ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Метод організації еластичних обчислень в автономній системі

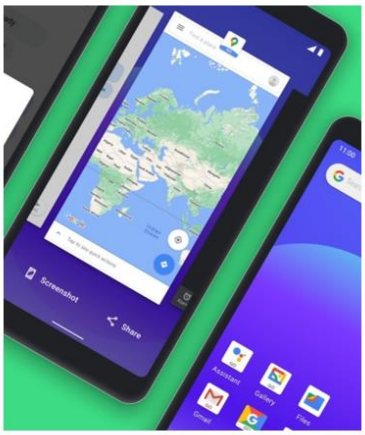
КВАЛІФІКАЦІЙНА РОБОТА

Другий (магістерський)

Автор
Настиченко Т.А.
студ. гр. СПм-22-2

Керівник
Кучук Н.Г.
проф. каф. ЕОМ

Актуальність



Обмеженість обчислювальних ресурсів

- Сучасні процесори розвиваються не в швидкість, а в кількість ядер
- Технологічний прогрес виробництва чіпів підходить до свого максимуму

Мета та завдання

Мета кваліфікаційної роботи полягає в розробці методу організації еластичних обчислень.

Завдання дослідження:

1) Проаналізувати методи організації еластичних обчислень з використанням інтернету речей.

2) Обґрунтувати вибір модель туманних обчислень.

3) Розробити застосунок для еластичних обчислень.

3

Хмарні обчислення

- Вирішення проблеми за допомогою інтернету речей
- Дані завантажуються в хмару для подальшої обробки
- Не підходить у випадках коли потрібна низька затримка

4

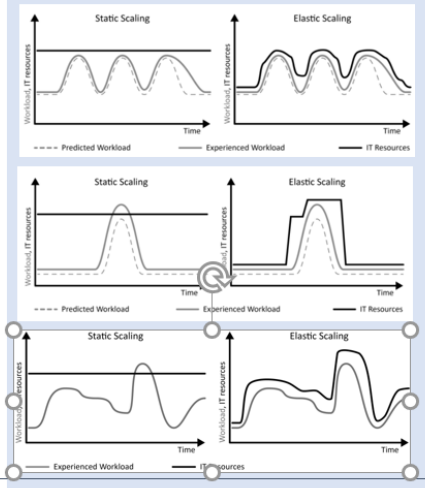
Туманні обчислення

- Низька та передбачувана затримка передачі інформації
- Немає необхідності підтримувати зв'язок із хмарою
- Об'єднання великої кількості вузлів у єдину мережу

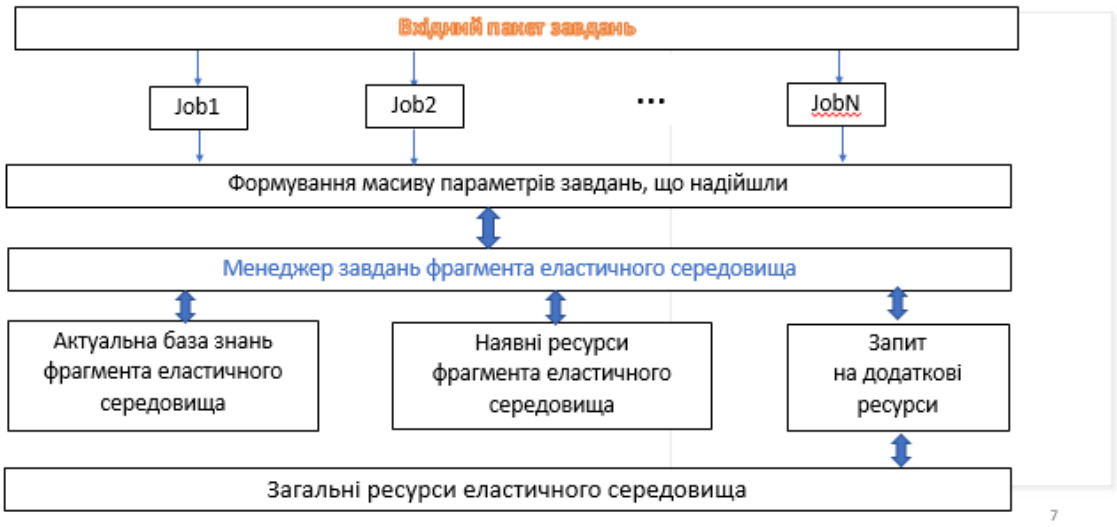
Еластичні обчислення

Еластичні обчислення здатні динамічно підключати нові та відключати старі ресурси, а також підтримувати активним оптимальну кількість ресурсів, яких буде достатньо для виконання потрібної роботи до дедлайну

