Adaptive Data Stream Processing for Hybrid Coalition Networks: Challenges and Initial Solutions

Jim Kurose*, Dan O'Keeffe†, Peter Pietzuch†, Theodoros Salonidis‡, Anand Seetharam*, Don Towsley*
*Univ. of Massachusetts, Amherst; †Imperial College London; ‡IBM T.J. Watson Research Center

***Abstract*—In this paper, we consider the problem of adaptive stream processing in hybrid coalition networks (HCNs). We first introduce and motivate the data stream management (DSM) model that naturally captures the processing, communication and data access characteristics of several real-time applications. Using a situational awareness scenario as motivation, we identify the key challenges and requirements for efficient DSM operation in an HCN environment and then focus on two specific issues.**

First, we consider routing of real-time, streamed tuples of data (e.g., objects extracted from images) that must be matched against data stored in an archival database in the backend cloud. We outline two approaches, Minimum Delay routing and backpressure, that can be used to dynamically route tuples either to the backend cloud infrastructure for processing, or to a virtual cluster in the MANET in a congestion-sensitive manner. In the first case, real-time image data must be uploaded to the backend; in the latter case, archived data must be downloaded to the MANET's virtual cluster.

Our second challenge involves adaptive query planning for DSM systems deployed in dynamic HCNs. Here, we introduce a network-aware query planning approach that replicates operators within the MANET and then routes tuples to operator replicas over the highest quality available network paths. Our initial experiments show that this approach can achieve more effective stream processing in dynamic networks compared to static query planning, which computes a single deployment plan for a dataflow graph.

I. Introduction

Future coalition operations envision processor- and data-intensive real-time applications, including situational awareness, location tracking, intrusion detection, context sharing, and multimedia analytics, to aid data-to-decision at the tactical edge. These analytics services could provide situational awareness to troops deployed in the field or enable each commander to gather relevant information and take mission-critical decisions. This information could also be sent to a remote command center for a more global view of operations and high-level decisions.

The above applications can be realized using a *Hybrid Coalition Network* (HCN). As shown in Figure 1, an HCN consists of field-deployed tactical MANETs in which coalition nodes (e.g., vehicles and troops) with heterogeneous processing and storage capabilities can communicate among themselves (using both MANET radio channels and infrastructure networks such as 3G/4G cellular) as well as with backend data centers via the infrastructure networks.

A *Data Stream Management* (DSM) model [4] naturally captures the behavior of many real-time, context-aware, analytical services in HCNs. It is a relatively new concept and contrasts with the traditional database model which executes one-shot queries on stored data only. Instead, a DSM model executes *continuous* queries on real-time streaming data. For example, a commander could submit a query that continuously reports events and infers the danger level to troops and vehicles gathered within 100m of an explosion site. In another example, troops deployed in a mission and equipped with head-mounted displays could run situational awareness streaming queries sharing contextual information that continuously detect, recognize and share any suspicious enemy objects, faces, motion or points of interest within 10m of a location. In this paper, we identify challenges and address problems related to key operational requirements of DSM systems in an HCN environment.

In a DSM model, a continuous stream of data items is processed by a *dataflow graph* that represents a query to extract knowledge and information in real time. In such a graph, vertices correspond to data processing operators and directed edges describe the movement of tuples that are streamed between operators [4]. These data items could be database tuples, network measurements, location readings, or images from video sources. The operators could perform simple arithmetic, filtering or aggregation calculations as well as more computationally-intensive database, signal processing or data mining operations on the streamed data. A *query deployment plan* for a given dataflow graph assigns processing operators

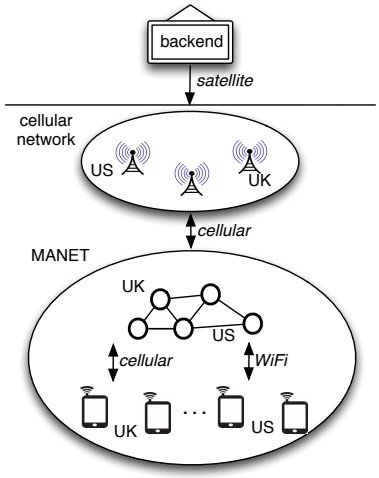

Fig. 1. Hybrid Coalition Network (HCN) with a backend connected to heterogeneous MANETs

