

# Databases 2 Project

## Performance analysis of VoltDB using top-k keyword and top-k document queries

Nicoara Bogdan-Cristian

# Abstract

The goal of this paper is to evaluate the performance of a new SQL database, namely VoltDB by running complicated queries on a large data set, in order to see the average response time depending on query complexity and data set size. One other aspect that this paper approaches is analysing the top-K keywords and documents benchmark.

## 1. Introduction

### 1.1 About VoltDb

VoltDb is a solution for high performance business-critical applications, taking advantage of the modern computing environment by using in-memory storage, serialization of all data access and clustering and replication on multiple server in order to achieve a high reliability, availability and scalability.

The main focus of VoltDb is *Fast Data*. Applications that must process large amounts of data quickly.

VoltDB is a NewSQL in-memory, distributed relational database with high-throughput, low-latency transactions, written in JAVA. It uses SQL as it's primary interrogation language for querying and data management although it has some limitations when it comes to interweaving SELECTS, and uses a combination of Java and SQL for Stored Procedures.

The biggest advantages it has are it's :

High performance, being able to process millions of transactions per second by keeping the data in memory.

Scalability as it can easily scale horizontally.

Strong consistency : Ensures ACID compliance across distributed environments.

Real-time Analytics : Supports real-time data processing and analytics.

When it comes to indexing, VoltDB uses unique and partial and non-unique indexes which are stored as B-tree in order to maximize search efficiency on large datasets.

When it comes to distribution, VoltDB supports partitioning and replication. Partition enables each node of the cluster to hold some data which leads to parallel processing and increased efficiency.

Replication is also supported across every node of the cluster enabling data to be accessed fast from any node and execute queries locally decreasing processing time for common queries.

### 1. Passive Database Replication

This focuses on creating a read-only replica to be used in case of disaster and is done by selecting from a master cluster, the databases you want to replicate and respective tables.

### 2. Active(n) Cross Datacenter Replication ( or XDCR)

This focuses on increasing availability and decreasing latency by enabling reading and updating multiple clusters in between themselves, in different geographical locations.

## 2. Resume

	VoltDB
DBMS Type	New SQL (RDBMS)
Data format	SQL (relational)
Implementation	Java
Transaction	ACID
Consistency	Strong Consistency
In-memory	Yes
Replication	Yes
Partitioning	Yes
Ad-hoc queries	SQL, Java stored procedures
MapReduce	No
Secondary indices	Yes
Geospatial indices	No
Text indices	Yes

### 3. Setup

For this project I decided to use Docker in order to create the database environment. And for the test I used an environment with one node (Single Instance) and one with 3 nodes (Cluster).

Therefore prerequisites for this project are : Docker ( in my case I used docker desktop for windows), Python.

#### 3.1 Single instance – Docker

Because voltDB is a in-memory database, it will use a lot of RAM, and we will need to compensate for that by providing the docker with enough memory. In order to make the necessary configuration we will need a custom **deployment.xml** file.

First we configure a network for VoltDB :

1 - docker network create -d bridge voltLocalSingle

Then we ran the container:

2 - docker run --privileged -v  
C:\Path\To\The\custom\XML\deployment.xml:/etc/voltdb/deployment.xml -p  
21212:21212 -p 8080:8080 -p 8081:8081 -e HOST\_COUNT=1 -e HOSTS=node4  
--name=node4 --network voltLocalSingle voltdb/voltdb-community:6.6

As voltDb is now a licensed product, I am using the community edition image. We are running in privileged mode in order to be able to modify some read-only files in the docker that would block our custom xml update.

We connect to the docker :

3 - docker exec -it node4 bash

And we run these 2 commands :

4 - echo never > /sys/kernel/mm/transparent\_hugepage/enabled

5 - echo never > /sys/kernel/mm/transparent\_hugepage/defrag

Then in the same directory with the deployment.xml on the local host we give this command:

```
6 - docker cp .\deployment.xml node4:/var/voltdb/voltdbroot/config/
```

And then back in the docker :

```
7 – voltadmin update /var/voltdb/voltdbroot/config/deployment.xml
```

And this should make the query's run with the required capabilities.

