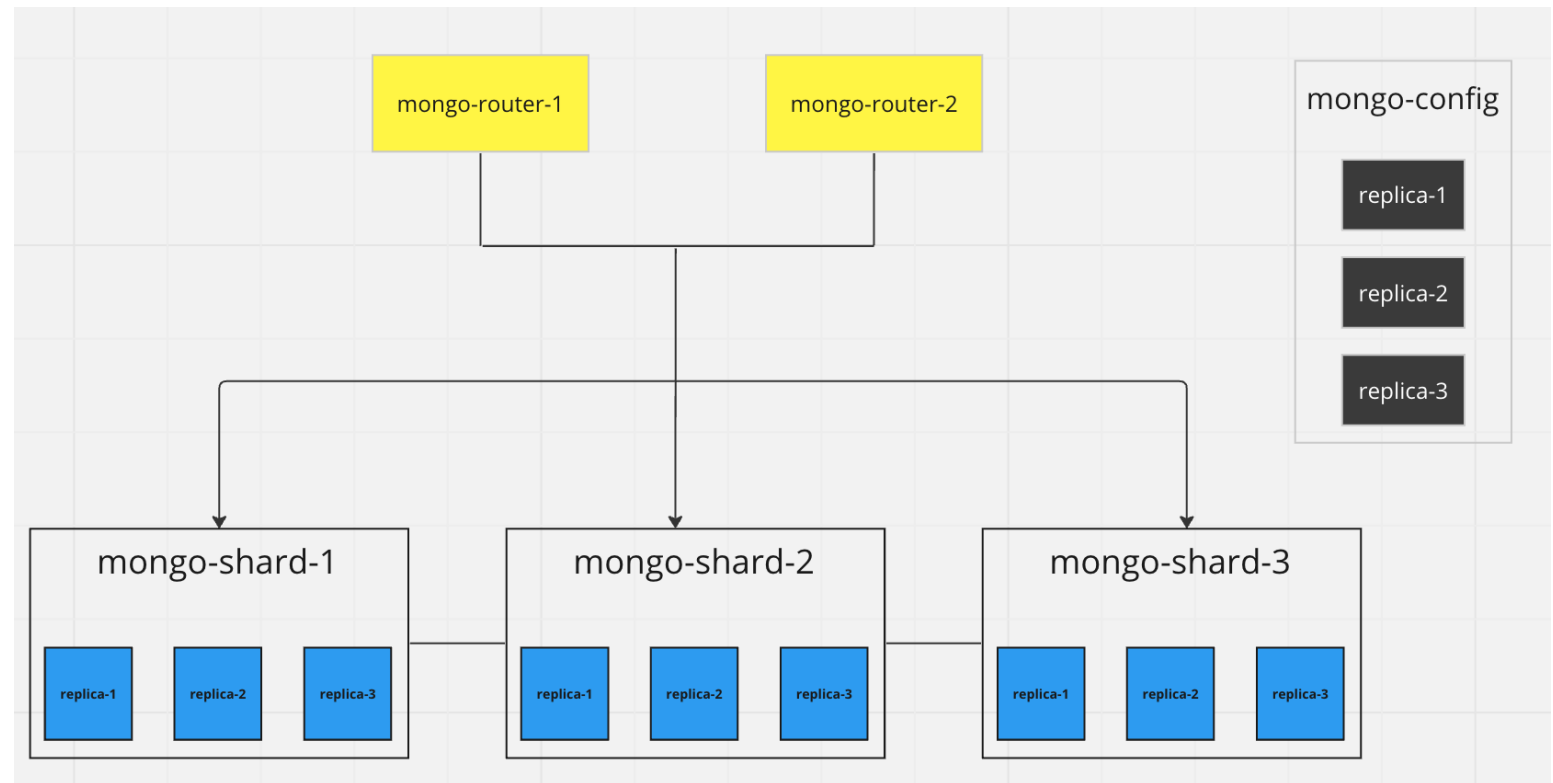


Assignment 3

Mantas Bagdonas, Alina Alencenovič, Martynas Lazdauskas

MongoDB cluster



```
1 db.createCollection('ship_positions')
2 sh.enableSharding('ships')
3 sh.shardCollection('ships.ship_positions', {'mmsi': 'hashed'});
4
5 db.createCollection('filtered_positions')
6 sh.enableSharding('ships')
7 sh.shardCollection('ships.filtered_positions', {'mmsi': 'hashed'});
8
9 db.filtered_positions.createIndex({'id': 1}, {'name': 'id', 'unique': true});
10 db.filtered_positions.createIndex({'timestamp': 1}, {'name': 'timestamp', 'unique': false});
11
12 db.getMongo().setReadPref("primaryPreferred")
13 db.adminCommand({setDefaultRWConcern: 1, defaultReadConcern: {level: "available"}, defaultWriteConcern: {w: 1}})
```

