Future Generation Computer Systems 102 (2020) 514–523

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

π-Hub: Large-scale video learning, storage, and retrieval on heterogeneous hardware platforms☆

Jie Tang [a,*], Shaoshan Liu [b], Jie Cao [c], Dawei Sun [b,d], Bolin Ding [e], Jean-Luc Gaudiot [f], Weisong Shi [c]

[a] South China University of Technology, Guangzhou, PR China
[b] PerceptIn, United States of America
[c] Wayne State University, Detroit, United States of America
[d] TSingHua University, Beijing, PR China
[e] Alibaba DAIL Lab, United States of America
[f] University of California, Irvine, United States of America

ARTICLE INFO

Article history:
Received 31 May 2019
Accepted 5 August 2019
Available online 28 August 2019

Keywords:
Robotic cloud
Video retrieval
Internet of Things
Heterogeneous platform

ABSTRACT

The burgeoning of Internet of Things (IoT) and camera-equipped mobile devices contributes a tremendous amount of video data generated at the edge of the network. At the same time, we have witnessed the fast deployment of many video-based application services, such as plate recognition for public safety, intelligent transportation, Industry 4.0 and so on. The success of these services, in turn, requires large-scale video data being learned, stored, and retrieved in a more efficient way. A generic software and hardware framework for large-scale IoT video analysis and service support is still missing. To address this challenge, we present π-Hub, PerceptIn's robotic cloud solution which supports large-scale video data analysis, storage, and query by implementing the learn-store-retrieve paradigm. Interestingly, we found that among the learning, storage, and retrieval services each of them stresses one type of resources on heterogeneous computing servers, i.e., GPU, CPU, and Memory, respectively, therefore it is extremely cost-efficient to co-locate these services together to fully utilize the resources. In addition, several optimization techniques for data writing, reading, and data reduction are proposed and evaluated. The evaluation results show that these techniques improve the performance of the learning, storage and retrieval services significantly as well as notably reduce the cost of the system. We also verify π-Hub's scalability by reliably running a 1000-machine deployment to support up to one million users. Finally, we conclude the paper by discussing several lessons learned from this study and future work.

1. Introduction

The rise of IoT applications has imposed tremendous pressure on our existing cloud infrastructure [1]. For instance, every day, a mobile phone sends out at least 30 MB of data to the clouds of the service provider, the application operators, etc. In contrast, even a very simple robot can easily generate over 1 GB of video data per day. An extreme form of robot, driver-less cars, can generate as much as 2 GB of video data per second [2]. The pervasive deployment of video surveillance cameras in major cities also collect a tremendous amount of video data for traffic management, crime control, and public services, to name a few [3–6]. For instance, in a real-world scenario, in-city service robots may act in a surveillance role, patrolling residential area and recording captured videos. Then, residents and citizen demand the capability of video playback based on intelligent queries using time, location, as well as objects in the scene as inputs. Several new challenges raised for video analysis and streaming systems in this typical use case

Many research efforts of IoT video analysis have been contributed to the community [7–13]. The task for analyzing, storage, and service query of the video collected from a large scale of cameras and robots could be very resource hungry [14–17]. In video network systems the performance requirements are multi-dimensional and diverse notably. Traffic control applications usually require long time data storage [18–20]. However, AMBER Alerts desire a large coverage area or a large scale video source [21]. Autonomous vehicles have need for real-time video

☆ This work is supported by Guangzhou Technology Grant No. 201707010148, Guangdong NSF Grant No. 2018A030310408, and GuangDong Rearch and Development key Project of China Grant No.2018B010107003.

* Corresponding author.
E-mail address: cstangjie@scut.edu.cn (J. Tang).

https://doi.org/10.1016/j.future.2019.08.006
0167-739X/

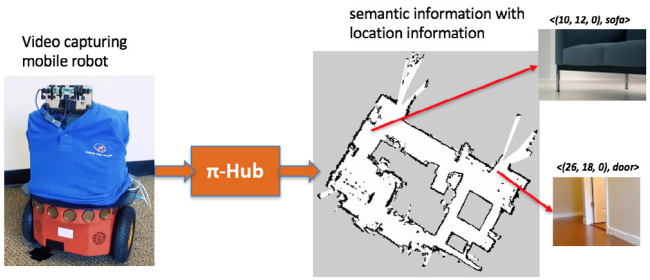


Fig. 1. A typical use case of π-Hub in home environment.

processing [22–24]. Public service systems for video streaming usually stress from the large number of queries [25,26]. In order to meet the specific performance requirements, heterogeneous resources including network bandwidth, GPU, CPU, and storage are usually allocated for traditional video analysis applications in the cloud [27]. In [28], researchers present a Cloud-based system to share the video data among various applications.

To the best of our knowledge, a generic cloud infrastructure which can process and store video collected from multiple sources, in the meanwhile support a large scale of queries simultaneously is still missing [29]. Moreover, to implement and deploy this infrastructure, efficient resource allocation mechanism is another challenge. To efficiently process and store the massive video data, we present π-Hub, PerceptIn's robotic cloud solution for large-scale video learning, storage, and retrieval with efficient resource utilization on heterogeneous hardware platforms [30]. π-Hub supports the learning, storage, and retrieval services, where learning is about how to automatically understand the video data and convert it into structured information; storage is about how to effectively store the massive amount of video data; while retrieval is about how to efficiently retrieve the data when needed.

Fig. 1 illustrates a typical scenario of using π-Hub in a residential environment. In this case, a robot captures a video while generating the map of the environment and sends the video to the cloud. Then π-Hub would be able to extract semantic information from the video, in this case, it extracts the objects sofa and door from the video and associate location information ($\langle x, y\rangle$coordinates) with the detected labels.

The performance evaluation of π-Hub shows that it can co-locate the learning, storage and retrieval services on the same server with full utilization of CPU, GPU, memory, and disk resources. Also π-Hub can be scaled up to support one million users. To further optimize the performance of π-Hub, we propose and evaluate several techniques in data writing, reading, and data reduction.

The contributions of this work are as follows:

(1) We designed and built π-Hub, which is a system for large-scale video data learning, storage, and retrieval;
(2) π-Hub can be deployed on one server and fully utilize its heterogeneous hardware resources, such as CPU, GPU, memory, and disk;
(3) We run a series of stress test to evaluate the performance of π-Hub comprehensively, results showed that π-Hub can be scaled up to support 1 million users reliably on a 1000-machine cluster.
(4) We introduced several approaches to improve the efficiency of data reading and writing. These approaches optimized the performance of π-Hub's storage layer and can also be applied to other IoT systems.

The rest of this paper is organized as follows. In Section 2, we introduce the design of π-Hub, which implements the learn-store-retrieve paradigm, and discusses how π-Hub can meet the challenges in data understating, storing, and retrieving. Section 3 evaluates the performance of π-Hub from aspects of data processing, storage, and network. We also validate the scalability of π-Hub in Section 3. Section 4 introduces the implementation and evaluation of several techniques for performance optimization. In Section 5, we discussed multi-layer deployment of π-Hub as a solution for hybrid Cloud-Edge analytics in the future. Finally, the paper concludes in Section 6 .

