|\text{pred}_i\left(\text{obj}_{i-1}^l\right)\right| \cdot \left|\text{pred}_i^{-1}\left(\text{obj}_i^l\right)\right|}.$$

Because $\left|\text{pred}_i^{-1}\left(\text{obj}_i^l\right)\right| \geqslant 1$, the second factor is not larger than

$$\sum_l \prod_{i=1}^{m} \frac{1}{\left|\text{pred}_i\left(\text{obj}_{i-1}^l\right)\right|} = \sum_{\text{obj}_1 \in \text{pred}_1(\text{obj}_0)} \times$$

$$\times \left(\frac{1}{\left|\text{pred}_1(\text{obj}_0)\right|} \cdots \left(\sum_{\text{obj}_m \in \text{pred}_m(\text{obj}_{m-1})} \frac{1}{\left|\text{pred}_m\left(\text{obj}_{m-1}\right)\right|} \right) \cdots \right).$$

Consider the expression in the innermost parentheses. If $\text{pred}_m(\text{obj}_{m-1}) = \varnothing$ then it is equal to 0, otherwise it is equal to

$$\sum_{\text{obj}_m \in \text{pred}_m(\text{obj}_{m-1})} \frac{1}{\left|\text{pred}_m\left(\text{obj}_{m-1}\right)\right|} =$$

$$= \frac{1}{\left|\text{pred}_m\left(\text{obj}_{m-1}\right)\right|} \cdot \sum_{\text{obj}_m \in \text{pred}_m(\text{obj}_{m-1})} 1 = 1.$$

Therefore, we can remove multiplication by this expression, and the whole sum will not decrease. After such removal the innermost parentheses will contain

$$\sum_{\text{obj}_{m-1} \in \text{pred}_{m-1}(\text{obj}_{m-2})} \frac{1}{\left|\text{pred}_{m-1}\left(\text{obj}_{m-2}\right)\right|},$$

which is similarly equal to 0 or 1. Continuing such removals, after each of which the second factor does not decrease, we finally get that this factor is equal to 0 or 1. Therefore, initially it also did not exceed 1. This means that the total contribution of all chains with the given predicates does not exceed $\sigma(\text{pred}_1, \ldots, \text{pred}_m)$.

So the total sum $\sum_{j=1}^{n} c_j$ which is the sum of contributions of all chains, does not exceed

$$\sum_l \sigma\left(\text{pred}_1^l, \ldots, \text{pred}_m^l\right),$$

but this expression is 1 by construction. Q.E.D.

Similarly it can be proved that the sum of contributions of a particular feature to thematic portraits that are built for the same list of tuples of predicates is not larger than 1.

Assessing the significance of objects

The developed model provides the users with a way to dynamically define rules that determine whether an activity result should be considered in the search and which rank it should have.

CRIS systems may have information about many different types of activity. For example, in the ISTINA information analysis system, which was developed in Lomonosov Moscow State University, there is information about scientific activity (articles, books, conference talks, dissertations, reports), teaching activity (courses developed and taught, scientific advisorship), taking part in research projects, and more.

The main principles of the model were described in [14]. Here we summarize the main ideas of the model. Its key notions are rules and rule sets.

Definition 4. A rule is a structure that specifies the following things.

- The type T of activity results that are being considered.
- One or several restriction functions: $T \to \{0,1\}$, that can be used to filter out unwanted results. A result matches the rule if the value of all restriction functions is 1 for that result.
- A function that is used to calculate the significance coefficient of the result: $T \to \mathbb{R}$.

For each type of activity results, the administrators of an information analysis system can define the standard numeric and boolean indicators, which can then be combined using various arithmetic functions, aggregate functions (such as sum or maximum of values in a set) or logical operators to build the restriction and significance functions. The software implementation, which is described below, allows the users to do this in a graphical interface.

The indicators can use the values of predicates that are available for nodes of the given type, or chains of predicates if it is needed to get a property of an adjacent node. It is also possible to apply a filter on any step of the chain.

All these constructs have their direct equivalents in SQL, the structured query language that is used in many database management systems based on the relational model.

An example of a rule is:

- type: journal articles;
- restriction function: number of pages $\geqslant 5$;
- significance function: journal impact factor for year 2020, divided by the number of coauthors.

Here the significance function is based on a chain of two predicates: "published in" (mapping an article to a journal) and "has impact-factor", and to this chain a filter "year = 2020" is applied.

This rule can be mapped to the following SQL query:

```
select JOURNALRANG.F_JOURNALRANG_VAL
       /(select count(AUTHORA.F_AUTHORA_ID) from AUTHORA
         where AUTHORA.F_ARTICLE_ID = ARTICLE.F_ARTICLE_ID)
from ARTICLE
join JOURNALRANG on JOURNALRANG.F_JOURNAL_ID=ARTICLE.F_JOURNAL_ID
where ARTICLE.F_ARTICLE_NUMPAGES >= 5
and JOURNALRANG.F_JOURNALRANG_YEAR = 2020
```

The significance function may use any scientometric or other numeric indicators of results. For example, for a participation in research projects it may use the amount of funding. The restriction functions can take into account any numeric or boolean indicators. For example, for conference talks such indicators include the conference scope (local, national or international) and talk type (ordinary, invited, plenary or poster).

Sometimes it may make sense for rules to be applied not to the activity result itself, but to the intermediate node that contains the authorship information. This allows taking into account the position of the author in the list, their role (e.g. author or translator), or the participation rate if such information is available. As this intermediate node has an edge pointing to the actual result, all functions that can be evaluated for the result can be also evaluated for the authorship node.

A collection of several rules is called a rule set. There can be multiple rules for the same type of activity results in a rule set. In this case, to get the significance of a result, all these rules are evaluated, and the maximum value among all rules that the result matches is used.

Rule sets may also contain additional restrictions that are useful for all rules. Examples of such restrictions in a CRIS system are: requiring the results to be verified by the organization's representative users, and restricting the year in which the results have been published.

Within the developed model, the information analysis systems may also provide a service which will allow to check if a particular result matches the rules of a set, and for each rule which it does not match provide the value of all restriction functions applied to the result. This will allow the users to understand why the result gets a certain significance coefficient, and whether it is possible to improve it by adding the missing data.

Apart from the application to search, the described model may be used on its own for the automated generation of various reports, ratings and statistics based on the dynamically specified rules. It can be also used for defining data access policies in logical access control models.

Searching for activity results and for experts

The input data for the search algorithm are a thematic portrait and a rule set. The search can also be performed when the subject is given by an activity result — then the portrait can be built using its relations as described above — or by a set of features without weights. In the latter case the weights can be distributed either equally, or giving more weight to the features appearing earlier. For example, the weight of i^{th} token from n can be set to $\frac{1}{n} + \frac{n-2i+1}{n-1}\varepsilon$, where ε is a number between 0 and 1 corresponding to the importance of the order.

The rank of an activity result is calculated based on both its relevance to the search, i.e. the similarity function between its portrait and the portrait from the search query, and its significance coefficient. The natural way to take both these parameters into account is using the product of two values.

So the steps of the search algorithm are:

- The activity results that have non-zero similarity with the query portrait q are determined. We assume that thematic portraits of all activity results are built in advance. To avoid looping over all activity results, it is possible to take only those results, in the portraits of which there are features that have common neighbors with the features from search query.
- For each such result obj, it is first checked whether it matches the restrictions that are associated with the rule set. If it does not, it is excluded. Otherwise, for each rule in the set, it is checked whether the result matches that rule. If there is no matching rule, the result is excluded. Otherwise, for matching rules the values of significance functions are calculated, and the maximum value is used as its significance coefficient.
- The product of the significance coefficient and $s(p(obj), q)$ is used as the rank $\lambda(\text{obj})$ of the activity result. The results are sorted in descending order of rank.

This algorithm can be used to search not only for activity results themselves, but also for experts in a given subject area. In that case, the set of all authors of the found results becomes the set of candidates. Rank of a candidate is calculated as sum of ranks of all their results. The rules can be applied to authorship nodes to take the contribution of a particular author into account.

When searching for experts in order to examine a research project application, it is often needed to detect candidates that may have a conflict of interest. For example, such a conflict may occur if a candidate expert is a colleague or a coauthor of one of the applicants. It is natural for other activity results of the applicants to have a similar subject matter, and thus a high value of similarity with their own application. Such cases may be detected by specifying tuples of predicates with coefficients in range [0, 1], that map the node corresponding to the application to sets of people which may have a conflict of interest. For example, if predicate a means "is an author of", then the set $\left(a^{-1}, a, a^{-1}\right)$ maps the application to the set of coauthors of the applicants. If the coefficient is zero, then all candidate experts that are matched by this set are excluded from search results. If the coefficient is non-zero, then the ranks of the candidates are multiplied by that coefficient.

Software implementation

The presented models and algorithms have been implemented by the author as an experts search module within the ISTINA information analysis system [15]. This system is used in 28 organizations (as of 2021), including Lomonosov Moscow State University and a number of institutes of the Russian Academy of Sciences. The system is

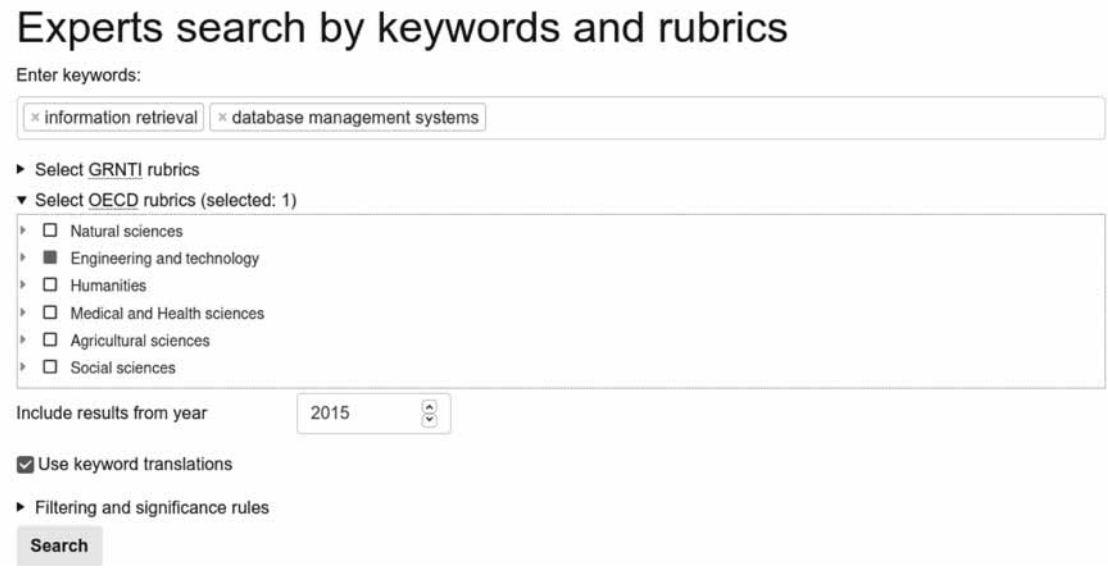


Fig. 2. Experts search user interface

written in Python language using the Django web framework. The PostgreSQL database management system is used as a primary data storage, and the SQL queries are generated using the Django object-relational mapper (QuerySets). The user interface uses the Bootstrap CSS framework.

