# Evaluating the benefits of Key-Value databases for scientific applications

Pol Santamaria[1], Lena Oden[2], Eloy Gil[1], Yolanda Becerra[1,3], Raül Sirvent[1], Philipp Glock[2], Jordi Torres[1,3]

[1] Barcelona Supercomputing Center, Barcelona 08034, Spain
{pol.santamaria, eloy.gil, yolanda.becerra, raul.sirvent, jordi.torres}@bsc.es
[2] Forschungszentrum Jülich, Jülich 52428, Germany
{l.oden,p.glock}fz-juelich.de
[3] Polytechnic University of Catalonia, Barcelona 08034, Spain

**Abstract.** The convergence of Big Data applications with High - Performance Computing requires new methodologies to store, manage and process large amounts of information. Traditional storage solutions are unable to scale and that results in complex coding strategies. For example, the brain atlas of the Human Brain Project has the challenge to process large amounts of high-resolution brain images. Given the computing needs, we study the effects of replacing a traditional storage system with a distributed Key-Value database on a cell segmentation application. The original code uses HDF5 files on GPFS through an intricate interface, imposing synchronizations. On the other hand, by using Apache Cassandra or ScyllaDB through Hecuba, the application code is greatly simplified. Thanks to the Key-Value data model, the number of synchronizations is reduced and the time dedicated to I/O scales when increasing the number of nodes.

**Keywords:** Key-Value Distributed Databases · HPC · Big Data · NoSQL.

## 1 Introduction

In recent years, the data produced in scientific workflows increased massively. One of the objectives of the Human Brain Project is the creation of a digital *brain atlas* identifying the regions of the human brain. The project support getting a better understanding of the microscopic structure of the human brain. For this purpose, Jülich researchers slice the brain into small tissues and obtain high-resolution images through 3-D Polarized Light Imaging. Since one scanner produces 1 Terabyte of data per day, a whole brain imaged at $1\mu m$ resolution generates one Petabyte worth of data.

The scans are analyzed with image processing techniques to identify cells, regions and, neuron densities. Only HPC resources are capable of processing the sheer amount of data. However, parallel file systems are the principal cause of performance and scalability issues. Besides, working with files enforces strict application workflows with synchronizations and complex code, hard to adapt under new requirements or hardware changes.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

Key-Value *(KV)* databases are widely used in data analytics and represent a realistic alternative. They are well-suited for scientific applications such as time-series or spatial data. Furthermore, they allow analyzing partial results and react, for instance, by discarding a chemical simulation as soon as a certain event occurs.

In this work, we analyze the introduction of KV distributed databases in a high-performance data-analytic from the Human Brain Project. The use case originated in the *brain atlas* after facing a real problem driven by I/O, cells detection in high-resolution brain images. The Cell Segmentation Application *CSA* analyzes TIFF images through MPI-IO [3] and saves the results on large HDF5 files [8]. Post-execution queries will later refine the results.

The *brain atlas* is built on pipelines including sequential post-processing caused by the use of files. In this scenario, KV databases would allow processing the partial results in parallel. With this document, our goal is to discuss not only the performance but also the usability, advantages, and disadvantages of KV databases in HPC workflows.

The rest of the paper is organized as follows. In Section 2 we introduce the necessary knowledge to understand this work. Afterward, in Section 3 we discuss similar approaches and research works. Section 4 describes the problem we are trying to solve, the current procedure, and our proposed solution. We evaluate and discuss both approaches in Section 5. Finally, in Section 6 we present our conclusions and describe future lines of research.

## 2 Background

To perform parallel writes and to solve synchronization issues, HDF5 relies on buffering and aggregation of multiple POSIX operations. Its concurrency is based on the Single Writer Multiple Readers *(SWMR)* model which allows concurrent reads but enforces writes serialization. MPI-IO allows parallel non-contiguous writes in HDF5 but requires a POSIX-compliant file system.

Parallel File Systems *(PFS)* have performance issues on Big Data workloads even if they are HPC-oriented, such as GPFS. PFSs are typically deployed on a dedicated cluster with a set of raid disks to maximize the throughput. Often, these filesystems are located in a separate cluster, causing high-latencies and bottlenecks, and they rely on master - many slaves architectures that do not scale. Moreover, in these situations splitting a dataset for concurrent access or managing dynamic allocations is complicated. As a result, the I/O parallelism is low, and synchronizations are frequent. Besides, keeping POSIX metadata coherent is expensive.

To reduce the complexities of working with files and the associated performance, coherency, and availability issues, Key-Value *(KV)* distributed databases were introduced. They handle data identified by a key and an associated value comprised of multiple elements. For many years they have been present in business analytic deployments but seldom in HPC. These systems delay and optimize the indexation of keys for a better throughput

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

Frequently, data analytics rely on Apache Cassandra [11], a distributed and highly scalable KV database with homogeneous architecture. An alternative is ScyllaDB [1], which targets applications with real-time requirements. They reduce the operations latency by increasing the concurrency degree through a lock-less design. The client-server communication protocol and configuration are inherited from Cassandra, which makes the transition instantaneous. Moreover, they designed the core architecture to overcome the CPU bottleneck of Cassandra with a thread-per-core and a shared-nothing architecture. Both solutions were developed to offer tuneable consistency and storage management for data analytics with commodity hardware.

