High Performance Video Processing in Cloud Data Centres

Muhammad Usman Yaseen, Muhammad Sarim Zafar, Ashiq Anjum, Richard Hill
College of Engineering and Technology,
University of Derby,
Derby, United Kingdom
Email: {m.yaseen, m.sarim-zafar, a.anjum, r.hill}@derby.ac.uk

***Abstract*—Mobile phones and affordable cameras are generating large amounts of video data. This data holds information regarding several activities and incidents. Video analytics systems have been introduced to extract valuable information from this data. However, most of these systems are expensive, require human supervision and are time consuming. The probability of extracting inaccurate information is also high due to human involvement.**
We have addressed these challenges by proposing a cloud based high performance video analytics platform. This platform attempts to minimize human intervention, reduce computation time and enables the processing of a large number of video streams. It achieves high performance by optimizing the occupancy of GPU resources in cloud and minimizing the data transfer by concurrently processing a large number of video streams.
The proposed video processing platform is evaluated in three stages. The first evaluation was performed at the cloud level in order to evaluate the scalability of the platform. This evaluation includes fetching and distributing video streams and efficiently utilizing available resources within the cloud. The second evaluation was performed at the individual cloud nodes. This evaluation includes measuring the occupancy level, effect of data transfer and the extent of concurrency achieved at each node. The third evaluation was performed at the frame level in order to determine the performance of object recognition algorithms. To measure this, compute intensive tasks of the Local Binary Pattern (LBP) algorithm have been ported on to the GPU resources. The platform proved to be very scalable with high throughput and performance when tested on a large number of video streams with increasing number of nodes.

Keywords– Cloud Computing, Video Processing, High Performance and GPUs, Video analytics.

I. Introduction

Conventional monitoring systems have been deployed for the security of organizations and individuals. These systems are usually manual, do not scale well and consume considerable time in finding objects of interest. Humans are prone to errors, therefore, the probability of errors and incorrect information becomes higher in these systems. With advancements in video capture, storage and computing technologies, it has been realized that organizations and individuals can be protected by deploying automated monitoring systems. The video monitoring systems have been introduced and excelled since the last two decades. These systems use object detection [8], object recognition [8], object tracking [10][12] and motion prediction [16] algorithms. These algorithms work on the individual pixels of a video frame, requiring considerable computing resources and processing time which becomes quite expensive for large sets of video data. These systems are required to process video streams data rapidly and with a reasonable computational cost. However, the algorithms perform slowly and in-efficiently due to their compute intensive nature. In order to reduce the processing time, video processing algorithms have been ported to a cluster of CPUs [14][15]. A CPU consists of a limited number of processing cores and is well suited for sequential and serial processing of data. The CPU based implementations are unable to achieve the desired performance levels due to their sequential processing and the compute intensive nature of the object detection, recognition and tracking algorithms.
Specialist hardware resources such as GPUs have recently been used to speed-up the compute intensive parts of the algorithms by offloading the processing from CPUs to GPUs. GPUs are massively parallel devices consisting of a large number of light weight processing cores to perform compute intensive tasks concurrently. The number of processing core in a GPU varies according to its specification. Several studies [2][3] have been performed recently to harness the power of GPUs for video processing algorithms. The size and numbers of video streams necessitate a system that can be scaled up to process the increasing volumes of video data. In this regard cloud computing, due to its scalability and agility, has become a mainstream platform that has been consistently demonstrated as an infrastructure for scalable processing of massive sets of video data [26][27].
In this paper, we propose a cloud based high performance video processing platform. The platform aims to speed-up the processing of large number of video streams stored in the cloud based data centres. It also aims to provide high throughput processing of video streams by maximizing the use of GPUs mounted on the cloud nodes. In order to achieve these objectives, the following three measures have been employed within the platform. The first measure addresses scalability by processing large sets of video streams within the cloud platform. These video streams are fetched from the cloud based video storage and are distributed to the cloud nodes where each node processes these video streams in parallel. The second measure parallelizes the video processing tasks within the GPU mounted cloud nodes, thus achieving high performance and high throughput within each node of the cloud system. These tasks include decoding a video stream and the processing of each decoded video frame on GPU mounted cloud nodes. The results are later stored back in a cloud database. The third measure has been taken to parallelize the

processing of individual frames and keep the GPUs saturated once the video streams have been decoded. This involves optimizing the data transfer within the cloud nodes, achieving optimal occupancy by concurrently processing multiple video frames and offloading compute intensive tasks of the object recognition algorithms on GPUs.
The rest of the paper is organized as follows. Section II reviews the literature and identifies the research gaps in the existing research. In Section III, we present our approach and architecture for the cloud based high throughput platform. In Section IV, we explain the algorithmic details and discuss the functionality of the local binary pattern histogram algorithm. The implementation details including the workflow and dataflow of the platform are discussed in Section V. Section VI describes the experimental environment and the evaluation parameters of the platform. In Section VII, the results of the evaluations are presented whereas the paper is concluded with some future directions in section VIII.

II. Related Work

The research community has realized the compute intensive nature of video processing algorithms and a number of approaches have been proposed in the literature. In this section, we will be discussing different studies that have made contributions towards the resolution of this problem and their strengths and weaknesses.

Object Detection and Recognition

Video processing and video analytics is an extensively studied research topic and has applications in various domains. Quite a large number of algorithms are available today making noticeable contributions for the detection of various types of objects. These objects may include persons, faces or vehicles. The Haar Cascade Classifier algorithm [8] is one of the most widely used algorithms for object detection. It is based on Haar like features and uses Adaboost [9] for dimension reduction and classifier training. Each feature is represented by a weak classifier in the cascade. A combination of all these weak classifiers is used to make a strong classifier which is further used for detecting objects. The HOG features based person detection [10] works by dividing an image into multiple cells connected with each other. The algorithm then computes histograms of oriented gradients for each cell. These gradients describe the appearance of an object in an image and can be used to train a classifier for detection.
The object recognition is a much more complicated endeavor than object detection. Local Binary Patterns (LBP) [11], Local Ternary Patterns (LTP) [12] and others [13] are widely used approaches for object recognition. These algorithms compare the pixels of each cell with its neighborhood. If the value of cell is greater than its neighbor than 1 is written, otherwise 0 is inserted. In the case of LTP, the difference between central pixel and neighboring pixels is encoded into a ternary code. A normalized histogram then provides the feature vector. Computing the value of each cell is fairly compute intensive.

