DIAMOND: Correlation-based Anomaly Monitoring Daemon for DIME

Ahsan Arefin, Klara Nahrstedt, Raoul Rivas
Department of Computer Science
University of Illinois at Urbana Champaign
Urbana, USA
Email: {marefin2, klara, trivas}@illinois.edu

Jiawei Han, Zixia Huang
Department of Computer Science
University of Illinois at Urbana Champaign
Urbana, USA
Email: {hanj, huang21}@illinois.edu

***Abstract*—Distributed Interactive Multimedia Environments (DIMEs) show important dependency constraints between application and underlying system components over time. For example, the video frame rate and the underlying bandwidth usage have a strong performance dependency. Performance dependencies must also be considered among distributed components. These dependencies over a time-span form correlation relationships. Violations of such correlation relationships represent collective anomalies. Users and most specifically DIME application developers face problems of finding (detecting), localizing such anomalies, and adapting against them in real-time. Current practices are to collect *joint application-system metadata* characterizing behaviors of application and system components while a DIME session is running, and then analyze them offline. Our goal is to provide a framework, called DIAMOND, that allows for real-time and unobtrusive collection and organization of joint application-system metadata in order to assist in finding such correlation violations in the system. DIAMOND works in four steps: (a) real-time metadata collection, (b) metadata processing to allow efficient computation of correlation constraints, (c) metadata distribution for efficient clustering of distributed metadata, and (d) anomaly detection, localization, and evolution monitoring based on violations of correlation relationships. Our results from real implementations and simulations with PlanetLab traces show the effectiveness of DIAMOND in terms of network overhead and anomaly detection time.**

I. Introduction

Distributed Interactive Multimedia Environments (DIMEs) represent multi-site interactive real-time systems using rich multimedia sensing and actuating environments across geographically distributed locations. A large number of correlated multimedia devices is connected at each site and accessed via a single entry point (called a site-gateway or rendezvous-point). Some DIME examples include video-conferencing [2], 3D Tele-Immersion [19] and on-line multi-player gaming [5]. Real-time and highly resource consuming communications are important characteristics of such applications.

DIMEs also have important characteristics of dependency constraints between applications and underlying systems components over time. For example, video frame rate and its underlying bandwidth usage have a strong performance dependency. Strong dependencies also exist among distributed components. For example, sending and receiving streams and their rates are dependent on each other. Moreover, application-specific semantic dependencies may exist. For example, sensor data across sites are highly dependent on each other in immersive gaming applications. Dependencies of these software components and their allocated resources over a time-span are called correlations.

Due to mis-configurations, heterogeneity and unreliability of today's hardware and software, violations of such correlation relationships may happen in any part of the system while a DIME session is running. These violations represent anomalies. According to Chandola *et al.* [4], "anomaly refers to the patterns in data that do not conform to expected behavior". Thus the violation of expected correlations clearly defines anomalies in DIMEs and may cause degraded performance, QoS violation, wrong outputs and even application crash. [4] *et al.* defines it *collective anomaly* as follows: "collection of time-series metadata shows collective anomaly if certain relationships do not exist to the past collection of same metadata (auto-correlation) or the current collection of other metadata (cross-correlation)". Hence, in order to detect collective anomaly, we collect joint application-system metadata of correlated application-system components in DIMEs and monitor the violation of defined correlation relationships among them. We only consider linear correlations in this paper.

Users and most specifically application developers face problems in finding the violation of such linear relationships and localizing them while the multimedia session is running to adapt system configurations, application design and/or QoS management. For example, in our tele-immersive environments [18], we observed a violation of expected positive correlation between sending and receiving stream rate at the application level between two distance sites while using DCCP communication protocol even though we observed sufficient network bandwidth. These observations localized the possible errors in the underlying system-level and we identified bounded queue lengths in our LINUX kernel that caused the frame loss. As DCCP allows frame drop, violation of dependency equations at any single point in time is not significant. Without detecting the existence of linear correlations, it is hard to localize such anomaly alone while the system is running. Current practice is to

collect joint application-system metadata (e.g., frame rate, bandwidth usage, stream rate and so on) in run-time and then analyze them offline to detect anomalies.

The major challenges here are that (1) DIME requires real-time monitoring, detection and localization of such dependency violations to adapt system and application configurations, and QoS management; (2) the management plane for the anomaly monitoring, detection and localization should be unobtrusive so that the real-time application streaming is not interfered. Maintaining these requirements is non-trivial due to the large number of joint application-system metadata in terms of quantity and dimensionality distributed across sites.

Our goal is to provide a framework, called DIAMOND (correlation-based Anomaly MONitoring Daemon for Distributed Interactive multimedia environment), that allows real-time and unobtrusive monitoring of metadata correlation to assist in anomaly detection and localization in DIMEs. DIAMOND works in four major steps: (1) real-time monitoring and collection of joint application-system metadata, (2) DFT-based compression for efficient computation of correlations considering time lag, (3) metadata distribution to cluster co-dependent metadata that eases anomaly detection, and (4) finally distance-based anomaly detection, localization and evolution monitoring depending on constraints. Constraints can be predefined or redefined/ updated while the system is running. Periodic queries are used in this distributed domain to check for violation of such constraints in unobtrusive manner. Our results from real implementations and simulations with PlanetLab traces show the effectiveness of DIAMOND in terms of network overhead and anomaly detection time in highly, resource-consuming, real-time DIMEs.