To simplify the interaction with the distributed storage, we introduced Hecuba. It brings a set of tools and interfaces to simplify the interaction with distributed storage and improved performance. So far, support for Cassandra and Scylla has been built-in, enabling manipulation of distributed datasets transparently. Through Hecuba, Python applications access persistent data like regular objects stored in memory. To develop an application, the user describes the data model by extending a Hecuba class and instantiate as many objects as needed. The user can persist or retrieve the in-memory objects by giving an identifier to the constructor or calling the method *make_persistent*.

## 3 Related Work

There is an increasing requirement to save and analyze large amounts of data in real-time. With this aim, researchers have evaluated and proposed storage technologies that take advantage of hardware advancements. Based on the level of abstraction and data granularity, research has followed different directions.

On one side, there have been discussions and advancements in parallel file systems such as Lustre and GPFS. They have a considerable degree of complexity, translated into difficulties to configure, maintain and trace issues. Consequently, researchers often write and share knowledge about good practices as Cornell [17] did, or performance tuning as Latham et al. [12]. New HPC-oriented filesystems have been devised to overcome the limitations on data-intensive workloads. BeeGFS is a relevant technology that delivers great performance as in the work of Eekhoff et al. [5]. Another major topic is the convergence of data analytics with HPC-oriented file systems. For instance, Tantisiriroj et al. [16] adapted PVFS to match HDFS performance for Hadoop workloads.

Given the complexity of managing parallel access to bytes, Key-Value *(KV)* and Object stores were designed and implemented after studying common data access patterns. They abstract the data access with high-level APIs managing a set of values or complex objects. Ceph is an object storage which also provides a FileSystem API. It is widely used in the Cloud and has been deployed in supercomputing centers, but, HPC users often report integration issues. Alternatively, Intel´s DAOS and Segates´ MIRO were developed for HPC data analytic frameworks. Liu et al. [13] demonstrated that DAOS scales up to 32 nodes and outperforms Lustre and Ceph in different types of workloads.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

On the other hand, research advanced in different ways for KV data stores. In 2015, Islam et al. [10] started by placing Memcached, an in-memory KV data store, between Lustre and HDFS to enable low latency and high throughput on reads. Their work continued with Shankar et al. [14, 15] who replaced internal Memcached operations to improve the performance. This approach differs from our proposal because their storage is not durable data might be lost on failures.

Likewise, Wu et al. [18] proposed Anna, an in-memory KV data store. They claimed that Anna outperformed Redis thanks to a thread-per-core architecture that scales vertically. Recently, Wu et al. [19] introduced an adjustable replication factor based on access frequency with significant benefits. In this work, they also evaluated an integration with enterprise storage obtaining promising results. In our future work, we will consider Anna an alternative.

In 2015, Greenberg et al. [7] developed an HPC-oriented KV database to take advantage of specialized hardware which outperformed Apache Cassandra. However, further experimentation would be needed to test the behavior in a more realistic scenario since only a uniform distribution was used.

Our research on KV databases for scientific applications started with Hernandez et al. [9] where we evaluated scientific queries on Cassandra. We analyzed a molecular dynamics simulation in real-time with a significant speedup compared to traditional files. Then, Artigues et al. [2] proposed Qbeast based on our previous work on D8trees [4], a distributed multi-dimensional index generated at runtime which can be paired with Cassandra. These works were data analytics oriented while this paper focuses on data generation and usability.

Related to the cell segmentation on images in HPC, Gabriel et al. [6] tested the PVFS and NFS file systems to provide the images and preserve the results. They discovered that persistent storage limits the scalability while being one of the most expensive tasks. On the other hand, Zhang et al. [20] proposed a framework to reduce the time needed to process and analyze the cell segmentation results. They stated the need to move to short-cyclic analytical workflows with queries analyzing partial results.

## 4 Cell segmentation on large-scale brain images

The segmentation and morphological characterization of neuronal cell bodies enable the study of the cells distribution, density, and other features. Retrieving and classifying the cell's information represents a large data analytical challenge. Even a small brain region covering a few millimeters contains several hundred thousands of neurons, while the number of neurons in a complete human brain is almost 90 billion. Each human brain is sliced in up to 2,500 sections, which must be individually scanned and processed. A single image has a size of around $80,000 \times 100,000$ pixels, is stored with one channel which results in 20-30 GB.

