Performance Improvement of Hybrid Content Based Image Retrieval

Devashish Purandare
Savitribai Phule Pune University
devashish@outlook.com

Prashant Kumar
Savitribai Phule Pune University
prashantkmr846@gmail.com

Amol Rokade
Savitribai Phule Pune University
rokadeamol25@gmail.com

I. abstract

In today's times, the number of images are growing exponentially. Billions of images are shared everyday, and there's a need to search and sort through these images efficiently. Traditional text based search falls short on many levels such as linguistic barriers, different interpretations and the added cost of annotation. Content based approach allows us to use visual content of the images to retrieve similar images. Research on Hybrid approach in Content Based Image Retrieval (CBIR) has so far shown promising results, providing higher accuracy as compared to a global approach but at the same time it comes with the high computational complexity of the local approach. Our implementation of the hybrid CBIR system has reduced complexity and better performance than the original approach. Various methods used include caching, parallelism, map-filter, approximation and reducing comparisons. Our results indicate that the performance can be improved by a huge margin without sacrificing accuracy.

Categories and Subject Descriptors

I.4 [IMAGE PROCESSING AND COMPUTER VISION] Scene Analysis, Object Recognition, Shape, Range Data, Applications

Keywords

Content based image retrieval; Wavelet transform; Shape feature; Information Retrieval.

II. Introduction

Content Based Image Retrieval (CBIR) system by Yen Do et al [1] uses both global features (Shape), and local features (Contours). However there's a major drawback of such systems that they are very complex computationally and performance is an issue. The study notes "In the future work, we will improve the method to shorten the run time when the database has high number of images." Our system builds upon this work by implementing these measures in parallel, and optimizing the execution time and complexity.

CBIR has wide applications [2] including but not limited to applications in Crime prevention, military, intellectual property, architectural and engineering design, fashion and interior design, journalism and advertising, medical diagnosis, geographical information systems (GIS) and remote sensing, cultural heritage, education and training, home entertainment, web searching. Due to advances in data storage, retrieval, and image acquisition technologies, huge datasets and archives are now available for analysis. With billions of new images, there's a need to efficiently store, process, and index images. While much effort has been spent on optimizing the processing, data storage and retrieval have largely been ignored. Storage, searching and retrieval form the main bottlenecks in a CBIR system as memory access is slow. Further storing images in NOSQL databases is difficult and SQL databases are not particularly suitable for similar image search.

We demonstrate a way to improve the storage, retrieval and search performance of the system, while keeping the accuracy intact.

III. Related Work

Kuldeep Yadav, et al worked using Nvidia CUDA programming to improve the performance of the system by making use of the inherent parallelism in the system[3]. In another approach using CUDA, Heidari uses color to retrieve images in addition to shape [4]. While these works speed up the processing part, the bottleneck remains at the storage, retrieval, and searching functions. Our work tries to improve this performance and can

work in conjunction with these systems.

IV. Methodology

We identified 8 areas to improve and optimize the performance of the system

1. Caching - MongoDB
2. Reducing Loops.
3. Parallel and multi-core processing.
4. Optimizing loops.
5. Reducing data size.
6. Junking lower and trivial values.
7. Reducing unnecessary computations.
8. Map - Filter to optimize search.

I. Caching – MongoDB

MongoDB is a document based database. MongoDB uses JSON for record storage and representation, and allows MAP-FILTER-REDUCE operations[5], which was the main motivation behind the choice.

The processed images are broken down into a descriptor, with the following properties :

1. path – string
2. global feature – matrix
3. local feature – matrix
4. number of ones – integer

This allows for easy storage and retrieval with a unique descriptor for each image. As image is not stored database, it reduces the size of the database and allows for quick parsing.

Recalculating these descriptors each time is redundant and causes major delays. Our system checks the database with available data and only updates the database if there is a change in data.

Caching these descriptors offers a huge performance boost, with processing time reduced to a fraction of the original. This scales linearly for images - figure 1.

