# Исследование производительности MongoDB для обработки больших векторных геоданных

**Работу выполнил:**

студент 3 курса 163 группы ПМИ ФКН Иванашев Илья

**Научный руководитель:**

Доцент департамента программной инженерии
Родригес Залепинос Рамон Антонио

**НУГ Геоинформатики**

http://geolab.gis.land/

# Постановка задачи

- Объект исследования: векторные геопространственные данные и система MongoDB
- Предмет исследования: производительность системы MongoDB
- Цель исследования: выделение сильных и слабых сторон системы MongoDB и определение способов ее оптимального использования для работы с теми или иными данными и запросами.

# Система MongoDB

- MongoDB – популярная документоориентированная NoSQL СУБД
- Поддерживает геопространственные запросы
- Имеет три типа индексов для работы с геоданными
- Документы сохраняются в формате JSON

# Python API

- У MongoDB имеется Python API – PyMongo
- В работе используется PyMongo версии 3.7.2
- База данных представляется отдельным объектом, для проведения запросов используются методы этого объекта:

```
client = MongoClient(
    "mongodb://login:password@localhost:27017",
    replicaset="replicaset",
)
db = client.geo_db
```

# OpenStreetMap

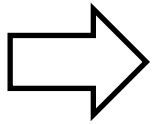
- OpenStreetMap – открытый картографический проект
- Устроен по принципу вики: изменения в карту могут вноситься любым зарегистрированным пользователем
- Позволяет получать геоданные через API

# Overpass API

- Позволяет получать данные в формате OSM XML
- Данные включают в себя точки, пути и отношения
- Для получения данных используется самый простой запрос, позволяющий извлечь объекты в заданном прямоугольнике: https://overpass.kumi.systems/api/interpreter?data=(node(55.7409, 37.5970, 55.7690, 37.6506);way(55.7409, 37.5970, 55.7690, 37.6506);relation(55.7409, 37.5970, 55.7690, 37.6506););out meta;

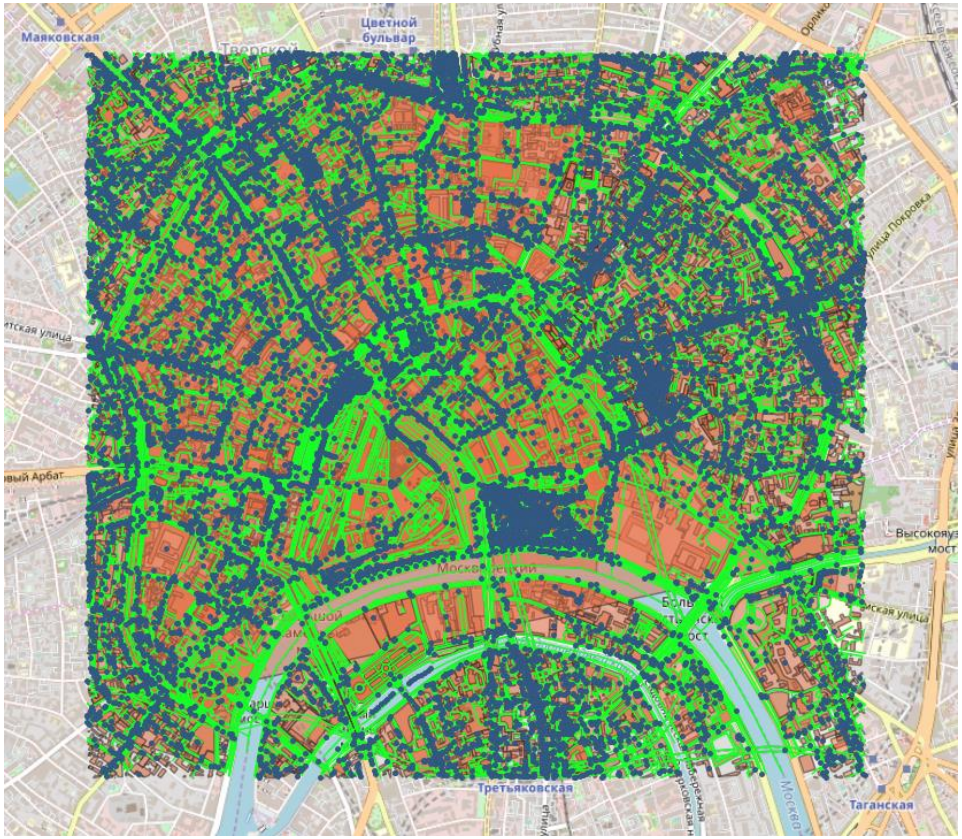
# Форматы OSM XML и GeoJSON

```xml
<node
    id="35881070"
    lat="55.7595345"
    lon="37.6247860"
    version="13"
    timestamp="2019-02-07T20:57:01Z"
    changeset="67007226"
    uid="830106"
    user="literan"
/>
```



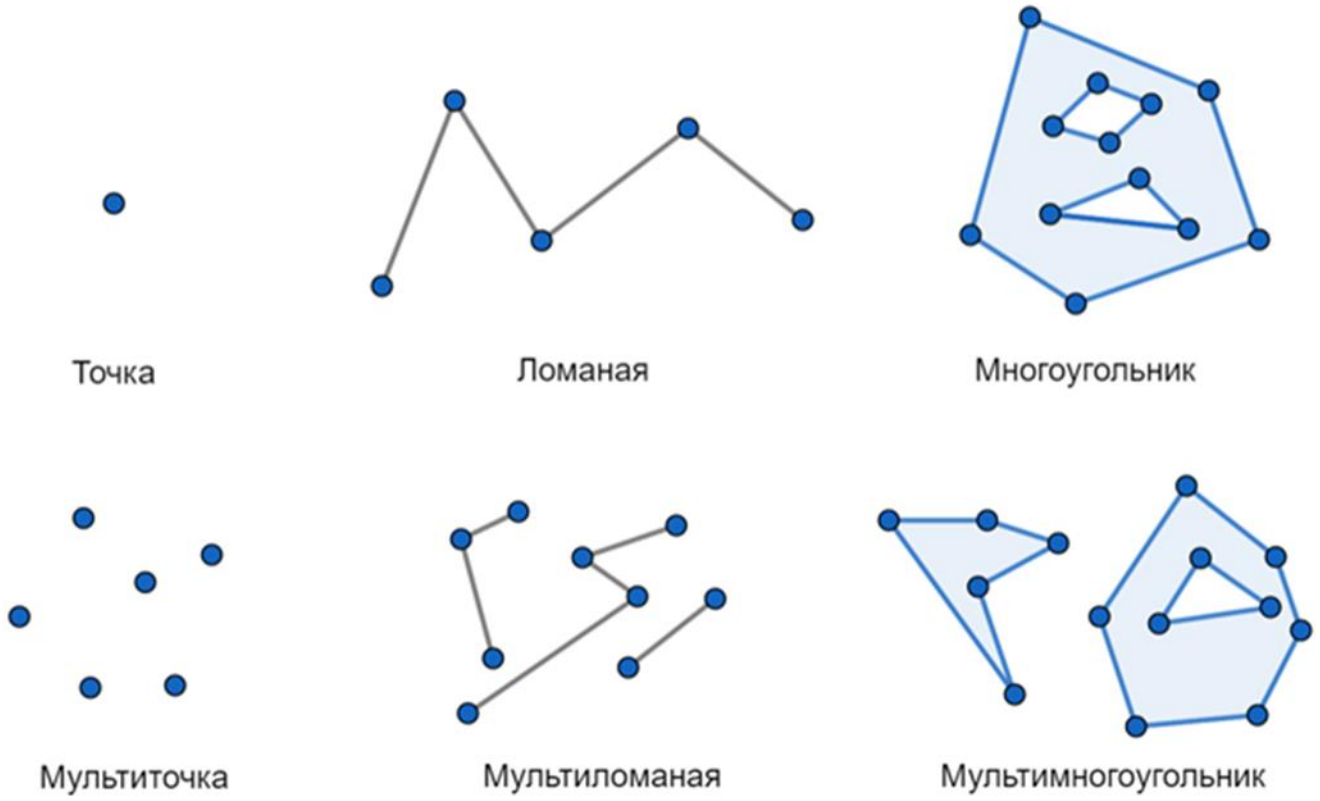
```geojson
{
    "type": "Feature",
    "properties": {
      "id": "35881070",
      "version": "13",
      "timestamp": "2019-02-07T20:57:01Z",
      "changeset": "67007226",
      "uid": "830106",
      "user": "literan"
    },
    "geometry": {
      "type": "Point",
      "coordinates": [
        37.624786,
        55.7595345
      ]
    }
}
```

# Визуализация данных




- 180052 точек (66,9 МБ)
- 40399 ломаных (26,9 МБ)
- 10870 многоугольников (15,04 МБ)

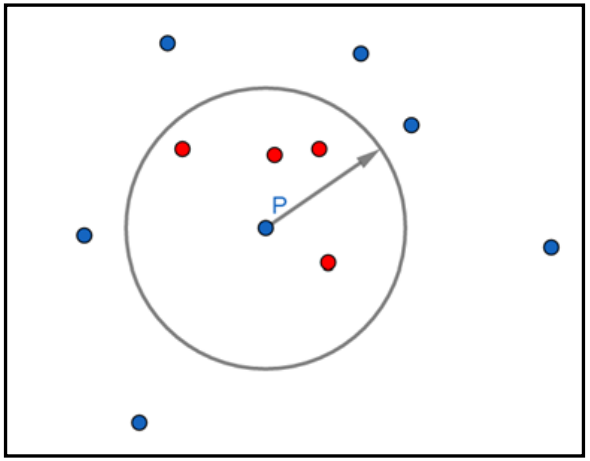
# Рассматриваемые типы векторных данных



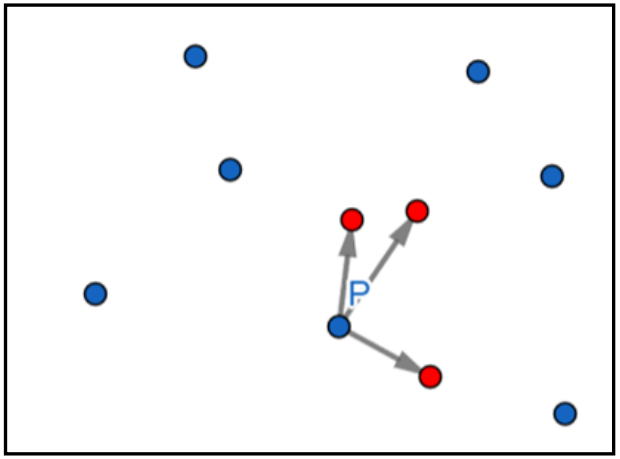
# Экспериментальная среда

- Тестирование проводится в облачном кластере на платформе Microsoft Azure
- Тип кластера – реплицированный
- Проводится сравнение работы на кластерах с двумя и тремя узлами
- Характеристики узлов кластера:
  - 2GB оперативной памяти
  - Одно ядро
  - MAX IOPS: 800

