Scalable Hadoop-Based Pooled Time Series of Big Video Data from the Deep Web

Chris A. Mattmann[1,2], Madhav Sharan[1, 2]
{mattmann,msharan}@usc.edu

[1]Computer Science Department
University of Southern California
Los Angeles, CA 90089 USA

[2]Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109 USA

ABSTRACT

We contribute a scalable, open source implementation [1] of the Pooled Time Series (PoT) algorithm from CVPR 2015. The algorithm is evaluated on approximately 6800 human trafficking (HT) videos collected from the deep and dark web, and on an open dataset: the Human Motion Database (HMDB). We describe PoT and our motivation for using it on larger data and the issues we encountered. Our new solution reimagines PoT as an Apache Hadoop-based algorithm. We demonstrate that our new Hadoop-based algorithm successfully identifies similar videos in the HT and HMDB datasets and we evaluate the algorithm qualitatively and quantitatively.

CCS CONCEPTS

•**Information systems** → **Information retrieval; Image search;**
•**Applied computing** → *Computer forensics;*

KEYWORDS

pooled time series, hadoop, darpa, memex, video.

1 INTRODUCTION

We have seen an increase in online video content used in "dark markets" that deal with illegal trade and sale of goods. In particular, working on the DARPA MEMEX [2] effort our team has worked in conjunction with law enforcement to mine videos on the web to help thwart human, weapons and arms trafficking, and trafficking of other products, such as counterfeit electronics. Multimedia content is a tool used by those selling their wares in these markets to remain hidden from traditional web searches and from bulk analysis - MEMEX is working to change this.

Our team collected a large video set from human-trafficking (HT) focused sites. The dataset - dubbed `HT video dataset` - is 26Gb in size and 6805 in number (including duplicates); 14.3 Gb and 3266 in number (without duplicates). All videos are in MP4 format with an average video size of 3.8MB, and at least 1Mb, and at most ˜10Mb in the deduplicated set - in the set with duplicates the average size is 4Mb, the minimum size 1Mb and the maximum size ˜10Mb. Total length of recording is ≈ 2250 hours with average length of recording = 19.8 seconds.

Law enforcement and other stakeholders have challenged us to reliably and at scale find similar videos in the `HT video dataset`. We used the Apache Tika system [3] initially for this task. Tika is an open source content detection and analysis system that automatically identifies a file's type; and automatically selects a relevant parser to extract text, metadata and language information from it. Tika supports multimedia metadata extraction such as EXIF metadata that is present in images and videos and that tells us scene, content editing, and authorship properties. Simply looking at the metadata is fast, reliable, and effective[4].

While metadata forensics were initially useful, it became necessary to perform actual video pixel (*gradient*) analysis in order to find similar videos. We also found *temporal* relationships and *motion* present in the videos were effective at relating them. Members of our team previously developed the Pooled Time Series (PoT) approach [5] that combines both oriented gradient (image differencing) and optical flow (image motion differencing) to realize such an approach. With PoT we identified relationships between victims; common scenes and housing (*gradient*); common movements such as dancing and/or derobing (*flow*); common clothing (*gradient*) and more.

PoT's existing implementation worked on up to 500 videos from the `HT video dataset`. We could analyze them in reasonable amounts of time (maximum: 2 days). However, as soon as we entered the realm of ˜7000 videos, we were never able to get the PoT reference implementation in Java to complete. We discerned several causes: (1) *Out of Memory (OoM)* – Since PoT computes time-interval based descriptions and must pool and summarize those for motion and gradient difference over many video frames over many 1000s of videos, we constantly ran out of memory; (2) *Sequential Code* – PoT steps were sequentially implemented in Java. That is, first the histogram of oriented gradients (HoG) was computed; then histogram of optical flow (HoF) - each of these are very large matrices with N dimensional vectors for each video. Then the differences between HoG and HoF for each video and mean chi square is computed; and (3) *Instrumentation and Checkpointing* – PoT code lacked critical logging information.

While we added logging and checkpointing to PoT's code, OoM issues and sequential code prevented us from running PoT across the entire `HT video dataset`. As our team includes a member of the Apache Nutch committee that helped to build Apache Hadoop [6], we were inspired to develop a Hadoop-version of PoT that was parallel, and memory efficient. Hadoop is an open source

ICMR '17, Bucharest, Romania
© 2017 ACM. 978-1-4503-4701-3/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3078971.3079019

implementation of Google's Map Reduce [7] and Google File System (GFS) [8] for fast parallel data processing and highly reliable, available and redundant data on commodity clusters.