II. Models and Assumptions

A. DIME Application Model

Interactivity, real-time guarantee and distributed communications are the key characteristics of DIMEs [17]. Users not only interact with the computing environment, but also with other users and their computing environments through various communication channels. Figure 1 presents the architectural model of DIMEs containing five sites. Each site consists of several local nodes (LNs), representing DIME's input and output sensing and actuating components, and an entry point called rendezvous point (RP), all connected via a local area network. Communications to/from remote sites are done through RPs. Input components (e.g., cameras, microphones, sensors) are sensing devices to capture user's actions in real-time, whereas output components (e.g., displays, haptic interfaces) are visual and force feedback preceptors and actuators.

B. Metadata Model

Sites monitor and store joint application-system metadata information regarding application-specific devices they connect with or system resources they maintain. Examples of application metadata include frame size, image reconstruction time, image resolution, and object positions in active space. Examples of system metadata include communication delay and bandwidth across sites, packet loss rate, stream rate, CPU utilization, and memory utilization. Each node generates a collection of metadata periodically at each τ interval[1]. For example, at time t_i ($i > 0$), a camera generates a metadata stream m_{t_i} containing current device and resource metadata. We can define $m_{t_i} = [x_i, y_i, ..., z_i]$, where x_i, y_i, and z_i are statistical values (such as avg, min, max, or stdev) of metadata x, y and z respectively generated during interval $[t_{i-1}, t_i]$ (where $t_i - t_{i-1} = \tau$). Let assume x=frame size, y = CPU overhead, and z = frame rate, then the variables with subscript i denotes their statistical values of interval $[t_{i-1}, t_i]$. m_{t_i} is stored in a local database. Thus, each metadata type generates a time-series (e.g., $x = x_1, x_2, x_3, ..., x_n$ is a time-series metadata for frame size). Note that, frame size metadata of two different cameras are considered as two different metadata types. Sometimes derived metadata are created by using arithmetic operations on two or more *atomic* metadata. For example, we can create a new metadata type a, which is the multiplication of x and z (i.e., $a_n = x_n * z_n$). Applications should define such derived metadata before the system starts.

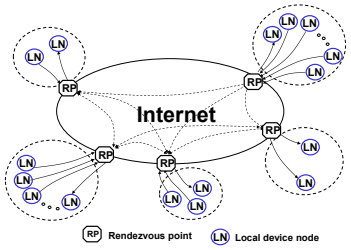

Figure 1. General DIME architecture. Each site contains an RP node and a set of LNs. Examples of LNs are cameras, sensors, microphones and displays.

C. Anomaly Model

Our goal is to monitor the linear dependency or linear correlation relationship between two time-series metadata (both atomic and derived) collected at distributed sites. Metadata can be correlated with two *primitive* relationships: *positive* and *negative*. If x and y are two different metadata types in joint application-system metadata collection, such relationships between them are denoted by $(x \oplus y)$ and $(x \ominus y)$ respectively. Relationship constraints can be defined before the system starts and during the system runs.

For example, network rate (RN) is dependent on application frame size (MA) and application frame rate (RA) by relationship equation $RN = \lceil \frac{MA}{MN} \rceil RA$, where MN=network packet size and MN is constant. If RA is constant in any application (for example in audio), this

[1]It only defines the sampling rate of joint application-system metadata and is not related to the sampling rate of application data.

relationship can be defined by ($RN \oplus MA$), i.e., network rate is positively correlated with application frame size. Similarly, if MA is constant, the relationship can be defined by ($RN \oplus RA$). Some applications require both MA and RA to change. In that case, we define a new derived metadata type $MRA = MA * RA$, and the relationship is defined by ($RN \oplus MRA$). Likewise, a negative correlation exists between MA and RA (provided that network rate RN is fixed) and can be denoted by ($MA \ominus RA$). Violation of these defined correlation relations are anomalies in the system.

Note that, we do not consider anomalies by using the current values in the relationship equation at any particular point in time. Because this may not reflect the true system anomaly as we mention in the previous DCCP example, where frame drop is allowed. It is also difficult to know which particular values in the equation cause the wrong output, which makes it hard to localize the reason of anomalies. Moreover, many relationships including semantic correlations between joint application-system metadata can not be well represented by equality equations (e.g., the relationship between frame size and rendering time, or correlation between sensor metadata across sites). Hence in our current work, our goal is to monitor only linear correlation-based relationship constraints present in DIMEs even with time lag between correlated metadata. It does not cover the detection of complete set of anomalies possible in the system, but definitely helps developers as well as users to detect a large subset of it.

III. The DIAMOND Architecture

Figure 2 illustrates the architecture of DIAMOND. Each site has a local *metadata store* node in addition to LNs and an RP node. DIAMOND works in four major steps given below:

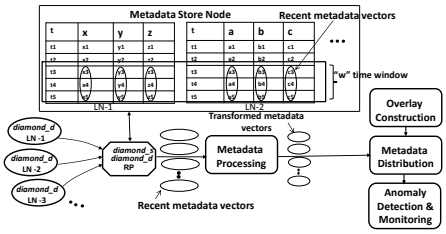

Figure 2. The DIAMOND Architecture.

