ViCo: An Adaptive Distributed Video Correlation System

Xiaohui Gu, Zhen Wen, Ching-Yung Lin, Philip S. Yu
IBM T. J. Watson Research Center
Hawthorne, NY 10532
{xiaohui, zhenwen, chingyung, psyu}@ us.ibm.com

ABSTRACT

Many emerging applications such as video sensor monitoring can benefit from an on-line video correlation system, which can be used to discover linkages between different video streams in realtime. However, on-line video correlations are often resource-intensive where a single host can be easily overloaded. We present a novel adaptive distributed on-line video correlation system called ViCo. Unlike single stream processing, correlations between different video streams require a distributed execution system to observe a new correlation constraint that any two correlated data must be distributed to the same host. ViCo achieves three unique features: (1) *correlation-awareness* that ViCo can guarantee the correlation accuracy while spreading excessive workload on multiple hosts; (2) *adaptability* that the system can adjust algorithm behaviors and switch between different algorithms to adapt to dynamic stream environments; and (3) *fine-granularity* that the workload of one resource-intensive correlation request can be divided and distributed among multiple hosts. We have implemented and deployed a prototype of ViCo on a commercial cluster system. Our experiment results using both real videos and synthetic workloads show that ViCo outperforms existing techniques for scaling-up the performance of video correlations.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications; H.1.1 [**Information Storage and Retrieval**]

General Terms

Design, Algorithms, Performance

Keywords

Video correlation, Adaptive stream processing

1. INTRODUCTION

Video streams have become prevalent with the emergence of video sensor networks [5, 12], web cameras, and Internet TVs. Many applications such as sensor monitoring call for on-line continuous analysis over video streams that can be produced at distributed locations. In these applications, distributed video streams are continuously pushed into a stream processing system, where different processing functions (e.g., classification[2], similarity comparison [22]) are continuously applied to the input streams. Compared to traditional off-line analysis systems, on-line processing facilitates realtime decision-making and eliminates excessive storage requirements by continuously processing video streams as they flow into the system. To support unbounded streams, the stream processing system often associates a sliding-window with each stream. The window contains the most recently arrived data items on the stream. The window can be either time-based (e.g., video frames arrived in the last 60 seconds) or number-based (e.g., the last 1000 video frames). One of the most important video processing functions is *sliding-window correlation* between different video streams. The output of the correlation contains all pairs of video frames that satisfy a correlation predicate and are simultaneously present in their respective windows. On-line video correlations have many interesting applications, such as (1) correlating different news video streams for hot topic detection; (2) correlating video streams from multiple views for constructing environment models; and (3) correlating multiple surveillance video streams for abnormal events identification and verification.

One major challenge in processing on-line video correlations is to handle continuous, high-volume, and time-varying stream workload under system resource constraints. A single host can be easily overloaded by the video correlation workload where each video frame has to compare with a number of video frames on the other stream in realtime. Previous work has proposed load shedding techniques (e.g., [10, 20]) to intelligently drop some "unimportant" data based on the system predictions. However, data dropping can affect the accuracy of stream correlations (i.e., some correlation results can be missing). Thus, the goal of our work is to provide an efficient distributed execution solution for on-line video

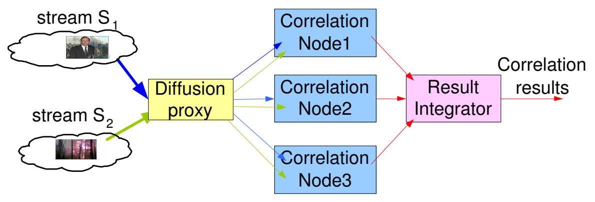


Figure 1: Distributed on-line video correlation.

correlations, which is illustrated by Figure 1. Video streams from different distributed locations are continuously pushed into the system for correlation processing. We provide a load diffusion proxy service that can efficiently spread the correlation workload among multiple hosts. Although load distribution has been extensively studied in conventional distributed environments (e.g.,[13]), we are motivated to re-examine the problem for video correlations by two key observations. First, stream correlations impose an additional *correlation constraint* for the distribution system: any two frames that need to be correlated must be distributed to the same host exactly once. Second, distributed stream environments are highly dynamic where both data arrival rates and computation workloads can dynamically change over time. The change of data arrival rates can be caused by video scene changes or network delay jitters when data arrive from remote sites. The computation workloads can vary over time when users require different video comparison algorithms or the video scenes experience changes. Thus, the distribution system must perform fast, on-line adaptations for maintaining optimal on-line video correlation performance.

In this paper, we present the design and implementation of a novel distributed on-line video correlation system called ViCo. We first theoretically prove that any distributed correlation scheme has to replicate some frames on multiple hosts in order to preserve the correlation accuracy, which is called *diffusion overhead*. Thus, the goal of our design is to achieve minimum diffusion overhead while balancing the correlation workload among different hosts. The ViCo system employs a diffusion proxy to adaptively dispatch input video frames to different hosts for correlation processing based on both stream correlation constraint and host load conditions. We propose two light-weight correlation-aware stream partition algorithms for the load diffusion proxy. The stream partition algorithm dynamically splits a high volume input stream into multiple sub-streams, each of which only contains a subset of data on the original input stream. The data on these sub-streams are routed to different hosts for concurrent correlation processing. Hence, each host only shares a partial workload since it only processes a subset of data on the original stream. We prove that both stream partition algorithms can preserve the accuracy of stream correlations and analyze their theoretical properties in terms of diffusion overhead. We then present a spectrum of on-line adaptation algorithms based on our diffusion overhead analysis. We first describe the *micro-adaptation* algorithms that can dynamically adjust the distribution behaviors within each stream partition algorithm. We then present the *macro-adaptation* algorithms that can dynamically switch between the two stream partition algorithms. We propose a novel data marking technique to assure that the on-line adaptation algorithms can still preserve the correlation constraint.

We have implemented a prototype of the ViCo system on top of our distributed stream processing infrastructure called System S [11, 7] that can perform scalable large-scale stream processing using hundreds of commercial IBM blade servers. We also implemented a news video correlation application on top of the ViCo system. We conduct extensive experiments using both real video data and a range of different synthetic workloads. Our experiments show several interesting results: (1) the load diffusion algorithms can achieve much higher correlation throughput than conventional distribution schemes; (2) the diffusion proxy is light-

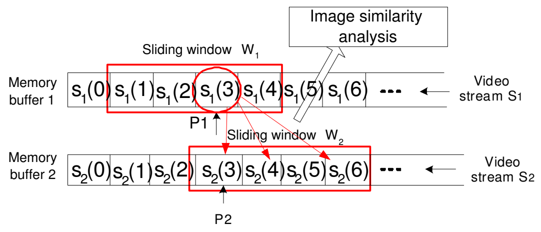


Figure 2: On-line video correlation using sliding-window stream join.

weight and fast, which can dispatch a video frame within tens of micro-seconds on a commercial server host; and (3) the performance of the diffusion algorithms is sensitive to stream environment changes and the adaptation algorithms can effectively achieve much better performance than static diffusion algorithms. The rest of the paper is organized as follows. Section 2 introduces the system model and the problem definition. Section 3 presents the detailed design and algorithms of our distributed video correlation system. Section 4 presents the prototype implementation and experimental results. Section 5 reviews related work. Finally, the paper concludes in Section 6.

2. PRELIMINARY

In this section, we first present the on-line video correlation application model followed by the centralized video correlation algorithm. We then present the formal definition of the distributed video correlation problem.