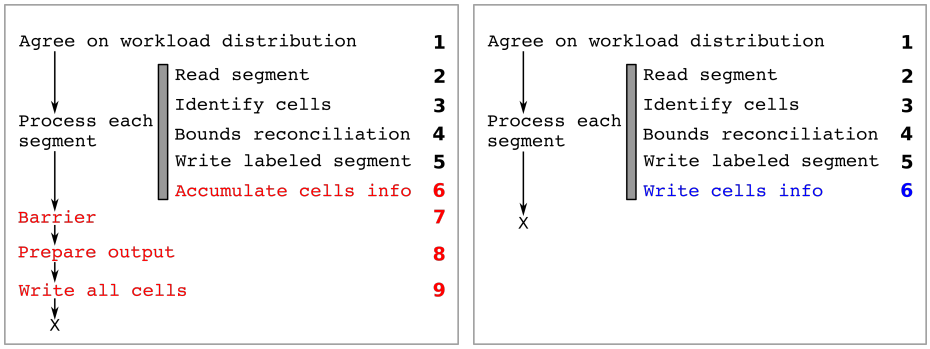
In this section, we give a short overview of the original cell segmentation application. Note that we use an already optimized version with a reduced number of synchronization and MPI-IO to allow a fair comparison. Finally, we describe the changes introduced to adapt the application to the Key-Value databases.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

### 4.1 Original Application using HDF5 files

The brain-images are provided in the BigTIFF format, a TIFF extension that enables files larger than 4GB. It has an internal multi-page hierarchy where each page has a copy of the image with different resolution. The page organizes the data as a bitmap divided into tiles or stripes. In this case, 2D tiles of 256x256 pixels with elements encoded as 16 bits are used, resulting in tiles of 128KB. Big tiles (>4KB) improve the throughput of sequential operations at the expenses of storage requirements because the edging tiles are filled with padding bytes.

The Cell Segmentation Application *(CSA)* is written in *Python* and parallelized with `mpi4py`, an interface to MPI. Figure 1a illustrates the original workflow based on HDF5. Steps *(1-5)* are inherent to the application and independent of the storage solution. Steps *(6-9)* indicate specific operations required by HDF5, and vary depending on the storage system as reviewed in Section 4.2.




(a) CSA on HDF5 steps (b) CSA on Key-Value databases steps

Fig. 1: Cell Segmentation Application workflow

The application starts by creating an empty, uncompressed, HDF5 file through `h5py`. The *labeled image* array is the first dataset to be created. Each position of the *labeled image* array will have the identifier of the cell identified in the corresponding image, and zero otherwise. At the end of the execution, the *cells table* dataset is generated, which includes the identifiers, bounding boxes, and centroids of the cells. A resizable dataset for the *cells table* would allow partial writes. However, they become slower, and memory consumption is barely affected. Besides, it requires a chunk tunning process based on the underlying file system and future data access patterns. On the contrary, we know that the *labeled image* dataset will have the shape as the original image. Therefore, we preallocate the dataset for optimized write performance.

As a parallelization strategy, the application splits the image into multiple fix-sized segments *(Line 1 in Figure 1a)* with a small overlap. The segment size represents a trade-off between memory consumption, performance and workload imbalance. We determined that memory consumption follows a quadratic increase with the size of segments. These segments are distributed evenly in a

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

randomized manner to reduce the imbalance. After agreeing on the work distribution, each process gets a fixed range of cells IDs, which is wide enough to accommodate more cells than potentially found.

Next, every process works on its segments by fetching the data through the *Pytiff* module *(2)*. The cells are identified along with their bounding box and the rest of the characteristics *(3)* using *Sklearn*, `Scipy` and *OpenCV*. Then, the bounds reconciliation *(4)* discards the cells whose centroid does not belong to the segment. This step is necessary to remove the cells identified twice in the overlapping area by different processes. Then, the index of the current segment is written *(5)* into the *labeled image* through MPI-IO. Because the size of the *cells table* dataset is unknown in advance and it represents a small fraction of the output, the data is kept in memory *(6)* temporarily.

After completing the analysis, all processes wait on a barrier *(7)*. They need to reach an agreement *(8)* on the *cells table* layout to avoid overlapping data during the collective MPI-IO write *(9)*. Half the steps are only necessary to work with files, which requires complex APIs to synchronize or preallocate storage. We analyze these calls and their code in Section 4.3.

### 4.2 Implementation using Key-Value Storage

The introduction of a Key-Value *(KV)* storage required minimal modifications to the original HDF5 application described in Section 4.1. The original workflow in Figure 1a has been simplified to Figure 1b.

In this approach, we save both datasets to the distributed KV database managed by Hecuba. We write the cells' characteristics *(6)* as soon as they are identified *(3)* and filtered by the bounds reconciliation phase *(4)*. Consequently, the *cells table* and *labeled image* are written one after the other and they can be accessed by other applications at this point. The operations *(6-9)* in Figure 1a that introduced synchronizations are replaced with a simple step identified as *(6)* in Figure 1b, greatly simplifying the code.

