Systems and Architecture Support for Large-Scale Video Search

Carlo C. del Mundo Vincent Lee
University of Washington

Summary. *We propose a scalable system design for video-based search via hardware/software co-design of the memory subsystem and networking layer in concert with indexing and search algorithms.*

To index and search all of the videos in the Internet requires massive resources. Using commodity parts and existing libraries, a conservative estimate for a system requires 32 petabytes to store feature data, 480,000 CPU cores to build and update the index, and 1000 servers to service 1 million concurrent clients at a latency on the order of seconds. Such requirements in both power consumption and performance rival those of the top supercomputers [1]. While it may be feasible to add commodity CPUs to increase raw processing power, the exponential rates of multimedia content creation and consumption exacerbate the scalability of network infrastructure and memory subsystems. In 2013 alone, approximately 127 billion images and 620 million videos were uploaded to Facebook and YouTube respectively [2, 3]. In the same year, traffic volumes from Netflix and YouTube accounted for more than 50% of global Internet traffic [4].

This growth in data has created a landscape where multimedia search engines will be essential to organize and reason about the available data. To date, advances in feature extraction and visual search algorithms have enabled image-based search engines. However, increased I/O and compute requirements have stymied the progress towards video-based search engines. To enable video search at scale, rearchitecting the roles of the memory subsystem and network layer is imperative to address the challenges we encounter below.

I **Feature Storage and Retrieval.** The core search algorithm is highly memory bound. Accessing feature data quickly is critical to satisfying low-latency demands in user-facing applications. We propose to: (1) exploit locality by intelligently sharding *similar* feature vectors and (2) minimize data movement across the networking layer and within the memory subsystem.

II **Indexing.** All feature vectors are indexed into an efficient search data structure which enables sublinear search time scaling with respect to data size. The one time cost of indexing is amortized by the number of searches performed over it. Indexing is three orders of magnitude slower than search time, and while search time grows linearly with the dimensionality of the data, the indexing time grows linearly with both dimensionality and *cardinality.* The index must also adapt and be rebuilt when new video content is uploaded to or deleted from the database. We propose to: (1) design a scalable, distributed indexing data structure for video search, (2) develop a methodology and implementation for incremental reindexing, and (3) integrate application awareness into the system architecture to minimize data movement.

To address the above challenges, we will combine application optimizations with hardware/software co-design techniques. Within a single machine or intra-node level (Section 2), we introduce architectural support for the application using *in-memory* computation and evaluate the potential of a lightweight *accelerator* design for the core search algorithm. At the cluster or inter-node level (Section 3), we propose to also co-design hardware and software subsystems to enable *scalable, distributed data indexing* with extensions for incremental indexing. We also propose integrating in-network application awareness to support *intelligent routing* of data. Many applications in Computer Vision, Robotics, and Machine Learning such as large-scale 3D reconstruction [5], point cloud perception [6], and classifier construction [7] rely on the same algorithms used in video search. Improvements in the scalability and performance of video search ultimately improve the performance of such data-intensive applications.

1 Visual Search

A visual search engine uses images or videos as search input and returns media of similar visual characteristics. Visual search consists of feature extraction and similarity search. State-of-the-art feature extractors generate 4K-dimensional feature vectors per image and approximately 128K-dimensional feature vectors for 5-minute videos [8, 9]. Similarity search compares each feature vector pairwise using k-nearest neighbors (kNN).