2.1 On-line Video Correlation

On-line video correlations perform continuous comparisons between two different video streams, illustrated by Figure 2. A video stream, denoted by S_i, consists of a sequence of video frames denoted by $s_i \in S_i$. We assume that each frame $s_i \in S_i$ carries a time-stamp, denoted by $s_i.t$, to record the time when the frame arrives on the stream S_i. We use r_i to denote the average arrival rate of the stream S_i. The input video streams can be either local archived video streams or live video streams from remote sites (e.g, video sensors). Thus, the stream arrival rate can be affected by several factors such as the video source production speed, the network delays, and scene changes. For example, a powerful video server can produce video streams much faster than a resource-constrained video sensor. The arrival rate of a stream can also dynamically change over time due to network delay jitters (from the remote video source to the processing host) and dynamic scene changes. To support unbounded streams, we associate a sliding-window $S_i[W_i]$ with each video stream S_i that contains the most recently arrived video frames[1]. The on-line video correlation can be performed by a sliding-window correlation between two streams[2] S_1 and S_2 over a correlation predicate θ, denoted by $C_i = S_1[W_1] \bowtie_\theta S_2[W_2]$. The correlation output consists of all pairs of video frames (s_1, s_2), such that (1) s_1 and s_2 satisfy the correlation predicate θ and (2) s_1 and s_2

[1]For illustration, we only discuss time-based windows. The frame-based windows can be translated into time-based windows.

[2]In this paper, we focus on the two-way stream correlations. We can extend our scheme to support multi-way correlations, which is beyond the scope of this paper.

belong to their respective sliding windows. The correlation predicate can be configured as finding similar or dissimilar images according to the user's requirements. For example, a news video correlation application may want to find similar images between different news videos for hot topic detection while a detailed video analysis system may want to filter out similar images for workload reduction and only perform detailed analysis on distinct video frames. The video data arrived at the video correlation system can be MPEG raw data, low-level feature vectors (e.g., color features), or high-level concept values (e.g., people, aircraft). Based on the input data content, the similarity analysis can be performed using different content analysis algorithms (e.g., [16, 22, 2]). To reduce the number of correlation computations, we perform shot detection based on low-level features (e.g., color features) and only correlate the key frames of the two input streams.

2.2 Centralized Video Correlation

The video correlation can be performed on a single host using the following centralized algorithm. The host maintains two memory buffers B_1 and B_2 for caching incoming video frames from two input streams S_1 and S_2 since on-line correlation requires in-memory processing. We use $s_i(t)$ to denote a frame that arrives at S_i at time t. The video frames are processed in a temporal order, i.e., if s_1 in the buffer B_1 arrives before s_2 in the buffer B_2, s_1 is processed first. For example, in Figure 2, $s_1(3)$ is processed before $s_2(4)$. Each buffer $B_i, i = 1, 2$ maintains a pointer $P_i, i = 1, 2$ to refer to the frame currently being processed by the correlation operator. The basic steps for processing a video frame $s_1 \in B_1$ include: (1) update the memory buffer B_2 by removing expired data s_i that arrived earlier than $s_1.t - W_2$ and has been processed by the correlation operator; (2) produce correlation results between s_1 and $S_2[W_2]$ by evaluating the correlation predicate between s_1 and $\forall s_2 \in S_2[W_2]$; (3) update the pointer P_1 to refer to the next frame in B_1. A symmetric procedure is followed for processing a frame $s_2 \in B_2$.

2.3 Problem Formulation

On-line stream correlations are often extremely resource-intensive since they need to perform a large number of correlations over continuously arrived data in realtime [20, 10]. Thus, a single host can be easily overloaded by the correlation workload. The problem exacerbates in the video correlation case since video analysis is often more resource-intensive than ordinary data analysis. Thus, it is necessary to provide a distributed solution for scaling-up the on-line video correlation processing. However, unlike traditional workload distribution where data can be arbitrarily distributed on different hosts, stream correlation requires the distribution scheme to observe an additional *correlation constraint* in order to preserve the accuracy of correlation results. The correlation constraint requires that any two data items that need to be correlated must be distributed to the same host once and only once. This constraint implies that the correlation workload cannot be divided cleanly into independent partitions. In fact, we have proved that distributed video correlations must replicate some frames on multiple hosts in order to preserve correlation accuracy. The proofs of all the theorems in this paper can be found in appendix.

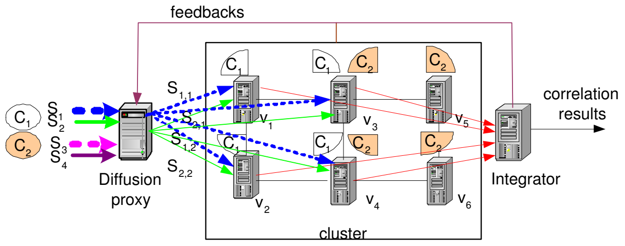


Figure 3: Distributed video correlation architecture.

THEOREM 2.1. *To distribute the workload of a stream correlation $S_1[W_1] \bowtie_\theta S_2[W_2]$ on multiple hosts, there must exist some video frames that are replicated on multiple hosts for preserving the correlation accuracy.*

Thus, a distributed stream correlation system has to replicate some video frames on multiple hosts, which are called the *diffusion overhead*. These overhead frames can consume CPU time, memory space, and network bandwidth in the system. Thus, the goal of our distributed execution scheme is to achieve minimum replication overhead while balancing the stream correlation workload among different hosts, which is formally defined as follows,

DEFINITION 2.1. **Distributed stream correlation problem:** *given a correlation operator $C_i = S_1[W_1] \bowtie_\theta S_2[W_2]$ and m hosts $\{v_1, ..., v_m\}$, each data item is distributed to one or more hosts such that (1) the correlation constraint is satisfied, (2) the workload at different hosts is balanced, and (3) the diffusion overhead is minimized.*

Different from traditional distributed systems, streaming environments are highly dynamic where data can arrive in bursty and unpredictable fashions. The stream correlation workload can dynamically change over time. As we will describe in the next section, there are different ways to achieve distributed stream correlations. The relative merit of different algorithms depends on the properties of the input streams (e.g., relative stream rates). Thus, the system must be adaptive in order to achieve optimal distributed video correlation performance in dynamic stream environments.

3. DESIGN AND ALGORITHMS

In this section, we first present an overview of our distributed video correlation system. We then describe two different stream partition algorithms to achieve correlation-aware load diffusion, followed by a set of on-line adaptation algorithms for maintaining optimal video correlation performance in dynamic stream environments.