A. Metadata Collection

A lightweight metadata collection daemon ($diamond\_d$) runs inside each node in the system. In addition, the RP node runs a server daemon $diamond\_s$. At each τ interval, each $diamond\_d$ sends the collection of metadata (m_t) of the residing node to the $diamond\_s$ server, which stores them in a local metadata store. τ should be higher than the frequency of any events (such as sending or receiving of streams) in the system. We use interval-based metadata collection instead of event-based so that we can also detect anomalies for aperiodic metadata. Figure 2 shows how time-series metadata (x, y, and z) of LN1 and time-series metadata (a, b, and c) of LN2 are stored at metadata store node at different time points t_i ($1 \leq i \leq 5$), where $t_{i+1} - t_i = \tau$.

B. Metadata Processing

If we compute correlations considering time-series metadata starting from system startup, the value may not reflect any violations instantaneously when it appears in the system. To address this, we define a sliding window of recent w metadata points. Metadata points inside this window are only used to compute current correlations exist in the system. In Figure 2, w is set to 3. The window slides with the arrival of a new metadata point. Hence, the size of the window in time is $(w-1)\tau$. The time-series vector of each metadata type inside this sliding window is transformed to a coefficient vector by using DFT (Discrete Fourier Transform). DFT is used because of two reasons; (1) it helps to compress the size of the time-series metadata within the window by considering only several dominating points in the coefficient vector that represent the original time-series, and (2) coefficient vectors preserve correlation relationships. As the window slides for each new metadata point, DFT coefficients of recent w points should be computed incrementally. The detail of metadata processing step is shown in Section IV-A.

C. Metadata Distribution

DIAMOND aims to store coefficient vectors of correlated distributed metadata closely regardless of their sources for faster detection of violation. To guarantee this, each coefficient vector is mapped as a single point in a multi-dimensional hypercube. The dimensionality depends on the cardinality (α) of the coefficient vector. DFT transformation ensures that points who are positively or negatively correlated are mapped closely in this hypercube, which helps easier detection of correlation violations depending on their positions in the hypercube. As correlated metadata are distributed across sites, we need a distributed structure to embed this hypercube in the system. For this purpose, a distributed overlay is created by embedding (or indexing) dis-joint regions of hypercube into overlay nodes. The overlay only considers RP nodes from participating sites. Section IV-B and Section IV-C discuss about this overlay construction and its indexing algorithms respectively. The next step is to distribute coefficient vectors into the overlay node depending on this indexing of regions. This allows correlated points to be stored in a set of nodes whose regions are adjacent in the hypercube. The detail is given in Section IV-D.

D. Anomaly Monitoring

Finally, at anomaly monitoring step, defined relationship constraints are checked periodically for violation. Periodic queries with these constraints are fused in the overlay. A central administrator node fuses such queries to any node in the overlay. The query is routed using the overlay index

structure to a set of candidate nodes, where correlated metadata mentioned in the query should reside. Primitive correlations between them are then evaluated by using Euclidean distance (Section IV). Replies are sent in an aggregated form and carries violated relationships, which help the administrators or developers to localize the anomalies in the system. Section IV-E discusses more details about query routing, selection of candidate nodes and query responses. Section IV-F shows how this approach can be extended to find correlations with arbitrary time lags.

IV. The DIAMOND Design and Implementation

A. Correlation Computation

DIAMOND computes primitive correlations to check the violation of defined relationships between metadata. To define correlation, we adopt the definition from [16]. Let us represent the time-series of metadata (both atomic and derived) type x as $x_i(n-w+1 \leq i \leq n)$, where x_n represents the most recent measurement point in the window of recent w points and $n \geq w$. The *cross correlation* ($\rho(x,y)$) between two series of metadata x and y is defined by:

$$\rho(x,y) = \frac{\sum_{i=n-w+1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sigma(x)\sigma(y)} = \sum_{i=n-w+1}^{n} \hat{x}_i\hat{y}_i (1)$$

where $\bar{x}$ and $\sigma(x)$ are the average and standard deviation of x respectively, and $\hat{x}_i$ is the z-norm of x_i of recent w points in the window. The same notations apply for y. An important observation here is that the cross correlation is the inner product of z-norms, therefore we can calculate ρ using the Euclidean distance of their normalized series:

$$d^2(\hat{x},\hat{y}) = \sum_{i=n-w+1}^{n}(\hat{x}_i-\hat{y}_i)^2 = 2-2\rho(x,y) \quad (2)$$