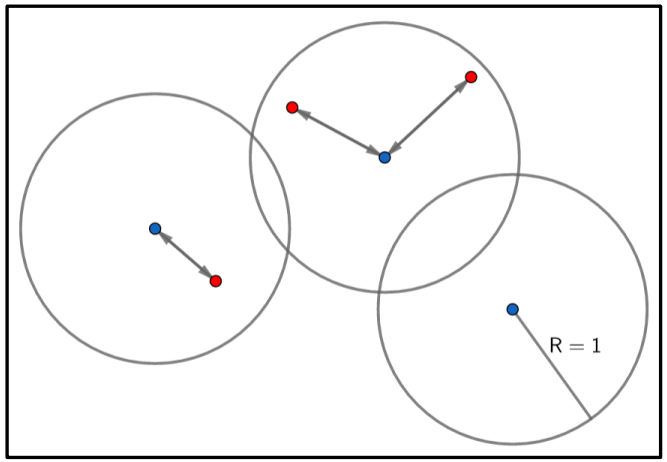
# Рассматриваемые типы запросов




*Range query*




*kNN query*




*Spatial join*

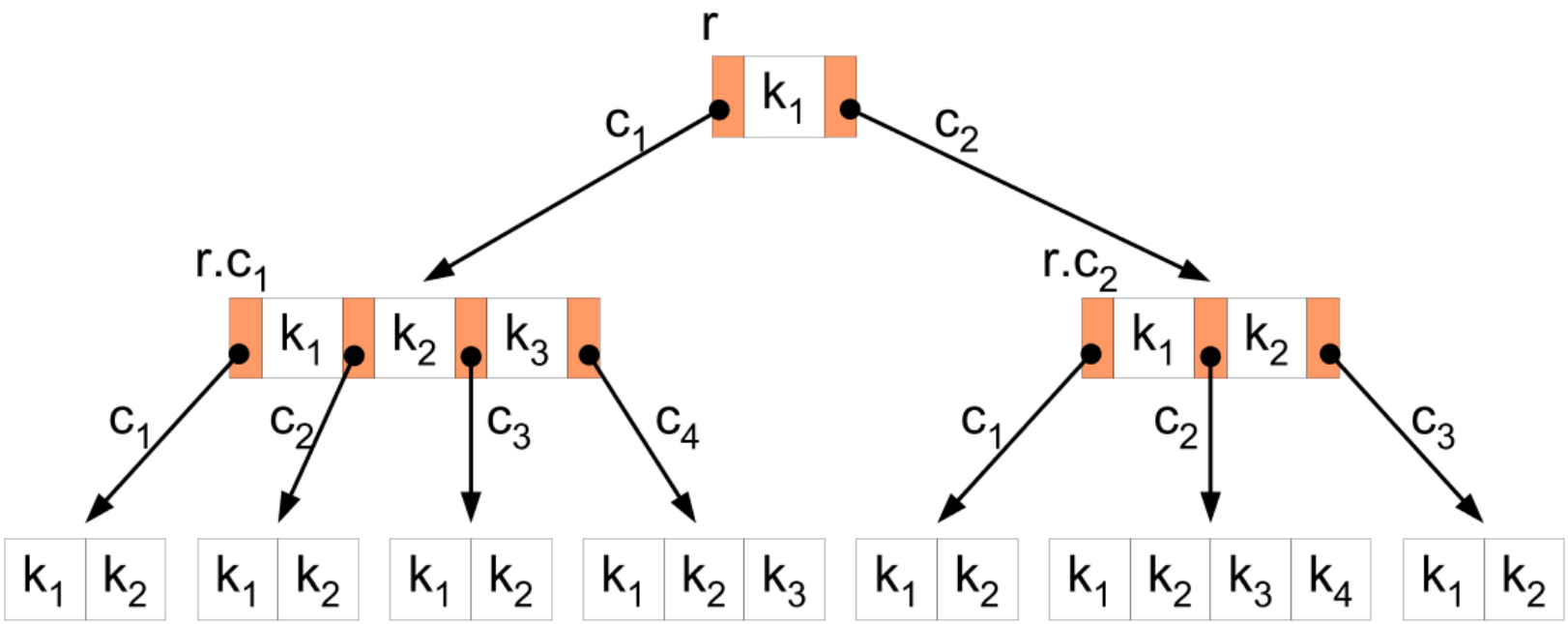
# Тестируемые индексы

- 2d Index – индекс для работы с объектами на плоскости, также ограниченно поддерживающий запросы для работы с объектами на сфере. Позволяет регулировать величину геохеша.
- 2dsphere Index – аналогичный индекс для работы с объектами на земной сфере.
- geoHaystack Index – индекс, оптимизированный для быстрого поиска объектов, расположенных на малом расстоянии от заданной точки. Для этого, объекты разбиваются на «корзины» радиуса bucketSize.

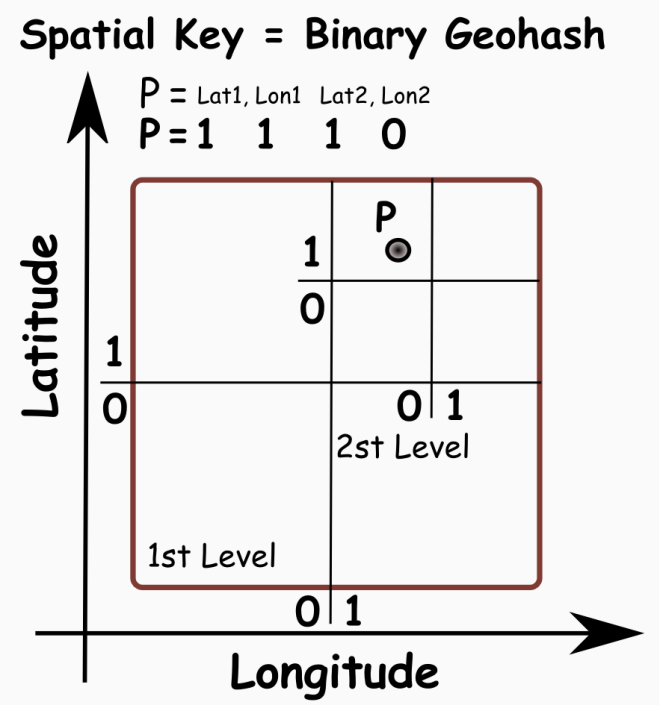
# Отличия в работе индексов

- Индекс 2dsphere поддерживает хранение точек, ломаных и многоугольников, другие два индекса – только коллекции точек
- В индексе 2dsphere все объекты хранятся в формате GeoJSON, в других индексах точки представляются просто как пары координат
- При создании коллекции с индексом geoHaystack требуется выбрать дополнительное категориальное поле, по которому проводится фильтрация. Я использую для этого поле user

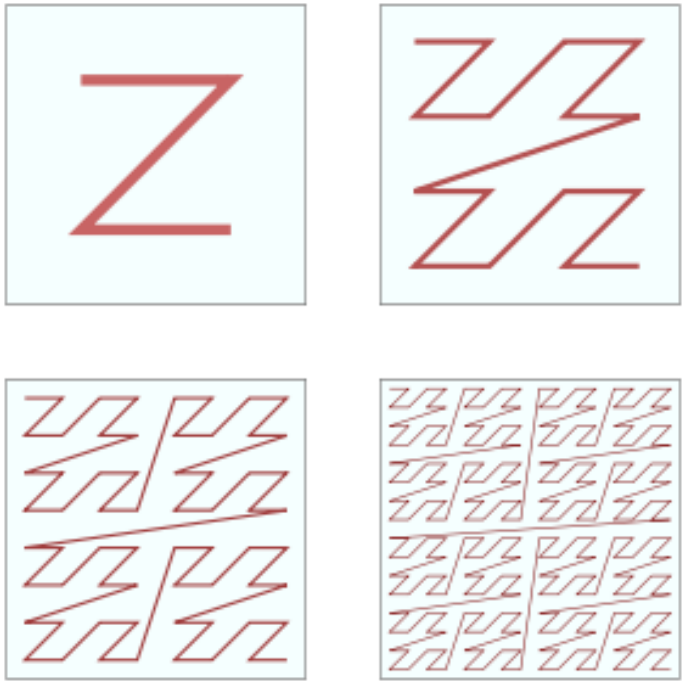
# Внутреннее устройство индексов: Б-деревья