With regards to the data model, we considered saving the cell identifier for each pixel using the x-y coordinates as keys. Early experiments demonstrated that the number of small IOPS was huge and resulted in a performance impact, and we opted to store the labeled image as an array.

### 4.3 Interfacing with Hecuba versus HDF5

We introduced some minor changes to replace the storage interface from HDF5 to Hecuba. First, the initialization of the HDF5 files shown in Listing 2 and the description of the data model is replaced with the code in Listing 1, resulting in cleaner code. The data model definitions can be defined on separate files and imported to reuse the data model in subsequent analytics.

With Hecuba, we save each dataset as distributed Numpy NDArrays. Note that it is not necessary to explicit the internal type nor the shape since Hecuba extracts them at run-time. Describing the data model requires learning a simple

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

```
from hecuba import StorageDict

class CellData(StorageDict):
  '''
  @TypeSpec <<rows:int,cols:int>,
             data:numpy.ndarray>
  '''
class Labels(StorageDict):
  '''
  @TypeSpec <<rows:int,cols:int>,
             data:numpy.ndarray>
  '''
# Create Hecuba persistent dictionaries
cell_table = CellData(ksp+'.cell_data')
data_table = Labels(ksp+'.labeled_data')
```

Listing 1: Hecuba objects in CSA

```
import mpi4py as MPI
import h5py

# Create HDF5 file with the MPI communicator
out_file = h5py.File(out, "w", driver="mpio",
                             comm=MPI.COMM_WORLD)
# Create and preallocate the indexing dataset
out_file.create_dataset("pyramid/00",
                        shape=img.shape,
                        dtype=np.int64)
# Create dataset after processing the image
out_file.create_dataset("cells",
                shape=(n_cells, n_features),
                          dtype=np.float32)
```

Listing 2: HDF5 objects in CSA

and intuitive syntax. On the contrary, HDF5 requires more information to pre-allocate and optimize the file and has many optional arguments. Besides, MPI has to be initialized before the creation of HDF5 files.

```
# Persistent objects already instantiated
for (row, col, indexes) in local_slices:
  # ...
  # Write labeled data (numpy.ndarray)
  data_table[rows[0],cols[0]] = labels
  # Write cells information (numpy.ndarray)
  cell_table[rows[0], cols[0]] = table
```

Listing 3: Hecuba CSA main code

```
# Persistent objects already instantiated
total_table = None
for (row, col, indexes) in local_slices:
  # ...
  total_table = np.concatenate((total_table,
                                table), axis=0)
  # Write labeled data
  out_labeled[row[0]:row[1],
              col[0]:col[1]] = labels
table_length = np.array(total_table.shape[0],
                        np.uint64)
# Synchronize to unify the cells information
comm.Barrier()
len_cells = np.zeros(comm.size,
                     dtype=np.uint64)
comm.Allgather(table_length, len_cells)
offsets = [0] + np.cumsum(len_cells).tolist()
n_cells = len_cells.sum()
n_features = total_table.shape[1]
# Create and preallocate the cells information
out_file.create_dataset("cells",
         shape=(n_cells, n_features),
                dtype=np.float32)
# Write cells information
out_file["cells"][offsets[comm.rank]:
         offsets[comm.rank + 1]] = total_table
```

Listing 4: HDF5 CSA main code

The line of code used to pass the data to either Hecuba or HDF5 is almost identical. However, to do so, Hecuba does not require synchronizations or the collective initializations displayed in Listing 3. At the end of the execution, the HDF5 code introduces a global MPI barrier to unify the cells characteristics metadata as in Listing 4. It is necessary to allocate the dataset and to perform a collective write to non-overlapping areas.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

## 5 Evaluation

In this section, we study the performance of the Cell Segmentation Application *(CSA)* supported by HDF5 files and Key-Value *(KV)* distributed databases. The objective is to identify the key aspects that limit the CSA from matching the data processing requirements of the Human Brain Project. Also, we demonstrate that KV distributed databases satisfy the HPC needs while offering a pleasant API for scientists.

First, we present the hardware setup and the experiments characteristics. Next, we describe how the experimentation evolved based on the results obtained. With this idea, we present a brief description and motivation of each experiment alongside the results obtained and an analysis.

### 5.1 Experimental setup

The Juron Pre-Commercial Procurement (PCP) is one of the pilot systems granted to the Human Brain Project. The cluster features fast local NVMe disks facilitating the deployment of a wide variety of distributed storage technologies. For this reason, we selected the Juron PCP to run our experiments.

With regards to the storage technologies, we evaluate Apache Cassandra and ScyllaDB as Key-Value *(KV)* databases. Their results will be confronted with storing HDF5 files on a GPFS node pool. GPFS features a 2GB pagepool on each node acting as a cache. On the contrary, Hecuba implements a client-side write queue for Cassandra and Scylla of 4 MB per process, resulting in 76 MB per node. However, both KV databases acknowledge writes only when they are persisted in disk.