Thus the cross correlation is simply another *similarity measure* directly based on the Euclidean distance. Computing correlations using direct method (Equation 1) is slow and can not meet the requirements of real-time computation across sites for large burst of time-series metadata, rather our goal is to use an efficient way of computing $d(\hat{x},\hat{y})$ so that correlation relationship can be computed using Equation 2. To address this, we use DFT. It helps to compress the size of the time-series metadata by considering only dominating frequencies, and it preserves the Euclidean distance in the compressed form. Therefore, to calculate correlations between metadata type x and y, we calculate the Euclidean distance between the DFT coefficients ($\hat{X}$ and $\hat{Y}$ respectively) of their normalized series according to the following equation, where $d(\hat{x},\hat{y})$ and $d(\hat{X},\hat{Y})$ are the Euclidean distances between $\hat{x}$ and $\hat{y}$ time-series and their DFT coefficients respectively:

$$\rho(x,y) = 1 - \frac{1}{2}d^2(\hat{x},\hat{y}) = 1 - \frac{1}{2}d^2(\hat{X},\hat{Y})$$

We use uniform cut-off frequency α in the DFT coefficient vector to select dominating frequencies. So the cardinality of the compressed coefficient vector is α. α is very smaller than the total number of original metadata points in the sliding window [21]. Next we show how the Euclidean distance can compute primitive correlations.

1) Defining Correlations using Distance: Positive and negative correlations can be defined depending on the distance value between coefficient vectors. A correlation exists between two time-series only when the Euclidean distance between their coefficient vectors is less than or equal to a user defined threshold ϵ [21]. ϵ can define positive and negative correlations by the following two equations[21]:

$$\rho(x,y) \geq 1-\epsilon^2 \Rightarrow d(\hat{X},\hat{Y}) \leq \epsilon \quad (3)$$

$$\rho(x,y) \leq -1+\epsilon^2 \Rightarrow d(\hat{-X},\hat{Y}) \leq \epsilon \quad (4)$$

Hence, the time-series series x and y are positively correlated if $d(\hat{X},\hat{Y}) < \epsilon$ and negatively correlated if $d(-\hat{X},\hat{Y}) < \epsilon$. To compute correlations, we first compute normalized DFT coefficient for each metadata. Note that ϵ value can be different for different relations and for different metadata. As the window slides, coefficient vectors should be calculated incrementally.

2) Incremental Updates: It is inefficient to calculate the normalized DFT coefficients from the scratch every time a single value changes in the sliding window. Rather, we can efficiently maintain the DFT coefficient incrementally. Suppose, $X^{old} = X_1^{old}, X_2^{old}, ..., X_w^{old}$, is the current DFT coefficient vector of the time-series data x in sliding window $x_{n-w+1}, x_{n-w+2}, ..., x_n$. With the arrival of new data point x_{n+1}, we calculate new DFT coefficient vector $X^{new} = X_1^{new}, X_2^{new}, ..., X_w^{new}$ of new series $x_{n-w+2}, x_{n-w+3}, ..., x_{n+1}$ using the following equation[21] ($k = 1, 2, ..., w$):

$$X_k^{new} = e^{\frac{j2\pi(k-1)}{w}}({X_k}^{old} + \frac{(x_{n+1}-x_{n-w+1})}{\sqrt{w}})$$

Then to calculate normalized DFT coefficient $\hat{X}$ we use: $\hat{X}_1 = 0$ and $\hat{X}_i = \frac{X_i}{\sigma_x}$ where $2 \leq i \leq \alpha$ and $\sigma_x = \sqrt{\sum_{i=n-w+2}^{n+1}(x_i-\bar{x})^2}$. After computing the normalized coefficient vector, we consider only first α frequencies. The primitive correlations are then computed using Equation (3) and (4). Constraints are violated if the equations are not satisfied. The value of each coefficient is bounded in $[-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$ [21]. Thus the normalized vector can be mapped as a point in a 2α-dimensional hypercube space (doubled because Fourier coefficients are complex numbers), where each dimension value is bounded between $[-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$. DFT ensures that correlated metadata are mapped closely in this hypercube.

B. Overlay Construction

The purpose of overlay construction is not only to embed a hypercube space in a distributed structure, but also to

allow queries to traverse in an efficient way to check for violation of constrains. For this purpose, DIAMOND considers a degree-bounded, height balanced and locality aware overlay tree structure. Overlay is created using RP nodes. We use degree-bounded structure, because in a multimedia distributed system RP nodes have resource constraints to serve arbitrarily many children nodes in the tree [1]. Also as any node in the overlay can initiate a query, a height-balance tree helps to reduce communication overhead. Locality awareness makes the communication in the tree faster.

DIAMOND's overlay tree is constructed centrally in a central administrator node called *session-controller*. The intuition behind this is that most of the distributed interactive applications like 3D collaborative dancing or immersive video conferencing require all sites to join at the very beginning of a session by registering to a central session controller. This works well when the scale of the system is small (order of 10^{th}). Some applications may require large number of sites or nodes may join during run-time. In that case, we construct the degree-bounded height-balanced tree with the available nodes and allow other nodes to join dynamically using node joining algorithm similar to our earlier work [1]. Figure 3(a) shows an example of DIME metadata routing architecture. Sensors, cameras and display devices are examples of LN nodes connected at each site. A, B, C, D, E and F are RP nodes. Figure 3(b) presents the overlay created by these RP nodes. We set degree equals to two to build up the overlay tree as shown in Figure 3(c). DIAMOND uses off-the-self *nodebuddy-based* [1] failure detection and overlay re-organization and so, we skip them in this paper. Note that, DIAMOND's design is orthogonal to the overlay choice and works with any choice of the tree overlay.