# Внутреннее устройство индексов: геохеш

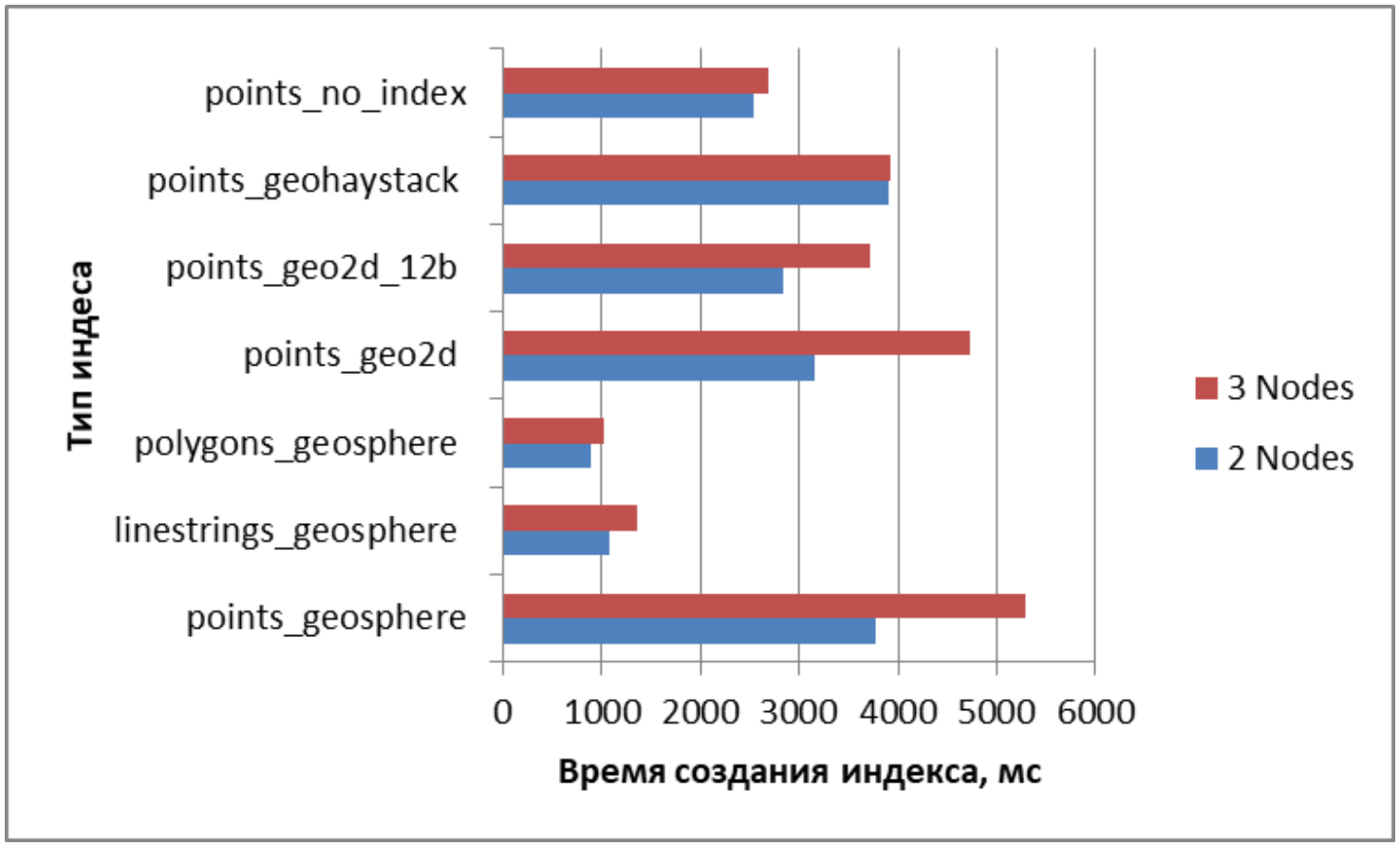



*Разбиение плоскости на квадранты*

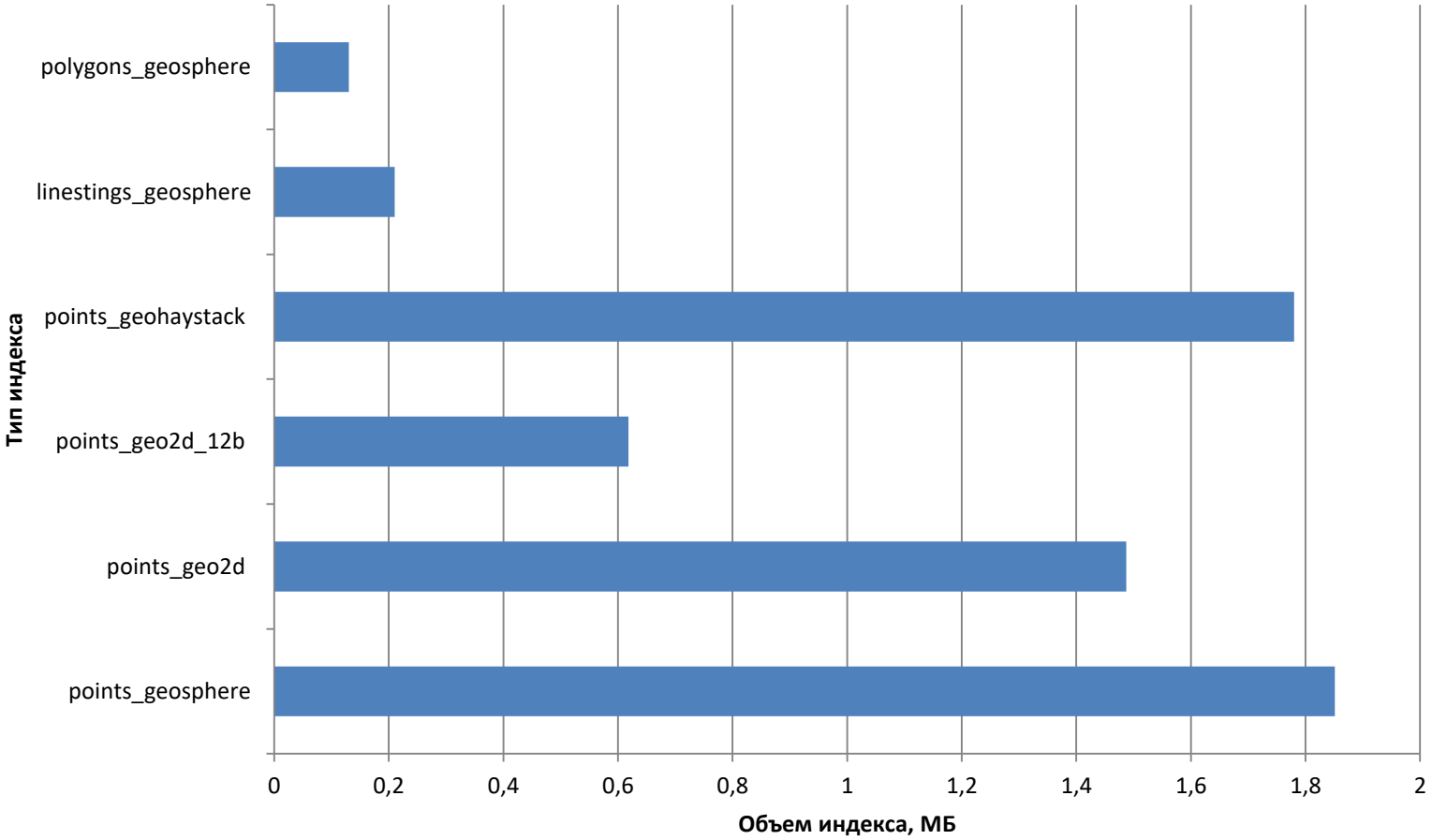


*Кривая Мортона (Z-curve)*

# Загрузка данных



# Размеры индексов



# Исследование производительности индекса geoHaystack

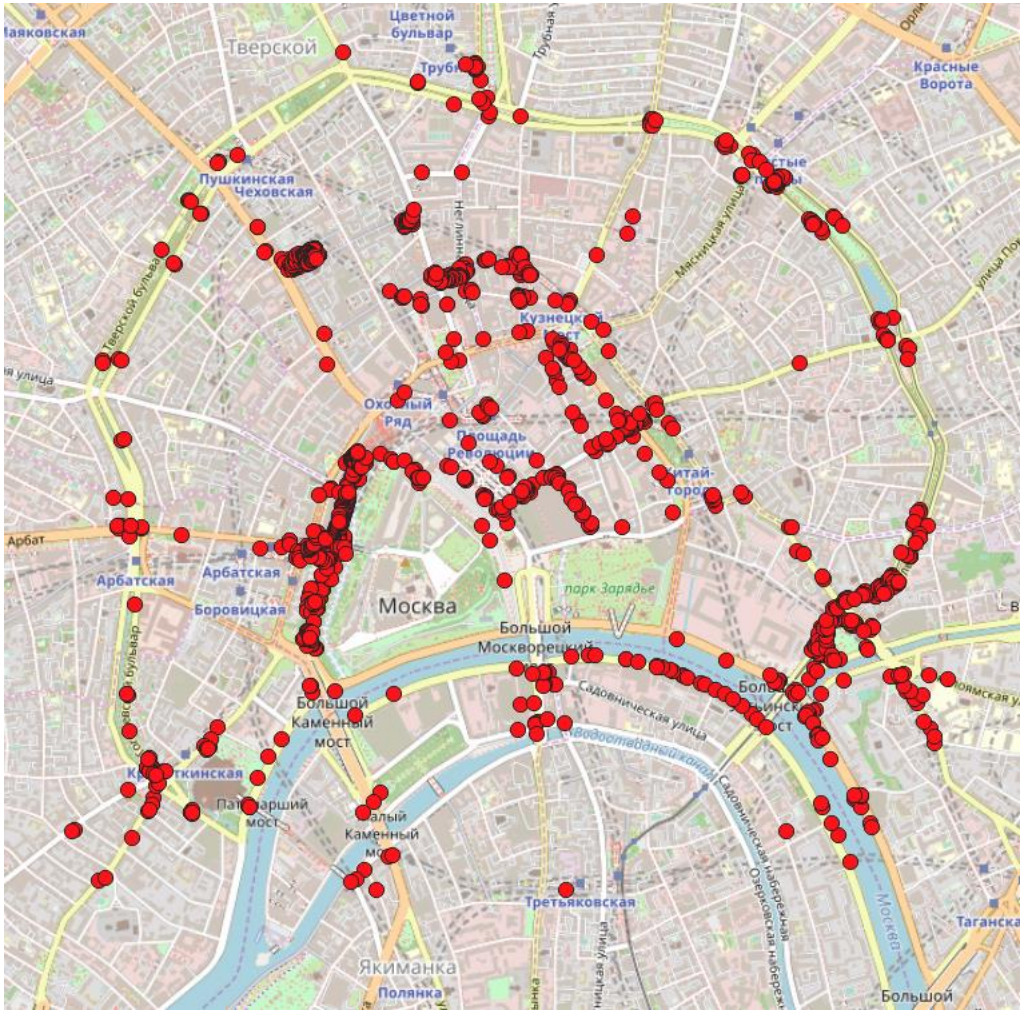
```
db.command(
    "geoSearch", "points_geohaystack",
    search={"properties.user": "luch86"},
    near=center,
    maxDistance=max_distance / radius * (180 / 3.1415),
    limit=50000,
)
```

*Запрос для индекса geoHaystack*

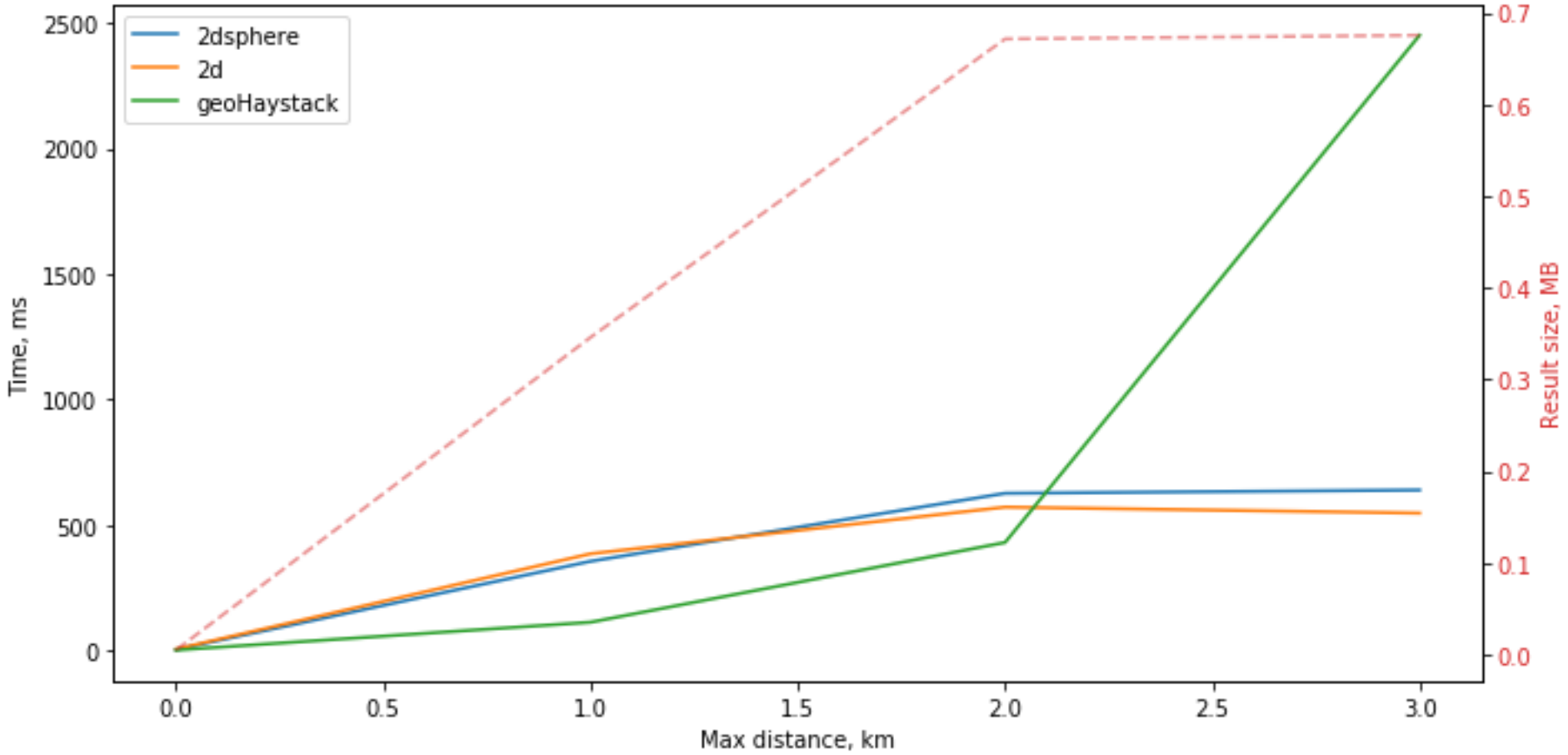
```
db["points_geosphere"].find({
    "geometry": {
        "$geoWithin": {
            "$centerSphere": [center, max_distance / radius],
        }
    },
    "properties.user": "luch86",
})
```

*Запрос для индекса 2dsphere*

# Исследование производительности индекса geoHaystack



# Исследование производительности индекса geoHaystack



# Исследование работы оператора nearSphere

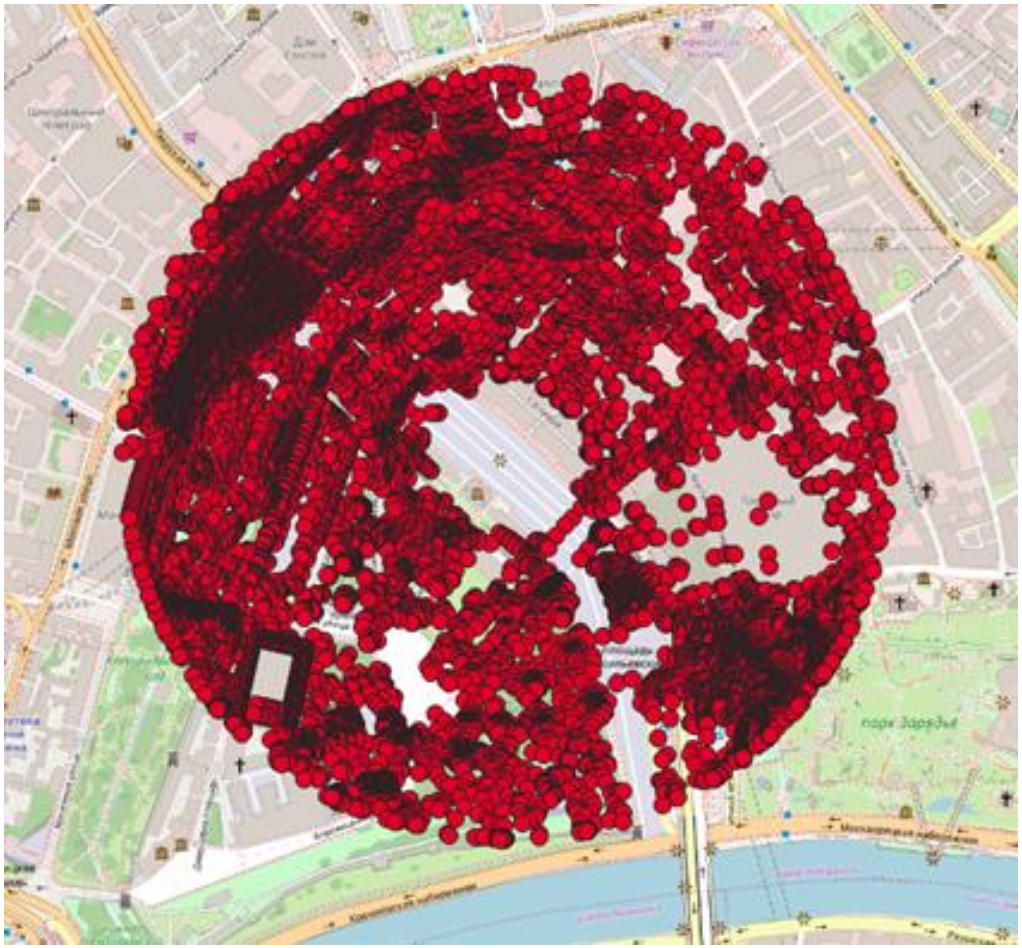
```
{
    "geometry" : {
        "$nearSphere": {
            "$geometry" : { "type": "Point", "coordinates": center },
            "$maxDistance": 500,
            "$minDistance": 100,
        }
    }
}
```