The cluster features 19 high-density memory nodes with 2xPower8 processors clocked at 3.5Ghz. Each processor has ten cores with a Simultaneous Multithreading (SMT) of 8, potentially executing 160 threads per node concurrently. Every node has a 1.6 TB Ultrastar SN100 NVMe local disk available to users. Additionally, 256GB DDR4 and 4 NVIDIA Tesla P100 interconnected with NVLink are available on each node.

Regarding the inter-node communication, a 100Gbps CoonectX-4 EDR Infiniband interconnect is used. Besides, the PCP system is connected to the Jülich GPFS system (JUST) to offer centralized storage to users. Likewise, multiple clusters rely on the storage provided by the JUST system, built on top of 22 Lenovo Distributed Storage Solution and three older GPFS Storage Servers, achieving a throughput of 220 GB/s. The connection between the Juron PCP and JUST is made through two Ethernet adapters, each with two 10 Gbps Ethernet ports, delivering a speed of 40 Gbps. The number of concurrent users is highly variable and the service does not offer on-demand scalability.

Early experiments performed best by running a process on each physical core and taking advantage of the SMT with underlying threads. For this reason, we run the original HDF5 application by launching 20 processes on each node with MPI. On the contrary, we left a physical core free when deploying Apache Cassandra or ScyllaDB resulting in 19 application processes on each node. The

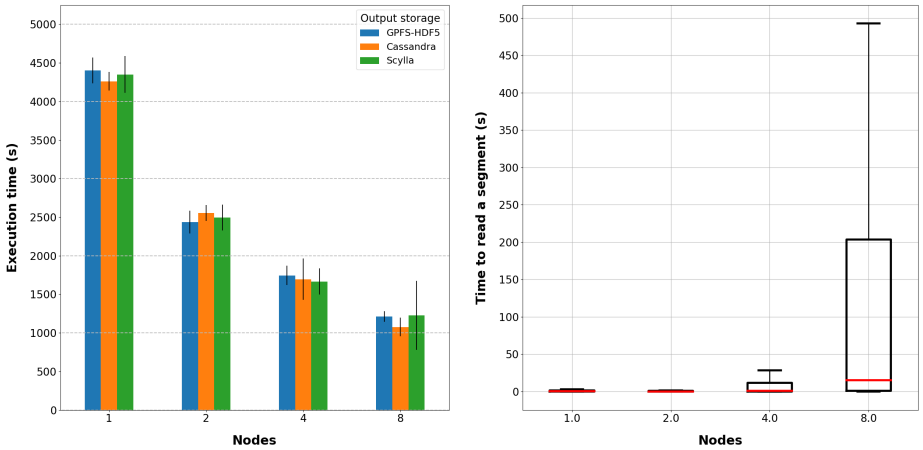
ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

OS schedules the processes on cores, neither the application nor the database performs CPU pinning. This decision has been made to compare all solutions using the same amount of physical resources. The local NVMe disks are used to deploy the KV databases.

To evaluate the workflow, we selected a representative image from the dataset with a size of 24GB and producing an output file of 87GB in the HDF5 format containing the information of 16.6 Million cells individuated.

### 5.2 Performance Results

The first set of experiments illustrate the performance of the original Cell Segmentation Application *(CSA)* and the consequences of introducing Key-Value distributed databases. Figure 2a reports the CSA execution times when writing the output into HDF5 on GPFS, Scylla and Cassandra. As already mentioned, the input images are stored on the JUST GPFS, which also supports the HDF5 files.




(a) CSA execution times (b) GPFS read times *(output to Scylla)*.

Fig. 2: CSA: Input from GPFS and output to different storage solutions.

We evaluated the scalability from 1 to 8 nodes, running the HDF5 setup with 20 to 160 processes, and the KV counterpart with 19 to 152 application processes. We observed that the CSA does not scale, and it is subject to high variance without any visible performance pattern.

To understand these results, we measured the time dedicated to I/O. In every configuration, we found out that reading image segments from GPFS was the slowest operation and the main contribution to variability. For instance, the configuration whose performance varies the most in Figure 2a is Scylla with 8 nodes. In Figure 2b, we can appreciate the correlation between the variability and the GPFS read latencies of said execution. The measured read latencies grow

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

exponentially when increasing the number of nodes. After a brief investigation, we found out that the GPFS itself caused the variance, and not the network, which is sufficiently overprovisioned.

We can compare the IOPS performed with the performance metrics obtained with the FIO benchmark. The CSA usually retrieves segments of 5200x5200 pixels made of tiles of 256x256 pixels. Therefore, each read operation accesses 441 tiles of 128 KB, and thus, fetches a total of $\sim 55MB$. On the other hand, when the application starts, every process asks for a segment and produces more than 50 thousand read operations which the GPFS is unable to serve in a reasonable time. The read latencies observed for 8 nodes in Figure 2b indicate that the lowest throughput obtained on GPFS is 106KB/s and the average is 527KB/s. The results are consistent with the FIO benchmark which reported 1337 IOPS of 4 KB or 1058 IOPS of 64 KB on GPFS.

