Evaluation of SciDB for Image and Video Processing Tasks

Mohammed Suhail Rehman Advisor: Kayvon Fatahalian
School of Computer Science
Carnegie Mellon University

Abstract

In this report, we evaluate the use of SciDB, an array database, for image processing workloads. We explore the representation of images in SciDB, select four specific image-processing tasks to implement in SciDB, and evaluated them on an AWS cluster using a dataset of a 1000 high-resolution images. Even with a simple and non-optimized MPI baseline, we find SciDB to be an order of magnitude slower. We conclude this report by analysing our observations and propose future work that may improve SciDB's feasibility for such workloads.

1 Introduction

SciDB[41] is a computational DBMS designed to cater to the data storage needs of the scientific community. SciDB envisions to bring the convenience and performance of database management systems to the scientific community for large-scale computational workloads.

SciDB is designed to be multi-dimensional, with an array-based data model. It also provides support for data versioning and provenance which is a oft-cited feature requirement of data management systems from scientists.

SciDB has been used in various scientific domains, including geo-spatial data, genomics[40] as well as astronomy[21].

On paper, SciDB's array-based data model seems to align with how images are digitally represented. Thus, we were interested in investigating the use of SciDB for Image processing workloads. Through our work, we want to answer the following questions:

- Can we represent and store images, videos, and image patches[1] efficiently in SciDB?
- Can image processing tasks be expressed in the form of database queries on images stored within SciDB?
- What is the performance of the proposed SciDB based solution on a suite of simple image/video analysis tasks vs. a comparable distributed implementation of the same tasks?

As a stretch goal, we further wanted to evaluate the possibility of integrating existing deep network evaluation systems with SciDB storage via SciDB's UDF mechanisms. However, given our experiences with SciDB's performance in all of our initial benchmarks, we focused on optimizing the basic image processing tasks and were unable explore the use of SciDB for deep network evaluation. The interested reader will be referred to one such attempt, discussed in the related work section (§3) of this report.

The rest of the report is organized as follows: §2 talks about the motivation of this work, followed by the related work in §3. We then discuss the project implementation in §4, starting with the representation of images in SciDB in §4.4, followed by the description of the individual image processing tasks in §4.5. Our evaluation platform and methodology is discussed in §5, followed by the results and analysis in §6. Finally, we summarize our contributions in §7 and discuss the implications for future work.

2 Motivation

Image processing finds importance in various fields: from search, to robotics, to next-generation mobile applications. With the advent of smart phones and other devices, the amount of user-generated images and video generated daily is only exploding.

Large-scale image processing is being used for projects such as 3D world reconstruction from photos and live, interactive, virtual reality video. As a result, systems need to process streams of images and video and

[1] An image patch is a small square region (also known as a tile) of an image

be able to store and retrieve them in increasingly efficient ways.

Array databases such as SciDB offer the ability to store large volumes of multidimensional data and perform computation on the data in the form of database queries. One can envision a SciDB-based image retrieval solution that automatically performs the required computation to answer queries such as "`Select all images which average intensity greater than some threshold`", or using higher-order features such as "`Select all videos containing Cats`". For this to happen, we must be able to assess the ability of such systems to perform the required computation for image and video processing. In this work, we explore the ways in which images can be represented in SciDB and how low-level image processing operations can be expressed in the form of database queries.

3 Related Work

3.1 Image Storage and Retrieval

Image and video retrieval systems have been of major interest over several decades; many approaches have been discussed and studied. *Concept-based image retrieval systems*, (also known as *description-based* [20] systems) use auxiliary information and meta-data (such as image descriptive tags, image dimensions, EXIF and geolocation tags) to index images; although this approach is used to supplant systems such as web-based image/video search, its ability to locate and retrieve images and/or video is limited to the quality of information that is supplied in context or in the meta-data.

On the other-hand, there has been a lot of attention paid to *content-based image retrieval systems* (CBIR), which use image processing and classification methods to automatically classify and index images. There are many approaches in this domain, an exhaustive list is beyond the scope of this report; interested readers may refer to a fairly recent survey by Datta et. al [22] for details. Traditional methods involved the extraction of low-level features, such as edges, color and texture in images, and the use of distance measures (such as Euclidean or Cosine) to rank image search results and narrow down candidate images for retrieval.

The accuracy of these traditional methods have been surpassed by machine-learning-based approaches, most notably, deep learning[43]. Unlike conventional machine learning methods that often use only a few stages of learning, deep learning mimics the human brain; the processing is organized in a deep architecture and passes information through multiple stages of transformation and representation. By exploring deep architectures to learn features at multiple levels of abstraction in an automated fashion, deep learning methods allow a system to learn complex functions that directly map raw sensory input data to the output, without relying on human-crafted features.

Among various deep learning approaches, the *Convolutional Neural Network* (or CNN / ConvNet) approach for deep learning has recently surpassed all known methods for large-scale visual recognition, and have been adopted for use in commercial systems by Google, Facebook and Baidu. Multiple open-source deep-learning frameworks based on CNNs have emerged, such as [4, 23, 39]. However, they have been surpassed in terms of features, performance and popularity by the Caffe system[28], which I intend to study and use for this capstone project. Caffe currently uses Google's Protocol Buffers[5] to store Models and LevelDB[6] to store training and input data. Caffe has been designed to support multiple CPUs or GPUs in a shared memory context, no efficient distributed implementations of Caffe currently exist to my knowledge.

3.2 Scale-out Approaches to Image Storage and Retrieval

The methods discussed in the previous section with regards to content-based image retrieval (CBIR) are largely designed for execution in shared-memory systems or systems with GPU-accelerated support. In this section, we will review some of the research efforts into scale-out approaches for image storage, processing and retrieval systems.

Image Storage: The rise in popularity of cloud-based object storage systems such as Amazon's S3[1] has become a boon for web application developers looking for a scalable solution for BLOB storage. Similarly, a number of highly-distributed image storage systems have been built using a distributed architecture such as Twitter's Blobstore[12] and Facebook's Haystack[17] and f4[33]. These systems rely on external databases to keep track of image metadata and are simple image storage and serving systems. They are optimized to serve content to millions of users while optimizing throughput and latency, and are incapable of performing non-trivial image processing tasks on the images stored in the systems.

Scale-out CBIR and Machine Learning: The research regarding the feasibility of cluster-based distributed CBIR implementations, mostly using the Message-Passing Interface (MPI) framework is sparse, interested readers are directed to [38] for an example. Renewed interest in cluster-based CBIR emerged with the rise in popularity of Hadoop MapReduce [37, 24, 27].

