Processing collections of images and videos on big-data frameworks

Alex Poms, Ravi Teja Mullapudi,
Suhail Rehman

Abstract

1 Introduction

Images and video already dominate internet traffic and occupy a significant amount of datacenter storage. The Cisco Visual Networking Index predicts that IP video traffic will consume over 80% of total internet bandwidth in 2019. In 2009, Facebook reported 1.5PB of image storage with 25TB added weekly and 550,000 images served per second at peak. This visual data is mostly just served directly to a human user for viewing. However, recent advances in computer vision, such as feasible image/object understanding and reconstruction of 3D structure from image collections, motivate large-scale analysis of images and videos for deriving insights and information that is not accessible from other sources. For example, a relief organization could automatically reconstruct the damage done by a natural disaster by having vehicles with mounted cameras observe the affected area or a state could perform automatic demographics estimation by analyzing the public Facebook profile photos of their residents.

Currently, visual computing applications are few and far between due to the inherent complexities of working with large visual datasets. The most apparent of these difficulties is that the massive size of the datasets vastly exceed the storage and computational capacity of a single machine. Thus any application must be distributed and parallelized across a large cluster, managing data transfer and communication between nodes. Unlike traditional analytics workloads, visual computing applications also require highly tuned and efficient image processing kernels that take advantage of the heterogeneous hardware increasingly seen in cloud infrastructure, such as throughput-oriented accelerators including GPUs and even FPGAs. Having a predefined set of optimized implementations is not enough because image processing pipelines require global code restructuring that crosses function boundaries; each application may require custom image processing code that is composed with other functionality. Lastly, visual computing applications exhibit tight coupling between traditional analytics operations, such as those of Spark [Zaharia et al. 2012] and MapReduce [Dean and Ghemawat 2008], and non-streaming operations, like distributed optimization or graph processing.

It is unclear if existing systems can simultaneously satisfy these three requirements. Systems optimized for analytics operations, like Spark, tend to sacrifice explicit access to their data collections that prohibit efficient integration of non-streaming operations. Analytics systems also lack the ability to specialize user routines to heterogeneous hardware resources. On the other hand, systems that perform non-streaming operations, like the Parameter Server [Li et al. 2014], generally lack the flexibility in their programming model to allow direct incorporation of streaming operations like map, filter, or reduce.

2 Background

2.1 Spark

2.2 Legion

2.3 SciDB

SciDB[Stonebraker et al. 2013] 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 arrays are expressed in terms of two basic parameters: the *dimensions* of the array, as well as *attributes* of the array.

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.

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.

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[Sci]. 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.

3 Workloads

Visual data applications contain a diverse set of operations, but a few of them stand out as particularly important and distinctly core components of the analysis algorithms they are a part of. In particular, two classes of operations that seem to span a significant amount of the space of important operations in visual data applications are: very compute-intensive kernels applied individually across an entire dataset, and aggregate, high-communication operations, like computing nearest neighbors or clustering. For a distributed computing framework to serve as a platform for creating performant applications that process large collections of images and videos, it must at least support very efficient and scalable implementations of the following operations:

3.1 High-performance map kernel

3.2 Clustering

4 Distributed collections on Legion

4.1

4.2 Task scheduling

4.3 KMeans Implementation

5 Evaluation

5.1 Spark vs Legion

5.1.1 Partition startup time

5.2 SciDB vs MPI

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

Weighted Image Average: In Figure 1(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: In Figure 1(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 1(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 Neighbours: For the APNN benchmark, we perform the APNN step with an additional set of images: 10, 100 and 1000 64x64 thumbnails (Figure 1(d). The missing data-points indicate SciDB failing to finish execution even after 6 hours of runtime.

5.2.1 Timing Breakdown and Analysis

We now provide the timing breakdown for two of the operations of interest, the WIA and APNN benchmarks in Figure 2. We can see from the figures that the runtime for these queries is dominated by the cross join query, more so for APNN.