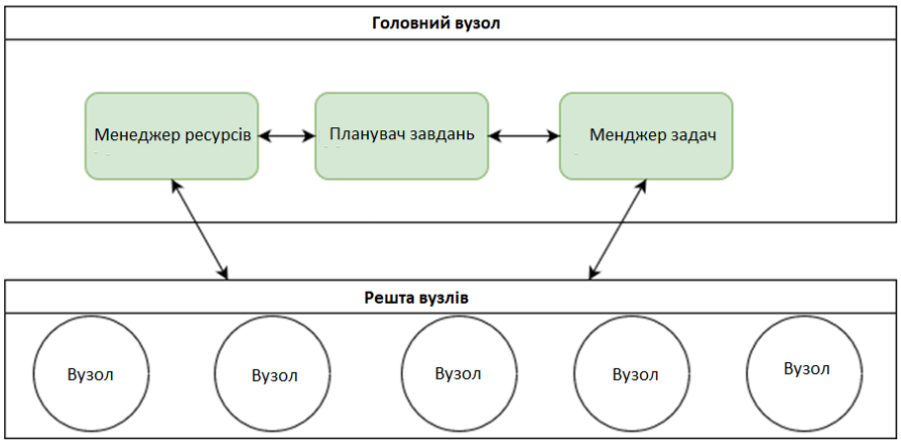
Основні види навантажень



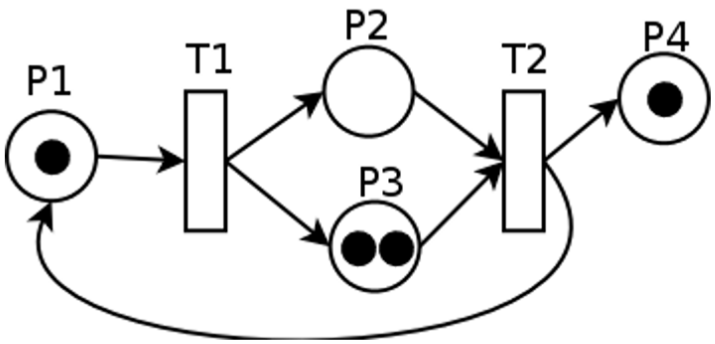
Метод організації еластичних обчислень



Взаємодія між вузлами



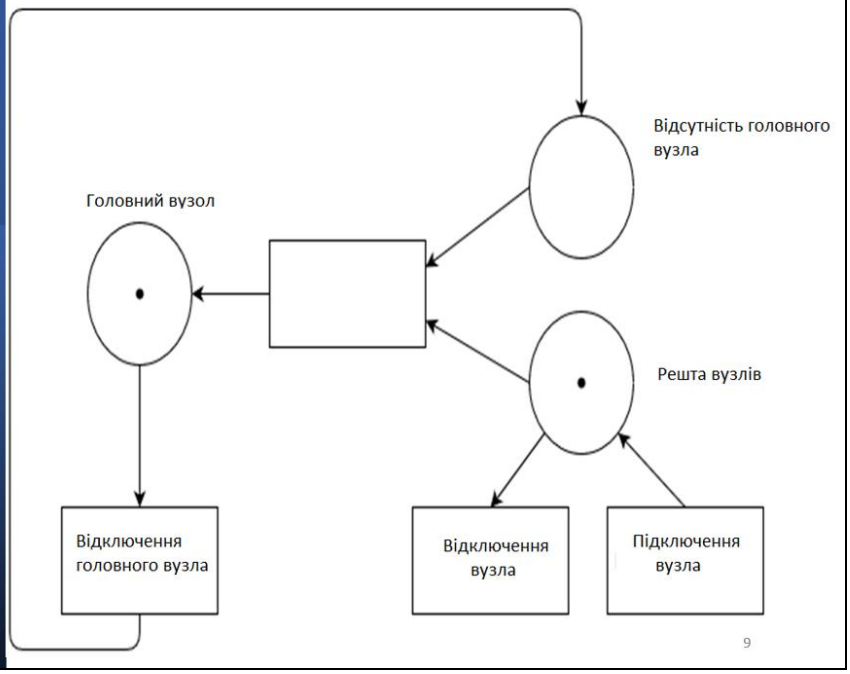
Мережі Петрі



- Складається з позицій та переходів
- Перехід може спрацювати, якщо є фішки в позиціях
- Не може спрацювати 2 переходи одночасно
- Моделювання систем із паралельними взаємодіючими компонентами

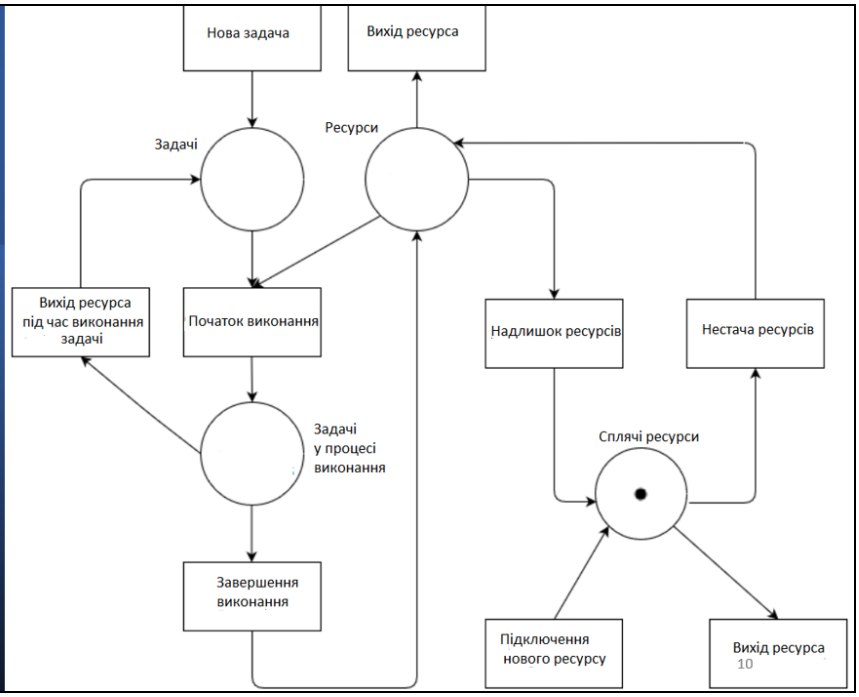
8

Головний вузол до якого підключаються інші, якщо відключен основний вузол, вибирається новий з інших вузлів.



9

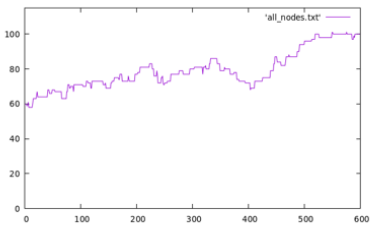
Логіка управління ресурсами за допомогою мереж Петрі на прикладі завдань одного типу

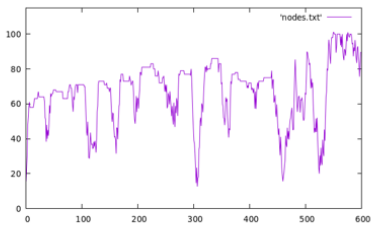


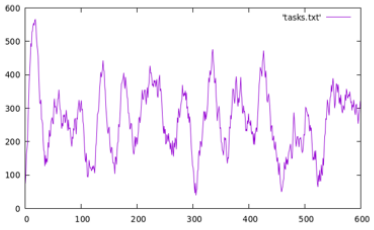
10

Реалізація

- Вебфреймворк flask
- Використання модулів threading та asyncio для паралельності
- Черга завдань реалізована за допомогою модуля queue



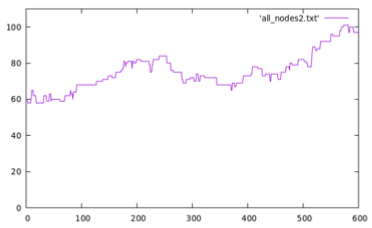


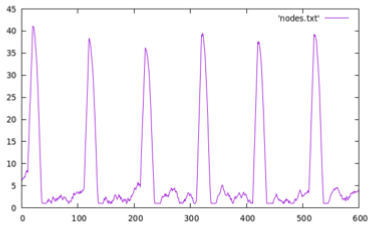


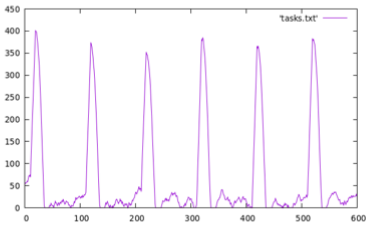
11

Реалізація

- Вебфреймворк flask
- Використання модулів threading та asyncio для паралельності
- Черга завдань реалізована за допомогою модуля queue