Distributed frameworks for machine learning are aplenty, a example system based on Hadoop is Mahout[35]. GraphLab[31] is a distributed framework

designed specifically for iterative graph-based computation, which can be applied for certain types of machine learning systems. A fairly recent technique for scaling out machine learning computation is based on the concept of a parameter server[29], a high performance distributed key-value that is used to store model parameters for machines that are working in parallel to train and use machine learning models. CloudCV[14] is distributed image processing framework that uses a number of underlying distributed systems including GraphLab. StormCV[11] enables the use of Apache Storm for video processing by adding computer vision (CV) specific operations to Apache Storm[2].

For deep learning, Chilimbi et. al.[19] describe the design and implementation of a scalable framework called Project Adam, which relies on data serving machines providing training inputs to model training machines that asynchronously update a shared model via a global parameter server. The authors evaluated Project Adam's performance using 120 cluster nodes for image classification - this work presents a promising baseline to compare our proposed system against.

3.3 Array and Computational Databases

For my work, I intend to explore the use of multi-dimensional array databases for a variety of image processing tasks, including CBIR. In this section, I will discuss the history and evolution of array databases.

The limitations of the relational database model with regards to multi-dimensional data was understood early on[15]; the ANSI/ISO SQL standard[13] supports only one-dimensional arrays. Array data such as Images can be stored in relational databases using *binary large object* types (BLOBs), but have many disadvantages - BLOB data fields typically have poor performance as modern SQL engines are not optimized for storage and retrieval of large binary objects. More importantly, BLOBs cannot be effectively used in a query, users can simply store, retrieve or possibly query the size of BLOBs but cannot make queries against its content.

Modern web-based frameworks such as Django [25] rely on a two-tiered approach to storing images: Image information and metadata that is required for the application is stored in a traditional database, which typically contains a location path (file or URL) that can be used to retrieve the actual binary contents of the image. As discussed in §3.2, several internet-scale companies have built special-purpose distributed storage systems to support high-throughput, low-latency storage and retrieval of BLOBs.

Early work describing multi-dimensional, array-style databases was first stipulated by Baumann[15]. Libkin et. al.[30] formulated a calculus for arrays and implemented a query language that specifically targets multi-dimensional arrays. Further work in the domain resulted in the creation of the Raster Data Manager (RasDaMan) system[16, 8]. RasDaMan has been in active development since 1998 and is the longest-running array database project.

Commercial projects in the 90s and 2000s that involved the storage of multi-dimensional data was driven by the lucrative Geographic Information Systems (GIS) market. Oracle GeoRaster[7], TerraLib[18] and PostGIS[34] are examples. More recently, databases that support the nested data model as part of the Data Definition Language (DDL), such as MongoDB are being increasingly used in web-based applications that interact with map data provided by public map APIs such as Google Maps and OpenStreetMap.

SciDB[41] is a computational array DBMS designed to cater to the data storage needs of the scientific community. SciDB is designed to be multi-dimensional, with an array-based data model. It also provides support for data versioning and provenance which is a oft-cited feature requirement of data management systems from scientists. SciDB differs from the previous related work in its focus on computation - SciDB intends to push computation to the data by supporting queries that can perform some type of computation on the stored arrays, similar to RasDaMan. However, RasDaMan uses Postgres in the back-end for blob storage, while SciDB is built from ground-up to support efficient storage and retrieval of array data in a scale-out environment. Although the newest of all the work described in this section, SciDB has already been used in various scientific domains, including geo-spatial data, genomics[40] as well as astronomy[21].

To my knowledge, no known framework or system has explored the use of array databases for CBIR, however some work exists that is similar in flavor. Planthaber et. al. [36] describe the use of SciDB to process and store NASA's Moderate Resolution Imaging Spectroradiometer (MODIS) data on a SciDB cluster. Using SciDB, they have demonstrated the ability to write fairly simple queries that generate a composite RGB image from the MODIS data by selecting the appropriate spectral bands (one for each color channel), within specific geographic coordinates; a second workload they demonstrate is the use of the red and near-infrared spectral bands to compute the presence of vegetation (known as the Normalized Difference Vegetation Index (NDVI)). In [32], Moyers e.t al. describe the AscotDB, a web-based tool that enables the collaborative analysis of telescope images and metadata. AscotDB uses SciDB for array processing; iterative-processing features were added to SciDB to support this specific use-case.

4 Implementation

4.1 SciDB Architecture and Array Model

4.1.1 SciDB Architecture

SciDB uses a shared-nothing architecture which is shown in Figure 1 below.

SciDB is deployed on a cluster of servers, each with processing, memory, and local storage, interconnected using a standard Ethernet and TCP/IP network. Each physical server hosts a SciDB instance that is responsible for local storage and processing. A PostgreSQL database is used to store the SciDB catalog of array schema and the distribution of data in the cluster.

External applications, when they connect to a SciDB database, connect to one of the instances in the cluster. While all instances in the SciDB cluster participate in query execution and data storage, one server is the coordinator and orchestrates query execution and result fetching. It is the responsibility of the coordinator instance to mediate all communication between the SciDB external client and the entire SciDB database. The rest of the system instances are referred to as worker instances and work on behalf of the coordinator for query processing.

For testing and evaluation purposes, SciDB also works on a single physical machine in a "pseudo-distributed fashion", with multiple SciDB engines spawned on a single physical machine.

4.2 The SciDB Data Model - Arrays

SciDB arrays are expressed in terms of two basic parameters: the *dimensions* of the array, as well as *attributes* of the array.

4.2.1 Array Dimensions

An n-dimensional SciDB array has dimensions (d_1, d_2,...,d_n). The *size* of each dimension is the number of ordered values in that dimension. For example, a 2-dimensional array may have dimensions i and j, each with values $(1,2,3,...,10)$ and $(1,2,...,30)$ respectively. SciDB uses 64-bit integers to represent the values in each dimension, but also support non-integer dimensions such as variable-length strings or floating point integers. Furthermore, SciDB supports the notion of dimensional bounds. When the total value of a dimension is known in advance, the array can be declared with a *bounded* dimension. Sometimes, the cardinality of the array may not be known at array creation time. In such cases, the array can be declared with an *unbounded* dimension. It must be noted, however, that certain array operations are restricted to arrays with integer bounded dimensions.

In addition to the dimension size attributes, the user must define two parameters for each dimension: the *chunk size* and *chunk overlap* parameters. These parameters affect the distribution of the array data among the worker nodes and need to be studied with respect to their effect on the performance of the image operations that we intend to deploy in SciDB.

4.2.2 Array Attributes

An n dimensional array in SciDB refers to a single cell or element of an array. However, each array in the SciDB array can hold multiple data values known as *attributes*. Each data value is referred to as an *attribute*, and can be any of the supported data types in SciDB.

