Processing of Mixed-Sensitivity Video Surveillance Streams on Hybrid Clouds*

Chunwang Zhang, Ee-Chien Chang
School of Computing, National University of Singapore
{*chunwang, changec*}*@comp.nus.edu.sg*

***Abstract*—We consider a hybrid cloud model for video surveillance systems with mixed-sensitivity video streams. The hybrid cloud naturally addresses the issues on security by keeping sensitive data in the private cloud, and relieves seasonal workload by pushing computation to the elastic public cloud. Nevertheless, to enhance usability and reduce cost, it is desired to have a middleware that seamlessly integrates the two clouds and schedules the tasks effectively. We first present a stream processing model that is specifically designed for this hybrid cloud setting. Based on this model, we formalize the scheduling issue as an optimization problem that minimizes overall monetary cost to be incurred on the public cloud, with resource, security and Quality-of-Service (QoS) constraints. Our proposed scheduler exploits special properties of hybrid clouds for more effective solutions. Experiments through both large-scale simulations and prototype runs on Amazon EC2 show that the proposed approach is effective in outsourcing computational workload with overheads lower than other alternatives.**

***Keywords*-Video surveillance; hybrid clouds; scheduling; security and privacy**

I. INTRODUCTION

Video surveillance systems are inherently data-intensive, and often compute-intensive with various operations like transcoding, indexing and video analysis. Such computational workload could be seasonal, for example, heavier workload in the morning of workdays while lighter workload during weekend nights, as observed in typical video streaming systems [1]. An organization's in-house private datacenter may be overloaded during peak hours due to its limited computing capability. While with cloud computing, it is possible to offload all the video streams and computation to a public cloud like Amazon AWS, such strategy can incur high monetary cost [2]. More importantly, video surveillance streams may contain sensitive information that cannot be directly handled on the public cloud due to potential data leakages [3]. Although it is possible to protect the sensitive streams by processing them in the encrypted domain using homomorphic encryptions [4], such techniques are still too expensive for large-scale video data.

A recent trend in cloud computing is that of the *hybrid cloud*, whereby an organization's private datacenter is seamlessly integrated with the elastic public cloud. A hybrid cloud allows an organization to strategically push certain computation to the public cloud, potentially addressing both of the aforementioned issues on seasonal workload and security. This hybrid cloud model has gained adoptions and is still undergoing rapid development [5].

However, the scheduling decisions of how much and which part of the computation to outsource is generally hard to predict due to frequent changes of system status. Also, from application developers' point of view, it is preferred that they only need to specify how computations are to be carried out, without caring about where they are executed and how data are moved during execution. Such transparency makes it easier for developers with no experience in parallel/distributed systems to write programs working on large clusters. Therefore, it is desired to have a middleware that unifies the two clouds and effectively schedules the processing of mixed-sensitivity video streams on the hybrid cloud.

Many stream processing systems have been developed in the past few decades [6]–[9]. Alongside with these systems, there are also a large number of scheduling models proposed [10]–[16]. Although our problem can be treated as a special case of some known general scheduling models, its specialized settings can be exploited for more effective solutions, making it scalable to larger instances. Observed that in our setting, servers within each cloud are typically connected by a high-bandwidth, low-latency network (e.g., Gigabit LAN), whereas connections across the two clouds have to go through a wide area network or the Internet, having relatively smaller bandwidth and higher latency. Also, according to today's typical cloud pricing models [17], data transmission within each single cloud is free-of-charge while data traffic across the two clouds incurs high monetary cost, e.g. $0.12/GB. Hence, we can group and treat the servers in our setting as two servers: one private server with a fixed amount of computing power and one public server with elastic resources, and connections between these two servers are costly. In addition, in many scenarios the effect of public cloud's computation cost is much lower than the inter-cloud communication cost [17]. Therefore, we can stress less on the public servers' computation cost in the cost model. The security requirement places another hard constraint on where certain streams can be processed. These specialized settings in turn allow us to focus on the processing of larger number of streams (e.g., hundreds) with reasonable length of tasks, e.g., around 10 operations per task.

In this paper, we model stream processing as a set of task templates whereby each template can be independently instantiated to multiple video streams. Each task template

*This work is supported by the Singapore NRF under its IRC@SG Funding Initiative and administered by the IDMPO, and by the research grant R-252-000-514-112.

is represented as a loop-free, directed graph of operations, with the code provided by application developers. In addition, the developers can specify multiple connection points [6] in a task graph whereby clients can dynamically tap into the stream data during execution. The locations of the connection points provide useful information to the scheduler, so that dynamic changes to the task graphs do not necessarily trigger rescheduling. However, as sensitivity of video streams can change during run-time, rescheduling might be required occasionally. In particular, if the sensitivity of a stream in the public cloud changes from non-sensitive to sensitive, the stream must be rescheduled to the private cloud to prevent potential data leakages. This can be done by buffering or dropping data frames before the rescheduling is completed.