Object Detection and Recognition in Clouds

When object recognition is performed on large scale data, it requires large computational resources. Leveraging cloud infrastructure to fulfil the needs of high computational resources has been a preference in both academic and industrial sectors. An approach based on SIFT [19] and Gabor [20] descriptors was proposed in [21], [22] to recognize food images. These features were clustered by K-means algorithm [23] to achieve an acceptable recognition rate. To achieve scalability and performance gains, cloud computing paradigm was utilized [22]. It was concluded that for small amount of data, cloud computing performance was not very promising. This is because of the fact that cloud needs to prepare job runtime environment before a job is served. However, for large datasets cloud processing efficiency is far better because clouds are designed to support these kinds of workloads.
A cloud computing based object recognition system using two dimensional principle component analysis was implemented on a Hadoop based cloud system. However, this approach was less effective for object images with multi-illumination. The massively parallel cloud computing based approach [23] was also used to process astronomical images where a parallelised mapper without applying a parallelised reducer was used. However, there was no improvement in the image processing routines. Another cloud based system for analyzing large scale videos was developed using the MapReduce based clusters [23]. A cluster of six computers was created and video processing algorithms were ported to it. The execution time of algorithms was reduced as compared to a single system. Since the experiments were executed on a small dataset, scalability was not addressed. The improvement in accuracy was also not the focus of this work.
The use of GPUs as a high performance general purpose computing resource was commenced in 2009 [2]. The initial study was based on the implementation of large scale unsupervised machine learning algorithms, deep belief networks [17] and sparse coding [18] on GPU resources. These studies unfolded the power of GPUs to the research community by achieving a speedup of 5 to 15 times as compared to the CPU implementations. A parallelised object detection approach achieved a speed-up of 1.91 times on GPUs [7]. A parallelised motion estimation approach using a full search algorithm on GPUs achieved a speedup up of around 50 times than its CPU implementation [4]. Furthermore, studies such as [5] proposed a GPU implementation of Haar Cascade Classifier algorithm [8] and achieved a speedup of 13.8 times than a CPU implementation. Another Haar based face detection approach on GPUs [6] achieved 2.5 times speedup as compared to its CPU based implementation.
These approaches provide a good overview of previously proposed video processing platforms and algorithms. However, most of the above cited approaches lack scalability and agility. These are also computationally expensive as the dimensionality of features is very high. In the proposed platform, we have adopted the LBPH object recognition algorithm. LBPH is one of the most commonly adopted object recognition algorithm. The reason behind this is its computational simplicity and high accuracy rate. The compute intensive parts of LBP have been ported as a GPU kernel on each cloud node. Additionally, in order to harness the computation power of each node within the cloud, three level parallelism has been proposed in this paper.

III. Approach and Architecture

Figure 1 depicts the architecture of our high throughput video processing cloud platform. The main objective of the platform is the optimal utilization of each available resource on a cloud node to provide high throughput multi-video processing. These video streams are captured from video sources and are stored in a cloud based storage. These stored video streams are later acquired by the platform for processing. The following three level parallelism has been proposed to achieve high throughput in the analytics process.

1) Node Level Parallelism: The video processing cluster manager fetches videos streams from the cloud based video storage and is responsible for managing workload across each node in the cluster. The workload represents the number of video streams being analysed/processed. At the node level, multiple video streams are received from the video cluster manager and processed concurrently. Each video stream is allocated to a particular node where the video stream manager launches the tasks for decoding and processing the video stream.

2) Video Stream Level Parallelism: Each video stream is processed in three stages. These stages are 1) video decoding, 2) frame processing and 3) storage of the processed frames. The video stream manager executes the video processing stages in parallel to yield better performance and throughput. The video processing stages are handled by the sub modules, namely the decoding, processing and storage modules. The decoding module reads a video stream, extracts encoded video frames, decodes these frames and stores the decoded frames in an input frames buffer. The input frames buffer resides within the main memory of the node. Hence, as soon as the input frames buffer starts receiving frames from the decoding module, the processing module fetches the decoded frames in parallel as input and applies the video processing algorithms on it.

3) Frame Level Parallelism: The third level of parallelism is achieved at the frame level where each pixel within a frame is processed on a GPU. Compute intensive parts of the video processing algorithm are executed on a GPU by the GPU Kernel. This GPU Kernel is launched in parallel within a grid. Each grid consists of thread blocks having a number of threads. Each thread is mapped with each pixel of a frame and is responsible for launching the kernel and processing the assigned pixel, in parallel. Hence, the GPU kernel is launched as per grid and thread block size which is defined by the size of a frame. Once the threads are finished and synchronized, the results of the processed frames are pushed to the output frame buffer. This is another temporary memory buffer for maintaining the processed frames. It is accessed by the storage module working in a separate thread and is responsible for fetching the processed frame from the output buffer and saving it to the cloud database. In order to evaluate the platform, the compute intensive parts of the object recognition LBP algorithm have been ported on the platform. The main aim of LBP is face recognition from video streams. The algorithmic details of LBP are explained in Section IV.
The components of the platform and the interaction between these components are depicted in Figure 1. The cluster manager fetches videos streams from the cloud based video storage and distributes these to the participating nodes in the cluster.

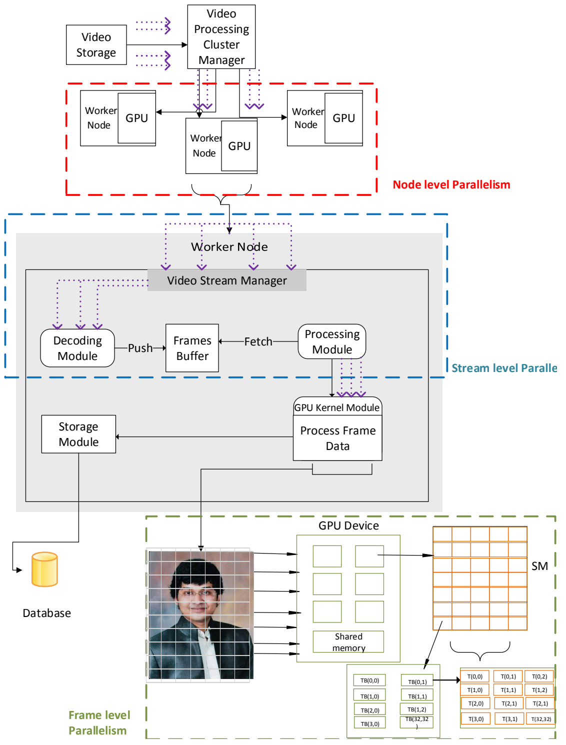


Figure 1: Approach and Architecture within the cloud node

Each node processes multiple video streams concurrently. The processing of video streams is handled by the video processing manager. As stated earlier, each video stream is processed in three stages by the video stream manager. These stages are video decoding, frame processing and storage of the processed frames. The decoding module reads a video stream, extracts encoded video frames, decodes these frames and stores the decoded frames in an input frame buffer. The processing module takes the decoded frames as an input and applies the video processing algorithm to extract useful information from the frames. The GPU kernel implements and executes the compute intensive part of the video processing algorithm on a GPU. It launches a number of threads on GPUs to process the compute intensive part of the video processing algorithm. Once the threads are finished and synchronized, the results of the processed frame data are pushed to the output frame buffer.