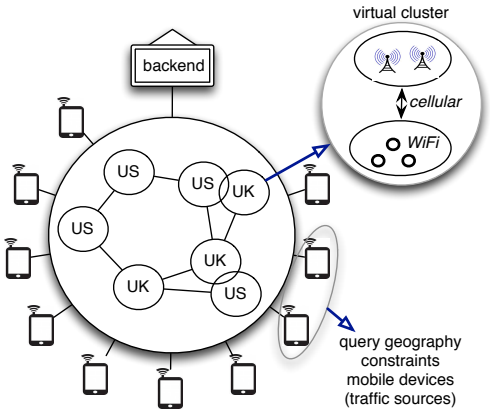

Fig. 2. Virtual Cluster (VC) concept

from the dataflow graph to physical nodes for execution. It is deployed in the HCN's distributed computational resources, taking troop and vehicle locations, and various sensor streams, and continuously performing the requested tasks in real-time.

The DSM model has been applied to commercial applications such as social networking sites and financial risk calculations. Such applications typically run in centralized Internet data centers where high volumes of data traffic are processed by high-performance networked compute clusters at low cost. HCNs differ radically from data centers in both nature and objectives and yield a unique set of challenges that have not yet been addressed. Centralized data processing is neither feasible nor desirable due to bandwidth limitations. Data sources and processing resources are heterogeneous, distributed, mobile and connected via the HCN. A key objective is resilience of critical mission operations rather than low cost. Resource allocation depends on the location of data sources, network security policies and mission context. HCNs have multiple administrative domains, and therefore require a security framework that can handle dynamic streams.

As shown in Figure 2, we envision a logical computing, networking, and storage architecture in which HCN nodes are organized into *Virtual Clusters* (VCs), implemented logically above the HCN physical architecture. VCs provide aggregate processing resources for streaming queries. Cluster head nodes are responsible for managing VC resources and providing processing capacity to streaming queries. Organizing well-connected HCN nodes into VCs offers the possibility of locating computation close to query sources and sinks.

In this paper, we focus on two critical challenges arising in the context of distributed and adaptive DSM operation in the face of dynamic HCN conditions:

(1) The first challenge involves stream processing when real-time, streamed tuples of data (e.g., objects extracted from images in a situational awareness scenario, as described in §II) need to be matched against data stored in an archival database in the backend cloud. Thus, unlike the traditional DSM model which operates exclusively on streamed real-time data, we consider the case that archival data must be accessed as well. In this scenario, one option is to send a tuple via the cellular infrastructure to backend cloud infrastructure for processing; an alternative would be to download (and cache) archived images in the MANET VC, and perform processing locally in the VC. The choice between these alternative should be made with the goal of minimizing overall system processing and communication delay.

We formulate this problem in §II, and describe distributed approaches for adaptively determining whether to route image tuples (in the context of a face/scene recognition scenario) to the backend cloud or the MANET VC for processing. We describe both a back-pressure approach that routes tuples based on instantaneous queue lengths in neighboring nodes, and Minimum Delay Routing approach that makes tuple routing decisions over longer time scales.

(2) The second challenge focuses on stream data processing within the MANET that forms a VC. We investigate how *adaptive query planning* that takes the current conditions of the HCN into account can achieve higher availability and performance. While adaptive query planning has been explored in the database research community to cope with changes in distributions of data patterns in a centralized environment [7], our goal is to perform query planning and routing in a manner that reacts to the network conditions in a distributed HCN.

We propose a new network-aware approach for adaptive query planning that consists of two steps: (i) operators in the dataflow graph are replicated within the MANET in a manner that provides robust availability and access in the face of network congestion; and (ii) data stream items are then routed to operator replicas over the highest quality available network paths. Our initial experiments show that this approach can achieve more effective stream processing in dynamic networks compared to static query planning, which computes a single deployment plan for a given dataflow graph.

The rest of the paper is organized as follows: in §II, we describe an object recognition application involving both stream and archival data, and describe two approaches optimizing tuple routing in this scenario; §III describes the problem of adaptive query planning, our approach for network-aware adaptation of plans using dynamic tuple routing, and initial experimental results; §IV concludes the paper.

II. Virtual Cluster Formation and Adaptive Stream Query Routing

In this section, we discuss our newly initiated effort, begun as a new task P2.3. in BPP13, on adaptive stream query routing and cluster formation in hybrid coalition networks.