In case that step 7 gives an error. It's enough to restart the docker container.

### 3.2 Multiple instance – Docker

Again we configure a network for VoltDB :

```
1 - docker network create -d bridge voltLocalCluster.
```

Then we ran for all the nodes in the net-work :

```
2 - docker run --privileged -v
```

```
C:\Users\bogda\Desktop\Diverse\Anul_4\Bd2\proiect\docker_runnables\deployme  
nt.xml:/etc/voltdb/deployment.xml -p 21212:21212 -p 8080:8080 -p 8081:8081 -e  
HOST_COUNT=3 -e HOSTS=node1,node2,node3 --name=node1 --network  
voltLocalSingle voltdb/voltdb-community:6.6
```

```
3 - docker run --privileged -v
```

```
C:\Users\bogda\Desktop\Diverse\Anul_4\Bd2\proiect\docker_runnables\deployme  
nt.xml:/etc/voltdb/deployment.xml -P -e HOST_COUNT=3 -e  
HOSTS=node1,node2,node3 --name=node2 --network voltLocalCluster  
voltdb/voltdb-community:6.6
```

```
4 - docker run --privileged -v
```

```
C:\Users\bogda\Desktop\Diverse\Anul_4\Bd2\proiect\docker_runnables\deployme  
nt.xml:/etc/voltdb/deployment.xml -P -e HOST_COUNT=3 -e  
HOSTS=node1,node2,node3 --name=node3 --network voltLocalCluster  
voltdb/voltdb-community:6.6
```

After the entire network is up, in order to update the xml file we need to do it **one by one** on the nodes, without the other 2 being up. Otherwise those will refuse the modification of the xml, as it's no longer identical. All 3 nodes need to have identical xml files.

5 - docker exec -it <node number> bash

6 - echo never > /sys/kernel/mm/transparent\_hugepage/enabled

7 - echo never > /sys/kernel/mm/transparent\_hugepage/defrag

Outside the docker

8 - docker cp ./deployment.xml <node number>:/var/voltdb/voltdbroot/config/

Back inside the docker :

9 – voltadmin update /var/voltdb/voltdbroot/config/deployment.xml

At the end, restart the hole network once more and it will be good to go.

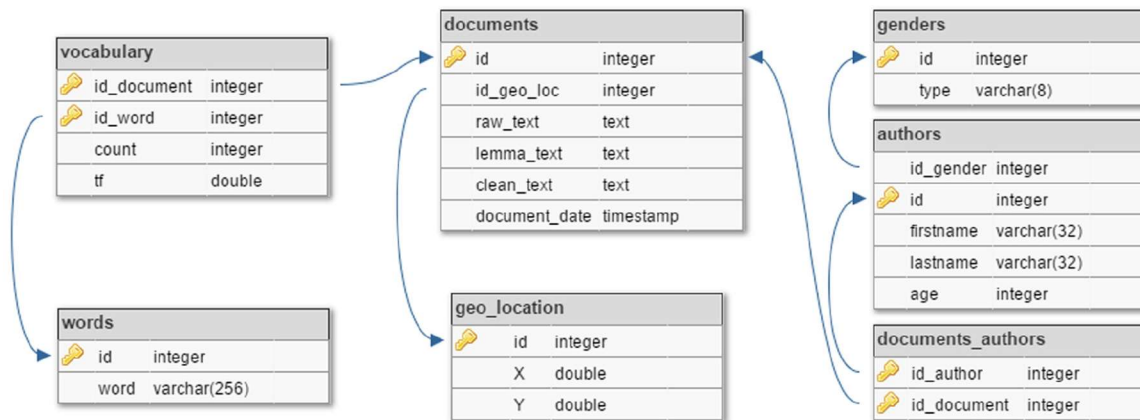
## 4. Running the query's

After getting the setup done I imported the JSON Data Sets and I made 2 Python scripts converting them into CSV files. I did this because I am going to use the default CSV LOADER from VoltDB as it is optimized for import/export operations.