The search takes the following activity results into account: articles in journals and collections, participation in research projects, defended candidate and doctoral dissertations.

The user interface, as shown in Fig. 2, allows one to enter a set of keywords, and select rubrics from two classifications: the OECD Field of Science and Technology Classification, and GRNTI. Keywords in Russian language can be automatically translated to English using two external sources: ABBYY Lingvo API and Wikipedia API. For each of the supported result types, there is also a form that allows one to specify the criteria for filtering the results and assigning significance coefficients to them.

Conclusion

Searching for activity results and experts is a demanded task in current research information systems. Two common factors that need to be taken into account are relevance to the subject area specified in the search query, and significance of the results. In this paper, we have proposed models and algorithms that consider both these factors by analyzing the relations of objects in the graph of the information analysis system. The algorithms have been tested in a real CRIS system.

For citation:

Shachnev D. A. Searching for Activity Results and Experts in a Given Subject Area, Taking Results Significance into Account, *Programmnaya Ingeneria*, 2021, vol. 12, no. 5, pp. 260—266.

DOI: 10.17587/prin.12.260-266

References

1. **Callaway E.** Beat it, impact factor! Publishing elite turns against controversial metric, *Nature,* 2016, vol. 535, no. 7611, pp. 210—211.

2. **Maron M. E., Curry S., Thompson P.** An inductive search system: Theory, design, and implementation, *IEEE Transactions on Systems, Man, and Cybernetics*, 1986, vol. 16, no. 1, pp. 21—28.

3. **Campbell C. S., Maglio P. P., Cozzi A., Dom B.** Expertise identification using email communications, *Proceedings of the twelfth International conference on information and knowledge management*, New York, NY, USA, Association for Computing Machinery, 2003, pp. 528—531.

4. **Craswell N., Hawking D., Vercoustre A.-M., Wilkins P.** P@NOPTIC expert: Searching for experts not just for documents, *In Ausweb*, 2001, pp. 21—25.

5. **Yimam-Seid D., Kobsa A.** Expert-finding systems for organizations: Problem and domain analysis and the DEMOIR approach, *Journal of Organizational Computing and Electronic Commerce*, 2003, vol. 13, no. 1, pp. 1—24.

6. **Wan H., Zhang Y., Zhang J., Tang J.** AMiner: Search and mining of academic social networks, *Data Intelligence*, 2019, vol. 1, no. 1, pp. 58—76.

7. **Sinha A., Shen Z., Song Y.** et al. An overview of Microsoft Academic Service (MAS) and applications, *Proceedings of the 24th international conference on world wide web*, New York, NY, USA, Association for Computing Machinery, 2015, pp. 243—246.

8. **Mikolov T., Chen K., Corrado G., Dean J.** Efficient estimation of word representations in vector space. arXiv:1301.3781. 2013.

9. **Deerwester S., Dumais S., Furnas G.** et al. Indexing by latent semantic analysis, *Journal of the American Society for Information Science,* 1990, vol. 41, no. 6, pp. 391—407.

10. **Biemann C.** Ontology learning from text: A survey of methods, *LDV-Forum*. 2005, vol. 20, no. 2, pp. 75—93.

11. **McIlwaine I. C.** Universal decimal classification (UDC), *Encyclopedia of library and information sciences,* third edition / Eds. M. J. Bates, M. N. Maack, CRC Press, 2009, pp. 5432—5439.

12. **Lunev K. V.** Graph methods for computing semantic similarity of a pair of keywords and their application to the problem of keywords clustering, *Programmnaya Ingeneria*, 2018, vol. 9, no. 6, pp. 262—271 (in Russian).

13. **Sidorov G., Gelbukh A., Gómez-Adorno H., Pinto D.** Soft similarity and soft cosine measure: Similarity of features in vector space model, *Computación y Sistemas. Centro de Investigación en Computación, IPN*, 2014, vol. 18, no. 3, pp. 491—504.

14. **Afonin S. A., Kozitsyn A. S., Shachnev D. A.** Software mechanisms for scientometrical data aggregation based on ontological representation of the relational database structure, *Programmnaya Ingeneria*, 2016, vol. 7, no. 9, pp. 408—413 (in Russian).

15. **Krivchikov M., Shachnev D., Vasenin V., Zenzinov A.** "ISTINA" data analysis system: Cross-cutting technologies for science and education, *2019 Actual Problems of Systems and Software Engineering (APSSE),* IEEE, 2019, pp. 146—156 (in Russian).

УДК 004.051 DOI: 10.17587/prin.12.267-273

П. М. Николаев, д-р техн. наук, начальник отдела, prokopiy.nikolaev@tsagi.ru,
Федеральное государственное унитарное предприятие "Центральный аэрогидродинамический институт имени профессора Н. Е. Жуковского", Жуковский

Повышение эффективности расчета B-сплайнов в задачах параллельного программирования

Представлен способ повышения эффективности расчета B-сплайнов в задачах параллельного программирования за счет устранения блокировок при обращении к общим модифицируемым данным. Представлена программная реализация в виде шаблона класса C++, обеспечивающего перенос временного массива, используемого при расчете B-сплайна, в локальный буфер заданного размера с возможностью его увеличения в случае необходимости. Использование разработанного шаблона совместно с квалификатором thread_local позволяет сократить число запросов на увеличение буфера для B-сплайнов высокой степени (большей, чем изначально заданный размер буфера). Приведены результаты применения разработанного класса при расчете значений B-сплайнов в многопоточной среде, показывающие сокращение времени расчета пропорционально увеличению числа вычислительных процессоров.

Ключевые слова: *параллельное программирование, многопоточный код, асинхронный доступ к данным, локальная память потока, B-сплайны*

Введение

Использование программных и аппаратных средств параллельных вычислений позволяет существенно сократить время выполнения расчетов во многих инженерных задачах. Доступность вычислительных систем с многоядерными процессорами, а также наличие инструментальных средств с необходимым набором базовых строительных блоков и удобной средой отладки программ с параллельно выполняющимися частями позволяют разработчикам создавать эффективный многопоточный код.

Тем не менее одной из основных трудностей при разработке многопоточных программ остается организация одновременного доступа из разных потоков к совместно используемым данным [1]. Наиболее распространенным решением данной задачи является использование средств блокировки при доступе к разделяемым данным. Такие средства предоставляются как операционной системой, так и современными языками программирования. Например, в язык C++, используемый для листингов кода в настоящей статье, начиная с версии 11, в стандартную библиотеку включены соответствующие шаблоны классов и ключевые слова.

Существует ряд задач, где совместный доступ к модифицируемым данным не нужен, однако требуется обеспечить последовательное использование ограниченного ресурса, например, временного буфера. В таких задачах не происходит обмен данными между разными потоками, но имеется объект, который в конкретный момент времени может использоваться кодом только одного потока. В число подобных задач входит расчет значения B-сплайна.

B-сплайны получили широкое распространение в системах, основанных на геометрическом моделировании предметной области [2]. B-сплайновые кривые и поверхности используются в качестве базовых элементов для представления математических моделей плоских и пространственных объектов. Интерес к ним обусловлен целым рядом свойств [2]. Прежде всего, это простота и наглядность геометрического описания при проектировании сложных форм, а также унифицированное представление данных, позволяющее выполнять обмен геометрической информацией между различными системами автоматизированного проектирования и производства.

В то же время программная реализация функций расчета B-сплайнов, выполненная согласно ставшим уже классическими алгоритмам, приведенным в работах [3, 4], не подходит для использования в многопоточных программах. Требуется модификация данных алгоритмов для обеспечения возможности их параллельного выполнения. В статье [5], также посвященной расчету B-сплайнов в задачах параллельного программирования, описана суть проблемы и возможные варианты ее решения. Приведенные в этой статье решения значительно уменьшают число блокировок при совместном доступе к данным, но не устраняют их полностью. В настоящей работе предлагается способ, позволяющий полностью исключить блокировки при расчете B-сплайнов, повысив таким образом степень параллельности выполняемого кода.

Ограничения при параллельном расчете B-сплайнов

В целях более точного понимания представленных далее результатов повторим приведенные в статье [5] рассуждения, показывающие суть проблемы при расчете B-сплайна в многопоточной среде.

Как было отмечено выше, для расчета значений B-сплайнов в большинстве случаев используются алгоритмы, описанные в работах [3, 4]. Эффективность данных алгоритмов с точки зрения минимизации времени вычислений обеспечивается использованием вспомогательных массивов для хранения промежуточных результатов расчета. На листинге 1 приведен пример реализации класса объектов B-сплайнов. Функциональность класса сильно сокращена, чтобы показать только основу алгоритма расчета.

```
class Bspline
{
   public:
      Bspline::Bspline(const double *knots, int degree, int nmb)
         : the _ p(degree), the _ n(nmb − degree − 2), U(knots, knots + nmb)
      {
         left.reset(new double[degree + 1]);
         right.reset(new double[degree + 1]);
      }
      void basis _ funs(int i, double u, double *b) const;

   private:
      std::vector<double> U;
      int the _ n;
      int the _ p;
      mutable std::unique _ ptr<double[]> left;
      mutable std::unique _ ptr<double[]> right;
};
void Bspline::basis _ funs(int i, double u, double *b) const
{
      b[0] = 1.;
      for (int j = 1; j <= the _ p; ++ j)
      {
         left[j] = u − U[i + 1 − j];
         right[j] = U[i + j] − u;
         double saved = 0.;
         for (int r = 0; r < j; ++ r)
         {
               double temp = b[r] / (right[r + 1] + left[j − r]);
               b[r] = saved + right[r + 1] * temp;
               saved = left[j − r] * temp;
         }
         b[j] = saved;
      }
}
```

Листинг 1. Реализация класса B-сплайнов

Как видно из кода функции `basis _ funs`, для расчета используются вспомогательные массивы `left` и `right`. В однопоточной среде функция работает без побочных эффектов. Однако при обращении к `basis _ funs` для одного и того же объекта `Bspline` сразу из нескольких потоков произойдет повреждение сохраненных в массивах `left` и `right` промежуточных результатов расчета.