Therefore, during the creation of an array in SciDB (analogous to the declaration of a table schema in a relational database), the user must specify:

- An array name - a simple string which can be used to refer to the array in all operations involving the array.
- The dimensions of the array. The name and size of each dimension must be declared, with the exception of unbounded dimensions, whose size is represented using the asterisk character (`*`).
- At least one attribute for the array. Attributes can be added to an array as the result of an operation in SciDB.

An example of an array definition in SciDB is given below:

Listing 1: Creating an Array in SciDB

```
AQL% CREATE ARRAY open <val:double>[I
    =0:9,10,0,J=0:*,10,0];
```

In the example, the name of the array is `open`, with one attribute named `val`, of type `double`. This array consists of two dimensions, `I` and `J`. `I` is a bounded dimension with values ranging from 0 to 9, with chunk size 10 and chunk overlap 0. `J` another dimension similar to `I`, but the size of this dimension is unbounded.

4.3 Query Languages and Array Operations

SciDB currently supports two distinct query languages: the *Array Query Language* (AQL) the and *Array Functional Language* (AFL). AQL is very similar to SQL, where operations are described as `SELECTS, INSERT, UPDATE` with support for various types of `JOINS`. On the other hand, AFL allows for a more functional description of array operations, using a function syntax which

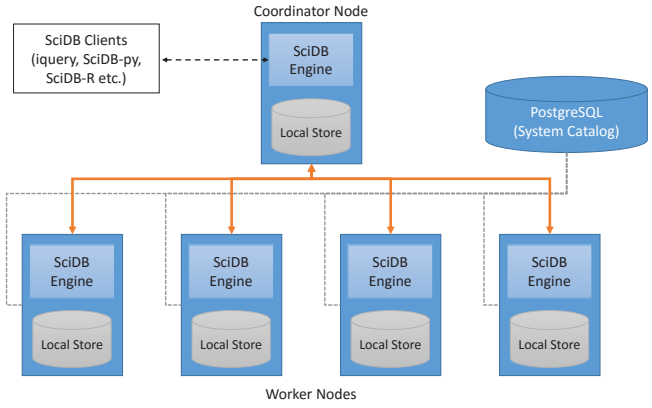


Figure 1: SciDB Cluster Architecture

allows for nesting of operations. AFL has a few advantages over AQL in terms of the verbosity of queries, however, from my initial work with SciDB I believe that the query languages do not intersect perfectly in terms of the operations supported; some operations may only be performed in AQL (such as joins along a dimension), while some others can only be performed in AFL.

SciDB is designed to perform a number of operations on Arrays. An extensive listing of array operations is beyond the scope of this document, the interested reader is referred to the SciDB manual[9]. However, the operations that can be performed on an array can be broadly classified into the following: array selection, array operations (such as cross product, joins etc.), aggregation operations (which can return either arrays or scalars) and so on.

4.4 Representing Images in SciDB

Images can be represented in multiple ways, and this has an effect on the type and performance of image array operations that are performed in SciDB.

4.4.1 Loading Image Data and Conversion to SciDB Arrays

SciDB allows multiple client types to interact with the database (see §4.1). For this project, I chose the SciDB-py client interface to python[10]. This interface allows for SciDB arrays to be accessed and manipulated using python variable bindings. In addition, it supports the conversion of SciDB arrays to and from numpy arrays. This is useful since the scikit-image library[42] in python allows for the reading of various image file formats and loading of the pixel data into numpy arrays.

4.4.2 Representing a single image in SciDB

An image can be considered to be an array of intensity values for each pixel in an image. A single image can be represented in SciDB in a number of ways:

- A 3-dimensional array consisting of a single attribute: intensity. The dimensions are image width, height and color channel (such as red, green and blue (RGB)) The array schema for this representation (assuming an image has 480x320 pixels) is represented in listing 2

Listing 2: A 3-dimensional Image Array in SciDB

```
<f0:uint8>
[width=0:479,1000,0,
height= 0:319,1000,0,
channel=0:2,1000,0]
```

- A 2-dimensional array with 3 attributes, one attribute per channel.

4.4.3 Representing Multiple Images in SciDB

It follows from the previous discussion that there are again many ways in which multiple images can be represented in SciDB. This design decision influences the way in which queries are constructed for the image processing operations. We have identified two specific ways in which multiple images can be represented in SciDB (Figure 2):

Individual Arrays: Each image can be represented as an individual array in SciDB using the methods described in §4.4.2. One drawback of this method is that the operations that involve multiple images require array joins, as will be described in §4.5.

Volume of Images: Multiple images can be encapsulated as a single array of multiple images with an extra dimension that represents an index to an image (see Listing

3). In this approach, multiple images can be "flattened" into a single image using aggregation for the image operations that we explore in §4.5. This approach is great for images that have the same dimension (such as individual frames for a video). SciDB also has support for sparse arrays and unbounded dimensions and it remains to be seen if the approaches that discuss in §4.5 are applicable for spare and/or arrays with unbounded dimensions.

Listing 3: A 4-dimensional image volume in SciDB

```
<f0:uint8>
[width=0:479,1000,0,
height= 0:319,1000,0,
channel=0:2,1000,0,
image=0:*,1,0]
```

(a) Images stored as individual arrays in SciDB

(b) Images stored as a volume in a single array in SciDB

Figure 2: Ways to represent Images in SciDB

4.5 Image Processing Benchmarks

We plan to develop queries sec:WIAorm the following fundamental image primitives in SciDB. For each operation that we discuss in this section, we have to develop the equivalent SciDB expressions to perform the image operation and evaluate their performance. The image processing benchmarks that have been constructed are:

- Weighted Image Average (**WIA**)
- Image Patch Extraction and Average (**IPE**)
- Convolution (**CONV**)
- All Pairs Nearest Neighbours (**APNN**)

4.5.1 Weighted Image Average

In this operation, a collection of related images are combined to produce a single image using some kind of average. The specific variant we look at in this study is the weighted average of a volume of images.

As an example, the queries required to average N individual arrays is presented in program listing 4. This operation requires a cross-join between each individual array and the target array in an iterative fashion, followed by a sum of the individual elements. Each element of the resulting array is then divided (scalar division) to produce the final average image.