2. System description

In this section, we present high-level architecture of π-Hub and discuss the learning, storage, and retrieval services in π-Hub.

2.1. System requirements

Before introduce the architecture of π-Hub, we would like to list the requirements of the system first.

(1) *Large-scale video processing*. As we have discussed above, we would like to deploy π-Hub to handle massive IoT video data. Specifically, we want an individual π-Hub instance on an ordinary server machine be able to learn and store video data from 10 robots as well as simultaneous video retrieval request from 100 users. Moreover, to reach the goal of 1 million user support, π-Hub is also required to be robustly running on a 1000-machine cluster.
(2) *Response time*. For the learning service, we target to have each deployment of π-Hub supporting at least 10 robots, and each robot streams images once every two seconds, this means that each second π-Hub performs learning on at least five images, or having a processing time of less than 0.2 s per image. For the storage service, we require an average write throughput of at least 100 MB/s to make sure that the multimedia data gets stored. For the retrieval service, to provide good user experiences, we require at least 90% of all queries to be completed within 30 s.
(3) *Cost effectiveness*. The deployment and maintenance of a large-scale server cluster is high-priced. To control the cost of π-Hub and support as many video producers/consumers as possible, π-Hub is required to fully utilize all the available computing resource including CPU, GPU, memory, disk, and network.

2.2. π-Hub architecture

Fig. 2 shows the architecture of π-Hub. From the service point of view, π-Hub supports three services, which are learning, storage, and retrieval. From the system point of view, π-Hub consists of the following components:

- Client Devices: these devices capture multimedia feeds and send the feeds to the cloud along with their meta-data.
- Streaming Server: it handles multimedia and streams on-demand live multimedia feed to users as requested.
- Object Recognition: deep-learning evaluation engine for automatic extraction of semantic information from incoming videos.
- Key–Value Store: this key–value store organizes the video feeds along with the learned/extracted semantic information.
- Query Engine: this query engine supports retrieval of video feeds. One can search using any combination of time, location, as well as extracted labels.

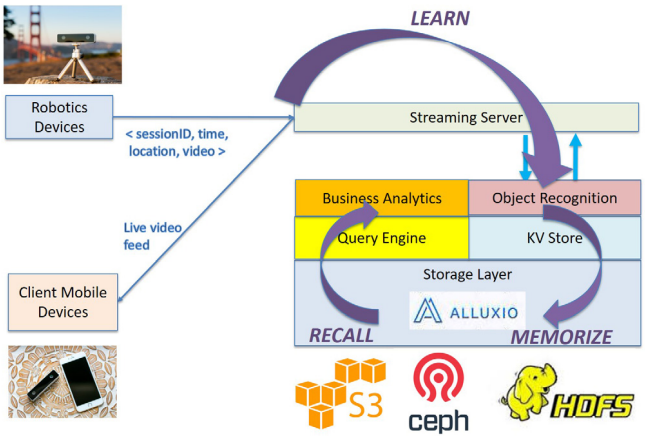
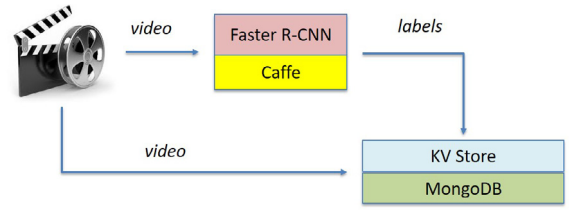


Fig. 2. The π-Hub system architecture.



Schema *<user ID, timestamp, duration, location, <list of labels>, path to video file>*

Fig. 3. Video stream processing.

- Business Analytics Engine: this engine generates high-level statistics of all multimedia data. For example, one can be interested in knowing which are the most common objects appear in living rooms, etc.
- Storage Layer: the storage layer needs to provide high throughput for data persistence and low latency for fast retrieval of video feeds. Also, it must manage heterogeneous storage systems including S3, GCS, Swift, HDFS, OSS, GlusterFS, and NFS.

2.3. Learning service

As multimedia data comes into the cloud system, the first task is to learn from the raw data and to extract semantic information out of the multimedia data. For video streams, we can extract object labels from frames and associate these labels with the video stream. For audio streams, we can extract sentences of the spoken language. Then this semantic information can be used as keys, and the raw data streams can be used as values, and together they are stored in the key–value store.

As presented in Fig. 3, we need a learning engine that extracts object labels from a video stream. This engine needs to be accurate in terms of recognition rate, and this engine needs to be fast in order for us to capture as many objects as possible in the video. Therefore, in this implementation, we utilize faster r-cnn [31] network running on Caffe [32]. Faster r-cnn introduces a Region Proposal Network (RPN), a fully convolutional network that simultaneously predicts.

2.4. Storage service

After extracting the semantic information from the raw multimedia data, the extracted labels, along with the raw data get stored in the key–value store for easy retrieval, as shown in Fig. 4. We implemented the key–value store using MongoDB [33]. In this case, the key is the meta-data including $\langle sessionID, timestamp, duration, location, \langle list\ of\ labels \rangle \rangle$, and the value is a file path of the raw data in the storage layer.

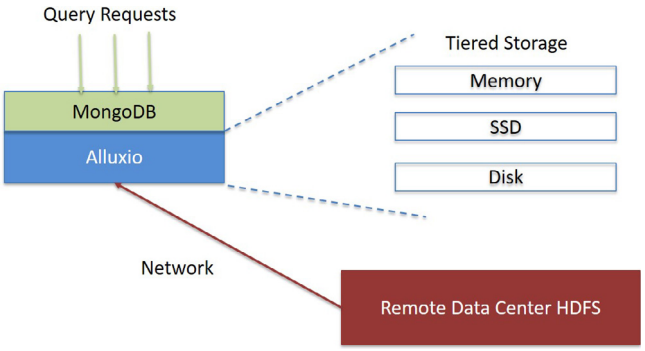


Fig. 4. Storage layer architecture.

This poses two challenges: first, in real-world scenarios, users vary in their choice of persistent storage for their data. Some prefer Amazon S3, some their own deployment of HDFS, others Ceph, etc. One way to get around this problem is to create a set of APIs for each persistent storage, but this would become impossible to manage as the number of persistent storage options grows. A second, and probably better way, to handle this is to create a unified storage layer to abstract all underlying storages. To this end, Alluxio enables effective data management across different storage systems through its use of transparent naming and mounting API [34]. Transparent naming maintains an identity between the Alluxio namespace and the underlying storage system namespace. When users create objects in the Alluxio namespace, they can decide whether these objects should be persisted in the underlying storage system. For objects that are persisted, Alluxio preserves the object paths, relative to the underlying storage system directory in which Alluxio objects are stored. With this feature, we can now manage multiple persistent storages using a single set of storage layer API, which greatly simplifies the management of the storage part of the π-Hub computing paradigm. The second challenge is that the write throughput directly impacts the performance of the whole system. If the write speed of the storage is slower than the detection speed in the learning stage, then it becomes the bottleneck and leads to "memory loss". We will discuss in the next section how we can use Alluxio's tiered storage feature, along with write optimization to improve write throughput.