Одним из решений, позволяющих устранить отмеченные выше недостатки, является применение блокировок, обеспечивающих взаимно исключающий доступ к совместно используемым данным из разных потоков. В случае с B-сплайном придется поместить практически все тело функции `basis _ funs` в критическую секцию, как показано на листинге 2.

```
void Bspline::basis _ funs(int i, double u, double *b) const
{
      b[0] = 1.;
      std::lock _ guard<std::mutex> lg{mtx};
      for (int j = 1; j <= the _ p; ++ j)
      {
         left[j] = u − U[i + 1 − j];
         right[j] = U[i + j] − u;
...
```

Листинг 2. Применение блокировки

Как видно, отмеченный подход сводит на нет все преимущества параллельного выполнения функции `basis _ funs`. Критическая секция, охватывающая весь код функции, фактически заставляет ее выполняться в последовательном режиме, попутно добавляя накладные расходы на работу объектов синхронизации.

Для анализа эффективности работы представленного решения рассмотрим пример интенсивного использования расчета B-сплайнов в задаче обработки результатов координатных измерений технических объектов, полученных в результате сканирования с помощью оптической системы.

Задача включает в себя этап совмещения массива измеренных точек с математической моделью изделия для последующего расчета статистических характеристик. Предполагается, что математическая модель задана с помощью набора поверхностей NURBS, использующих в своей основе B-сплайны [2—4]. Наилучшее совмещение осуществляется путем поворота и сдвига измеренных точек для минимизации суммы квадратов расстояний от них до поверхности модели. Детали алгоритма совмещения подробно описаны в работе [6]. Для работы алгоритма совмещения необходимо выполнить поиск точек на поверхности, ближайших к каждой точке замера. Используемая процедура поиска, описанная в работе [7], выполняет многократные обращения к функции вычисления точки на поверхности с соответствующими вызовами функции расчета B-сплайнов.

Массив измеренных точек, полученных в результате сканирования с помощью оптической системы, характеризуется большим объемом данных (10^5 и более точек), что приводит к значительному времени расчета. Однако поиск может выполняться для каждой точки независимо от остальных. Поэтому для повышения эффективности имеет смысл разделить весь массив измеренных точек на части и осуществлять расчет в параллельно выполняющихся потоках, количество которых определяется числом доступных вычислительных единиц (ядер, процессоров).

На рис. 1 изображена тестовая модель и измеренные точки. При этом для наглядности отображены не все точки, а только их очень малая часть. Реальное

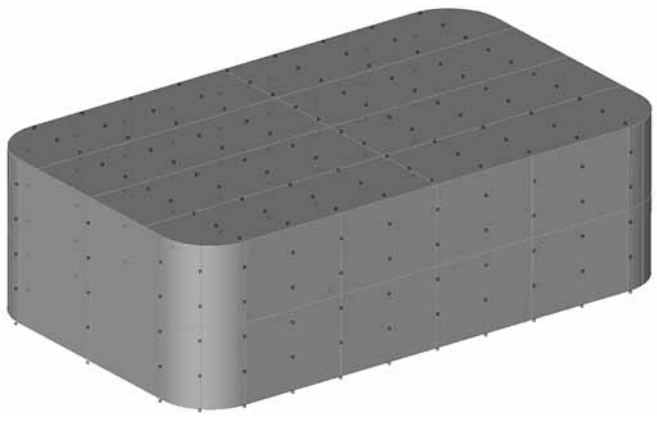


Рис. 1. Тестовая модель объекта и измеренные точки

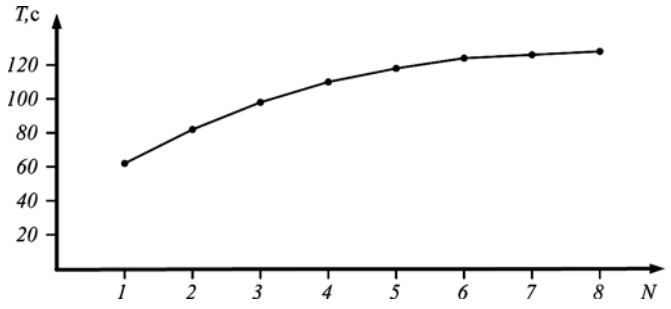


Рис. 2. Зависимость времени выполнения *T* от числа потоков *N* для кода, использующего блокировки

число измеренных точек — порядка 10^5. Размер детали — 50×30×15 мм. Процедура расчета ближайших точек для данного тестового примера потребовала более 10^9 обращений к функции вычисления значений B-сплайнов.

На графике (рис. 2) изображена зависимость времени выполнения процедуры поиска ближайших точек от числа потоков. Результаты получены на компьютере с процессором Intel® Core i7-10700K 3.80GHz (8 вычислительных ядер) и оперативной памятью 64 GB. На рис. 2 видно, что увеличение числа вычислительных потоков не только не уменьшает общее время расчета, а наоборот, увеличивает его. Этот факт можно объяснить тем, что потоки значительную часть времени проводят в состоянии взаимной блокировки. При этом дополнительное время тратится на обслуживание самих блокировок — опрос состояния, захват, освобождение и т. д. Таким образом, решение с использованием блокировок в рассматриваемой задаче не обеспечивает удовлетворительного результата. В связи с этим далее рассматриваются альтернативные способы решения, позволяющие обойтись без блокировок параллельно выполняемого кода при доступе к модифицируемым данным.

Исключение совместного доступа из разных потоков к модифицируемым данным

Наиболее предпочтительным способом организации параллельных вычислений является использование в расчетных процедурах только локальных данных. Причина в том, что доступ к таким данным осуществляется монопольно из текущего потока, при этом нет необходимости заботиться о синхронизации с другими потоками. В задаче с B-сплайнами размеры вспомогательных массивов `left` и `right` определяются на этапе выполнения программы и неизвестны на этапе ее компиляции. Поэтому они обычно размещаются в динамической памяти во время создания объекта `Bspline` (см. листинг 1). Как было показано в предыдущем разделе, такая реализация требует применения блокировок для исключения одновременного изменения массивов `left` и `right` из разных потоков, что крайне негативно сказывается на времени выполнения параллельных программ. Если бы была возможность разместить

массивы локально в функции basis _ funs, то отпала бы необходимость в использовании блокировок. Так как размер массивов left и right известен только на этапе выполнения программы, то естественным способом является их размещение в динамической памяти. Такое решение хотя и устраняет проблему, но влечет за собой дополнительные накладные расходы, связанные с необходимостью выделения динамической памяти с последующим ее освобождением при каждом вызове функции basis _ funs.

Предлагаемое компромиссное решение основывается на наблюдении того, что большая часть B-сплайнов, используемых для моделирования объектов в современных системах автоматизированного проектирования, редко имеет степень, большую трех. А размеры вспомогательных массивов left и right как раз определяются степенью B-сплайнов. Поэтому можно размещать массивы left и right фиксированного "достаточно большого" размера в стеке, а в случае необходимости, когда требуется больший размер, использовать динамическую память.

Программная реализация описанного подхода в виде шаблона класса BufArray приведена на листинге 3.

```
template <size _ t N, typename T>
class BufArray
{
public:
   BufArray(size _ t n = N)
   {
   pbuf = (n <= N)? buf: new T[n];
   }
   ~BufArray()
   {
   if (pbuf != buf) delete[] pbuf;
   }
   T& operator[](size _ t i) { return pbuf[i]; }
   const T& operator[](size _ t i) const { return pbuf[i]; }
   operator T*() const { return pbuf; }
private:
   T buf[N];
   T *pbuf;
};
```

Листинг 3. Шаблон BufArray

Шаблон класса BufArray реализует семантику массива данных необходимого размера. При этом, если запрашиваемый в конструкторе размер *n* массива не превышает значения, заданного шаблонным параметром *N*, то используется размещенный в стеке массив buf. В противном случае выполняется запрос на выделение динамической памяти.

Пример использования шаблона BufArray для организации локальных массивов в функции basis _ funs показан на листинге 4. В качестве "достаточно большого" размера стекового массива взято значение 8, означающее, что расчет сплайнов до седьмой степени включительно будет выполняться без выделения динамической памяти.

```
void Bspline::basis _ funs(int i, double u, double *b) const
{
   BufArray<8, double> left(the _ p + 1);
   BufArray<8, double> right(the _ p + 1);
   b[0] = 1.;
   for (int j = 1; j <= the _ p; ++ j)
   {
        left[j] = u — U[i + 1 — j];
        right[j] = U[i + j] — u;
...
```

Листинг 4. Использование шаблона BufArray

На графике (рис. 3) изображена зависимость времени выполнения процедуры поиска ближайших точек от числа потоков для модифицированной программы, использующей BufArray. Видно, что теперь время расчета убывает при возрастании числа расчетных единиц. При этом удвоение числа процессоров приводит к сокращению времени расчета почти в два раза. Также следует обратить внимание, что время выполнения с одним потоком тоже сократилось по сравнению с вариантом программы, использующей блокировки (см. рис. 2). Это обусловлено тем, что даже в однопоточном режиме существуют накладные расходы на выполнение до-

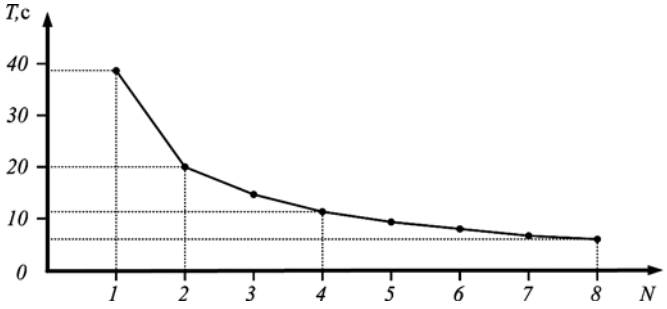


Рис. 3. Зависимость времени выполнения *T* от числа потоков *N* для вычисления без блокировок

полнительного кода, присутствующего в программе из-за объектов блокировки (см. листинг 2).

