Project Wendelin

The Wendelin project started in the beginning of 2015 with Nexedi as consortium leader in charge of managing the development of a big data solution "made in France". Wendelin is based on open source software components and combines a number of widely used libraries for data analysis, modification and visualization. The project also includes the development of prototype applications in the automotive and green energy sectors underlining its purpose of being immediately applicable for the development of industrial solutions.

Motivation

Wendelin was bumped to version 0.4 alpha recently with bugfixes and faster installation. The Wendelin core itself has just been upgraded to version 0.5 bringing some interesting new features. It now supports two different formats to store data: ZBlock0 and ZBlock1. The major difference between these two formats is that ZBlock0 stores just one object in the database, which makes it faster at the price of using a lot of storage space, while ZBlock1 stores many objects, but this causes it to lose in speed when compared to the other format. As one of the main features of Wendelin are its out-of-core capabilities, which allow it to easily extend computation capacity beyond the limits of available hardware in a cluster, the time has come to run some performance tests and see how it compares to a well known and widely used competitor: MariaDB.

The Test

We ran tests to compare the read performance of Wendelin, pure NumPy (both in-memory and using memory maps) and MariaDB’s SQL functions. Write speed was also measured for Wendelin. These were the different tool sets used and how they will be later referenced:

- **wendelin.numpy**: NumPy features on top of a Wendelin’s ZBigArray;
- **wendelin.pandas**: Pandas features on top of a Wendelin’s ZBigArray. This test had a monkey-patch added to Pandas, which will be explained later on this post;
- **numpy.memmap**: NumPy used with it’s memmap function to map memory addresses to a file to persist data;
- **numpy.memory**: pure in-memory NumPy;

All those tool sets were tested in a environment with cold and warm cache. Cold and warm cache have a analogy with cold and warm engine of a car. A cold cache does not help performance, it is an empty cache. While the warm cache has some values and can give a speedup.

A virtual machine provided by Vifib was used to run the tests. It has the following hardware: 4 cores, 4GB of RAM (swap was disabled), 25GB of SSD storage and runs Debian 8.2 Jessie. An all-in-one Wendelin instance of ERP5 was used and ZEO was chosen as storage backend for ZODB. This instance was installed using our "how to get started with Wendelin" tutorial. The test data consisted of 1.430.394 stock objects.

The test’s code for Wendelin and NumPy can be found in the repository at https://lab.nexedi.com/Camata/wendelin.performance/. It is splitted in server and client codes. The server code is responsible for running the test for a specific tool set according to the parameters it receives. The client code is in charge of providing these parameters to trigger each tool set's test and also for cleaning the server's cache and restarting services when it's running with cold cache. The schema used for the data in the Python part can be found in the repository code and was as close as possible to MariaDB’s schema.

In the **wendelin.pandas** test a simple numpy.ndarray was used as index and Pandas was monkey-patched to avoid data copy in the DataFrame constructor by changing the _consolidate function at https://github.com/pydata/pandas/blob/master/pandas/core/internals.py#L4074 to this more simple version:

```
def _consolidate(blocks):
    return blocks
```

MariaDB was tested with the query “SELECT SUM(quantity) FROM stock;” after a cache flush and reset. System cache were properly cleaned and the service restarted for cold cache runs. The table schema was as follows:

```
CREATE TABLE `stock` (
    `uid` bigint(20) unsigned NOT NULL,
    `order_id` bigint(20) unsigned NOT NULL,
    `explanation_uid` bigint(20) unsigned DEFAULT NULL,
    `node_uid` bigint(20) unsigned DEFAULT NULL,
    `section_uid` bigint(20) unsigned DEFAULT NULL,
    `payment_uid` bigint(20) unsigned DEFAULT NULL,
    `function_uid` bigint(20) unsigned DEFAULT NULL,
    `project_uid` bigint(20) unsigned DEFAULT NULL,
    `funding_uid` bigint(20) unsigned DEFAULT NULL,
    PRIMARY KEY (`uid`)
) ENGINE=ZEO DEFAULT CHARSET=utf8;
```
Results

After 5 samples of each test were taken, the average and standard deviation in percentage were calculated. All data is measured in seconds. MariaDB's data is provided by MariaDB itself and this is the reason behind lesser precision numbers than our code's measurements. Lower numbers are always better.