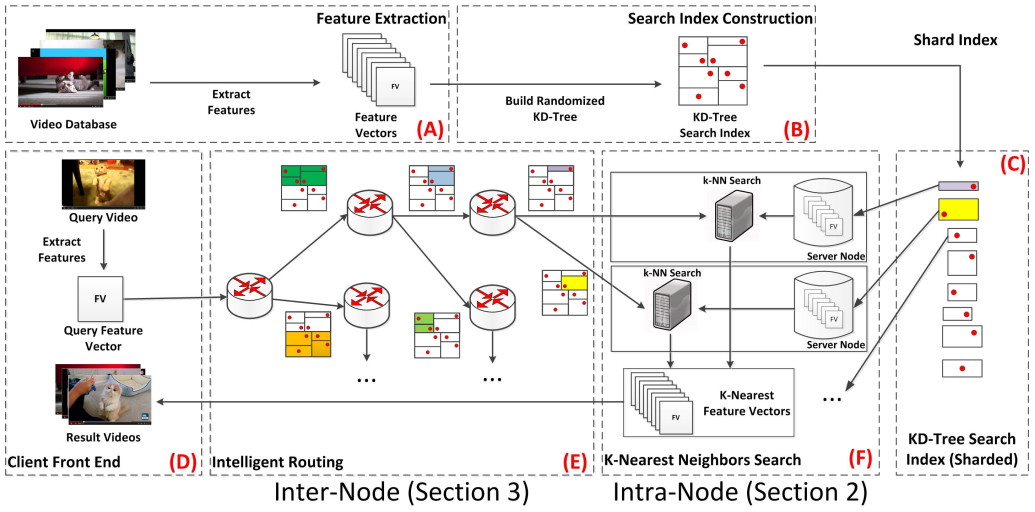

Figure 1: **Proposed Video Search Pipeline**

Prototype for Visual Search. Figure 1 is our proposed system design for video search using kd-trees to partition feature data. In (A), raw videos are transformed into feature vectors. In (B), an index is built from all feature vectors. In (C), feature vectors are intelligently sharded based on their corresponding kd-tree leaves and distributed to individual compute nodes. The client front-end is shown in (D) where a query feature vector is routed through the system in (E). In (F), the query feature vector is routed to a leaf computing node where kNN is performed. The most similar video content is then returned back to the user after performing a reverse lookup on the resulting feature vectors.

2 Intra-Node

In this phase of the project, we will push the performance of the system purely from a software standpoint using traditional CPU optimizations. We will then explore the viability of a lightweight accelerator design.

Characterization, Analysis, and Optimization of kNN (Figure 1F). We will characterize the Fast Library for Approximate Nearest Neighbors (FLANN), a popular library for kNN [10], for its three kNN algorithms: parallel randomized kd-trees, priority search k-means, and linear search. We will analyze the trade-offs and limitations for each algorithm, and we will evaluate system performance based on latency and throughput targets. To characterize each algorithm as a function of load, we will devise a kNN microbenchmark that varies: (1) number of videos in the database or *cardinality*, (2) number of features per video or *dimensionality*, (3) number of requested results or *the k most similar results*, (4) degree of precision or *approximation*, and (5) number of cores. For each of the three kNN search algorithms, we will measure: (1) search time, (2) indexing time, and (3) reindexing time. Then, we will apply software-level optimizations including but not limited to vectorization, cache blocking, and prefetching.

Lightweight Accelerators for kNN (Figure 1F). Once a query vector is routed to a node containing a shard of the indexed kd-tree, we search for the nearest neighbors among the vectors on the node. This memory bound calculation requires pairwise Euclidean distance calculations between each of the feature vectors and the query vector. In a von Neumann style architecture, this requires moving a large number of bytes from memory to perform a relatively small amount of computation. To alleviate this mismatch in I/O versus computation, we will design and embed processing units such as floating point adder, multiplication, and reduction units in memory to enable lightweight computation offloading. We will evaluate the efficacy of such in-memory computation in the context of memory bound workloads such as kNN. We intend to develop a model for in-memory offloading and validate the design using circuit simulation.

3 Inter-Node

The computation, communication, and storage requirements for video search calls for a distributed solution as single machine performance reaches its limits. Distributed computing brings challenges in network communication such as load balancing and efficient routing. To avoid a costly broadcast operation to every compute node given a search query, we plan to integrate application-aware routing schemes and intelligent data sharding into the network infrastructure to reduce both data movement and server compute load.

Scalable and Incremental Indexing (Figure 1B). The lack of a scalable, distributed index is a critical shortcoming of existing kNN libraries. In FLANN, a distributed index is constructed by partitioning data equally between nodes; each node then builds a separate index for its subset of the data. Though this scheme reduces the indexing time by the number of nodes in the system, it naively broadcasts the query to all of the computing nodes. When the system receives a query, all nodes search for the k-nearest neighbor candidates for their data partition. The results from all nodes are then aggregated and sorted by distance to determine the final set of k-nearest neighbors. In a system with N nodes, the aggregation and sort phase processes $k*N$ candidate vectors and ends up discarding all but the top k candidates. As a result, the computation and memory I/O used to generate the $k*(N-1)$ discarded candidate vectors is wasted. To avoid this wasteful computation, we will extend existing indexing techniques in randomized kd-trees and priority search k-means to improve scalability.

A second scalability issue arises as new video content is uploaded to the database; as new content arrives, the index must be updated to reflect the latest uploaded content. When using an indexing data structure such as a kd-tree, blindly adding new data to the leaves causes the tree to become imbalanced. While libraries such as FLANN permit such an operation, the search time of an unbalanced kd-tree index quickly degrades from logarithmic to linear. To solve this problem, eventually the search index must be completely rebuilt to rebalance the search tree and recover to sublinear search times. However, rebuilding the search index is extremely costly and scales poorly if performed frequently. Instead we will extend current indexing algorithms to enable *incremental rebalancing* of indices when adding new content. In particular, we will focus on parallel kd-trees and priority search k-means as our baseline indexing algorithms.