IV. Video Processing Approach

We present our cloud based object recognition approach in this section. A user of the system submits an object recognition job and specifies the video streams to be analysed. These video streams are first fetched from the cloud based video data store and are decoded to extract individual video frames. Each frame is then processed individually for object recognition. The recognition algorithm is applied on each frame for the recognition purpose.

The system is evaluated with a case study for face recognition. The face area of each person is analysed. The algorithm used for face recognition is LBPH [10]. This was originally designed by the texture classification community. Later it was used for facial recognition tasks and was tested on various datasets under complex conditions. The algorithm performed well and achieved high accuracy rates. It has the capability to capture fine grained details in the video frames. Because of these advantages of LBPH we have employed it on our platform. It makes use of local binary patterns in order to generate feature vectors. LBP features are computed by dividing the examined window into cells. Each cell contains a window of 16x16 pixels. Then each pixel in the cell is compared to its neighbouring pixels. If the value of centre pixel is greater than its neighbour pixel, 1 is stored at that location. If it is the other way round then 0 is stored at that location. This is known as the labeling of pixels. Then histogram is calculated and normalized over the frequency of each number occurrence. These normalized histograms give a feature vector of the window.

The original LBP operator was later on extended to work for images at different scales. This was achieved by enabling the operator to use the neighborhoods of different sizes. The neighborhood was defined by a circle containing sampling points which are evenly spaced. The radius of the circle can be increased to cover large neighborhood which in turn increases the number of sampling points. In this way images at different scales can be covered by the operator. The operator was further extended to use the uniform patterns only. A uniform pattern is the one in which there are maximum two bitwise transitions. This transition can be from 0 to 1 and it can also be other way around. The pattern will said to be non uniform if there are more than two transitions. Since, it was proved by experiments that uniform patterns account 90 percent of all the patterns so for each uniform pattern, histogram has a separate bin and all non-uniform patterns are assigned to one separate bin.

In order to perform face recognition, the face image is divided into multiple blocks or regions. Then for each block or region, LBP histogram is computed as explained above. The feature vector of the whole image is a combination of all LBP histograms of all regions in an image. This combination of histograms in one extended histogram makes it spatially enhanced. It contains not only the appearance information but also contains the information of spatial relations between different regions. The spatially enhanced histogram contains the information about an image at three levels: pixel level, region level as the pixels of a specific region are summed up, and holistic level as the histograms of all regions are combined together to form an enhanced histogram.

Compute Intensive modules of LBPH

In order to reduce the computation demands, the LBPH algorithm is used for object recognition in video streams. Figure 2 depicts the compute intensive parts of LBPH algorithm. It can be seen that computing LBP and histogram are the most compute intensive steps of this algorithm.

The core LBP computation algorithm is the same in both CPU and GPU implementations. However, working of both the algorithms is different which affects the performance and use of available computing resources. In the CPU implementation, all the pixels are processed sequentially. For a video frame of height 528 pixels and width 704 pixels, the total number of pixels to be processed is 371712. This sequential processing is very slow and time consuming when processing data for large number of video streams.

On the other hand, the GPU implementation is called GPU kernel. GPU executes numerous threads as soon as a kernel is launched. The number of threads on a GPU depends upon the available cores, GPU memory and a number of registers. These number of threads on a GPU can be controlled by defining the thread block and grid sizes. However, the number of threads may exceed the frame height and frame width. This needs to be controlled by processing only those threads which come within the boundaries of frame height and frame width. As frame data is available on GPU memory, each pixel is processed by the threads executing in parallel on a GPU. Once the processing of frame data is finished, the processed frame data is copied back from GPU memory buffer (device) to the CPU memory buffer (host).

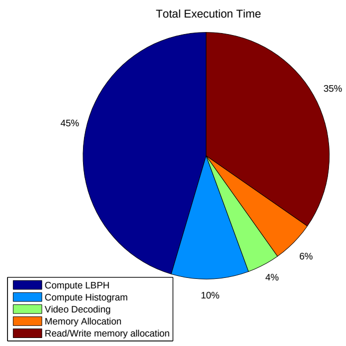


Figure 2: Total Execution time of a stream within a CPU node

V. Implementation

Figure 3 depicts the workflow implementation within a node in the platform. Video streams are assigned to each node by the cluster manager after fetching from the cloud based video storage. In each node, videos are decoded by the decoding module and each frame is pushed to the frame buffer. For this paper, we will name this buffer as an Input frame buffer. The input frame buffer is responsible for maintaining the list of frames available in a video and makes it available for the processing module.

The processing module fetches frames from the Input frame buffer and calls the LBP kernel. Once the results are received from a GPU, they are pushed to the processed frame buffer location in a node. The storage module is working in a separate thread which is responsible for fetching the processed frame from the buffer and saving it to the database.

During this process, methods provided with CUDA have been utilized. CUDA provides a set of APIs in order to perform computation on GPUs. The first and foremost step is the allocation of memory size. The device (GPU) memory is first allocated according to the size of frame data. The frame data is then transferred from the host RAM to the GPU memory. We implemented and evaluated different data transfer mechanisms including page-able memory, pinned memory and zero copy. Once the frame data has been transferred, the kernel is launched with an appropriate configuration. Once the processing is complete, the frame data with changed values of pixels is transferred back to the host memory location.