Listing 4: Averaging N individual image arrays in SciDB

```
store(project(apply(join(sum,image), y, sum.
    x + image.x), y), sum)
store(project(apply(sum, z, sum.x / n ), z),
     sum)
```

Listing 5: Weighted Average of images in a single SciDB Volume

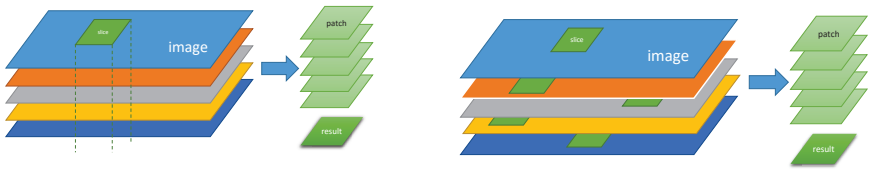
```
SELECT * INTO temp_average FROM average_img
    JOIN weights ON average_img.i3 = weights
    .i0;
store(aggregate(apply(temp_average,
    weighted_image,f0*weight),sum(
    weighted_image),i0,i1,i2),average_image)
    ;
store(project(apply(average_image,
    weighted_image_avg, weighted_image_sum /
     $WEIGHTTOTAL ),weighted_image_avg),
    average_image2)
```

The weighted image averaging operation (as presented in listing 5) when applied to a volume of images can be expressed as three separate operations: A dimension join to associate each individual image in the volume with the associated weight of the image, an aggregation operation where each element of each image is multiplied with its associated weight and then flattened, and finally a scalar division operation to divide each element of the resulting image with the sum of weights to produce the final, weighted average image.

4.5.2 Image Patch Extraction and Average

The IPE benchmark consists extracting a square patch (also known as tile) portion of the image volume from each image to construct a new image volume of patches. This volume is then averaged using the weighted image average query described in §4.5.1. This operation can be done in two distinct ways:

- Extract patches from each image using a fixed coordinate for the patch location, as illustrated in Figure 3(a).
- Extract the patches from each image at variable locations, as illustrated in Figure 3(b). This is a more complex query, which requires iterating over the image array for each image and extracting the required subregion of the image and adding it to the patch volume.



(a) Extracting image patches from a fixed region on each image

(b) Extracting image patches from a variable region on each image

Figure 3: Patch extraction implementations for our study

Listing 6: Fixed Image Patch Extraction

```
aggregate(apply(cross_join(subarray(
    image_volume,$PATCH_X_START,
    $PATCH_Y_START,0,0,$PATCH_X_END,
    $PATCH_Y_END,2,$NUM_FILES) as
    patch_volume, weights,patch_volume.i3,
    weights.i0),weighted_image,f0*weight),
    sum(weighted_image),i0,i1,i2);
```

4.5.3 Fixed Patch Location Implementation

Extraction of patches from a fixed region in each image is fairly straight-forward in SciDB. The `subarray()` function allows for a fixed region of an array to be extracted from the volume of images. Once the patches have been extracted and stored as a single volume of patches, the average patch image can be computed using the same algorithm described in §4.5.1. The exact query used for this operation is provided in Listing 6.

4.5.4 Variable Patch Location Implementation

To extract patches from different coordinates for each image in a volume of images, a single DB query can not longer be used. Instead, we have to iterate over each image in the volume, and slice out the image patch. The patch is appended into the the volume of patches. Once the patches have been extracted and stored as a single volume of patches, the average patch image can be computed using the algorithm described in §4.5.1.

4.5.5 Convolution

The convolution operation on an image involves the calculation of each pixel value as being some kind of weighted aggregate of itself and its neighbouring pixel values. The weights are specified as a convolution *kernel*. Convolution is illustrated in Figure 4. Different convolution kernels result in different output images being generated, aggregations generally result in image smoothing, while special kernels with negative weights at certain neighbouring pixels can be used to highlight the edges in an image.

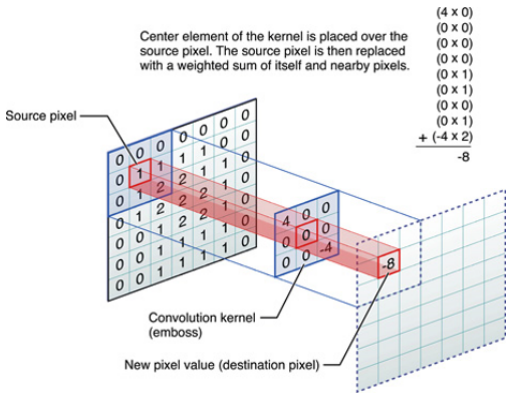


Figure 4: Convolution Operation on an Image using a convolution kernel[3]

While SciDB does not explicitly support convolution with arbitrary kernels, it support windowed aggregation using the `window()` function. This allows the values within an N-dimensional neighbourhood of an array element to be aggregated using some aggregation function, such as `sum()` or even a user-defined aggregation function. Image convolution using specific kernels, such as the box-blur kernel can be performed in SciDB by using a `3x3` window over each image plane and using the `sum()` aggregation function.

4.5.6 All-Pairs Nearest Neighbours

The APNN benchmark for a given image dataset involves the calculation of a distance matrix and computing the k Nearest Neighbours for every image represented in the distance matrix. Typically, this is computed using higher-order features, but for this benchmark, we will use the sum of squared distance between each pair of pixels as the distance function for two images.

[3]Image Source: `https://developer.apple.com/library/ios/documentation/Performance/Conceptual/vImage/Art/kernel_convolution.jpg`

In SciDB, the operations involved in the APNN benchmark are illustrated in figure 5. First, a cross-join of the image volume is performed with itself across the image dimension to produce all pairs of pixel values. Next, the sum of square distances is computed using the `apply()` and `aggregate()` functions to produce a distance matrix for all pairs of images. Finally, a special kNN function that computes the ranks of image similarities can be computed.

5 Evaluation Methodology and Experimental Setup

For each of the image operations discussed in §4.5, we produce different sequence of queries in SciDB that achieve the intended image processing operation. These can be compared to a baseline implementation in MPI.

5.1 Image Datasets

We plan to use a number of datasets that can be used to evaluate the performance of our SciDB image processing operations. All the operations described in §4.5 are low-level pixel operations that are not dependent on the actual content of the images, we can use any collection of images, while varying image resolution to assess the performance impact of small images versus larger images. For our experiments, we went with a common video resolution of 1920x1080. In the case of APNN, due to the sheer volume of data generated by the cross join as illustrated in Figure 12(d), we restricted the SciDB experiments to 64x64 thumbnails obtained from the MIRFLICKR thumbnail dataset[26]. A Large number of image frames of the same resolution, channels and bit depth can be easily generated from videos. For this experiment we generated the 1080p video frames from Big Buck Bunny [3], an open-source 3D animated short film, available under a creative commons license.

5.2 Baseline MPI Implementations

In order to perform a baseline comparison of SciDB's performance for these types of image processing tasks, we implemented the same algorithms using the Message Passing Interface (MPI) framework. In this model, we assumed a shared storage model for input and each MPI process can independently access each and every image file in the dataset. In general, we had two types MPI program flows for the baseline implementations:

Map, Gather, Reduce: This flow consists of a map-style phase where each processor independently processes image files that are assigned to it, produces an intermediate result that is then gathered on the master process and finally reduced to the final output. Represented in Figure 6(a), this flow was used to implement the baseline MPI version for the WIA, IPE and APNN benchmarks.