2.5. Retrieval service

The last stage is retrieve, as shown in Fig. 5. By Query semantic process, users can use any combination of meta data, such as time, location, or detected labels to accurately retrieve the target multimedia data. In our implementation, we use MongoDB to perform the intelligent search using the meta-data as keys to first retrieve a file path of the storage layer. Then we issue another request to retrieve the raw data using the file path.

Therefore, the performance of the retrieve stage greatly depends on the read performance of the storage layer. It is helpful to exploit locality of the storage layer to improve read performance. Ideally, the performance is highest if the requested data is located in the memory of the local machine. However, it is impossible to store all data in the memory of the local machines. We need a mechanism to provide a cache-like structure such that we have different levels of the storage at a different speed, including local

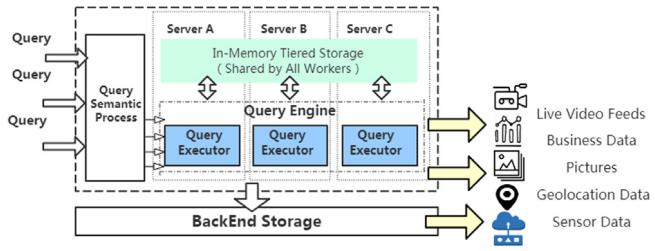


Fig. 5. Retrieve architecture.

memory, local Solid State Drives (SSD), local Hard Disk Drives (HDD), and remote storage.

This requirement can be fulfilled by Alluxio's tiered storage feature. With tiered storage, Alluxio can manage multiple storage layers including Memory, SSD, and HDD. Using tiered storage Alluxio can store more data in the system at the same time, since memory capacity may be limited in some deployments. With tiered storage, Alluxio automatically manages blocks between all the configured tiers, so users and administrators do not have to manually manage the locations of the data. Users may specify their own data management strategies by implementing allocators and evictors.

In a way, the Memory layer of the tiered storage serves as the top level cache, SSD serves as the second level cache, HDD serves as the third level cache, while it is the persistent storage is the last level storage. We will discuss Alluxio's tiered storage along with prefetching optimization later.

3. Performance evaluation

In this section, we delve into the details of the performance and scalability of the implementation of π-Hub. The machine configuration used in this implementation consists of an Intel Core-i7 CPU running at 3 GHz, and a Titan X GPU with 12 GB of graphics memory, and 32 GB of system memory.

3.1. Object recognition performance

We look at the performance of the object recognition engine. We stress the engine by feeding it video streams and measure its performance. As shown in Fig. 6, when under stress, on a single server, it takes an average of 0.16 s to process an image. Also, this workload is GPU-bound, as it uses 95% of GPU resources, but only 23% of CPU resources and 3% of system memory.

In our real-world use case, the in-home service robots move at a speed of about 30 centimeters per second. Therefore, in most cases, we only need to extract labels from an image every two seconds without missing an object in the scene. This implies that with each server, we could support 10 simultaneous incoming video streams. This can be further tuned based on the robot speed and user requirements.

3.2. Query engine performance

We now examine the performance of the query engine. We stress the engine by launching 100 clients repeated send queries to the query engine and we measure the response time. As shown in Fig. 7, when under stress, on a single server, it takes on average of 4 ms to process a query. This workload is CPU-bound, in that it uses 98% of CPU resources, none of the GPU resources, and 2% of system memory. This confirms that with one server, we could easily handle over 100 users simultaneously.

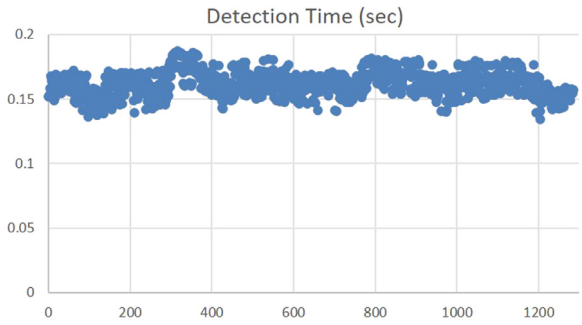


Fig. 6. Faster RCNN detection stress test.

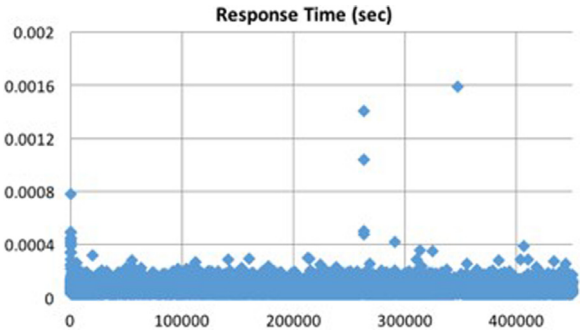


Fig. 7. MongoDB query stress test.

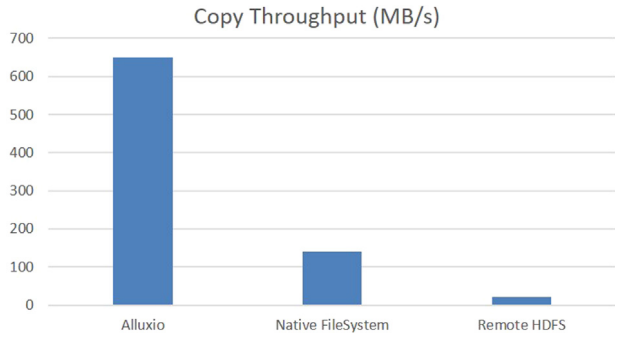


Fig. 8. Alluxio stress test.

3.3. Storage performance

Then we examine the performance of the storage engine. We first compare the throughput of a copy operation with Alluxio, with the Native File System, and with Remote HDFS. This parameter is critical as it determines how fast we can write a video feed to storage. If the throughput is too low, then the storage layer may become the bottleneck of the whole multimedia data pipeline. As shown in Fig. 8, with Alluxio's in-memory storage engine, we could easily achieve greater than 650 MB/s throughput whereas, with Native File System, we could only achieve 120 MB/s. Using remote HDFS, the performance is the worst, only sustaining less than 20 MB/s throughput. This result indicates that for the storage engine not to become the bottleneck of the storage part, it is important to keep most of the write operations to hit the in-memory storage layer.

Then we evaluate the video retrieval latency, using Alluxio's in-memory storage engine, we are able to retrieve a video within 500 ms. However, when the video is stored on remote machines, the latency can be as high as 20 s. Therefore, using Alluxio to buffer "hot" video data could reduce retrieval latencies by as

Table 1
Resource utilization.

