SwaCom: A Swarm Computing framework for image-sensing applications in surveillance domain

Arpita Ambavane[1], Sai Akhil Teja[2] and Tushar Gautam[3]

Abstract— **Recognition and perception based applications such as image recognition are on the rise with advancement in camera technologies that act as the end-nodes in the IoT ecosystem. These applications recognize the user's surroundings and augment it with information and/or media. These applications are latency sensitive and have a soft-real time nature - late results are meaningless. On the one hand, given the compute intensive nature of the tasks performed by such applications, processing is typically offloaded to the cloud. On the other hand, offloading such applications to the cloud incurs network latency, which can increase the user-perceived latency. With modern technologies like edge computing which place edge nodes closer to the end nodes, the cloud communication is avoided significantly. Though edge nodes reduce the overall perceived latency, they still require network connectivity which becomes a bottleneck for end devices installed in remote areas. Cameras today have more in-built compute resources than before. Moreover, with advancement in networking technologies such as WiFi-HaLow, NB-IoT, LoRaWAN etc the network connectivity among end devices has improved. We explore the potential of off-loading processing to the end nodes i.e. Smart camera systems which can then leverage distributed computing framework with its peers to perform a recognition problem - image recognition - collectively and efficiently in the domain of surveillance where network connectivity is generally poor.**

I. INTRODUCTION

Intelligent processing of the continuous stream of visual data collected by end-devices like cameras has been a crucial and ever growing challenge. Initially, cameras had limited capacity for interpreting images and were mainly assigned to basic projects, such as reading bar codes. With advancement in networking technologies, cameras could upload data to the cloud where visual data processing can happen. Cloud based solutions have the curse of latency and are not best suited for real-time analysis [8] [9]. With recent advancements such as edge computing some of the computation could be off-loaded to edge nodes which are not only closer to the end-devices but also have relatively more compute power leading to overall reduction in data transfer to the cloud [9]. As the latency of sending data to the cloud for analytics could be high, particularly when processing real-time events, performing inference on real-time data may require the inference engine to be as close to the data source as possible.

For certain applications deployed in remote areas where network connectivity is poor, both cloud-based and edge-computing based processing might not be the most efficient option [3]. For instance, surveillance cameras installed in remotely located industry warehouses or national border security may not have good network connectivity. In such a scenario, it's crucial to further optimize the data transfer and perform computation at the end nodes as much as possible. With more processing power available in Smart cameras today and further advancement in the IoT communication technologies such as Wi-Fi HaLow, NB-IoT, LoWARAN, etc it's rewarding to leverage the two and perform computations at the end-nodes [1].

Low power robust communication is a crucial factor in the success of performing end nodes computations. While networking technologies like Bluetooth and LoRaWAN exist, they are not best suited for long range communications, of the order of hundred meters, which might be desired in the security camera installations [1]. Recently introduced Wi-Fi HaLow [1] for IoT devices which works in sub 1 GHz frequency enables individuals and businesses to use their own infrastructure in private, controlled networks. Moreover it's low

[1]arpita.ambavane@colorado.edu
[2]sai.mekala@colorado.edu
[3]tushar.gautam@colorado.edu

power and energy efficient with native IP support which suits the requirements of end nodes like camera remarkably well.

Furthermore, there are now 64-bit Arm based processors in cameras supporting GNU compiler toolchain and support for OpenCV - a popular computer vision library [5]. The 64-bit CPU architectures come with enhanced register support and larger memory maps. The increase in both number and width of registers mean that larger data sets can be processed with a reduced number of memory accesses, resulting in faster data processing. Such advancements in networking and processing at the end-nodes enables it to perform advanced analysis at the end-nodes itself. Such smart cameras will leverage the computing capabilities of other peer smart cameras to perform computation collectively.

There are two main challenges for designing such a distributed framework for camera systems. The first challenge is device heterogeneity i.e. devices having heterogeneous hardware characteristics like different ISAs, processors and cores. Sometimes a smart camera might have to interact with low computational/no computational power cameras. The second challenge is network conditions i.e. for peer cameras with low network connectivity - stragglers - or cameras with no network connectivity. Our distributed computing framework attends to these two challenges and aims to solve them.

For measuring the performance of our proposed framework, we consider processing delay/latency for processing each image. Once the processing is complete at the peer nodes, they send an acknowledgment back to the request camera node which marks the end of the computational task. Here, processing delay is the time taken between the request to be offloaded to a peer camera and for the acknowledgement to be received.