Highly responsive data stream query processing in HCNs must simultaneously address challenges in (i) data placement and access, (ii) networking, and (iii) computing. *Data* generated within a MANET deployed at the HCN edge must often be processed with historical or archived data stored in back-end repositories. *Communication* among MANET nodes occurs via MANET routing, 3G, or other infrastructure links. *Computing* must be located to efficiently process MANET and

back-end data streamed through the HCN. Organizing well-connected MANET nodes into virtual clusters (VCs) offers the possibility of exploiting statistical multiplexing gains while locating computation close to query sources and sinks.

A. Motivating Example: Facial and Scene Recognition

We begin with the motivating example shown in Figure 3 to illustrate the data, computing, and networking challenges in adaptive query routing and VC formation. Consider a scenario in which field-deployed sensor nodes $1, \ldots, N$ generate a stream of image-based tuples (e.g., containing an image of a face or scene, a timestamp, and physical location) which must then be processed in conjunction with archived images for identification. λ_i is that rate at which node i generates tuples. Processing may require several steps—face detection followed by face recognition, for example. In the latter case, processing might involve a detailed comparison of the top-K signature matches between the newly generated image and images stored in the database; thus each query would require accessing the K images in the database whose signatures most closely matches the signature of the newly generated image.

One option is for a sensor node to send the tuple for query processing in the backend cloud via the hybrid infrastructure (e.g., a 4G network). This is the upper path shown in Figure 3, where a fraction $(1 - p_i)$ of node i's tuples are uploaded to the cloud. In this case, the constraining resource is the cellular upload bandwidth, shown as a multiserver queue (since uploads from different sensors may proceed concurrently), with a service time corresponding to the amount of time that a sensor uses the infrastructure to upload its image tuple.

Alternatively, a sensor node may route the tuple for processing in the virtual cluster located within the MANET. The VC is a cluster of compute-capable nodes that can serve as a "mini data center" in the MANET; we assume that the VC maintains a cache of images previously downloaded from the backend cloud. Of the K images needed for processing the tuple/query, some may be found in the cache, while others may need to be downloaded over the cellular infrastructure from the backend server. The VC will need to request those images from the server, and wait until the missing images are downloaded before performing additional processing. In the case of query processing in the VC, the constraining resources are the cellular infrastructure download bandwidth and the VC processing capacity. As shown in Figure 3 and as in the upload case, downstream cellular bandwidth could be modeled as a multiserver queue, with service corresponding to the amount of time needed to download the needed images from the backend server; VC processing is shown as a single queue in Figure 3.

B. Adaptive Query Routing

As shown in Figure 3, the primary decision to be made by a sensor node is whether to route its tuple to the backend cloud or to a VC within the MANET. In the former case, cellular network resources are required to transmit the sensor's image tuple into the cloud. We model the uplink infrastructure network as having C_{up} channels, with a query requiring an exponentially distributed channel holding time with mean $1/\mu_{up}$ to upload a tuple.

Sensor node i routes its query for processing in the VC with probability p_i. In this case, the query experiences a delay as it is routed through the MANET to the VC; MANET delays are modeled by a single queue in Figure 3; more realistic (and complicated) models can easily be accommodated in the framework of Figure 3. If the query can be satisfied from the VC's cache of downloaded images, query processing is performed entirely in the VC. Suppose that, as discussed above, each query requires K additional images in order to perform processing, and suppose that the probability that a needed image is in the cache (i.e., that an image requests hits in the cache) is h. In a simple model of the scenario in Figure 3, we can model each query as requiring an average of $(1-h)K$ images to be downloaded from the backend server.

We thus model the downlink infrastructure network as having C_{down} channels, with a query requiring an exponentially distributed channel holding time with mean $(1-h)K/\mu_{down}$ to download the needed images (i.e., images not found in the cache). Note that the choice of μ_{down} allows enough flexibility to model the VC as having a higher bandwidth downlink channel than the sensor nodes uplink channel and/or the difference in sizes between up-loaded (sensor-to-cloud) and downloaded (cloud-to-VC) images. Note that we currently assume that K and h are given, but plan to investigate caching dynamics. We also note that it may be possible to partially process a tuple given in-cache images while waiting for the remaining images to be downloaded; this could be modeled using more sophisticated synchronous fork-join operators, that have been investigated in data streaming applications in [20].

C. Online Adaptive Routing

The goal, then, is for sensor nodes to adaptively route their tuples in a congestion-sensitive manner in order to optimize some performance metric. We are currently exploring two different approaches towards adaptive online routing—minimum delay routing [8], [18] and backpressure routing [17], [13], which naturally result in distributed iterative protocols that in essence are a distributed solution to an optimization problem.

