Matrix representations in Scylla database

1 Introduction

The problem of storing matrices in Scylla database had us face the inevitable question: what is the most effective way to represent our data? Among the most popular representations of sparse matrices used in numeric calculations are:

- dictionary of keys (**DOK**)
- list of lists (**LIL**)
- coordinate list (**COO**)
- compressed sparse row (**CSR**)

Each of the representations has its own advantages and disadvantages. Scylla, as a data storage system, has its limitations we expect overhead caused by collection of data through queries instead of direct access, and need to accomodate for the peculiar partitioning feature (which, to some degree, can also be taken advantage of). This is why we had to treat the aforementioned formats as an inspiration rather than direct specification. This document aims to summarize our representations based on each of the models listed above, describing all their advantages and disadvantages found to-date. The summary will serve as a basis on which the most suitable representation will be picked for future experiments.

2 Dictionary of keys

3 List of lists

4 Coordinate list

Blah blah some text. [WIP]

I used blocks, and each block is represented with a list. This format is pretty meh. With blocks you need to do lots of data collection and multiplications to get the result for a single cell (admittedly, though, the overhead should not be too high for this, however, we'd need to tweak the database extraction queries. It's quite easy to build result matrices. We should be able to query portions of data so large that we can ignore the cost of queries. Block operations works rather intuitively.

We can transpose blocks easily in RAM efficient storage. Currently storing values in Scylla's $set < tuple < int, int, int >>$ type may not be the best idea. It is, however, true to the idea of COO. Scylla's *lists* are allegedly even worse.

Perhaps it would be better to combine blocks with dictionary of keys? Wouldn't it be worse efficiency-wise, though, with data scattered all over a partition, instead of keeping it close together? IDK.

5 Compressed sparse row

```
#include <stdio.h>
int main(void)
{
    printf("Hello World!");
}
```