### Wendelin write tests

<table>
<thead>
<tr>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZBlock0</td>
<td>10.279 ± 11.77%</td>
</tr>
<tr>
<td>ZBlock1</td>
<td>63.990 ± 7.88%</td>
</tr>
</tbody>
</table>

### MariaDB and NumPy read tests

<table>
<thead>
<tr>
<th>Tool set</th>
<th>Hot Cache</th>
<th>Cold Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>MariaDB</td>
<td>0.470 ± 6.80%</td>
<td>2.470 ± 25.70%</td>
</tr>
<tr>
<td>numpy.memmap</td>
<td>0.050 ± 7.20%</td>
<td>0.236 ± 10.21%</td>
</tr>
<tr>
<td>numpy.memory</td>
<td>0.014 ± 4.37%</td>
<td>0.013 ± 10.13%</td>
</tr>
</tbody>
</table>

### Wendelin - ZBlock0 read results

<table>
<thead>
<tr>
<th>Tool</th>
<th>Hot Cache</th>
<th>Cold Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>wendelin.numpy</td>
<td>0.034 ± 1.69%</td>
<td>3.525 ± 11.34%</td>
</tr>
<tr>
<td>wendelin.pandas</td>
<td>0.462 ± 1.68%</td>
<td>4.026 ± 6.64%</td>
</tr>
</tbody>
</table>

### Wendelin - ZBlock1 read results

<table>
<thead>
<tr>
<th>Tool</th>
<th>Hot Cache</th>
<th>Cold Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>wendelin.numpy</td>
<td>0.022 ± 0.76%</td>
<td>58.047 ± 1.42%</td>
</tr>
<tr>
<td>wendelin.pandas</td>
<td>0.226 ± 3.25%</td>
<td>58.492 ± 2.07%</td>
</tr>
</tbody>
</table>

Conclusion

First, with regards to writes, format ZBlock0 is much faster than ZBlock1. This is because of the nature of ZBlock0, which uses just one object in the database, while ZBlock1 uses many objects in the database. This was an expected result. But writes will often be executed with a big frequency and small amounts of data each time, so this is not a big problem. Also write speed can be improved by a better storage backend.

Now about tools that are not dependent on the Wendelin block format usage: NumPy with **memmap** shows itself 10 times faster than MariaDB with cold cache, which is very interesting. NumPy in-memory crushes everything, as you would expect, and takes only 0.013 seconds to process the whole data. This means we can get significant speed improvements by modifying our most time-consuming reports to take advantage of NumPy when data fits in memory, and/or its memory map...
function when data doesn't fit entirely in the memory.

The Wendelin tests (either with NumPy or Pandas) with format ZBlock0 shows that it can read very fast with cold cache - almost close to MariaDB - which is a good and motivating result for such a new library. It also shows that Wendelin with NumPy is 20 times slower than pure NumPy's memmaps. This means that by using Wendelin and a distributed storage backend for ZODB, like NEO, we can achieve performance close to NumPy's memmap. Hot cache result is 10 times faster for wendelin.numpy and about the same speed as MariaDB for wendelin.pandas.

The read test results with Wendelin using format ZBlock1 shows that it is much slower than ZBlock0 with cold cache, but this was already known and explained earlier in the write tests and in the motivation section. With hot cache almost all tool sets behaved the same, except for wendelin.pandas, which still has a tiny overhead introduced by Pandas processing data in its block structure.

During tests execution the data size was measured for each tool set. For Wendelin-based tool sets, wendelin.pandas and wendelin.numpy, the data size was 354 megabytes, measured with `ls -la` before and after data was written. For MariaDB the data size was 342 megabytes, as provided by the column `data_length` of the query `SHOW TABLE STATUS WHERE NAME == 'stock';`. The data size for pure NumPy tool sets, numpy.memory and numpy.memmap, was exactly 250 megabytes.

But there is still room for improvements. Our team is currently very active and performance gains are expected in the next versions. Optimizations are not only coming to Wendelin though, but also to all tools involved in the storage process.

Stay tuned for more information about Wendelin Exanalytics platform and its evolution! We are working hard to also upgrade the whole Wendelin platform to the next release, so stay tuned.