*Запрос для индекса 2dsphere*

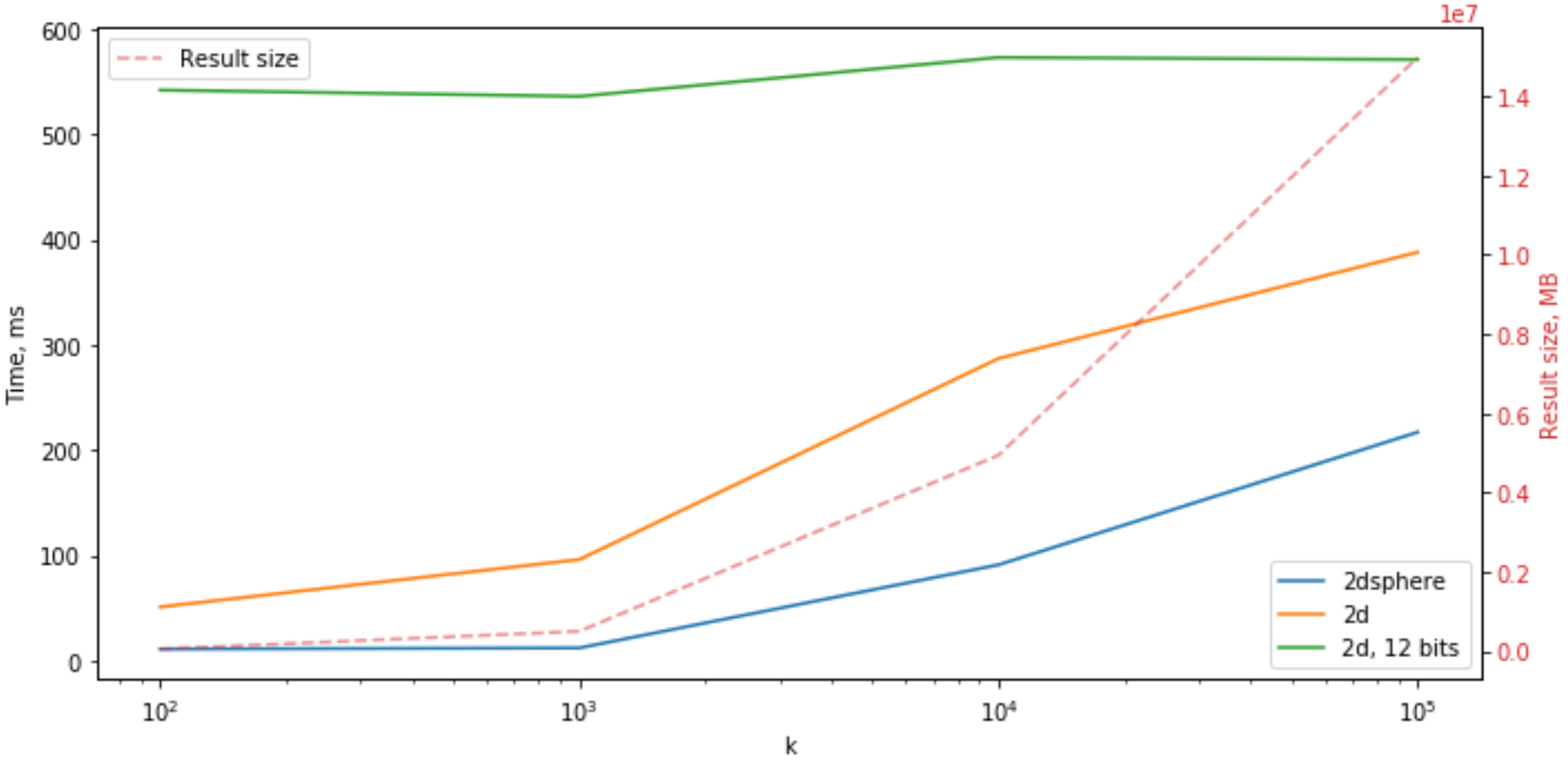
```
legacy_query = {
    "coordinates" : {
        "$nearSphere": center,
        "$maxDistance": 0.5 / radius,
        "$minDistance": 0.1 / radius,
    }
}
```

*Запрос для индекса 2d*

# Исследование работы оператора nearSphere



# Исследование работы оператора nearSphere



# Исследование работы оператора nearSphere

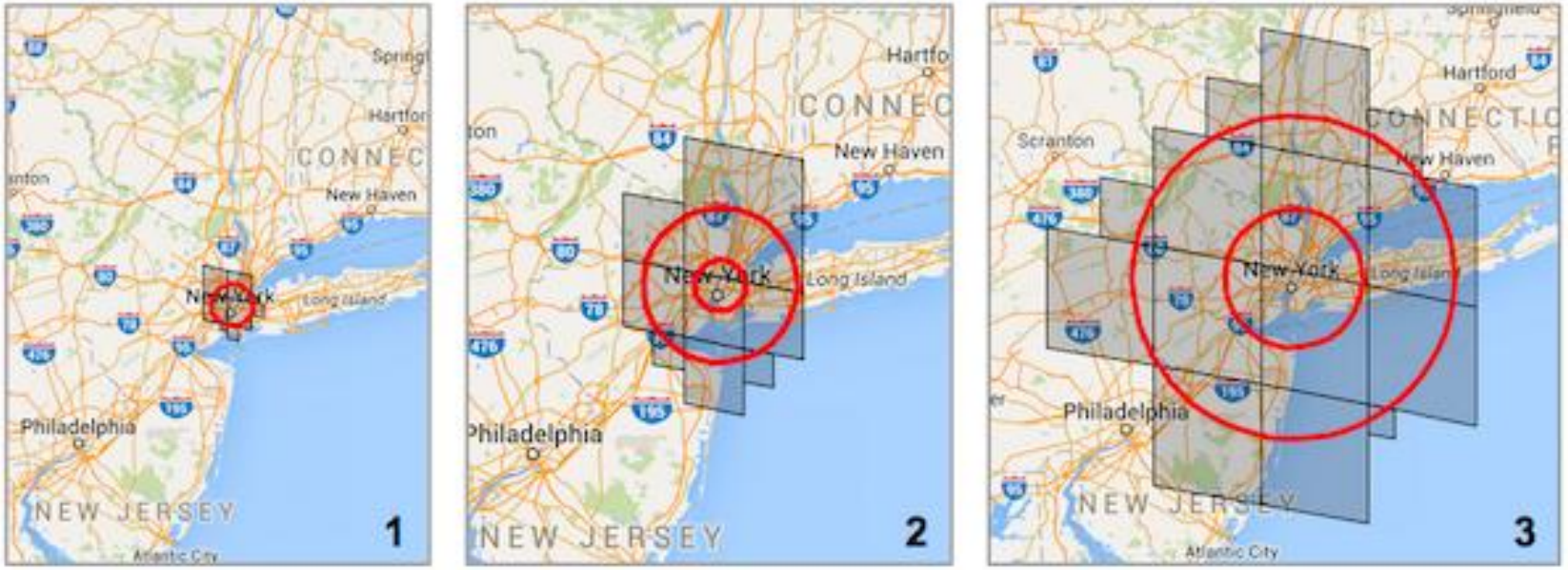
```
GEO_NEAR_2DSPHERE time: 162, keysExamined: -, docsExamined: -
        FETCH time: 10, keysExamined: -, docsExamined: 1990
                IXSCAN time: 10, keysExamined: 2012, docsExamined: -
        FETCH time: 11, keysExamined: -, docsExamined: 1717
        FETCH time: 11, keysExamined: -, docsExamined: 593
        FETCH time: 11, keysExamined: -, docsExamined: 1722
                IXSCAN time: 11, keysExamined: 1756, docsExamined: -
        FETCH time: 11, keysExamined: -, docsExamined: 1132
                IXSCAN time: 11, keysExamined: 1176, docsExamined: -
        FETCH time: 12, keysExamined: -, docsExamined: 760
                IXSCAN time: 12, keysExamined: 807, docsExamined: -
```

*План выполения запроса для индекса 2dsphere*

```
GEO_NEAR_2D time: 2159, keysExamined: -, docsExamined: -
        FETCH time: 1326, keysExamined: -, docsExamined: 180052
                IXSCAN time: 364, keysExamined: 180052, docsExamined: -
```

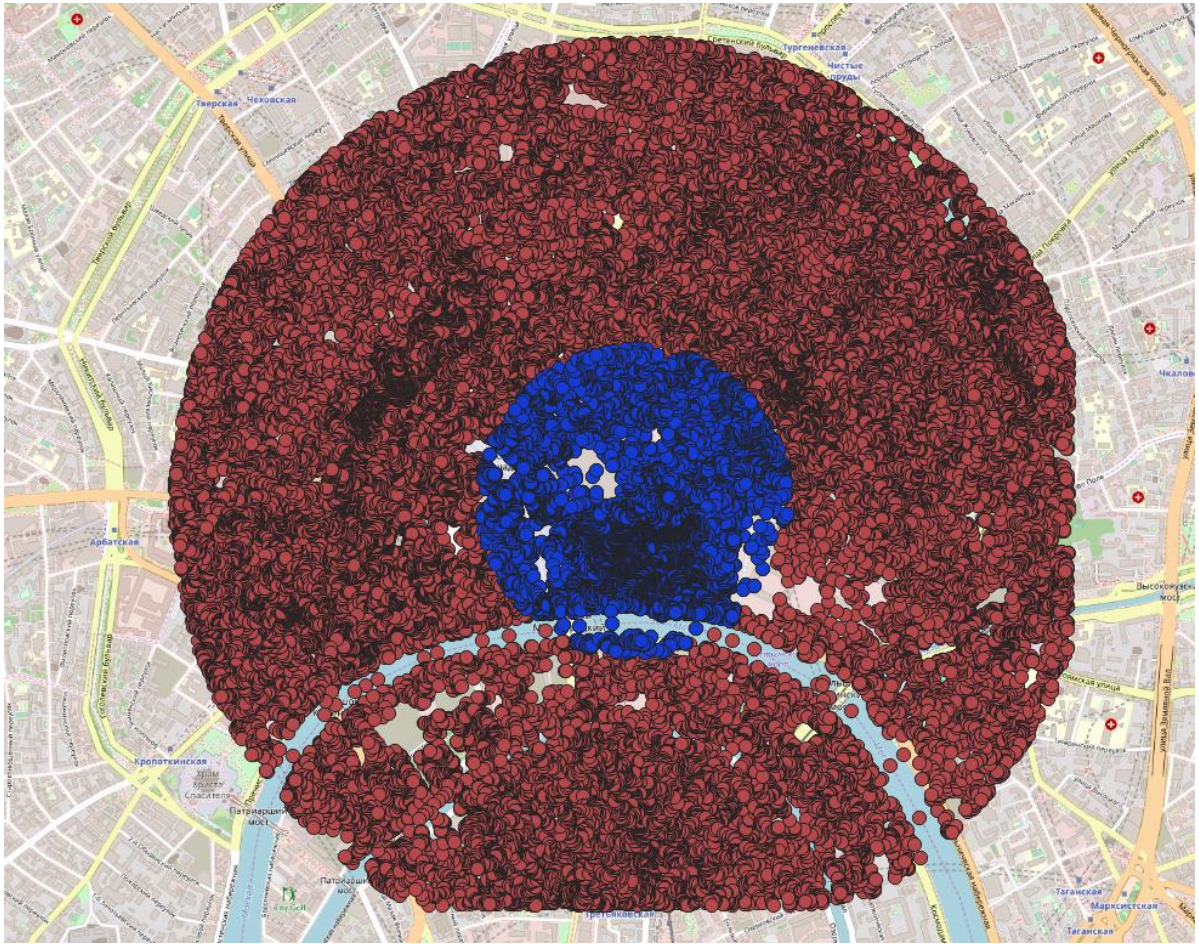
*План выполения запроса для индекса 2d с 12 битами геохеша*