We formalize the scheduling problem as an optimization problem that minimizes the overall monetary cost to be incurred on the public cloud, with the resource, security and Quality-of-Service (QoS) constraints. We propose an algorithm that exploits the aforementioned specialized properties of hybrid clouds for more efficient solutions. Essentially, for each task template of the input, we search for the set of "minimal configurations" and then employ integer programming to select the desired configurations. For templates that are reasonably short ($\sim$10 operations), the set of minimal configurations is typically sufficiently small for state-of-the-art solvers [18]. In cases where the number of minimal configurations is large, we provide a heuristic that selects only a few representatives to further improve the performance. The challenge in our scheduling problem is more on determining how a large number of relatively short tasks are to be scheduled on the two servers, instead of scheduling a single large task among multiple servers considered by many existing works.

The proposed stream processing model and scheduling mechanism can be built on top of existing stream processing systems like Borealis [7] and Storm [19]. To facilitate experiments and testing, instead of using existing platforms, we implemented a proof-of-concept system with basic functionality of video streaming and several operations like transcoding, face detection, etc. We conducted extensive experiments, through both large-scale simulations and actual runs with our proof-of-concept system on Amazon EC2. The result shows that it is feasible to process video streams in a hybrid cloud, preserving data-privacy and reducing monetary cost as compared to a pure public cloud deployment. The overheads of our scheduler are also much lower than other alternatives.

II. Hybrid Cloud Video Surveillance Model

A. System Model

We consider a hybrid cloud model as shown in Fig. 1. In this model, the private cloud has a fixed number of servers, each of which has limited computing power. In contrast, the public cloud has elastic computing resources that can be allocated and de-allocated on-demand. Servers within each cloud are connected to each other by a high-bandwidth, low-latency

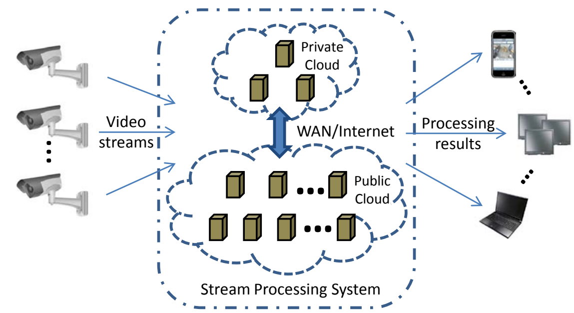


Figure 1. System model for hybrid cloud video surveillance.

network (e.g., Gigabit LAN) whereas connections between servers across the two clouds have to go through a WAN or the Internet. Hence, the inter-cloud connections have relatively smaller bandwidth and larger delay. In addition, under current typical cloud pricing models, data transmission within each single cloud is free-of-charge while data traffic to the Internet (i.e., inter-cloud data traffic) incurs high monetary cost, e.g., \$0.12/GB [17]. Based on these observations, we group and treat the servers in our system as two servers: a private server with a limited aggregated computing power, denoted as C, and a public server with elastic computing capacity. These two servers are connected by a long-distance link with an estimated bandwidth B and link delay L, while data transfer within each server incurs no cost.

The system contains a large number of surveillance cameras distributed over a wide area. Each camera generates a video stream that is to be sent to some servers in the two clouds. The stream is processed in the hybrid cloud, in a way specified by application developers, and then streamed to multiple clients. Besides, input/output streams can also be originated from/written to storage servers. Note that both the cameras and clients can be within or outside the local area network of the private cloud.

B. Stream Processing Model

A *task template* consists of a sequence of operations that can be applied to multiple input streams. We model a task template T as a directed, acyclic and labeled graph $G(V, E)$ where $V = V_{src} \cup V_{op} \cup V_{sink}$. The set V_{src} is the set of stream sources which could be cameras or recorded videos retrieved from storage systems; V_{sink} is the set of streaming sinks which could be display devices or storage systems as well. V_{op} contains the set of *operations* such as transcoding, background extraction, object detection etc. Application developers can provide the code for each operation or select it from a library. The edges in E define the data flows between the vertexes in V. Fig. 2 gives an example of task template. Recall that a single template can be independently instantiated to multiple input streams. In an *instantiated task*, each source and sink node is associated with a location, e.g., IP address, indicating in which cloud, private or public, the node resides, while operations in V_{op} have not been assigned. In contrast, in an *assigned task*, not only the source and sink

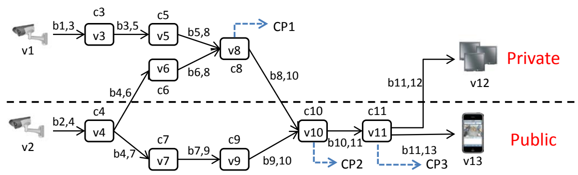


Figure 2. Illustration of a stream processing task.

nodes are instantiated, each operation in V_{op} is also assigned a label, *private* or *public*, indicating in which cloud the operation is to be executed.

Similar to Aurora* [6], an application developer can specify multiple connection points in a task graph where data streams will be cached in some storage for a certain amount of time. Such connection points are useful in supporting ad-hoc queries and dynamically joined clients. For example, one might be interested in finding out whether a particular person appeared in a building yesterday evening between 6-12pm. Such a query can be similarly defined and attached to the connection points of existing running tasks that have the required data, without rescheduling the whole set of tasks.

C. Security Model

We consider servers on the public cloud to be *honest-but-curious*, that is, they will follow the protocol and carry out the required computation honestly, but may retain information collected from the computation for malicious purposes. In contrast, the private servers are fully trusted.