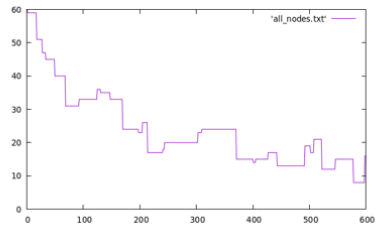


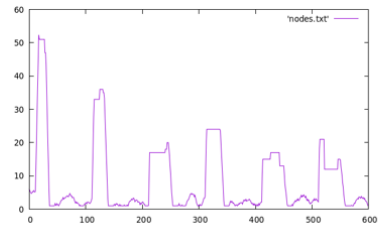


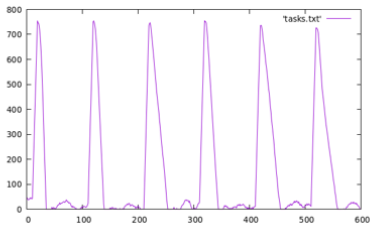
12

Реалізація

- Вебфреймворк flask
- Використання модулів threading та asyncio для паралельності
- Черга завдань реалізована за допомогою модуля queue







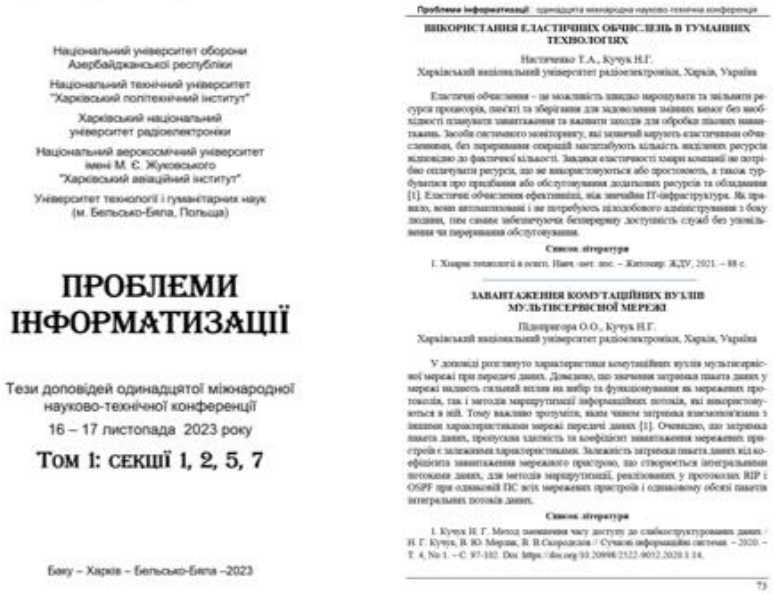
13

Висновки

- Еластичні обчислення дають більшу гнучкість при використанні ресурсів
- Вимагають додатковий оверхед і складніші у реалізації
- Дозволяють вирішувати деякі завдання майже без затримки та без зв’язку з хмарою

14

Апробація результатів кваліфікаційної роботи



Національний університет оборони Азербайджанської республіки

Національний технічний університет "Харківський політехнічний інститут"

Харківський національний університет радіоелектроніки

Національний аерокосмічний університет імені М. Є. Жуковського "Харківський авіаційний інститут"

Університет технології і гуманітарних наук (м. Бельсько-Бяла, Польща)

ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ

Тези доповідей одинадцятої міжнародної науково-технічної конференції

16 – 17 листопада 2023 року

Том 1: секції 1, 2, 5, 7

Баку – Харків – Бельсько-Бяла –2023

Проблеми інформатизації: одинадцята міжнародна науково-технічна конференція

ВИКОРИСТАННЯ ЕЛАСТИЧНИХ ОБЧИСЛЕНЬ В ТУМАННИХ ТЕХНОЛОГІЯХ

Настаченко Т.А., Кучук Н.Г.
Харківський національний університет радіоелектроніки, Харків, Україна

Еластичні обчислення – це можливість швидко нарощувати та звільняти ресурси процесорів, пам'яті та зберігання для задоволення змінних вимог без необхідності планувати завантаження та вживати заходів для обробки пікових навантажень. Засоби системного моніторингу, які зазвичай керують еластичними обчисленнями, без перерв вимикають операції масштабують кількість виділених ресурсів відповідно до фактичної кількості. Завдяки еластичності хмари компанії не потрібно оплачувати ресурси, що не використовуються або простоюють, а також турбуватися про придбання або обслуговування додаткових ресурсів та обладнання [1]. Еластичні обчислення ефективніші, ніж звичайна IT-інфраструктура. Як правило, вони автоматизовані і не потребують цілодобового адміністрування з боку людини, тим самим забезпечуючи безперервну доступність служб без уповільнення чи перерв на обслуговування.

Список літератури

1. Хмарні технології в освіті. Навч.-мет. пос. – Житомир: ЖДУ, 2021. – 88 с.

ЗАВАНТАЖЕННЯ КОМУТАЦІЙНИХ ВУЗЛІВ МУЛЬТИСЕРВІСНОЇ МЕРЕЖІ

Підпригора О.О., Кучук Н.Г.
Харківський національний університет радіоелектроніки, Харків, Україна

У доповіді розглянуто характеристики комутаційних вузлів мультисервісної мережі при передачі даних. Доведено, що значення затримки пакета даних у мережі надають сильний вплив на вибір та функціонування як мережевих протоколів, так і методів маршрутизації інформаційних потоків, які використовуються в ній. Тому важливо зрозуміти, яким чином затримка взаємопов'язана з іншими характеристиками мережі передачі даних [1]. Очевидно, що затримка пакета даних, пропускна здатність та коефіцієнт завантаження мережевих пристроїв є залежними характеристиками. Залежність затримки пакета даних від коефіцієнта завантаження мережевого пристрою, що створюється інтегральними потоками даних, для методів маршрутизації, реалізованих у протоколах BGP і OSPF при однаковій ПС всіх мережевих пристроїв і однаковому обсязі пакетів інтегральних потоків даних.