6 Discussion

6.1 Native code on Spark

6.2 Improving distributed collection abstractions

6.2.1 Batch size

6.2.2 Iterators

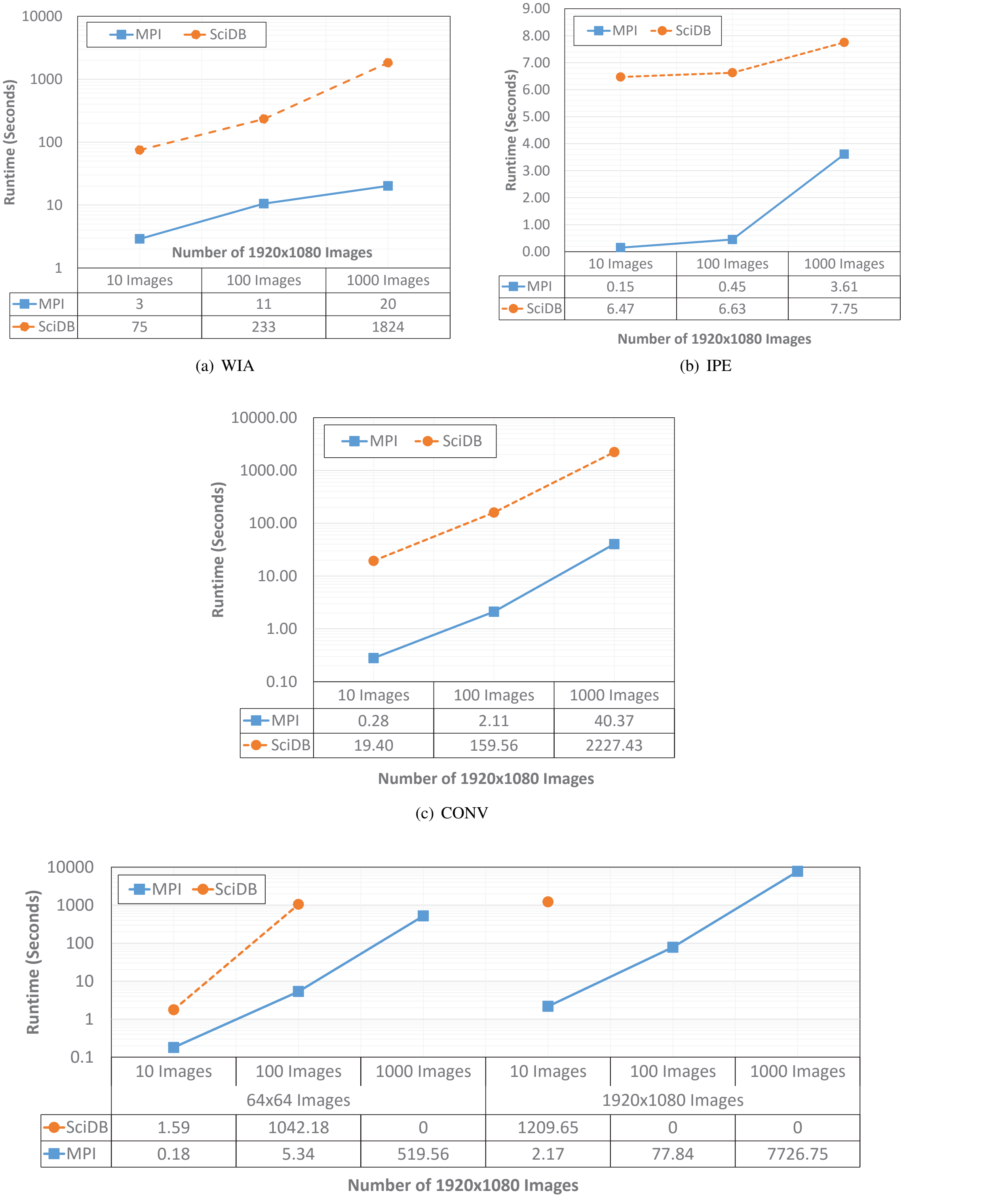
6.3 Improving SciDB

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.