The scripts : JsonToSqlRegularConverter.py and JsonToSqlStarConverter.py take as input a Json file with the format as those given in the project statement and create files with multiple CSV corresponding to both databases schemas :

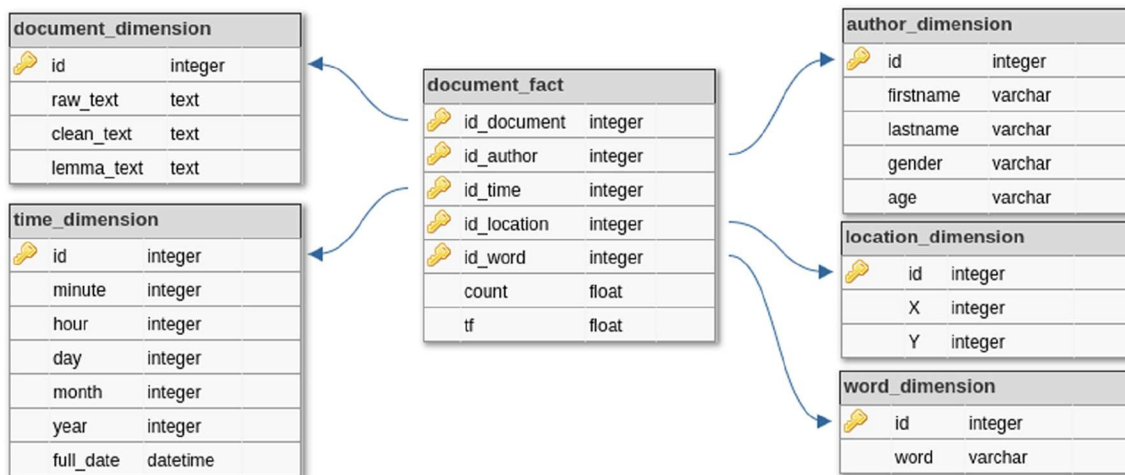
## REGULAR :

C.-O. Truică et al. / Future Generation Computer Systems 85 (2018) 60–75



## STAR:

C.-O. Truică et al. / Future Generation Computer Systems 85 (2018) 60–75



After having the CSV file, I import them in the containers in the docker, in the cluster it's enough to import it on the leader node, by using this command :

```
docker cp <source_path> <container_id>:<destination_path>
```

And then I load the databases corresponding to the test I want to run using TablesLoader.py

And then Load the data from the CSV into the tables using :

RegularDataLoader.py or

StarDataLoader.py

In order to clean the tables and load new data I use :

ClearContainerTabls.py

After the tables are loaded I use the script :

execution.py to run the tests.

The command I run the queries with is :

```
time sqlcmd --query-timeout=30000 <sql script>
```

And I extract the “Real” variable after the run.

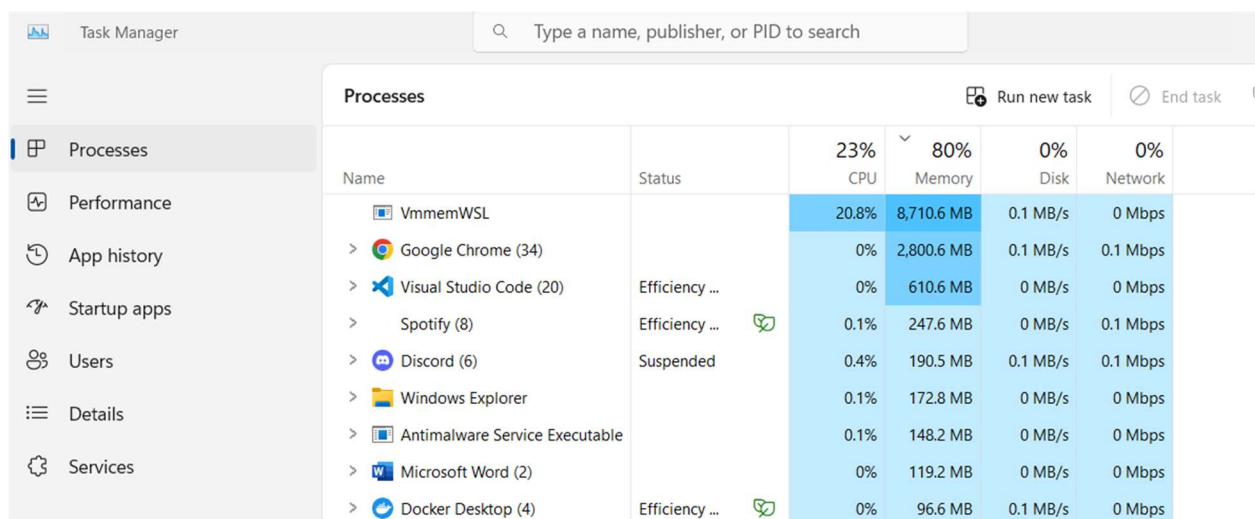
All the SQL scripts that I used ar in the Topk\_doc, Topk\_keywords and BD\_SQL\_scripts folders.