3.1 Approach Overview

The ViCo distributed video correlation system is a multi-tier distributed system consisting of a load diffusion proxy, a set of worker nodes, and a result integrator, illustrated by Figure 3. The load diffusion proxy serves as the gateway of the system to distribute video correlation workload across all servers. For each video correlation request, the load diffusion proxy selects a number of worker nodes to instantiate the sliding-window correlation operator that implements the video correlation. The load diffusion proxy intercepts input streams and re-directs them to proper servers responsible for handling the stream correlations. In contrast, the result integration proxy aggregates the dispersed correlation results into complete correlation answers based on the Join-ID attached to each data item. Due to the memory and CPU speed limits, a single worker node can only accommodate up to a certain data arrival rate in order to keep the unprocessed data in the memory. When frames arrive too fast, the worker node has to drop frames using some load shedding technique (e.g., [10]). However, dropping data can affect the accuracy of stream correlation results. Thus, the goal of our load diffusion scheme is to avoid dropping data as much as possible by spreading stream correlation workload across multiple servers. Compared to video correlation processing, the operation performed by the load diffusion proxy is much simpler and faster[3]. Thus, the load diffusion proxy is not the bottleneck[4]

The diffusion proxy is the major controller in the ViCo system, which performs two-level load distribution: (1) *inter-operator* distribution where the diffusion proxy first selects a *worker set* $\{v_1, ..., v_k\}$ for processing a video correlation request; and (2) *intra-operator* distribution where the diffusion proxy can adaptively distribute the workload of a single correlation request among the hosts in its worker set. Different correlation operators can have overlapped worker sets (i.e., including common worker nodes), which allows them to share resources on the same host. For example, in Figure 3, the correlation operator C_1 is instantiated on the server set $\{v_1, v_2, v_3, v_4\}$ and the correlation operator C_2 is instantiated on the server set $\{v_3, v_4, v_5, v_6\}$. The load diffusion proxy can use different algorithms for diffusing the workload of C_1 and C_2. This hierarchical approach has several advantages: (1) it allows each sliding-window correlation operator to scale independently by adding more resources to its worker set; (2) it allows each sliding-window correlation operator to adapt independently to the changes of its input streams; and (3) it allows different operators to share resources on the same server by supporting overlapped server set. The inter-operator load distribution is similar to the conventional load distribution problem, which can be addressed using conventional resource management mechanisms (e.g., [21]). In contrast, the intra-operator load diffusion needs to observe the new correlation constraint, which is the focus of our work.

To allow a single correlation operator to be executed on multiple hosts, the diffusion proxy adaptively split a high-volume stream into multiple sub-streams, each of which are sent to different worker nodes for concurrent processing. Conceptually, the load diffusion proxy decomposes a resource-intensive correlation operator into multiple sub-operators executed on different worker nodes. Each sub-operator only processes a subset of frames on the original input streams. For example, in Figure 3, the diffusion proxy splits the input stream S_1 into four sub-streams $S_{1,1}$, $S_{1,2}$, $S_{1,3}$, $S_{1,4}$ that are sent to the worker node v_1, v_2, v_3, and v_4, respectively. The load diffusion proxy can use different stream partition algorithms for different correlation operators based on the properties of its input streams. Such fine-grained load partition has two advantages: (1) it allows a single resource-intensive correlation operator that cannot be executed by any single host to utilize aggregated resources on multiple hosts; and (2) it enables multiple correlation operators to more efficiently share resources at fine-granularity.

The load diffusion proxy performs runtime adaptations at different levels to achieve optimal performance in dynamic stream environments. The adaptations are informed by the feedbacks from the worker nodes, the result integrator and the users. First, the workload of a correlation operator can increase when its input stream rates increase or a more complicated video analysis algorithm is employed. Thus, the diffusion proxy can dynamically add or reduce the worker set to adapt to such workload changes. Second, the diffusion proxy can adaptively adjust stream partition algorithm parameters or switch between different stream partition algorithms to maintain optimal performance in dynamic stream environments. The performance of different stream partition algorithms are affected when the system changes the worker set allocated to the correlation operator, or the user changes the correlation requirements (e.g., sliding-window size). Finally, the diffusion proxy can also switch between different video analysis algorithms based on the feedbacks from the users or the result integrator.

3.2 Single Stream Partition

We now present a simple *single stream partition* (SSP) algorithm that splits one stream for load balancing and replicates the other stream for observing the correlation constraint. The partitioned stream is called the *master stream* and the replicated stream is called the *slave stream.* Let us consider a correlation operator $C_i = S_1[W_1] \bowtie_\theta S_2[W_2]$ that is instantiated on a host set $\{v_1, ..., v_k\}$. For each frame on the master stream, SSP sends the frame to one of the allocated k hosts based on the least-loaded-first (LLF) policy. Since we need to consider different resources (e.g., CPU, memory, bandwidth) in the distributed stream processing system, we define a combined metric w_i to represent the load condition of a host v_i. For each resource type R_i (e.g., CPU, memory, network bandwidth), we define a load indicator $\phi_{R_i} = \frac{U_{R_i}}{C_{R_i}}$, where U_{R_i} and C_{R_i} denote the usage and capacity of the resource R_i on the host v_i, respectively. We then define the load metric L_i as follows,

$$L_i = \omega_1 \cdot \phi_{cpu} + \omega_2 \cdot \phi_{memory} + \omega_3 \cdot \phi_{bandwidth} \qquad (1)$$

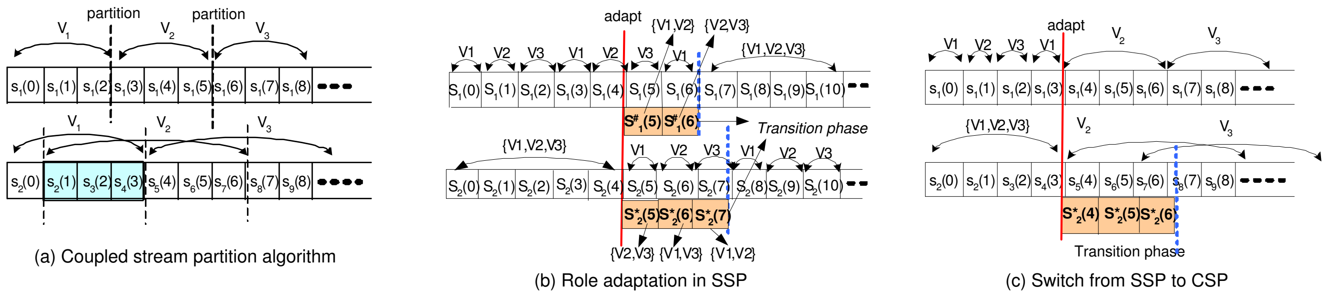
where $\sum_{i=1}^{3} \omega_i = 1, 0 \leq \omega_i \leq 1$ denotes the importance of different resource types that can be dynamically configured by the system[5]. For each frame on the slave stream, SSP sends one copy of the frame to each of the k hosts. By partitioning the master stream, the workload of the correlation operator is distributed among all k hosts since each host only processes a subset of required correlation operations. We have proved that SSP algorithm can preserve the correlation accuracy by observing the correlation constraint.

Theorem 3.1. *Let $\Theta(C_i)$ and $\Theta'(C_i)$ denote the sets of correlation operations performed by the original correlation operator and the distributed correlation operator using the SSP algorithm, respectively. We have $\Theta(C_i) = \Theta'(C_i)$.*

[3]In our experiments, the processing time of the load diffusion proxy is often several orders of magnitude less than the correlation computation

[4]Moreover, the load diffusion function can be easily replicated, with the only constraint that the pair of streams belonging to the same correlation operator pass through the same load diffusion proxy.

[5]We can assign higher weight to CPU resource if the join computation is CPU-bound or assign higher weight to network resource if network bandwidth is limited (e.g., sensor networks).



(a) Coupled stream partition algorithm
(b) Role adaptation in SSP
(c) Switch from SSP to CSP

Figure 4: Correlation-aware load distribution and adaptations.

Since the number of total correlation operations is not changed by the SSP algorithm, each server in $\{v_1, ..., v_k\}$ only processes on average one $k'th$ of the original correlation operations. We now analyze the overhead of the SSP algorithm. Since SSP replicates each frame of the slave stream on all allocated hosts, the diffusion proxy pushes more data into the cluster than the original slave stream. The data of the master stream are unchanged. We define the overhead of the SSP algorithm as the number of the additional video frames produced per time unit compared to the centralized video correlation algorithm. Let r_2 denote the rate of the slave stream S_2. Let k denote the host number. Let O_{ssp} denote the overhead of the SSP algorithm. We have

$$O_{ssp} = (k-1) \cdot r_2 \tag{2}$$

We can easily derive the CPU, memory and network bandwidth cost of SSP from the above overhead data number. Let α denote the average frame size, β denote the average network packet size for each frame, and γ denote the average CPU cost for receiving one frame from the network. Then, SSP incurs on average $O_{SSP} \cdot \alpha$ memory cost, $O_{SSP} \cdot \beta$ network cost, and $O_{SSP} \cdot \gamma$ CPU cost in the system.