In Section II we discuss the related work corresponding to the existing communication technologies, various computing frameworks and processing capabilities of smart cameras.

II. Related Work

We explored the following computing approaches: *Cloud-only system*, *Edge Computing*, *Computer Vision*, *Mobile Cloud Computing*, *Mobile Cloud Image Recognition* and *Web Caching Communication Technologies*, *Modern CPU architecture in Smart Cameras*, *Swing for remote distributed computation on mobile devices*.

A. Cloud-Only System

Since, the recognition problems are compute intensive tasks and the IoT devices are short on computing resources, offloading the computation load to the cloud shall drastically reduce the computing resource usage at end-device. In [6], [7] the authors discuss about various cloud computing technologies and trends and their impact on the low compute intensive devices. However, the drawback of this approach is that there is an increase in latency due to network delays between cloud and end device.

B. Edge Computing

The issue of cloud computing hurting latency-sensitive perception and recognition applications has been highlighted multiple times in literature. [8] and [9]. Compared to the cloud, a "cloudlet" [8], a server one wireless-hop away from the mobile user, provides significant speedup. It makes the case for moving compute resources close to the user to avoid network latency. [9] presents a more extensive computing paradigm called fog computing. Fog computing proposes to extend the idea of cloud computing all across the network between the cloud and the user, thus bringing computational resources closer to the user and making the network smarter. However, in these approaches there would be a service interruption/outage if there is a network failure or a high network delay.

C. Computer Vision

There have been advances in the field of computer vision and image recognition algorithms to enable efficient processing locally on mobile devices. Feature extractors and descriptors such as SURF [10] and ORB have certainly made it possible to do recognition on the devices. However, as reported in [11], doing it locally does not scale beyond tens of images, and we want to achieve recognition of thousands of objects. Deep learning and neural networks have also made it possible to achieve high recognition accuracy, and it has been

achieved on mobile devices as well, but again the device alone cannot achieve the diverse recognition that is needed [11]. Given that mobile devices alone cannot support recognition across more than a few categories, they still need to offload these tasks to a the backend server, typically in the cloud. Cachier [12] is a system that accelerates such applications using edge servers as a recognition cache.

D. Mobile-Cloud Computing

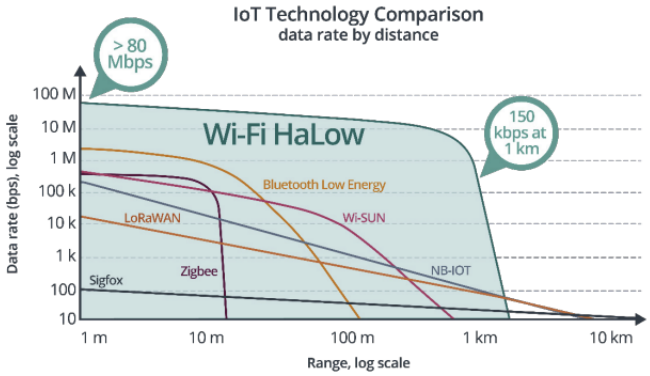
There are several systems that have been proposed to offload computing from mobile devices onto the backend servers or the cloud to enable more efficient applications. These systems divide the application at different granularities to offload parts of it to the cloud, optimizing for different objectives. [19] offloads specific functions from an application to the cloud and optimizes for energy consumption. [18] on the other hand offloads threads of execution to the cloud to minimize latency. Mobile-cloud image recognition: Techniques have also been proposed, in the mobile-cloud paradigm, to optimize recognition related applications. Glimpse [17] proposes using a video frame cache locally on the device that hides the latency of the cloud based pipeline and also tracks objects locally on the device to select frames that actually get offloaded. OverLay [24] creates an inter-object distance map and uses it to shrink the search space.

E. Communication Technologies

For local distributed computation, it's essential to have a reliable communication network which not only supports low power consumption but also has a long range with good data transmission rates. Bluetooth communication, though reliable, has limitations in long range communications. Similarly LoRaWAN [1] also has poor data transmission rates beyond 10 meter range. Wi-Fi HaLow, based on IEEE 802.11ah solves some of the key problems outlined before. With sub 1 GHz frequency, unlicensed spectrum and free usage, it enables individuals and businesses to use their own infrastructure in private, controlled networks. Key highlights of the Wi-Fi HaLow are as follows:

- Long range (upto 1km from Access Point), penetration through walls, removes the need, in many scenarios, for expensive meshes and repeaters.
- Low power and energy efficient.
- Data rates 150 Kbps - 86 Mbps (6Mbps in 88 meter range)
- Native IP support including UDP/TCP.
- WPA3 and Enhanced Open security.
- Wi-Fi HaLow also reduces the hardware costs of devices that would otherwise require multiple radios. For example, a device using LoRaWAN for long-distance connections at very low data rates might require a second radio utilizing another technology, such as Wi-Fi, to provide occasional firmware updates. A single Wi-Fi HaLow connection can accomplish both goals.
- Network topology: Star / Relays - simplifies the network architecture.