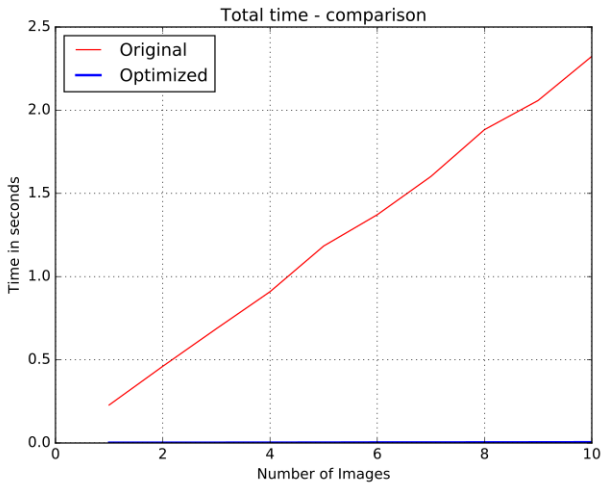

Figure 1: *Original implementation vs Caching (Optimized)*

The optimized implementation scales linearly – figure 2 even for a large number of images.

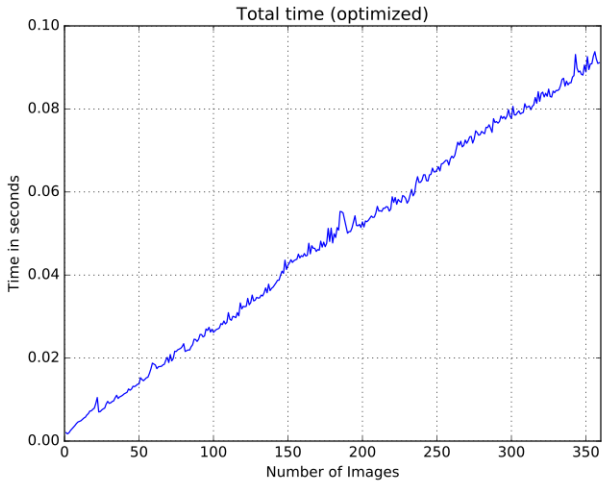

Figure 2: *Caching vs number of images*

II. Reducing loops, Parallel processing, Optimizing loops and reducing data size

Use of Python allows for drastically reducing the computation and making it parallel using Numpy. Numpy makes matrix and array manipulation fast and easy. Various routines available in `numpy.linalg`[6]. This reduces loops and optimizes the code for parallel processing. The next step is to cast every value as an

integer. While this results in a slight loss of precision, the difference is negligible with respect to the system. This allows subtraction in place of comparisons and use of logic primitives to boost the performance.[7] Various Numpy operations further improve performance in this scenario. [8]

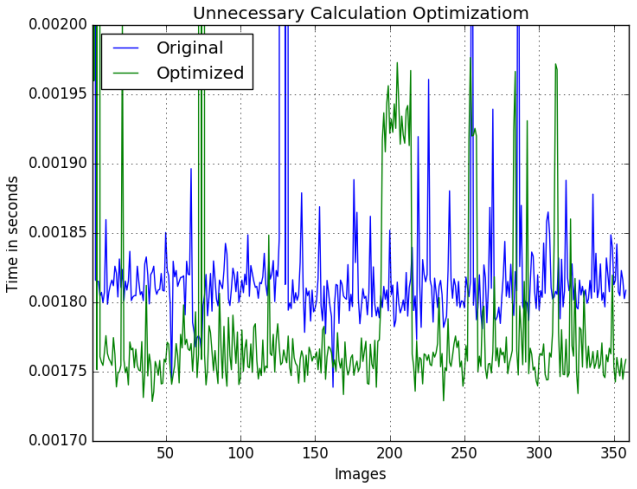

Figure 3: *Reducing loops, Parallel processing, Optimizing loops and reducing data size*

III. Junking lower values and reducing unnecessary computations

While we get a huge array for singular values after singular value decomposition (SVD)[9], not all values are equal. Lower values of singular value vectors don't affect the accuracy but higher values make a huge difference. This can be used to optimize the performance by using a suitable 'k' value to boost the performance of the system, where we use the top k values in descending orders and get rid of the rest.[10]

Another way to improve performance is to reduce unnecessary computations. SVD results in two orthogonal matrices U and V given by the formula $A = USV^T$ where A is the current matrix and S is the array of singular values. Here we can skip over the computation of matrices U and V. Scipy gives us an easy way to do this.[11]