A roadmap for the paper follows. Sec. 2 describes the general steps of PoT motivating their conversion to Hadoop. Sec. 3 describes the steps of our `Hadoop-POT` algorithm and approach. We evaluate our approach in Sec. 4. Sec. 5 rounds out the paper.

2 POOLED TIME SERIES VIDEO SIMILARITY

Pooled Time Series (PoT) [5] is an algorithm that takes as input a set of N videos and generates as output an $N \times N$ matrix with the pair-wise similarities between each video $1...N$. The diagonals of the matrix are unused, as is half of the matrix since it contains duplicative information. The first step in computing PoT is generating two histograms: one histogram of oriented gradients (HoG) to identify object movements and one histogram of optical flow (HoF) to model relative motion between actor and viewer in a video.

To calculate raw similarity scores we generate all possible video pairs. If the initial video dataset size is N we end up with $\frac{N(N-1)}{2}$ pairs and an $N \times N$ sized matrix V where V_{ij} is the pair of videos v_i and v_j and $i, j \in [1, N]$. For each V_{ij} we assign a similarity score. The chi squared distance (CSD) between all the features of all the possible V_{ij} in V is calculated next. For each video $1...N$ we have two series histograms (HoF and HoG) and three pooling operators so for each video we have total of six features where feature f is denoted $f(s,p)$. After computing CSD, PoT computes the mean CSD (mCSD) for the whole video dataset. After mCSD is computed, PoT computes kernel distances (KD) for the dataset by calculating CSD for a given feature $f(s,p)$ between two videos. We then divide the CSD by its corresponding mCSD. Adding all the kernel distances for one video pair gives us the total distance between the pair.

The overall time complexity of the PoT algorithm is roughly $O(N^2)$. The main factor that influences complexity is the number of videos, all other factors like number of series and pooling operators are constant and chosen manually. In the `HT video dataset` there are nearly 24.5 million pairs. If both pairs have 800 frames with a 200 pixel representation in HoF and HoG we are processing 640, 000 raw values for each pair. This restricts PoT to a few hundred videos to complete in any meaningful amount of time - we ran it for nearly a week without completion on a 20 node Amazon cloud cluster, where each node was M3.xlarge, with 32Gb of RAM.

3 HADOOP POOLED TIME SERIES

Re-envisioning PoT as a Hadoop Map Reduce [6] job was an iterative process. We began by using a *list of video files* as the core data structure to be split and partitioned using Hadoop. Hadoop requires us to define the following constructs: (1) a *Split/Partition* function that will partition the video datasets into equally sized, independent jobs that can proceed in parallel; (2) a *Mapper* function, that, given a set of equally sized independent data from the split, will process that data, and produce some intermediate results; (3) a *Reducer* function, that takes the intermediate output from the *Mapper* jobs, and that combines them into final output results. Hadoop handles *Mapper* and *Reducer* job execution; re-execution in the case of failure; data input/output storage in a highly redundant and available filesystem, HDFS. Hadoop provides native capabilities to split datasets e.g., via hashing.

Since PoT is data-flow dependent e.g., HoF and HoG must be calculated *before* raw similarity score generation, then the raw similarity scores need to be computed *before* mean chi squared computation, and that mean chi square values must be present *before* the final kernel distance can be computed, we were unable to define a single Mapper and Reducer job to represent each of these steps. Instead, the best way to proceed was to define several sets of Map/Reduce tasks, representing each of the stages of the algorithm.

3.1 Implementation Considerations

We considered using the Hadoop Image Processing Interface (HIPI) [9], but instead chose to use core OpenCV since the early implementation of the sequential PoT algorithm was done using direct calls to OpenCV and we wanted to remain backwards compatible with the prior code as much as possible. We also looked at Xuggle [10] but its main focus was streaming video.