# Исследование работы оператора nearSphere

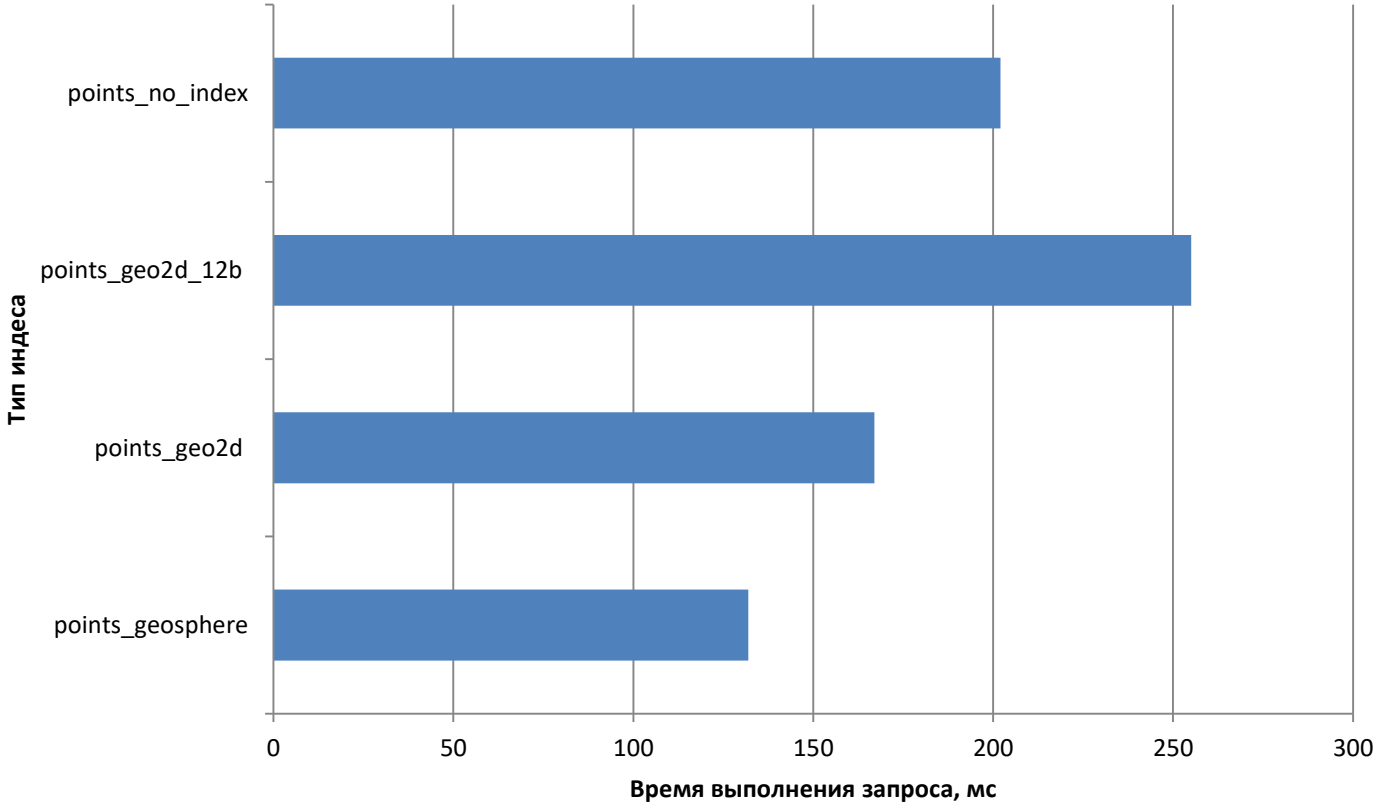



*Итерации алгоритма nearSphere*

# Исследование работы оператора geoWithin

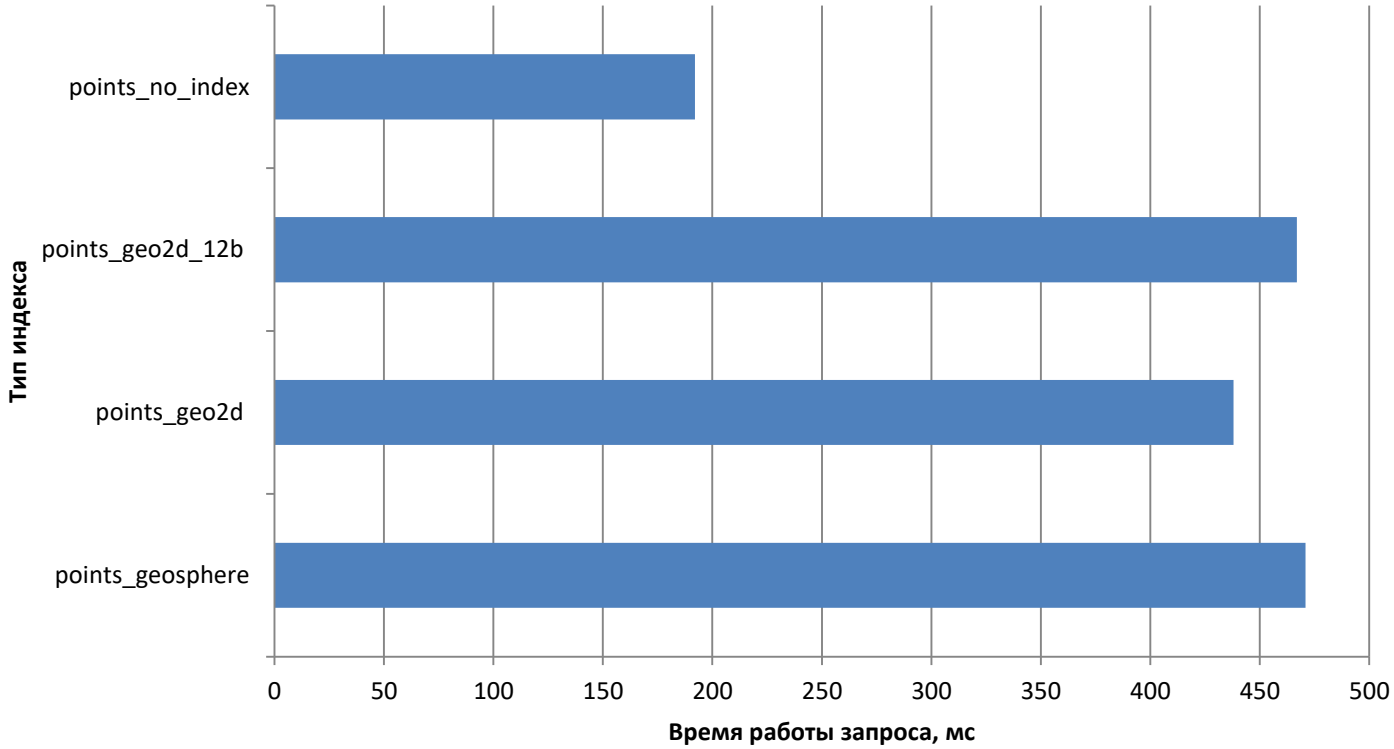


# Исследование работы оператора geoWithin




*Время работы запроса с оператором geoWithin для малого радиуса (извлекается 19.6% точек)*

# Исследование работы оператора geoWithin




*Время работы запроса с оператором geoWithin для большого радиуса (извлекается 70% точек)*

# Более сложные области в операторах geoWithin и geoIntersects

```
within_query = {
    "$and": [
        {
            "geometry": {
                "$geoWithin": {
                    "$centerSphere": [[37.629373, 55.7414147], 1.5 / radius]
                }
            }
        },
        {
            "geometry": {
                "$geoWithin": {
                    "$centerSphere": [[37.614153, 55.7576014], 1.5 / radius]
                }
            }
        },
    ],
}
```

*Запрос на пересечение двух окружностей*

# Более сложные области в операторах geoWithin и geoIntersects

```
FETCH time: 290, keysExamined: -, docsExamined: 76273
    IXSCAN time: 40, keysExamined: 76292, docsExamined: -
```

*План выполнения для запроса с оператором geoWithin*

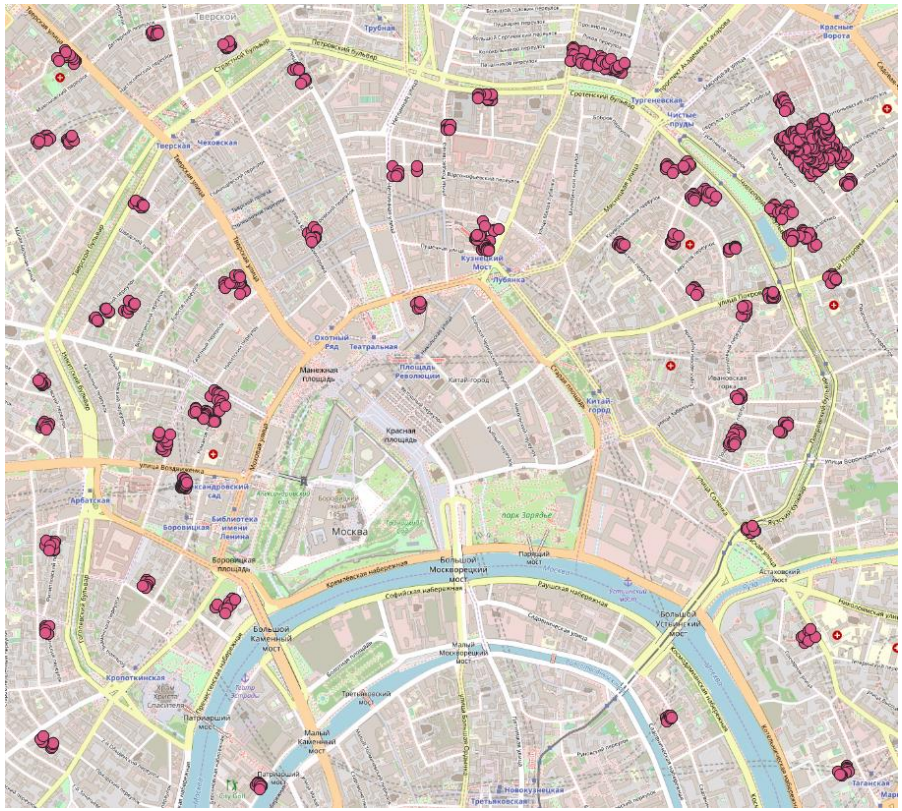
```
FETCH time: 220, keysExamined: -, docsExamined: 76273
    IXSCAN time: 60, keysExamined: 76292, docsExamined: -
```