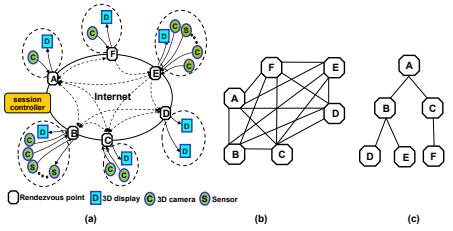

Figure 3. (a) DIME with 5 sites, (b) Corresponding overlay considering RP nodes, and (c) DIAMOND overlay tree for degree=2.

C. Distributed Indexing

We use a divide and conquer based algorithm to partition a hypercube region into several disjoint regions to embed them into overlay nodes. This allows the correlated coefficient vectors to physically map to a set of nodes in the overlay whose regions are adjacent in the original hypercube. To describe this section, we introduce some basic notations. A region r is a 2α dimensional space. $\hat{X}$, the normalized DFT coefficient of metadata x lies inside r, i.e., $\hat{X} \in r$, if and only if $\hat{X}$ is inside r. If a region r is portioned into two equal regions r_1 and r_2, the we define $r_1 = (r/2)_u$ and $r_2 = (r/2)_l$, where u and l subscripts indicate two regions of r after partition. A region r_1 is contained within r_2, hence $r_1 \prec r_2$, if all points in r_1 also lie in r_2. Two regions r_1 and r_2 are said to be equal if they have the identical boundary. Two consecutive regions can be 'merged' to form a larger region like $r = r_1 \cup r_2$. We also define another relation called neighbor ($\sim$). A region r is the neighbor of $\hat{X}$ (i.e., $\hat{X} \sim r$), if and only if r is within ϵ distance from $\hat{X}$.

Let Tr be the rooted tree where $p(E)$ and $c(E)$ are the parent node and the set of child nodes of node E respectively. Each node E is assigned with two regions, *self-region* $\delta(E)$ and *subtree-region* $\Delta(E)$. Subtree-region $\Delta(E)$ specifies the region of the entire subtree rooted at node E. A point $\hat{X}$ is stored at node E if $\hat{X} \in \delta(E)$. If $\hat{X} \in \Delta(E)$, then $\hat{X}$ is stored somewhere in the subtree rooted at node E. By definition, $\delta(E) \prec \Delta(E)$.

The following four properties are hold by the regions assigned to nodes:

i) **Disjointness:** $\delta(E1) \cap \delta(E2) = \emptyset$, all any pair of nodes $E1$ and $E2$.
ii) **Subtree range:** $\delta(E1) \prec \Delta(E1)$ and $\Delta(E1) = \delta(E1) \bigcup_{E2 \in c(E1)} \Delta(E1)$.
iii) **Hierarchy:** If $E1$ is an ancestor of $E2$, $\Delta(E2) \prec \Delta(E1)$.
iv) **Entirety:** $\Delta(root(Tr))$=*complete hypercube*.

To guarantee these properties, we use an incremental algorithm to assign regions to nodes. Let us consider the overlay shown in Figure 3(c). At the beginning, the whole region r is assigned at node A and hence $\Delta(A) = \delta(A) = r$. For the first child B, node A partitions its self-region into two equal regions and assigns one to node B, i.e., $\delta(A) = (\delta(A)/2)_u$ and $\delta(B) = \Delta(B) = (\delta(A)/2)_l$. For node C, node A further divides the self-region and assigns $\delta(A) = (\delta(A)/2)_u$, and $\delta(C) = \Delta(C) = (\delta(A)/2)_l$. Thus each node divides its self-region into two for each of its child node and assigns regions equally. This continues for all non-leaf nodes. We use the heuristic algorithm mentioned in [3] to partition a region into two. Figure 4 shows the range assignment for a two dimensional grid space.

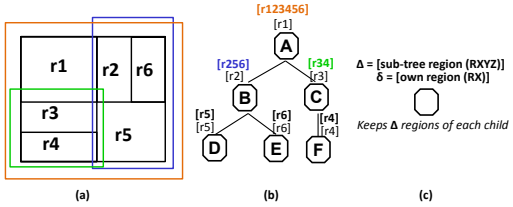

Figure 4. (a) Example of partitioning a 2-dimensional grid region, (b) Mapping of regions into the tree, and (c) Region-based indexing structure at nodes.

D. Metadata Distribution

Overlay is constructed at bootstrap. For a fixed value of α the hypercube space is fixed and so the partitioning of regions and indexing of nodes are also done at the

bootstrap. At run-time, normalized vectors originating at each site are fused in the overlay to be stored by remote nodes depending on this indexing. The vector $(\hat{X})$ is stored in the node, which contains the region r such that $\hat{X} \in r$. Such assignment ensures that coefficient vectors of metadata who are correlated to x (i.e., within ϵ distance of $\hat{X}$ in the hypercube space) will map to a set of nods with regions r such that $\hat{X} \in r$ or $\hat{X} \sim r$. Each compressed vector is stored with *metadata-info*. *Metadata-info* contains the name of the metadata and the location of the metadata origin. Coefficient vectors are updated in the overlay at μ $(\geq \tau)$ intervals defined by the administrator. Having larger value of μ reduces the update frequency of coefficient vector into the overlay by sacrificing the *resolution* of anomaly detection, i.e., the delay between the time when an anomaly actually happens in the system and the time when DIAMOND detects it.