Also, confronting Figure 2b with Figure 2a, we can see that the reading time would represent only 25.6% of the execution time if the workload were perfectly balanced. Since 92.5% of the processes analyze three slices and the rest only two, we can say that the average processing time of a segment, including reading and writing, is 409 seconds. However, we find read operations taking $\sim 500\,seconds$, meaning that sometimes fetching a single segment from GPFS took longer than processing a complete segment on another process.

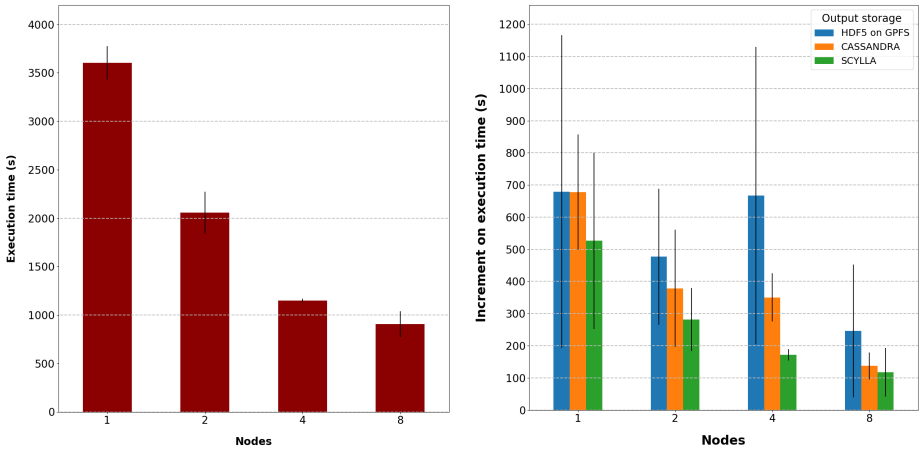
The GPFS performance issues are also related to the tiling size, which is too small for the GPFS to take full advantage of parallel reads. Synthetic tests demonstrated that tilling the image at 5200x5200 pixels, and thus, performing one read request, improved the read performance but not enough to compensate the task of reformatting the image and the extra space needed. Besides, since the input image segments are read while writing the *labeled image* segments, the GPFS pagepool is ineffective and evictions are continuously triggered. As a result, writing the *labeled image* dataset can result in slow operations which pollute the GPFS distributed pagepool and interfere with reads.

All in all, we can say that the GPFS introduces large amounts of variability and is unable to serve the input images. The causes are an inappropriate tilling of the input image, the large number of small IOPS and its inability to scale on-demand. On the other hand, the CSA itself introduces variability by randomly distributing the segments to processes.

After the initial tests with GPFS, we decided to obtain a baseline performance without the storage technologies, and we disabled the output in the CSA. Additionally, we distributed a copy of the image to all local NVMe disks. We considered moving the image to a ramdisk, but it might have interfered with the memory usage. Besides, the NVMe already provided sub-microsecond read operations. Figure 3a reports the results illustrating a lower bound of the achievable performance of the provided CSA. The variability has been reduced in all scenarios but when storing HDF5 files on GPFS. Reading the images from the NVMe disks compared to GPFS show a speedup of $\sim 2400$ of the read latencies.

In Figure 3a we can appreciate that the CSA does not scale even under ideal I/O conditions. The main cause is the trade-off between data granularity and

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30




(a) Lower bound of the CSA. (b) Storage impact on execution times.

Fig. 3: Fetching images from local NVMe disks.

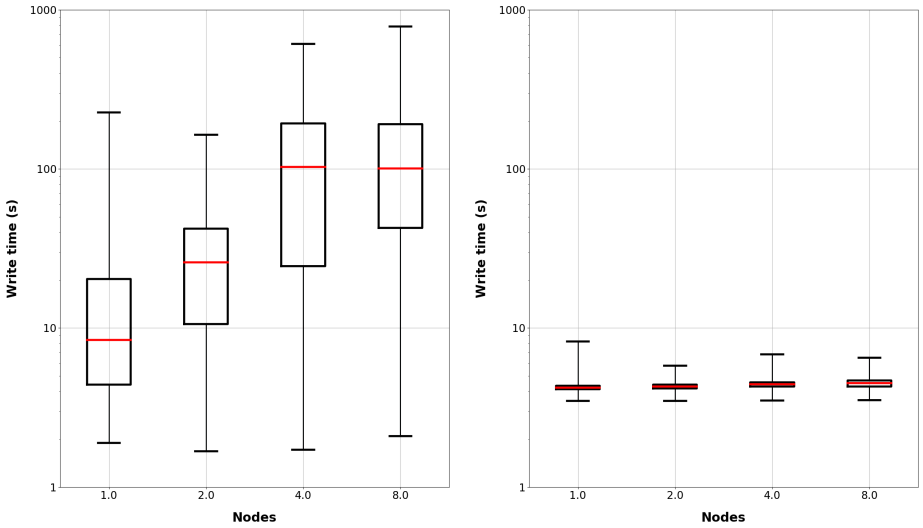
cell reconciliation time. By having segments of 5200x5200 pixels, the data can be easily imbalanced even with a random distribution. However, if we reduce the size of the segment, the cells boundaries reconciliation time grows exponentially. As we add nodes, the imbalance has a greater impact on the execution times.