Таким образом, применение локальных временных массивов в процедуре расчета B-сплайна ограниченной степени позволяет полностью отказаться от использования блокировок, что положительно сказывается на эффективности выполнения прикладных задач в многопоточном режиме. Однако, если природа задачи такова, что имеется достаточно большая вероятность появления сплайнов высокой степени или требование по времени вычисления значений сплайнов любых степеней критично, то можно воспользоваться другим подходом, описываемым в следующем разделе.

Использование локальной памяти потока

Если рассмотреть код функции `basis _ funs` для расчета значений B-сплайна для конкретного объекта `Bspline` (листинг 1), то видно, что в нем отсутствуют обращения к функциям других объектов `Bspline`, которые могут вызвать `basis _ funs`. То есть в конкретный момент времени буферы `left` и `right` могут быть использованы только одним объектом `Bspline` в рамках одного потока. Поэтому теоретически вспомогательные буферы l`eft` и `right` можно хранить в статической памяти, так что они будут использоваться всеми объектами `Bspline`. При этом нужно обеспечить соблюдение условия их достаточного размера для B-сплайна любой степени, а также иметь их отдельные копии в каждом потоке. Последнее условие реализуется путем применения квалификатора памяти `thread _ local`, предназначенного для глобальных и статических объектов. Объект класса, объявленный с таким квалификатором, имеет свою отдельную копию по отношению к каждому потоку, где он используется. При обращении к объекту берется та его копия, которая соответствует потоку, из которого происходит обращение. То есть объект находится в локальной памяти потока.

Для обеспечения достаточного размера вспомогательного буфера можно проверять его текущий размер непосредственно перед использованием, и в случае необходимости увеличивать, выполняя запрос к динамической памяти. В отличие от запроса памяти при каждом вызове `basis _ funs`, такие запросы будут выполняться максимум по одному разу для каждого объекта `Bspline` каждого потока в случае, если степень B-сплайна окажется выше текущего размера буфера.

На листинге 5 представлена модификация шаблона класса `BufArray` с добавленной функцией `CheckSize` и текущим размером буфера `bufsize`.

```
template <size _ t N, typename T>
class BufArray
{
public:
   BufArray(size _ t n = N)
{
      pbuf = (n <= N) ? buf: new T[n];
      if (n > N) bufsize = n;
}
   ~BufArray()
{
      if (pbuf != buf) delete[] pbuf;
   }
   T& operator[](size _ t i) { return pbuf[i]; }
   const T& operator[](size _ t i) const { return pbuf[i]; }
   operator T*() const { return pbuf; }
   void CheckSize(size _ t n)
   {
      if (n <= bufsize) return;
      if (pbuf != buf) delete[] pbuf;
      pbuf = new T[n];
      bufsize = n;
}
private:
   T buf[N];
   T *pbuf;
   size _ t bufsize{N};
};
```

Листинг 5. Шаблон `BufArray` с возможностью динамического увеличения размера

Листинг 6 демонстрирует способ применения данного шаблона в функции `basis _ funs`.

```
static thread _ local BufArray<8, double> left;
static thread _ local BufArray<8, double> right;

void Bspline::basis _ funs(int i, double u, double *b) const
{
   left.CheckSize(the _ p + 1);
   right.CheckSize(the _ p + 1);
   b[0] = 1.;
   for (int j = 1; j <= the _ p; ++ j)
   {
      left[j] = u — U[i + 1 — j];
      right[j] = U[i + j] — u;
...
```

Листинг 6. Размещение объектов `BufArray` в локальной памяти потока

Массивы `left` и `right` определяются в статической памяти, локальной для каждого потока. В функции `basis _ funs` перед использованием массивов проверяется их размер на предмет соответствия степени сплайна. Функция `CheckSize` в случае необходимости увеличивает текущий размер массива. Зависимость времени работы программы от числа задействованных потоков при данном подходе совпадает с зависимостью, представленной на рис. 3.

В заключение следует отметить, что в данном случае вместо шаблона `BufArray` можно применить контейнер `std::vector` из стандартной библиотеки STL C++ (листинг 7).

```
static thread _ local std::vector<double> left;
static thread _ local std::vector<double> right;

void Bspline::basis _ funs(int i, double u, double *b) const
{
   if (left.size() < the _ p + 1) left.resize(the _ p + 1);
   if (right.size() < the _ p + 1) right.resize(the _ p + 1);
   b[0] = 1.;
   for (int j = 1; j <= the _ p; ++ j)
   {
      left[j] = u — U[i + 1 — j];
      right[j] = U[i + j] — u;
...
```

Листинг 7. Применение `std::vector` для вспомогательных массивов

Следует обратить внимание на то обстоятельство, что при использовании описанного подхода создание любого потока, даже не связанного с расчетом B-сплайна, будет сопровождаться построением среды со всеми объявленными статическими объектами с модификатором `thread _ local` [1]. Поэтому желательно, чтобы объявление объекта не сопровождалось "тяжелой" инициализацией. Например, не стоит объявлять статический локальный к потоку (`thread _ local`) буфер типа `std::vector` с заданным ненулевым размером, так как это может сопровождаться обращением к динамическому выделению памяти, что отрицательно скажется на общей производительности приложения, активно использующего создание новых потоков.

Заключение

Рассмотрены различные подходы к организации массивов для сохранения промежуточных результатов расчета B-сплайна. Переменный размер массивов, зависящий от степени сплайна, не позволяет в общем случае задать массив в локальной памяти (стеке) расчетной функции. Показано, что применение блокировок при обращении к общему массиву требуемого размера или обращение к динамической памяти при каждом вызове расчетной функции значительно снижают эффективность программы, особенно в многопоточном режиме. Предложенные решения позволяют обойти описанные ограничения. При этом потенциально возникают дополнительные затраты памяти (в стеке или локальной памяти потока). Однако в современных условиях компьютеры обычно оснащены достаточным количеством оперативной памяти, а требования к скорости расчета более приоритетны.

Также следует отметить, что рассмотренные в статье способы задания массивов неизвестного заранее размера для хранения промежуточных результатов расчета могут быть использованы и в других задачах параллельного программирования.

Список литературы

1. **Williams A.** C++ Concurrency in Action, 2nd Edition, NY: Manning Publications, 2019. — 592 p.
2. **Farin G.** Curves and surfaces for CAGD, San Francisco: Morgan Kaufmann, 2002. — 449 p.
3. **de Boor C.** A practical guide to splines, New York: Springer, 2001. — 348 p.
4. **Piegl L., Tiller W.** The NURBS book, Berlin: Springer, 1997. — 646 p.
5. **Николаев П. М.** Использование локальной памяти потока для расчета B-сплайнов в задачах параллельного программирования // Программная инженерия. — 2019. — Т. 10, № 4. — С. 178—185.
6. **Николаев П. М.** Алгоритм совмещения измеренных точек и математической модели изделия с использованием локальной линеаризации поверхности // Контроль. Диагностика. — 2014. — № 9. — С. 44—48.
7. **Николаев П. М.** Алгоритм поиска точки на параметрической поверхности, ближайшей к заданной точке, для задач контроля точности изготовления изделий // Вестник Брянского государственного технического университета. — 2014. — № 3. — С. 135—137.

Improving the Efficiency of Calculating B-splines in Parallel Programming Tasks

P. M. Nikolaev, prokopiy.nikolaev@tsagi.ru, Central Aerohydrodynamic Institute, Zhukovsky, 140181, Russian Federation

Corresponding author:

Pokoliy M. Nikolaev, Head of Department, Central Aerohydrodynamic Institute, 140181, Zhukovsky, Russian Federation
E-mail: prokopiy.nikolaev@tsagi.ru

Received on April 27, 2021
Accepted on May 26, 2021

The use of parallel computing tools can significantly reduce the execution time of calculations in many engineering tasks. One of the main difficulties in the development of multithreaded programs remains the organization of simultaneous access from different threads to shared data. The most common solution to this problem is to use locking facilities when accessing shared data. There are a number of tasks where data sharing is not needed, but you need to synchronize access to a limited resource, such as a temporary buffer. In such tasks, there is no data exchange between different threads, but there is an object that at a given time can be used by the code of only one thread. One such task is calculating the value of a B-spline. The software implementation of the functions for calculating B-splines, performed according to classical algorithms, requires the use of blocking objects when accessing the common array of intermediate data from different threads. This reduces the degree of parallelism and reduces the efficiency of computational programs using B-splines running on multiprocessor computing systems. The article discusses a way to improve the efficiency of calculating B-splines in parallel programming tasks by eliminating locks when accessing general modified data. A software implementation is presented in the form of a C++ class template, which provides placement of a temporary array used for calculating a B-spline into a local buffer of a given size with the possibility of increasing it if necessary. Using the developed template in conjunction with the thread_local qualifier reduces the number of requests for increasing the buffer for high degree B-splines (larger than the initially specified buffer size). It is also possible to implement this scheme using the std::vector template of the C++ STL Standard Library. The results of the application of the developed class when calculating the values of B-splines in a multithreaded environment, showing a reduction in the calculation time in proportion to an increase in the number of computational processors, are presented. The methods of specifying arrays for storing intermediate calculation results considered in this article can be used in other parallel programming tasks.

Keywords: *parallel programming, multithreaded execution, asynchronous data access, local thread storage, B-spline*

For citation:

Nikolaev P. M. Improving the Efficiency of Calculating B-splines in parallel Programming Tasks, *Programmnaya Ingeneria,* 2021, vol. 12, no. 5, pp. 267—273.