Before execution of the GPU kernel, GPU memory allocation, thread allocation and Kernel configurations are vital part of a GPU application. Kernel is the code in the CUDA application model where the compute intensive parts are executed in parallel. The Kernel is executed in separate threads, grouped

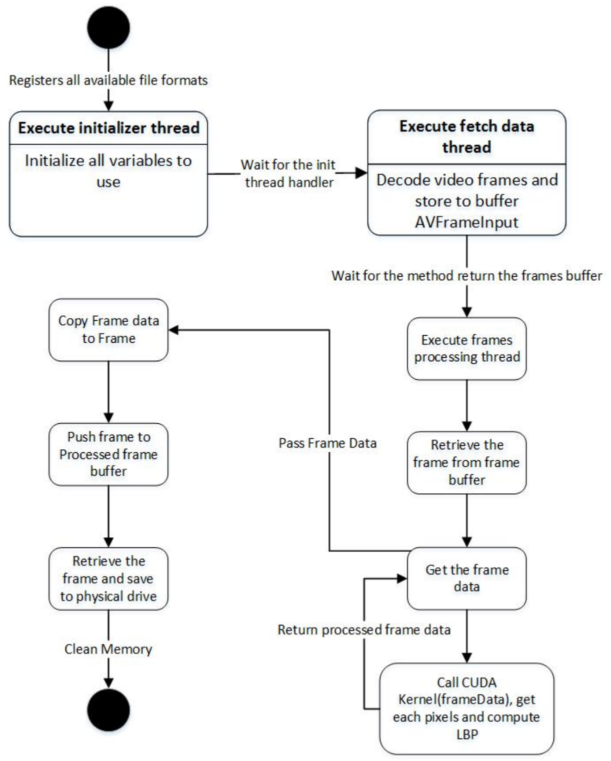

Registers all available file formats
Execute initializer thread
Initialize all variables to use
Wait for the init thread handler
Execute fetch data thread
Decode video frames and store to buffer AVFrameInput
Wait for the method return the frames buffer
Execute frames processing thread
Retrieve the frame from frame buffer
Get the frame data
Call CUDA Kernel(frameData), get each pixels and compute LBP
Return processed frame data
Pass Frame Data
Copy Frame data to Frame
Push frame to Processed frame buffer
Retrieve the frame and save to physical drive
Clean Memory

Figure 3: Workflow Implementation Within a Node

in blocks and grids. A single block is a collection of warps, whereas, a grid is a collection of blocks.
Each warp can handle a maximum of 32 threads simultaneously. The CUDA Work Distributor (CWD) is responsible for allocating threads blocks on GPU. These threads blocks are allocated at the initialization stage of kernel execution. The CUDA streams provide the mechanism to perform kernel execution concurrently. Kernel is executed on the default stream if multiple CUDA streams are not initialized. We varied the number of CUDA streams from one to ten in our implementation.

Data Flow: We explain the flow of data for processing a video stream in this section. A video stream is fetched by the video cluster manager and is assigned to one of the nodes in the platform. The node starts decoding the video stream. While decoding is being performed, the decoded frames are pushed to the memory buffer in CPU RAM.
In order to process a decoded frame data on a GPU, this frame data needs to be transferred to the GPU RAM. Once the necessary memory allocations are made, the frame data is later transferred to the GPU RAM. An appropriate memory allocation and memory transfer mechanism is required to launch GPU kernel on the decoded frame data. Once the data has been processed, it is then returned back to the memory buffer of the CPU. This process is performed in each node of the cluster for each video assigned to the node by the video processing cluster manager and is well depicted in Figure 4.

Workflow: The CUDA Integrated development environment comprises of host machine (CPU) and the device (GPU). The host is responsible for executing the host code, whereas, the device is responsible for executing instructions on GPUs. The distribution of workflow between the host and device is depicted in Figure 5.

Our platform is initialized with the acquisition of video streams from the cloud based video storage. Video streams are assigned to each node by the cluster manager. These video streams are decoded and individual video frames are extracted. These frames are then stored to the CPU RAM in the memory buffer. A separate thread is responsible for fetching the frame from the buffer and processing the frame data. In order to process frame data on an available GPU, a memory allocation request on GPU is sent from the host to device. Once the memory allocation on the host memory is successful, the frame data is transferred. The GPU kernel implements the compute intensive parts of the algorithm and consists of a set of instructions to be executed on the device. A number of parallel threads are launched on the device. Each thread is responsible for launching the kernel and processing the assigned pixel data. Once the processing is completed, threads are synchronized and the processed frame data is transferred to the memory buffer of frame, located in the main memory of the host.

VI. Experimental Setup

The experiments are executed on a GPU powered cloud infrastructure. Each cloud node is equipped with one GPU. Each node has an Intel Core i7 3.60 GHz processor, 16 GB RAM and an ASUS GeForce GTX 780 GPU. The GTX 780 is a Kepler architecture based GPU and consists of 12 Streaming Microprocessors (SM), 2304 CUDA cores and 3 GB memory. Each streaming processor can execute 2048 threads in parallel. These threads are executed in 64 warps with each warp executing 32 threads in parallel. Each thread has a local memory of 512 KB and a total of 255 registers per thread. A total of 16 thread blocks are used by each SM with 2048 bytes of shared memory per block.
The video processing platform is evaluated for concurrency, throughput, resource consumption and the time taken to process the video streams in the experimental results presented in this paper. The concurrency of the platform is evaluated by processing multiple video streams on each node using CUDA streams. Each CUDA stream is processing a single video stream. The total video data analysed during these experiments consists of 5 hours of video recordings. The length of each individual video stream is around 2 minutes and 51 seconds. Each video stream is H.264 encoded. The resolution of a the

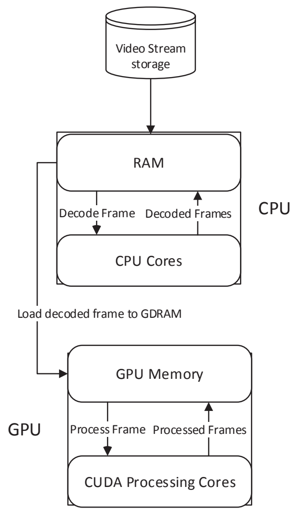


Figure 4: Data Flow within the cloud node

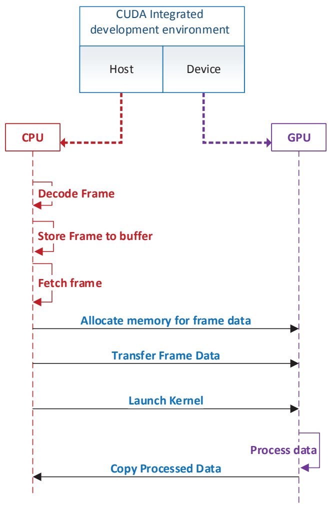


Figure 5: Sequence of Process Execution in the node

video stream is 704x528 with a frame rate of 30fps. The data rate of the video stream is 421 kbps and bitrate is 461 kbps. The total numbers of video frames in each video stream are 5130 and the data size of an individual video frame is 371KB. The execution time, memory bandwidth, memory consumed, GPU occupancy, throughput and performance are the evaluation parameters of the platform. These parameters are evaluated against single and concurrent CUDA streams. The time taken for processing individual video frames and multiple video streams is an important evaluation parameter. The total video processing time includes video reading time and processing the video stream for extracting the useful information from it.