3.3 Coupled Stream Partition

We now describe a coupled stream partition (CSP) algorithm that splits the input streams simultaneously, which is illustrated by Figure 4 (a). Similar to SSP, CSP also selects one stream as master stream and the other stream as slave stream. The master stream is partitioned into *disjoint* stream segments that are dispatched to different hosts based on the pre-defined load balancing policy. In contrast, the slave stream is partitioned into *overlapped* segments in order to meet the correlation constraint. More specifically, let us consider a correlation operator $C_i = S_1[W_1] \bowtie_\theta S_2[W_2]$. We use $S_i[t, t+T]$ to denote a segment of S_i including all the frames arrived on S_i between time[6] $[t, t+T)$, where t is called the segment's start time and T is called the segment length. When the diffusion proxy receives a video frame s_1 on the master stream, it first checks whether s_1 belongs to the current segment or starts a new segment based on its time-stamp $s_1.t$ and the end time of the current segment. All the frames belonging to the current segment $S_1[t, t+T]$ are sent to the same host based on the load metric L_i (Equation 1). The CSP algorithm sends each frame on the master stream only once since its segments do not have any overlap with each other. In contrast, CSP splits the slave stream into overlapped segments.

[6]The purpose of this definition is to avoid overlapping between $S_i[t, t+T]$ and $S_i[t+T, t+2T]$.

If CSP sends the segment $S_1[t, t+T]$ to the host v_i, CSP needs to send the slave stream segment $S_2[t-W_2, t+T+W_1]$ to the same server v_i. The two segments $S_1[t, t+T]$ and $S_2[t-W_2, t+T+W_1]$ are called *coupled segments* that are sent to the same host for producing correlation results. Similarly, if CSP sends the master stream's next segment $S_1[t+T, t+2T]$ to the server v_j, CSP needs to send the slave stream segment $S_2[t+T-W_2, t+2T+W_1]$ to the same server v_j. Thus, the frames arrived on S_2 during the period $[t+T-W_2, t+T+W_1]$ are sent to both v_i and v_j. The number of duplicate frames is $r_2 \cdot (W_1 + W_2)$. Figure 4 (a) shows a simple example where $r_1 = r_2 = 1$, $W_1 = 1$, $W_2 = 2$. The master stream S_1 is partitioned into three disjoint segments $S_1[0, 3]$, $S_1[3, 6]$, $S_1[6, 9]$ that are sent to the hosts v_1, v_2, and v_3, respectively. Correspondingly, the slave stream S_2 is partitioned into three overlapped segments $S_2[0, 4]$, $S_1[1, 7]$, $S_1[4, 10]$ that are sent to the server v_1, v_2, and v_3, respectively. The video frames $s_2(1)$, $s_2(2)$, $s_2(3)$ are replicated on both v_1 and v_2. We have also proved that CSP can preserve the accuracy of stream correlations.

THEOREM 3.2. *Let $\Theta(C_i)$ and $\Theta'(C_i)$ denote the sets of correlation operations performed by the original correlation operator and by the distributed correlation operator using the CSP algorithm, respectively. We have $\Theta(C_i) = \Theta'(C_i)$.*

According to the above theorem, CSP can reduce the average workload of each host in the worker set into one $k'th$ of the original workload. However, CSP also pushes extra video frames into the cluster due to the partial replication of the slave stream. Let r_2 denote the rate of the slave stream S_2. Let T denote the segment length. Let W_1 and W_2 denote the sliding window sizes of the stream S_1 and S_2. For each segment $S_1[t, t+T]$ over the time period T, the CSP algorithm introduces $r_2 \cdot (W_1 + W_2)$ more frames than the original input streams. Thus, let O_{csp} denote the overhead of the CSP algorithm, we have

$$O_{csp} = \frac{(W_1 + W_2)}{T} \cdot r_2 \tag{3}$$

Similar to the SSP algorithm, we can easily derive the CPU, memory and network bandwidth cost of the CSP algorithm based on the overhead data number.

3.4 Adaptation Algorithms

We now present a set of adaptation algorithms that can dynamically adjust the behaviors of SSP and CSP to make them maintain optimal performance in dynamic stream environments. Equation 2 indicates that the overhead of SSP is only decided by the rate of the slave stream. For minimum overhead, SSP should always select the higher rate stream as the master stream and the lower rate stream as the slave stream. Due to video scene changes and network delay jitters from the remote stream sources, the arrival rates of two correlated streams can dynamically change over time. Thus, our first micro-adaptation algorithm is to dynamically switch the roles of master stream and slave stream between the two input streams, which is illustrated by Figure 4 (b). For easy illustration, we use a simple case where $r_1 = r_2 = 1$, $W_1 = 3$, $W_2 = 2$, and host set $V = \{v_1, ..., v_3\}$. In practice, the stream role adaptation is triggered only when the slave stream rate r_2 becomes faster than the master stream rate r_1. At time 5, the system decides to switch the stream roles between S_1 and S_2. After time 5, the frames of S_1 are replicated on all allocated hosts while the frames of S_1 is distributed among different hosts.

However, a brute-force stream role switching can miss some correlation results or produce duplicate correlation results. For example, in Figure 4 (b), the frame $s_2(5)$ needs to correlate with the frame $s_1(4)$. However, they are distributed to different hosts v_1 and v_2, respectively. Moreover, correlation results between $s_1(5)$ and $s_2(4)$ are duplicated[7] on all hosts since S_1 becomes the replicated slave stream after time 5 while S_2 is the replicated slave stream before time 5. To address the problem, we propose a novel data marking scheme to assure that the stream role adaptation preserves the correlation constraint. Let t_s denote the role switching time (e.g., $t_s = 5$ in Figure 4 (b)). The system generates a set of marked frames during a transition phase. The transition phase of S_1 covers the time period $[t_s, t_s+W_2)$ while the transition phase of S_2 is defined as $[t_s, t_s + W_1)$.

For each frame s_1 arrived during the transition phase, we send a frame s_1 to the host selected by the ordinary SSP algorithm and an additional marked copy $s_1^{\#}$ to all the other hosts. The marked frame $s_1^{\#}$ only correlates with unmarked frames of S_2 arrived after t_s. In contrast, for each frame s_2 arrived during the transition phase of S_2, we send an ordinary copy to one selected host using the ordinary SSP algorithm and a marked frame s_2^* to all the other hosts. However, the marked frame s_2^* only correlates with ordinary data s_1 arrived before the switching time t_s. After the transition phase, the system restores to the ordinary SSP operations. The rationale behind our approach is that the marked frames are replicated on all hosts to avoid missing correlation results. To avoid duplicate correlation results, the marked frames do not correlate with those frames that their ordinary copies can cover. For example, in Figure 4 (b), after the adaptation is triggered at time 5, we send $s_1(5)$ and $s_1(6)$ to the selected host using the ordinary SSP algorithm but also send three marked frames of $s_1^{\#}(5)$ and $s_1^{\#}(6)$ to all the other hosts for correlating with the frames of S_1 whose frames are distributed on different hosts after time 5. We have rigorously proved that the stream role adaptation algorithm preserves the correlation constraint.