DOI: 10.17587/prin.12.267-273

References

1. **Williams A.** *C++ Concurrency in Action*, 2nd Edition, NY, Manning Publications, 2019, 592 p.
2. **Farin G.** *Curves and surfaces for CAGD*, San Francisco,Morgan Kaufmann, 2002, 449 p.
3. **de Boor C.** *A practical guide to splines*, New York, Springer, 2001, 348 p.
4. **Piegl L., Tiller W.** *The NURBS book*, Berlin, Springer, 1997, 646 p.
5. **Nikolaev P. M.** Using thread local memory for calculating B-splines in parallel programming tasks, *Programmnaya Ingeneria,* 2019, vol. 10, no. 4, pp. 178—185 (in Russian).
6. **Nikolaev P. M.** Applying measured points to a mathematical model using local surface linearization, *Kontrol'. Diagnostika*, 2014, no. 9, pp. 44—48 (in Russian).
7. **Nikolaev P. M.** Calculating a point on the parametric surface closest to the specified point for the tasks of geometrical shape control, *Vestnik Brjanskogo gosudarstvennogo tehnicheskogo universiteta*, 2014, no. 3, pp. 135—137 (in Russian).

УДК 004.4, 004.9 DOI: 10.17587/prin.12.274-280

Д. Н. Кобзаренко, вед. науч. сотр. НИИ РПИ, kobzarenko_dm@mail.ru,
С. Э. Савзиханова, директор НИИ РПИ, sse1122@yandex.ru, ГАОУ ВО "Дагестанский государственный университет народного хозяйства", Махачкала,
Б. И. Шихсаидов, проф., декан, dekan-52@mail.ru, ВГБОУ ВО "Дагестанский государственный аграрный университет им. М. М. Джамбулатова", Махачкала

Средства автоматизации контроля корректности типовых разделов документа преподавателя вуза

Рассмотрен подход к решению задачи, направленной на автоматизацию процессов контроля и редактирования типовых разделов в документе преподавателя вуза в потоковом режиме. В качестве инструментария для решения задачи предложено использовать язык программирования Python с подключением библиотек python-docx и pandas.

Ключевые слова: *контроль содержимого текстового документа, работа с документами в Python, библиотека python-docx, библиотека pandas*

Введение

Автоматизация потокового контроля и частичного редактирования типовых разделов рабочих документов преподавателя вуза, таких как рабочая программа и оценочные материалы по дисциплине, является прикладной задачей информационных технологий в сфере образования. Ее решение ставит цель перевода рутинной трудоемкой работы по проверке и переводу документов, представленных преподавателем в изложении по шаблону, на вычислительную систему.

Рассматриваемый объект — документ преподавателя вуза — имеет две составляющие — типовую (или шаблонную) и содержательную. В типовую составляющую входят структурные элементы, которые касаются содержания, оформления титульного листа, утверждений, информации о дисциплине, о помещениях и т. д., т. е. все то, что делается по определенному вузом шаблону в рамках рекомендаций Минобрнауки РФ. Содержательная часть является творческой прерогативой преподавателя в рамках преподаваемой дисциплины.

При рассмотрении любого документа с точки зрения его проверки, редактирования и анализа можно выделить три аспекта:

- проверка и редактирование стиля и формы изложения текста в документе;
- проверка и редактирование структуры и типовых разделов документа;
- содержательный анализ текста документа.

С технической и алгоритмической точек зрения самой простой задачей является проверка и редактирование параметров оформления документа, самой сложной — содержательный анализ. Содержательному анализу текстовой информации посвящено много научных исследований, разработано множество методик и алгоритмов. В последнее время для анализа текстовой информации используют методы машинного обучения (нейросети). Для содержательного анализа необходимо сформулировать результат, который хочется получить, что он может дать адресату, которому направляется. Если рассматривать содержательный анализ в контексте рабочей документации преподавателя вуза, то с помощью методов машинного обучения можно ставить и решать, например, следующие задачи: степень изменчивости документов при их доработке или нахождение так называемого "выброса" — документа, который явно отличается по содержанию в данной предметной области от подобных документов.

Разработка системы содержательного анализа текста — дело будущих исследований. Текущая же разработка направлена на решение задачи автоматизированной проверки и редактирования структуры и типовых разделов документа.

Следует отметить важность поставленной задачи в рамках учебного процесса, особенно если в вузе отсутствует унифицированная автоматизированная система составления таких документов, как рабочая программа и оценочные материалы. Нередко даже самые ответственные преподаватели могут механически совершать ошибки при составлении подобных документов. Если предположить, что в вузе автоматизирован процесс составления документов преподавателем, то средства их контроля и корректировки также имеют смысл с точки зрения контроля изменения документа от версии к версии или ввода оперативных правок.

В литературе можно встретить большое число работ, например, в части анализа неструктурированных текстовых документов [1]. Существуют примеры работ на тему обработки документов типа XML [2]. Относительно рассматриваемой задачи не найдено полных аналогов. Найдена работа по автоматизации формирования рабочих программ учебных дисци-

плин [3]. Что касается непосредственно обработки текстовых документов — существует работа [4], которая рассматривает разработку редактора, но это не есть задача потоковой автоматизации и тем более без привязки документа предметной области. Следует также выделить работу [5] как наиболее близкую к тематике настоящей статьи. В ней авторы рассматривают разработку автоматизированного контроля оформления документа, под оформлением понимается его форматирование по определенному шаблону, о контроле содержимого речи не идет. При этом не рассматривается процесс потоковой обработки документов.

Концепция построения средств автоматизации

На рис. 1 показаны основные компоненты среды средств автоматизации. Языком программирования для разработки программного обеспечения принят популярный на настоящее время, современный и богатый библиотеками язык Python. Предполагается, что составные части среды размещаются в заранее подготовленной структуре каталогов. В корневом каталоге размещаются пять каталогов для составных компонентов:

- [_LIB] — каталог для размещения функциональных библиотек;
- [_SCRIPTS] — каталог для размещения скриптов подзадач;
- [SOURCES] — каталог для размещения исходных документов, предназначенных для выполнения потоковой процедуры контроля;
- [DOCUMENTS] — каталог для размещения сопутствующих документов, необходимых для выполнения процедуры проверки;
- [SHELL] — каталог, содержащий интерфейсную оболочку.

Принцип функционирования среды состоит в следующем.

Однородные документы (или так называемые пакеты документов) собираются в едином каталоге и помещаются в каталог [SOURCES]. Например, если стоит задача обработать пакет рабочих программ по специальности 09.03.03, то в каталоге [SOURSES] создается каталог [WORK_PROGR] и внутри него помещается каталог с исходными документами [09.03.03]. Для обработки рабочих программ потребуются дополнительные документы — учебные планы по специальности (в нашем случае выполнены в формате Excel). Для размещения учебных планов дневного и заочного отделений в каталоге [DOCUMENTS] создается каталог [PLANS]. Далее в каталоге [09.03.03] наряду с документами для обработки помещается текстовый файл "references.txt" со словарем ссылок на дополнительные документы в рамках проверки. В нашем случае данный файл будет содержать следующую информацию:

```
Учебный_план_ДО={корневая папка}\
DOCUMENTS\PLANS\{файл документа}
Учебный_план_ОЗО={корневая папка}\
DOCUMENTS\PLANS\{файл документа}
```

Для проверки документов определенного шаблона разрабатывается пакет скриптов подзадач Python и помещается в соответствующий каталог (для скриптов проверки рабочей программы логично озаглавить его [WORK_PROGR]) каталога [SCRIPTS]. Пакет может состоять из одного скрипта для полной проверки, однако гораздо удобней как для программиста с точки зрения написания кода и поиска ошибок, так и для администратора сделать разбивку на множество скриптов для разных разделов документа, например, один скрипт для титульной страницы, другой — для первого раздела и т. д.

Скрипт подзадачи Python:

— выполняет соединение с функциональными библиотеками, настраивает пути и базовые переменные;

— задает словарь входных данных (в рамках собственной задачи);

— получает список документов для обработки и сопутствующих документов;

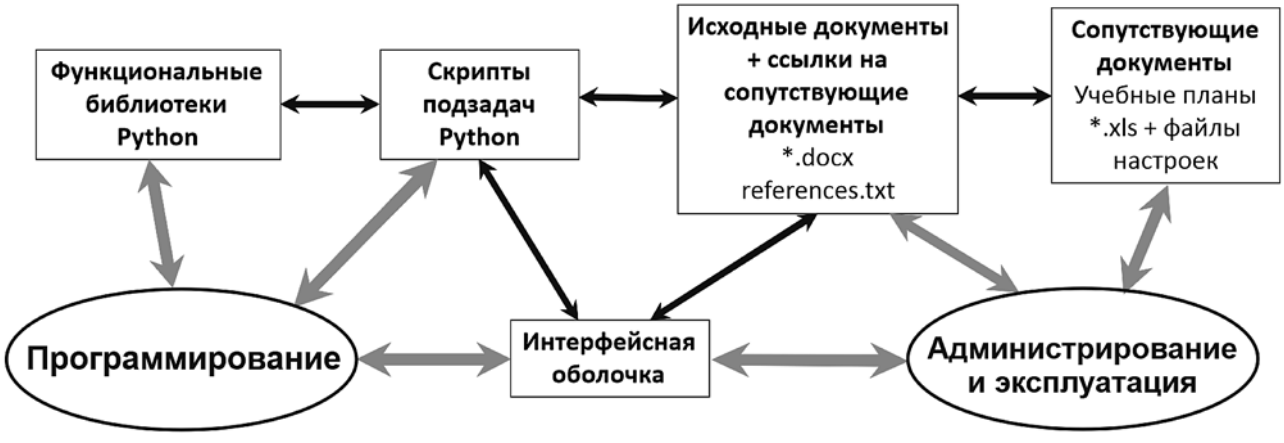


Рис. 1. Структура средств автоматизации контроля корректности типовых разделов документов

— выполняет потоковую проверку документов и регистрирует ошибки в одноименных текстовых файлах — журналах ошибок.

Журналы ошибок (текстовые файлы) генерируются с началом проверки для каждого проверяемого документа с тем же именем и расширением ERR. По мере проверки каждый скрипт добавляет в них свою информацию об ошибках.

Предусмотрены перечисленные далее четыре типа ошибок, регистрируемых в журнале ERR.

1. Критическая ошибка — ошибка, при которой проверка документа становится невозможной. Например, ошибка чтения информации из файла или отсутствие файла.