Map only: This flow consists of a map-style phase where each processor independently processes image files that are assigned to it, and produces an output file that is written independently to the shared disk. Represented in Figure 6(b), this flow was used to implement the baseline MPI version for the convolution (CONV) benchmark.

Category	Configuration
Instance Type	`m3.large`
Virtual CPUs	2 vCPUs from Intel Xeon E5-2670 (Ivy or Sandy Bridge)
RAM	7.5 GB
Root EBS Volume	25 GB
Instance Storage	32 GB SSD

5.3 Cluster Configuration

Part of our study requires us to explore the performance of these image processing applications on a virtual cluster deployed on AWS. For this, we have devised an cluster configuration that suits both the use of SciDB as well as MPI. Figure 7 illustrates our cluster configuration while Table 5.3 contains the configuration of the individual nodes. Each node has a single attached SSD-based instance store of 32 GB which is used by SciDB as the Database store and scratch space. In addition, a single EBS volume is exported using NFS to all the nodes in the cluster, allowing for the input dataset to be accessible to all of the nodes via NFS.

6 Results

In this section, we describe a few performance results obtained in implementing SciDB and comparing the performance of SciDB against a few baselines. First, we present a few results of tests performed during the implementation of these image processing operations on a laptop. We then scale up to a cluster on AWS with 16 nodes.

6.1 Single-Machine Experiments

All of the results presented in this section are from running SciDB on a laptop with Core i7 and 8GB of Memory. Even though SciDB was running on a single machine, it was configured to launch 4 instances of the database nodes.

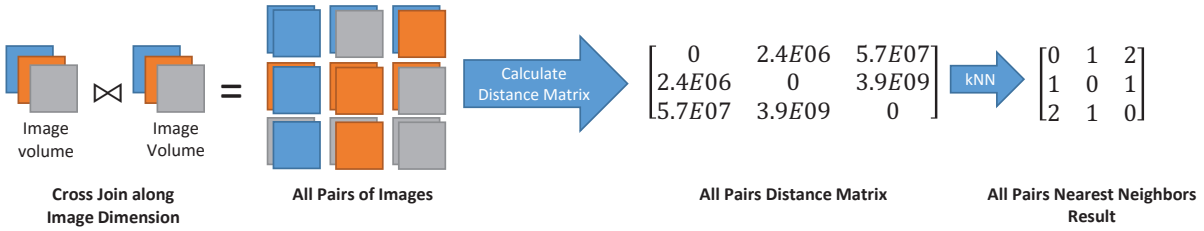
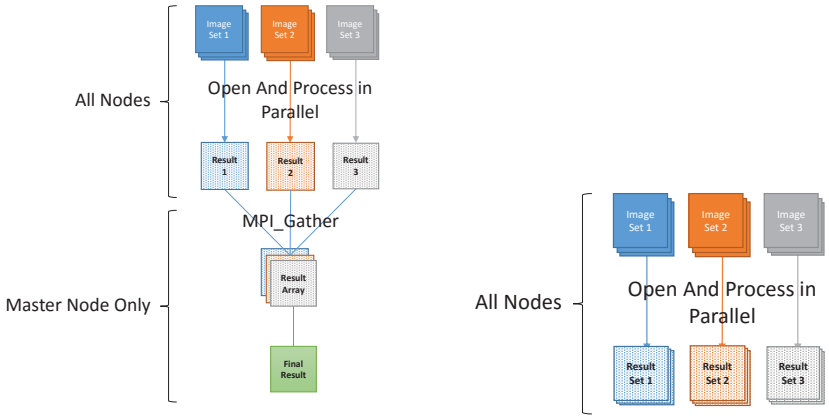


Figure 5: All Pairs Nearest Neighbors in SciDB



(a) Map, Gather & Reduce flow for the WIA, IPE and APNN benchmarks

(b) Map-only program flow for the Convolution (CONV) benchmark

Figure 6: Baseline MPI Implementations

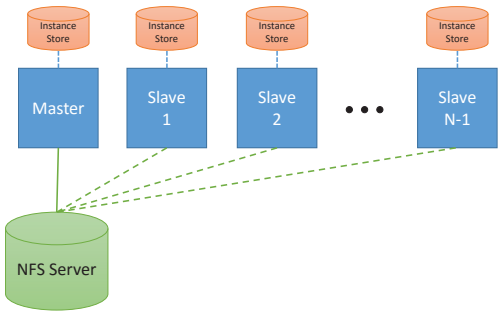


Figure 7: Cluster Layout

6.1.1 Weighted Image Average

The first experiment was to ascertain the performance of WIA using either images in individual arrays or an array of images as a single image volume. As discussed in the system design diagram, we can represent a set of images in individual arrays or as a volume of images as shown in Figure 2.

To evaluate the difference in the approaches, we show run an experiment to show the time taken to average N small (64x64) images, where $N = \{10, 100, 1000\}$. The results are presented in Figure 8.

Performing the image averaging operation across N individual arrays requires a cross-join between each individual array and the result array in an iterative fashion, followed by a sum of the individual elements. Each element of the resulting array is then divided by N (scalar division) to produce the final average image. The performance of this approach is in Figure 8, represented as the series `Averaging N Arrays`.

A marked improvement is noticed when the N images are combined into a single array, effectively containing a volume of images. The performance of this approach is in Figure 8, represented as the series `Weighted Volume Average`. In this case, Image averaging is expressed as three separate operations: A dimension join to associate each individual image in the volume with the respective weight of the image, an aggregation operation where each element of each image is multiplied with its

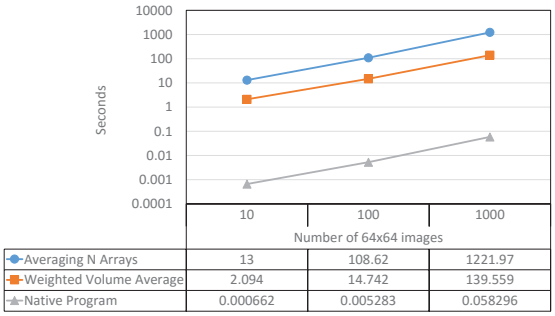


Figure 8: Single Machine Image Averaging Time

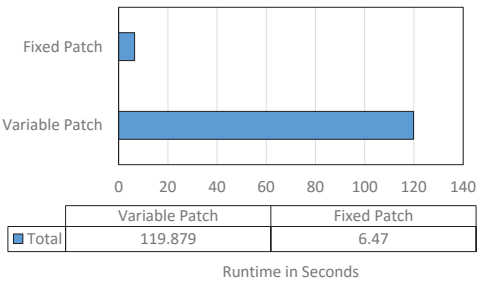


Figure 9: Time taken to perform patch extraction and averaging for fixed versus variable patch locations on a single machine