Replica-set vs Sharded replica-set

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
ae604fae851a	mongocfg2	0.55%	168.9MiB / 7.75GiB	2.13%	33.9MB / 13.2MB	213kB / 0B	120
f0345c9b7890	mongors2n3	38.95%	901.5MiB / 7.75GiB	11.36%	218MB / 196MB	0B / 0B	134
19f414e338f8	mongocfg3	0.47%	162MiB / 7.75GiB	2.04%	8.44MB / 7.52MB	0B / 0B	110
eb93270fb6c6	mongors3n3	41.36%	441.2MiB / 7.75GiB	5.56%	174MB / 183MB	0B / 0B	134
c8b3cad074a2	mongocfg1	0.62%	163.4MiB / 7.75GiB	2.06%	7.39MB / 7.49MB	0B / 0B	111
561d51e58191	mongors3n2	33.11%	446.1MiB / 7.75GiB	5.62%	73MB / 25.7MB	0B / 0B	128
733f702c6d82	mongors2n2	30.26%	867.5MiB / 7.75GiB	10.93%	159MB / 28.4MB	0B / 0B	130
9b5cd161d493	mongors2n1	52.10%	850.3MiB / 7.75GiB	10.71%	262MB / 235MB	0B / 0B	131
6e4eec5495fc	mongors1n1	30.67%	431.5MiB / 7.75GiB	5.44%	74MB / 26.8MB	0B / 0B	131
1c1c90f26bfa	mongors3n1	31.50%	397.3MiB / 7.75GiB	5.01%	74.9MB / 25.5MB	0B / 0B	129
3d56ba649c74	mongors1n2	32.71%	401.8MiB / 7.75GiB	5.06%	74.1MB / 26.9MB	0B / 0B	130
75810288b23f	mongors1n3	39.99%	441.3MiB / 7.75GiB	5.56%	173MB / 183MB	0B / 0B	136
f272579c6b38	mongos1	47.26%	370.5MiB / 7.75GiB	4.67%	351MB / 230MB	135kB / 53.2kB	295
263da26e544c	mongos2	29.65%	302.7MiB / 7.75GiB	3.81%	315MB / 247MB	311kB / 0B	273

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
03f44be5c518	mongors1n1	4.88%	220.3MiB / 7.75GiB	2.78%	259kB / 240kB	2.19MB / 0B	129
9c418fb38d8b	mongocfg3	7.39%	188.9MiB / 7.75GiB	2.38%	425kB / 536kB	111kB / 0B	128
c76a678e2abf	mongors1n3	3.25%	221.5MiB / 7.75GiB	2.79%	276kB / 440kB	483kB / 0B	146
69d5ec28c40a	mongors2n2	142.81%	1.046GiB / 7.75GiB	13.50%	51.7MB / 149MB	184kB / 0B	158
3e93a4203481	mongocfg2	6.71%	186.3MiB / 7.75GiB	2.35%	334kB / 401kB	0B / 0B	117
5f44e39ec309	mongocfg1	6.60%	192.3MiB / 7.75GiB	2.42%	880kB / 473kB	442kB / 0B	131
bbc151d15455	mongors3n1	5.01%	224.4MiB / 7.75GiB	2.83%	247kB / 227kB	0B / 0B	127
abfa161f5b6e	mongors3n3	5.28%	227.3MiB / 7.75GiB	2.86%	239kB / 220kB	0B / 0B	123
a745d689da00	mongors3n2	3.02%	231.1MiB / 7.75GiB	2.91%	261kB / 419kB	168kB / 0B	144
458d4b05ea6f	mongors2n3	104.70%	1GiB / 7.75GiB	12.91%	85.7MB / 434kB	65.5kB / 0B	140
78bf8a867160	mongors1n2	3.41%	219MiB / 7.75GiB	2.76%	251kB / 239kB	28.7kB / 0B	129
08db38dad963	mongos1	11.90%	223.5MiB / 7.75GiB	2.82%	100MB / 27.4MB	41.1MB / 57.3kB	189
6ff16d771123	mongors2n1	114.95%	822.9MiB / 7.75GiB	10.37%	60.8MB / 392kB	0B / 0B	140
b78ffffc2ef8	mongos2	7.00%	172.4MiB / 7.75GiB	2.17%	90.6MB / 24.5MB	0B / 0B	163

```
chunk 17: inserted 20000 rows of data in 4.212652624999464
chunk 20 processing
chunk 7: inserted 20000 rows of data in 5.277012124999601
chunk 22 processing
chunk 19: inserted 20000 rows of data in 5.602744709001854
chunk 9: inserted 20000 rows of data in 5.506783666998672
```

```
chunk 74: inserted 20000 rows of data in 50.87183599999844
chunk 67: inserted 20000 rows of data in 32.21405933400092
chunk 69: inserted 20000 rows of data in 32.5972504579986
chunk 65: inserted 20000 rows of data in 31.247211040994443
chunk 71: inserted 20000 rows of data in 34.481893791999903
```