2. Ошибка идентификации. Генерируется, когда искомый объект проверки невозможно идентифицировать в документе. В таком случае в журнале ошибок необходимо по возможности понятно отразить, на основе чего объект должен был быть идентифицирован, чтобы разработчик документа понимал, в чем суть проблемы.

3. Ошибка содержания. Генерируется, когда объект в документе по содержанию не соответствует тому, что ожидалось. Например, присутствует компетенция, которой не существует в учебном плане, или ошибка в количестве часов.

4. Исправление. Когда объект идентифицирован, но он имеет некорректность, которая исправляется в автоматическом режиме.

После проверки документ считается корректным, если в журнале проверки отсутствуют ошибки типов 1—3.

Содержимое функциональных библиотек приводится далее по тексту в разделе "Реализация программного обеспечения".

Задача интерфейсной оболочки состоит в реализации следующей последовательности действий: 1) выбрать каталог с проверяемыми документами; 2) выбрать каталог со скриптами; 3) обнулить журналы предыдущих проверок; 4) запустить скрипты подзадач на выполнение; 5) визуализировать показатели ошибок и содержимое журналов ошибок.

Интерфейсная оболочка может быть реализована в любой среде программирования в оконном режиме для Windows-приложения или в виде веб-приложения.

В вычислительной системе, на которой устанавливаются рассматриваемые средства автоматизации, должны быть установлены интерпретатор Python, необходимые библиотеки — pandas, python-docx, а также прописан путь в окружении к файлу python.exe.

Реализация программного обеспечения

Для работы на Python в вычислительной системе устанавливаются базовый интерпретатор и оболочка для написания программного кода, в качестве которой использована имеющая свободную лицензию среда PyCharm Community Edition.

Структура функциональной части программного обеспечения среды представлена на рис. 2.

Библиотека pandas используется для чтения информации из файла Excel. Для этого вызывается метод read_excel, позволяющий считать данные в удобном табличном виде. Из таблицы извлекаются все необходимые параметры учебной дисциплины, такие как индекс, число часов, зачетных единиц, компетенции. Для того чтобы корректно извлекать данные о дисциплине, необходимо предварительно подготовить словарь настройки чтения данных из учебного плана, включающий в себя названия нужных страниц и номера нужных столбцов. Словарь записывается в файл текстового формата.

Задачу по извлечению данных о дисциплине из учебного плана формата Excel решает модуль subject_info.py. В модуле спроектированы классы SubjectInfo и Semester. В задачу класса Semester входят инкапсуляция данных в рамках одного семестра и ее представление в виде свойств. Вся работа модуля заключена в классе SubjectInfo. При создании экземпляра класса в конструктор передается название дисциплины. В рамках класса реализованы несколько приватных методов решения подзадач. Публичный метод Run запускает на выполнение задачу по извлечению данных о дисциплине из учебного плана. На входе через параметры функции подается имя файла настроек учебного плана. Функция возвращает 0, если

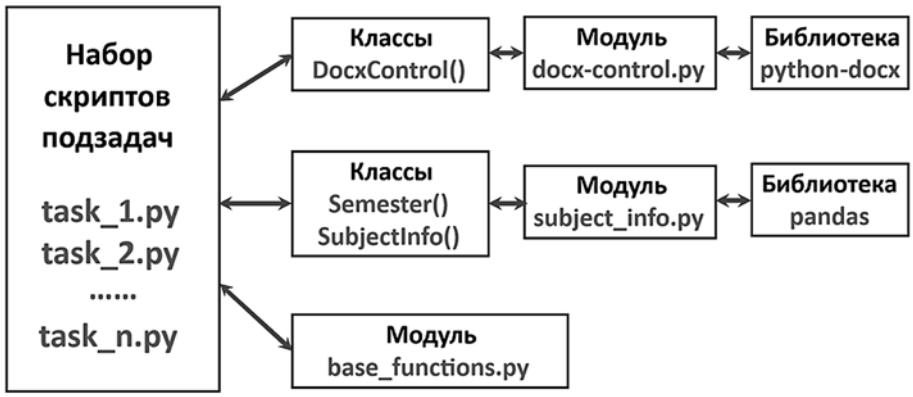


Рис. 2. Структура программной реализации функциональной части среды на языке Python

все успешно прочиталось или код ошибки (1 — файл не существует; 2 — ошибка чтения файла настроек; 3 — отсутствие файла Excel; 4 — ошибка чтения файла Excel). При успешном выполнении заполняются внутренние переменные — словари с прочитанными данными. Доступ к прочитанным данным осуществляется через спроектированные для этого свойства класса.

В модуле base_functions.py собраны типовые функции средств автоматизации, такие как чтение/запись файла журнала ошибок, получение списка docx-файлов папке и др.

Для работы с файлом формата MS Word используется библиотека python-docx. Для этой цели имеется класс Document. Принцип работы объекта, созданного на основе класса Document, следующий. В конструктор передается либо ничего, либо имя файла. Если ничего не передано, то объект инкапсулирует данные пустого файла формата docx, который можно редактировать методами класса и сохранить под любым именем. Если в конструктор передано имя файла, который успешно читается, то его содержимое считывается в переменные объекта для дальнейшей работы. Класс Document считывает документ MS Word в структуру, состоящую из трех списков, доступных в виде свойств Paragraphs для параграфов, Tables для таблиц и Pictures для изображений. Для рассматриваемых задач имеет смысл работать только с параграфами и таблицами. Откровенным минусом класса Document является то, что последовательности прочитанных таблиц, параграфов и других объектов обособлены и нет возможности определить последовательность разнородных объектов в документе. То есть невозможно определить, где конкретно в тексте позиционируется та или иная таблица. Эта особенность класса доставляет неудобства при анализе документа, однако если точно известно, что должно в документе быть в параграфах, а что в таблицах, то проблема решаема.

На базе экземпляра класса Document спроектирован модуль docx_control.py. Модуль предоставляет типовые программные решения для использования в скриптах подзадач. Все это реализовано в виде класса DocxControl.

При создании экземпляра класса DocxControl ему передается имя файла документа MS Word. В публичных методах класса реализованы типовые функции. Функциональные возможности расширяются по мере потребностей в задачах контроля документа. Далее перечислены несколько публичных методов:

— **def** search_paragraph_by_substring(self, substring:str, str_case=True) — возвращает индекс параграфа (или −1 в случае его отсутствия) с полностью совпадающей подстрокой substring, str_case — учет регистра;

— **def** write_paragraph(self, index:int, text:str, format={}) — выполняет запись текста text в параграф index с форматированием format;

— **def** search_paragraph_by_sample(self, sample:str, start=None, finish=None, filter=[], str_case=True, accuracy=100) — возвращает индекс параграфа (или −1 в случае его отсутствия), который с точностью accuracy совпадает со строкой sample, поиск начинается с параграфа start и заканчивается параграфом finish; в str_case задается чувствительность регистра, а в filter — набор символов, которые не учитываются при сравнении строк;

— **def** search_table_cell_by_sample(self, sample:str, index:int, filter=[], str_case=True, accuracy = 100) — возвращает координаты ячейки таблицы index, в которой текст с точностью accuracy совпадает со строкой sample; в str_case задается чувствительность регистра, а в filter — набор символов, которые не учитываются при сравнении строк;

— **def** search_table_by_cell_sample(self, sample:str, row:int, col:int, filter=[], str_case=True, accuracy=100) — возвращает индекс первой найденной таблицы (или −1 в случае ее отсутствия), в которой с точностью accuracy содержимое ячейки row, col совпадает со строкой sample; в str_case задается чувствительность регистра, а в filter — набор символов, которые не учитываются при сравнении строк.

Как видно из представленных функций, во многих из них для идентификации текста рассчитывается точность совпадения строк. Она рассчитывается по следующему алгоритму.

Шаг 1. Входные данные: строки для сравнения arg1 и arg2, список символов фильтра filter, чувствительность к регистру str_case.

Шаг 2. Если str_case = Ложь, то все символы в arg1 и arg2 переводятся к нижнему регистру.

Шаг 3. Из строк arg1 и arg2 удаляются символы из списка filter.

Шаг 4. Строка arg1 переводится в список слов *t*.

Шаг 5. Строка arg2 переводится в список слов *r*.

Шаг 6. Находится *n* — число слов из *t*, имеющихся в *r*.

Шаг 7. Выходные данные точности: accuracy = $= n/\max(\text{длина}(t), \text{длина}(r)) \times 100$.

Скрипты подзадач для удобства создания программного кода выполняются по единому шаблону. Шаблон состоит в следующем:

1) подключение к внешним библиотекам, настройка видимости библиотек в каталоге [_LIB] и подключение к ним;

2) чтение первого и второго параметров командной строки — пути к источнику данных в переменную source_path;

3) заполнение словаря входных данных inp;

4) разработка тела функции Run с параметрами doc_name — файл обрабатываемого документа и inp — словарь входных данных; данная функция реализует проверку документа и начинается с создания экземпляра класса DocxControl;

5) получение списка документов files в папке-источнике;

6) для всех документов списка files запуск функции Run.

Интерфейсная оболочка в текущей версии выполнена в среде Embarcadero Delphi (рис. 3). Она содержит интерфейсные элементы задания папки исходных документов и папки скриптов подзадач. При запуске проверки каждый скрипт запускается как Windows-процесс Python.exe с двумя параметрами,

Рис. 3. Интерфейсная оболочка

а именно — имя скрипта и имя каталога — источника данных. После выполнения проверки интерфейсная оболочка при выборе исходного документа отображает журнал ошибок. Статистика ошибок визуализируется в таблице исходных документов.

Таким образом, средства автоматизации контроля корректности типовых разделов документа являются открытыми для добавления и редактирования новых функций. Проектирование программного кода выполняется по следующей схеме:

1) определяется контролируемый объект в документе и ставится задача контроля;

2) если необходим параметр из сопутствующего документа, то либо идет обращение к существующему объекту, позволяющему считывать данный параметр, либо разрабатывается новый класс с необходимым набором функций (в случае его отсутствия);

3) если класс DocxControl содержит не все методы, необходимые для решения задачи, то в нем проектируются недостающие методы;

4) в скрипте подзадачи на основе экземпляра класса DocxControl и его методов проектируется код решения задачи контроля для объекта в документе в рамках функции Run.