	CPU%	GPU%	MEM%
Faster RCNN	3	95	3
MongoDB stress	98	0	2
Alluxio stress	3	0	95

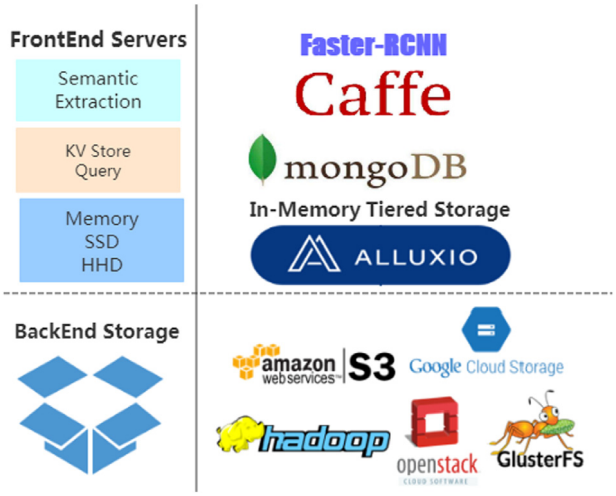


Fig. 9. Deployment architecture.

much as 40 folds, which is quite critical to the user experience, especially in the retrieve stage.

3.4. Deployment

After understanding the performance of the individual components, we take a look at the deployment setup as well as the scalability of this architecture.

First, Table 1 summarizes the resource utilization of the three components under stress. It is interesting that each component stresses one type of system resource: the faster R-CNN recognition engine stresses the GPU, MongoDB stresses the CPU, while Alluxio stresses the system memory. This indicates that it is best to co-locate these three services on the same servers. This approach provides the following benefits:

- Cost Efficiency: as we co-locate the services, we reduce costs.
- Performance: the storage layer, Alluxio, is now co-located with the same servers as the learning engine and the query engine, which ensures high write throughput and low read latency.

The deployment architecture is shown in Fig. 9. It is divided into two parts, the front-end servers and the back-end storage servers. The frontend servers process the incoming data and write the raw data in Alluxio. Alluxio then asynchronously persists the raw data in the backend storage servers. Also, when the users generate new queries, the frontend servers process the queries and retrieve the raw data from Alluxio. If the requested data is buffered in Alluxio's tiered storage, it is immediately returned to the user. Otherwise, it is a cache miss and Alluxio requests the data from the backend storage servers.

For Alluxio, we use a two-level deployment, allocating 20 GB of memory in the top tier and 200 GB of HDD in the second tiered. With a 10-machine deployment, we provide 2.2 TB of cache space. Assuming 10 MB average video file size, the system can buffer around 200,000 video files. Also, with ten machines, we can simultaneously support 100 video streams as well as 1000 simultaneous queries. From our experiences, at any moment in time, at most 10% of all users will be active, therefore with this setup, we are able to handle 10,000 users.

3.5. Scalability

Next, we seek to answer the question as to whether we could support 1 million users. As mentioned above, with 10 machines, we can support 10,000 users, therefore, whether we can support 1 million users is really a question of whether we can have a stable 1000-machine deployment. To verify this, we deployed a 1000-machine Alluxio cluster and conducted a stress test using a background script to repeatedly write to and read from the system. As shown in Fig. 10a, the stress test consists of three items:

- The block read test repeatedly reads a block from the storage system, and thus repeatedly stress one storage node.
- The block write test repeatedly writes a block to the storage system, therefore, not only stress test the storage nodes but also the underlying storage as well (when the storage nodes are full). Note that for the block read test and block write test throughput and latency numbers, we have reported those in the deployment section and will provide more details in the Optimization section.
- The master stress test, in this test we start 100 threads to perform small file reads/writes simultaneously, and the purpose is to try to crash the master node, and therefore bringing down the whole system. A snapshot of the test results is shown in Fig. 10b, indicating with 100 threads simultaneously stressing the master, all requests were completed within 30 s.

Note that we repeatedly ran these test for two weeks, and within the two weeks we did not observe any master node crash or storage node crash, and therefore we were able to confirm that Alluxio could reliably scale to 1000 instances. This confirms that we can use Alluxio to support one million users as our implementation scales.

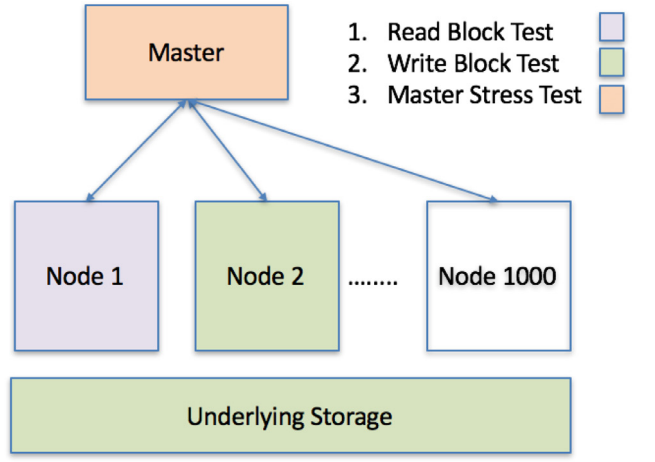
4. Optimization

As mentioned in Section 2, the storage layer write throughput is critical to the performance of the storage stage and the storage layer read latency is critical to the performance of the retrieve stage. In this section, we delve into the optimizations we implement in the storage layer to improve both write throughput and read latency.

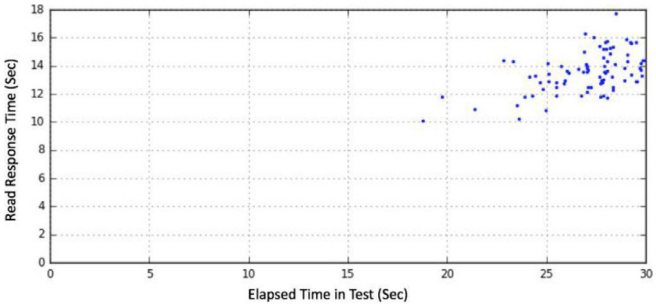
4.1. Write optimization

With the default Alluxio tiered storage implementation, when a user writes a new block, it is written to the top tier by default, as shown in Fig. 11a. If there is not enough space for the block in the top tier, the system checks for free space in the next layer. This gets repeated for each layer until free space is found. For instance, if both the memory layer and the SSD layer are full, free space may be found in the HDD layer. Then, the evictor moves a block in the SSD layer to the HDD layer and then moves a block in the memory layer to the SSD layer. Finally, the block can be written back to the memory layer.

This approach is inefficient when the tiered storage is fully occupied as each time a new block is written, the evictor needs to move blocks across all layers before the new block can be written. This greatly reduces the write throughput as it takes a long time to write each block in the storage layer.



(a) Test design.



(b) Test result.

Fig. 10. Stress test of π-Hub on 1000 machines.