Figure below summarises the comparison of WiFi HaLow with other communication technology.



All of the above are significant advancements which are essential for building a distributed computation framework at the end-nodes to perform computation locally.

F. Modern CPU architecture in Smart Cameras

Arm based 64-bit CPU architectures [5] come with enhanced register support and larger memory maps. The increase in both number and width of registers mean that larger data sets can be processed with a reduced number of memory accesses, resulting in faster data processing. The increased register support also means that the developer can take advantage of the state of the art in compiler optimizations, which will further boost performance. When the benefits of improved register support are combined with the increased

memory map of 64-bit architectures, the software has direct access to more local data for processing, reducing need for the memory swapping, which impacts performance as data sets are swapped in and out of local memory.

Imaging requirements are increasing, smart cameras are embracing 4K resolution in the low-end cameras and 8K resolution in the mid- and high-end cameras. Moving from 1080p to 4K doubles the data rate of the incoming video streams from the cameras with the same encoding schemes. Furthermore, if the frame rate needs to increase from 15fps to 30fps to 60fps to better identify objects, then the need for data rate and performance would continue to scale accordingly. Higher fps will enable cameras to detect, identify, and recognize smaller objects and fast-moving objects with much better precision. The need for simultaneous 4K encoding from multiple streams further increases the demand for faster CPU performance. 64-bit processors are well suited to meet these demands.

Furthermore, to meet such a need for increasing performance and bandwidth requirements for imaging and ML workloads, smart cameras need support for floating point operations. The Armv8-A 64-bit architecture increases the FLOPS[1]-per-cycle performance for single-precision by 2x and double-precision by 5x, which improves the user experience.

G. Swing - Swarm Computing for Mobile

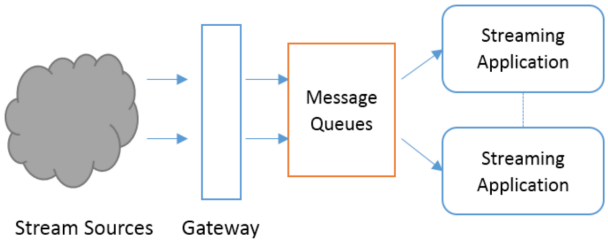
Swing framework [3] addresses the problem of carrying out distributed computation on mobile devices in scenarios when they are remote and dynamic in nature such that devices can join and leave the network sporadically. Furthermore, distributing the computation to mobile devices allows to carry out intensive computation which is overly intensive for a single device. Such remote devices may not have good connectivity to the edge nodes or the cloud. Hence, the need for performing distributed computations locally is critical. We have similar requirements for smart camera systems but unlike mobile devices, smart camera systems are not dynamic and are expected to be part of the network for a long time.

H. The Atos Approach

In this approach [2], authors combined the network and cloud capabilities to create an on-demand, autonomous and decentralized computing system which we call a computing swarm. The processing required for immediate action and extremely low latency happens at the edge of the network, while other processes will run in a range of cloud models. This shall all be executed in an automated, self-organized and self-managed model called swarm computing. The resources here can be any IoT[1] device.

I. Distributed Stream Processing Framework

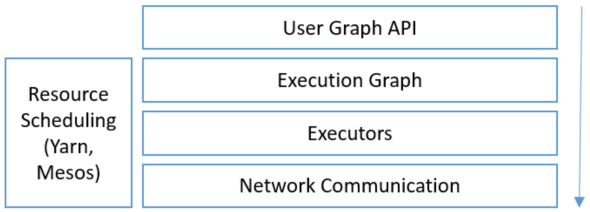
In the below section, we detail a general architecture for DSPF[2] and do a literature review for SEEP [17] - a prominent Distributed Stream Processing System. An SPS[3] contains a programming API for developing the streaming applications,and another execution engine that executes the streaming application. Examples of a few stream processing frameworks are IBM System S, Apache Storm, Twitter's Heron, Neptune, etc.