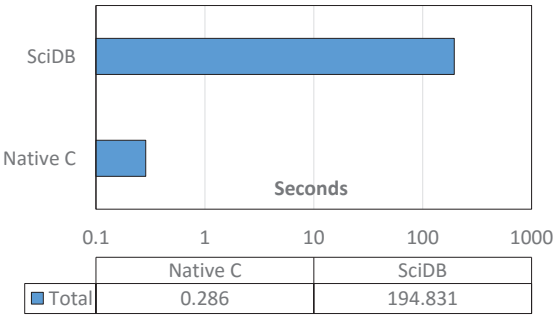


Figure 10: Performance of `window()` operation in SciDB versus an Image convolution operation in Native C code.

associated weight and then flattened, and finally a scalar division operation to divide each element of the resulting image with the sum of weights to produce the Final, weighted average image. Although this operation performs more work than the queries used to average N arrays, it performs an order of magnitude faster.

Notice however, that in spite of the improved performance of SciDB using the image volume approach, it is still many orders of magnitude slower than a simple in-memory calculation of the average image using a simple C program that sequentially iterates over each pixel, adding up the pixel values for each image, represented as the series `Native Program` in Figure 8.

Given that storing *N* images as a single "image volume" array is significantly faster for this simple task than storing them as separate arrays, we will continue using this volume of images approach as the image representation in SciDB for the remaining experiments.

6.1.2 Image Patch Extraction

For the IPE benchmark, we wanted to see the difference in runtime for the two approaches for IPE explained in §4.5.2. The two different techniques are compared in Figure 9.

We show the time taken to extract 100x100 patches from 7 images sized 1920x1080 in SciDB. Surprisingly, even though fixed patch extraction can be done as a single nested SciDB query, iterating and slicing out individual image planes from the image volume and then assembling the patch volume is significantly faster on a single node. We will revisit this experiment on the cluster in §6.2.2.

6.1.3 Convolution

We compare the performance of SciDB's windowed aggregation function when compared to a hand-written image convolution function in C (Figure 10). SciDB is once again over an order of magnitude slower than Native C code implementations of Image processing, for which the exact reasons not fully understood. We will continue our experiments on a much larger set of machines in S 6.2.2

6.2 Cluster-Based Experiments

In this section, we present the results of running our image processing operations on a much larger cluster of 16 nodes on AWS, using the cluster setup explained in §5. We begin with a comparison of each of the benchmarks against the MPI baselines, and then proceed to show a breakdown of the runtime in §6.2.3.

6.2.1 Image Load Times

A non-trivial portion of time was devoted to the actual loading and conversion of the compressed JPG data into the raw pixel values loaded into the image volume. Figure 11 shows the time taken to load 10,100 and 1000 images respectively on the 16 node cluster. It should

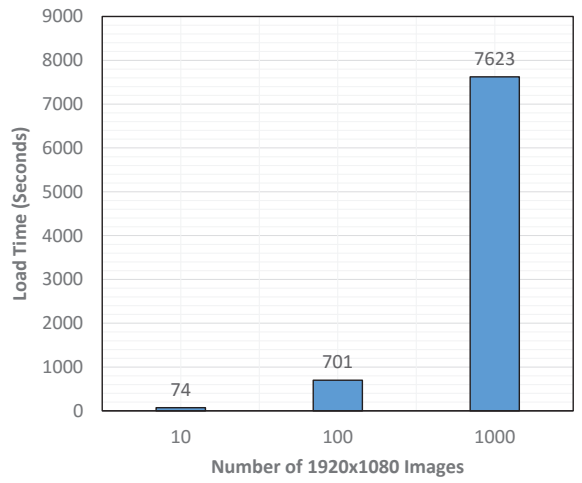


Figure 11: Time taken to convert and load the image dataset from files on the shared NFS mount into raw pixel values in a 16-node SciDB Cluster

be noted that the 10, 100 and 1000 images are roughly 0.474, 4.74 and 47.4 GB respectively.

6.2.2 Comparison with MPI

In this section, we compare the runtime of the image processing operations in SciDB against their MPI counterparts.

Weighted Image Average: In Figure 12(a), we see the runtime of SciDB for the WIA benchmark for 10, 100 and 1000 images on the 16 node cluster. The MPI version of this operation is roughly between $21\times$ to $91\times$ faster than the SciDB counterpart for the same operation.

Image Patch Extraction: Contrary to our experiences with the IPE benchmark in a single node, Figure 13, shows a slowdown in the performance of IPE when using a variable patch rather than a fixed patch from a volume of images.

In Figure 12(b), we see the runtime of SciDB for the IPE benchmark (fixed patches) when varying the dataset size. Unlike the WIA benchmark, where the entire image volume was flattened to a single image, the IPE benchmark shows significant improvement in runtime and is within an order of magnitude in terms of performance when compared to the baseline MPI version, as the 100×100 image patch extraction and averaging of a 1000 1080p images is only $\sim 2.14\times$ slower than its corresponding MPI version.

Convolution In Figure 12(c), we compare the performance of the MPI baseline that performs convolution on a set of images against the SciDB `window()` function, which performs the equivalent operation. Again, SciDB is an order of magnitude slower than the MPI implementation across the board.

All Pairs Nearest Neighbors: For the APNN benchmark, we perform the APNN step with an additional set of images: 10, 100 and 1000 64x64 thumbnails (Figure 12(d). The missing data-points indicate SciDB failing to finish execution even after 6 hours of runtime.

6.2.3 Timing Breakdown and Analysis

We now provide the timing breakdown for two of the operations of interest, the WIA and APNN benchmarks in Figure 14. We can see from the figures that the runtime for these queries is dominated by the cross join query, more so for APNN.

7 Conclusions and Future Work

In this work, we have implemented a number of image processing operations in SciDB and compared its performance against a simple MPI implementation. SciDB is many orders of magnitude slower than the corresponding MPI implementation and we conjecture the following reasons that contribute towards the slow performance of SciDB:

- SciDB stores raw pixel values and is unable to take advantage of domain-specific image compression schemes such as JPEG. This leads to a rapid expansion of the number of images stored which, in turn leads to massive I/O traffic during loading and execution of operations in SciDB.
- Except for Convolution, all of the image processing operations studied require some kind of cross-join operation, with APNN requiring a cross-join across the image dimension, leading to massive increase in the amount of data to be materialized.