These parameters are compared for different GPUs and with the CPU implementation. For this purpose, GT 610 GPU has been used. It consists of a total 48 CUDA cores and has 1 GB memory. It is a Fermi based GPU and consist of one SM. A total of 8 thread blocks per SM are supported and a total of 48 warps per SM are supported where each warp consist of 32 threads. Each thread is supported with a total of 63 registers per thread with a local memory of 512 KB.

The difference in architectures led us to see differences in the results as well. This enabled us to select a suitable GPU for a certain type of applications. The final results of our 4 node GPU enabled cloud platform are compared with a 4 node Hadoop cluster without GPUs. The 4 nodes Hadoop cluster was deployed on Openstack with each node having 8 GB RAM and data storage of 100 GB. The installed operating system is cloud image of Ubuntu 15.04. The algorithms of OpenCV 3.0 RC1, using the JNI wrappers compiled in Java 1.8 for native C++ library, were utilized for the LBP algorithm.

VII. Results and Discussion

In this section we present the evaluation results of our high performance video processing platform using the case study discussed in section IV. These results have been produced by executing the face recognition case study on the platform. The video processing results from the GPU platform are compared with the CPU based cluster. The same dataset is executed on different sets of GPUs to evaluate the suitability of a GPU architecture for face recognition applications.

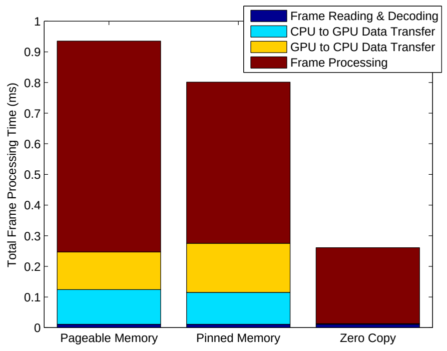


Figure 6: Frame Processing Operations and Processing Time

Different aspects of the GPU platform such as memory allocation, memory bandwidth, GPU occupancy and the time taken by algorithm are evaluated to analyse the performance of the platform . These experiments also cover several other aspects such as the effect of increasing number of concurrent video streams on the platform with an increasing cluster size in order to evaluate the scalability of the platform. The remainder of this section explains the evaluation results of the platform.

1. Effect of Data Transfer between CPUs and GPUs in a Cloud Node: This set of experiments evaluates the effect of page-able, pinned and zero copy memory allocation and data transfers on the throughput of the video processing platform. The number of video streams in these experiments is varied from 1 to 10 within a node. For a 704 x 528 video resolution, each video frame required 371.712 KBs of memory allocation. The data transfer per second is 10.89MB for one video stream recorded at 30fps. The maximum data transfer per second varied from 10.89MB to 108.9MB for 1 to 10 video streams in this set of experiments.

The total elapsed time to transfer video frame data from CPU to GPU memory with pageable memory, pinned memory and zero copy is summarized in Table I. The zero copy memory transfer took the least time as it allows the GPU to access video frame data directly from the main memory by mapping the CPU and GPU memory address space.

Transferring the processed video frame data from GPU to CPU memory was fastest with zero copy memory transfer and took the least time. The time taken to transfer processed video frame data from GPU to CPU memory is summarized in Table I.

CUDA Streams	Data Transfer Time (in Milliseconds)					
	CPU to GPU			GPU to CPU		
	Pageable	Pinned	Zero Copy	Pageable	Pinned	Zero Copy
1	0.113	0.104	0.001	0.123	0.1	0.001
2	0.212	0.117	0.025	0.16	0.151	0.011
3	0.321	0.208	0.12	0.311	0.233	0.0869
4	0.36	0.215	0.126	0.374	0.293	0.126
5	0.42	0.286	0.197	0.438	0.415	0.196
6	0.471	0.313	0.216	0.489	0.502	0.275
7	0.56	0.373	0.267	0.597	0.686	0.328
8	0.612	0.431	0.316	0.65	0.83	0.38
9	0.643	0.499	0.322	0.795	0.878	0.485
10	0.733	0.517	0.397	0.872	0.982	0.509

Table I: Data Transfer Time from CPU to GPU and GPU to CPU

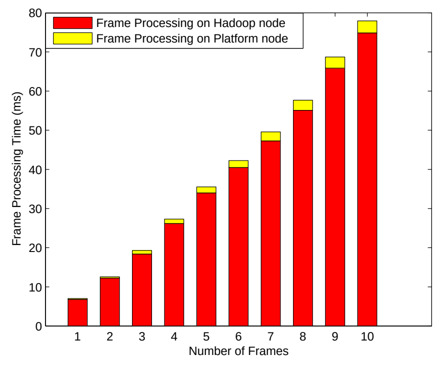


Figure 7: Elapsed Time to Process multiple frames

***2. Frame Processing Time**:* The processing time of an individual video frame is analysed on CPU, GT 610 and GTX 780 GPUs. We have also analysed the video frame processing time with page able, pinned and zero copy memory transfer mechanisms.
The first experiment was performed to determine the effect of data transfer mechanism while processing a single frame. Figure 6 depicts the required time to process a single video frame with different data transfer mechanisms. The figure depicts the impact of Frame Reading and Decoding time, frame data transfer from CPU to GPU and GPU to CPU and frame Processing time on the total Frame Processing time of each video stream. The zero copy remains the fastest because of the direct video frame data access from GPU to CPU. In zero copy mechanism, the GPU memory address space is mapped to the CPU memory address space and a GPU accesses CPU memory as its own address space. This means that whenever data is to be copied from host to device, a GPU is able to access the particular memory location in host memory. The processed video frame data is also copied back to the host from the GPU memory in the same way. The zero copy memory is used in the rest of the experiments reported in this paper.

The second set of experiments was performed for evaluating the impact of processing multiple frames on the platform against the CPU based Cluster. Figure 7 shows the time it takes to process multiple frames on the platform. The processing on platform is faster because of the availability of dedicated hardware processing cores. Secondly, the three level approach discussed previously spawns numerous threads for each frame within the dedicated hardware which launches further numerous lightweight threads to operate on individual pixels of each video frame in parallel. In each lightweight thread, the kernel is launched for each pixel. The processing time of multiple video frames within multiple video streams reduces significantly as depicted in Figure 7.

The third set of experiments was performed to evaluate the effect of data transfer in order to achieve high performance within the platform. The performance was measured with the number of frames processed per second. Figure 8 depicts the number of frames processed per second using different data transfer mechanisms. The highest throughput was also achieved by the zero copy mechanism with the increasing number of video streams. From the graph it is observed that the highest throughput is achieved with two video streams. The frame processing per second for one video stream remained 135 whereas with two concurrent video streams, frames processing per second reaches 100 for each video stream. With three video streams, a steep reduction was observed in the frame processing per second which reaches to 38 for each video stream. It is due to the reason that data transfer time from CPU to GPU and GPU to CPU remains optimized with two CUDA streams as summarized in Table I.

