Building a Scalable Multimedia Search Engine Using Infiniband

Qi Chen[1], Yisheng Liao[2], Christopher Mitchell[2], Jinyang Li[2], and Zhen Xiao[1]

[1]Department of Computer Science, Peking University
[2]Department of Computer Science, New York University

Abstract

The approach of vertically partitioning the index has long been considered as impractical for building a distributed search engine due to its high communication cost. With the recent surge of interest in using High Performance Computing networks such as Infiniband in the data center, we argue that vertical partitioning is not only practical but also highly scalable. To demonstrate our point, we built a distributed image search engine (VertiCut) that performs multi-round approximate neighbor searches to find similar images in a large image collection.

1 Introduction

With the explosion of multimedia information on the Web, there is an increasing demand to build bigger and faster search engines to index such data. Inevitably, such a scalable search engine must be distributed in order to leverage the aggregate memory and CPU resources of many machines.

Distributed search has long been an open challenge. The traditional approach is to *horizontally partition* the index such that each machine stores a subset of all documents and maintains a corresponding local in-memory index. To process a request, the search engine first dispatches the query to *all* machines each of which performs a search locally. It then aggregates the partial results from all machines before returning the final answer to the user. Although this scheme can use the aggregate memory of many machines, it does not have scalable performance: as each request is processed by *all* machines, query latency and performance do not improve as more machines are added.

A promising alternative is *vertical partitioning*. In this scheme, the index of the entire document collection is cut vertically such that each machine stores a subset of the indexed features. To process a request, the search engine needs to fetch multiple indexed features (each of which is located on a different machine) and then filter or join them locally to obtain the final results. This scheme is scalable: since the number of features being looked up is independent of the number of machines, one can potentially improve performance by adding more machines.

Despite its promise for scalability, vertical partitioning has long been considered impractical [9]. This is because multimedia search engines need to sift through tens of thousands of indexed features, resulting in huge communication cost per query. Optimizations that reduce communication significantly increase the number of roundtrips during the search and hence are not practical when running on top of the Ethernet where a roundtrip is around ~ 0.1ms. As a result, existing distributed search engines are almost always horizontally partitioned [3].

In this paper, we argue that now is time to adopt vertical partitioning for distributed search. This revolution is made possible by recent technological trends in datacenter networks that aim to incorporate High Performance Computing (HPC) network features such as ultra-low latency [1, 14]. With a roundtrip latency of several microseconds, a vertically-partitioned search engine can potentially issue tens of thousands of lookups sequentially to refine its results while still achieving sub-second query latency.

We demonstrate the practicality of vertical partitioning by building VertiCut, an image search engine running on top of Infiniband, a popular HPC network. VertiCut implements a distributed version of the multi-index hashing algorithm [15] which performs K-Nearest-Neighbor (KNN) search in a high-dimensional binary space occupied by all images. VertiCut uses a distributed hash table to (vertically) partition the indexed binary codes. To process a request quickly, VertiCut also uses two crucial optimizations. First, it performs approximate KNN search by issuing hash table reads sequentially and stopping early as soon as enough "good" results are found. This optimization drastically reduces the amount of hash table reads done by each query. Second, VertiCut elim-

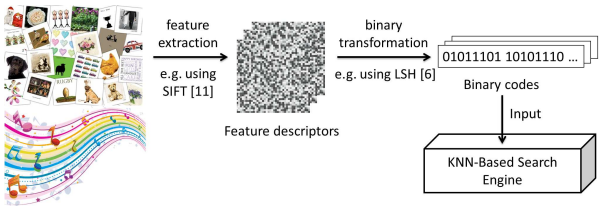

Figure 1: The indexing process of the multimedia search

inates a huge number of lookups for non-existant keys by keeping a local bitmap at each machines. Our experiments show that VertiCut achieves better and more scalable performance compared to a horizontally partitioned search engine. Furthermore, VertiCut's KNN approximation has very little impact on the quality of the search results.