*План выполнения для запроса с оператором geoIntersects*

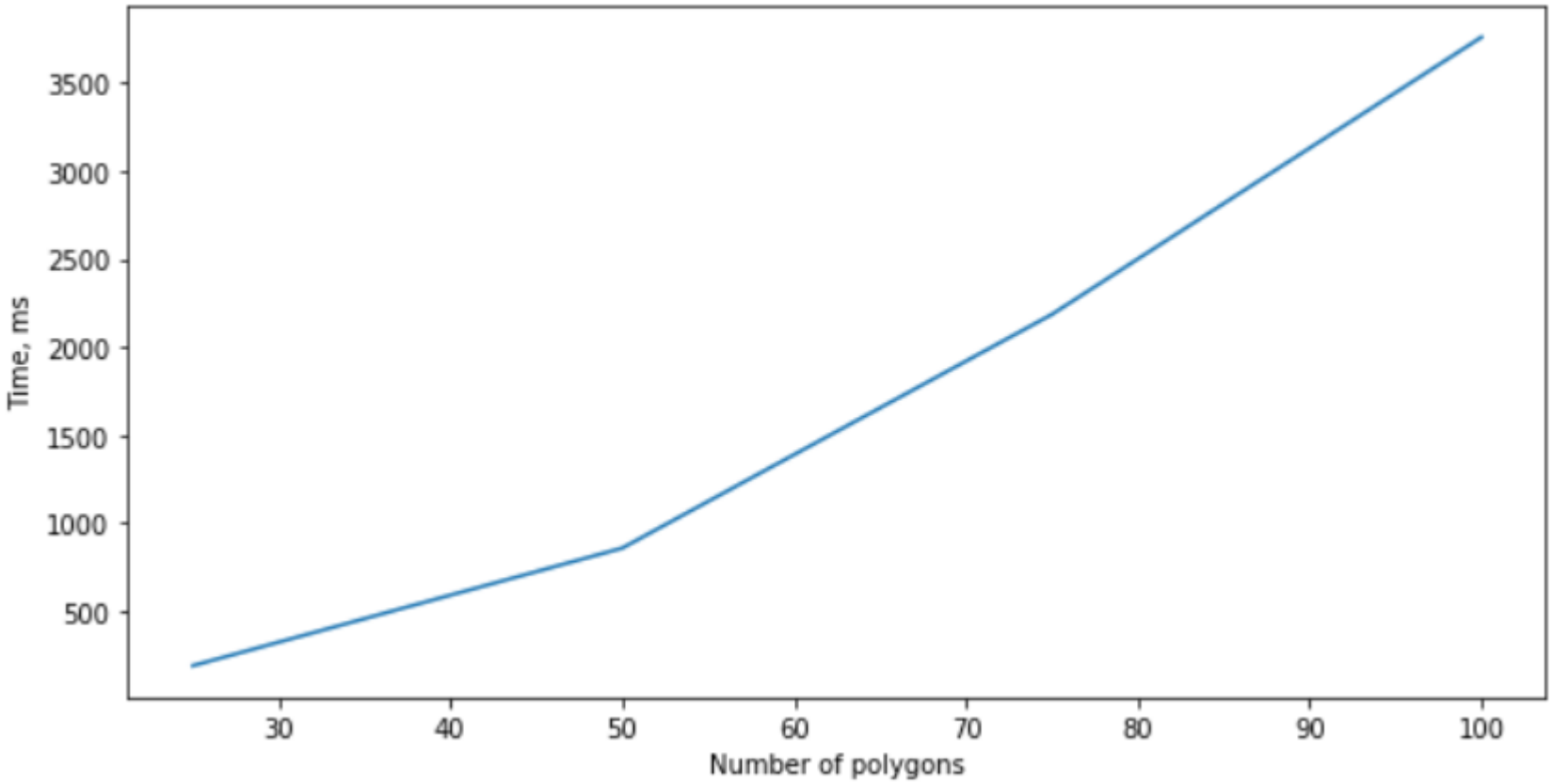
# Мультиполигон в качестве области в операторе geoWithin

```
{
    "geometry": {
        "$geoWithin": {
            "$geometry": {
                "type": "MultiPolygon",
                "coordinates": poly,
            }
        }
    }
}
```

*Код запроса*



# Мультиполигон в качестве области в операторе geoWithin

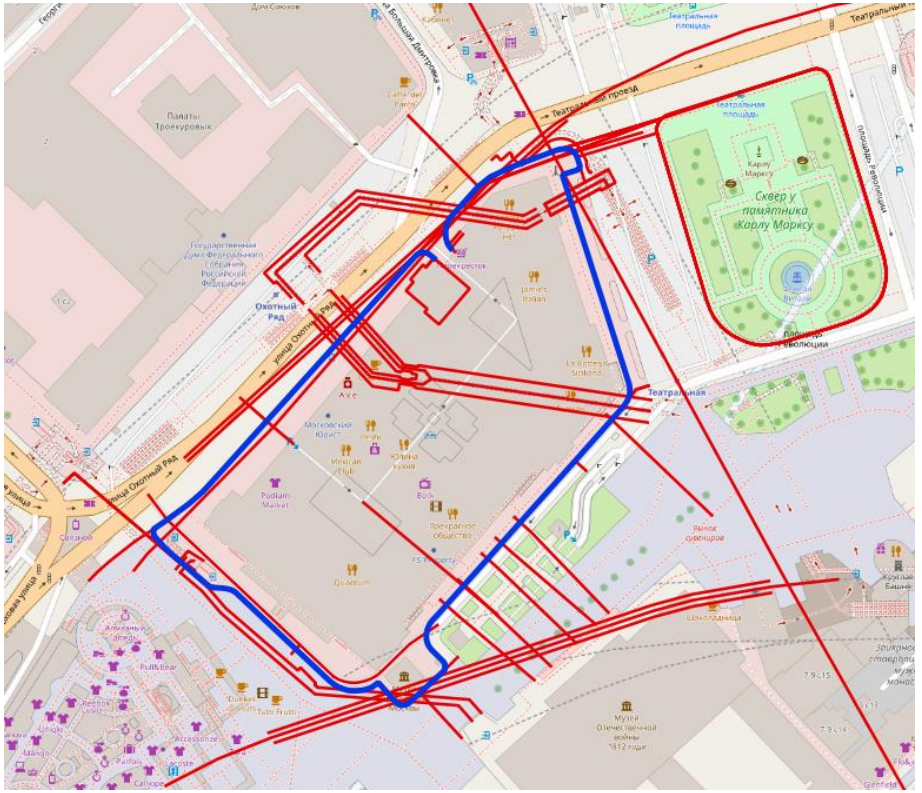



- Время работы зависит от количества многоугольников приблизительно линейно

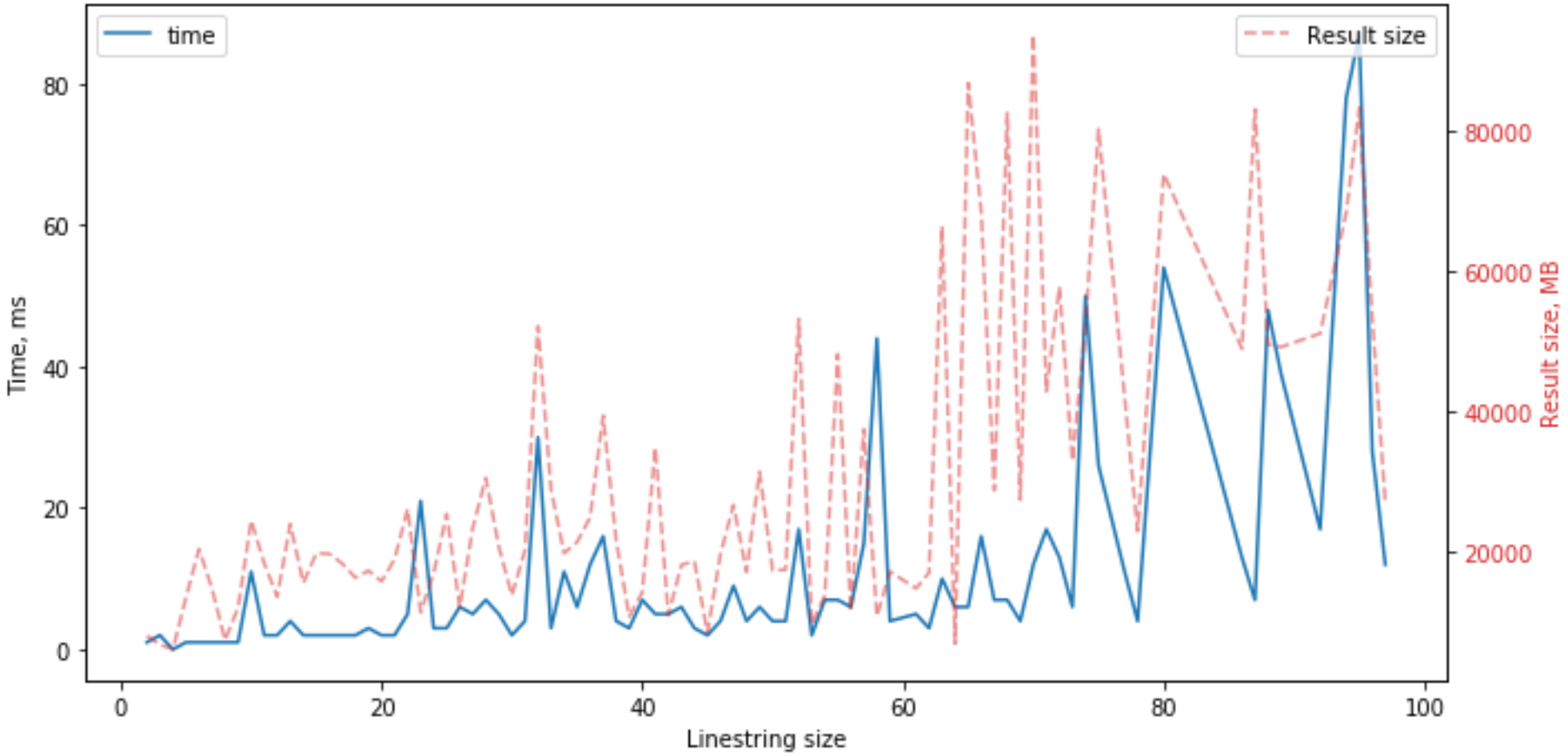
# Запросы к коллекции ломаных

```
{
    "geometry": {
        "$geoIntersects": {
            "$geometry": line["geometry"]
        }
    },
}
```