In this section, we consider the simple model shown in Figure 3, making assumptions that ease the analysis so that we may focus on the problem formulation and adaptive routing protocols. We plan to relax assumptions, seeking a better match with realistic scenarios, as we better understand the model and the adaptive routing protocols.

Minimum Delay Routing

As discussed above, "jobs" (i.e., image tuples) arrive at mean rate λ_i to sensor node i as shown in Figure 3. p_i denotes the fraction of traffic routed to the VC and $(1 - p_i)$ denotes the fraction of traffic sent to the back-end. Each job that originates from the sensor node can be thought of as a message (e.g., containing the image for which the top K image signature matches will be scrutinized in more detail) with an exponentially distributed length of mean $1/\mu_i$ bits. Each

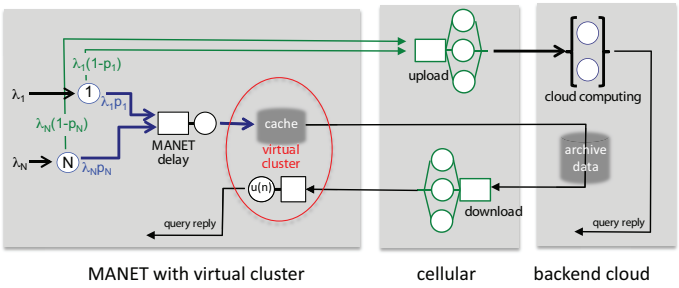

Fig. 3. Adaptive query processing model

job that arrives at the VC needs to fetch some number of images from the backend database. The length of the message containing these images is exponentially distributed with a mean of $1/\mu(M,h)$, i.e., the mean rate is a function of M (i.e, the number of top-M matches) and h is the cache hit rate (assumed to be a given and constant).

After exiting the "download queue", each job then passes through the "processor queue" at the VC. Let the service time at the processor queue have mean $1/\mu'$. Let f^u_{3G}, f^d_{3G}, and f_M be the mean arrival rate (messages/sec) at the 3G uplink, 3G downlink and MANET queue, respectively. Let f_p be the job arrival rate at the VC processor queue. Let C^u_{3G}, C^d_{3G} and C_M denote the capacities (in bits/sec) of the 3G uplink, 3G downlink, MANET link respectively. With this notation, we have:

$$f^u_{3G} = \sum_{i=1}^{N} \lambda_i(1-p_i) \tag{1}$$

$$f_M = f^d_{3G} = f_p = \sum_{i=1}^{N} \lambda_i p_i \tag{2}$$

The mean delays over the upper and lower paths, D_1 and D_2, respectively, are:

$$D_1 = \frac{f^u_{3G}}{\mu_i C^u_{3G} - f^u_{3G}} \tag{3}$$

$$D_2 = \frac{f_M}{\mu(M,h)C_M - f_M} + \frac{f^d_{3G}}{\mu_i C^d_{3G} - f^d_{3G}} + \frac{f_p}{\mu' - f_p} \tag{4}$$

The sum delay of the nodes within the network [8] is:

$$D_T = D_1 + D_2 \tag{5}$$

Minimizing the objective function in (5) in a centralized manner determines $\{p_i\}\forall i$, i.e., determines the traffic split at every sensor node that minimizes the objective function. Gallager provides a simple distributed algorithm for minimizing (5) and shows that the solution obtained from this distributed approach converges to the centralized solution asymptotically for a general network with multiple flows and links. The algorithm can be roughly described in two steps. In the first step, for each of its outgoing links, every node determines the partial derivative of the delay over the link with respect to the total flow through it. This partial derivative information is then communicated backwards from the destination to the source. In the second step the algorithm iteratively optimizes routing by deviating flow from links that have large marginal delay to links which have small marginal delay. In order to implemenet Minimum Delay routing, delay derivatives must be estimated.