2 Challenges of Distributed Multimedia Search

Background on Multimedia Search: To search multimedia files such as images, music and video, a common approach is based on mapping multimedia data to binary codes. Under this approach, an offline indexer first extracts an array of features using some domain specific algorithms (e.g. using SIFT [11] descriptors for images) for each file. It then maps each high-dimensional feature descriptor into a compact binary code using a transformation function that preserves semantic similarity such as a locality sensitive hash function [6]. The common length of the binary code is 128 bits since 128-bit binary code can achieve more accurate result according to [15]. The indexer then builds an index out of the collection of binary codes. Figure 1 illustrates the indexing process. To search for images that are similar to a given query image, the search engine finds its k nearest neighbors (KNN) in terms of Hamming distance among the collection of binary codes.

There are many ways to perform KNN in the binary code space. The recently proposed multi-index hashing algorithm, MIH [15], provides a particularly efficient way to index and search these binary codes. Specifically, it divides each binary code into m disjoint pieces and indexes each part into a separate hash table (shown in Figure 2).

To perform k nearest neighbor search, the MIH algorithm first divides the query binary code Q into m pieces and then searches each piece Q_i using the corresponding i-th hash table. Suppose the length of the binary code is s. The algorithm performs search in rounds with increasing search radius r starting from 0. In a round that handles a specific search radius r, the algorithm does the following steps:

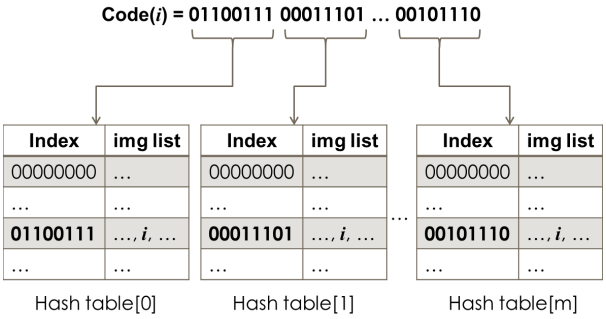

Figure 2: Multiple index hash tables

- For each hash table i, enumerate all $\binom{r}{\frac{s}{m}}$ indices whose Hamming distance with Q_i are equal to r and return all the items (i.e. images' binary codes) in those entries.
- Aggregate the results from all m hash tables into a candidate set of binary codes.
- When there are more than k candidates whose Hamming distance with Q are less than $(r+1)*m$ (the minimum Hamming distance in the next round of search radius $(r+1)$), stop search procedure and return the top k items.

By dividing the binary code into m pieces, the algorithm drastically reduces the number of enumerated index entries within a given hamming distance. Therefore, this algorithm performs much better on large datasets than the naive linear scan algorithm. However, MIH cannot cut the binary code into too many pieces, because a shorter substring length means fewer entries in the hash tables, which cannot separate similar items from far apart items well and leads to a huge but mostly useless candidate set. For 128-bit binary code, it is discovered that the best choice of m is 4 since 32-bit substring code can be effectively used as the index in each hash tables.

Challenges of Distribution: How to distribute the MIH algorithm effectively among n machines? With horizontal partitioning, we can assign a subset of images to each machine which indexes and searches through them using a local MIH algorithm. A central coordinator dispatches the query to all n machines and ranks the $n*k$ results collected from them to return the final k nearest neighbors to the user. As we discussed earlier, this approach does not scale with n since each query is processed by all machines.

With vertical partitioning, we index each of the m pieces of binary code on a potentially different machine. In particular, we can view the MIH algorithm as operating on top of a distributed instead of local hash table to store its indexed codes. When a request comes, the