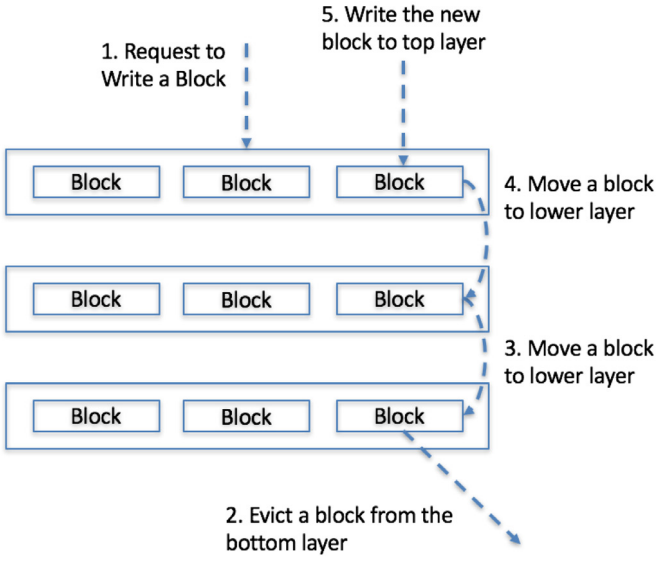
To allay this problem, we implement a Direct-Write allocator, such that it writes a block to the first layer that has free space, as shown in Fig. 11b. When we run a stress test to continually write blocks into Alluxio, compared to the default allocator, our Direct-Write approach reduces the write latency by half and consequently doubling the write throughput.

One could argue that the Write-Direct approach impacts read performance since the new block is now not in the top tier. However, in our experiences, it is very rare that a block gets read right after being written to storage. Also, if the block gets read, say ten minutes later, by that time, it is highly likely that the block has already been evicted to lower level of the storage layer.

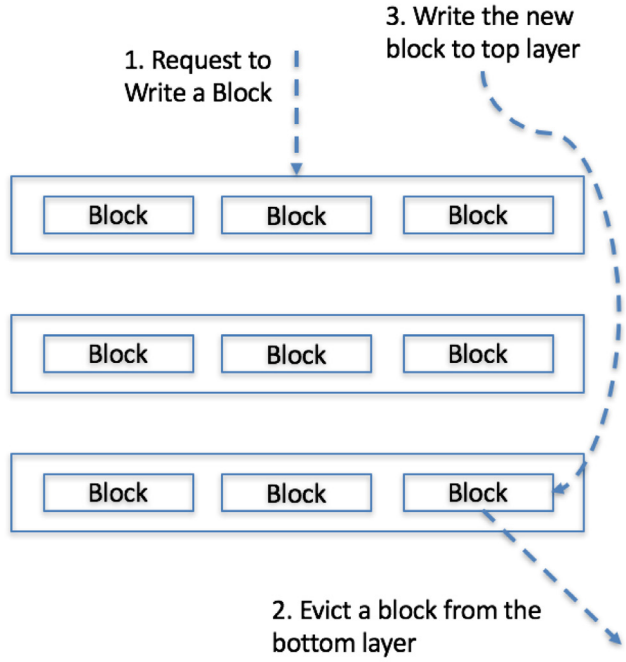
4.2. Read optimization

To optimize read performance, we strive to keep as much data in the frontend servers as possible, thus avoiding the latency of fetching data blocks from remote backend storage servers. We have collected over six months of text-based query data from over 1000 users. We discovered that, without any optimization, we reached a hit rate of about 50%, meaning that 50% of the queries are satisfied the frontend servers without hitting the remote backend storage servers, while the other 50% end up hitting the remote servers, thus leading to high latency.

An effective approach to improve this situation is to use prefetching [35]. A simple improvement, when the frontend servers are less busy, we prefetch data from the most requested table into the frontend servers. This simple improvement immediately boosts the hit rate to 70%. For the next step, we break the day into 6 time periods, each is four-hour long. For each time period, we prefetch the most requested table from the same



(a) Default allocator.



(b) Direct-Write allocator.

Fig. 11. Write optimization in π-Hub.

period in the previous day. By doing this, we boost the hit rate to 80%, as shown in Fig. 12.

This verifies that prefetching is indeed a very effective technique for improving read latency. As we get more queries for multimedia data in our storage, we plan to design more fine-grained prefetching techniques to further improve the performance of reading operations. Some of the prefetching strategies we plan to use include:

- Label: since we now have label information associated with each video stream, and we observe an initial trend that people tend to search for certain labels, such as a dog. Prefetching the videos with the "hottest" label will be an effective way to improve read performance.

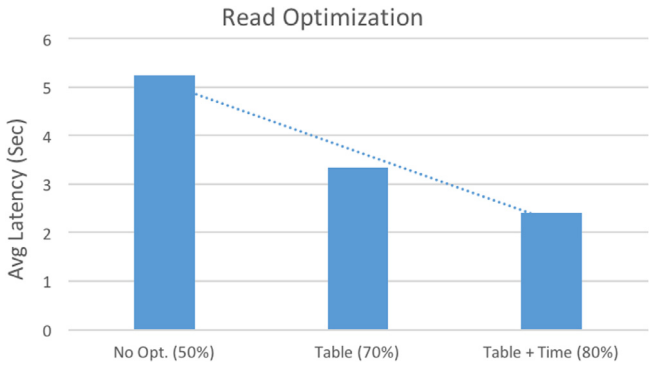


Fig. 12. Read optimization in π-Hub.

- Location: also, now that each video stream is associated with location information, such as bedroom, living room, etc., it will also be effective to prefetch the most searched locations into the frontend storage servers. We will explore these options in the next step.

4.3. Data reduction

As mentioned in the introduction, the volumes of data collected from robotic devices can easily grow exponentially over time, which on one hand, enables our π-Hub to provide data-analytics and decision-support services to users, but on the other hand, imposes performance challenges on all the three stages, learn-store-retrieve, in terms of both storage and query response time. Although the architecture presented in this paper is optimized for robotic workloads, with a fixed and limited budget of computing resources, the users eventually have to discard more and more data and tolerate longer response time.

4.3.1. Data reduction techniques

Data reduction techniques, e.g., sampling, can be a feasible solution to handle the above issue on π-Hub, especially when small errors are tolerable. For example, when the data collected from robots is used to estimate traffic volume/speed or to learn trajectory patterns, small errors in the results are usually inevitable anyway due to the noise in the initial data collection and in the learning stages. For such tasks, the goal of our π-Hub is to provide data-analytics services with low storage cost and interactive query response time at the cost of small errors in the analytical results and answers.

To achieve this, we utilize techniques from approximate query processing (AQP), which has been studied for decades mostly in the context of relational databases. The goal of AQP is to provide approximate answers to a subclass of SQL queries with interactive response time and estimated error bounds. This line of work has become very active recently because of the explosion of data. The recent efforts result in both innovative techniques and more mature sampling-based systems, e.g., BlinkDB [36], QuickR, and Sample+Seek.

Interested readers can refer to for a brief survey of the progress. Sampling-based AQP techniques can be characterized by where the sampler (for the data-reduction purpose) is in the whole pipeline. In one way, the sampler can be placed right before the data is streamed into the query execution engine, or even pushed to the nodes of a query plan tree with the execution engine (e.g., QuickR). In the other way, samplers are used pre-computed samples of the data tables, and SQL queries are rewritten to be executed on these sample tables for fast response (e.g., BlinkDB and Sample+Seek).