## 5. Problems and observations

### 5.1 Problems

As VoltDB is an in-memory database, it uses a lot of RAM.

Here is an example the queries for 1000k running :



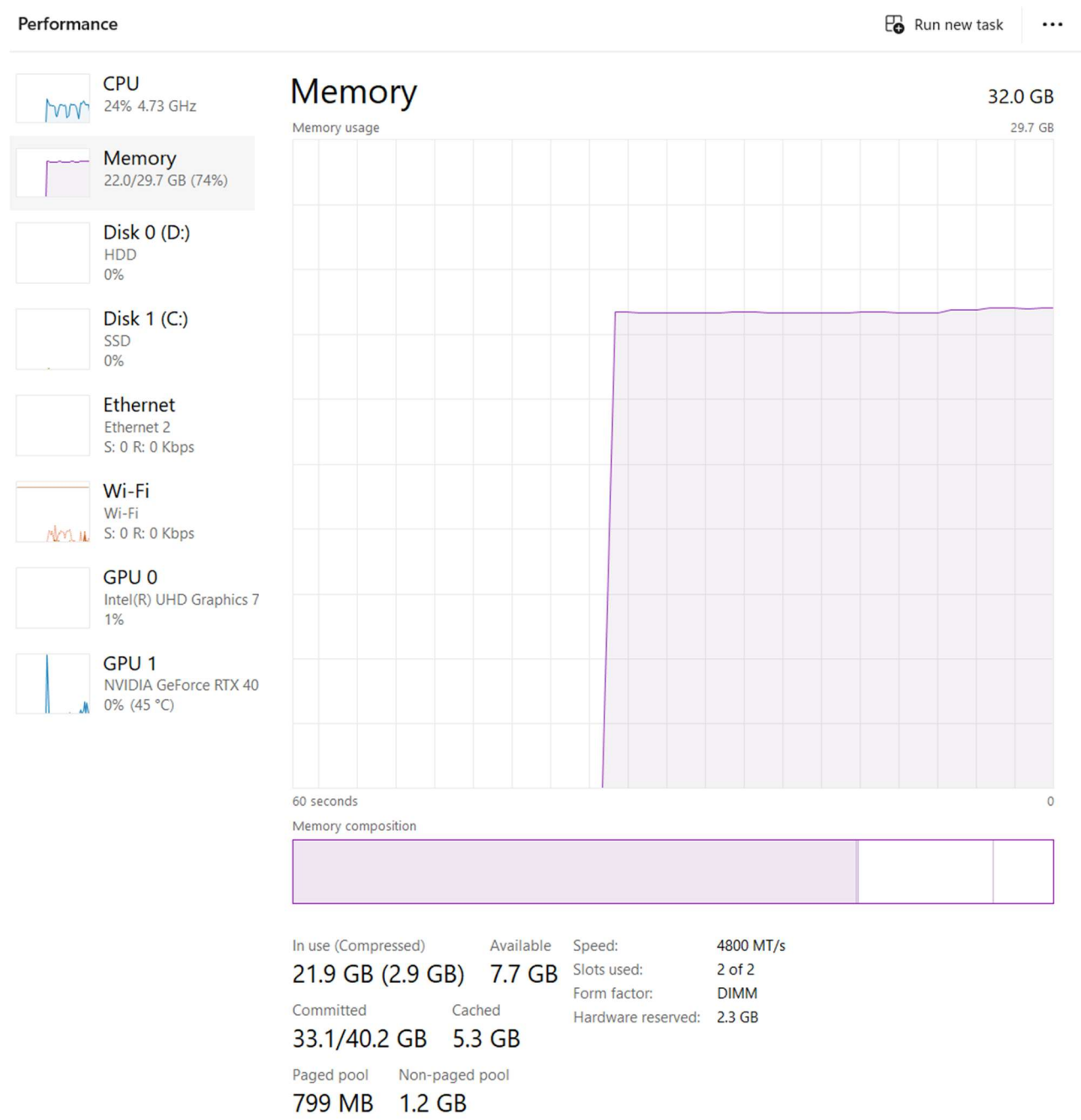
Name	Status	23% CPU	80% Memory	0% Disk	0% Network
VmmemWSL		20.8%	8,710.6 MB	0.1 MB/s	0 Mbps
> Google Chrome (34)		0%	2,800.6 MB	0.1 MB/s	0.1 Mbps
> Visual Studio Code (20)	Efficiency ...	0%	610.6 MB	0 MB/s	0 Mbps
> Spotify (8)	Efficiency ...	0.1%	247.6 MB	0 MB/s	0.1 Mbps
> Discord (6)	Suspended	0.4%	190.5 MB	0.1 MB/s	0.1 Mbps
> Windows Explorer		0.1%	172.8 MB	0 MB/s	0 Mbps
> Antimalware Service Executable		0.1%	148.2 MB	0 MB/s	0 Mbps
> Microsoft Word (2)		0%	119.2 MB	0 MB/s	0 Mbps
> Docker Desktop (4)	Efficiency ...	0%	96.6 MB	0.1 MB/s	0 Mbps