Результаты тестирования

Разработанное программное обеспечение успешно прошло тестирование на решении реальной задачи по изменению в потоковом режиме дат утверждений, номеров протоколов, а также изображений — скринов подписей в конце документов. Обрабатывались одновременно до 40 рабочих программ и оценочных материалов. Под поставленные задачи были разработаны соответствующие скрипты проверки. Для решения задачи замены изображений в класс DocxControl были добавлены методы для удаления абзацев, добавления абзаца в конец документа и вставки изображения с регулируемым параметром ширины.

Тестовые испытания показали, что библиотека python-docx "выбрасывает" исключение при попытке прочитать документ старого формата MS Word (с расширением .doc). Поэтому перед обработкой все документы старого формата .doc должны быть пересохранены в формат .docx. Кроме того, в поле зрения оказался один документ из 200, для которого "выбрасывались" исключения ошибки чтения по невыясненным причинам (формат документа был правильный). С учетом этого обстоятельства важным положительным моментом разработки является тот факт, что журналы ошибок позволяют выявлять документ, на котором обработка завершилась по причине ошибки чтения файла. При устранении дефекта или удалении документа обработка запускается заново.

Тестирование показало, что для удобства использования в интерфейсную часть средств автоматизации целесообразно добавить возможность открывать документ для редактирования в среде MS Word непосредственно из таблицы журнала ошибок. Имеется

ввиду то, что после обработки потока файлов пользователь может передвигаться по таблице журналов ошибок и в том случае, если присутствуют, например, ошибки идентификации объекта, нажатием на соответствующей строке таблицы открыть среду MS Word с искомым файлом для выяснения причин возникновения ошибки.

При обработке документов с помощью разработанных средств автоматизации обнаружена большая доля (около 30 %) ошибок идентификации, связанных либо со склонением в словах, либо с отсутствием какого-то слова в типовых параграфах, либо в силу других подобных механических ошибках составителя документа.

Заключение

Несмотря на то, что проектируемые средства автоматизации ориентированы на конкретную задачу — контроль документов преподавателя вуза, разработка концепции их построения выполнена независимо от типа подлежащей контролю документации. Набор функций может варьироваться в зависимости от решаемой задачи.

В результате работы над созданием средств автоматизации контроля корректности типовых разделов документа преподавателя вуза спроектирован каркас открытой программной среды потоковой проверки типовых разделов документов на языке Python, способной адаптироваться под требуемые задачи проверки. Проект в настоящее время реализован для автономной работы в ОС Windows, предполагается также его реализация в виде веб-приложения. Для активного применения на практике под конкретные задачи требуется разработка соответствующих пакетов скриптов.

Список литературы

1. **Ильвовский Д., Черняк Е.** Системы автоматической обработки текстов // Открытые системы. СУБД. — 2014. — № 1. — С. 51—53.
2. **Трусов А. Н., Иванченко П. Ю., Кацуро Д. А.** Редактирование и внесение информации в XML-документы автоматизированных информационных систем // Программные продукты и системы. — 2017. — № 1. — С. 81—84.
3. **Космичева И. М., Квятковская И. Ю., Сибикина И. В.** Автоматизированная система формирования рабочих программ учебных дисциплин // Вестник АГТУ. Серия Управление, вычислительная техника и информатика. — 2016. — № 1. — С. 90—96.
4. **Щукин М. Ю., Иванова Н. Н.** Разработка автоматизированной системы "Технический редактор текстовых документов MS WORD" // Информатика и вычислительная техника. Сб. науч. трудов / Отв. ред. Н. В. Первова. — Чебоксары, Изд-во ЧГУ имени И. Н. Ульянова, 2019. — С. 258—264.
5. **Стариченко Б. Е., Устинов М. А.** Программа автоматизации контроля оформления текстовых документов // Педагогическое образование в России. — 2018. — № 8. — С. 163—168.

Automation Means for Controlling the Correctness of Typical Sections in a University Teacher's Document

D. N. Kobzarenko, kobzarenko_dm@mail.ru, **S. E. Savzikhanova,** sse1122@yandex.ru, Dagestan State University of National Economy, Makhachkala, 367008, Russian Federation,
B. I. Shikhsaidov, dekan-52@mail.ru, Dagestan State Agrarian University named after M. M. Dzhambulatov, Makhachkala, 367032, Russian Federation

Corresponding author:

Kobzarenko Dmitry N., Leading Researcher, Dagestan State University of National Economy, Makhachkala, 367008, Russian Federation
E-mail: kobzarenko_dm@mail.ru

Received on May 12, 2021
Accepted on May 26, 2021

The paper discusses approach to automating the processes of monitoring and editing standard sections in a document of a university teacher in streaming mode. As a toolkit for solving the problem, it is proposed to use the Python programming language with the connection of the python-docx and pandas libraries.

Despite the fact that the projected automation tools are focused on a specific task to control documents of a university teacher, the development of the concept of their construction is carried out with an abstraction from the type of documentation. The set of functions may vary depending on the problem being solved.

As a result of work on the creation of tools for automating the control of the correctness of typical sections of a university teacher's document, a framework of an open software environment for streaming verification of typi-

cal sections of documents in the Python language was designed, capable of adapting to the required verification tasks. The project is currently implemented for autonomous operation in Windows OS, it is also expected to be implemented as a web application. For active use in practice for specific tasks, the development of appropriate script packages is required.

The main purpose of the development is to release human resources from the routine process of document control and direct it to the implementation of the employee's creative ideas.

Keywords: *MS WORD document, streaming processing, Python, python-docx library, pandas library*

For citation:

Kobzarenko D. N., **Savzikhanova S. E.**, **Shikhsaidov B. I.** Automation Means for Controlling the Correctness of Typical Sections in a University Teacher's Document**,** *Programmnaya Ingeneria,* 2021, vol. 12, no. 5, pp. 273—280.

DOI: 10.17587/prin.12.273-280

References

1. **Ilivovsky D., Chernyak E.** Automatic Word Processing Systems, *Otkrytye sistemy. SUBD*, 2014, no. 1, pp. 51—53 (in Russian).

2. **Trusov A. N., Ivanchenko P. Yu., Katsuro D. A.** Editing and Entering Information into XML Documents of Automated Information Systems, *Programmnye produkty i sistemy*, 2017, no. 1, pp. 81—84 (in Russian).

3. **Kosmicheva I. M., Kvyatkovskaya I. Yu., Sibikina I. V.** Automated system for the creation work programs of academic disciplines, *Vestnik AGTU. Seriya Upravlenie, vychislitel'naya tekhnika i informatika*, 2016, no. 1, pp. 90—96 (in Russian).

4. **Shchukin M. Yu., Ivanova N. N.** Development of the automated system "Technical editor for text documents MS WORD", *Informatika i vychislitel'naya tekhnika — Sbornik nauchnyh trudov. otv. redaktor. N. V. Pervova. Cheboksary*, *Izd-vo CHGU imeni I. N. Ul'yanova*, 2019, pp. 258—264 (in Russian).

5. **Starichenko B E., Ustinov M. A.** Automation program to control text documents, *Pedagogocheskoe obrazovanie v Rossii*, 2018, no. 8, pp. 163—168 (in Russian).

ИНФОРМАЦИЯ

Международная научная конференция "Суперкомпьютерные дни в России" 27—28 сентября 2021 г.

Тематика конференции охватывает следующие основные направления:

- Проблемы создания экзафлопсных суперкомпьютеров: архитектура, программирование, сопровождение
- Суперкомпьютерные технологии в промышленности
- Конвергенция высокопроизводительных вычислений, машинного обучения и технологий больших данных: теория, практика, перспективы, истории успеха
- Перспективные модели, языки и технологии параллельного программирования
- Теория и практика решения больших и сверхбольших задач
- Эффективность и масштабируемость параллельных программ и вычислительных систем
- Новые принципы организации высокопроизводительных вычислений. Нетрадиционные архитектуры вычислительных систем
- Суперкомпьютерные технологии и защита информации
- Технологии распределенных вычислений и распределенной обработки данных, Grid-технологии, облачные технологии
- Большие данные: хранение, обработка, аналитика
- Визуализация в суперкомпьютерном мире: методы, технологии и системы
- Суперкомпьютерное образование

Подробности: **http://russianscdays.org/**

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева*. Корректор Н. В. Яшина

Сдано в набор 03.06.2021 г. Подписано в печать 26.07.2021 г. Формат 60×88 1/8. Заказ PI521
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**

Рисунки к статье **С. Е. Попова, Р. Ю. Замараева, Н. И. Юкиной, О. Л. Гиниятуллиной, Л. С. Микова, И. Е. Харлампенкова, Е. Л. Счастливцева**

«ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ РАСЧЕТА ДЕФОРМАЦИЙ ЗЕМНОЙ ПОВЕРХНОСТИ С ИСПОЛЬЗОВАНИЕМ СПУТНИКОВЫХ РАДАРНЫХ ДАННЫХ»