Список літератури

1. Кучук Н. Г. Метод зменшення часу доступу до слабкоструктурованих даних / Н. Г. Кучук, В. Ю. Мордик, В. В.Скородзієв // Сучасні інформаційні системи. – 2020. – Т. 4, No 1. – С. 97-102. Doi: https://doi.org/10.20998/2522-9052.2020.1.14.

73

16

ДОДАТОК Б

Лістинг програми

```
from variables import *

app = Flask(__name__)
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['MAX_CONTENT_LENGTH'] = 16 * 1000 * 1000
# with app.test_request_context():


for i in jobs_nums:
    tasktype_queues[i] = queue.Queue(jobs_nums[i])  #
{'0':queue.Queue(),'1':queue.Queue(),'2':queue.Queue()}


'''
import importlib
tree = os.listdir('modules')
for i in tree:
    importlib.import_module(i)
'''

trace_count = {'0': safe_int(0), '1': safe_int(0), '2': safe_int(0)}


def self_adr():
    return IP + ':' + str(SOCKET)


def write_to_file(fname, vals, time):
    # for i in vals:
    i = vals[0]
    with open(fname.format(str(i)), 'a') as f:
        f.write(str(time) + ' ')
        f.write(str(i) + ' ')
        f.write('\n')


def work_count(j_types=['0', '1', '2']):
    amount = [0] * len(j_types)
    interval = 3
    while True:

        try:
            tim = time.time() - t_begin
            with open('tasks_amount.txt', 'a') as f:
                f.write(str(tim) + ' ')
                f.write(str(tasks_to_be_done.value()) + ' ')
                f.write('\n')
            for n in node_dict:
                amount[0] = 0
                # response = requests.post('http://'+n+'/get_count')
                # response.raise_for_status()
                # t_count=json.loads(response.text)
                for i in range(len(j_types)):
                    amount[0] = amount[0] + trace_count[j_types[i]].value()
# t_count[j_types[i]]
                write_to_file('{}'.format(n) + '_j{}.txt', amount, tim)  #
```

```
interval*iteration
        except Exception as ex:
            perror("Profile Except")
            template = "An exception of type {0} occurred. Arguments:\n{1!r}"
            message = template.format(type(ex).__name__, ex.args)
            print(message)
        finally:
            time.sleep(0.5)


def fails_time(func, count):
    fails = 0
    while fails < count:
        try:
            return func()
        except:
            perror('failed')
            fails = fails + 1


def create_job_dir(job_num, dirs):
    for dir in dirs:
        path = './jobs/j{}/{}'.format(job_num, dir)
        if not os.path.exists(path):
            try:
                os.makedirs(path)
            except OSError as exc:  # Guard against race condition
                if exc.errno != errno.EEXIST:
                    raise


def main_work(iname, oname):
    with open(iname, 'rb') as file, open(oname, 'wb') as f:
        for line in file:
            f.write(line)
    pinfo(oname)
    time.sleep(1)


@app.route('/check_central')
def check_central():
    return self_adr() == CENTRAL_NODE


def is_central(node):
    fails = 0
    while fails < MAX_FAILS:
        try:
            response = requests.get('http://' + node + '/check_central')
            response.raise_for_status()
            b_resp = bool(response.text)
            return b_resp
        except Exception:
            perror("Disconect Exception")
            fails = fails + 1
    return False


@app.route('/change_central_node/<node>')
def change_central_node(node):
    global CENTRAL_NODE
    CENTRAL_NODE = node
    return CENTRAL_NODE
    """
```

```
    if not is_central(CENTRAL_NODE):

    else:
        logging.warning("Got new Central node, while old one still working.
Keep old one")
        """


def dict_to_j_types(dict_j_types, node=None):
    pinfo(dict_j_types)
    j_types = {}
    for i, it in dict_j_types.items():
        j_types[i] = node_count()
        pinfo(it)
        for k, kt in it.items():
            if not k == node:
                j_types[i].set_node(k, my_queue(kt))
    return j_types


@app.route('/disconnect', methods=['GET', 'POST'])
def disconnect():
    global CENTRAL_NODE
    global jobs_types
    try:
        if request.method == 'GET':
            # global node_available
            # node_available=False
            node = self_adr()
            node_dict.pop(node, None)
            if node == CENTRAL_NODE:
                CENTRAL_NODE = next(iter(node_dict))
                min = node_dict[CENTRAL_NODE]
                for i, it in node_dict.items():
                    if it < min:
                        CENTRAL_NODE = i
                        min = it
            pinfo(CENTRAL_NODE)
            # TODO: Cloud storing central node
            pinfo(node_dict)
            for i in node_dict:
                fails = 0
                while fails < MAX_FAILS:
                    try:
                        response = requests.post('http://' + i +
'/disconnect',
                                                 params={'node': node,
'c_node': CENTRAL_NODE,
                                                         'j_types':
json.dumps(jobs_types, default=lambda o: o.val())})
                        response.raise_for_status()
                        break
                    except Exception as ex:
                        perror("Disconect Exception: {}. Exception type: {}.
Arguments: {!r}".format(fails,

type(ex).__name__,

ex.args))
                        fails = fails + 1
        elif request.method == 'POST':
            node = request.args.get('node')
            c_node = request.args.get('c_node')
            j_types = request.args.get('j_types')
```

```
            jobs_types = dict_to_j_types(json.loads(j_types), node)
            pinfo("Disc: node {}".format(node))
            if node:
                try:
                    node_dict.pop(node, None)
                    pinfo(node_dict)
                except:
                    perror("Disconect: delete node exception. Exception type:
{}. Arguments: {!r}".format(
                        type(ex).__name__, ex.args))

            change_central_node(c_node)
    except Exception as ex:
        perror("Unexpected exception occured on disconnect. Exception type:
{0}. Arguments: {1!r}".format(
            type(ex).__name__, ex.args))
        raise InvalidUsage("Unexpected exception occured on disconnect.
Exception type: {0}. Arguments: {1!r}".format(
            type(ex).__name__, ex.args))
    pinfo(CENTRAL_NODE)
    pinfo(node_dict)
    return 'Disconected {}'.format(node)


@app.errorhandler(InvalidUsage)
def handle_invalid_usage(error):
    response = jsonify(error.to_dict())
    response.status_code = error.status_code
    return response


def collect_function(work):
    global collectData
    try:
        data = work['data']  # json.load(f)
        offset = int(work['offset'])
        for i in range(len(data)):
            collectData.insert(offset + i, data[i])
    except Exception as ex:
        print("Error occured on work:", work)