Therefore the main limitation of this database is the RAM memory the host device has.







Because of this, tests larger than 1000k document entries (with a few exceptions) would take up to much RAM upon loading and crash the entire system.

Out of the 32 GB of ram I have, 29.7 Are usable. 8 are used by the operating system and I allocated the rest into running the containers.



Crash example :

Processes		 Run new task  End task 			
Name	Status	5% CPU	81% Memory	2% Disk	0% Network
 VmmonWSL		1.1%	14,004.9 ...	6.8 MB/s	0 Mbps

```
author dimension completed.
SQL script executed successfully:
Read 1500000 rows from file and successfully inserted 1500000 rows (final)
Elapsed time: 24.259 seconds
Invalid row file: /opt/voltdb/csvloader_DOCUMENT_DIMENSION_insert_invalidrows.csv
Log file: /opt/voltdb/csvloader_DOCUMENT_DIMENSION_insert_log.log
Report file: /opt/voltdb/csvloader_DOCUMENT_DIMENSION_insert_report.log

document dimension completed.
Error executing SQL script:
document facts completed.
Error executing SQL script: OCI runtime exec failed: exec failed: unable to start container process: error executing setns process: exit status 1: unknown

location dimension completed.
Error executing SQL script: OCI runtime exec failed: exec failed: unable to start container process: error executing setns process: exit status 1: unknown

time dimension completed.
Error executing SQL script: OCI runtime exec failed: exec failed: unable to start container process: error executing setns process: exit status 1: unknown
```

One abnormality that I met on my system is that when the date are loaded it adds by default 4 hours. I corrected for this error running the following query on the tables with the dates:

```
UPDATE <table_name> SET <table_field> = DATEADD(MILLISECOND, -
14400000, <table_field>);
```

## 5.2 Observations

VoltDB has good protections against failure, especially regarding memory usage failures. At 80% of the RAM capacity used it enters into a READ-ONLY mode in order to save the data it has and prevent a system fail. In case this protection is overwritten, it will start to use disk memory. This decreases the performance a lot.