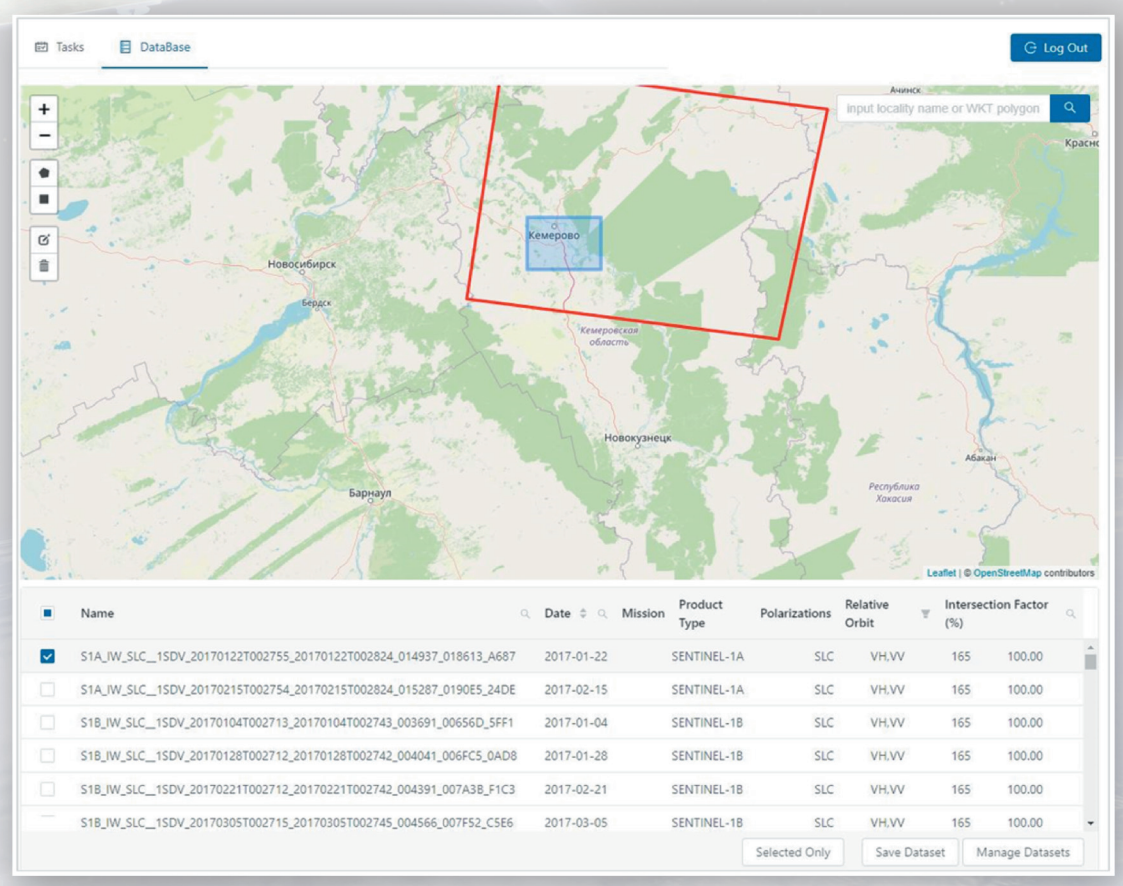


Рис. 10. Графический интерфейс модуля «База данных»

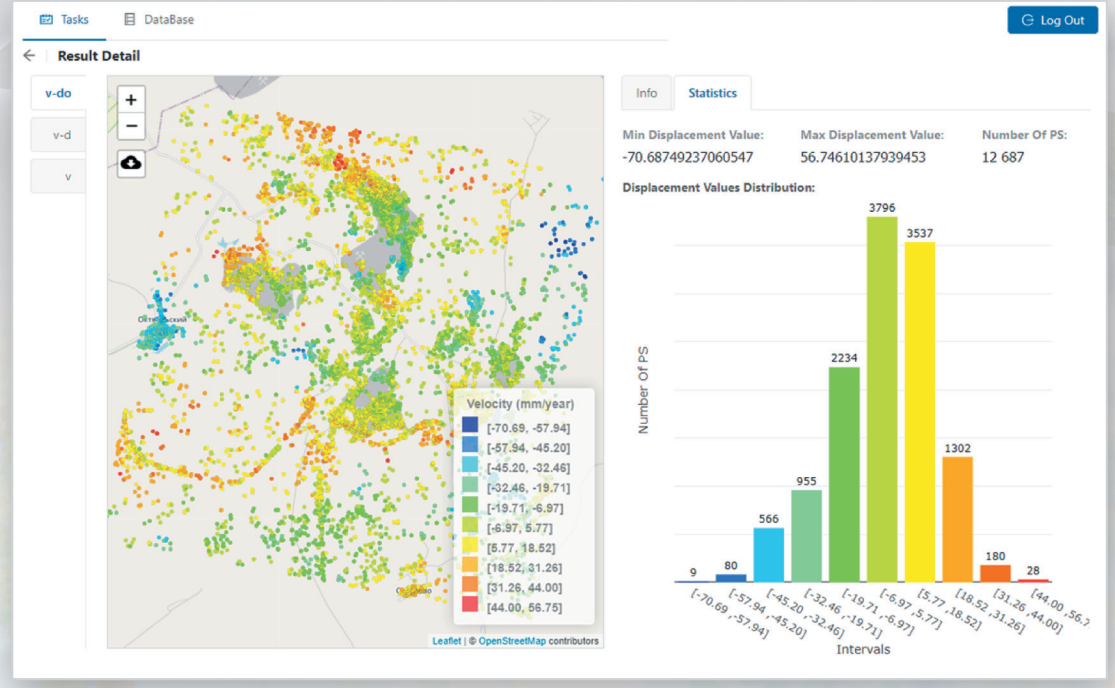
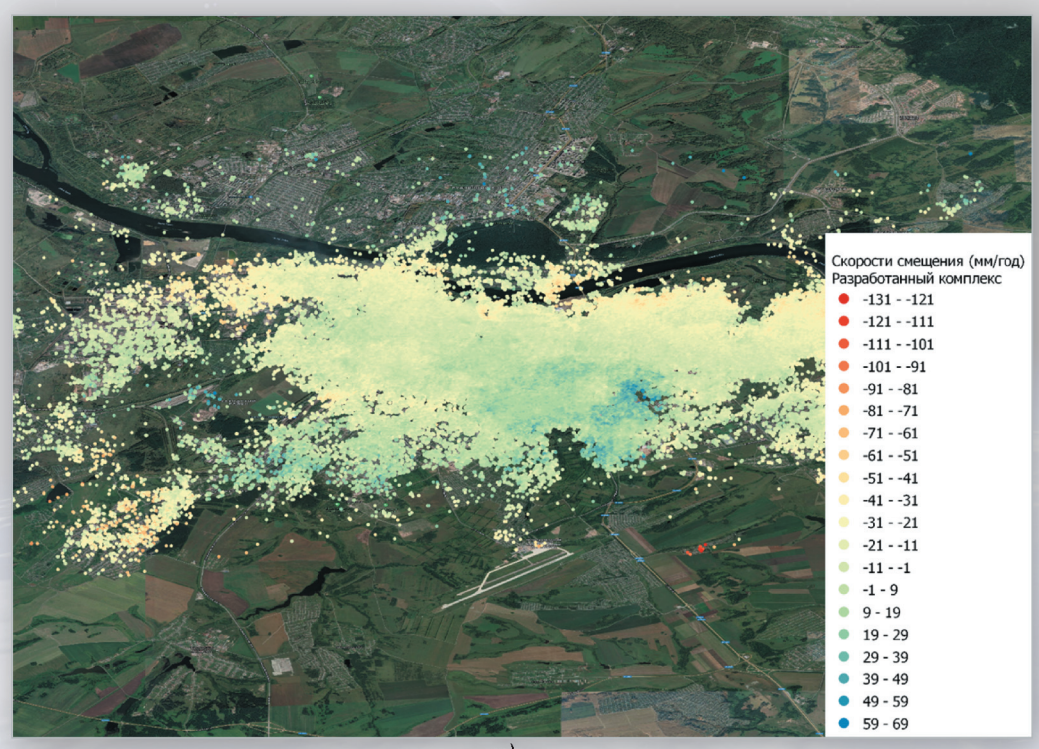


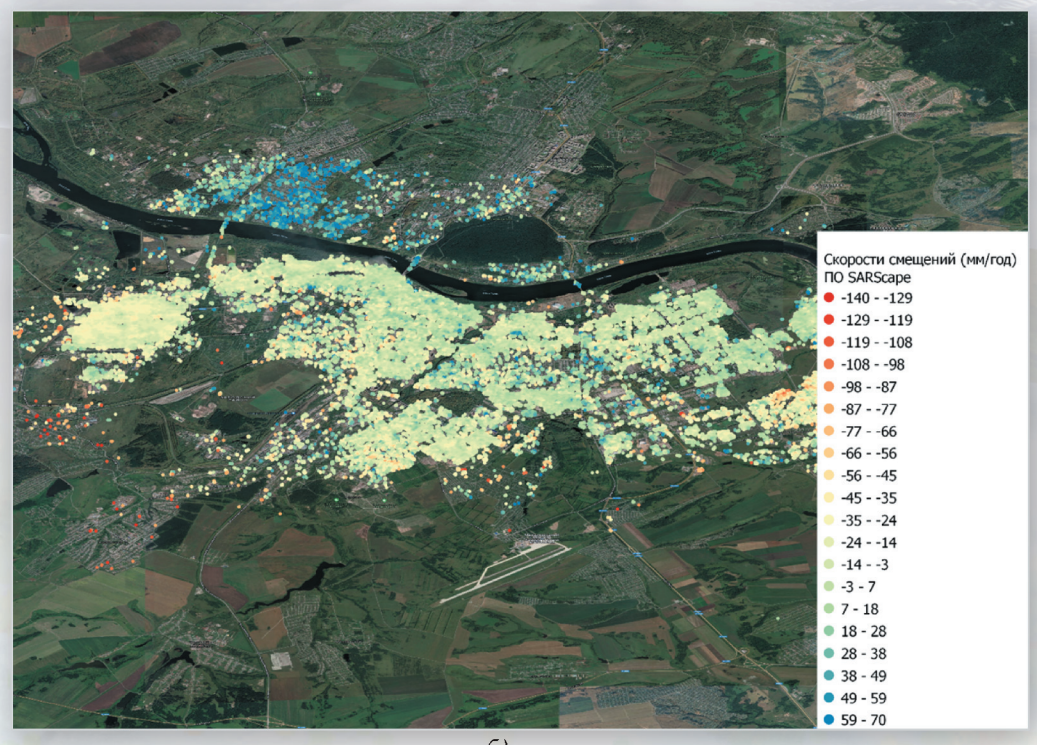
Рис. 14. Графический интерфейс модуля «Результаты (PS)»

Рисунок к статье **С. Е. Попова, Р. Ю. Замараева, Н. И. Юкиной, О. Л. Гиниятуллиной, Л. С. Микова, И. Е. Харлампенкова, Е. Л. Счастливцева**

«ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ РАСЧЕТА ДЕФОРМАЦИЙ ЗЕМНОЙ ПОВЕРХНОСТИ С ИСПОЛЬЗОВАНИЕМ СПУТНИКОВЫХ РАДАРНЫХ ДАННЫХ»



а)



б)

Рис. 16. Результаты расчета скоростей смещений:
а – по схеме метода SBaS в разработанном программном комплексе;
б – методом SBaS в ПО ENVI с модулем SARScape

ISSN 2220-3397. Программная инженерия. 2021. Том 12. № 5. 225 – 280. Подписной индекс по Объединенному каталогу "Пресса России" – 22765