        template = "An exception of type {0} occurred. Arguments:\n{1!r}"
        message = template.format(type(ex).__name__, ex.args)
        print(message)


def job_failed(job, piece):
    tasks_done[job][piece]['failed'] = True


@app.route('/discard')
def discard():
    job = request.args.get('job')
    piece = request.args.get('piece')
    if job in tasks_done:
        if piece in tasks_done[job]:
            job_failed(job, piece)


# worker[i].exit()
class Worker(threading.Thread):
    def __init__(self, q, other_arg, *args, **kwargs):
        self.q = q
        self.other_arg = other_arg
        super().__init__(*args, **kwargs)
```

```
    def run(self):
        global tasks_done
        while True:
            try:
                work = self.q.get()  # 3s timeout
                print("Got work", work)
                if work['fail'] > 3:
                    pwarning('Discard work:', work)
                    adr = CENTRAL_NODE  # ''+work['ip'] + ':' +
work['socket']
                    url = 'http://' + adr + '/discard'  # socket to 5000?
                    response = requests.post(url, params={'job': work['job'],
'piece': work['piece']})
                elif work['type'] in self.other_arg:
                    type = work['type']
                    trace_count[type].add(1)
                    pinfo(work)
                    adr = CENTRAL_NODE
                    pinfo(CENTRAL_NODE)
                    url = 'http://' + adr + '/result'  # socket to 5000?
                    not_done = False
                    pinfo(tasks_done)
                    with tasks_done_lock:
                        if not (work['job'] in tasks_done):
                            tasks_done[work['job']] = {}
                        elif not work['piece'] in tasks_done[work['job']]:
                            not_done = True
                    if not_done:
                        main_work(work['input_file'], work['result_file'])
                        tasks_done[work['job']][work['piece']] =
work['result_filename']
                    with open(work['result_file'], 'r+') as f:  #
f=work['file']
                        response = requests.post(url, params={'offset':
work['offset'], 'job': work['job'],
                                                              'piece':
work['piece']},
                                                 files={'file':
(work['result_filename'], f)})
                        response.raise_for_status()
                        pinfo(response.content)
                    trace_count[type].add(-1)
                    # If the response was successful, no Exception will be
raised
                else:
                    logging.error("Not mine type")
                    work['fail'] = work['fail'] + 1
                    self.q.put(work)
            except Exception as ex:
                print("Error occured on work:", work)
                template = "An exception of type {0} occurred.
Arguments:\n{1!r}"
                message = template.format(type(ex).__name__, ex.args)
                print(message)
                work['fail'] = work['fail'] + 1
                self.q.put(work)
                if not (type is None):
                    trace_count[type].add(-1)
            finally:
                if close_workers:
                    break
                # response = jsonify(error.to_dict())
                # response.status_code = error.status_code
```

```
                # return response
            # do whatever work you have to do on work
            self.q.task_done()
        pinfo("Worker down")
        down_workers.add(1)


def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


auth_dict = {'adm': 'admin'}


def valid_login(log, pas):
    return auth_dict[log] == pas  # log=='sas' and pas=='asa'


@app.route('/uploads/<name>')
def download_file(name):
    return send_from_directory(app.config["UPLOAD_FOLDER"], name)


tte = 0


async def send_work(work):
    fails = 0
    pinfo('Send')
    while fails < 100:
        try:
            piece, value, shift, type = work['piece'], work['val'],
work['shift'], work['type']
            pinfo('Type {}'.format(type))
            pinfo('Dict {}'.format(jobs_types))
            node = jobs_types[type].get()
            pinfo('Node {}'.format(node))
            if node == None:
                raise NoFreeNodes("Don't get free nodes in timeout")
            url = 'http://' + node + '/upload'
            with open(work['input'], 'rb') as f:
                response = requests.post(url, params=work, files={'file':
(work['input_filename'], f)})
            response.raise_for_status()
            print(response.content)
            jobs_types[type].put_to(node)

            return True
        except HTTPError as ex:
            perror("Error occured on HTTP request:", piece)
            template = "An exception of type {0} occurred. Arguments:\n{1!r}"
            message = template.format(type(ex).__name__, ex.args)
            perror(message)
            if ex.status_code == 404:
                faulty_nodes.put(node)
            else:
                f_count = node_fail_count.get(node)
                if f_count == None:
                    faulty_nodes.put(node)
                if f_count > 3:
                    faulty_nodes.put(node)
                node_fail_count[node].add(1)
        except Exception as ex:
```

```
            perror("Error occured on thread: {}".format(piece))
            perror(ex)
            template = "An exception of type {0} occurred. Arguments:\n{1!r}"
            message = template.format(type(ex).__name__, ex.args)
            perror(message)
            jobs_types[type].put_to(node)
            fails = fails + 1
    return False


async def recv_work(work):
    fails = 0
    global tasks_done
    while fails < 100:
        try:
            job = tasks_done.get(str(work['job']))
            piece = str(work['piece'])
            if job == None:
                break
            with update:
                while True:
                    update.wait()
                    filename = job.get(piece)
                    if filename != None:
                        # trace_count[work['type']].add(-1)
                        tasks_to_be_done.add(-1)
                        return filename
        except Exception as ex:
            print("Error occured on recv work:{}".format(work))
            template = "An exception of type {0} occurred. Arguments:\n{1!r}"
            message = template.format(type(ex).__name__, ex.args)
            print(message)
            fails = fails + 1
    return None


async def job_manage(work, timeout_send=60.0, timeout_recv=120.0):  #
оставить
    done = False
    filename = None
    try:
        pinfo(work['input_filename'])
        response = send_work(work)  # await asyncio.wait_for(send_work(work),
timeout=timeout_send)
        filename = recv_work(work)  # await asyncio.wait_for(recv_work(work),