***3. Resource Consumption**:*

GPU Memory Consumption:

GPU has limited memory and this memory is further divided into global/shared and local memories. In this evaluation, we analysed the GPU memory usage while processing multiple video streams using the CUDA streams. The GPU memory consumption is also dependent on the memory bandwidth of a GPU and the number of stream pipelines. As mentioned earlier, the GTX 780 has 3GB of memory, 16 stream pipelines and its memory bandwidth is 288.4 GB/s. The memory consumption kept on increasing with the increase in number of CUDA streams. The GT 610 has 1GB memory and only one stream pipeline. GT 610 has a memory bandwidth of 14.4 GB/s. It was able to process only one CUDA stream at any given time and the memory consumption is constant. The processing of multiple video streams on GT 610 is sequential and CUDA streams has no effect on the overall throughput of the system. This behavior is well depicted in Figure 9.

CPU Memory Consumption:

The video processing is initiated on CPUs and only the compute intensive parts of the algorithm are off loaded to a GPU. One thread is launched for each CUDA stream. Each CPU thread is responsible for decoding a video stream, extracting video frames from it, storing the frames in frame buffers and initializing a CUDA stream. A linear increasing behaviour in the memory usage is observed with an increasing number of CUDA streams. This behaviour is depicted in Figure 10.

GPU Occupancy:

GTX 780 has 2304 processing cores and can execute 2048 threads in parallel. These threads execute in blocks of threads with a minimum of one warp of threads per block. Each warp consist of a maximum of 32 threads. A thread starts execution after all of its required resources (memory, registers etc.) and the data become available. A higher number of thread blocks leads to a higher GPU occupancy.

GPU Occupancy is an indicator of the utilization of a GPU. A higher utilization of GPU can improve the memory data

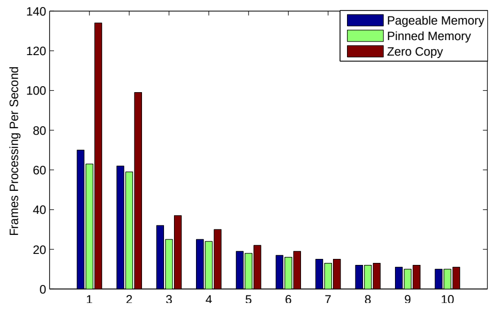


Figure 8: Frame Processing Time and number of Video Streams

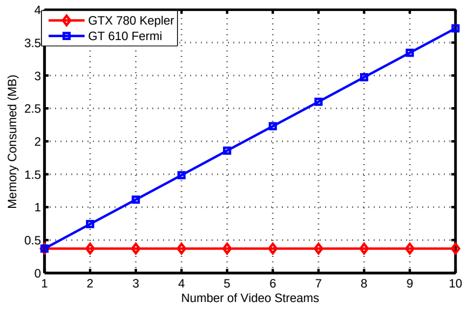


Figure 9: Memory Consumption of GPU on Each Cloud Node

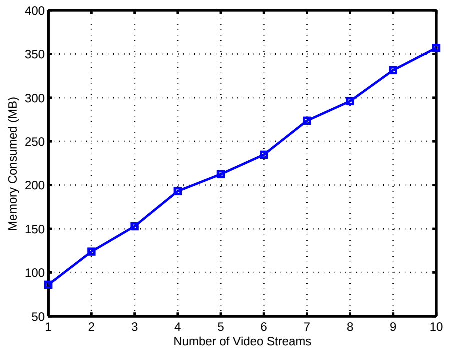
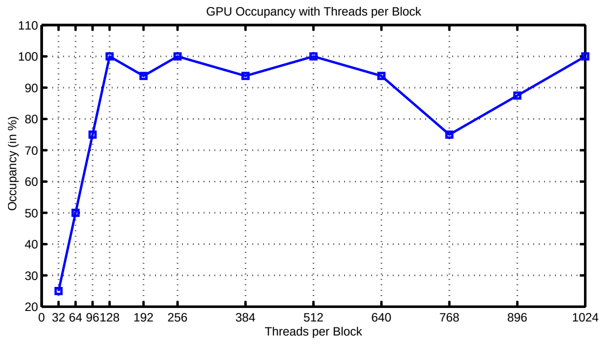
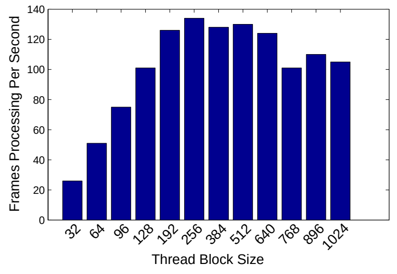


Figure 10: CPU Memory with Increasing number of Video Streams

transfer and global memory load latencies. However, a higher occupancy does not necessarily mean a higher performance. Memory/data bound applications such as video processing do not benefit from a higher GPU occupancy. The processing is



(a) Highest Occupancy Achieved with Varying Block Sizes



(b) Frames Processing Per Second Achieved with Varying Block Sizes

Figure 11: Optimal Performance Achieved

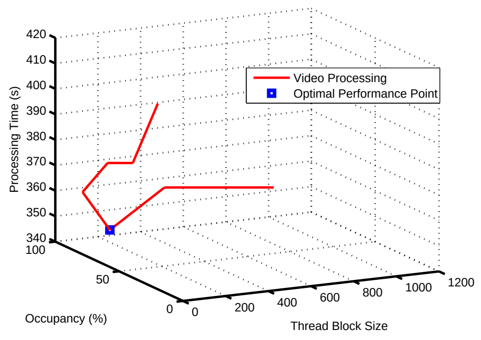


Figure 12: Effect of Thread Block Size and Occupancy on Five Hour Video Stream Processing

dependent on the data transfer from CPU to GPU as well as data accessibility within the GPU RAM. This means that while accessing the data from the GPU memory, threads do not stall the memory access.

The first experiment was performed to evaluate the GPU Occupancy at each block size when executing the video processing kernel for each pixel. As shown in Figure 11(a), the GPU occupancy depicted an increasing trend with an increasing block size until 128 thread blocks and became 100% . A block size of 768 thread blocks depicted lowest performance of 75% occupancy because threads stalled the memory access for the frame data within the GPU memory. In order to validate this experiment, the frame processing per second was measured at different block sizes depicted in Figure 11(b). The highest number of frames processed per second was found on the block size of 256 thread block size. The effect of thread block size and occupancy achieved on video processing is depicted in Figure 12.