A streaming source relays heterogeneous stream data sporadically onto a gateway using arbitrary protocols. The gateway, a lightweight middle-ware layer - to reduce overhead and minimize processing and buffering - shall push it onto a message queue which is then consumed by the intended streaming applications.

A layered architecture for a DSPF would look as below:

[1]Internet of Things
[2]Distributed Stream Processing Framework
[3]Stream Processing System

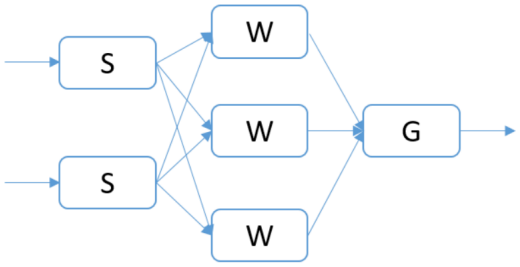


The top layer defines the user APIs where a streaming application is defined as a graph. The edges represent the events flow and the processing happens at the nodes. The user graph is then converted to an execution graph and components of this are distributed to a set of workers/executors across a cluster of nodes. The bottom three layers can work with a resource management layer such as Apache Yarn [18] or Mesos [22] to acquire the necessary cluster resources to run the execution graph.

The role of message brokers in a Stream Processing System is to buffer incoming requests whenever the rate of consumption by the streaming applications is less than the rate of queuing. We can either user transient message brokers such as RabbitMQ [20] and ActiveMQ [33] or storage first message brokers like Kafka [21]. Each of these message brokers is intended for different applications and environments.

1) Stream Processing System Model: A stream s is an infinite series of tuples t belongs to s. A tuple t = (τ, k, p) has a logical timestamp τ, a key field k, and a payload p. The timestamp $\tau \epsilon$ N+ is assigned by a monotonically increasing logical clock of an operator when a tuple is created in a stream. Tuples in a stream are ordered according to their timestamps. Keys are not unique and are used to partition tuples. They can be computed as a hash based on the payload.

2) User Graph and Execution Graph: Usually, the graph defined by the user is converted to an execution graph by the runtime engine. The below figure is a user-defined graph for stream processing with 3 streaming processing operators connected by streams. The second figure below shows the same graph converted into an execution graph where 2 instances of the operator S and 3 instances of operator W are running in parallel. The execution graph is generated from the user graph by DSPF and is different for every DSPF implementation. Fault tolerance, Message delivery guarantees, Throughput vs. latency, Resource management and scheduling of streaming operators, and Communication between the operators are considered while designing an execution graph.



The following table explains how different DSPFs name the components of the graph and how it is represented.

	Storm	Spark	Flink	Samza	Neptune
Graph Node	Spout or Bolt	An operator on a RDD	Operator on a DataStream	Task	Stream Sources and Stream Processors
Graph Edge	Stream	Defined implicitly by the operators on RDDs	Defined implicitly by the operators on Data Stream	Kafka Topic	Links
Graph is directly created by user	Yes	No	No	Yes	Yes
Message abstraction	Tuple	RDD	Data Stream	Envelope	Stream Packet
Primary operator implementation language	Java	Java/Scala	Java/Scala	Java	Java
Name of Graph	Topology	Stream processing job	Stream Processing job	Samza Job	Stream Processing Graph

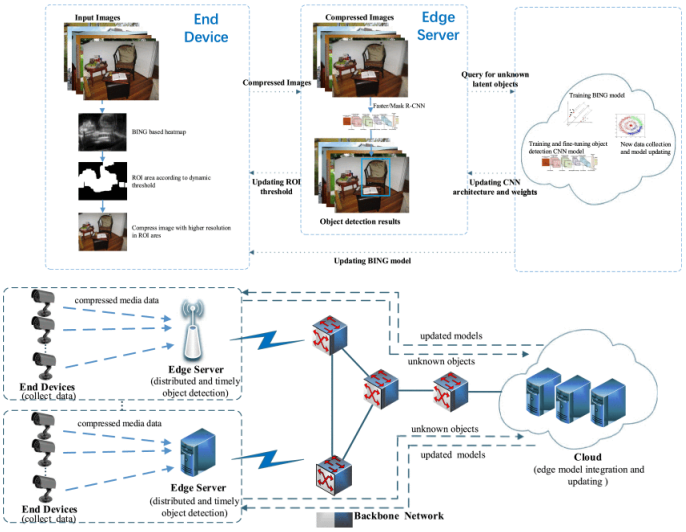
3) Execution Model: The graph defined by the user is converted to an execution graph by the runtime engine. Other than Spark all the DSPF implementations distribute the nodes of the execution graph into machines running in the cluster and run them as continuous operators that react to the events streaming in through the communication links. Usually, a running instance of the execution graph node is called a task. The objective of the underlying engine is to schedule these tasks across a cluster of machines and connect them via network communications. The execution model has three major aspects. 1. Task scheduling 2. Task execution 3. Communication. The runtime engine gets a resource allocation before scheduling the tasks. The resource allocation can be static or dynamic using a separate resource allocator. The most popular resource allocators are Yarn[18] and Mesos[22]. Tasks are instances of user-defined operators that run continuously reacting to events streaming in from the communication links. Most DSPFs use a worker based approach for running