timeout=timeout_recv)
        response = await response
        pinfo(response)
        filename = await filename
        pinfo("Done recv")
        if filename == 'failed':
            done = False
        else:
            done = True
    except asyncio.TimeoutError as ex:
        print("Task take too long. Arguments:\n{}".format(ex.args))
    except Exception as ex:
        print("Error occured on thread:", piece)
        template = "An exception of type {0} occurred. Arguments:\n{1!r}"
        message = template.format(type(ex).__name__, ex.args)
        print(message)
    finally:
        return done, filename
```

```
def concatenate_files(w_file, r_file, i=0, offset=0):
    if offset == 0:
        with open(w_file, 'a') as outfile:
            with open(r_file, 'r') as infile:
                shutil.copyfileobj(infile, outfile)
    else:
        pass


async def wait_job_done(work, shift=0, close_loop=True):
    pinfo("Waiting for job to be done")
    done = True
    messed_tasks = []
    work_directory = './jobs/j{}/'.format(work['job'])
    input_dir = work_directory + 'inputs/'
    done_job_dir = work_directory + 'results/'
    pinfo(done_job_dir)
    try:
        job_filename = done_job_dir + 'job{}.txt'.format(work['job'])
        results = list()
        work_amount = work['amount']
        for i in range(work_amount):
            work['piece'] = i
            work['input_filename'] = 'i_job{}_{}.txt'.format(work['job'], i)
            work['result_filename'] = 'r_job{}_{}.txt'.format(work['job'], i)
            # results.append(job_manage(work))
            pinfo(i)
            results.append(asyncio.create_task(job_manage(copy.copy(work))))
        for i in range(work_amount):
            try:
                result = await results[i]
                # result=results[i]
                if result[0]:

                    task_filename = result[1]
                    concatenate_files(job_filename, task_filename)  #
i,offset
                else:
                    print("Task {} not complete".format(i))
                    messed_tasks.append(i)
                    done = False
            except Exception as ex:
                print("Error occured on job piece:", i)
                template = "An exception of type {0} occurred.
Arguments:\n{1!r}"
                message = template.format(type(ex).__name__, ex.args)
                print(message)
                done = False
                messed_tasks.append(i)
        print("Job done")
        return done, messed_tasks
    except Exception as ex:
        print("Error occured on job: {}", work['job'])
        template = "An exception of type {0} occurred. Arguments:\n{1!r}"
        message = template.format(type(ex).__name__, ex.args)
        print(message)
        done = False
    finally:
        return done, messed_tasks


def do_job(job):
    global tasks_done
```

```
    job_input[job['job']] = job
    asyncio.run(wait_job_done(job))


@app.route('/start_job', methods=['GET', 'POST'])
def start_job():
    global tte
    global cur_node
    global job_num
    global tasks_done
    if request.method == 'POST':
        # job_loop = asyncio.get_event_loop()
        cur_job = job_num.add(1)
        value = request.args.get('value')
        dir_list = request.args.get('dir_list')
        if dir_list == None:
            dir_list = STD_JOB_DIR_LIST
        tasks_done[str(cur_job)] = {}
        work_amount = request.args.get('amount')
        type = request.args.get('type')
        work = {}
        work['val'] = value
        work['amount'] = work_amount
        work['shift'] = '2'
        work['job'] = cur_job
        work['type'] = type
        work['adr'] = '' + self_adr()
        work['offset'] = 0
        work['socket'] = SOCKET
        work['input'] = 'random.txt'
        create_job_dir(cur_job, dir_list)
        tasks_to_be_done.add(work_amount)
        threading.Thread(target=do_job, args=[work]).start()
        return redirect(url_for('result', name=value))
    return '''
    <!doctype html>
    <title>Upload new File</title>
    <h1>Upload new File</h1>
    <form method=post>
      <label for="fname">Value:</label><input type="text" id="value"
name="fname"><br><br>
      <label for="amount">Amount:</label><input type="text" id="amount"
name="fname"><br><br>
      <label for="type">Type:</label><input type="text" id="type"
name="fname"><br><br>
      <input type=submit value=Submit>
    </form>
    '''


@app.route('/get_count', methods=['GET', 'POST'])
def get_count():
    temp = {}
    for i, v in trace_count.items():
        temp[i] = v.value()
    return json.dumps(temp)


dataUpdated = False


@app.route('/show_nodes', methods=['GET', 'POST'])
def show_nodes():
    pinfo(str(jobs_types))
```

```
    return json.dumps(jobs_types, default=lambda o: o.val())


@app.route('/new_node', methods=['GET', 'POST'])
def new_node():
    global node_num
    global node_dict
    global jobs_types
    temp = dict(request.args)
    print(temp)
    adr = request.remote_addr + ':' + request.args.get('socket')
    task_types = request.args.get('task_types')
    if task_types is None:
        raise InvalidUsage("Task types not provided")
    task_types = json.loads(task_types)
    pinfo(task_types)
    for i in task_types:
        jobs_types[i].set_node(adr, my_queue(task_types[i]))
    pinfo(jobs_types)
    node_dict[adr] = node_num.add(1)
    return json.dumps(node_dict)


@app.route('/connect', methods=['GET', 'POST'])
def connect():
    global cur_node
    global jobs_nums
    global node_dict
    if request.method == 'GET':
        fails = 0
        while fails < 10:
            try:
                adr = CENTRAL_NODE
                url = r'http://' + adr + r'/new_node'
                response = requests.post(url, params={'task_types':
json.dumps(jobs_nums), 'socket': SOCKET})
                response.raise_for_status()
                pinfo(response.text)
                node_dict = json.loads(response.text)  # TODO
                pinfo(node_dict)
                cur_node = node_dict[self_adr()]
                pinfo(cur_node)
                return str(cur_node)
            except HTTPError as http_err:
                print(f'HTTP error occurred: {http_err}')
            except Exception as ex:
                perror("Error occured on connection. Number of
failes:{}".format(fails))
                template = "An exception of type {0} occurred.
Arguments:\n{1!r}"
                message = template.format(type(ex).__name__, ex.args)
                print(message)
                time.sleep(1)
            finally:
                fails = fails + 1
        raise InvalidUsage("Failed To Connect", status_code=500)
    elif request.method == 'POST':
        cur_node = request.form['node']
    return str(cur_node)


@app.route('/upload', methods=['GET', 'POST'])
def upload():
    global tte
```

```
