СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1 Методики прогнозирования состояния береговой зоны

Таким образом, для обеспечения возможности проведения научных исследований в области инженерной геологии береговых зон водохранилищ необходимо создать ресурсы хранения информации и вычислительные ресурсы, обеспечивающие продуктивную среду прогнозирования с использованием различных математических моделей. .... модели предназначены для решения задач и ранжируются по видан задач, масштабу исследуемого объекта, размеру интервала времени, степени точности.

2 Проектирование ГИС для прогнозирования береговой зоны

Анализ предметной области, представленный в Главе **??**, показал, что для получения качественно новых результатов научных исследований в инженерной геологии береговых зон внутренних водоемов, необходимо создать ресурсы хранения, преобразования, обеспечения эффективного доступа к этой информации, а также ее визуализации. Данные требования реализуются в виде информационно-вычислительную инфраструктуры, включающей набор сервисов (серверов), взаимодействующих согласно согласованному плану решения задачи.

2.1 Проектирование информационно-вычислительной среды

2.1.1 Функциональные требования

... Необходимо обеспечить доступ к данным из разных ГИС (QGIS, ArcGIS и др.). Современные ГИС позволяют обеспечивать доступ пространственно-распределенным данным, представленным в специальных форматах, данным баз реляционных данных, а также виртуальных географических сервисов.

2.1.2 Архитектура системы

Перечисленные в предыдущем разделе функции формируют собой элементы архитектуры распределенной системы обработки ГИС-данных (рисунок **??**).

Система состоит из двух основных частей, обычно обозначаемых как “клиент” и “сервер”. В нашем случае они взаимодействуют друг с другом по протоколу HTTP и реализуют взаимодействие через интерфейс “архитектуры REST” (Representation State Protocol). QGIS и ArcGIS - это клиентская часть, она включает также процедуры DSAS, ShRec (распознавание контура), ExpEv (экспертная оценка). QGIS также выступает клиентом для сервера изображений (формат JPG, TIFF, JP2 и др.), получаемых, в том числе, с сервера Google Maps.

Серверная часть – набор модулей, запускаемых удаленно из модуля `shore_qgis_module`, т.е. процедурами распознавания управляет полностью клиентская часть. Управление осуществляется через интерфейс REST, где в качестве объектов выступают сервисы хранения исходных изображений (OCV2 + HDF5), распознавания объектов на изображении (SegAny), анализ масок и изображений с целью получения дополнительных характеристик (FeatEx), сервис выгрузки данных с представлением в JSON (внутри REST). Приведем пример реализации интерфейсов REST для загрузки изображений и управления процессом распознавания.

Рисунок 2.1 – Архитектура распределенной системы

2.1.3 Интерфейс REST загрузки и обработки изображений

Задача данного программного объекта - принимать бинарный файл изображения, преобразовать его в матрицы слоев, записать в базу данных на сервере, выдать глобальный идентификатор, при помощи которого далее идентифицируется сохраненные данные изображения.

```
img = Service(name='imgstore',
              path='/sa-1.0/image/{img_name}',
              description="Image collection")

@img.put()
def put_image(request):
    """Принимает бинарные данные картинки,
    возвращает JSON с UUID сохраненного изображения
    """
    name = request.matchdict['img_name']

    imgg, ds = add_image(name, request.body)

    uui = uuidgen()
    uuis = str(uui)
    pth = ds.name
    STORAGE, INGRP, UUIDGRP = storage_begin()
    if name in UUIDGRP:
        ouuis = gs(UUIDGRP[name])
        del UUIDGRP[name]
        del UUIDGRP[ouuis]
    # Отображение UUID <-> имя изображения
    UUIDGRP.create_dataset(name, data=uuis)
    UUIDGRP.create_dataset(uuis, data=name)
    STORAGE, INGRP, UUIDGRP = storage_end()
    return {
        "error": None,
        "ok": True,
        "uuid": uuis,
        "content": pth,
        "name": name,
        "namepath": imgg.name
    }

def add_image(name, content, replace=True):
    """Добавление изображения в БД"""
    nparr = np.frombuffer(content, np.uint8)
    image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
```

```
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Открытие БД
    STORAGE, INGRP, UUIDGRP = storage_begin()
    if name in INGRP:
        del INGRP[name]
    imgg = INGRP.create_group(name)
    ds = imgg.create_dataset('content',
              data=image, compression="lzf")
    log.info("Image '{}' loaded".format(name))
    # Закрытие БД
    STORAGE, INGRP, UUIDGRP = storage_end()
    return (imgg, ds)
```