THEOREM 3.3. *The SSP's stream role adaptation algorithm satisfies the correlation constraint.*

Similarly, the overhead of CSP is also merely decided by the rate of the slave stream (Equation 3), which should always use the high-rate stream as the master stream and the low-rate stream as the slave stream. To preserve correlation accuracy, CSP also needs to send some marked frames during a transition phase. The data marking strategies are very similar to that of the SSP algorithm, which are omitted here due to space limitation. Different from SSP, CSP has another tunable parameter, the segment length T. Equation 3 indicates that larger segment length incurs less replication overhead. However, larger segment also implies coarser load balancing granularity since a larger number of frames are enforced to go to the same host. Thus, the segment length denotes the trade-off between the replication overhead and load balancing granularity. The optimal segment length should lie between its minimum and maximum values based on the current system conditions. For example, under high stream rates, we need to employ a smaller segment length to avoid including too many frames in one segment. Figure 5 shows the measured performance of the CSP algorithm as a function of segment length for two different correlation operators in our experiments. To adapt to dynamic stream changes, CSP employs the following profiling process to dynamically search for the optimal segment length T^* when the stream environment or user correlation specifications experience significant changes. Let T denote the current segment length and ΔT denote the adaptation step value. The adaptation algorithm tests both $T + \Delta T$ and $T - \Delta T$. If the performance of $T + \Delta T$ is better, the optimal segment length should be larger than the current segment length. The system gradually increases the segment length until the measured system performance reaches its peak value. Otherwise, if $T - \Delta T$ produces better performance, the system gradually decreases the segment length to search for the optimal value. The CSP algorithm always performs micro-adaptations at the end of one segment to assure that the adaptation does not violate the correlation constraint.

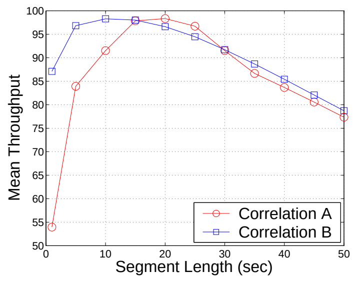


Figure 5: Segment length effect.

In addition to micro-adaptations within each algorithm, we can also perform macro-adaptations to switch between different algorithms. According to Equation 2 and Equation 3, SSP has larger overhead than CSP if

$$k > \frac{(W_1 + W_2 + T)}{T} \tag{4}$$

The above comparison indicates that the comparison between SSP and CSP depends on the size of the host set k, the sliding window sizes W_1 and W_2, and the segment length T. For example, if the workload of a correlation operator can be satisfied by a small number of hosts, the sliding window sizes are big, and the segment length is small, the SSP algorithm has less overhead than the CSP algorithm. Otherwise, the CSP algorithm is more cost-efficient. To achieve minimum replication overhead, our system performs macro-adaptation to dynamically switch between the SSP algorithm and CSP algorithm when the stream environment experiences changes. Similar to the micro-adaptations,

[7]Figure 4 (b) did not show the replication of S_1 between time [5,6].

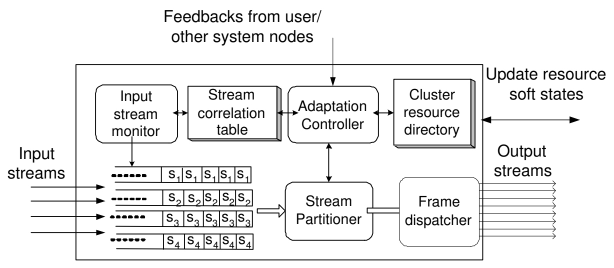


Figure 6: The diffusion proxy node structure.

macro-adaptations also need to employ a transition algorithm to preserve the correlation accuracy.

Let us consider the case of switching from SSP to CSP, illustrated by Figure 4 (c). For easy illustration, we use a simple case where $r_1 = r_2 = 1$, $W_1 = 1$, $W_2 = 3$, and the host set $V = \{v_1, ..., v_3\}$. At the beginning, the system employs the SSP algorithm where S_1 is the master stream and S_2 is the slave stream. At time 4, the system decides to switch to the CSP algorithm for lower overhead. The master stream S_1 is partitioned into disjoint segments while the slave stream S_2 is partitioned into overlapped segments. However, this brute-force algorithm switch can affect the correlation accuracy by missing some of the correlation results. For example, in Figure 4 (c), the frame $s_2(4)$ should correlate with $s_1(5)$, which however is distributed to two different hosts v_2 and v_1. To address the problem, we again introduce marked frames on the slave stream during its transition phase $[t_s, t_s + W_1)$. For each frame s_2 arrived at S_2 during its transition phase, we send an ordinary frame s_2 to the selected host for the current segment, and a marked frame s_2^* to each of the other $k-1$ hosts. The marked frame s_2^* only correlates with the master stream's frames arrived before the algorithm switching time t_s. We have proved that our macro-adaptation algorithm can preserve the correlation accuracy.

THEOREM 3.4. *The macro-adaptation algorithm for switching from SSP to CSP satisfy the correlation constraint.*

The case of switching from CSP to SSP is similar, which is omitted here due to space limitation.

4. EXPERIMENTAL EVALUATION

We now present an experimental evaluation of our systems using both prototype experiments with real cluster deployment and trace-driven simulations. The experiment results show that our approach can better scale-up video correlation processing than existing distribution schemes.

4.1 Evaluation Methodology

We have implemented a prototype of the ViCo distributed video correlation system (as shown by Figure 3) in about 10K lines of C++ code and successfully deployed it on a commercial cluster system. The load diffusion proxy consists of the following major modules illustrated by Figure 6: (1) *input stream monitor* that keeps track of the arrival rate of each input stream. The diffusion proxy keeps a counter for each input stream to record the number of arrived frames within a sampling period. The average arrival rate of the input stream can be estimated by dividing the counter value by the sampling period; (2) *stream correlation table* that records the specifications (e.g., sliding-window sizes, stream-IDs) of all currently running correlation operators; (3) *stream partitioner* that executes the SSP or CSP algorithms to split input streams; (4) *adaptation controller* that performs micro-adaptations and macro-adaptations based on the runtime monitoring and feedback information; (5) *cluster resource directory* that maintains dynamic information about load conditions (e.g., CPU, memory usages) of different cluster nodes and network bandwidth usage on the network links from the diffusion proxy to different cluster nodes; and (6) *frame dispatcher* that sends video frames to different cluster nodes for correlation processing based on the decisions made by the stream partitioner.

In addition to testing the prototype on the real cluster system, we also conduct trace-driven simulations to perform large-scale controllable experiments. The simulator consists of a workload generator, a diffusion proxy, a set of correlation operators and a result integrator. All these components are fully implemented. Only the hardware, network, and underlying CPU scheduler are simulated. We simulate a heterogeneous cluster where the memory space of each host is uniformly distributed in the range of [500,1000] MB, the CPU capacity of each host is distributed in the range of [500,1000] units, and the network bandwidth between cluster nodes is distributed in the range of [100,1000] Mbps. The workload generator can reproduce real application streams from trace data using specified parameters. The real application data used by our experiments are news videos (e.g., CNN, MSNBC, NTDTV) from the TRECVID-2005 data set. We compare our algorithms with existing load distribution algorithms:(1) *least-loaded-first distribution (LLF-Distribution)* algorithm that instantiates each correlation operator as a whole on the currently least-loaded host; and (2) *least-loaded-first diffusion (LLF-Diffusion)* algorithm that distributes each input frame to the currently least-loaded hosts without considering the correlation constraint. Compared to our load diffusion algorithms (i.e., SSP and CSP), the LLF-Distribution algorithm cannot divide the workload of a correlation operator and only perform coarse-grained load balancing among multiple correlation operators. The major goal of distributed correlation processing is to scale up the processing capacity to support complex correlation predicate computation over high-rate streams. Thus, we evaluate different algorithms using the *throughput* metric that is defined as the number of correlation comparisons performed by the system during a period time (e.g., every second or during the whole experiment run).