tasks. The following table explains how different DSPFs handle tasks.

	Storm	Spark	Flink	Samza	Neptune
Data serialization	Kryo serialization of Java objects	RDD serialization	Data Stream Serialization	Custom serialization	Java Objects
Task Scheduler	Nimbus, can use resources allocated by Yarn	Mesos, Yarn	Job Manager on top of the resources allocated by Yarn and Mesos	Yarn	Granules
Communication framework	Netty	Netty	Netty	Kafka	Netty
Message Batching for High throughput	Yes	Yes	Yes	Yes	Yes
Flow control	No	Yes	Yes	Yes	Yes
Message delivery	Pull	Pull	Pull	Poll	Pull

J. Distributed and Efficient Object Detection via Interactions Among Devices, Edge, and Cloud[23]

The framework used for object detection in edge computing is distributed over three parts - end devices, edge servers and clouds. The end device layer is responsible for compressing the real-time captured pictures before transmitting them to edge servers. This is done by deploying a lightweight region of interest (RoI) based image compression method on end devices. The RoI is found with a data driven method which is improved from binarized normed gradients (BING[24]). Edge servers work in a distributed manner and have deployed state-of-the-art object detection methods on them. Faster R-CNN[25] method is used in their implementation and experiments. As designing an object detection model for lightweight end devices for extracting feature maps is a challenge, they have used the compressed raw image instead of feature maps for object detection at the edge side. RoI compression means the background of the image is highly compressed and the accuracy of the RoI is kept and this approach was named as BING based RoI Extraction (BRE). The collaborations and interactions among end devices, edge servers and the cloud was designed to dynamically update the local object detection models on edge servers and BRE methods on end devices for detecting new objects. Then, the cloud collects data from the edge servers and incorporate global knowledge to update the model (BRE Model which are trained in the cloud) of the edge servers and end devices.



Looking at the application side, Object detection and Classification being important applications of image analysis, we have studied a few literature papers and summarized ahead. Object detection models need full-size feature maps to evaluate proposals throughout the image, and the sizes of the feature maps among the network layers may be even larger than the original raw image. Designing an object detection model for lightweight end devices for extracting Feature maps are challenging. This was handled by using the compressed raw image instead of feature maps for object detection at the edge side in the edge architecture in the paper.

K. Software Engineering and Distributed Computing in image processing intelligent systems[26]

From this paper, we noted that the results indicate for data management and processing Apache Spark is the most utilized framework. And Among the big cloud platforms, Amazon Web Services is the most widely used in all industry sectors, followed by Google cloud. Also, using Tensorflow (powerful) with Keras is a good combination for building deep learning models from image datasets.

L. Amazon SageMaker

While looking for different approaches or architectures to do computation on end devices, we came across Amazon Sagemaker[27], a machine learning service. SageMaker provides distributed training libraries for data parallelism and model parallelism. The libraries are optimized for the

SageMaker training environment which help adapt your distributed training jobs to SageMaker, and improve training speed and throughput.

Using Spark as a framework allows data transformation and feature engineering workloads to be run at scale within machine learning workflows. Apache Spark as well as other dependencies required to run distributed data processing jobs on Amazon SageMaker are included in a set of prebuilt Docker images provided by Amazon SageMaker. Important features of Amazon SageMaker are to build highly accurate training datasets for machine learning quickly, integrates with AWS Marketplace which enables developers to offer their algorithms and model packages for sale to other Amazon SageMaker users, SageMakerNeo automatically optimizes any trained model built with a popular machine learning framework for the hardware platform that you specify and with no loss in accuracy, RL fully manages your training and prediction infrastructure and Amazon Mechanical Turk provides an on-demand, scalable, human workforce to complete jobs that humans can do better than computers (for example, recognizing objects in photos). In Amazon SageMaker, a library named SageMaker Python SDK[28] can be used to train and deploy models using popular deep learning frameworks and algorithms.