Every stream in a task is tagged with an attribute, *sensitive* or *non-sensitive*. Sensitive streams must not leave the private cloud in order to prevent data leakages while there is no constraint for non-sensitive streams. Unless otherwise specified, all streams in a task are tagged.

Not all possible ways of tagging streams are valid. If all the input streams of an operation are non-sensitive, the output stream has to be non-sensitive as well. We refer it to as the *non-upgrading policy*. The non-upgrading policy imposes a constraint which excludes certain undesired scenarios, e.g., excluding cases where non-sensitive streams that have been pushed to the public cloud are later tagged to be sensitive. On the other hand, it is possible that on sensitive input, the output is non-sensitive. For example, an operation that takes in a sensitive video stream may output a lower resolution stream that is deemed as non-sensitive. We refer it to as the *downgrading policy*. This policy allows pushing more computation to the public cloud which is useful in many scenarios.

Note that the sensitivity of a stream can change during runtime, from sensitive to non-sensitive or vice versa, and a rescheduling may be required due to such realtime changes. In particular, if a stream assigned to the public cloud suddenly becomes sensitive, it must be rescheduled to the private cloud so as to meet the security requirement. Data frames have to be properly "buffered" or dropped before the rescheduling is fully carried out. This will introduce certain performance overhead, either in terms of extra delays or data losses. Fortunately, there are existing techniques supporting fast operation migration during runtime [6], [13].

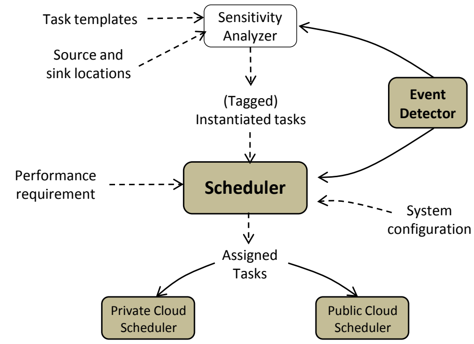


Figure 3. Architecture of hybrid cloud video surveillance system.

D. Cost Model

Each operation in the task graph implements a video processing function, requiring a certain amount of computing power, denoted as c, to generate the output *in realtime*. In this paper, we measure computing cost in terms of the number of ECUs (Amazon EC2 Compute Unit).[1] We assign the cost of 1 ECU to an operation if the operation can be carried out in realtime by a machine with 1 ECU capacity. The computing cost can be estimated based on the input streams' frame size and data rate, combined with pre-conducted resource profiling baselines for the operations. Each connection in the graph represents a data flow from one operation to another, which requires a certain amount of bandwidth, denoted as b, to transfer the data in realtime. The bandwidth cost is measured in MB/s, which can be similarly estimated from the user-desired stream rate and frame size.

One advantage of our cost model is that, it directly approximates monetary cost, giving system administrators a good overview of the projected cost.

E. System Architecture

Fig. 3 shows the overall architecture of the proposed hybrid cloud video surveillance system. The Sensitivity Analyzer, provided by application developers, takes as input a set of task templates and the corresponding source and sink locations, evaluates the sensitivity of the streams, and outputs a set of instantiated tasks.

The instantiated tasks are then fed into the Scheduler, together with the information of performance requirement and system configuration including private cloud's computing power, inter-cloud bandwidth and link delay etc. Based on these inputs, the Scheduler decides how to assign the operations in each task to the two clouds.

Each cloud, public or private, has an intra-cloud scheduler. As scheduling within a cluster is not within the scope of this paper, in our experiment, we use a simple greedy algorithm that always picks the next available server.

[1] According to Amazon, each ECU provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or Xeon processor.

The Event Detector detects changes in the task graphs, stream sensitivity and other system configurations, and initiates rescheduling when necessary.

III. Problem Formulation

Although our scheduling problem can be treated as a special case of known general scheduling models, its specialized "two-server" setting leads to more efficient solutions and thus allows scaling up to larger instances. This section formalizes the scheduling problem, with possible extensions of the stream processing model.

A. Optimization Problem

Usage of public cloud resources incurs additional monetary cost, including both the compute and bandwidth cost. Given a set of tasks each consisting of multiple operations, we want to assign each operation to either the public or private cloud, such that the total monetary cost to be incurred on the public cloud is minimized, subject to the constraints that the private cloud cannot be overloaded, sensitive streams cannot flow into the public cloud and the QoS requirements can be met.

1) The Scheduling Problem: The input is a sequence of task templates $\mathcal{T} = \langle T_1, \ldots, T_m \rangle$ and a sequence of integers $R = \langle r_1, \ldots, r_m \rangle$ where each template T_i is to be instantiated r_i times to different sources and sinks. For ease of exposition, let us rewrite the input in the equivalent form of $\hat{\mathcal{T}} = \langle \hat{T}_1, \ldots, \hat{T}_n \rangle$ where each $\hat{T}_i$ is an instantiated task, and $n = \sum_{i=1}^{m} r_i$. Each operation v_j^i in $\hat{T}_i$ is associated with a computing cost c_j^i and each connection from v_j^i to v_k^i in $\hat{T}_i$ requires a bandwidth cost b_{jk}^i. Let the QoS requirement be the maximum allowed end-to-end delay d_i for each $\hat{T}_i$. The scheduling problem is to decide the binary assignment x_j^i for each operation v_j^i in $\hat{T}_i$ (where value 0 and 1 corresponds to being assigned to public and private respectively), in such a way that the total incurred monetary cost on the public cloud