We realized early on that we could have separate Mapper/Reducer tasks for HoG and HoF, and that those largely could be done once, offline, and then kept in HDFS for future use in the other Mapper/Reducer task combinations. For example, once the HoF and HoG were generated for all videos, we could create a Reducer that could be used to compute similarities (initially raw), and also mean chi squared distance and kernel distance. We wanted to keep the output format for the matrix simple, so we chose a simple ASCII-text file to represent it, and also to take advantage of Hadoop HDFS and its native support for text files. HoF and HoG outputs were similarly stored as matrices in ASCII oriented text file formats (such as CSV), as were the output raw similarity scores and distances. The initial `Hadoop-PoT` design includes two jobs for HoF and HoG, one job for calculating mean distance (using the raw distances and mean chi squared) and another job for calculating similarity (using kernel distance).

3.2 `Hadoop-POT` v1

We have one job each for calculating HoF and HoG from videos. Each of these jobs takes a file path as parameter and uses OpenCV to calculate features. These features are saved as text files, **.of.txt* and **.hog.txt*.To compute similarity we generate a cartesian product of video set with itself. Consider that we are processing a video set of three videos. For this, we will generate pair like v_1v_2, v_1v_3, v_2v_3. These pair of video names are stored in a CSV file which is provided as input to both similarity calculating jobs. The first job calculates mean chi square distance for whole data set. It reads **.of.txt* and **.hog.txt* for each respective video pair and generates one output file having mean chi squared distance (mCSD). We calculate mCSD for each series and for each pooling operator which gives us $2 \times 3 = 6$ mean distances. Our *Mapper* jobs in this step calculate chi squared distances (CSD) for one pair and the associated *Reducer* job calculates the mean of all the distances. We create a second *Mapper* job that takes a pair of filenames as input and that then calculates similarity score using the kernel distance. The output of this job is a CSV file with a similarity score per video.

The original Hadoop-POT v1 design involved dumping the output results (CSVs, text files) of individual steps from HDFS to the local file system and then by copying those local file system results from the local file system to HDFS for the next stage. This was performed for the following reason - we initially had great difficulty getting the Java OpenCV APIs to read data from HDFS. Errors ranged from intermittent exceptions to I/O errors with Java itself. We later traced these to the implementation bindings in the Java OpenCV library and its API. Another flaw with our v1 implementation was that while dumping *video_name.of.txt* and *video_name.hog.txt* we saved `double` values in a text file. This caused both of our similarity jobs to read these files repeatedly $O(N^2)$ times and to, during the process, convert them from `string` values to `double` values for each value in the matrices. Considering previous example *Mapper* jobs had to convert 640, 000 values to `double` at a cost of about 1.5 seconds per file, scaling this up to 24 million jobs as required by `HT video dataset` would require more than a year of processing time which we did not have. We also realized that we were calculating features repeatedly for each pair instead of caching them.

Instead of text files, we decided that the next Hadoop-POT iteration should use better representations for matrices and vectors and that it should store *all of the values* for our video data set in a single file. Our initial approach of using a CSV with file paths pointing to the computed text files for HoF and HoG for each video masked the real input and its size was thus determined via a single file, rather than the actual total *N* video files present in the dataset. This led to a smaller input splits as Hadoop had no idea about the size of the real input. Input splits should contain values of features rather than path to features for fair input splits. Using less input splits underutilized the cluster and as a result, our Hadoop-POT v1 was quite slow.

Apart from input splits with this input structure we could not use Hadoop's distributed file processing effectively. Hadoop HDFS is made to deal with large files wherein which a file is broken into small splits and each split can be on a different machine. When a job is assigned to a container it is assigned to container that is running on same machine where the split is stored to reduce IO and contention. With input files being paths to features we were unable to use Hadoop's I/O data locality, and instead *Mapper* jobs read data from different nodes instead of those that contained the actual data from the split.

3.3 Redis and Hadoop-POT v2

Our Hadoop-POT v2 architecture converted `text` to `double` format for the values in our matrices by using a Redis cache server. Redis is an efficient, scalable key value based caching utility that operates in a client server fashion with automatic data typing. A client connects to a Redis server and then `PUTS` a key, value pair, where the key and value can be any typed data. Redis efficiently stores that key, value pair, providing a `GET` based interface to retrieve it later rapidly and efficiently. We built our v2 system using a Redis cache with the key being a single video file name and the value being computed feature vector. This reduced job execution time to milliseconds instead of 1.5 seconds per job as in the v1 architecture. However, in doing so, we introduced yet another component into our architecture besides Hadoop and went around Hadoop's native support for similar data structures.