M. Distributed RPC Framework[29]

The distributed RPC framework provides mechanisms for multi-machine model training through a set of primitives to allow for remote communication, and a higher-level API to automatically differentiate models split across several machines. The distributed RPC framework makes it easy to run functions remotely, supports referencing remote objects without copying the real data around.This framework also provides different categories of autograd and optimizer APIs which transparently run backward and update parameters across RPC boundaries. Building a distributed training application using Pytorch is done using torch.distributed package which includes components like distributed data-parallel training, RPC based training and collective communication with the group. But there is no monitoring of the actions or scheduling of messages passed from one model to another. This framework mostly focuses on delegating the amount of work to different machines using APIs so this won't help in our problem statement in full fledged.

N. Real-Time Textured Object Recognition on Distributed Systems[30]

A distributed network architecture can support parallelism and achieve real-time performance, in contrast to current approaches that mostly rely on multiprocessor architectures for quick processing. Keeping the basis of object localization and positioning for image matching, a better solution can be created for best matching between two images. This combination of texture feature extraction and interesting point detection can be implemented on low cost PVM (parallel VM) network to speedup the processing without specific hardware requirements A mask based scholastic method is introduced and summarized in the paper. In paper, a new approach to recognize and localize objects in textured images, which involves dynamic texture feature extraction and parallel implementation of hierarchical image matching was implemented.

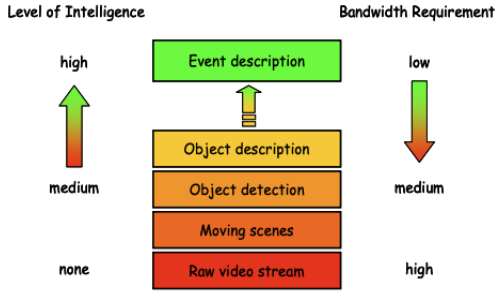
III. Approach

All smart cameras are equal but some smart cameras are more equal.

In the remote surveillance ecosystem, different smart cameras within the network might have different processing requirements at different times and are independent of each other. In a typical scenario, a small group of cameras might detect the object of interest entering its field of view and require more compute resources for further advanced image processing tasks. Such a group of camera nodes can delegate the image processing task to other camera nodes as per their current resource utilization.

For instance in the MeshEye mote architecture [15], as a first level of intelligence, the camera nodes could use a motion detection scheme such that only moving scenes are streamed to the surveillance center. At a second level, the camera nodes could perform object detection and classification such that only moving scenes containing persons or more general objects of interest are forwarded. Going even further, the smart camera nodes could collaborate to identify objects and only transmit their textual description

along with a snapshot. Continuing this train of thought of adding intelligence to surveillance, a network of smart cameras could possibly just notify the surveillance center in case of events of interest by providing a hybrid textual/visual or fully textual description of the event. Figure below illustrates this concept of levels of intelligence in surveillance. As the level of intelligence increases, bandwidth requirements on the underlying data transmission network decrease accordingly.



A. *Design Requirements*

1) Low resource footprints
2) Handle camera device failures
3) Handle streaming data
4) Job scheduling based on heuristics such as resource utilization, data upload rate, historical average rate of job completion, etc.

For cameras to act independently and judiciously delegate processing tasks to other peer camera nodes, each camera should know the resource utilization of its peer nodes. With this information it can schedule a task to peer nodes efficiently. While some of the proposed architecture design in the survey solves distributed scheduling and message passing, they might not be best suited for low resource end-devices such as smart cameras. For instance, deploying Kafka or Spark on camera systems might be an impractical choice due to its high resource footprint [31] [32]. Architecture diagram below shows 4 camera nodes each running the following processes:

1. Tracker

1) Receives and broadcasts heartbeat periodically (configurable).
2) Group membership protocol to manage devices joining or failure.
3) Piggybacks load metrics along with heartbeat message.
4) Maintains resource utilization for each peer.

2. Scheduler

1) Runs as an independent process on each device and is responsible for scheduling job created by its own device only. No global scheduler.
2) Requests Tracker on its local device to get resource availability for all peers.
3) Provides list of peer IPs to Streaming Manager to distribute the data.
4) Scheduling based on: Least Loaded, Good Upload Rate, Avg rate of job completion time for each peer.
5) If the input image buffer is growing at slower rate than image processing rate, then scheduler can decide to not distribute the images to peer and instead process it locally.
6) If peers are unavailable then send processing to the Edge servers.