4.3.2. Usages in π-hub

In a π-Hub, we have more options about where to place the samplers and more considerations to be addressed for each option. Following are our preliminary proposals.

- Sampling before learning. We can reduce the amount of raw data collected from users by placing samplers in the client devices or the streaming server. A tuple (e.g., an image) in the raw data is selected into the object recognition component only with some probability, which is determined by some light-weight calculation based on the tuple's meta-data, e.g., time and location.
- Sampling before storage. After the semantic information is extracted from the raw data tuples, we can pre-compute and store only a subsample into the key–value store. For example, a row $\langle sessionID, timestamp, duration, location, \langle list\ of\ labels \rangle \rangle$ is selected into the store with a probability determined by the "list of labels"—we can select more sample rows for some hot and more important labels and fewer sample rows for the others.
- Online sampling during retrieval. When a query is issued in the retrieve stage, we can invoke a sampler to select only a subset of rows from the key–value store into the query execution engine (and of course, rewrite the query to rescale the answer). For example, we may want to count the number of times a label appears between a time interval. A number of samplers can be borrowed from AQP systems to provide estimates of answers and error bounds for such queries.

In our experiences, after reaching a significant scale (e.g. one million users), data reduction sampling techniques are able to reduce 90% of data processing. As discussed in Section 2, with 1000-node deployment, the proposed architecture can support up to one million users; also, from our experiences, the operation cost of each server per year is around $3000. In Table 2, we summarize the estimations of the cost reduction and performance gain of deploying the sampling techniques. When sampling happens before the learning service, we reduce the whole data processing pipeline's workload by 90%, and thus we can reduce 90% of the deployed servers, leading to an annual saving of 2.7 million USD, and we boost the performance of all stages by 10X. When sampling happens before the storage service, we cannot reduce the number of servers needed with the current server configuration, but we boost the storage stage and the retrieve stage by 10X. Lastly, when sampling happens in the retrieve stage, only the retrieve performance will be boosted by 10X. The observation from this is that the early we apply data reduction techniques, the more we can reduce operation cost and boost performance. As a next step, we will study how data reduction techniques can be applied at the computing edges to further reduce cost and improve performance, more will be discussed in the future works section.

4.3.3. Property of data reduction schema

Fig. 13 shows a simple data reduction schema that can be used in robotic clouds. Offline samples are drawn from KV Store and are maintained in a separate in-memory store so that query engine can efficiently retrieve them to process analytical queries. When a query has a very selective predicate, e.g., to select data with an extremely rare label or during a short time period, offline samples can become useless (because very few tuples satisfying the predicate will appear in the offline samples). Therefore, we also need an online sampler to aid the query engine, to draw a sample in an online fashion for a given analytical query with selective predicates. For both cases, in order to achieve interactive

Table 2
Cost reduction and performance gain of the sampling techniques.

	Cost reduction (thousand $)	Performance gain in learning	Performance gain in storage	Performance gain in retrieve
Sampling before learning	2700	×10	×10	×10
Sampling before storage	0	0	×10	×10
Online sampling during retrieve	0	0	0	0

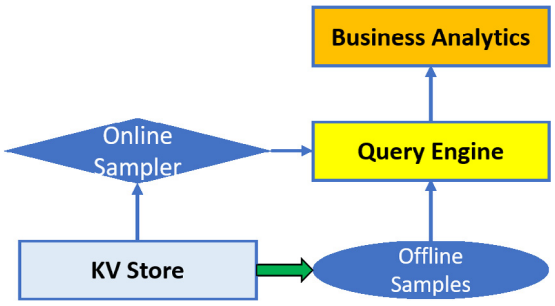


Fig. 13. Data reduction schema and property.

response time, the query engine takes samples of tuples with size only sublinear in the store size n, e.g., $O(\sqrt{n})$.

On the other hand, it is important to note that we always want to guarantee that the errors are no more than some threshold in our analytical results or answers, or at least, we want to provide estimates of the errors. Such guarantees or estimations are relatively easier for SQL-like aggregation queries (e.g., counting the number of cars on a street). For example, for both online sampler and offline samples, we can use measure-biased sampling and priority sampling to processing Group-By queries and Subset Sum queries, respectively, on only $O\left(1/e^2\right)$ sample tuples to obtain e-approximations to the query answers. It is, however, more challenging for complex tasks (e.g., predicting the trajectory of a car) to achieve the above precision guarantees.

We will start with adopting the sampling-based data reduction techniques in our robotic cloud to provide cheap and fast data-analytics services for users with relatively simpler analytical workloads, and extend our services for more complex tasks (e.g., building ML models) in future.

5. Lesson learned and future work

In this section, we first discuss the lessons learned in this work, followed by the description of potential future direction.

5.1. Lessons learned

There are several lessons learned from this study:

(1) *Co-location of services lead to cost-efficiency and performance gain*: with heterogeneous computing architectures, we can co-locate different services, each stresses one type of computing resource, this greatly reduces the number of servers required, leading to cost-efficiency. In addition, by co-locating services, the system has fewer data to move over the network, and thus less prone to network instabilities. Based on this observation, we plan to deploy more heterogeneous computing servers, one ongoing work is to add FPGA to our server architecture to handle the robot trajectory generation tasks.

(2) *Unified storage architecture with caching capability is imperative*: when there is a huge influx of different types of multimedia data, heterogeneous backend storage infrastructures may be necessary, but managing heterogeneous backend infrastructures often incur significant RD and maintenance overheads, a unified storage interface that shields the details of the heterogeneous storage backend infrastructures from the users and system administrators is key to scalability. In addition, the communications between the frontend services and the backend storage infrastructure are prone to network instabilities. A caching layer in the unified storage interface (e.g. delayed write and buffering of hot datasets) has proven to be a great solution to this problem.

(3) *Data reduction is an effective means for performance and cost optimization*: we have developed effective mechanisms to reduce the amount of data before calling the learning service, before calling the storage service, as well as online sampling during retrieval. These mechanisms all lead to significant reduction of required computing and storage resources, as well as performance gains. Specifically, the impact is greater when data reduction techniques are applied early in the whole data processing pipeline. Therefore, we are currently working on designing and implementing a hybrid edge-cloud robotic data analytic architecture, by moving more data processing capabilities to the edge, we can use a much thinner and cost-effective cloud infrastructure for performing the learn-store-retrieve services.

5.2. Future work

Edge computing as a new computing paradigm has been pushed by the burgeoning of the Internet of Things and the success of rich cloud services. With the increasing number of devices added to the Edge of the network, massive video data will be collected and shared among a number of services at the Edge of the network. Real-time video analytics is channeled as the killer application for Edge Computing.