4. Video Processing Operations: Video processing operations include video reading and decoding, data transfer between CPU memory and GPU memory and algorithm execution on the data. This experiment was performed to evaluate the time required to complete these operations and their implications on video stream processing in the platform. As the video streams in a cloud node are increased from 1 to 10, we analysed that video reading and decoding operations take maximum percentage of time as shown in Figure 13.

It is because of the reason that in our platform a video stream is acquired by the video processing cluster manager from the physical storage. This means that a constant transfer of video

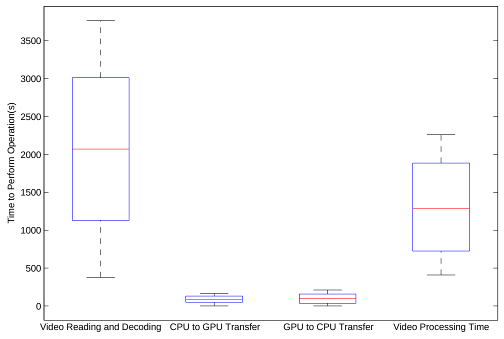


Figure 13: Mean Time while Processing Multiple Video Streams

streams from the physical storage to the memory of video processing cluster manager remained a major bottleneck during this study. This aspect of the platform is being dealt in the next phase of our experiments which will make use of in memory storage approaches.

5. Comparison with non- GPU based Cloud:

High Throughput Comparison:

A higher video resolution means processing of more data. The effect of varying video stream resolution in our platform is depicted in Figure 14 where video streams of different resolutions are used. The length of the video stream remained 2 minutes 51 seconds in this set of experiments.
The platform achieves 15 times higher throughput than our CPU based cloud. The video processing time on a GPU also includes overall time for decoding a video stream. Table II summarizes the processing times for different video resolutions. The highest speedup was achieved for the high resolution videos. It is due to the reason that processing each pixel sequentially is inefficient and does not utilize all of the available CPU resources. Whereas, a GPU processes each pixel of a video frame in parallel. No significant speedup was observed for the low resolution video streams due to the data transfer overheads from CPU to GPU. It is due to the fact that a GPU waits for the data transfer from a CPU.
The results of experimentation performed on our platform using multiple videos of 704x528 resolution were 15X times faster than a non GPU-based cloud as shown in Figure 15. The reasons behind the achieved speed-up is an optimized frame data transfer from CPU to GPU and parallel processing of video frames data on GPUs. We analysed that using zero copy data transfer mechanism provides a GPU with direct access to CPU memory location and provided a higher throughput than any other CPU to GPU data transfer mechanism. The other reason behind the achieved speed-up is an optimized utilization of resources on the GPUs. This was achieved by determining the optimal number of thread block size required to process each frame. The concurrent processing of multiple video streams also contributed in achieving the high throughput. The maximum throughput is achieved when each node is processing two video streams concurrently. The GPU memory allocation and the video frame data transfer remain the fastest with two video streams. Increasing the number of video streams to three or more resulted in a sequential video data transfer and consequently the processing on the GPUs became slower.

Scalability Comparison:

Figure 15 depicts vertical as well as horizontal scalability of the platform against the CPU based Cloud platform. Multiple video streams were processed on the CPU based cluster. Over the non GPU based cloud, 5 hours of video data took nearly 25.416 minutes to process on a cluster having 4 worker nodes. The figure also provides a comparison of scalability between the CPU based cloud and the proposed platform. The figure provides a comparison of an increasing number of nodes with the increasing number of video streams. From Figure 15(a) it is visible that the overhead within the CPU based Cluster is continuously increasing resulting in an increase of inefficiency and processing time. Figure 15(b) depicts the

Video	Video Processing Time		
Resolution	CPU	GT 610	GTX 780
256x144	1.19	0.913	0.84
352x240	5.41	2.338	1.86
480x360	11.363	3.104	2.508
640x480	19.147	4.340	3.804
704x528	21.474	3.839	3.202
1280x720	41.437	7.783	7.033
1920x1080	95.999	15.391	12.341

Table II: Processing of Video Streams with Different Resolutions

scalability of our high performance video processing platform. From the figure it is visible that the platform is highly scalable, both vertically and horizontally because of the exploitation of a heterogeneous architecture, the three level parallelism approach and due to the optimal utilization of resources.

VIII. Conclusion and Future Work

A high performance video processing platform has been presented in this paper. The platform is capable of processing multiple videos concurrently and efficiently. A three level parallelism approach has been implemented to achieve high performance, high throughput and scalability in the platform. Several factors such as an optimized resource utilization of GPUs, the optimal data transfer mechanisms, an improved occupancy and an efficient memory allocation contributed towards the high throughput. Whereas, mapping each and every pixel of video stream on the light-weight GPU threads remained vital factor for achieving high performance in the platform.
The results of face recognition case study showed an accuracy of 95 percent with a performance gain of 15 times as compared to other contemporary approaches. The reason behind this performance is the optimal load distribution and resource allocation approach where the load is intelligently shared between the GPUs and CPUs on each node. Because of this approach, CPUs on each node keep on decoding the frames and GPUs keep on processing them in parallel. The performance achieved is dependent on the resolution of each video stream. When the resolution of a video stream is increased, a higher performance was achieved, meaning the platform is better suited for large scale video analytics.

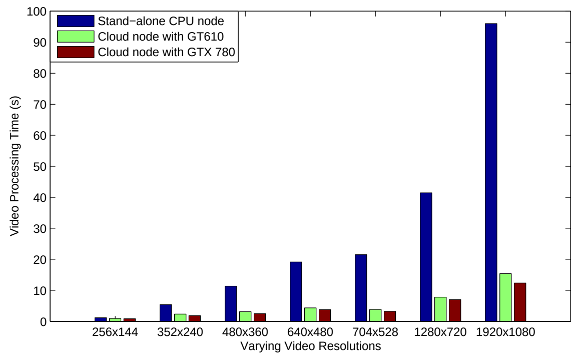
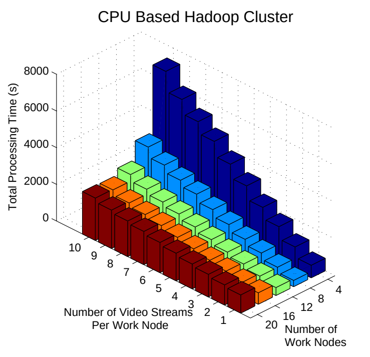
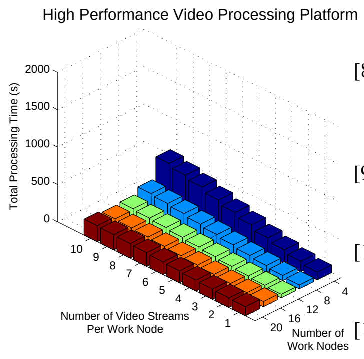