3. Streaming Manager

1) Responsible for streaming data to the executors and creating a new Job ID (device id, stream seq) and also Reports executor failure to scheduler.
2) Requests executor list from Scheduler before starting a new job or requesting upon executor failure.
3) Maintains a buffer of a few seconds (config) to resend data in case of executor failure.
4) Stops streaming to the executors upon receiving processing completion signal from Action Manager.
5) Source SM can handle out-of-order packets by adopting FIFO ordering with each message having a sequence number. If the packet sequence number (n+1) and the most recent packet in the Source-SM's buffer has sequence number (n), then the packet will be added to the buffer. If the packet sequence number (n+1) and the most recent packet in the Source-SM's buffer does not have sequence number (n), then the Source-SM will put the packet in an alternate buffer until it places a packet with sequence number (n) into the buffer. The Source-SM submits a packet back to the Source-Scheduler in the same order that the Source-Scheduler submitted packets.

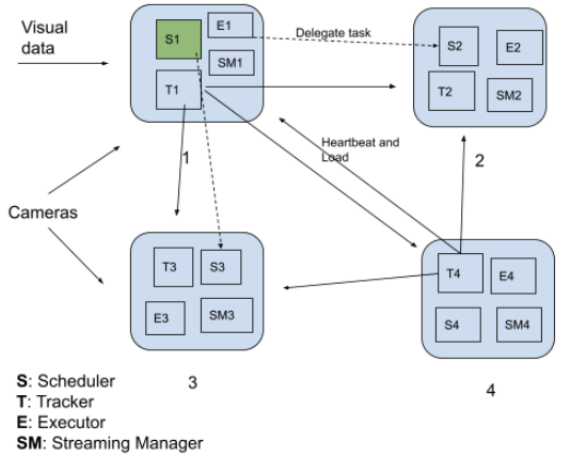
4. Executor

1) Runs as an independent process on each device.
2) Isolated environment for executing jobs.
3) Receives data from Streaming Manager and delivers with Reliable FIFO ordering.
4) Isolation provided by containerization using Linux Container (LXC) or Docker.
5) Well defined execution environment with container image.
6) Sends result back to source Action Manager.

5. Action Manager

1) Runs as an independent process on each device.
2) Receives results from all executors for a given Job ID.
3) Upon receiving majority the executor results, it can take pre-defined actions such as sending PagerDuty alerts or triggering siren alarm.
4) Well-defined action module can support addition of user-defined actions to enable
5) Action Manager support various action types.

The architecture diagram looks as below:



With advancement in compute power for smart cameras, such as ADLINK's NEON-1040 [16] which has quad-core x86 architecture with 1.9 GHz processor, 2GB RAM and upto 32 GB storage. It is feasible to spawn multiple-processes dedicated to managing the distributed system as outlined in the architecture diagram.

IV. Conclusion

In this paper, we formulated and designed a swarm computing framework for image-sensing applications especially in the surveillance domain. These applications are latency sensitive and have a soft real-time nature. For the scope of our project, we have considered low bandwidth, high range and possible high latency networks and smart camera systems that have good computational power. We discussed several approaches from the academia. Approaches such as Cloud-only system and Cloudlets will incur high network latency which is not good for our real time sensing applications. Though Edge Computing solutions are an improvement to the above approach, the network latency here is also too high for a real time application especially under bad network conditions. Among the discussed related work, Swarm[2] is a good option that does the whole processing locally and is a good approach for our problem statement. DSPFs[4] as discussed above are also a good option especially since they can run in low bandwidth network conditions. However, both Swarm and DSPF solutions expect good computing resources for the cluster nodes. Also, formulating our problem statement in a user graph for DSPF is not best modeled for our distributed peer-to-peer network of camera devices. Hence, we couldn't use any of the above approaches directly and have designed a new system model inspired from above literature review.

[4]Distributed Stream Processing Framework

REFERENCES

[1] https://www.wi-fi.org/downloads-registered-guest/Wi-Fi$_{C}ERTIFIED_{H}aLow_{p}aper_{2}$0211102.$pdf$/36881

[2] https://atos.net/wp-content/uploads/2018/12/atos-swarm-computing-white-paper.pdf

[3] S. Fan, T. Salonidis and B. Lee, "Swing: Swarm Computing for Mobile Sensing," 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 2018, pp. 1107-1117, doi: 10.1109/ICDCS.2018.00110.