We also tried resource localization by archiving all the *video_name.of.txt* and *video_name.hog.txt* into one big zip file and passing that file as a cached archive in our Hadoop similarity jobs. This reduced time in reading the file for the very first time. We also experimented with different split sizes to increase resource utilization. Redis brought down computation time, but increased our system complexity.

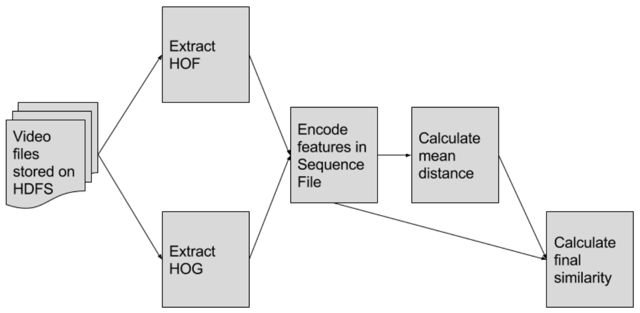
Video files stored on HDFS
Extract HOF
Extract HOG
Encode features in Sequence File
Calculate mean distance
Calculate final similarity

Figure 1: Our final Hadoop-PoT v3 architecture.

3.4 Hadoop-POT-v3

Realizing that Hadoop provided native data structures for fast storage and retrieval of multiple objects, and structures that themselves were easily splittable and usable as input to *Mapper* jobs, and *Reducer* jobs, we eliminated the Redis component introduced in v2 of our architecture and replaced it with Hadoop `SequenceFiles`. `SequenceFiles` are a native Hadoop construct that allow easy combination and summarization of many small input files. Sequence files can be thought of as a zipped version of multiple files. As shown in Fig. 1 we take both features of one video and compute its PoT representation as a java object and serialize that object to bytes in v3 of our architecture. This serialized vector is stored as the value in sequence file, wherein which the key is the name of video file. We use this key later while outputting the overall PoT similarity CSV as output of the algorithm. This sequence file is also generated natively by our *Mapper* jobs and read natively by our *Reducer* jobs, removing any specialized code that we had present in v2 of our architecture.

The final step in our v3 architecture was computation of the Cartesian product on sequence files. We did this by overriding Hadoop's `RecordReader` and `FileInputFormat` classes. Our similarity jobs take generated `SequenceFiles` as input and use our own `CartesianInputFormat` which first permutes through splits and then permutes within a split to generate a pair of video features for our *Mapper jobs*.

4 EVALUATION

Quantitatively our Hadoop-POT v3 algorithm reduced the time necessary for each of our *Mapper* jobs to complete to $\approx$ 42 milliseconds evaluated on `HT video dataset` allowing the needed 24 million+ computations to be executable in a reasonable amount of time. This entire `HT video dataset` was processed using Hadoop-POT v3 on a 10 node cluster in Amazon using M3.xlarge instances with 32Gb memory each, and with EBS storage with 8 containers on each node in ≈ 26 hours. We used cached vector files for this run which takes ≈6.5 hours to generate from input videos in the `HT video dataset`. Complete quantitative evaluation for Hadoop-POT jobs is provided at [11].

Qualitatively evaluating our results involves multiple areas. Simply being able to compute PoT on the `HT video dataset` which we were unable to do using the previous PoT algorithm and implementation is a core contribution of our work. Without our `Hadoop-POT` v3 algorithm and its implementation, large video datasets cannot be run through the PoT algorithm and its previous sequential implementation. One of the satisfying results was identifying almost identical video pairs. Some videos had same content but they differed in their hash codes due to subtle differences. For example, Hadoop-POT was able to identify the same video clips of different length. In addition, the same videos with different resolutions (e.g., 800×600 versus 1024×768) were also easily identifiable using our method.

We also evaluated our algorithm by comparing how well it grouped videos that were identified as similar by automated tagging. We used the Inception-V3 model packaged with Google's Tensorflow [12] to label each video using video frame extraction and image tagging. We then leveraged the labels from Tensorflow to compare `Hadoop-POT` recall rates. We extracted the five most similar videos for all videos in the `HT video dataset` and grouped results by Tensorflow label. We observed that for the top $k = 20$ categories (e.g., beer, toilet, bikini, brassiere) that our algorithm could find related videos accurately and efficiently. Experimentations performed with another labeling toolkit, sklearn[13] and its DBSCAN [14] algorithm, produced similar observations.