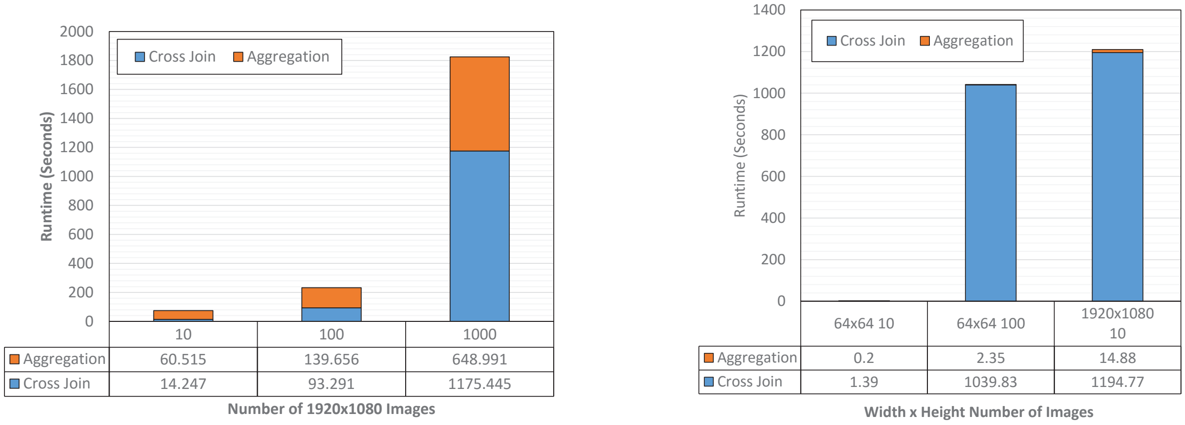
(a) WIA

(b) IPE

(c) CONV

(d) APNN

Figure 1: *Performance of Image Operations in SciDB versus the Baseline MPI Versions. All run-times are the average of 3 runs, with variation being within 2̃% of the actual runtime. APNN benchmarks for larger datasets did not finish running even after 6 hours.*



(a) WIA

(b) APNN

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

7 Conclusions

References

AGARWAL, S., FURUKAWA, Y., SNAVELY, N., SIMON, I., CURLESS, B., SEITZ, S. M., AND SZELISKI, R. 2011. Building rome in a day. *Communications of the ACM 54*, 10, 105–112.

BAUER, M., TREICHLER, S., SLAUGHTER, E., AND AIKEN, A. 2012. Legion: expressing locality and independence with logical regions. In *Proceedings of the international conference on high performance computing, networking, storage and analysis*, IEEE Computer Society Press, 66.

DEAN, J., AND GHEMAWAT, S. 2008. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1, 107–113.

HEINLY, J., SCHÖNBERGER, J. L., DUNN, E., AND FRAHM, J.-M. 2015. Reconstructing the world* in six days *(as captured by the yahoo 100 million image dataset). In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

LI, M., ANDERSEN, D. G., AND PARK, J. W. 2014. Scaling distributed machine learning with the parameter server. In *OSDI*.

RAGAN-KELLEY, J., BARNES, C., ADAMS, A., PARIS, S., DURAND, F., AND AMARASINGHE, S. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *ACM SIGPLAN Notices 48*, 6, 519–530.

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

STONEBRAKER, M., BROWN, P., POLIAKOV, A., AND RAMAN, S. 2011. The architecture of scidb. In *Scientific and Statistical Database Management*, Springer, 1–16.

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

ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULEY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, USENIX Association, 2–2.

ZHU, J.-Y., LEE, Y. J., AND EFROS, A. A. 2014. Averageexplorer: Interactive exploration and alignment of visual data collections. *ACM Transactions on Graphics (TOG) 33*, 4, 160.