Considering the previous example in Figure 3, lets assume, site A generates metadata x and y, site B generates metadata b, site C generates metadata c and site E generates metadata z. After computing the normalized coefficient vectors over the sliding window, their value logically map into a 2-dimensional hypercube as shown in Figure 5(a)[2]. Figure 5(b) shows how the insertion works for metadata x and y depending on this mapping. The insertion request of [*metadata-info*, $\hat{X}$] and [*metadata-info*, $\hat{Y}$] are routed from their origin site A in the overlay for insertion into the nodes that has the corresponding regions. The request follows $A \rightarrow B \rightarrow E$ path for metadata x and $A \rightarrow C$ for metadata y to be stored at node E and C.

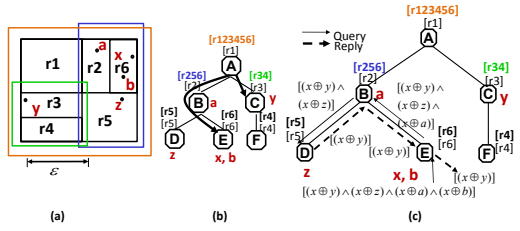

Figure 5. (a) Example of metadata mapping in 2-dimensional grid (each point represents the coefficient vector of associated metadata), (b) Insertion path for x and y according to their coefficient vectors, and (c) Routing of query with constraint "$(x \oplus y) \wedge (x \oplus z) \wedge (x \oplus a) \wedge (x \oplus b)$" and its' response.

E. Run-time Anomaly Monitoring

Once values (normalized coefficient metadata vectors along with *metadata-info*) are fused, the overlay is now ready to serve queries to find anomalies. Queries containing defined constraints are fused periodically into the overlay by an administrator node.

Each query (Q_x) is associated with a specific metadata (x), and constraints (C_x) in the query are defined by the relationship of this metadata over other metadata in the system. Queries are performed in two steps. The first step collects the normalized coefficient vector $(\hat{X})$ for which the query is generated and the second step routes the query into the overlay according to this vector. The first step is done requesting coefficient vector directly from the origin. The next step follows routing in the overlay. The routing algorithm is given in Algorithm 1. It ensures that the query reaches the nodes which contain region r where $\hat{X} \in r$ or $\hat{X} \sim r$.

After routing, violation is calculated by computing the distance of $\hat{X}$ to other coefficient vectors of metadata mentioned in the query constrains if they are available in the candidate nodes. Equation (3) and (4) are then used to check for violations. Replies are done in an aggregated form. Each node replies with locally non-satisfied constraints mentioned in the query. Replies are aggregated to send back to the parent node.

Algorithm 1 *Query*(E, Q_x)

E: The node, where the query comes;
Q_x: $[\hat{X}, C_x]$;
$\mathcal{R}(E)$: Set of coefficient vectors stored at E;
$\mathcal{M}$: Set of metadata constrained at C_x;
$\otimes$: $\oplus$ or $\ominus$;
if $(\hat{X} \sim \delta(E)$ or $\hat{X} \in \delta(E))$ **then**
  for all $(y \in \mathcal{M})$ **do**
    if $(\hat{Y} \in \mathcal{R}(E))$ **then**
      calculate distance between $\hat{Y}$ and $\hat{X}$;
      $C_x = C_x - (\hat{X} \otimes \hat{Y})$, if constraint is satisfied.
    end if
  end for
end if
for all $(child \in c(E))$ **do**
  if $(\hat{X} \in \Delta(child))$ **then**
    $C_x \leftarrow C_x \cup$ *Query*$(child, Q_x)$;
  end if
end for
if (none of the above is satisfied) **then**
  Query$(p(E), Q_x)$;
end if
return C_x;

Continuing the previous example, we use Q_x to define the query associated with metadata x with constraints $C_x = [(x \oplus y) \wedge (x \oplus z) \wedge (x \oplus a) \wedge (x \oplus b)]$ (we consider only positive correlations for simplicity of example). All metadata correlated to x should have their normalized coefficient vectors within ϵ distance of $\hat{X}$ value. According to Figure 5(a), these values are possible to be mapped into regions $r2$, $r5$ and $r6$, and corresponding vectors are stored into overlay nodes B, D and E respectively. Hence these are the candidate nodes to check for violation. In the first step, $\hat{X}$ is collected from its origin site A. Then a query $Q_x = [\hat{X}, C_x]$ is fused randomly at any node to route to the candidate nodes using Algorithm 1. In this example, we assume that the query is fused to node E by the administrator node. It traverses E, B, and D nodes as shown in Figure 5(c) and checks the violations using Equation (3) and (4). The reply follows the same path back to the administrator node and shows violated constraints (here $(x \oplus y)$) to localize the anomaly.