Backpressure Routing

We are also exploring the use of a distributed backpressure algorithm. In traditional backpressure, every node in the network maintains a queue for every flow passing through it. On each of its outgoing links, a node then computes the difference in queue length for every flow with its neighbor on the other side of the link and then determines which flow (and how much) to transmit over that link. This approach has been shown to be throughput optimal [17], [11]. In our scenario, there are only two paths for any flow—the upper and lower paths in Figure 3. Therefore, at each sensor node, the backpressure approach answers the following question—should an incoming image tuple be routed to the "MANET queue" (and from there to the VC, which in turn will cause needed images to be downloaded) or routed to the "3G uplink queue"?. At the network queues the decision is whether or not to forward packets to the next network queue; this question too will be answered based on the difference in queue lengths between the upstream and downstream queues. While routing packets using backpressure has been shown to be optimal with respect to throughput, packets can incur unreasonably large delays at low traffic arrival rates and due to packets being circulated in loops. Recent work on backpressure has improved its delay performance [13], [11], [1], [3]. In [13], the authors demonstrate (primarily via simulation) that by changing the queuing behavior from FIFO to LIFO can decrease packet delay. Ji et al. [11] show that considering sojourn time difference of the Head-of-Line packet, instead of queue length difference can reduce packet delay while maintaining throughput optimality. Duplicate transmission of packets has been suggested in [1] to improve delay performance while maintaining throughput optimality.

The advantage of adopting Minimum Delay routing over backpressure is the objective function (delay vs. throughput). The main advantage of backpressure over Minimum Delay

routing is the amount of feedback required at a node for routing traffic. In backpressure, traffic forwarding takes place on a per-packet basis because a node has to only compute the difference in queue lengths between it and its neighbors. In Minimum Delay routing, for a node to compute the marginal delay on its links it must know the partial delay on its outgoing links and the marginal delay from its neighbors to the destinations for the various flows. This information for the whole path from source to destination for each route needs to be computed and communicated. The time scale at which the algorithms act are also different—backpressure will act is on a per packet basis while Minimum Delay routing acts at a much coarser time granularity.

III. Network-Aware Adaptive Data Stream Management in MANETs

In this section, we introduce an approach for adapting query plans of DSM applications deployed in HCNs. We consider an HCN with several mobile devices connected through multi-hop wireless ad-hoc communication. We assume that the processing operators of a query are hosted across multiple devices because individual devices cannot host the complete DSM application due to their limited processing capacity.

An important observation is that query plans capable of adapting at runtime perform better for many real-world DSM applications. This is particularly true in dynamic network environments such as HCNs, in which devices may fail and experience intermittent connectivity. As we demonstrate in §III-D, the dynamic nature of HCNs implies that a DSM system without runtime adaptation cannot achieve an efficient and robust deployment. While a query plan executes, changes in the network mean that any statically deployed query plan will eventually become outdated, leading to low performance or a failure of processing.

Existing approaches for plan adaptation [7], however, only react to changes in the characteristics of the data, such as its cardinality, distribution, or the selectivity of operators. Little prior work exists on query plan adaptation in response to changes in the network environment, i.e., the connectivity between nodes or network link bandwidths.

We propose an approach for *network-aware query plan adaptation* for DSM applications in dynamic multi-hop wireless networks. Our core idea is to enable DSM applications to exploit redundant operators to improve their performance and robustness when faced with dynamic networks. Redundancy is added by generating query plans with multiple physical operators for each operator in the dataflow graph. The number of physical operators and their mapping to nodes in the network is pre-computed based on the expected dynamicity of the network. At runtime, the execution of the query plan becomes a dynamic *routing* problem, in which data stream items (i.e., tuples) are routed at the application level over the network paths with the lowest cost. Cost is defined according to processing latency or throughout, based on the requirements of the DSM application.

The adaptation of query plans in DSM system in response to network changes raises a number of challenges, which we outline in the next section, before we discuss related work on query planning and execution (§III-B). We then describe our approach for query plan adaptation (§III-C), and report preliminary simulation results on its effectiveness (§III-D).

A. Challenges

To be useful in practice, query plan adaptation for DSM applications in HCNs must meet a number of requirements:

Expressiveness. Query plan adaptation must be able to support arbitrary dataflow graphs. In particular, it should support directed acyclic graphs with user-defined operators. We do not want to restrict DSM applications to a predefined set of operators, such as "select", "project", and "join", which would make it challenging to implement advanced DSM applications, e.g., for real-time face/scene recognition or online machine learning.

Performance. A key issue of plan adaptation is to ensure that the performance of query processing in terms of throughput and latency is not affected by runtime decisions. In addition, any adaptation mechanism requires information about the current state of the network and the query plan, which must be collected without a high communication overhead.