4.2 Cluster Prototype Experiments

We implemented a news video correlation application on top of the ViCo system and deployed it on a commercial cluster system. Each host in the cluster has an Intel Xeon 3.2GHZ CPU and 3G memory connected by gigabips networks. The experiment uses the news video correlation scenario illustrated by Figure 1 where all modules (including the two video sources) are instantiated on different cluster nodes. The diffusion proxy receives video streams from the two sources and then spreads each video frame to different hosts. Each host performs a set of operations for image correlation: (1) extract a 576-dimensional feature vector [?]; (2) perform shot detection based on the feature vector; (3) perform 40 concept classification on the fist frame of a shot; and (4) correlate two key frames based on the concept values. The correlation request is to discover similar images between

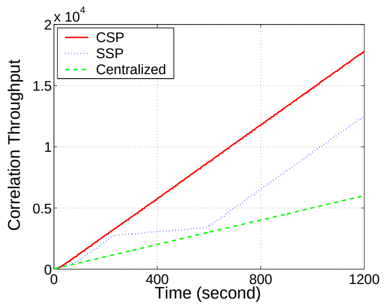


Figure 7: Prototype performance on a real cluster system.

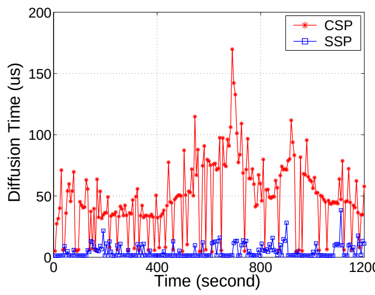


Figure 8: Prototype per-frame diffusion time on the cluster.

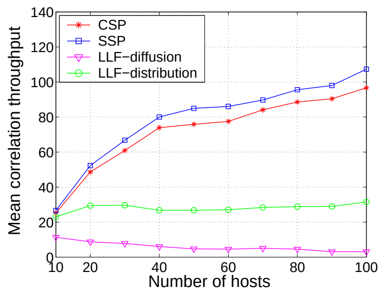


Figure 9: Scalability results under slow stream rates.

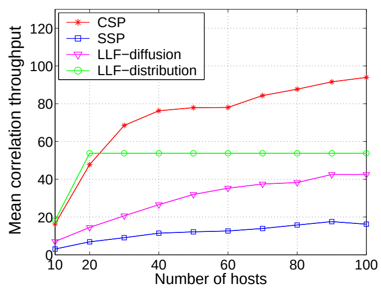


Figure 10: Scalability results under fast stream rates.

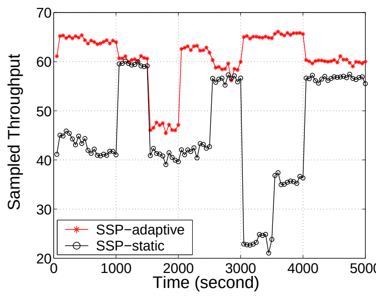


Figure 11: Adaptation results of SSP.

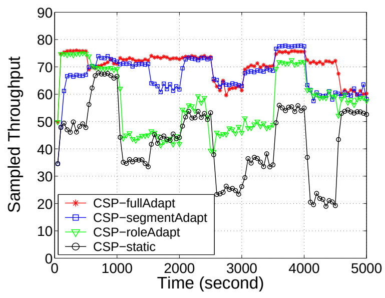


Figure 12: Adaptation results of CSP.

two real news videos (CNN and MSNBC) within two 60-second sliding windows. Figure 7 shows the measured total throughput of different algorithms during a 1200-second duration. The centralized algorithm runs the correlation operator on one cluster node while SSP and CSP can execute the correlation using three cluster nodes. The throughput value is sampled every second where the throughput at time t measures the total number of correlation comparisons performed by the cluster system from the beginning to the time t. We observe that both CSP and SSP can achieve higher throughput than the centralized processing. CSP performs better than SSP due to smaller diffusion overhead. SSP has little performance improvement since it is overloaded by the diffusion overhead. Figure 8 shows the measured per-frame diffusion time of CSP and SSP during the experiment. The diffusion time is sampled every second and is averaged over all the frames that are processed by the diffusion proxy during one second. We observe that both CSP and SSP have low diffusion time (i.e., tens of micro-seconds), which is several magnitudes less than the correlation operations (i.e., tens of or hundreds of milli-seconds).

4.3 Trace-Driven Simulation Experiments

We now present the performance of our systems under large-scale cluster system and different workloads using simulations. The workload generator can emulate remote video sources by adding a dynamic network delay to each video frame. The inter-arrival times follow an exponential distribution with a mean set to the current average data rate. The average CPU cost for receiving a frame from network is 10 units and processing one correlation comparison is 250 units. The sliding-windows are all set as 60 seconds. The video data are taken from the TRECVID-2005 data set. Each simulation run lasts 5000 seconds and has a certain warm-up period for the system to reach its stable performance after all the correlation operators start. Each throughput value is the average number of correlation comparisons performed by the system per second. Each average value is calculated over the whole simulation duration after the warm-up period. To achieve unbiased results, we repeat each experiment 5 times with different random seeds and report the average results.

We first test the scalability of our system by running 20 correlation operators concurrently on a cluster system with 10 to 100 hosts. Figure 9 shows the scalability comparison results between different algorithms under a slow-rate workload. The mean rates of all input streams are uniformly distributed in the range of [1,5] frames/second. This workload simulates the scenario that the remote video sources can perform some pre-processing on the raw video data and only send key frames to the correlation cluster. With diffusion algorithms (i.e., CSP or SSP), each correlation operator is allowed to use all the hosts in the cluster and different operators share the resources of one host proportionally. We observe that both CSP and SSP can achieve much better than the other two algorithms. The performance of the LLF-Distribution algorithm becomes nearly unchanged after the host number exceeds the correlation operator number since each correlation operator is allowed to use at most one host. In contrast, both CSP and SSP can utilize all the cluster hosts. SSP performs a bit better than CSP since (1) SSP employs finer-grained load balancing than the CSP algorithm; and (2) the diffusion overhead is small under low stream rates. The correlation-unaware load diffusion scheme LLF-Diffusion cannot properly scale-up the system throughput since many correlation comparisons are missing due to misplacing correlated frames on different hosts. However, since the stream rates are low, the diffusion overhead is insignificant compared to the correlation processing cost. We then repeat the above experiments under a high-rate workload where mean rates of input streams are uniformly distributed in the range of [10,30] frames/second. Figure 10 shows the throughput results achieved by different algorithms. We observe that both CSP and SSP achieve less throughput capacity than the previous case since the diffusion cost is increased due to higher stream rates. However, CSP still consistently performs much better than the other alternatives while SSP becomes the worst since its diffusion overhead becomes significant with too many data replications.