*Код запроса*



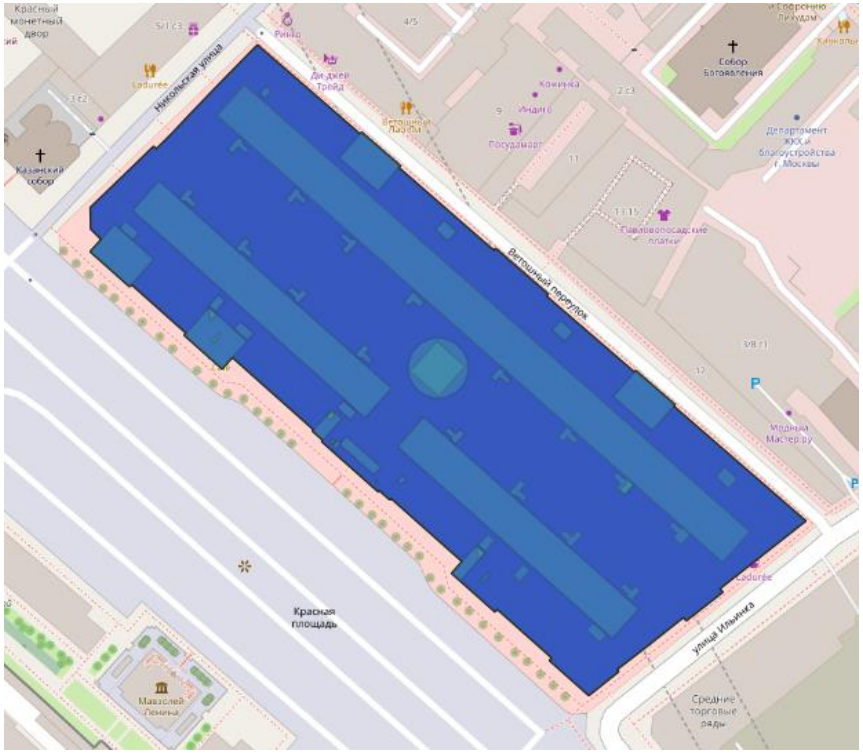
# Запросы к коллекции ломаных



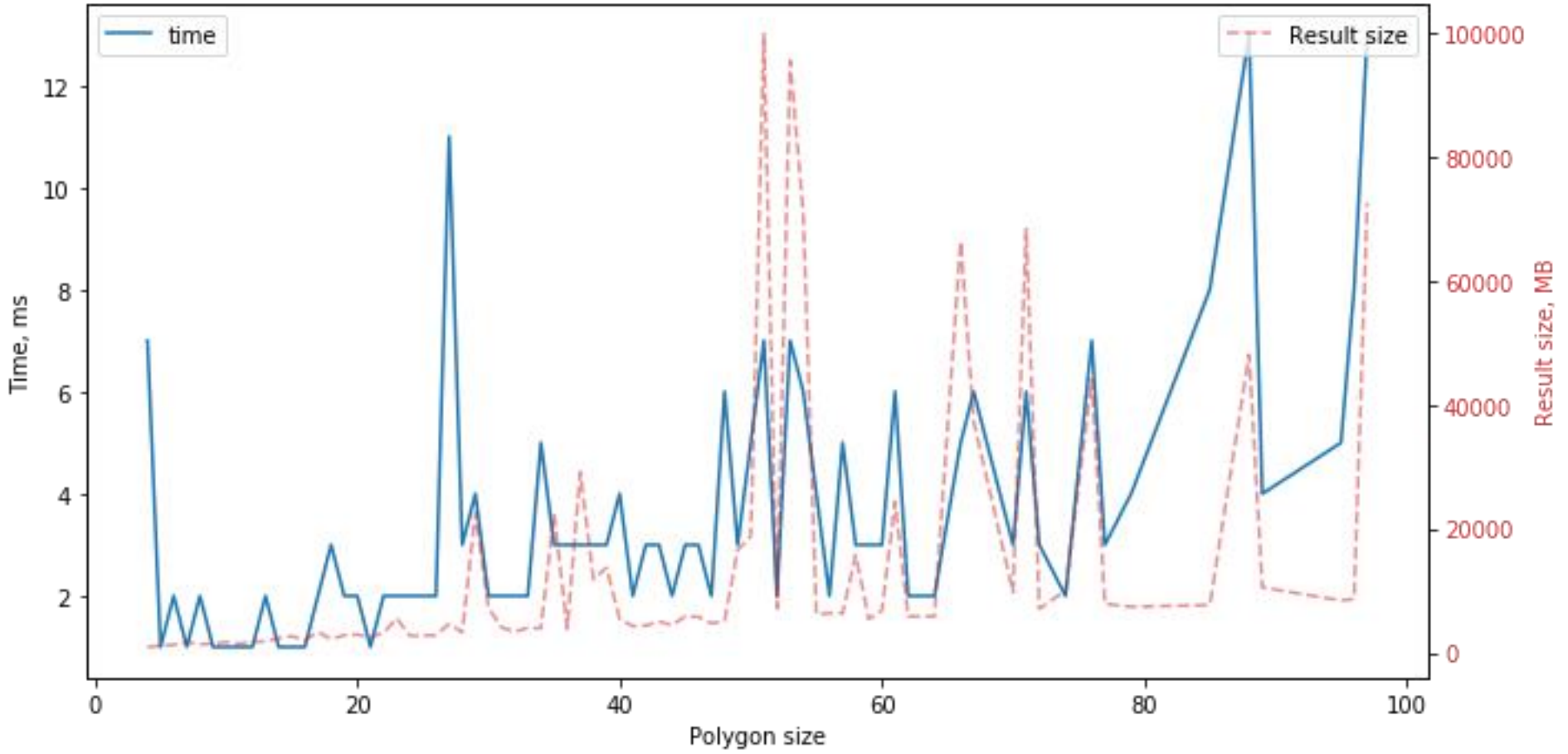
# Запросы к коллекции многоугольников

```
{
    "geometry": {
        "$geoWithin": {
            "$geometry": polygon["geometry"],
        }
    }
}
```

*Код запроса*



# Запросы к коллекции многоугольников



# Пользовательские функции в запросах

- MongoDB позволяет использовать для фильтрации данных произвольный JavaScript код
- Можно сохранить на сервере пользовательскую JavaScript функцию
- Реализована функция для вычисления длины ломаной на сфере (по формуле гаверсинусов)
- Нет возможности использования индексов, поэтому запрос работает неэффективно

# Пользовательские функции в запросах

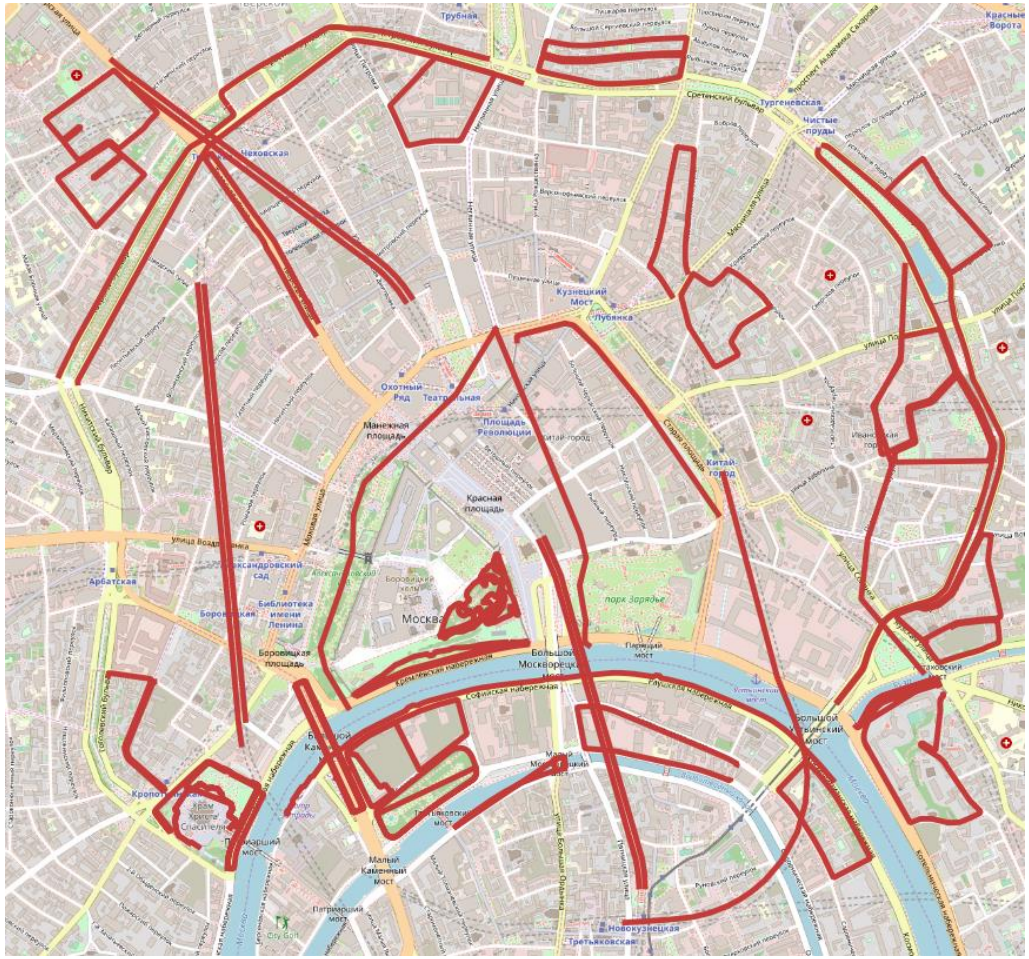
```
db.system_js.len = Code(
"""
function(point1, point2) {
    var haver = hav(point2[1] - point1[1]) +
                Math.cos(rad(point1[1])) * Math.cos(rad(point2[1])) * hav(point2[0] - point1[0]);
    return 6371 * archav(haver);
}
"""
)
```

*Функция расстояния между двумя точками*

```
db.system_js.linestring_len = Code(
"function(data) {"
    "var res = 0;"
    "for(var i = 0; i < data.length - 1; ++i)"
        "res += len(data[i], data[i + 1]);"
    "return res; "
"}"
)
```

*Длина ломаной на сфере*

# Пользовательские функции в запросах



# Update-запросы

```
db["points_geosphere"].update_many(filter=geojson_query, update={
        "$set": { "close_to_point": "true" },
    }
)
```

*Запрос на добавление поля*

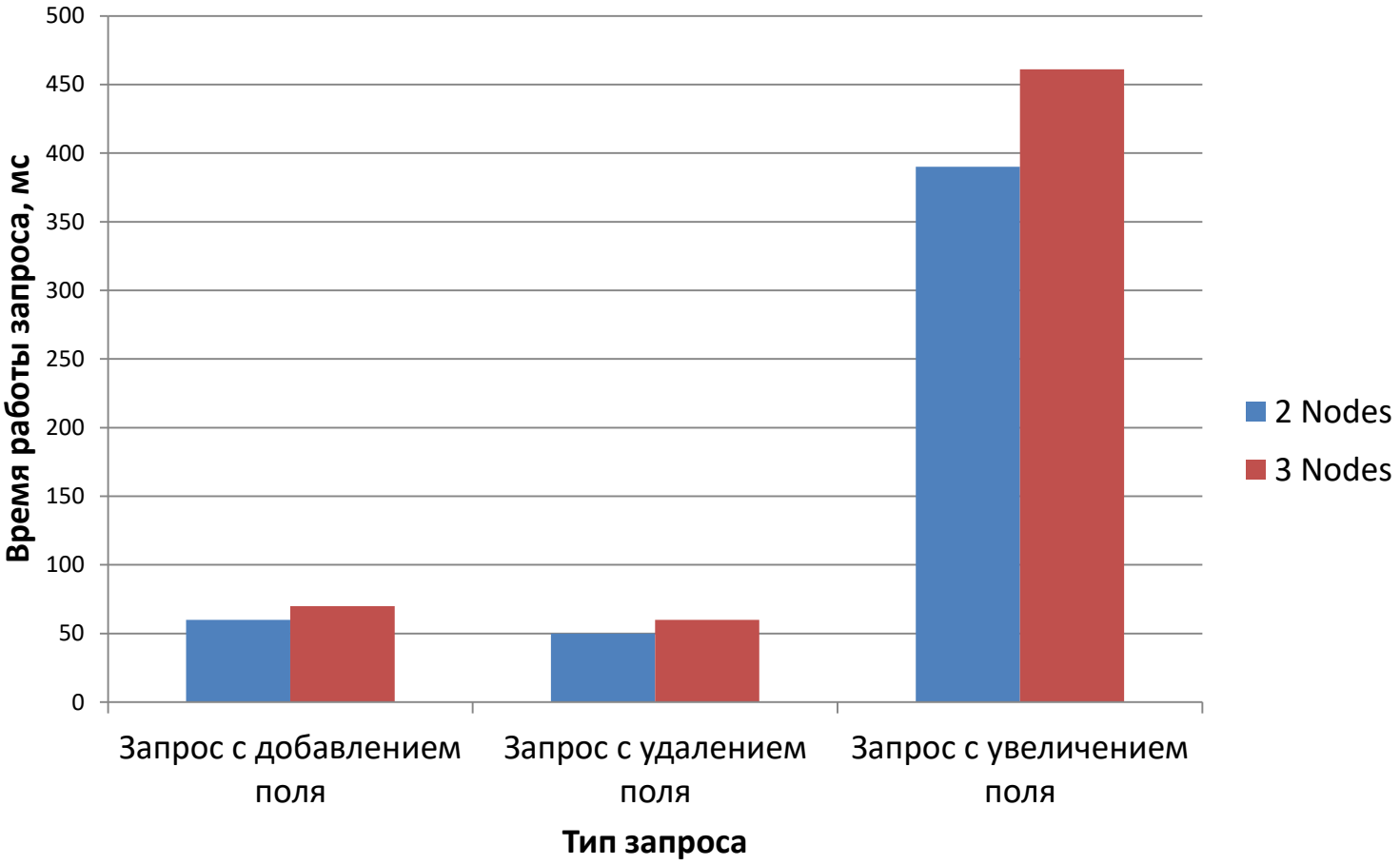
```
db["points_geosphere"].update_many(filter=geojson_query, update={
        "$unset": { "properties.changeset": "" },
    }
)
```