Figure 14: Video Processing comparison on different Platforms



(a) Scalability of CPU Based Hadoop Cluster



(b) Scalability of Video Processing Platform

Figure 15: Vertical and Horizontal Scalability

It is important to mention that transferring video stream from video storage to the main memory of worker nodes remaines a bottleneck in achieving further performance gains. Video streams are first loaded from video storage to the main memory and are then processed on the platform. I/O delays occur due to the constant movement of video streams and this becomes one of the major performance bottlenecks. Leveraging an in-memory processing cluster which provides distributed in-memory storage architecture is likely to overcome this problem and is one of the future directions of this work.
We will also like to make the system more generic by detecting and recognizing objects from different object classes such as cars, bikes and pedestrians. The optimization of detection and recognition algorithms will also be the part of our future work. Since the platform and algorithms are not suitable for live streaming analytics, mechanisms will be addressed where GPUs can be utilized on the in-memory processing cluster. With the utilization of an in- memory cluster coupled with the computation power of GPUs, we can anticipate a higher performance of the video processing platform for live video streams analysis.

References

[1] "CUDA programming guide",http://docs.nvidia.com/cuda/cuda-c-programming-guide/

[2] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors", *In Proceedings of the 26th ACM Annual International Conference on Machine Learning*, pp. 873-880, 2009.

[3] P. K. Chong, , E. K. Karuppiah, and K. K. Yong, "A Multi-GPU Framework for In-Memory Text Data Analytics", *27th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2013, pp. 1411-1416.

[4] Z. Jing, , J. Liangbao, and C. Xuehong, "Implementation of parallel full search algorithm for motion estimation on multi-core processors". *In IEEE 2nd International Conference on Next Generation Information Technology (ICNIT)*, pp. 31-35, 2011.

[5] W. Wang, Y. Zhang, S. Yan, Y. Zhang, and H. Jia, "Parallelization and performance optimization on face detection algorithm with OpenCL: A case study" . *Tsinghua Science and Technology*, vol. 17, no. 3, 287-295, 2012.

[6] D. Oro, C. Fernández, J. R. Saeta, X. Martorell, and J. Hernando,"Real-time GPU-based face detection in HD video sequences.", *In 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 530-537, November 2011.

[7] E. Totoni, M. Dikmen , and M. J. Garzarán, "Easy, fast, and energy-efficient object detection on heterogeneous on-chip architectures", *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 4, 45, 2013.

[8] P. Viola, and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features", *In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. I-511-I- 518, 2001.

[9] Y. Freund, and R. E. Schapire, . "Experiments with a new boosting algorithm", *In Proceedings of the 13th Conference on Machine Learning (ICML)*, vol. 96, pp. 148-156, July 1996.

[10] N. Dalal , and B. Triggs, "Histograms of oriented gradients for human detection", *In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 886-893, June 2005.

[11] Y. Fang,, and Z.Wang, "Improving LBP features for gender classification". *In International Conference on Wavelet Analysis and Pattern Recognition ICWAPR*, vol. 1, pp. 373-377, 2008.

[12] K. B. Low, and U. U. Sheikh, "Gait recognition using Local Ternary Pattern (LTP)", *In International Conference on Signal and Image Processing Applications (ICSIPA)*, pp. 167-171, 2013.

[13] T. Tuytelaars, and K. Mikolajczyk, "Local invariant feature detectors: a survey", *Foundations and Trends® in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177-280, 2008.

[14] "Hadoop image processing interface," http://hipi.cs.virginia.edu/

[15] T. Abdullah, A. Anjum, M. F. Tariq, Y. Baltaci, and N. Antonopoulos, "Traffic Monitoring Using Video Analytics in Clouds", *In Proceedings of the IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pp. 39-48, December 2014.

[16] J. Spoerk, C. Gendrin, C. Weber , M. Figl, S. A. Pawiro, H. Furtado, and W. Birkfellner, "High-performance GPU-based rendering for real-time, rigid 2D/3D-image registration and motion prediction in radiation oncology", *Zeitschrift für Medizinische Physik*, vol. 22, no. 1, pp. 13-20, 2012.

[17] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets". *Neural computation*, vol. 18, no. 7, pp. 1527-1554, 2006.

[18] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data", *In Proceedings of the 24th International conference on Machine learning*, pp. 759-766, June 2007.

[19] J. Luo, E. Takikawa, S.Lap, M. Kawade, L. Bao-Liang "Person-Specific SIFT Features for Face Recognition", *IEEE International Conference on Acoustics, Speech and Signal Processing* ICASSP, 2007.

[20] C. Liu, and H Wechsler. "Independent Component Analysis of Gabor Features for Face Recognition", *IEEE Transaction on Neural Networks*, 2003.

[21] P. Duan, W. Wang, W. Zhang, F. Gong, P. Zhang, and Y. Rao, "Food Image Recognition Using Pervasive Cloud Computing", *IEEE International Conference on Cyber, Physical and Social Computing*, 2013.

[22] W. Wang, W. Zhang, F. Gong, P. Zhang, and Y. Rao, "Towards a Pervasive Cloud Computing based Food Image Recognition", *IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, 2013

[23] A. Likas, N. Vlassis, J. Verbeek, "The Global K-means Clustering Algorithm", *Pattern Recognition*, vol. 36, pp. 451-461, 2003.

[24] L. Zhu, X. Zheng, P. Li, and Y. Wang, "A Cloud Based Object Recognition Platform for IOS", *International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI)*,pp. 68-71, 2014.

[25] K. Wiley, A. Connolly, S. Krughoff, J. Gardner, M. Balazinska, B. Howe, Y.Kwon, Y.Bu, "Astronomical Image Processing with Hadoop", *Astronomical Data Analysis Software and Systems*, p.93, 2011

[26] T. Abdullah, A. Anjum, M F Tariq, Y. Baltaci, N. Antonopoulos, Traffic Monitoring Using Video Analytics in Clouds, *IEEE/ACM 7th International Conference on Utility and Cloud Computing*,pp. 39-48, 2014

[27] A. Anjum, T. Abdullah, M. Tariq, Y. Baltaci, N. Antonopoulos, Video Stream Analysis in Clouds: An Object Detection and Classification Framework for High Performance Video Analytics, *IEEE Transactions on Cloud Computing*, pp. 1, 2016