[4] https://www.fed4fire.eu/wp-content/uploads/sites/10/2019/12/19-10-15-fec6-04-keynote-3-swarm-computing-pages.pdf

[5] https://semiengineering.com/the-future-of-smart-cameras-is-64-bit-processing/

[6] Research Agenda in Cloud Technologies - I Sriram https://arxiv.org/ftp/arxiv/papers/1001/1001.3259.pdf

[7] A. R. Biswas and R. Giaffreda, "IoT and cloud convergence: Opportunities and challenges," 2014 IEEE World Forum on Internet of Things (WF-IoT), 2014, pp. 375-376, doi: 10.1109/WF-IoT.2014.6803194.

[8] M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," in IEEE Pervasive Computing, vol. 8, no. 4, pp. 14-23, Oct.-Dec. 2009, doi: 10.1109/MPRV.2009.82.

[9] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, Sateesh Addepalli Cisco Systems Inc. 170 W Tasman Dr. San Jose, CA 95134, USA https://dl.acm.org/doi/pdf/10.1145/2342509.2342513

[10] COmputer Vision ECCV2006 https://link.springer.com/content/pdf/10.1007

[11] OverLay: Practical Mobile Augmented Reality https://synrg.csl.illinois.edu/papers/overlay.pdf

[12] U. Drolia, K. Guo, J. Tan, R. Gandhi and P. Narasimhan, "Cachier: Edge-Caching for Recognition Applications," 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 276-286, doi: 10.1109/ICDCS.2017.94.

[13] https://www.bluetooth.com/learn-about-bluetooth/recent-enhancements/mesh/

[14] https://en.wikipedia.org/wiki/Low-power$_{w}ide - area_{n}etwork$

[15] Stephan Hengstler, Daniel Prashanth, Sufen Fong, and Hamid Aghajan. 2007. MeshEye: a hybrid-resolution smart camera mote for applications in distributed intelligent surveillance. In Proceedings of the 6th international conference on Information processing in sensor networks (IPSN '07). Association for Computing Machinery, New York, NY, USA, 360–369. DOI:https://doi.org/10.1145/1236360.1236406

[16] ADLINK: Breaking the boundaries of smart cameras and embedded vision systems https://www.imveurope.com/sites/default/files/Smart

[17] Raul Castro Fernandez, Matteo Migliavacca, Evangelia Kalyvianaki, and Peter Pietzuch. 2013. Integrating scale out and fault tolerance in stream processing using operator state management. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13). Association for Computing Machinery, New York, NY, USA, 725–736. DOI:https://doi.org/10.1145/2463676.2465282 https://raulcastrofernandez.com/papers/sigmod13-seep.pdf

[18] Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S.R., Reed, B.C., Baldeschwieler, E. (2013). Apache Hadoop YARN: yet another resource negotiator. Proceedings of the 4th annual Symposium on Cloud Computing.

[19] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: a platform for fine-grained resource sharing in the data center. In ¡i¿Proceedings of the 8th USENIX conference on Networked systems design and implementation¡/i¿ (¡i¿NSDI'11¡/i¿). USENIX Association, USA, 295–308.

[20] A. Videla and J. J. Williams, RabbitMQ in action: Manning, 2012.

[21] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in Proceedings of the NetDB, 2011.

[22] A. Videla and J. J. Williams, RabbitMQ in action: Manning, 2012.

[23] Y. Guo, B. Zou, J. Ren, Q. Liu, D. Zhang and Y. Zhang, "Distributed and Efficient Object Detection via Interactions Among Devices, Edge, and Cloud," in IEEE Transactions on Multimedia, vol. 21, no. 11, pp. 2903-2915, Nov. 2019, doi: 10.1109/TMM.2019.2912703.

[24] M.-M. Cheng, Z. Zhang,W.-Y. Lin, and P. Torr, "Bing: Binarized normed gradients for objectness estimation at 300fps," in IEEE Conf. Comput. Vis Pattern Recognit., 2014, pp. 3286–3293.

[25] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detectionwith region proposal networks," IEEE Trans.PatternAnal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[26] https://laccei.org/LACCEI2021-VirtualEdition/full$_{p}apers/FP175.pdf$

[27] https://docs.aws.amazon.com/sagemaker/index.html

[28] https://sagemaker-examples.readthedocs.io/en/latest/sagemake_processing/sparkprocessing.html

[29] https://pytorch.org/docs/stable/rpc.html

[30] https://www.academia.edu/71060947/Real_Time_Textured_Object_Recogniti

[31] https://docs.confluent.io/platform/current/kafka/deployment.html

[32] https://spark.apache.org/docs/latest/hardware-provisioning.html

[33] https://activemq.apache.org/