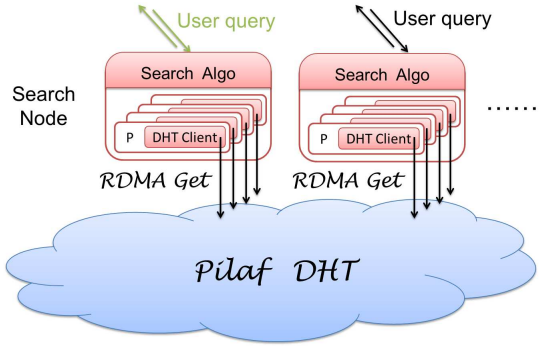

Figure 3: System Architecture

search engine does the KNN search in the same way as the MIH algorithm. This scheme has constant lookup cost regardless of the number of machines n. Better yet, we expect it to perform fewer lookups per query as the underlying image collection gets bigger. This is because the more images in each entry of the hash tables, the smaller the search radius r is required to find k closeby neighbors. The main challenge with this approach is its large communication cost since there is a huge number of binary codes to be enumerated and checked with even a modest search radius r.

3 Design of VertiCut

In this section, we present a fast and scalable multimedia search engine VertiCut that leverages the features of Infiniband to address the distribution challenge.

Basic Design: Figure 3 shows the system architecture of VertiCut. It contains two layers: search layer and storage layer. In search layer, each node starts multiple processes to deal with the user request in parallel. In storage layer, we run the fast in-memory storage Pilaf [14] on each server which uses the RDMA read interface of Infiniband and organize the memory of the servers into a transparent DHT.

Before VertiCut can answer queries, we first build up multiple index hash tables for the whole data collection. We vertically cut the binary codes into multiple disjoint small parts (m parts) with each part consisting no more than 32 bits and build an index hash table for each part of the codes (just like the MIH algorithm does). Then instead of storing different hash tables to different servers, we store these hash tables into our fast distributed in-memory storage (each entry in each hash table generates an entry in our DHT).

When a query binary code Q arrives, the search node divides Q into m parts and starts m processes, with each process searching one index hash table using our simplified Infiniband "get" interface. A master process takes responsibility for performing search iteratively with increasing search radius r, controlling each process to do the search in parallel and aggregating the results from all the processes. When there are more than k candidates whose Hamming distance with Q is less than the minimum Hamming distance in the next iteration of search radius $(r+1)$, the master process stops the search procedure and returns the top k items to the user.

The naive MIH algorithm is not practical due to its huge communication cost which increases explosively with the search radius r. Therefore, we introduce some optimizations to cut down this cost.

Optimization I: Approximate nearest neighbor. In the MIH algorithm, in order to get the exact k nearest results, for each search radius r, we need to check whether there are more than k candidates within a Hamming distance of $(r+1)*m$ after aggregating the results from all the processes. This may cause the search radius r to become large in some cases although there have already been nearly k exact nearest neighbors in the candidate set. We notice that the larger the search radius r is, the faster the search cost grows. Since we are not very strict with the exact k nearest search results, we propose an approximate search algorithm to further reduce the synchronization cost and search radius r while preserve an acceptable precision at the same time. The optimization we make is to change the search stop condition to $|Candidates| >= Factor_{Approx} * k$. This intuitive optimization can greatly reduce the search cost confirmed by our evaluation results.

The trade off is that it may miss some exact search results. For example, suppose we have a full candidate set A ($|A| = Factor_{Approx} * k$), an item b and the search item Q. All candidates in A have some substring whose Hamming distance with the corresponding substring of Q is zero, while other parts of them are far away from Q. b has a small Hamming distance (e.g. 1) with Q in all of its substrings. Then we miss the item b whose Hamming distance to Q is closer than that of candidates in A. Therefore, we should carefully choose the $Factor_{Approx}$ so that we can achieve a proper search precision and query latency. According to our experiment, we find that when $Factor_{Approx}$ reaches 20, the search precision exceeds 80%. However, the query latency does not increase significantly, which is still much faster than the exact search. Moreover, with arbitrary k, the average Hamming distance of our approximate search result is always very close to that of the exact search (the error remains less than 1). Therefore, we choose 20 as our default $Factor_{Approx}$.