    global cur_node

    if request.method == 'POST':
        # check if the post request has the file part
        work = {}
        for i in request.args:
            work[i] = request.args.get(i)
        work_directory = './jobs/j{}/'.format(work['job'])
        input_dir = work_directory + 'inputs/'
        done_job_dir = work_directory + 'results/'
        """
        work['type']=request.args.get('type')
        work['ip']= request.remote_addr
        work['adr']=request.args.get('adr')
        work['offset']=request.args.get('offset')
        work['socket']=request.args.get('socket')
        work['job']=request.args.get('job')
        work['result_filename']=request.args.get('result_filename')
        work['input_filename']=request.args.get('input_filename')
        """
        pinfo(work['result_filename'])
        """
        for i in work:
            if work[i]==None:
                print('Parametr "{}" not found'.format(i))
                raise InvalidUsage('Parametr "{}" not found'.format(i))
        """
        if tasktype_queues[work['type']].full():
            raise InvalidUsage('Queue is full', status_code=503)
        file = request.files['file']
        if file.filename == '':
            raise InvalidUsage('No file found')
        if not file:
            raise InvalidUsage("Can't open file", status_code=506)
        work['input_file'] = os.path.join(input_dir, work['input_filename'])
        work['result_file'] = os.path.join(done_job_dir,
work['result_filename'])
        file.save(work['input_file'])
        work['fail'] = 0
        cur_job = work['job']
        dir_list = STD_JOB_DIR_LIST
        create_job_dir(cur_job, dir_list)
        tasktype_queues[work['type']].put(work)
        # tasksQueue.put(work)
        return str(cur_node)
        """
        file = request.files['file']
        # If the user does not select a file, the browser submits an
        # empty file without a filename.
        if file.filename == '':
            print("No save")
            flash('No selected file')
            return redirect(request.url)
        if file and allowed_file(file.filename):
            print("I save")
            tte=(tte+1)%4
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            return redirect(url_for('download_file', name=filename))
        print("Empty")
        """
    return '''
    <!doctype html>
    <title>Upload new File</title>
```

```
    <h1>Upload new File</h1>
    <form method=post enctype=multipart/form-data>
      <input type=file name=file>
      <input type=submit value=Upload>
    </form>
    '''


@app.route('/result', methods=['GET', 'POST'])
def result():
    global dataUpdate
    global tasks_done
    if request.method == 'POST':
        # check if the post request has the file part
        if 'file' not in request.files:
            raise InvalidUsage('No file part', status_code=500)
        file = request.files['file']
        # If the user does not select a file, the browser submits an
        # empty file without a filename.
        if file.filename == '':
            raise InvalidUsage('No file attached', status_code=500)
        if file and allowed_file(file.filename):
            print("I save")
            filename = file.filename
            work = {}
            # work['offset']=request.args.get('offset')
            # work['filename']=file
            # work['data']=file.read()
            work['job'] = request.args.get('job')
            work['piece'] = request.args.get('piece')
            print(tasks_done)
            with update:
                td = tasks_done.get(work['job'])
                if td == None:
                    print("Got job:{}".format(work['job']))
                    return "Job already done"
                pinfo(td)
                piece_num = td.get(work['piece'])
                # if piece_num != None:
                #    return "Task already done"
                work_directory = './jobs/j{}/results/'.format(work['job'])
                filename = os.path.join(work_directory,
secure_filename(filename))

                file.save(filename)
                td[work['piece']] = filename  # TODO
                pinfo(tasks_done)
                pinfo(filename)
                update.notify_all()
            return "Got it"  # redirect(url_for('result', name=filename))
        pinfo("Empty")
    # Should be image print
    # while dataUpdate:
    #    time.sleep(0.5)
    return '''
    <!doctype html>
    <title>Upload new File</title>
    <h1>Upload new File</h1>
    <form method=post enctype=multipart/form-data>
      <input type=file name=file>
      <input type=submit value=Upload>
    </form>
    '''
```

```
@app.route('/login', methods=['POST', 'GET'])
def login():
    error = None
    if request.method == 'POST':
        if valid_login(request.args.get('username'),
                       request.args.get('pass')):
            return 'suc'
        else:
            return 'Invalid username/password'
    return "It's get"


@app.route("/my_name/<name>")
def my_name(name):
    pass


def printer_task():
    print('Im thread')


from multiprocessing import Process


def foreverloop(loop):
    print("Loop started")
    try:
        loop.run_forever()
    finally:

        pending = all_tasks(loop)
        loop.run_until_complete(asyncio.gather(*pending))
        loop.close()


if __name__ == '__main__':
    """
    job_loop = asyncio.get_event_loop()
    threading.Thread(target=foreverloop,args=[job_loop]).start()
    task_loop = asyncio.get_event_loop()
    threading.Thread(target=foreverloop,args=[task_loop]).start()
    loop.create_task(hel_world(4))
    loop.create_task(hel_world(3))
    my_l=threading.Lock()
    my_l2=threading.Lock()
    tmp=[None for _ in range(100)]
    for i in range(100):
        if i%2==0:
            lock = my_l
        else:
            lock = my_l2

threading.Thread(target=tets.add_val,args=('127.0.0.1',1,lock)).start()
    print(tets.add_val('127.0.0.1',1,my_l))
    print(tets['127.0.0.1'])
    print(tasksQueue.qsize())
    for i in range(1):
        JobWorker(jobQueue, [0,1,2]).start()
    """
    SOCKET = int(input('SOCKET='))
    t_begin = time.time()
    dirs = ['inputs', 'results']
    create_job_dir(0, dirs)
```

```
    if SOCKET == 5000:
        print("Start trace")
        threading.Thread(target=work_count).start()
    logging.basicConfig(level=logging.INFO,
format='\t{%(pathname)s:%(lineno)d}%(process)d-:%(message)s')
    logging.info('ce')
    WORKS_TYPES = {'0': 10, '1': 5, '2': 5}
    for tsk in WORKS_TYPES:
        for _ in range(WORKS_TYPES[tsk]):
            Worker(tasktype_queues[tsk], WORKS_TYPES).start()
    app.run(port=SOCKET)
    # app.run(host="0.0.0.0", port=int("8000"), debug=True)
    """
    d1 = threading.Thread(target=flask_task)
    d2 = threading.Thread(target=printer_task)
    d2.start()
    d1.start()
    d2.join()
    d1.join()
    """
```