F. Lagged Correlations

So far, we only considered how can we find cross-correlations between two synchronized series (such as metadata between cameras). But in real-life scenarios, some

[2]Points in the 2-dimensional grid refer to the value of coefficient vectors of associated metadata.

metadata in DIMEs are cross-correlated and even auto-correlated by a certain time lag. To address this, we use similar approach defined in [21]. Each coefficient vector of metadata x is stored with a timestamp T_x, which defines when the value is inserted into the overlay. If x and y are positively correlated with a time lag l (where y follows x), then $\hat{X}$ inserted at time $T_x = (t - l)$ is still in ϵ distance of $\hat{Y}$ inserted at time $T_y = t$ in the hypercube space. Queries are defined for the metadata which are ahead of time in correlations (e.g., y in this case) and associated with l values. The lag time l depends on the delay at the network and application processing, and different for different metadata. As the time lag value may not be accurate, DIAMOND checks the lagged correlations upto largest time lag value (T_S) that is possible in the system, which is mainly driven by the end-to-end delay and bounded for real-time applications. This simplifies the system by removing the need for measuring the time lag value at finer granularity at different parts of the system. If the largest time lag assumption is violated in the system, then correlation constraint is also violated, which eventually signifies an anomaly in the system. Mapped vectors which have timestamp older than T_S from current time are removed periodically from the system.

V. Performance Evaluation

We measure the performance of DIAMOND using both simulation and real testbeds. Unless mentioned, we use the following values of parameters in all experiments: $\tau = 10ms$, $\alpha = 5$, $w = 200$, $\mu = 10sec$, $T_S = 1s$, and $\epsilon = 0.1$. Each insertion (or update) message for a metadata contains its compressed vector (4α bytes), name of the metadata (4 bytes), and the address of RP and LN nodes of its origin site ($32 + 32$ bytes). Queries are fused at the interval of 5 seconds to monitor the violations of defined constraints. Each constraint is defined by 4 bytes in the query. We compare DIAMOND with "naive"approach. In naive approach, metadata are stored locally and there is no distributed indexing and DFT compression. All queries to check for violation traverses all sites with uncompressed metadata within the sliding window. Note that, the naive approach still uses tree overlay structure for routing.

For real experimental set up, we use three separate 3DTI testbeds in the lab to emulate distributed environments. Each testbed contains a plasma display, 4 cameras and a single audio device. The 3D representations of users from three testbeds are merged into a joint virtual environment in real time for virtual interactions and remote collaborations. We run the application session experiments for about 2 hours. The purpose of this experiment is to show how DIAMOND scale for number of different metadata types.

We use 7 types of metadata for each camera device such as frame size, processing time, reconstruction time, frame rate, CPU overhead, memory utilization and bandwidth usage. With 4 cameras at each site, we generate 84 different metadata. We also use metadata for audio, display and gateway nodes. Metadata are stored into a SQL database locally at a separate metadata store node. We define a list of constraints among those metadata. For experimental purpose we vary number of metadata at each device to change the number of metadata types. Figure 6 gives the average execution time of queries fused to monitor violations. As the figure shows, even with large number of metadata, queries are responded very fast to detect the anomalies. The round-trip delay between sites is low as remote sites are connected over a Gigabit LAN in the lab. So, the query latency mainly shows the the computation overhead of DIAMOND at each node.

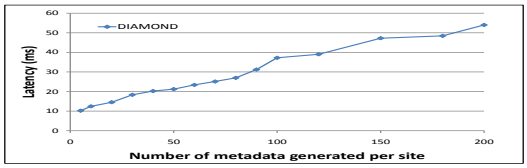

Figure 6. Query performance for different number of metadata types

Also, we simulate DIAMOND in an emulated network setup on top of a discrete network event simulator coded in Java. We emulate a 'virtual' network with node-to-node latencies obtained from 4 hours PlanetLab traces, which consists 250 distinct nodes. This trace gives us the connectivity information and rtt delays among the RP nodes. We use this information to simulate communication latency for DIME systems. A height balanced, degree(=2) bounded, and locality aware tree overlay is created at the beginning using the algorithm of Section IV-B.

Figure 7(a) shows the latency of different anomaly queries for increasing number of sites in the system. Even with large number of sites, the query latency is bounded to several hundred of milliseconds due to the distribution of metadata vectors over the overlay according to the embedding of hypercube regions. Each query is routed to several candidate nodes instead of routing the whole overlay. We compare DIAMOND performance with the naive approach that traverses all nodes for any query with raw metadata within the window.

Next, we measure the bandwidth overhead of DIAMOND. Each site has 20 nodes (19 LNs and an RP) and each node has 10 different metadata types (both atomic and derived). Figure 7(b) shows the total bandwidth used by DIMAOND for update and query message. Update takes slightly higher than the query because only a single query is fused for a specific metadata containing all constraints. We also measure the bandwidth overhead by varying the number of metadata types per site in simulation for 50 sites. The result is shown in Figure 7(c). As the figures show, the combined (update + query) DIAMOND overhead is very low compared to the naive approach. Hence, DIAMOND shows very low overhead in communication and processing and still achieves real-time response of violations.