Optimization II: Eliminate empty lookups.: We find that most of the entries in each index hash table are empty with no items in it. For example, we have one billion items in the whole data set, and the substring length is

32. Then each index hash table has at least $\frac{2^{32}-10^9}{2^{32}} \approx \frac{3}{4}$ empty entries. In fact, according to our experiment, there are almost 90% empty entries. Although RDMA reads are much faster than the Ethernet reads, they are still slower than local memory reads. Therefore, we should avoid looking up the empty entries in the distributed hash tables. We create a bitmap for each index hash table which records those empty entries and do the real RDMA get operations only if the binary index is in the hash table. We can also use a Bloom filter to avoid reading these empty entries. According to our experiment, using a Bloom filter is slower than using bitmap by 18% while it saves the memory usage by 46%. Since the memory usage of bitmap does not increase with the image size and using bitmap can bring us a 7.5x speedup of the search procedure, we use bitmap as our default choice.

4 Evaluation

In this section, we evaluate the performance of VertiCut on the image search application compared with the traditional horizontal cutting dispatch and aggregate scheme.

Experimental Setup: We set up our experiments on a cluster of 12 servers. Each server is equipped with two AMD or Intel processors, 16 or 32GB of memory, a Mellanox ConnectX-2 20 Gbps Infiniband HCA and an Intel gigabit Ethernet adapter. All servers run Ubuntu 12.10 with the OFED 3.2 Infiniband driver.

We use the one billion SIFT descriptors from the BIGANN dataset [7] and LSH [6] to map from high-dimensional data to binary codes. The default configuration of image search application follows that of [15]. The queries are generated by randomly choosing 1000 images from the dataset.

We run each test case three times and take the average query latency of 1000 queries as our primary metric to measure the effectiveness of our system.

Scalability and Query Latency: To demonstrate the scalability of VertiCut, we first run a 1000 nearest neighbors search with vary data size. As the data size increases from 10 million to 120 million images, we also increases the number of servers from 1 to 12 so that each server always processes 10 million images. We compare three designs, VertiCut on Infiniband, the traditional horizontal scheme on Infiniband (just use Ethernet over Infiniband since the network usage of this scheme is very small) and VertiCut on Ethernet, all of which use the same approximate search algorithm with bitmap optimization. Figure 4 shows that as the data size increases, the query latency of the traditional scheme increases rapidly due to the increased aggregation cost. Surprisingly, in VertiCut, the latency decreases. The reason why the query latency decreases in VertiCut is that as the data increase, the number of images in the same entry of a hash table also increases. Then searching for the fixed k nearest neighbors, the number of entries we need to enumerate decreases (This can be proved by the decreasing number of reads per query shown on the figure 4). This makes VertiCut more effective and scalable on the huge data set. Note that although the query latency of VertiCut on Ethernet has the same decline trend, it is still 8 times slower than VertiCut on Infiniband and 4.4 times slower than traditional scheme.

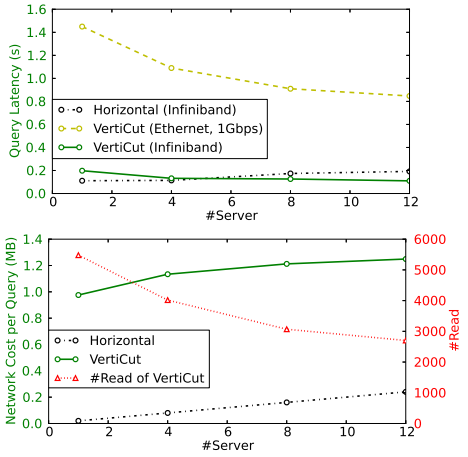

Figure 4: The latency of a query and its network cost (in terms of bytes sent per query) with the number of servers increases.

Effects of k: To show the low query latency of VertiCut on Infiniband in arbitrary k nearest neighbors search, we run a test on 120 million images using 12 servers with varying k (from 1 to 1000). Figure 5 shows that VertiCut on Infiniband is almost twice as fast as the traditional scheme for arbitrary k although its network cost is about 6 times larger than that of traditional scheme, while VertiCut on Ethernet is much slower than the other two. This demonstrates that VertiCut on Infiniband is the best scheme for the large scale multimedia search engine.