all data inserted in 410.76s

Data insertion

```
def insert_data(indices, chunkId):
    print("chunk", chunkId, "processing")
    start_time = perf_counter()

    client = mongo_clients[chunkId % len(mongo_clients)]
    db = client["ships"]
    collection = db["ship_positions_3"]

    data = pd.read_csv('data.csv', skiprows=(chunkId * chunksize), nrows=chunksize, names=colnames)
    data = data.rename(columns=lambda x: x.lower().replace(' ', '_'))

    insert = data.to_dict(orient='records')

    for _id, row in enumerate(insert):
        row['id'] = chunkId * chunksize + _id
        row['timestamp'] = row['#_timestamp']
        del row['#_timestamp']

    result = collection.insert_many(insert)

    timing = perf_counter() - start_time
    print(f"chunk {chunkId}: inserted {len(result.inserted_ids)} rows of data in {timing}")
```

```
def parallelize_inserts(indices, n_chunks, n_cpus):
    pool = mp.Pool(n_cpus)
    chunks = np.array_split(indices, n_chunks)
    pool.starmap(insert_data, [(chunk, i) for i, chunk in enumerate(chunks)])

if __name__ == '__main__':
    cpus_max = mp.cpu_count()
    indices = range(nrows)

    n_chunks = math.ceil(len(indices) / chunksize)

    start_time = perf_counter()
    parallelize_inserts(indices, n_chunks, cpus)
    timing = perf_counter() - start_time

    [client.close() for client in mongo_clients]

    print()
    print(f"all data inserted in {round(timing, 2)}s")
```

Data filtering

```
def get_distinct_vessels():
    client = MongoClient(mongos1)
    db = client["ships"]
    collection = db["ship_positions"]
    query = {'mmsi': {'$ne': '0'}}
    vessels = collection.distinct('mmsi', query)
    client.close()

    return vessels
```

```
def remove_nas(vessels):
    columns = [
        'timestamp',
        'navigational_status',
        'latitude',
        'longitude',
        'sog',
        'cog'
    ]

    return [
        doc for doc in vessels if all(
            column in doc and doc[column] is not None and (
                isinstance(doc[column], (float, int)) and not math.isnan(doc[column])
                or isinstance(doc[column], str) and doc[column].lower() != 'nan'
            )
            for column in columns
        )
    ]
```

```
def filter(vessel_mmsi, chunkId):
    start_time = perf_counter()

    conn_string = mongos1 if chunkId % 2 == 0 else mongos2
    client = MongoClient(conn_string)
    db = client["ships"]
    query = [
        {
            '$match': {
                '$and': [
                    {'mmsi': {'$in': vessel_mmsi}},
                ]
            },
            {
                '$group': {
                    '_id': '$mmsi',
                    'count': {'$sum': 1}
                }
            },
            {
                '$match': {
                    'count': {'$gte': 100}
                }
            },
            {
                '$lookup': {
                    'from': 'ship_positions',
                    'localField': '_id',
                    'foreignField': 'mmsi',
                    'as': 'documents'
                }
            }
        ]

    vessels = list(db["ship_positions"].aggregate(query))
    vessels = sum([vessel['documents'] for vessel in vessels], [])
    before_filter = len(vessels)
    vessels = remove_nas(vessels)

    if len(vessels) > 0:
        db['filtered_positions'].insert_many(vessels)

    timing = perf_counter() - start_time
```