IV. Map–Filter

For optimizing search operations, it is necessary to reduce the amount of data through which the system must search. The key challenge here is to reduce the search set without reducing the accuracy of the system. Map–Reduce paradigm [12] can be used to reduce the searching time to a fraction of the initial value. Our approach uses the number of ones in the global feature of the image as a criteria for Map–filter. With a threshold $\pm x$ the descriptors are mapped to a value. While searching, the number of ones $\pm x$ map is searched. This drastically reduces the search time, especially when the number of images is very large. MongoDB allows an easy way to perform this [13] and there's a significant reduction in search time 4.

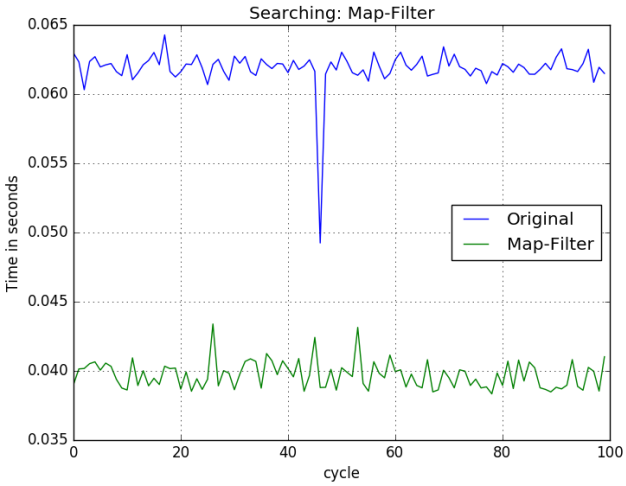

Figure 4: *Map-Filter-Reduce*

V. Conclusion

Our implementation improves the performance of hybrid CBIR by a huge margin, reducing searching to less than 1 second from more than 40 seconds. Processing speed initially is halved as file writes are eliminated and other optimizations speed up the processing. After initial creation of database, processing speed is limited just to the processing of newly added images to the database, giving a massive boost to the performance.

Figure 5: *Implementation*

VI. Future Work

While these results are encouraging, there are a lot of optimizations that can be applied further to improve the performance. The processing part of this system can be integrated with the GPU programming work done by Yadav et al.[3]. Currently this system uses a naive pre-processing algorithm and work needs to be done to improve that so that it can extract objects from real world scenario. The system can use sophisticated recognition[14] and tagging techniques such as Google vision to improve Map-reduce accuracy.

References

[1] Y. Do, S. H. Kim, I. S. Na, D. W. Kang, and J. H. Kim, "Image retrieval using wavelet transform and shape decomposition," in *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '13, (New York, NY, USA), pp. 92:1–92:8, ACM, 2013.

[2] M. G. John Eakins, "Content-based image retrieval." JISC Technology Applications Programme, Joint Information Systems Committee, University of Northumbria at Newcastle, October 1999.

[3] K. Yadav, A. Mittal, and M. Ansari, "Parallel implementation of similarity measures on gpu architecture using cuda," *Indian Journal of Computer Science and Engineering.*, pp. 1–9, 2012.

[4] H. Heidari, A. Chalechale, and A. A. Mohammadabadi, "Parallel implementation of color based image retrieval using cuda on the gpu," *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 6, no. 1, p. 33, 2013.

[5] K. Banker, *MongoDB in Action*. Greenwich, CT, USA: Manning Publications Co., 2011.

[6] *Numpy Documentation*. docs.scipy.org.

[7] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[8] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing, 2006.

[9] L. De Lathauwer, B. De Moor, J. Vandewalle, and B. S. S. by Higher-Order, "Singular value decomposition," in *Proc. EUSIPCO-94, Edinburgh, Scotland, UK*, vol. 1, pp. 175–178, 1994.

[10] M. Frank and J. M. Buhmann, "Selecting the rank of truncated svd by maximum approximation capacity," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pp. 1036–1040, IEEE, 2011.

[11] *Scipy Documentation*. docs.scipy.org.

[12] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[13] K. Banker, *MongoDB in action*. Manning Publications Co., 2011.

[14] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.