variables.py

```
from flask import Flask,request, url_for, flash,redirect
from flask import abort,send_from_directory
from flask import render_template

import threading
import queue
import os
import time
import requests
from requests.exceptions import HTTPError
from asyncio.tasks import all_tasks

from werkzeug.utils import secure_filename
from markupsafe import escape

import asyncio

from flask import jsonify
import logging
from inspect import currentframe
from logging import info as pinfo
from logging import error as perror
import shutil
import errno
import pkgutil
import json
import copy
class Object:
    def toJSON(self):
        return json.dumps(self, default=lambda o: o.__dict__,
            sort_keys=True, indent=4)

class InvalidUsage(Exception):
    status_code = 400
    def __init__(self, message, status_code=None, payload=None):
        Exception.__init__(self)
        self.message = message
        if status_code is not None:
            self.status_code = status_code
        self.payload = payload
    def to_dict(self):
        rv = dict(self.payload or ())
```

```
        rv['message'] = self.message
        return rv
class NoFreeNodes(Exception):
    __name__='NoFreeNodes'
    def __init__(self, args):
        Exception.__init__(self)
        self.args = args


class my_queue:
    def __repr__(self):
        return str(self.maxsize)
    def __str__(self):
        return str(self.maxsize)
    def __dict__(self):
        return str(self.maxsize)
    def val(self):
        return self.maxsize
    def __init__(self, maxsize=0):
        self.maxsize=int(maxsize)
        self.cur=int(maxsize)
        self.mutex = threading.Lock()
        self.not_empty = threading.Condition(self.mutex)
        self.not_full = threading.Condition(self.mutex)
        self.all_tasks_done = threading.Condition(self.mutex)
        self.unfinished_tasks = 0
    def _qsize(self):
        return self.cur
    def _add(self,val=1):
        temp =self.cur
        self.cur=self.cur+val
        return temp
    def empty(self):
        return self.cur==0
    def full(self):
        return self.cur==self.maxsize
    def put(self, item=1, block=True, timeout=None):
        with self.not_full:
            if self.maxsize > 0:
                if not block:
                    if self._qsize() >= self.maxsize:
                        raise Full
                elif timeout is None:
                    while self._qsize() >= self.maxsize:
                        self.not_full.wait()
                elif timeout < 0:
                    raise ValueError("'timeout' must be a non-negative
number")
                else:
                    endtime = time.time() + timeout
                    while self._qsize() >= self.maxsize:
                        remaining = endtime - time()
                        if remaining <= 0.0:
                            raise queue.Full
                        self.not_full.wait(remaining)
            self._add(item)
            self.unfinished_tasks += 1
            self.not_empty.notify()
    def get(self, block=True, timeout=None):
        with self.not_empty:
            if not block:
                if not self._qsize():
                    raise Empty
            elif timeout is None:
```

```
                while not self._qsize():
                    self.not_empty.wait()
            elif timeout < 0:
                raise ValueError("'timeout' must be a non-negative number")
            else:
                endtime = time.time() + timeout
                print(endtime)
                while not self._qsize():
                    print(endtime)
                    remaining = endtime - time.time()
                    if remaining <= 0.0:
                        raise queue.Empty
                    self.not_empty.wait(remaining)
            item = self._add(-1)
            self.not_full.notify()
            return item
class node_count(dict):
    def __init__(self, dictionary={}, timeout=0):
        print('Tout {}'.format(timeout))
        self.timeout=timeout
        self.node_update = threading.Condition(threading.Lock())
    def set_node(self,node,val):
        with self.node_update:
            self[node]=val
            pinfo(val)
            pinfo(self)
            self.node_update.notify_all()
    def put_to(self,node,val=1):
        with self.node_update:
            self[node].put(val)
            self.node_update.notify_all()
    def get(self):
        try:
            endtime = time.time() + self.timeout
            with self.node_update:
                while True:
                    for node in self:
                        pinfo(node)
                        try:
                            if self[node].empty():
                                continue
                            self[node].get(timeout=0.01)
                            pinfo("got_node")
                            return node
                        except queue.Full:
                                pinfo('{} node is full, procceding to next
node'.format(node))
                        except Exception as ex:
                            print("Node skipped:",self)
                            template = "An exception of type {0} occurred on
node. Arguments:\n{1!r}"
                            message = template.format(type(ex).__name__,
ex.args)
                            print(message)
                    if self.timeout != 0:
                        remaining = endtime - time()
                        if remaining <= 0.0:
                            raise NoFreeNodes
                        self.node_update.wait(remaining)
                    else:
                        self.node_update.wait()
        except Exception as ex:
            perror("Error occured on get() in node_count")
            template = "An exception of type {0} occurred. Arguments:\n{1!r}"
```

```
            message = template.format(type(ex).__name__, ex.args)
            print(message)
        return None
class safe_int(dict):
    def __repr__(self):
        return ''+self.val
    def __str__(self):
        return ''+self.val
    def __init__(self, val):
        self.lock=threading.Lock()
        self.val=val
    def value(self):
        return self.val
    def add(self,val):
        with self.lock:
            temp = self.val
            self.val=self.val+val
            return temp


MAX_FAILS=3

tasks_to_be_done=safe_int(0)
jobs_nums={'0':10,'1':5,'2':0}
MAX_QUEUE_SIZE=1
WORKERS_NUM=10
CENTRAL_NODE='127.0.0.1:5000'
IP='127.0.0.1'
STD_JOB_DIR_LIST=['inputs','results','modules']
SOCKET=5000
#CENTRAL_URL='http:\\'+CENTRAL_IP+':'+CENTRAL_SID
UPLOAD_FOLDER = '/mnt/d/Dipl/Source/Upload'
ALLOWED_EXTENSIONS = {'txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif',
'ini','data'}
JOB_TYPES=['0','1','2']

jobQueue=queue.Queue()
faulty_nodes=queue.Queue()
collectData=[]

node_num=safe_int(0)
cur_node=-1
node_dict={}#'127.0.0.1:5000':0}
MAX_QUEUE_SIZE_WORK0=10
MAX_QUEUE_SIZE_WORK1=5
MAX_QUEUE_SIZE_WORK2=5
recv_mutex=threading.Lock()
update=threading.Condition(recv_mutex)
faulty_update=threading.Condition(threading.Lock())
tasks_done_lock=threading.Lock()
dict_nodes0=node_count()#'127.0.0.1:5000':queue.Queue(MAX_QUEUE_SIZE),'127.0.
0.1:5001':queue.Queue(MAX_QUEUE_SIZE)}
dict_nodes1=node_count()#'127.0.0.1:5000':my_queue(MAX_QUEUE_SIZE_WORK1),'127
.0.0.1:5001':queue.Queue(MAX_QUEUE_SIZE_WORK1)})
dict_nodes2=node_count()#'127.0.0.1:5000':queue.Queue(MAX_QUEUE_SIZE_WORK2),'
127.0.0.1:5001':queue.Queue(MAX_QUEUE_SIZE_WORK2)}
#словарь в котором храняться очереди для всех работ
jobs_types={'0':dict_nodes0,'1':dict_nodes1,'2':dict_nodes2}
tasktype_queues={}

node_fail_count={}
tasks_done={}#'0':{2:'job0_2.txt',1:'job0_1.txt'}
job_num=safe_int(0)
#job_class=JobType({0:all_nodes0,1:all_nodes1,2:all_nodes2})
```

```
t_begin=0

close_workers=False
down_workers=safe_int(0)
job_input={}
close_workers=False
```