$$\alpha \sum_{i,j} c_j^i (1 - x_j^i) + \beta \sum_{i,j,k} b_{jk}^i |x_j^i - x_k^i| \tag{1}$$

is minimized, subject to the following constraints: (1) the private cloud must not be overloaded, i.e., $\sum_{i,j} c_j^i x_j^i \leq C$; (2) sensitive streams must not leave the private cloud; and (3) any assigned task must meet the delay requirement. Details on the determination of delay will be discussed in Sect. III-A3. Recall that in the input, the source and sink nodes are already labeled to be in either the public or private cloud and thus cannot be reassigned.

2) Determining α and β: The first term in the above objective function (1) represents the computation cost on the public cloud and the latter represents the bandwidth cost for inter-cloud data transmission.[2] Since we are handling stream data, we measure the cost rate, e.g., dollars per hour. The parameters α and β represent the unit-price, whose values could be determined according to the pricing model of the cloud provider. Taking Amazon EC2's pricing model as an example, each ECU costs \$0.08/hour and inter-cloud bandwidth usage costs \$0.19/GB (for Singapore regions), hence, α and β can be set as 0.08 and 0.684 accordingly. Due to the large video data, computation cost is typically much less than communication cost. To illustrate, let us assume that an instance with 1 ECU is able to handle realtime processing of one high-definition video stream with 1 MB/s inter-cloud bandwidth requirement, then the cost would be \$0.764 for a 1-hour run (\$0.08 for computation + \$0.684 for bandwidth), where the computation cost is around one-eighth of the bandwidth cost. This suggests that minimizing only the bandwidth could give good approximations to solutions that minimize the monetary cost. This is verified in our experiments in Sect. V. Hence, to speed up the scheduler, one could omit the computation cost in the cost model.

3) Estimating end-to-end delay: For an assigned task, the end-to-end delay is the maximum delay among all its paths. Along a path, the total delay is the sum of the processing time and the communication latency. We assume that the processing time of each operation is specified in the input, and the delay due to inter-cloud communication is a known constant. Recall that intra-cloud communication is assumed to incur no delay, however, it can be included in the calculation if required. From the information provided, we can estimate the end-to-end delay for each assigned task.

B. Extension of the Stream Processing Model

Our scheduler can also handle the variation where there are multiple ways to carry out a task. In this variation, an application developer can specify multiple task graphs that are considered to be functionally equivalent. For example, a task of performing face detection and drawing boxes on detected faces on a high-resolution video stream can also be carried out in another way: first, transcodes the high-resolution video stream to a low-resolution stream; performs face detection on the low-resolution stream; and then draws boxes on the original high-resolution stream. Note that face detection can achieve high accuracy on low-resolution videos [20]. If the low-resolution stream is tagged as non-sensitive, and the above two ways are specified as functionally equivalent, when necessary, the scheduler can push face-detection to the public cloud to reduce load in the private cloud.

IV. Proposed Approach

Not surprisingly, the scheduling problem defined in Sect. III is NP-hard.[3] Nevertheless, by pruning, we are able to handle fairly large instances. Essentially, for each task template, our algorithm searches for the set of "minimal configurations" and then employs integer programming to select the desired configurations. For a template with, e.g., 10 operations, although there are 2^{10} configurations, typically it can be

[2]While some cloud providers only charge for the data traffic out, in this paper, we assume that both two-way traffic incur monetary cost.

[3]This can be proved by a reduction from the 0-1 knapsack problem.

pruned down to around 20. For larger templates, we provide a heuristic to further reduce the number of configurations.

A. Transforming to Integer Programming

A task template with t operations gives 2^t ways of assigning its operations to the two clouds. Let us call each assignment a *configuration* and denote it as $f = \langle f^{pri}, f^{pub}\rangle$ where f^{pri} and f^{pub} are the set of operations assigned to the private and the public cloud respectively. Let us denote $\mathcal{F}(T)$ as the set of all configurations for a task template T.

For each $f \in \mathcal{F}(T)$, we can calculate a 2-tuple load-cost value (a, b) where a is the computing load on the private cloud and b is the cost. For our choice of objective function, b is the monetary cost to be incurred on the public cloud. Similarly, we can estimate the end-to-end latency $\ell(f)$ for each f as described in Sect. III-A3. Let $\mathcal{F}(\mathcal{T}) = \mathcal{F}(T_1) \cup \cdots \cup \mathcal{F}(T_m)$.