We repeated the cell segmentation tests by placing the input image on the local NVMe disks and storing the output to HDF5 on GPFS, Scylla and Cassandra. We took this decision after observing the reduction in the variability and the improved read latencies. As a result, we reduced the execution time and variability. To understand the influence of the I/O, Figure 3b illustrates the difference between the cell segmentation when reading from NVMe and the lower bound obtained previously. We opted for this analysis because by only measuring the time dedicated to I/O we would not have considered the side effects of having threads doing background operations.

We can observe that the impact of I/O decreases with the number of nodes. Also, the variance of the results obtained with HDF5 on GPFS is significantly higher than the rest of the configurations. Indeed, the I/O time did not decrease with GPFS when running on four nodes; it took almost the same as with a single node. Cassandra behaves better than GPFS, but Scylla obtains the best results. We should keep in mind that reading the image now interferes with the KV databases since both rely on the NVMe disks. To analyze the performance differences, we decided to take a look at the average time dedicated to write.

The CSA performs 468 write operations to store the labeled segments with 32-bit elements. Around 400 operations write arrays of 5000x5000 elements resulting in ∼100 MB. The remaining write operations write less information. In Figure 4 we report the time dedicated by the user code to write the labeled segment at each iteration. For simplification, we only included the GPFS and Cassandra setups, since Scylla performed similarly to Cassandra but with lower average latencies.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30




(a) HDF5 on GPFS (b) Hecuba writes on Cassandra

Fig. 4: Labeled segment writes distribution *(Log10 scale)*.

In Figure 4 we see that the mean write time on GPFS and its variance increase with the number of nodes. On the contrary, Cassandra achieves constant write times when scaling horizontally with little dispersion. The slowest write operations were observed on GPFS, reaching 600 seconds. Likewise, the fastest write operations were on GPFS, reaching subsecond write execution times. Cassandra reports write operation times in the order of 4 seconds. Since the interaction with Cassandra is done through the Hecuba prototype, we expect that further code optimizations should reduce the latencies. We also conclude that GPFS is unable to support the load with the addition of nodes.

The higher number of IOPS that Scylla achieves, thanks to its Thread Per Core architecture, explains the performance differences between KV databases in Figure 3b. Consequently, client communication and database's processes interfere less with the application and execution times are shorter. However, HDF5 on GPFS can be faster sometimes for a variety of reasons. First, KV databases are optimized for queries rather than raw data ingestion. Moreover, they were not designed for HPC and, for instance, communication is done through TCP/IP over Infiniband. Finally, we use 4 KB working units with Hecuba, while the GPFS counterpart relies on 256 KB.

On the contrary, HDF5 uses MPI I/O to share data among the processes and write to storage efficiently through RDMA. In its turn, GPFS performs caching of 4 GB in the node's memory and writes asynchronously, unless instructed with the direct I/O flag. Hecuba was configured with a cache of 20 blocks per-process. Despite these differences, KV datastores achieve reasonable performance and notably better than the current workflow with HDF5 on GPFS. Besides, the latter is still subject to unpredictable performance despite only performing writes.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

## 6 Conclusion and Future Work

The brain atlas use case provided by the Human Brain Project proved that deploying distributed Key-Value *(KV)* databases on HPC simplifies the programmers' task and achieves better performance than working with files. KV databases such as ScyllaDB and Apache Cassandra allowed horizontal scalability of the I/O with stable performance. Besides, the interface provided by Hecuba reduced the programming efforts and removed synchronizations.

We observed that the I/O does not scale with GPFS, which introduces high variability. For instance, running the application on eight nodes, the read operations on GPFS took $\sim 61$ seconds on average with a standard deviation of $\sim 101$ seconds. These latencies grow exponentially with the number of nodes. Besides, the GPFS is affected by other factors such as the coincidental load generated by separate applications. However, we proved that the I/O scales horizontally with distributed KV databases. We also identified the algorithmic factors that prevented the application to linearly scale, especially when running in more than four nodes. The work distribution policy, the large segments, and the cost of the bounds reconciliation strategy are the major impediments.

As a future work, we plan to enhance the integration of Cassandra and Scylla with HPC systems. In particular, replacing the TCP/IP communication with faster HPC protocols should provide significant benefits. In its turn, research on Hecuba should deliver performance improvements.

With regards to the Cell Segmentation Application, further research could be beneficial. First, by evaluating the effects of placing the input dataset to distributed storage and widen the set of technologies tested. Secondly, by designing and evaluating different mechanisms for the cell boundaries reconciliation would permit shrinking the image segments without an increase in execution time. As a positive collateral effect, smaller segments would reduce the imbalance.