Correctness. Finally, query plan adaptation must always produce correct results. This becomes challenging for *stateful* operators—i.e., operators whose output depends on the history of their inputs—when there are node failures or changes to network connectivity. In addition, query plan adaptation should not violate the exactly-once delivery semantics required by many DSM applications.

B. Related Work

Much research work in the database community exists on *adaptive query processing* [7]. Unlike traditional "optimize-then-execute" query processing, the goal of adaptive planning is to use runtime feedback to modify query processing dynamically in order to improve response times or utilization. Rundensteiner et al. give an overview of the design space [14], ranging pre-computed static plans to dynamic per-tuple routing.

Our approach is most similar to the dynamic tuple routing of *Eddies* [2]. An *eddy* is a query processing mechanism that adapts the order in which tuples visit the operators in a query, while ensuring correct results. An eddy modifies the processing order on a per-tuple basis with the goal of improving efficiency, e.g., when executing multi-way joins. Our approach allows for similar fine-grained routing decisions but, instead of reordering operators, routing decisions select between redundant operators.

A hybrid approach for plan adaptation is *QueryMesh* [14]. It reduces the runtime overhead of plan adaptation by pre-computing alternative plans and then only switching between them at runtime. In QueryMesh, a classifier examines windows of tuples and matches them to a given plan to improve performance. Compared to dynamic tuple routing, plan switching

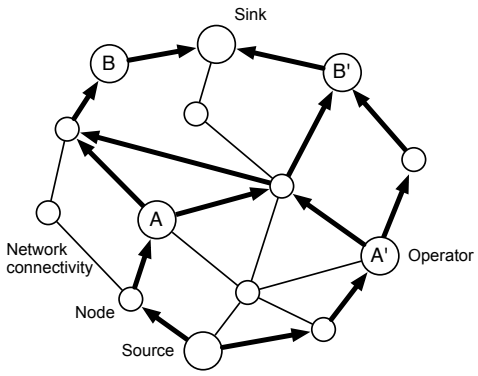

Fig. 4. Network-aware query plan adaptation using dynamic routing

is a more coarse-grained adaptation mechanism, sacrificing flexibility to achieve lower overhead. In our approach, we permit in-flight tuples to switch between operators at any point en route to the sink, resulting in fast adaptation times. Both Eddies and QueryMesh focus on plan adaptation in response to changing data patterns, and not network conditions.

A plethora of work exists on static query plan generation for DSM systems. Query planners such as SODA [19] and SQPR [12] formulate query planning and physical operator placement as an optimization problem, which can be solved using standard techniques, such as mixed integer linear programming (MILP) [10]. We can exploit these approach to generate an initial query deployment plan from a given logical dataflow graph.

In the context of DSM systems, redundant data processing has been explored to improve query processing. Hwang et al. describe how replicated plans with multiple paths between a tuple source and sink can improve reliability or reduce processing latency [9]. The main difference is that instead of sending a copy of each tuple along each path, we send only a single tuples, and paths along which successive tuples travel may differ. While replication-based approaches are likely to perform better in extremely dynamic networks, they incur the cost of redundant processing and network traffic.

For content dissemination in dynamic networks, many proposals exist on how to adapt routing trees dynamically. For example, Zhou et al. explore the problem of constructing dissemination trees that minimize the average loss of fidelity [21]. They propose a cost model that incorporates both processing and communication, and describe algorithms for incrementally transforming the topology of an application-level dissemination tree at runtime to improve performance. In contrast, we do not restrict tuples to follow a single path at any point in time but route tuples adaptively based on network conditions

C. Network-aware Query Plan Adaptation

Our approach for network-aware query plan adaptation has two steps: (1) the initial generation of query plans with redundant operators; and (2) the runtime routing of tuples along the redundant processing paths in the query plan based on network conditions.

1) Redundant query plan generation: First the DSM system must generate a query deployment plan for a given dataflow graph of operator. Based on existing query planning techniques [12], we use a query planner that obtains a suitable plan, incorporating information about the network characteristics and available resources. We assume that the initial query planning is performed in a logically centralized fashion. In future work, we plan to explore fully decentralized techniques.

Each operator in the query plan may be replicated according to a *replication factor* k, which means that k replicas of an operator are introduced in the query plan. The replication factor is chosen based on the anticipated dynamicity of the network. A large number of redundant operators means that query processing can continue, even if a substantial fraction of the query plan was lost due to failure or network disconnection.