The idea of SciDB or other array databases to perform image processing is interesting as it allows for these operations to be expressed in the form of queries. However, given the lack of support for domain-specific compression formats and the extremely slow execution times, we believe there a lot of work to do before SciDB becomes a viable platform for image and video processing.

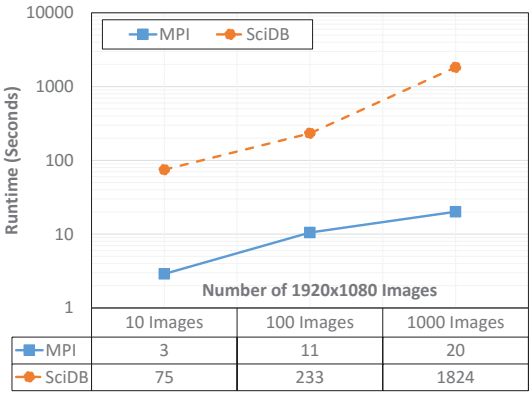
8 Code Availability

All of the code presented in this report is available here:

`http://github.com/suhailrehman/scidb-image`

The custom cluster launch scripts are also made available here:

`http://github.com/suhailrehman/scidb-starcluster`



	10 Images	100 Images	1000 Images
MPI	3	11	20
SciDB	75	233	1824

(a) WIA

	10 Images	100 Images	1000 Images
MPI	0.15	0.45	3.61
SciDB	6.47	6.63	7.75

(b) IPE

	10 Images	100 Images	1000 Images
MPI	0.28	2.11	40.37
SciDB	19.40	159.56	2227.43

(c) CONV

	64x64 Images			1920x1080 Images		
	10 Images	100 Images	1000 Images	10 Images	100 Images	1000 Images
SciDB	1.59	1042.18	0	1209.65	0	0
MPI	0.18	5.34	519.56	2.17	77.84	7726.75

(d) APNN

Figure 12: Performance of Image Operations in SciDB versus the Baseline MPI Versions. All run-times are the average of 3 runs, with variation being within $\tilde{2}\%$ of the actual runtime. APNN benchmarks for larger datasets did not finish running even after 6 hours.

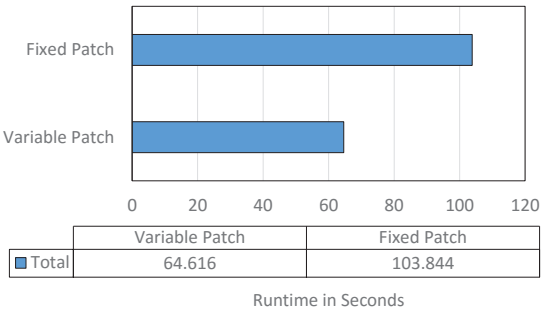


Figure 13: Fixed vs. Variable patch extraction on the 16 node cluster.

References

[1] Amazon S3. `http://aws.amazon.com/S3`. Accessed: 2015-10-09.

[2] Apache Storm. `https://storm.apache.org/`. Accessed: 2015-10-09.

[3] Big Buck Bunny. `https://peach.blender.org/`. Accessed: 2015-11-01.

[4] cuda-convnet2. `https://github.com/akrizhevsky/cuda-convnet2`. Accessed: 2015-10-09.

[5] Google Developers : Protocol Buffers. `https://code.google.com/p/protobuf/`. Accessed: 2015-09-25.

[6] LevelDB. `http://leveldb.org/`. Accessed: 2015-09-25.

[7] Oracle GeoRaster. `http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html`. Accessed: 2015-10-09.

[8] Rasdaman Array Database. `http://rasdaman.org/`. Accessed: 2015-10-09.

[9] SciDB Online Manual. `http://www.paradigm4.com/HTMLmanual/15.7/scidb_ug/`. Accessed: 2015-11-01.

[10] SciDB-Py Documentation. `http://scidb-py.readthedocs.org/en/latest/index.html`. Accessed: 2015-11-01.

[11] StormCV. `https://github.com/sensorstorm/StormCV`. Accessed: 2015-10-09.

[12] Twitter Blobstore. `https://blog.twitter.com/2012/blobstore-twitter%E2%80%99s-house-photo-storage-system`. Accessed: 2015-10-09.

[13] SQL Part 1: Framework (SQL/Framework). ISO 9075-1:2011, 2011.

[14] AGRAWAL, H., MATHIALAGAN, C. S., GOYAL, Y., CHAVALI, N., BANIK, P., MOHAPATRA, A., OSMAN, A., AND BATRA, D. Cloudcv: Large scale distributed computer vision as a cloud service. *CoRR abs/1506.04130* (2015).

[15] BAUMANN, P. Management of multidimensional discrete data. *The VLDB Journal 3*, 4 (1994), 401–444.

[16] BAUMANN, P., DEHMEL, A., FURTADO, P., RITSCH, R., AND WIDMANN, N. The multidimensional database system rasdaman. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 1998), SIGMOD '98, ACM, pp. 575–577.

[17] BEAVER, D., KUMAR, S., LI, H. C., SOBEL, J., VAJGEL, P., ET AL. Finding a needle in haystack: Facebook's photo storage. In *OSDI* (2010), vol. 10, pp. 1–8.

[18] CÂMARA, G., VINHAS, L., FERREIRA, K. R., DE QUEIROZ, G. R., DE SOUZA, R. C. M., MONTEIRO, A. M. V., DE CARVALHO, M. T., CASANOVA, M. A., AND DE FREITAS, U. M. Terralib: An open source gis library for large-scale environmental and socio-economic applications. In *Open source approaches in spatial data handling*. Springer, 2008, pp. 247–270.

[19] CHILIMBI, T., SUZUE, Y., APACIBLE, J., AND KALYANARAMAN, K. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (2014), pp. 571–582.

[20] CHU, H. Research in image indexing and retrieval as reflected in the literature. *Journal of the American Society for Information Science and Technology 52*, 12 (2001), 1011–1018.

[21] CUDRE-MAUROUX, P., KIMURA, H., LIM, K.-T., ROGERS, J., SIMAKOV, R., SOROUSH, E., VELIKHOV, P., WANG, D. L., BALAZINSKA, M., BECLA, J., DEWITT, D., HEATH, B., MAIER, D., MADDEN, S., PATEL, J., STONEBRAKER, M., AND ZDONIK, S. A demonstration of scidb: A science-oriented dbms. *Proc. VLDB Endow. 2*, 2 (Aug. 2009), 1534–1537.

[22] DATTA, R., JOSHI, D., LI, J., AND WANG, J. Z. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv. 40*, 2 (May 2008), 5:1–5:60.