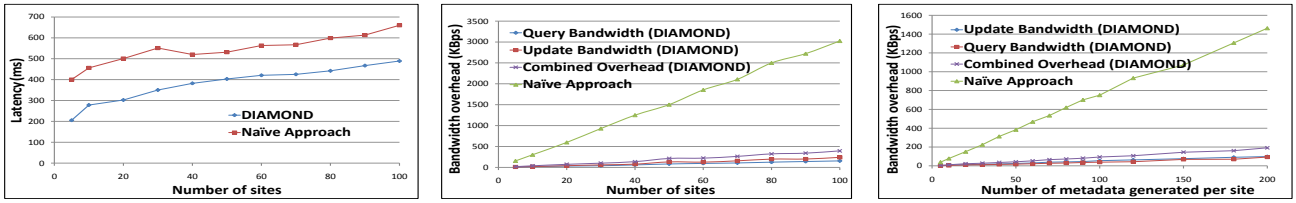

Figure 7. (a) Query latency for different number of participating sites, (b) Bandwidth overhead for different number of participating sites, and (c) Bandwidth overhead for different number of metadata types (50 sites).

VI. Related Works

Data stream analysis is currently a hot research topic in the area of data mining and databases. Some earlier works have addressed the real-time analysis of infinite stream data [7][11][16], but they all consider data streams to be present in a single site for real-time schema analysis. For example, algorithms [9][6] have been proposed for analyzing a financial transaction database stored locally. Some network monitoring applications such as SNMP, NetFlow, [14], and [20] address distributed monitoring for large scale system, but their problem domain is less challenging in the sense that they do not consider real-time analysis of such a large data burst.

CAN [12], Chord [15], Kelips [8], Pastry [13], etc. implement distributed hash structure to provide efficient lookup of a given key value. Since a hash function destroys the spatial relationship between values, they cannot be used in distance computation. MON [10] is a better solution for multicast rather than index-based queries, because it has no prior knowledge about the metadata space in values and time. Q-Tree [1] uses a range-based overlay similar to B-tree, but only considers multi-dimensional range-queries over metadata.

VII. Conclusion

DIAMOND addresses a practical problem that application developers and users face in DIMEs application domain. It presents an unobtrusive solution to monitor metadata effectively and detect the violations (anomaly) of correlation constraints defined by the dependent joint application-systems metadata patterns during run-time. Our real implementation and simulation result shows the efficiency of DIAMOND in terms of network overhead and query resolution time.

Acknowledgment

This research is supported by grants NSF CNS 05- 20182, NSF CNS 07-20702 and Grainger grant.

References

[1] A. Arefin, Y. Uddin, I. Gupta, and K. Nahrstedt. Q-tree: A multi-attribute based range query solution for tele-immersive framework. In *ICDCS*, 2009.

[2] H. Baker, N. Bhatti, D. Tanguay, I. Sobel, D. Gelb, M. Goss, W. Culbertson, and T. Malzbender. Understanding performance in Coliseum, an immersive videoconferencing system. *ACM TOMCCAP*, (1), 2005.

[3] S. Bezrukov and B. Rovan. On partitioning grids into equal parts. *Computers and Artif. Intell*, 1997.

[4] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. In *ACM Computing Surveys*, 2009.

[5] L. Gautier, , L. Gautier, and C. Diot. Design and evaluation of mimaze, a multi-player game on the internet. In *IEEE Multimedia Systems Conference*, 1998.

[6] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining frequent patterns in data streams at multiple time granularities, 2002.

[7] D. Goldin and P. Kanellakis. On similarity queries on time-series data: Constraint specification and implementation. In *Principles and Practice of Constraint Programming*, 1995.

[8] I. Gupta, K. Birman, P. Linga, A. Demers, and R. Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *IPTPS*, 2003.

[9] X. Li and J. Han. Mining approximate top-k subspace anomalies in multi-dimensional time-series data. In *VLBD*, 2007.

[10] J. Liang, S. Ko, I. Gupta, and K. Nahrstedt. Mon: On-demand overlays for distributed system management. In *WORLDS*, 2005.

[11] D. Rafiei and A. Mendelzon. Querying time series data based on similarity. In *KDE*, 2000.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *SIGCOMM*, 2001.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *ICDSP*, 2001.

[14] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen. cSAMP: a system for network-wide flow monitoring. In *NSDI*, 2008.

[15] I. Stocia, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, San Deigo, CA, 2001.

[16] T. Wang. Twstream: Finding correlated data streams under time warping. In *APWC*, 2006.

[17] W. Wu, A. Arefin, R. Rivas, Z. Y. Renata Sheppard, and K. Nahrstedt. Quality of experience in distributed interactive multimedia environments: Toward a theoretical framework. In *ACM Multimedia*, 2009.

[18] W. Wu, Z. Yang, D. Jin, and K. Nahrstedt. Implementing a distributed 3d tele-immersive system. In *ISM*, 2008.

[19] Z. Yang, Y. Cui, B. Yu, J. Liang, K. Nshrstedt, S. H. Jung, and R. Bajscy. TEEVE: The next generation architecture for tele-immersive environments. In *ISM*, 2005.

[20] Y. Zhao, Y. Tan, Z. Gong, X. Gu, and M. Wamboldt. Self-correlating predictive information tracking for large-scale production systems. In *ACM ICAC*, 2009.

[21] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, 2002.