Figure 4 shows an example of a redundant query plan. The corresponding dataflow graph consists of two operators, A and B, connecting a source node and a sink node in a linear fashion. Such a query could correspond to the continuous processing pipeline found in a face/scene recognition application. With $k = 2$, each operator is replicated twice (A' and B'), and placed independently in the network. A tuple emitted by the source node is processed correctly if it is processed by either of the replicated operators on its path towards the sink node.

2) Dynamic tuple routing: After the redundant query plan is deployed, the DSM system makes runtime decisions how to route tuples between nodes hosting operators. Each node uses a shortest path routing algorithm, such as distance-vector or link state routing [16], to determine the lowest cost path for tuples to take along a given query plan. Cost is expressed in terms of latency or throughout, i.e., preferring paths that achieve better processing performance. Based on the cost, each node makes a forwarding decisions for tuples, sending them towards one of the k redundant copies of an operator that exist in the network.

In Figure 4, the source and the operators independently decide which of the two downstream operator replicas to send a tuple to. As part of this decision, they consider the network-level routing costs, such as the number of hops traversed over intermediate nodes. For example, if path cost is defined in terms of the number of hops, operator A' has the lowest cost path to operator B'. If the path to B' becomes unavailable, it can alternatively route tuples via a higher cost path to operator B.

D. Evaluation

To evaluate the effectiveness of our approach, we implemented a prototype version in a network simulator. We use the *JiST/SWANS* wireless ad-hoc network simulation framework [5], [6]. The goal of our initial evaluation is to determine whether increasing the replication factor of a query plan allows it to handle better a network with more dynamic behavior.

We make several simplifying assumptions: (i) at the application layer, we assume that each node has perfect routing tables, which are updated instantaneously with the best route whenever a node loses or regains connectivity; and (ii) our initial experiments ignore congestion effects due to limited link capacities. We will explore more realistic network environments, together with bandwidth constraints, in future work.

Experimental set-up. We run simulations over a network with 25 wireless nodes, which form a 5-by-5 two-dimensional grid. Each node can communicate with its nearest neighbors in the cardinal directions only. Operators discard tuples if they cannot be forwarded to the next hop in the query graph because of a network failure. At the network layer, each node runs an instance of the AODV wireless ad-hoc routing algorithm [15]. All the data points obtained are the average over 5 runs with low variance.

To introduce dynamic behavior in the network, nodes alternate between an active and inactive state. The durations of the active and inactive periods are generated from an exponential distribution, whose mean is derived from a *dynamicity parameter* d between 0 and 1. We define the *mean time before failure* (MTBF) as $T(1-d)$, where T is the total simulation time. Node deactivations are scheduled according to a Poisson distribution with mean inter-arrival time MTBF: small values of d lead to fewer failures and lower dynamicity, and vice versa. Conversely, we define the *mean time before recovery* (MTBR) as Td, and reactivate nodes according to a Poisson distribution with mean inter-arrival time MTBR: here, small values of d lead to shorter sleep times and thus lower dynamicity, and vice versa.

All experiments use a single linear dataflow graph with 3 operators, connecting the source and sink. The corresponding physical query plans are generated with a replication factor k, controlling the number of replicas of each operator. Given a redundant plan, the simulator assigns the operators to network nodes at random. For future work, we plan to employ more advanced static query planning techniques [12]. At runtime, the simulator calculates the optimal next hop for each data tuple instantaneously after each change in the network state, and routes tuples accordingly.

Preliminary results. We evaluate how the replication factor k of a query plan affects the fraction of tuples processed successfully under dynamic conditions. The simulator sends tuples from the source, and outputs the percentage of tuples that reach the sink node.

Figure 5 shows the fraction of successfully processed tuples for different values of d and k. When there is no dynamicity in the network ($d = 0$), all tuples reach the sink node for all plans. As the dynamicity increases, the fraction of successfully processed tuples falls quickly for the plan without redundant operators ($k = 1$). Plans with redundant operators ($k > 1$) are also affected by an increased dynamicity, but less so than for $k = 1$. In general, the higher the replication factor k, the more tuples are processed successfully when the network is dynamic. There are, however, diminishing returns when increasing the replication factor k. This implies that it is important to take the overhead of higher replication factors into account during planning, as any increase in successfully processed tuples may be marginal, especially for less dynamic networks.