In the future, we would like to deploy π-Hub under the hybrid Cloud-Edge paradigm, as shown in Fig. 14. π-Hub will be running not only on the cloud server, but also on various computation devices at the edge of the network, such as mobile devices, robots, gateways, local data center, or base stations, and so on. On different layers in the edge computing paradigm, π-Hub could perform video learning and storage to different degree based on the available resources and service requirements. For instance, on edge devices such as mobile phones or robots, data prepossessing tasks could be performed to reduce the data quantity and stress of the network. Data learning algorithms requires more computing resources and larger storage space could be running on the higher layer such as local data center. On the Cloud, π-Hub will support data sharing among multiple applications for retrieval and query. We foresee that this deployment could serve more users and applications under edge Computing meanwhile save resources such as bandwidth, storage space, and computing power.

6. Conclusion

We have witnessed the rapid growth of camera-equipped mobile devices and things in the last decade, which generate a tremendous amount of video data on the network. Several new

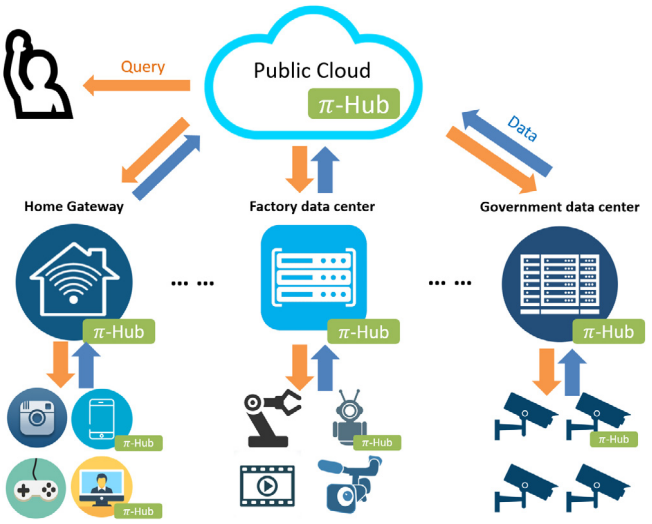


Fig. 14. π-Hub in edge computing.

challenges are raised to efficiently analyze, store, and retrieve the large amount of video data. In this paper, we take the challenges and introduce a learn-store-retrieve infrastructure called π-Hub for video analysis, storage, and retrieve. Our comprehensive evaluation results show that the optimization techniques improve the performance of the learning, storage and retrieval services significantly as well as notably reduce the cost of the system. We also verify π-Hub's scalability by reliably running a 1000-machine deployment to support up to one million users. Several lessons learned from this study and future work are also discussed.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

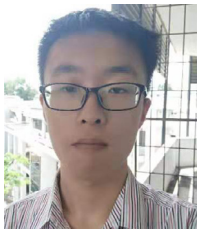
[1] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660.
[2] J. Tang, D. Sun, S. Liu, J.-L. Gaudiot, Enabling deep learning on IoT devices, Computer 50 (10) (2017) 92–96.
[3] H. Aghajan, A. Cavallaro, Multi-Camera Networks: Principles and Applications, Academic Press, 2009.
[4] L. Calavia, C. Baladrón, J.M. Aguiar, B. Carro, A. Sánchez-Esguevillas, A semantic autonomous video surveillance system for dense camera networks in smart cities, Sensors 12 (8) (2012) 10407–10429.
[5] G. Hancke, B. Silva, G. Hancke Jr, et al., The role of advanced sensing in smart cities, Sensors 13 (1) (2013) 393–425.
[6] A. Hampapur, L. Brown, J. Connell, A. Ekin, N. Haas, M. Lu, H. Merkl, S. Pankanti, Smart video surveillance: exploring the concept of multiscale spatiotemporal tracking, IEEE Signal Process. Mag. 22 (2) (2005) 38–51.
[7] X. Chang, Y.-L. Yu, Y. Yang, E.P. Xing, Semantic pooling for complex event analysis in untrimmed videos, IEEE Trans. Pattern Anal. Mach. Intell. 39 (8) (2016) 1617–1632.
[8] G. Kokkonis, K.E. Psannis, M. Roumeliotis, D. Schonfeld, Real-time wireless multisensory smart surveillance with 3D-HEVC streams for internet-of-things (IoT), J. Supercomput. 73 (3) (2017) 1044–1062.
[9] Z. Liu, T. Yan, Study on multi-view video based on IOT and its application in intelligent security system, in: Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer, MEC, IEEE, 2013, pp. 1437–1440.
[10] C. Long, Y. Cao, T. Jiang, Q. Zhang, Edge computing framework for cooperative video processing in multimedia IoT systems, IEEE Trans. Multimed. 20 (5) (2017) 1126–1139.
[11] S. Rani, S.H. Ahmed, R. Talwar, J. Malhotra, H. Song, IoMT: A reliable cross layer protocol for internet of multimedia things, IEEE Internet Things J. 4 (3) (2017) 832–839.
[12] M. Satyanarayanan, P.B. Gibbons, L. Mummert, P. Pillai, P. Simoens, R. Sukthankar, Cloudlet-based just-in-time indexing of IoT video, in: 2017 Global Internet of Things Summit (GIoTS), IEEE, 2017, pp. 1–8.
[13] A.R. Zamani, M. Zou, J. Diaz-Montes, I. Petri, O. Rana, A. Anjum, M. Parashar, Deadline constrained video analysis via in-transit computational environments, IEEE Trans. Serv. Comput. (2017).
[14] D. Acharya, K. Khoshelham, S. Winter, Real-time detection and tracking of pedestrians in CCTV images using a deep convolutional neural network, in: Proceedings of the 4th Annual Conference of Research@ Locate, Sydney, Australia, 2017, pp. 3–6.
[15] Z. He, D. Wu, Resource allocation and performance analysis of wireless video sensors, IEEE Trans. Circuits Syst. Video Technol. 16 (5) (2006) 590–599.
[16] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernandez, T. Vojir, G. Hager, G. Nebehay, R. Pflugfelder, The visual object tracking vot2015 challenge results, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2015, pp. 1–23.
[17] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[18] E. Bas, A. Tekalp, F. Salman, Automatic vehicle counting from video for traffic flow analysis, in: 2007 IEEE Intelligent Vehicles Symposium, IEEE, 2007, pp. 392–397.
[19] M. Bramberger, J. Brunner, B. Rinner, H. Schwabach, Real-time video analysis on an embedded smart camera for traffic surveillance, in: Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004, IEEE, 2004, pp. 174–181.
[20] V. Kastrinaki, M. Zervakis, K. Kalaitzakis, A survey of video processing techniques for traffic applications, Image Vis. Comput. 21 (4) (2003) 359–381.
[21] W. Wu, E.A. Bernal, R.P. Loce, M.E. Hoover, Multi-resolution video analysis and key feature preserving video reduction strategy for (real-time) vehicle tracking and speed enforcement systems, US Patent 8, 953, 044, 2015.
[22] J. Evans, P. Redmond, C. Plakas, K. Hamilton, D. Lane, Autonomous docking for intervention-AUVs using sonar and video-based real-time 3D pose estimation, in: Oceans 2003. Celebrating the Past... Teaming Toward the Future, IEEE Cat. No. 03CH37492, Vol. 4, IEEE, 2003, pp. 2201–2210.
[23] J.P. How, B. Behihke, A. Frank, D. Dale, J. Vian, Real-time indoor autonomous vehicle test environment, IEEE Control Syst. Mag. 28 (2) (2008) 51–64.
[24] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, S. Russell, Towards robust automatic traffic scene analysis in real-time, in: Proceedings of 12th International Conference on Pattern Recognition, Vol. 1, IEEE, 1994, pp. 126–131.
[25] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, M. Satyanarayanan, Scalable crowd-sourcing of video from mobile devices, in: Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services, ACM, 2013, pp. 139–152.
[26] T. Zhang, A. Chowdhery, P.V. Bahl, K. Jamieson, S. Banerjee, The design and implementation of a wireless video surveillance system, in: Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, ACM, 2015, pp. 426–438.
[27] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, M.J. Freedman, Live video analytics at scale with approximation and delay-tolerance, in: 14th {USENIX} Symposium on Networked Systems Design and Implementation, {NSDI} 17, 2017, pp. 377–392.
[28] S. Jain, V. Nguyen, M. Gruteser, P. Bahl, Panoptes: Servicing multiple applications simultaneously using steerable cameras, in: 2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN, IEEE, 2017, pp. 119–130.
[29] I. Stoica, D. Song, R.A. Popa, D. Patterson, M.W. Mahoney, R. Katz, A.D. Joseph, M. Jordan, J.M. Hellerstein, J.E. Gonzalez, et al., A Berkeley view of systems challenges for ai, 2017, arXiv preprint arXiv:1712.05855.
[30] L. Shaoshan, S. Dawei, Perceptin robotics get a performance boost from alluxio distributed storage, 2019, https://thenewstack.io/powering-robotics-clouds-alluxio. (Accessed 15 February 2019).
[31] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, in: Advances in Neural Information Processing Systems, 2015, pp. 91–99.
[32] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, in: Proceedings of the 22nd ACM International Conference on Multimedia, ACM, 2014, pp. 675–678.
[33] K. Chodorow, MongoDB: the definitive guide: powerful and scalable data storage, O'Reilly Media, Inc., 2013.
[34] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, I. Stoica, Tachyon: Reliable, memory speed storage for cluster computing frameworks, in: Proceedings of the ACM Symposium on Cloud Computing, ACM, 2014, pp. 1–15.