The root xml location for VoltDb is different than the usual locations of xml files. It is located at : /var/voltdb/voltdbroot/config/deployment.xml

## 6. SQL Queries

When it comes to the queries executed, in order to be able to translate them in SQL in a form that VoltDb could run them I had to play around with some extra tables. Mainly, as VoltDB does not allow aggregations like : SUM ( SELECT ...) or the

WITH clause, I made external tables that store the required data that is used in the main query.

## 7. Sources and data

All the programs used are in the following git repository:

[https://github.com/BogdanNC/VoltDB\\_Benchmark](https://github.com/BogdanNC/VoltDB_Benchmark)

All the times for the test are on the following drive folder:

[https://drive.google.com/drive/folders/1zZnYDq\\_H5cgaYXvmTDu4gO-sIOCfKePv?usp=drive\\_link](https://drive.google.com/drive/folders/1zZnYDq_H5cgaYXvmTDu4gO-sIOCfKePv?usp=drive_link)

## 8. System specifications

I have been running the project on Windows 11 Pro,

Processor 13th Gen Intel(R) Core(TM) i9-13900K, 3000 Mhz, 24 Core(s), 32 Logical Processor(s)

System Type x64-based PC

Installed Physical Memory (RAM) 32.0 GB

Total Physical Memory 29.7 GB

Item	Value
OS Name	Microsoft Windows 11 Pro
Version	10.0.22631 Build 22631
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	BOGDAN_PC
System Manufacturer	Dell Inc.
System Model	XPS 8960
System Type	x64-based PC
System SKU	0BC0
Processor	13th Gen Intel(R) Core(TM) i9-13900K, 3000 Mhz, 24 Core(s), 32 Logical Pro...
BIOS Version/Date	Dell Inc. 2.7.0, 5/29/2024
SMBIOS Version	3.4
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	Dell Inc.
BaseBoard Product	0XD433
BaseBoard Version	A00
Platform Role	Desktop
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\Windows
System Directory	C:\Windows\system32
Boot Device	\Device\HarddiskVolume3
Locale	United States
Hardware Abstraction Layer	Version = "10.0.22621.2506"
User Name	Bogdan_PC\bogda
Time Zone	GTB Daylight Time
Installed Physical Memory (RA...	32.0 GB
Total Physical Memory	29.7 GB
Available Physical Memory	10.8 GB
Total Virtual Memory	34.9 GB
Available Virtual Memory	5.87 GB
Page File Space	5.27 GB
Page File	C:\pagefile.sys
Kernel DMA Protection	On
Virtualization-based security	Running
Virtualization-based security ...	
Virtualization-based security ...	Base Virtualization Support, Secure Boot, DMA Protection, UEFI Code Read...
Virtualization-based security S...	Hypervisor enforced Code Integrity
Virtualization-based security S...	Hypervisor enforced Code Integrity

## 9. References

- [1] Ciprian-Octavian Truică, Elena-Simona Apostol, Jérôme Darmont, Ira Assent. *TextBenDS: a generic Textual data Benchmark for Distributed Systems*. Information Systems Frontiers, 23:81-100. ISSN 1387-3326, Springer. February 2021 (published online March 2020). DOI: 10.1007/s10796-020-09999-y
- [2] Ciprian-Octavian Truică, Elena-Simona Apostol, Jérôme Darmont, Torben Bach Pedersen. *The Forgotten Document-Oriented Database Management Systems: An Overview and Benchmark of Native XML DODBMSeS in Comparison with JSON DODBMSeS*. Big Data Research, 25:1-14(100205), ISSN 2214-5796, July 2021 DOI: 10.1016/j.bdr.2021.100205
- [3] Ciprian-Octavian Truică, Jérôme Darmont, Alexandru Boicea, Florin Rădulescu. *Benchmarking Top-K Keyword and Top-K Document Processing with T2K2 and T2K2D2*. Future Generation Computer Systems, 85:60-75, ISSN 0167-739X, August 2018. DOI: 10.1016/j.future.2018.02.037
- [4] <https://docs.voltdb.com/v11docs/>
- [5] <https://stackoverflow.com/>