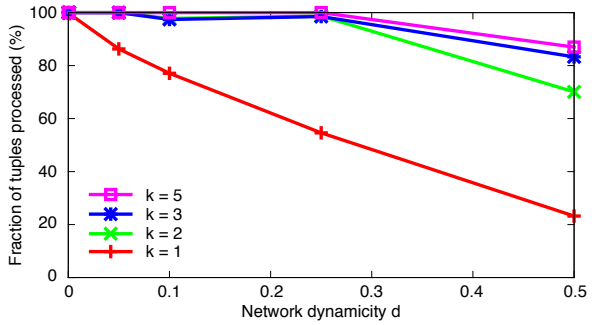

Fig. 5. Fraction of data tuples processed successfully at different degrees of network dynamicity

IV. Conclusions

In this paper, we explored the new challenges when processing continuous data streams to realize real-time applications in HCNs. We focused on two problems: (i) how to exploit a cloud-based backend infrastructure efficiently for the execution of a real-time face/scene recognition application; and (ii) how to change continuous query plans at runtime in response to changes in the network characteristics of a MANET. In contrast to existing approaches for DSM that target largely static deployment environments in well-provisioned data centers, both of our solutions to the above problems exploit runtime adaptation and exploit the tight integration of the DSM system with the network infrastructure. In future work, we plan to validate our approaches through a prototype implementation as part of a distributed DSM system deployed in an HCN-like network environment.

References

[1] M Alresaini, M Sathiamoorthy, B Krishnamachari, and M Neelyu. Backpressure with adaptive redundancy (BWAR). In *INFOCOM*, 2012.

[2] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *ACM SIGMOD Record*, volume 29, 2000.

[3] Venkataramana Badrala and Douglas Leith. On delay performance of throughput optimal backpressure routing: Testbed results. In *ICCCN*, 2011.

[4] H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, et al. Retrospective on Aurora. *VLDB Journal*, 13(4), 2004.

[5] Rimon Barr, Zygmunt J. Haas, and Robbert van Renesse. JiST: An efficient approach to simulation using virtual machines. *Software Practice and Experience*, 35(6), 2005.

[6] Rimon Barr, Zygmunt J. Haas, and Robbert van Renesse. *Scalable wireless ad-hoc network simulation*, chapter 19. CRC Press, 2005.

[7] Amol Deshpande, Zachary Ives, and Vijayshankar Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1), 2006.

[8] Robert Gallager. Minimum delay routing algorithm using distributed computation. *IEEE Trans. on Communications*, 25(1), 1977.

[9] Jeong-Hyon Hwang, Ugur Cetintemel, and Stan Zdonik. Fast and reliable stream processing over wide area networks. In *ICDEW*, 2007.

[10] IBM. ILOG CPLEX. www.ibm.com, 2010.

[11] Bo Ji, Changhee Joo, and Ness Shroff. Delay-based back-pressure scheduling in multi-hop wireless networks. In *INFOCOM*, 2011.
[12] Evangelia Kalyvianaki, Wolfram Wiesemann, Quang Hieu Vu, Daniel Kuhn, and Peter Pietzuch. SQPR: Stream query planning with reuse. In *ICDE*, 2011.
[13] Scott Moeller, Avinash Sridharan, Bhaskar Krishnamachari, and Omprakash Gnawali. Backpressure routing made practical. In *INFOCOM*, 2010.
[14] Rimma V. Nehme, Karen Works, Chuan Lei, Elke A. Rundensteiner, and Elisa Bertino. Multi-route query processing and optimization. *Journal of Computer and System Sciences*, 79(3), 2013.
[15] C. Perkins, E. Belding-Royer, and S. Das. Ad-hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, 2003.
[16] Andrew Tanenbaum. *Computer Networks*. Prentice Hall, 5th edition, 2010.
[17] L Tassiulas and A Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. on Automatic Control*, 37(12), 1992.
[18] Srinivas Vutukury and J.J. Garcia-Luna-Aceves. A simple approximation to minimum-delay routing. In *SIGCOMM*, 1999.
[19] J. Wolf, N. Bansal, K. Hildrum, S. Parekh, et al. SODA: An optimizing scheduler for large-scale stream-based distributed computer systems. In *Middleware*, 2008.
[20] H Zhao, C Xia, Zhen Liu, and D Towsley. Distributed resource allocation for synchronous fork and join processing networks. In *INFOCOM*, 2010.
[21] Yongluan Zhou, Beng Chin Ooi, and Kian-Lee Tan. Disseminating streaming data in a dynamic environment: an adaptive and cost-based approach. *VLDB Journal*, 17(6), 2008.