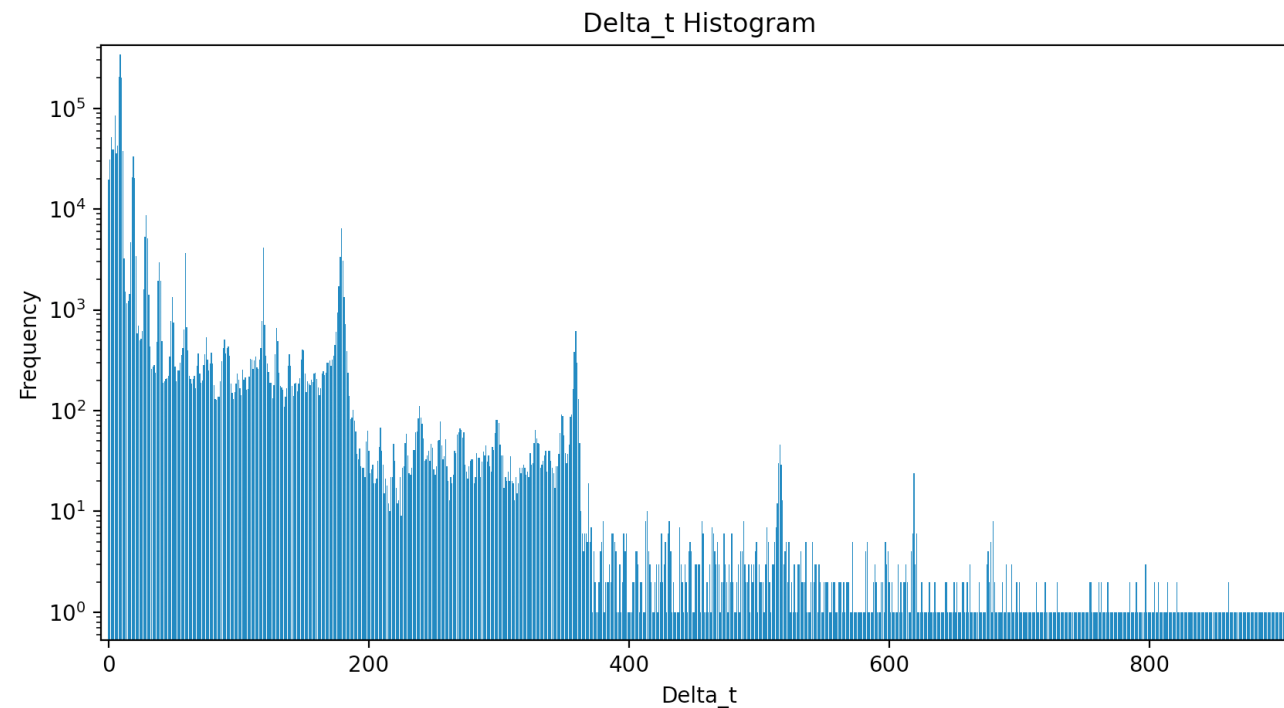
Histogram

```
def update_vessel_data_points(data_points, collection):
    delta_t = []
    print('updating', len(data_points), 'for vessel:', data_points[0]['mmsi'])
    data_points.sort(key=lambda x: x['timestamp'])

















































    for i in range(len(data_points)):
        if i == 0:
            delta_t.append((data_points[i]['_id'], 0))
            continue
        curr_time = datetime.strptime(data_points[i - 1]['timestamp'], '%d/%m/%Y %H:%M:%S')
        next_time = datetime.strptime(data_points[i]['timestamp'], '%d/%m/%Y %H:%M:%S')
        delta_t.append((data_points[i]['_id'], (next_time - curr_time).total_seconds() * 1000))

    for update in delta_t:
        collection.update_one({'_id': update[0]}, {'$set': {'delta_t': int(update[1])}})

    print('updated', len(data_points), 'for vessel:', data_points[0]['mmsi'])
```



Fail tolerance

<input type="checkbox"/>	 mongors3n1 bbc151d15455 	mongo	Running	27077	8 minutes ago     
<input type="checkbox"/>	 mongors2n3 458d4b05ea6f 	mongo	Exited (137)	27067	
<input type="checkbox"/>	 mongocfg3 9c418fb38d8b 	mongo	Running	-	8 minutes ago     
<input type="checkbox"/>	 mongocfg2 3e93a4203481 	mongo	Running	-	8 minutes ago     
<input type="checkbox"/>	 mongocfg1 5f44e39ec309 	mongo	Running	-	8 minutes ago     
<input type="checkbox"/>	 mongors2n1 6ff16d771123 	mongo	Exited (137)	27047	
<input type="checkbox"/>	 mongors1n1 03f44be5c518 	mongo	Running	27017	8 minutes ago     
<input type="checkbox"/>	 mongos2 b78fffc2ef8 	mongo	Running	27020	8 minutes ago     

```
[[direct: mongos] ships> db.filtered_positions.count()
```

```
1338347
```

```
1338347
```

```
[[direct: mongos] ships> db.filtered_positions.insert({mmsi: 5298719481, id:25098230598})
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
```

```
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("646f551fd2787e1319e8a2b8") }
}
```

```
[[direct: mongos] ships> db.filtered_positions.find({mmsi:5298719481})
```

```
[
  {
    _id: ObjectId("646f551fd2787e1319e8a2b8"),
    mmsi: 5298719481,
    id: 25098230598
  }
]
```