We conducted the second set of experiments to evaluate the efficiency of our adaptation algorithms. We execute one correlation operator on 10 hosts using either SSP or CSP algorithms. To emulate dynamic stream environments, the system dynamically changes the mean rates of both input streams every 500 seconds and the throughput value is sampled every 50 seconds. We first evaluate the micro-adaptation algorithm in SSP by dynamically switching the master and slave streams based on the stream rate changes, illustrated by Figure 11. We observe that the adaptive SSP consistently achieves higher throughput than the static algorithm and the improvement can be as much as three times better depending on the stream rates. Figure 12 shows the comparison results among the (1) fully adaptive CSP algorithm *CSP-fullAdapt* that performs both master/slave stream role switching and optimal segment length adjustment, (2) partially adaptive CSP: *CSP-segmentAdapt* that dynamically adjusts segment length but does not switch the master and slave streams, and *CSP-roleAdapt* that only switches master/slave streams but uses a fixed segment length $T = 10$ seconds, (3) the static CSP algorithm that does not switch master/slave streams and uses a fixed segment length $T = 10$ seconds. The results show that the performance of the CSP algorithm is sensitive to the stream rate changes and the adaptation strategies can effectively maintain optimal performance in dynamic stream environments.

5. RELATED WORK

Distributed multimedia processing has been extensively studied in prior work. For example, Amir et al. proposed the active service framework and applied it to a media transcoding gateway service[3]. Chandra et al. developed a quality-aware transcoding technique to enable differentiated multimedia web services [4]. Ooi and Renesse proposed a framework to decompose a media transformation computation into sub-computations and assign them to multiple gateways [17]. Compared to the transcoding service, video correlation needs to perform coordinated processing on multiple input streams, which presents new challenges to the system design. Our work is also related to various adaptation research (e.g., [14, 1, 6]). Different from previous work, our adaptation schemes are performed on continuous video streams and need to observe the correlation constraint.

Multimedia correlations. Measuring the similarity of key frames of video shots is a task that has been studied in many applications. For instance, in traditional content-based image retrieval, systems measure the similarity of images based on the feature vector distance[16]. Usually, color histogram, texture, edge and motion vectors (if key frames are extracted from video shots) are used as features. These features can be weighted based on user feedback using relevance feedback technique[18, 8]. Similarity can be also measured based on the combination of audio and video features[15]. Graph-based matching of images uses the relations of salient image parts to calculate the similarity of images[22]. Different from the above work, our research focuses on addressing the system infrastructure support to enable online, continuous video correlation processing.

Distributed join operation on stream data. The Flux project [19] supports parallel equijoin processing with dynamic value-based load balancing, which however cannot support non-equijoins required by video stream correlations. In contrast, our work supports both equijoins and non-equijoins. Ivanova and Risch proposed a customizable parallel execution platform for scientific stream queries [9], which did not consider the correlation constraint required by distributed execution of windowed stream joins.

6. CONCLUSION

In this paper, we have presented a novel adaptive distributed execution system for scalable processing of online video correlations. To the best of our knowledge, this is the first work that has addressed the problem of adaptive distributed executions of on-line video correlations. The major contributions of this paper are as follows. First, we formally define the optimal distributed stream correlation problem and theoretically prove that a diffusion overhead is unavoidable in order to preserve the accuracy of stream correlations. Second, we propose a set of correlation-aware, light-weight, stream partition algorithms that can dynamically distribute a video stream correlation workload among multiple hosts at fine-granularity. We theoretically prove both stream partition algorithms can preserve the accuracy of stream correlations and conduct theoretical analysis to derive their overhead models. Third, we propose a spectrum of on-line adaptation algorithms based on the overhead analysis that can dynamically adjust the behaviors within each algorithm and switch between different algorithms on-the-fly. We also provide a novel data marking technique to guarantee that on-line adaptations preserve the correlation accuracy. Finally, we have implemented a prototype of the system and conducted extensive experiments using real video and synthetic workloads. Our experimental results show the feasibility and efficiency of our approaches.

7. REFERENCES

[1] T. F. Abdelzaher, K. G. Shin, and N. T. Bhatti. User-Level QoS-Adaptive Resource Management in Server End-Systems. *IEEE Trans. Computers 52(5)*, 2003.

[2] A. Amir and et al. IBM Research TRECVID-2003 Video Retrieval System. *Proc. of NIST Text Retrieval Conference TRECVID Workshop*, Nov. 2003.

[3] E. Amir, S. McCanne, and R. H. Katz. An Active Service Framework and Its Application to Real-Time Multimedia Transcoding. *Proc. of SIGCOMM 1998*, Oct. 1998.

[4] S. Chandra, C. S. Ellis, and A. Vahdat. Differentiated Multimedia Web Services Using Quality Aware Transcoding. *IEEE INFOCOM*, 2000.

[5] W.-C. Feng and et al. Panoptes: A Scalable Architecture for Video Sensor Networking Applications. *ACM Multimedia*, 2003.

[6] D. Gotz and K. Mayer-Patel. A general framework for multidimensional adaptation. *ACM Multimedia*, 2004.

[7] X. Gu, P. S. Yu, and K. Nahrstedt. Optimal Component Composition for Scalable Stream Processing. *Proc. of IEEE*

International Conference on Distributed Computing Systems (ICDCS), June 2005.

[8] K. A. Hua, N. Yu, and D. Liu. Query Decomposition: A Multiple Neighborhood Approach to Relevance Feedback Processing in Content-based Image Retrieval. *ICDE*, 2006.

[9] M. Ivanova and T. Risch. Customizable Parallel Execution of Scientific Stream Queries. *Proc. of VLDB*, 2005.

[10] A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive Stream Resource Management Using Kalman Filters. *Proc of ACM SIGMOD*, June 2004.

[11] N. Jain, L. Amini, H. Andrade, R. King, Y. Park, P. Selo, and C. Venkatramani. Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core. *Proc. of SIGMOD*, 2006.

[12] P. Kulkarni, D. Ganesan, P. Shenoy, and Q. Lu. SensEye: A Multi-tier Camera Sensor Network. *ACM Multimedia*, 2005.

[13] V. Kumar, A. Y. Grama, and N. R. Vempaty. Scalable Load Balancing Techniques for Parallel Computers. *Journal of Parallel and Distributed Systems, 22:60-79*, 1994.

[14] B. Li and K. Nahrstedt. A Control-based Middleware Framework for Quality of Service Adaptations. *IEEE JSAC*, 1999.

[15] R. Lienhart, S. Pfeiffer, and W. Effelsberg. Scene Determination based on Video and Audio Features. *Proc. IEEE Conf. on Multimedia Computing and Systems*, 1998.

[16] W. Niblack and et. al. QBIC project: querying images by content, using color, texture, and shape. *Prof. SPIE, Vol. 1908, pp. 173-187, Storage and Retrieval for Image and Video Databases*, 1993.

[17] W. T. Ooi and R. V. Renesse. Distributing Media Transformation Over Multiple Media Gateways. *Proc. of ACM Multimedia*, Sept. 2001.

[18] Y. Rui, T. S. Huang, and S. Mehrotra. Content-based image retrieval with relevance feedback in MARS. *Proc of IEEE ICIP*, 1997.

[19] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin. Flux: An Adaptive Partitioning Operator for Continuous Query Systems. *Proc. of ICDE*, Mar. 2003.

[20] U. Srivastava and J. Widom. Memory Limited Execution of Windowed Stream Joins. *Proc. of VLDB*, Aug. 2004.

[21] B. Urgaonkar and P. Shenoy. Rsource Overbooking and Application Profiling in Shared Hosting Platforms. *Proc. of OSDI*, 2003.

[22] D.-Q. Zhang and S.-F. Chang. Detecting Image Near-Duplicate by Stochastic Attributed Relational Graph Matching with Learning. *Proc. of ACM Multimedia*, 2004.