Effects of Optimizations: To demonstrate the advantages of our two optimizations, we run k nearest neighbors search on 120 million images with and without our optimizations. We vary the k from 1 to 1000, the comparison of our VertiCut, MIH (No optimization), MIH with approximate search optimization (Approximate KNN) and MIH with bitmap optimization (Bitmap) is shown in Figure 6 (Note that the y axis is in log scale). From the result, we can find that the approximate optimization improves the search speed by 80 times, the bitmap optimization improves 25 times, and our VertiCut achieves at least 550 times improvement. This verifies that our two optimizations are quite reasonable and effective, which can make the distributed multimedia search much more scalable and practical in reality.

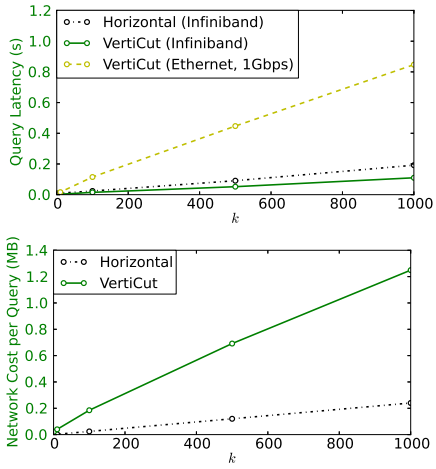

Figure 5: The latency of a query and its network cost as a function of k (the required number of neighbors return).

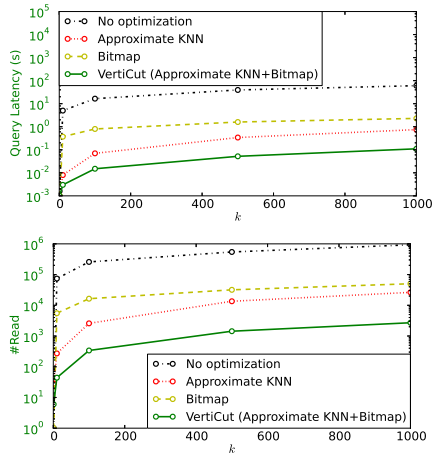

Figure 6: The latency of a query and its number of read operations with and without optimizations.

5 Related Work

There have been several previous works attempting to provide distributed content-based image retrieval [2, 5, 8, 10, 12, 13, 15, 17–20]. They can be divided into two categories: low dimensional and high dimensional approaches. In low dimensional approaches, [5, 18] focus on distributing search on peer-to-peer (P2P) networks based on Content-addressable Network (CAN) [16]. M-CAN [5] uses a pivot-based method to map images from the metric space to a low dimensional vector space. RT-CAN [18] implements a variational R-tree on top of CAN using low dimensional data (e.g. five dimensions). [10, 12, 20] build the nearest neighbors search on top of distributed computing framework MapReduce [4] for low dimensional datasets (data with no more than 30 dimensions). [10] constructs a multi-dimensional index using R-tree. [12] uses a Voronoi diagram-based partitioning to assign objects. [20] maps data into one dimension using space-filling curves and transforms KNN joins into one-dimensional range searches. Although these low dimensional approaches can do fast search in large scale data sets, they cannot achieve precise search results.

For high dimensional datasets, there are three major approaches. The first one is Bag-of-features approach [8, 13, 19], in which each image is represented as a histogram of occurrences of selected features ("visual words") and search is done by using an Inverted Index structure. Works belonging to this approach all use traditional horizontal cut scheme: each server stores and indexes a part of the dataset. We believe that our VertiCut can also achieve a better scalability for this approach. The second one is distributed KD-tree approach. [2] gives an implementation on MapReduce in which a master stores the root of the tree, while multiple leaf machines store the rest. When a query comes, the master forwards the query to a subset of the leaf machines. Unfortunately, this approach has high update cost: each time adding or removing an image, it needs to rebuild the tree. The third one is multiple index hashing approach. [15] provides a distributed scheme for MIH algorithm which stores different index hash tables to different machines. [17] uses the family of LSH functions based on p-stable distributions to conduct multiple hash tables and distributes them using MapReduce. As explained before, this approach is not practical due to its large communication cost.