Загрузка изображения делается при помощи запроса PUT с адресом `https://<адрес сервера>:6543/sa-1.0/image/<имя картинки>`. В тело запроса PUT помещается содержимое изображения. В качестве результата возвращается JSON-ответ, содержащий в поле “uuid” идентификатор сохраненного изображения. Список сохраненных изображений получается запуском запроса GET к этому же адресу без указания имени изображения. Возвращается JSON со списком отображений UUID-<имя файла изображения>. Полученный идентификатор используется для обозначения изображения, подвергающегося процедуре распознавания:

```
sactrl = Service(name='segment-any-control',
                 path='/sa-1.0/sa/{img_uuid}/{cmd}',
                 description="Functions of SA on a image\
                              identified by uuid")
@sactrl.post()
def start_recognition(request):
    # Импорт процедур анализа, выполняющихся в
    #  других процессах
    from ..tasks import (sa_start,
          ANSWERS, rc_set, rc_get, rc_delete,
          rc_update)

    uuids = request.matchdict['img_uuid']
    cmd = request.matchdict['cmd']
    # cmd = 'start'

    STORAGE, INGRP, UUIDGRP = storage_begin()
    # идентификатор изображения существует?
    isimg = uuids in UUIDGRP
    STORAGE, INGRP, UUIDGRP = storage_end()

    rd = {"error": False, "ok": True, "cmd": cmd}
```

```
23
24      if cmd == "flush":
25          ANSWERS.flushdb()
26          return rd
27
28      if not isimg:
29          return {
30              "error": "not found",
31              "ok": False,
32              "uuid": uuids,
33              "cmd": cmd,
34              "processuuid": None
35          }
36      # Команда запуска процесса распознавания
37      if cmd == "start":
38          prevrc = rc_get(uuids)
39          if prevrc is not None:
40              return {
41                  "error": "already running",
42                  "ok": False,
43                  "uuid": uuids,
44                  "cmd": cmd,
45                  "processuuid": prevrc.get("processuuid",
46                                            None),
47                  "ready": prevrc.get("ready", False)
48              }
49          del prevrc
50          rc = {"uuid": uuids, "ready": False}
51          rc_set(uuids, rc)
52          arc = sa_start.delay(uuids)
53          puuid = str(arc.id)
54
55          def _u(r):
56              r["processuuid"] = puuid
57
58          rc = rc_update(uuids, _u)
59          print(rc_get(uuids))
60          puuid = rd["processuuid"] = puuid
61      # Команда проверки завершения процесса
62      elif cmd == "status":
63          rd["ready"] = False
64
```

```
        def _a(v, rr):
            rd["ready"] = v
            rd["result"] = rr.get("result", None)

        rc = rc_get(uuids, "ready", _a)
        if rc is None:
            return {
                "error": "no process",
                "ok": False,
                "uuid": uuids,
                "cmd": cmd,
                "ready": None
            }

        rd["processuuid"] = rc["processuuid"]
    # Команда завершения процесса и удаления его данных
    elif cmd == "finalize":
        rcg = rc_get(uuids)
        if rcg is None:
            return {
                "error": "not running",
                "ok": False,
                "uuid": uuids,
                "cmd": cmd,
                "ready": None
            }
        rc_delete(uuids)
        rd.update({"ready": rcg["ready"],
                   "processuuid": rcg["processuuid"]})
    # Команда отмены процесса
    elif cmd == "discard":
        rcg = rc_get(uuids)
        if rcg is not None:
            return {
                "error": "still running",
                "description":
                "cannot stop SA, wait its finishing. \
                     Use status command.",
                "ok": False,
                "uuid": uuids,
                "cmd": cmd,
                "ready": None
```

```
            }
        STORAGE, INGRP, UUIDGRP = storage_begin()
        name = gs(UUIDGRP[uuids])
        imgg = INGRP[name]
        if "masks" in imgg:
            del imgg["masks"]
            rc = "removed"
        else:
            rc = "no mask"
        STORAGE, INGRP, UUIDGRP = storage_end()
        rd.update({
            "ready": None,
            "processuuid": None,
            "description": rc
        })

    return rd
```