Data Sharding (Figure 1C). Sharding the feature vector database across a distributed system imposes both performance and accuracy challenges. In the face of highly skewed traffic patterns, a single node may experience high contention by many search queries as a result of a "hot" query video resulting in unacceptably high performance degradation. To minimize performance degradation, we will implement a load balancing protocol which will bring up additional replicas of the contended server to help spread the load. Partitioning data across a system also creates accuracy challenges when similar search vectors may straddle the boundary between two or more data partitions. To solve this challenge, we plan to allow a limited amount of data duplication, and overlap partitions at their boundaries. A final challenge is determining optimal data shard size which has direct impact on both performance and accuracy. To gain insight and determine the optimal partitioning strategy, we will draw inspiration from big data graph analytics platforms such as GraphLab [11].

Intelligent Routing (Figure 1E). Rethinking the abstraction boundary between the application and networking layer offers benefits in bandwidth, latency, and throughput. We make the networking layer smarter by providing a mechanism to intelligently route a feature vector to a subset of the data it is most similar with. Such an optimization can be implemented by injecting information about the application data structures inside network hardware such as routers or middleboxes. Given this information, an application-aware intelligent router can hierarchically compute which partition a search query belongs to and route the query towards the appropriate cluster node containing the data. By coupling the search index structure with intelligent routing, we reduce server compute capacity requirements and minimize the data movement required to service a search query. We plan to prototype this network extension using 10G NetFPGAs [12] or comparable networking platforms.

4 Team and Execution Plan

This year long project requires co-design of hardware and software subsystems at both the node and cluster level. Carlo C. del Mundo, who has extensive experience in high-performance computing applications, will focus on designing, implementing, and optimizing the video search system at the node level. Vincent Lee will bring his extensive expertise with networking hardware to scale up the system. Carlo and Vincent are advised by Luis Ceze and Mark Oskin from Computer Architecture and Ali Farhadi from Computer Vision. By the end of the first year of the project, we will:

1. Design and implement an end-to-end visual search system for videos that exceeds single-node DRAM capacity and saturates network throughput (Figure 1).
2. Develop a scalable, distributed indexing structure with extensions for incremental indexing (Figure 1B and 1C).
3. Empirically determine data shard sizes and model its effect on node resource consumption (Figure 1C).
4. Prototype and integrate a NetFPGA-based intelligent load balancer with basic in-network functionality (Figure 1E).
5. Evaluate the trade-offs between different approaches for kNN (i.e., randomized kd-tree, k-means, linear) for high cardinality and high dimensionality feature data (Figure 1F).
6. Evaluate the efficacy of a lightweight accelerator for kNN via circuit-level simulation (Figure 1F).

References

[1] Top500.org, "June 2014 — TOP500 Supercomputer Sites," 2014. [Online]. Available: http://top500.org/lists/2014/06/

[2] Internet.Org, "A Focus on Efficiency: A Whitepaper from Facebook, Ericsson and Qualcomm," 2013. [Online]. Available: http://internet.org/efficiencypaper

[3] YouTube, "Statistics - YouTube," 2014. [Online]. Available: https://www.youtube.com/yt/press/statistics.html

[4] Intel, "What Happens in an Internet Minute?" Oct. 2014. [Online]. Available: http://www.intel.com/content/www/us/en/communications/internet-minute-infographic.html

[5] Agarwal, Sameer and Furukawa, Yasutaka and Snavely, Noah and Simon, Ian and Curless, Brian and Seitz, Steven M. and Szeliski, Richard, "Building Rome in a Day," *Commun. ACM*, vol. 54, no. 10, pp. 105–112, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/2001269.2001293

[6] Rusu, R.B. and Cousins, S., "3D is here: Point Cloud Library (PCL)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 1–4.

[7] Nitesh V. Chawla and Kevin W. Bowyer and Lawrence O. Hall and W. Philip Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[8] Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[9] Heng Wang and Klaser, A. and Schmid, C. and Cheng-Lin Liu, "Action recognition by dense trajectories," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, June 2011, pp. 3169–3176.

[10] M. Muja and D. G. Lowe, "Scalable Nearest Neighbor Algorithms for High Dimensional Data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, 2014.

[11] Y. Low *et al.*, "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, Apr. 2012. [Online]. Available: http://dx.doi.org/10.14778/2212351.2212354

[12] J. Naous *et al.*, "NetFPGA: Reusable Router Architecture for Experimental Research," in *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*, ser. PRESTO '08. New York, NY, USA: ACM, 2008, pp. 1–7. [Online]. Available: http://doi.acm.org/10.1145/1397718.1397720