The scheduling problem described in Sect. III can be easily transformed to the following integer programming problem:

1) Integer Programming: Given the input $\mathcal{T} = \langle T_1, T_2, \ldots, T_m\rangle$ and $R = \langle r_1, r_2, \ldots, r_m\rangle$ where each task template T_i is to be instantiated to r_i streams. We want to find x_i, the number of times a configuration f_i in $\mathcal{F}(\mathcal{T})$ is to be instantiated, such that the total monetary cost,

$$\sum_{\forall i, f_i \in \mathcal{F}(\mathcal{T})} b_i x_i$$

is minimized, subject to: (1) the private cloud resource constraint, i.e., $\sum_i a_i x_i \le C$; (2) the number of instances constraint, i.e., $\forall j \in [1, m]$, $\sum_{\forall i, f_i \in \mathcal{F}(T_j)} x_i = r_j$; (3) the security constraint, i.e., if $x_i > 0$, then the corresponding configuration f_i does not push sensitive streams to the public cloud; and (4) the QoS constraint, i.e., $\forall j \in [1, m]$, $\forall i$, if $f_i \in \mathcal{F}(T_j)$ and $\ell(f_i) > d_j$, $x_i = 0$. This is an integer programming problem [21] with $|\mathcal{F}(\mathcal{T})|$ unknowns and about $3m + 1$ constraints.

B. Minimal Configurations

For a template T, the set of all configurations in $\mathcal{F}(T)$ could be large. Fortunately, only a small number of them need to be considered. Let us consider two different configurations f and $\widetilde{f}$ with respective load-cost value of (a, b) and $(\widetilde{a}, \widetilde{b})$ satisfying $a \le \widetilde{a}$ and $b \le \widetilde{b}$. Note that $\widetilde{f}$ will not appear in an optimal solution (otherwise, we can replace it by f, yielding a solution with smaller cost). Hence, consider the partial order $\preceq$ on $\mathcal{F}(T)$ where $f_i \preceq f_j$ iff $a_i \le a_j$ and $b_i \le b_j$, the optimal solution must be in the minimal configurations. Fig. 4 gives an example of minimal configurations, marked as solid red diamonds. Let $\mathcal{MF}(T)$ be the set of minimal configurations for each T.

Study on the size of $\mathcal{MF}(T)$. We use 5 different task templates created in Sect. V-A where the number of operations varies from 8 to 12. For each template, we assign a random computing cost within $(0, 2]$ ECUs to each operation and a random bandwidth cost within $(0, 1]$ MB/s to each connection. The values of α and β are set to be 0.08 and 0.684 respectively. The process is repeated 1,000,000 times for each template.

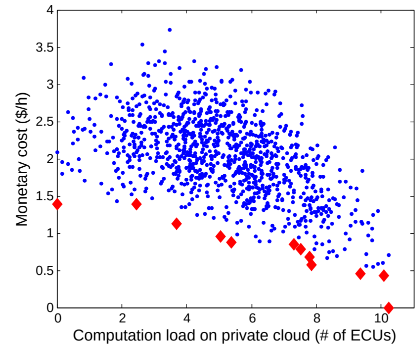


Figure 4. Illustration of configurations in the 2D load-cost graph. Those marked as solid red points are the minimal configurations.

Table I
STUDY ON THE SIZE OF MINIMAL CONFIGURATIONS $\mathcal{MF}(T)$.

Task Template	$\lvert\mathcal{F}(T)\rvert$	$\lvert\mathcal{MF}(T)\rvert$			
		min	**max**	**avg**	**95th percentile**
T_6	2^8	3	25	9.658	15
T_7	2^9	4	25	11.4	17
T_8	2^{10}	5	26	13.31	19
T_9	2^{11}	5	32	13.567	21
T_{10}	2^{12}	5	36	14.257	22

The result is shown in Table I. Interestingly, for more than 95% of the instances, the size of $\mathcal{MF}(T)$ grows linearly rather than exponentially. The maximal value of $|\mathcal{MF}(T)|$ observed in all the runs is only 36.

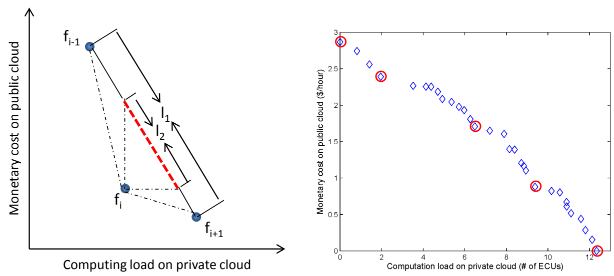
C. Heuristic Selecting Method

However, there could be cases where $|\mathcal{MF}(T)|$ is large, e.g., when T is large. For such cases, we provide a heuristic to select a constant number ϵ (e.g., $\epsilon = 20$) of representatives among the minimal configurations. Different from using all the minimal configurations, the heuristic may not lead to optimal solutions.

As illustrated in Fig. 5(a), let us consider three consecutive configurations f_{i-1}, f_i, f_{i+1} in $\mathcal{MF}(T)$, which form a concave curve. If both f_{i-1} and f_{i+1} contribute to a solution, then the aggregated load-cost value may fall on the dotted line l_2 as shown in Fig. 5(a), leading to a configuration that is greater than f_i under $\preceq$. In this sense, there is a good chance that both f_{i-1} and f_{i+1} are not in the optimal solution, and can be represented by f_i.

Let us define the ratio of l_2 over l_1 as the "likelihood" that f_{i-1} and f_{i+1} can be excluded. Our heuristic repeatedly picks the largest "likelihood" among consecutive minimal configurations, until ϵ configurations are selected. Fig. 5(b) illustrates the selection result of 5 representatives, marked as red circles.