Запуск процесса распознавания осуществляется при помощи запроса POST следующего формата - `http://<адрес сервера>:6543/sa-1.0/sa/<идентификатор изображения>/<команда>`. Командой выступает ключевое слово - одно из четырех:

start – запуск нового процесса,

status – запрос состояния распознавания,

finalize – завершение исполнившегося процесса,

flush – отмена процесса.

Требующие большие вычислительные ресурсы процессы выполняются в отдельных процессах сервера или на специализированных серверах, снабженных аппаратной поддержкой вычислений (CUDA и ему подобным). Реализация запуска таких задач реализована при помощи библиотеки celery языка программирования Python.

3 Реализация подсистем распределенной вычислительной среды

3.1 Тестирование

ЗАКЛЮЧЕНИЕ

Выпускная квалификационная работа посвящена проекту развития научного направления Лаборатории инженерной геологии и геоэкологии в аспекте повышения уровня информатизации и обработки данных полевых исследований. В работе решены следующие задачи:

1. Проведена формализация предметной области инженерной геологии, относящейся к исследованиям экзогенных геологических процессов на берегах крупного внутреннего водоёма, представлена концептуальная модель в виде онтологии,
2. Разработана информационно-вычислительной инфраструктура поддержки прогнозирования состояния береговой зоны на основе геоинформационной системы и распределенной вычислительной среды,
3. Предложен вариант архитектуры вычислительной среды, реализованы некоторые её подсистемы с использованием современных средств автоматизации распознавания объектов на изображениях,
4. Для реализованных подсистем предложены модели данных, предназначенных для хранения информации в процессе её обработки,
5. Продемонстрирована работоспособность предложенной среды на простом примере прогнозирования.

Рассмотренные в диссертации вопросы преследуют целью формирование вычислительных ресурсов поддержки принятия решений по результатам мониторинга, оценки и прогноза опасных геологических процессов. Дальнейшее направление научных исследований и опытно-конструкторских работ имеет смысл продложать по нескольким основным направлениям:

- Завершение реализации информационно-вычислительный среды,
- Наполнение информационных ресурсов архивными данными и данными современного мониторинга объектов исследования,
- Совершенствование методов прогнозирования состояния береговой зоны за счет реализации современного уровня информационного обеспечения,
- Разработка экспертной системы оценки результатов моделирования с целью формирования рекомендаций по использованию конкретных участков исследуемой береговой зоны.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Gruber, T. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human-Computer Studies. 43 (5–6): 907–928. doi:10.1006/ijhc.1995.1081. S2CID 1652449.

   Thomas R. Gruber, Toward principles for the design of ontologies used for knowledge sharing?, International Journal of Human-Computer Studies, Vol. 43, No. 5–6, 1995, Pp. 907-928, ISSN 1071-5819, doi:10.1006/ijhc.1995.1081, https://www.sciencedirect.com/science/article/pii/S1071581985710816 (дата доступа: 11.05.2024)