To demonstrate the full potential of distributed KV databases queries should be implemented and evaluated. For instance, by performing analytics with data locality on real-time. Another field of interest consists of the benefits of persisting and recovering data in scientific workloads upon failures. Both are exciting research lines to explore through a data-driven approach. Finally, in the event of dynamic scheduling, the time needed to complete the workflow could be reduced by better load-balancing the application at runtime.

## 7 Acknowledgments

This project/research has received funding from the European Unions Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 720270 (Human Brain Project SGA1) and the Specific Grant Agreement No. 785907 (Human Brain Project SGA2). This work has also been supported by the Spanish Government (SEV2015-0493), by the Spanish Ministry of Science and Innovation (contract TIN2015-65316-P), and by Generalitat de Catalunya (contract 2017-SGR-1414).

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30

## References


1. Scylla web page, www.scylladb.com
2. Artigues, A., Cugnasco, C., Becerra, Y., Cucchietti, F., Houzeaux, G., Vzquez, M., Torres, J., Ayguad, E., Labarta, J.: Paraview + alya + d8tree: Integrating high performance computing and high performance data analytics. Procedia Computer Science (Dec 2017)
3. Corbett, P., Feitelson, D., Hsu, Y., Prost, J., Snir, M., Fineberg, S., Nitzberg, B., Traversat, B., Wong, P.: Mpi-io: A parallel file i/o interface for mpi version 0.3. Tech. rep. (Jan 1995)
4. Cugnasco, C., Becerra, Y., Torres, J., Ayguadé, E.: D8-tree: A de-normalized approach for multidimensional data analysis on key-value databases. In: Proceedings of the 17th ICDCN (2016)
5. Eekhoff, A., Tweddell, B., Pleiter, D.: Beegfs benchmarks on juron: Streaming and metadata performance on openpower with nvme. Tech. rep., ThinkParQ and Jülich Forschungszentrum (Oct 2017)
6. Gabriel, E., Venkatesan, V., Shah, S.: Towards high performance cell segmentation in multispectral fine needle aspiration cytology of thyroid lesions. Comput. Methods Prog. Biomed. (Jun 2010)
7. Greenberg, H.N., Bent, J., Grider, G.: Mdhim: A parallel key/value framework for hpc. In: 7th USENIX HotStorage (2015)
8. Group, T.H.: Hierarchical data format, version 5 (2014), www.hdfgroup.org/HDF5/
9. Hernandez, R., Cugnasco, C., Becerra, Y., Torres, J., Ayguad, E.: Experiences of using cassandra for molecular dynamics simulations. In: 23rd Euromicro International Conference on PDP (Mar 2015)
10. Islam, N.S., Shankar, D., Lu, X., Wasi-Ur-Rahman, M., Panda, D.K.: Accelerating i/o performance of big data analytics on hpc clusters through rdma-based key-value store. In: 44th ICPP (Sep 2015)
11. Lakshman, A., Malik, P.: Cassandra: A decentralized structured storage system. SIGOPS Oper. Syst. Rev. (Apr 2010)
12. Latham, R., Ross, R., Thakur, R.: The impact of file systems on mpi-io scalability. In: Recent Advances in PVM and MPI (2004)
13. Liu, J., Koziol, Q., Butler, G.F., Fortner, N., Chaarawi, M., Tang, H., Byna, S., Lockwood, G.K., C., R., Kallback-Rose, K.A., Hazen, D., Prabhat, M.: Evaluation of hpc application i/o on object storage systems. In: PDSW (2018)
14. Shankar, D., Lu, X., Islam, N., Wasi-Ur-Rahman, M., Panda, D.K.: High-performance hybrid key-value store on modern clusters with rdma interconnects and ssds: Non-blocking extensions, designs, and benefits. In: IPDPS (May 2016)
15. Shankar, D., Lu, X., Panda, D.K.: High-performance and resilient key-value store with online erasure coding for big data workloads. In: 37th ICDCS (Jun 2017)
16. Tantisiriroj, W., Son, S.W., Patil, S., Lang, S.J., Gibson, G., Ross, R.B.: On the duality of data-intensive file system design: Reconciling hdfs and pvfs. In: SC (2011)
17. V., C.: How to ruin a perfectly good gpfs file system (2015), http://files.gpfsug.org/presentations/2015/CIUK-DDN.pdf
18. Wu, C., Faleiro, J., Lin, Y., Hellerstein, J.: Anna: A kvs for any scale (Apr 2018)
19. Wu, C., Sreekanti, V., Hellerstein, J.: Eliminating Boundaries in Cloud Storage with Anna (Aug 2018)
20. Zhang, X., Su, H., Yang, L., Zhang, S.: Fine-grained histopathological image analysis via robust segmentation and large-scale retrieval. In: CVPR (Jun 2015)


ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_30