*Запрос на удаление поля*

```
db["points_geosphere"].update_many(filter = geojson_query, update = {
        "$inc": { "views": 1 },
    }
)
```

*Запрос на удаление числового поля*

# Update-запросы



# Агрегирующий запрос: группировка точек по расстоянию до центра

```
stage1 = {
    "$geoNear": {
        "near": {
            "type": "Point",
            "coordinates": [37.6279178, 55.7527639],
        },
        "maxDistance": 500, "distanceField": "dist", "limit": 100000,
    }
}
```

*Сортировка точек по расстоянию до центра*

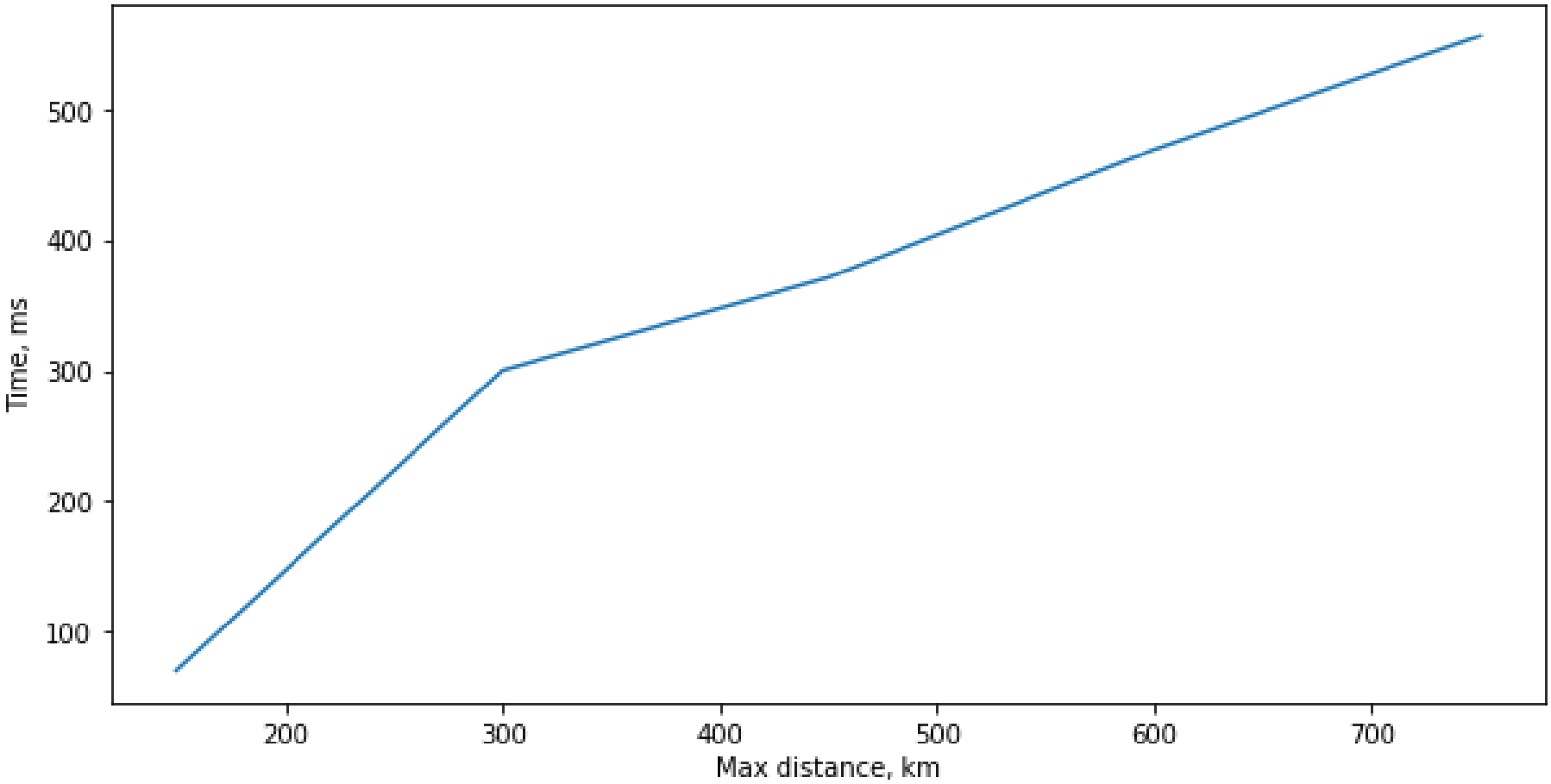
```
stage2 = { "$bucketAuto": { "groupBy": "$dist", "buckets": 10 } }
```

*Группировка точек*

```
db["points_geosphere"].aggregate([stage1, stage2])
```

*Запуск конвейера*

# Агрегирующий запрос: группировка точек по расстоянию до центра

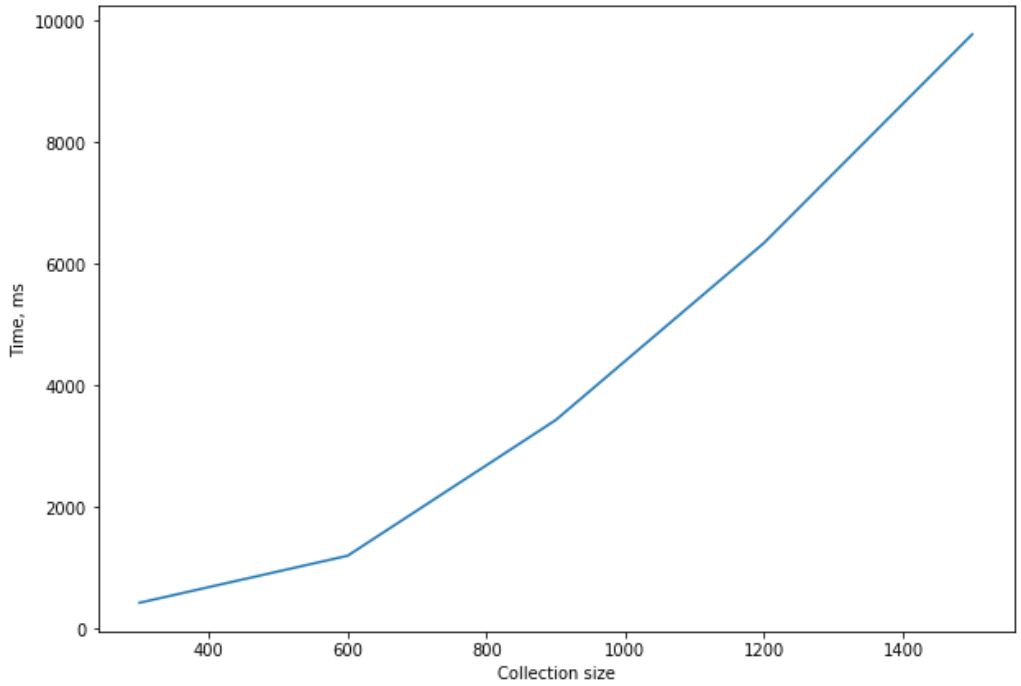


# Агрегирующий запрос: Spatial Join

```
stage1 = {
    "$lookup": {
        "from": "small_points",
        "localField": "geometry.type",
        "foreignField": "geometry.type",
        "as": "other",
    }
}
```

```
stage2 = {
    "$unwind": {
        "path": "$other",
    }
}
```

```
stage4 = {
    "$match": {
        "res": True,
    }
}
```




- График показывает квадратичную зависимость времени работы от количества точек

# Модель Map-Reduce в геопространственных запросах

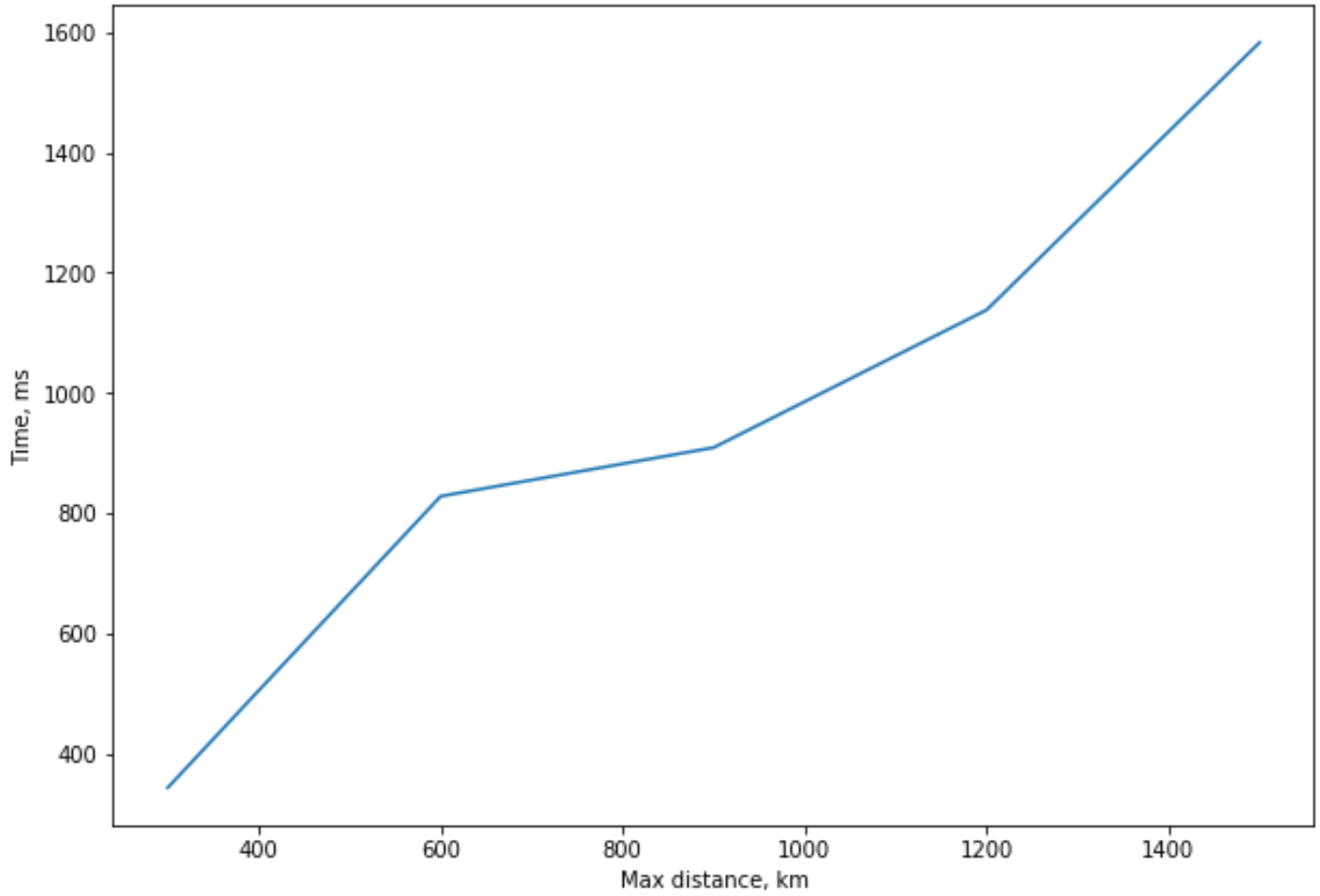
```
map_function = Code(
    """
    function () {
        var length = len(this.geometry.coordinates, [37.6279178, 55.7527639]);
        emit(length < 1 ? 0 : 1, length);
    }
    """
)
```

*Код для стадии Map*

```
reduce_function = Code(
    """
    function (key, values) {
        var count = 0;
        for (var i = 0; i < values.length; i++) {
            count += 1;
        }
        return count;
    }
    """
)
```

*Код для стадии Reduce*

# Модель Map-Reduce в геопространственных запросах



# Выводы

- MongoDB обеспечивает широкий выбор средств для работы с геоданными
- Отсутствуют реализации некоторых распространенных пространственных запросов и функций (Spatial Join, kNN Join)
- Это может быть компенсировано составлением более сложных запросов или добавлением собственных функций.
- Индекс geosphere оказывается самым полным и эффективным, и именно он сейчас развивается наиболее активно.
- У индексов 2d и geoHaystack также есть свои области применения