Effectiveness of the heuristic. To investigate the effectiveness of the heuristic, we use one task template created in Sect. V-A with $r = 100$ and C ranging from 400 to 800. We compare the optimal cost derived from all the minimal configurations, and the optimal cost derived from the selected configurations.



(a) Key observation (b) Selection result ($\epsilon = 5$)

Figure 5. Illustration of the heuristic.

Table II
EFFECTIVENESS OF THE HEURISTIC.

Template	Conf. Set	Private cloud computing power (# of ECUs)			Max Diff.	Avg Diff.
		400	600	800		
T_8	Minimal	94.920	65.829	36.739	0.10%	0.06%
	Selected	94.922	65.833	36.744		
T_9	Minimal	128.234	93.930	60.075	0.16%	0.07%
	Selected	128.440	93.981	60.075		
T_{10}	Minimal	142.152	106.684	71.171	0.14%	0.073%
	Selected	142.166	106.761	71.270		

The result is shown in Table II. The same test is also repeated on other 2 task templates.

The empirical result shows that our heuristic is effective, giving solutions that are within 0.1% of the optimal ones.

V. EVALUATION

We conduct experiments through both simulations and actual runs with our proof-of-concept system on Amazon EC2. The simulations can be repeatedly conducted on large instances, whereas the actual runs involve more accurate running environment but on relatively smaller instances.

A. Simulations

The simulations are conducted under two different settings: *with* and *without* security constraint. To this end, 10 different task templates are created using the method described in [11], where the number of operations in each template ranges from 3 to 12 (with a step of 1). Each operation is assigned a random computing cost within (0,2] ECUs and each connection has a random bandwidth cost within (0,1] MB/s. Each template is to be instantiated to 10 video streams, hence there are a total of 100 video streams. The source and sink nodes are in the private cloud. We apply a few schedulers (described below) and vary the computing power of private cloud from 200 to 600 ECUs with step size 100. The values of α and β are set to be 0.08 and 0.684. The bandwidth and delay of the inter-cloud connection are set to be 20 MB/s and 250ms respectively. The end-to-end delay constraint for each template T is set to be $P + 1$s where P is the total processing time along the longest path in T. Hence, the delay incurred by the communication is constrained to be at most 1 second.

We compare among the following 5 scheduling algorithms: 1) *Task-Level Water-filling (TLW)*: assign all operations in a task to private if one of the streams is tagged as sensitive, otherwise assign all operations to the public cloud; 2) *Task-Level Random (TLR)*: same as TLW for tasks with at least one sensitive stream. For tasks tagged with only non-sensitive streams, the whole task is randomly assigned to the public or private cloud; 3) *Greedy*: consider each task one-by-one iteratively. In each round, choose the optimal assignment (minimizing the cost) w.r.t. the updated resource requirements. 4) *ProposedC*: our proposed approach with objective to minimize the monetary cost; and 5) *ProposedB*: our proposed approach with objective to minimize the inter-cloud bandwidth usage.

Table III
TIME TO SOLVE THE INTEGER PROBLEMS.

	$\mathcal{F}(\mathcal{T})$	$\mathcal{MF}(\mathcal{T})$
# of Configurations	8184	157
Corresponding Solving Time	2.794s	0.078s

1) Simulation result without security constraint: In this simulation, all the 100 streams are non-sensitive. Fig. 6 shows the result under this setting. Observed that both TLR and TLW underutilize the private cloud resources (Fig. 6(c)). Both of our proposed schedulers outperform the others in all the three measures. The differences between ProposedC and ProposedB are indistinguishable as bandwidth cost dominates the total cost.

Table III shows the average time taken by the proposed scheduler. We can reduce the 8184 unknowns down to 157, which in turn reduces the computing time to be less than 0.1s.

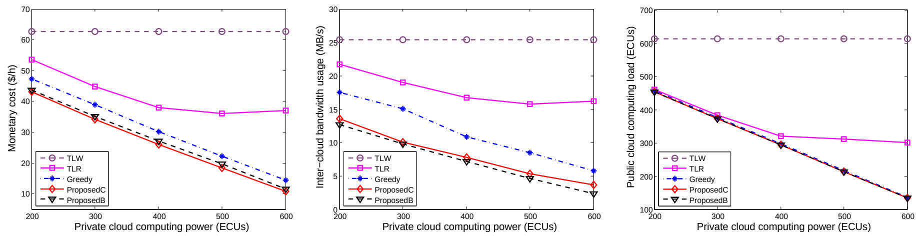
2) Simulation result with security constraint: We then randomly tag the 100 streams and repeat the above simulation. The result, averaged over 3 random runs, is shown in Fig. 7. TLW, TLR and Greedy cannot schedule all the tasks when the private cloud has low computing power (i.e., $C = 200$), partly due to the short-sightedness in using the private cloud resources. In contrast, the proposed schedulers exploit global knowledge on all the tasks and hence can schedule all of them. Our schedulers again outperform the other alternatives.

Observed that with 200 ECUs in the private cloud, and assuming at the peak overall workload of about 650 ECUs, the number of ECUs employed in the public cloud is about 450 (Fig. 6(c)), incurring about 13 MB/s inter-cloud bandwidth (Fig. 6(b)). This amounts to monetary cost of approximately 36 + 8.9 = $44.9/h during peak. Considering an "offload all" strategy that pushes all video streams and computation to the public cloud, the overall cost would be around $63.1/h. Hence, we have a reduction in cost of about 29%. With more private resources, the monetary cost can be further reduced, as indicated in Fig. 6 and 7.