APPENDIX

Proof sketch of Theorem 2.1: Let us consider a group of frames where each frame needs to correlate with at least one the other frame in this group. Assume to the contrary that any frame in this group is sent to only one host. Consider any two consecutive frames $s_i(t_i)$ and $s_j(t_j)$, $t_i \le t_j$. If $i \ne j$, since $s_j(t_j)$ must correlate with at least one frame in this group and $s_i(t_i)$ and $s_j(t_j)$ are two consecutive frames, we have $t_j - t_i \le W_i$. Thus, $s_j(t_j)$ should correlate with $s_i(t_i)$. Since $s_i(t_i)$ is sent to only one host, $s_j(t_j)$ must be sent to the same host as $s_i(t_i)$. If $i = j$, $s_j(t_j)$ needs to correlate with at least one the other frame $s_k(t_k)$ in this group. We have $0 \le t_j - t_k \le W_k$ and $t_k \le t_i$ since $s_i(t_i)$ and $s_j(t_j)$ are two consecutive frames. Since $t_i \le t_j$, we have $0 \le t_i - t_k \le W_k$. Thus, $s_k(t_k)$ should also correlate with $s_i(t_i)$. Since no frame is replicated, $s_k(t_k)$ and $s_i(t_i)$ must be located on the same host. Thus, $s_j(t_j)$ must be sent to the same host as $s_i(t_i)$. Thus, all the frames that have be correlated are sent to the same host, which becomes centralized correlation. Contradiction exists. Thus, replication is unavoidable.□

Proof sketch of Theorem 3.1: We first prove (1) $\Theta(C_i) \subseteq \Theta'(C_i)$ by showing that $\forall s_1$, if $s_1 \bowtie_\theta S_2[W_2] \in \Theta(C_i)$, then $s_1 \bowtie_\theta S_2[W_2] \in \Theta'(C_i)$, and $\forall s_2$, if $s_2 \bowtie_\theta S_1[W_1] \in \Theta(C_i)$, then $s_2 \bowtie_\theta S_1[W_1] \in \Theta'(C_i)$. Suppose SSP sends s_1 to the server v_i. Because SSP replicates the stream S_2 on all servers, $S_2[W_2]$ must be present on the server v_i, too. Thus, $s_1 \bowtie_\theta S_2[W_2] \in \Theta'(C_i)$. We now prove $\forall s_2$, if $s_2 \bowtie_\theta S_1[W_1] \in \Theta(C_i)$, then $s_2 \bowtie_\theta S_1[W_1] \in \Theta'(C_i)$. For any $s_2 \in S_2$, s_2 needs to correlate every frame in $S_1[W_1]$. Suppose SSP sends $s_1 \in S_1[W_1]$ to the server v_i. Because s_2 is also present at v_i, we have $(s_2, s_1) \in \Theta'(C_i)$. By aggregating all the results of $(s_2, s_1), \forall s_1 \in S_1[W_1]$, we have $s_2 \bowtie_\theta S_1[W_1] \in \Theta'(C_i)$. Thus, we have $\Theta(C_i) \subseteq \Theta'(C_i)$. We then prove (2) $\Theta'(C_i) \subseteq \Theta(C_i)$ by showing that $\forall s_1$, if $s_1 \bowtie_\theta S_2[W_2] \in \Theta'(C_i)$, then $s_1 \bowtie_\theta S_2[W_2] \in \Theta(C_i)$, and $\forall s_2$, if $s_2 \bowtie_\theta S_1[W_1] \in \Theta'(C_i)$, then $s_2 \bowtie_\theta S_1[W_1] \in \Theta(C_i)$. The proof is easy since any correlation operation in $\Theta'(C_i)$ follows the correlation specification, which thus should appear in $\Theta(C_i)$, too. Because $\forall s_1 \in S_1$, s_1 is only sent to one server, two different servers do not perform duplicate correlation operations. Thus, we have $\Theta'(C_i) \subseteq \Theta(C_i)$. Combining (1) and (2), we have $\Theta(C_i) = \Theta'(C_i)$. □

Proof of Theorem 3.2: The proof is similar to the proof of Theorem 3.1, which is omitted due to space limitation.□

Proof sketch of Theorem 3.3: First, we prove that SSP's role adaptation algorithm does not generate duplicate results. Assume to the contrary that $\exists s_1, \exists s_2 \in S_2[W_2]$, (s_1, s_2) appears on two different hosts. If s_1 is outside the transition phase, the duplicate s_1 only appears after $t_s + W_2$ while duplicate s_2 only appears before t_s. Thus, we have $s_1.t - s_2.t > W_2$, contradicting the sliding-window definition $s_1.t - s_2.t \le W_2$. If s_1 is within the transition phase, $s_1^{\#}$ only correlates with s_2 arriving after t_s, which does not have replica except the marked copy s_2^*. However, s_2^* only correlates with s_1 arrived before t_s. Thus, $(s_1, s_2), \forall s_2 \in S_2[W_2]$ does not have duplicate results. We now prove that $(s_2, s_1), s_1 \in S_1[W_1]$ does not appear on two different hosts. First, duplicate s_2 only appears before t_s while s_1 does not have replication before t_s. On the other hand, s_2^* only correlate with s_1 arrived before t_s. However, s_1 does not have duplication before t_s. Thus, SSP's role adaptation does not produce duplicate results for $s_2 \bowtie_\theta S_1[W_1]$. Second, we prove that SSP's role adaptation algorithm does not miss any correlation results. Any frames outside the transition phase will not be affected since they follow the original SSP algorithm. For $\forall s_1$ arrived during the transition phase (i.e., $s_1.t \in [t_s, t_s + W_2)$), s_1 needs to correlate with $S_2[s_1.t - W_2, s_1.t]$. The role adaptation algorithm assures that s_1 is co-located with $S_2[s_1.t - W_2, t_s]$ and $s_1^{\#}$ is co-located with $S_2[t_s, s_1.t]$. Thus, we get full coverage by combining the correlation results of s_1 and $s_1^{\#}$. For $\forall s_2$ arrived during the transition phase (i.e., $s_2.t \in [t_s, t_s + W_1)$), s_2 needs to correlate with $S_1[s_2.t - W_1, s_2.t]$. The adaptation algorithm assures that s_2^* is co-located with $S_1[s_2.t - W_1, t_s]$. For any frame in $S_1[t_s, s_2.t]$, either $s_1^{\#}$ or s_1 is replicated on all hosts. Thus s_2 must co-located with $S_1[t_s, s_2.t]$. Combining (1) and (2), we conclude that the SSP's role adaptation algorithm preserves the correlation constraint. □

Proof sketch of Theorem 3.4: First, we prove that the macro-adaptation algorithm does not generate any duplicate results. The proof is straight-forward since the master stream does not have any duplication. Second, we prove that the macro-adaptation does not miss any correlation results. We first prove that $\forall s_1$, s_1 and $S_2[W_2]$ are sent to the same host. The proof is straight-forward since $S_2[W_2]$ are either replicated on all hosts (before t_s) or co-located with s_1 (after t_s). We then prove that $\forall s_2$, s_2 and $S_1[W_1]$ are co-located on at least one host. If $s_2.t < t_s$ or $s_2.t \ge t_s + W_1$, we can prove that s_2 is co-located with $S_1[W_1]$ following the proof of theorem 3.1. If $s_2.t \in [t_s, t_s + W_1)$, s_2 and s_2^* covers all the hosts. Thus, s_2 and $S_1[W_1]$ are co-located on at least one host. □