To demonstrate that `Hadoop-POT` results were not specific to our own dataset, we evaluated it on the pre-annotated openly available HMDB - Human Motion Database [15]. HMDB is a large data set having 6.8 thousand annotated videos of 51 human motions. The dataset is 1.9Gb with the shortest video being 37KB and largest being 1.3 MB. The total duration of recording is ≈ 350 hours with mean length of a video equals 3.1 seconds. Actions include facial actions like laughing and talking and body movement like walking, climbing etc. HMDB includes object interaction e.g., smoking, kicking a ball, etc.

We ran `Hadoop-POT` on HMDB dataset and it took 26.85 hours to complete. The final size of the similarity matrix is $\approx 6800 \times 6800$ which is ≈23 million unique pairs of videos. We observed that for each video category in HMDB that the most similar category returned by `Hadoop-POT` is also the same. For example videos under category "golf" are most similar to videos under category "golf". We also see "walk" category to be most similar with other categories due to its uneven distribution. Nearly 10% of data set is "walk" videos as compared to 2% under equal division. The complete set of 49 labels we extracted/evaluated is at [11].

5 CONCLUSIONS AND FUTURE WORK

Several opportunities to improve `Hadoop-POT` exist. Videos exist in both datasets in which a person performed similar hand and head movements but that were not identified as similar. Intuition tells us in these examples that HoF and HoG were unable to effectively capture certain movements, e.g., patterns or shapes,and as such adding more descriptors would likely aid these situations. We will support more video features based on convoluted neural networks (CNN) as they have shown promise in detecting such motions.

We will also investigate: (1) removing banners at starting of a video; (2) dividing a video into a set of scenes as done in [16]; and (3) finding and extracting important parts of a video e.g., as in [17]. These steps will help in comparing videos of dissimilar length and removing unwanted parts from the video.

ACKNOWLEDGMENT

This work was supported by the DARPA XDATA/Memex/D3M programs. In addition, the NSF award numbers ICER-1639753, PLR-1348450 and PLR-144562 funded a portion of the work. Effort supported in part by JPL, managed by the California Institute of Technology on behalf of NASA. The authors would like to thank Soumya Ravi and Michael Joyce for working with Dr. Michael Ryoo on hardening his initial PoT implementation.

REFERENCES

[1] M. Sharan and C. Mattmann, "Hadoop pooled time series - github repository," 2016. [Online]. Available: https://github.com/USCDataScience/hadoop-pot/
[2] T. Fox-Brewster, "Memex in action: Watch darpa artificial intelligence search for crime on the fidark web.fi," 2015.
[3] C. Mattmann and J. Zitting, *Tika in action.* Manning Publications Co., 2011.
[4] V. Murdock, C. L. Clarke, J. Kamps, and J. Karlgren, "Second workshop on search and exploration of x-rated information (sexi'16): Wsdm workshop summary," in *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining.* ACM, 2016, pp. 697–698.
[5] M. S. Ryoo, B. Rothrock, and L. Matthies, "Pooled motion features for first-person videos," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, June 2015.
[6] T. White, *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.
[7] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.
[9] C. Sweeney, L. Liu, S. Arietta, and J. Lawrence, "Hipi: a hadoop image processing interface for image-based mapreduce tasks."
[10] C.-R. Lan, H.-H. Lu, C.-W. Yi, and C.-C. Tseng, "A p2p hd live video streaming system," in *Multimedia Technology (ICMT), 2011 International Conference on.* IEEE, 2011, pp. 475–478.
[11] M. Sharan and C. Mattmann, "Hadoop pooled time series - github wiki," 2016. [Online]. Available: https://github.com/USCDataScience/hadoop-pot/wiki/Evaluation
[12] M. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/
[13] F. e. a. Pedregosa, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[14] M. Ester, H. peter Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." AAAI Press, 1996, pp. 226–231.
[15] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "HMDB: a large video database for human motion recognition," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.
[16] Z. Rasheed and M. Shah, "Detection and representation of scenes in videos," *Trans. Multi.*, vol. 7, no. 6, pp. 1097–1105, Dec. 2005. [Online]. Available: http://dx.doi.org/10.1109/TMM.2005.858392
[17] C. Sun and R. Nevatia, "Discover: Discovering important segments for classification of video events and recounting," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.