B. Prototype Evaluation

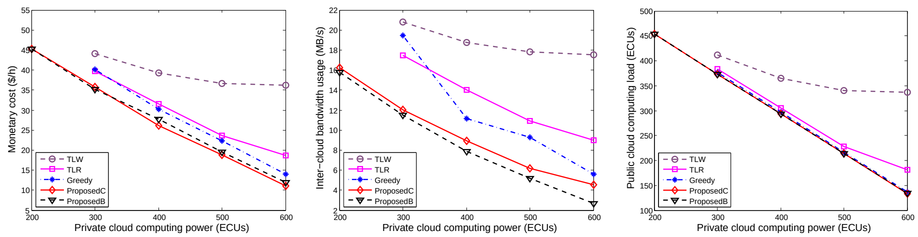
We also implemented a proof-of-concept system for hybrid cloud video surveillance, with basic functionality of video streaming and operations including transcoding, background extraction, face detection etc. We remark that the proposed scheduling mechanism can be incorporated into existing stream processing systems like Apache Storm.

1) Hybrid Cloud Setting: We build a hybrid cloud on Amazon EC2 across Singapore and US West. The private cloud



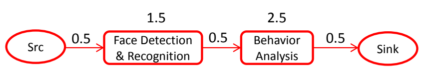
(a) Monetary cost (b) Inter-cloud bandwidth (c) Outsourced computation

Figure 6. Simulation result without security constraint (ProposedC, ProposedB and Greedy are indistinguishable in (c)).

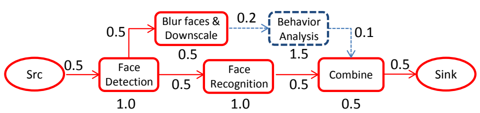


(a) Monetary cost (b) Inter-cloud bandwidth (c) Outsourced computation

Figure 7. Simulation result with security constraint (ProposedC, ProposedB and Greedy are indistinguishable in (c)).



(a) Alternative graph 1



(b) Alternative graph 2

Figure 8. Task template for prototype evaluation which has two alternatives.

has 4 standard large instances located at Singapore. Each instance provides 2 virtual cores with 4 ECUs, 7.5GB memory and 840GB storage. Hence, the private cloud in our setting has a total computing power of 16 ECUs. The public cloud, located at N. California, has 10 large instances that can be allocated and released on-demand. All instances run Ubuntu 12.04. The available bandwidth between these instances is not specified by Amazon. An informal test of file transfer using `scp` indicates ∼40 MB/s within the same region (e.g., from Singapore to Singapore), and 5–8 MB/s across different regions (e.g., from Singapore to California). The network delay is less than 1 millisecond for intra-cloud connections and around 250 milliseconds for inter-cloud connections.

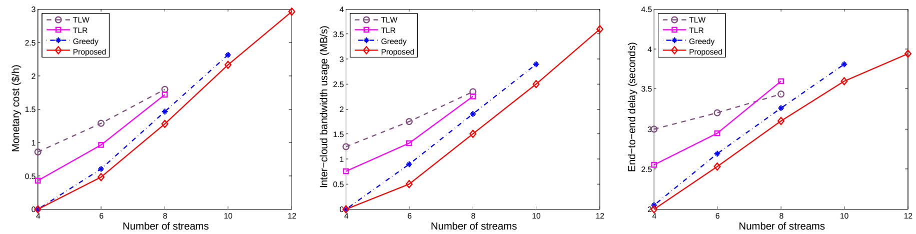
2) Experimental Setting: We experiment on one task template, which performs face recognition and behavior analysis that can be carried out in two different ways as illustrated in Fig. 8, with the cost estimated from a few test runs. We gradually increase the number of streams from 4 to 12, with randomly half of them being tagged as sensitive. The maximum allowed end-to-end delay is set to be 5s. We record the actual amount of data transfer across the two clouds, the average end-to-end delay, and calculate the monetary cost spent on the public cloud for a 1-hour run.

3) Result and Analysis: The result is shown in Fig. 9. Both TLW and TLR fail to schedule all the tasks when the number of streams is greater than 9. Greedy can handle more tasks by pushing some non-sensitive operations to the public cloud, but also fails when the number of streams reaches 12. The proposed schedulers give the smallest cost, bandwidth usage and average end-to-end delay. Since bandwidth cost dominates the total cost, in the experiment, ProposedC and ProposedB always choose the same configurations. Hence, they are rendered as one line (Proposed) in Fig. 9.

VI. Related Work

Many stream processing systems have been developed in the past few decades, from centralized settings like Aurora [6] to distributed settings like Borealis [7], Nephele [9], S4 [8] and Storm [19]. Together with these systems, there are also a large number of works focusing on scheduling among multiple servers, with various goals such as to minimize the end-to-end application latency [10], [11], maximize the aggregated throughput [14], [15], optimize a combination of latency and throughput referred to as network-usage [12], [13], balance the workload and resource usage among all servers [16], [22], or maximize the reuse among multiple queries [23], [24]. Our work differs from the previous works in its two-server setting, which can be exploited for more efficient solutions and hence enables scheduling of multiple tasks on a large number of video streams.