6 Conclusions

With the rapid growth of multimedia information, multimedia retrieval has become more and more popular in the recent years. How to effectively distribute the search for the increasing huge data collections has become an important challenge with immediate practical implications. In this paper, we present a fast high-dimensional multimedia search engine VertiCut based on the high performance computing network Infiniband to address this challenge. Experiments show that our design can achieve a better scalability and lower response latency, which makes the multimedia retrieval simpler and more practical in reality.

7 Acknowledgments

This work is partially supported by the National High Technology Research and Development Program ("863" Program) of China under Grant No.2013AA013203, the National Natural Science Foundation of China Project 61170056 and the China Scholarship Council (CSC No. 201306010143).

References

[1] ALEKSANDAR DRAGOJEVI, D. N., AND CASTRO, M. FaRM: Fast remote memory. In *USENIX Networked Systems Design and Implementation (NSDI)* (2014).

[2] ALY, M., MUNICH, M., AND PERONA, P. Distributed kd-trees for retrieval from very large image collections. In *Proceedings of the British Machine Vision Conference (BMVC)* (2011).

[3] DEAN, J. Achieving rapid response times in large online services. Berkeley AMP Lab Talk.

[4] DEAN, J., AND GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. *Commun. ACM 51*, 1 (Jan 2008), 107–113.

[5] FALCHI, F., GENNARO, C., AND ZEZULA, P. A contentaddressable network for similarity search in metric spaces. In *Proceedings of conference on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P)* (2007).

[6] INDYK, P., AND MOTWANI, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)* (1998).

[7] JEGOU, H., TAVENARD, R., DOUZE, M., AND AMSALEG, L. Searching in one billion vectors: Re-rank with source coding. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2011).

[8] JI, R., DUAN, L.-Y., CHEN, J., XIE, L., YAO, H., AND GAO, W. Learning to distribute vocabulary indexing for scalable visual search. *IEEE Transactions on Multimedia 15*, 1 (Jan 2013), 153–166.

[9] LI, J., LOO, B. T., HELLERSTEIN, J., KAASHOEK, F., KARGER, D., AND MORRIS, R. On the feasibility of peer-to-peer web indexing and search. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)* (Feb. 2003).

[10] LIAO, H., HAN, J., AND FANG, J. Multi-dimensional index on hadoop distributed file system. In *Proceedings of IEEE Fifth International Conference on Networking, Architecture and Storage (NAS)* (2010).

[11] LOWE, D. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision* (1999), vol. 2, pp. 1150–1157.

[12] LU, W., SHEN, Y., CHEN, S., AND OOI, B. C. Efficient processing of k nearest neighbor joins using mapreduce. *Proc. VLDB Endow. 5*, 10 (Jun 2012), 1016–1027.

[13] MARÉE, R., DENIS, P., WEHENKEL, L., AND GEURTS, P. Incremental indexing and distributed image search using shared randomized vocabularies. In *Proceedings of the International Conference on Multimedia Information Retrieval (MIR)* (2010).

[14] MITCHELL, C., GENG, Y., AND LI, J. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *Proceedings of the USENIX Conference on Annual Technical Conference (ATC)* (2013).

[15] NOROUZI, M., PUNJANI, A., AND FLEET, D. Fast search in hamming space with multi-index hashing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012).

[16] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *Proceedings of conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM* (2001).

[17] STUPAR, A., STUPAR, A., AND SCHENKEL, R. Rankreduce processing k-nearest neighbor queries on top of mapreduce. In *Proceedings of Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR)* (2010).

[18] WANG, J., WU, S., GAO, H., LI, J., AND OOI, B. C. Indexing multi-dimensional data in a cloud system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2010).

[19] YAN, T., GANESAN, D., AND MANMATHA, R. Distributed image search in camera sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys)* (2008).

[20] ZHANG, C., LI, F., AND JESTES, J. Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT)* (2012).