[23] DONAHUE, J., JIA, Y., VINYALS, O., HOFFMAN, J., ZHANG, N., TZENG, E., AND DARRELL, T. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)* (2014), T. Jebara and E. P. Xing, Eds., JMLR Workshop and Conference Proceedings, pp. 647–655.

[24] GU, C., AND GAO, Y. A content-based image retrieval system based on hadoop and lucene. In *Cloud and Green Computing (CGC), 2012 Second International Conference on* (2012), IEEE, pp. 684–687.

[25] HOLOVATY, A., AND KAPLAN-MOSS, J. *The definitive guide to Django: Web development done right*. Apress, 2009.

[26] HUISKES, M. J., AND LEW, M. S. The mir flickr retrieval evaluation. In *MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval* (New York, NY, USA, 2008), ACM.

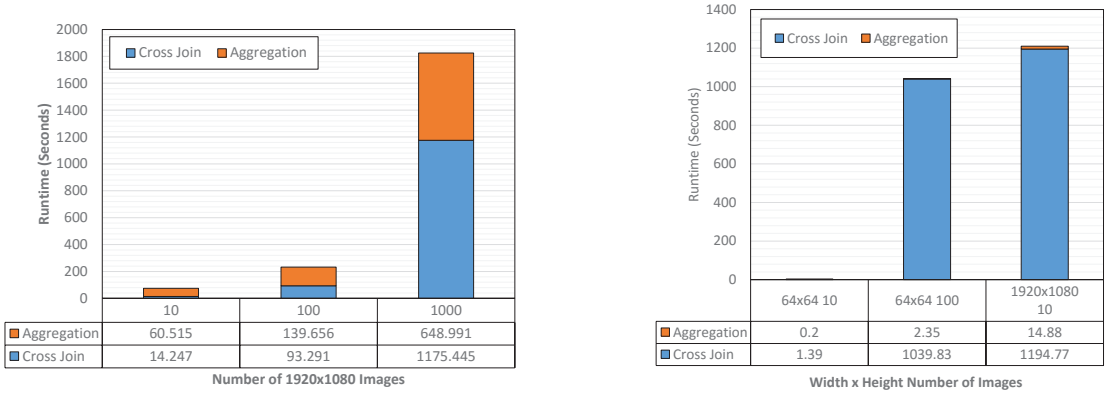
[27] JAI-ANDALOUSSI, S., ELABDOULI, A., CHAFFAI, A., MADRANE, N., AND SEKKAKI, A. Medical content based image retrieval by using the hadoop framework. In *Telecommunications (ICT), 2013 20th International Conference on* (May 2013), pp. 1–5.

[28] JIA, Y., SHELHAMER, E., DONAHUE, J., KARAYEV, S., LONG, J., GIRSHICK, R., GUADARRAMA, S., AND DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia* (New York, NY, USA, 2014), MM '14, ACM, pp. 675–678.

[29] LI, M., ANDERSEN, D. G., PARK, J. W., SMOLA, A. J., AHMED, A., JOSIFOVSKI, V., LONG, J., SHEKITA, E. J., AND SU, B.-Y. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2014), OSDI'14, USENIX Association, pp. 583–598.

[30] LIBKIN, L., MACHLIN, R., AND WONG, L. A query language for multidimensional arrays: Design, implementation, and optimization techniques. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 1996), SIGMOD '96, ACM, pp. 228–239.

[31] LOW, Y., BICKSON, D., GONZALEZ, J., GUESTRIN, C., KYROLA, A., AND HELLERSTEIN, J. M. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment 5*, 8 (2012), 716–727.



(a) WIA

(b) APNN

Figure 14: Performance of Image Operations in SciDB versus the Baseline MPI Versions. All run-times are the average of 3 runs, with variation being within $\tilde{2}\%$ of the actual runtime. APNN benchmarks for larger datasets did not finish running even after 6 hours.

[32] MOYERS, M., SOROUSH, E., WALLACE, S. C., KRUGHOFF, S., VANDERPLAS, J., BALAZINSKA, M., AND CONNOLLY, A. A demonstration of iterative parallel array processing in support of telescope image analysis. *Proc. VLDB Endow. 6*, 12 (Aug. 2013), 1322–1325.

[33] MURALIDHAR, S., LLOYD, W., ROY, S., HILL, C., LIN, E., LIU, W., PAN, S., SHANKAR, S., SIVAKUMAR, V., TANG, L., AND KUMAR, S. f4: Facebook's warm blob storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, Oct. 2014), USENIX Association, pp. 383–398.

[34] OBE, R., AND HSU, L. *PostGIS in action*. Manning Publications Co., 2011.

[35] OWEN, S., ANIL, R., DUNNING, T., AND FRIEDMAN, E. *Mahout in action*. Manning Shelter Island, 2011.

[36] PLANTHABER, G., STONEBRAKER, M., AND FREW, J. Earthdb: scalable analysis of modis data using scidb. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data* (2012), ACM, pp. 11–19.

[37] PREMCHAISWADI, W., TUNGKATSATHAN, A., INTARASEMA, S., AND PREMCHAISWADI, N. Improving performance of content-based image retrieval schemes using hadoop mapreduce. In *High Performance Computing and Simulation (HPCS), 2013 International Conference on* (2013), IEEE, pp. 615–620.

[38] ROBLES, O., BOSQUE, J., PASTOR, L., AND RODRIGUEZ, A. Performance analysis of a cbir system on shared-memory systems and heterogeneous clusters. In *Computer Architecture for Machine Perception, 2005. CAMP 2005. Proceedings. Seventh International Workshop on* (July 2005), pp. 309–314.

[39] SERMANET, P., EIGEN, D., ZHANG, X., MATHIEU, M., FERGUS, R., AND LECUN, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR abs/1312.6229* (2013).

[40] STONEBRAKER, M., BROWN, P., POLIAKOV, A., AND RAMAN, S. The architecture of SciDB. In *Scientific and Statistical Database Management* (2011), Springer, pp. 1–16.

[41] STONEBRAKER, M., BROWN, P., ZHANG, D., AND BECLA, J. SciDB: A database management system for applications with complex analytics. *Computing in Science & Engineering 15*, 3 (2013), 54–62.

[42] VAN DER WALT, S., SCHÖNBERGER, J. L., NUNEZ-IGLESIAS, J., BOULOGNE, F., WARNER, J. D., YAGER, N., GOUILLART, E., AND YU, T. scikit-image: image processing in python. *PeerJ 2* (2014), e453.

[43] WAN, J., WANG, D., HOI, S. C. H., WU, P., ZHU, J., ZHANG, Y., AND LI, J. Deep learning for content-based image retrieval: A comprehensive study. In *Proceedings of the ACM International Conference on Multimedia* (New York, NY, USA, 2014), MM '14, ACM, pp. 157–166.