With the recent advances in hybrid cloud computing, there are increasing research interests in workload scheduling on hybrid clouds. Zhang et al. [1] propose a hybrid cloud computing model for Internet-based applications with highly dynamic workload, and augment this model with a workload factoring service. De et al. [25] and Mattess et al. [26] similarly evaluate the cost-benefits of different strategies for scheduling workloads between a local cluster and a public



(a) Monetary cost (b) Inter-cloud bandwidth (c) Average end-to-end delay

Figure 9. Experimental result of prototype evaluation.

cloud. Zhang et al. [27] present a generic framework for secure MapReduce computation on mixed-sensitivity data, with various scheduling modes proposed to improve the efficiency and reduce cost. In addition, Neal et al. [2] investigate the possibility of moving video surveillance systems to the cloud and conclude that it is more expensive and requires additional reviews for legal implications as well as security threats. We remark that with the hybrid cloud model and effective scheduling, the above issues could be significantly mitigated.

VII. Conclusions

The hybrid of a trusted private cloud and an elastic public cloud naturally addresses the issues on security and seasonal workload in large video surveillance systems. Nevertheless, to fully utilise the potential of the hybrid setting, it is desired to have an effective scheduler, so as to reduce cost and enhance usability. We proposed a scheduler that exploits the two-server setting, and gave empirical result to show that, with an effective scheduler, it is feasible to process large-scale mixed-sensitivity video streams in the cloud. The costs are much lower than a pure public cloud deployment, with overheads smaller than other alternatives. For future work, it would be interesting to employ existing techniques of fast operation migration [6], [13] to facilitate realtime rescheduling.

References

[1] H. Zhang, G. Jiang, K. Yoshihira, H. Chen, and A. Saxena, "Intelligent workload factoring for a hybrid cloud computing model," in *SERVICES*, 2009.

[2] D. Neal and S. Rahman, "Video surveillance in the cloud?" *The Int'l Journal of Cryptography and Information Security (IJCIS)*, 2012.

[3] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *CCS*, 2009.

[4] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009.

[5] "2012 Cloud Computing Survey," http://northbridge.com/2012-cloud-computing-survey, 2012.

[6] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. B. Zdonik, "Scalable distributed stream processing," in *CIDR*, 2003.

[7] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine," in *CIDR*, 2005.

[8] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *ICDMW*, 2010.

[9] B. Lohrmann, D. Warneke, and O. Kao, "Massively-parallel stream processing under QoS constraints with nephele," in *HPDC*, 2012.

[10] G. T. Lakshmanan and R. E. Strom, "Biologically-inspired distributed middleware management for stream processing systems," in *Middleware*, 2008.

[11] M.-T. Yang, R. Kasturi, and A. Sivasubramaniam, "A pipeline-based approach for scheduling video processing algorithms on NOW," *IEEE Transactions on Parallel and Distributed Systems*, 2003.

[12] Y. Ahmad and U. Çetintemel, "Network-aware query processing for stream-based applications," in *VLDB*, 2004.

[13] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *ICDE*, 2006.

[14] L. Amini, N. Jain, A. Sehgal, J. Silber, and O. Verscheure, "Adaptive control of extreme-scale stream processing systems," in *ICDCS*, 2006.

[15] G. T. Lakshmanan, Y. G. Rabinovich, and O. Etzion, "A stratified approach for supporting high throughput event processing applications," in *DEBS*, 2009.

[16] M. Balazinska, H. Balakrishnan, and M. Stonebraker, "Contract-based load management in federated distributed systems," in *NSDI*, 2004.

[17] "Amazon EC2 Pricing," http://aws.amazon.com/ec2/pricing/, Sept. 2012.

[18] H. Mittelmann, "Mixed Integer Linear Programming Benchmark (MIPLIB2010)," http://plato.asu.edu/ftp/milpc.html, Oct. 2013.

[19] "Storm: Distributed and fault-tolerant realtime computation," http://storm.incubator.apache.org/, Oct. 2013.

[20] J. Zheng, G. A. Ramírez, and O. Fuentes, "Face detection in low-resolution color images," in *Image Analysis and Recognition*, 2010.

[21] L. A. Wolsey, "Integer programming," *IIE Transactions*, 2000.

[22] Y. Drougas, T. Repantis, and V. Kalogeraki, "Load balancing techniques for distributed stream processing applications in overlay environments," in *ISORC*, 2006.

[23] S. Seshadri, V. Kumar, and B. F. Cooper, "Optimizing multiple queries in distributed data stream systems," in *ICDEW*, 2006.

[24] G. T. Lakshmanan, Y. Li, and R. Strom, "Placement of replicated tasks for distributed stream processing systems," in *DEBS*, 2010.

[25] M. D. De Assunção, A. Di Costanzo, and R. Buyya, "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *HPDC*, 2009.

[26] M. Mattess, C. Vecchiola, and R. Buyya, "Managing peak loads by leasing cloud infrastructure services from a spot market," in *HPDC*, 2010.

[27] C. Zhang, E.-C. Chang, and R. H. C. Yap, "Tagged MapReduce: A general framework for secure computing with mixed-sensitivity data on hybrid clouds," in *CCGrid*, 2014.