[35] V.N. Padmanabhan, J.C. Mogul, Using predictive prefetching to improve world wide web latency, ACM SIGCOMM Comput. Commun. Rev. 26 (3) (1996) 22–36.
[36] S. Agarwal, Queries with Bounded Errors & Bounded Response Times on Very Large Data (Ph.D. thesis), UC Berkeley, 2014.

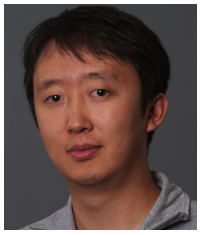
Dr. Jie Tang is currently an associate professor in School of Computer Science and Engineering of South China University of Technology, Guangzhou, China. She received her B.E. degree From University of Defense Technology in 2006, and Ph.D. degree from the Beijing Institute of Technology in 2012, both in Computer Science. She was previously a visiting researcher at the Embedded Systems Center at University of California, Irvine, USA, and a research scientist at Intel China Runtime Technology Lab. Dr. Tang is mainly doing research on Computer Architecture, Autonomous Driving, Cloud and Run-time System. She is a founding member and secretary of the IEEE Computer Society Special Technical Community on Autonomous Driving Technologies.

Dr. Shaoshan Liu is Founder and CEO of PerceptIn (www.perceptin.io), a company focusing on providing visual perception solutions for autonomous robots and vehicles. Before founding PerceptIn, Dr. Shaoshan Liu was a founding member of Baidu U.S.A. as well as the Baidu Autonomous Driving Unit, in charge of system integration of autonomous driving systems. Dr. Shaoshan Liu received Ph.D. in Computer Engineering from University of California, Irvine. His research focuses on Computer Architecture, Deep Learning Infrastructure, Robotics, and Autonomous Driving. Dr. Shaoshan Liu has published over 40 high-quality research papers and holds over 150 U.S. international patents on robotics and autonomous driving, he is also the lead author of the best selling textbook "Creating Autonomous Vehicle Systems", which is the fifirst technical overview of autonomous vehicles written for a general computing and engineering audience. In addition, to bridge communications between global autonomous driving researchers and practitioners, Dr. Shaoshan Liu co-founded the IEEE Special Technical Community on Autonomous Driving Technologies and serves as the Founding Vice President. Dr. Shaoshan Liu is a senior member of IEEE, an ACM Distinguished Speak, and a IEEE Computer Society Distinguish Speaker.

Jie Cao is an Assistant Professor of School of Information Security and Applied Computing at Eastern Michigan University. He received his Ph.D. and M.S. in Computer Science Department at Wayne State University. His research interests span the areas of Edge Computing, smart home, connected health, and autonomous vehicle. Especially, his work focuses on data management of IoT systems, including data quality, semantic annotation, data privacy, and security. Current, his team is investigating the vulnerability and protection of the autonomous vehicle. His research has been published in prestigious venues, including IEEE Internet of Things Journal and ICDCS, winning HealthCom Best Student Paper Award and an IEEE Best Paper Award. His book "Edge Computing: A Primer" by Springer concludes his research on Edge Computing.



Dawei Sun receieved his BE degree from the Department of Automation at Tsinghua University. He is a researcher working on machine learning and computer vision. This work was done when Dawei interned at PerceptIn.



Dr. Bolin Ding is a research scientist/director in the Data Analytics and Intelligence Lab (DAIL) at Alibaba DAMO Academy. Prior to joining Alibaba, he was a researcher in DMX group at Microsoft Research. I completed my Ph.D. in Computer Science at University of Illinois at Urbana-Champaign. His research goals and interests center on large-scale data management and analytics, including interactively querying and mining "big" data, privacy-preserving data analytics, and optimizations in databases and machine learning. He is particularly interested in algorithms which have guarantees in theory, and are effective and implementable in practice.

Dr. Jean-Luc Gaudiot is a Professor in Department of Electrical Engineering and Computer Science, University of California-Irvine. He is Fellow, IEEE, Fellow, AAAS, and 2017 IEEE Computer Society President, Eta Kappa Nu, Honor Society of IEEE, Professional Member.

His research interests include multithreaded architectures, fault-tolerant multiprocessors, and implementation of reconfigurable architectures. He has published over 200 journal and conference papers. His research has been sponsored by NSF, DoE, and DARPA, as well as a number of industrial organizations.

Dr. Weisong Shi (Fellow, IEEE) received the B.S. degree in computer engineering from Xidian University, Xi'an, China, in 1995, and the Ph.D. degree in computer engineering from the Chinese Academy of Sciences, Beijing, China, in 2000. He is currently the Charles H. Gershenson Distinguished Faculty Fellow and a Professor of computer science with Wayne State University, Detroit, MI, USA. His current research interests include edge computing, computer systems, energy efficiency, and wireless health. Dr. Shi was a recipient of the National Outstanding Ph.D. Dissertation Award of China and the NSF CAREER Award.