Decision-driven Execution: A Distributed Resource Management Paradigm for the Age of IoT

Tarek Abdelzaher*, Md Tanvir A. Amin*, Amotz Bar-Noy†, William Dron‡, Ramesh Govindan§, Reginald Hobbs¶, Shaohan Hu*, Jung-Eun Kim*, Jongdeog Lee*, Kelvin Marcus¶, Shuochao Yao*, and Yiran Zhao*
*University of Illinois at Urbana-Champaign, Urbana, IL 61801
†City University (CUNY), New York, NY 10016
‡Raytheon BBN Technologies, Cambridge, MA 02138
§University of Southern California, Los Angeles, CA 90089
¶U.S. Army Research Laboratory, Adelphi, MD 20783

***Abstract*—This paper introduces a novel paradigm for resource management in distributed systems, called *decision-driven execution*. The paradigm is appropriate for mission-driven systems, where the goal is to enable faster, leaner, and more effective decision making. All resource consumption, in this paradigm, is tied to the needs of making decisions on alternative courses of action. A point of departure from traditional architectures lies in interfaces that allow applications to specify their underlying decision logic. This specification, in turn, allows the system to reason about most effective means to meet information needs of decisions, resulting in simultaneous optimization of decision accuracy, cost, and speed. The paper discusses the overall vision of decision-driven execution, outlining preliminary work and novel challenges.**

I. Introduction

This paper introduces a new resource management model, suitable for the sensor rich, distributed, data intensive applications of the Internet of Things (IoT). We call it, *decision-driven execution*. It re-thinks scheduling and communication to consider application models that are more pertinent to the emerging data-rich IoT environments.

In traditional control loops, a basic loop might contain a sensor, controller, and actuator, used sequentially upon the occurrence of a corresponding stimulus, leading to time-driven and event-driven scheduling models (depending on whether the control is to be executed periodically or upon occurrence of events of interest). *In contrast*, the vision we entertain in this paper is one motivated by the emerging world of connected "things", where sensors are ubiquitous and the number of embedded sensing devices per user is very large. There might not exist an *a priori* structure (such a fixed control loop) statically imposed on these devices. Rather, they will generally be connected to shared media, allowing flexible reconfiguration to perform (both one-off and repetitive) tasks. For example, consider the myriad of sensing, storage, and computing devices that comprise a megacity's cyber-infrastructure. Many sensors will be embedded in the physical environment. Different application tasks may need subsets of these devices. A sensor is activated when the user needs to perform a task that needs data from that sensor. Resources are marshalled only when a relevant application needs to make a decision. In other words, resource consumption is decision-driven.

The decision-driven execution paradigm offers an exciting foundation for rethinking resource management, where the purpose is to aid application tasks (such as controllers and actuators) in making decisions on a course of action by supplying requisite data and executing decision results. The bulk of resource management in the system lies in arbitrating the acquisition and movement of data among processing components. This arbitration is guided by a novel set of questions. For example, what data would be more relevant for making a decision? Which sensors are most appropriate for collecting such data? When should the data be collected to meet freshness needs of the decision? What data should be cached in the distributed system, and which nodes should store it to support aggregate decision needs most efficiently? At least three conditions must be met by the resource management algorithms; (i) the data collected for making a decision must be of sufficient quality to support the decision, (ii) it should be acquired sufficiently recently such that it is not stale by the time the decision is made, and (iii) the decision made based on collected data must meet the relevant decision deadline. A distributed system that supports the decision-driven resource management paradigm must optimize communication, storage, and scheduling to meet these constraints.

Data that constitutes the primary objects of interest in a decision-driven execution system implicitly links together physical and cyber constraints. The data of interest typically comes from sensors, and as such captures aspects of the physical state of the world. Since world state is dynamic, data objects have expiration time constraints after which they become stale. The scheduling of data acquisition must obey these constraints. At the same time, as stated above, decisions have deadlines after which the window of opportunity to act will have passed. The system should therefore be cognizant of both the constraints arising from data freshness needs, as well as those arising from decision deadlines. Both sets of constraints are a function of models of the physical world. Therefore, the combination of these sets of constraints leads to interesting new scheduling problems, where the intellectual innovations arise from simultaneously addressing requirements from the cyber realm (e.g., resource capacity constraints) and requirements from the physical realm (e.g., data freshness).

Recent work has made initial progress at solving the above cyber-physical resource management problems [1]–[3].

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.318

The work bears resemblance to prior database research that considered explicit data access transactions and required a degree of data freshness. However, a plethora of exciting new research challenges remain that go beyond the above work.

The rest of this paper is organized as follows. Section II describes the decision-driven system architecture. Section III overviews decision-driven resource management challenges. Section IV elaborates challenges in real-time scheduling that arise in the decision-driven context. Section V details networking challenges. A proof of concept implementation and evaluation are described in Section VI and Section VII, respectively. A brief discussion of other remaining research questions is presented in Section VIII. The paper concludes with Section IX.

II. A Decision-driven System Architecture

Below, we develop a general vision for what a decision-driven resource management paradigm might look like, and what components need to be involved. We begin by exploring what a *decision* means. Clearly, a model of decision-making is needed in order to develop a decision-centric execution paradigm.

We start with simple models in order to lay an initial groundwork for investigation. The choice of decision model itself gives rise to interesting research questions that warrant further attention. In the initial simple model, we view decisions as choices of a course of action among multiple alternatives. The viability of each individual alternative depends on the satisfaction of several predicates. Making a choice can be thought of as an evaluation of a logical expression of multiple predicates. For example, "if it is (i) sunny and (ii) warm, I will not wear a sweater". Evaluating a predicate requires acquisition of corresponding evidence. Data objects such as images, videos, or sound clips, generated by appropriate sensors, can supply the needed evidence. Often, a piece of evidence (e.g., a picture that shows whether it is sunny or rainy) can be supplied by any of several alternative sources, such as multiple cameras overlooking the scene. In order to know how to furnish data needed for a decision, our system must use models of decisions (that specify the underlying logical expressions) and models of sources (that specify who can supply which pieces of evidence) to optimize delivery of objects that maximally help the decision-maker choose the right course of action.

A decision-driven resource management system allows applications to make queries we call *decision queries*, or decision tasks, that request information needed for a decision. The system comprises nodes that contribute, request, or help forward data needed for these decisions. It manages the acquisition of evidence needed to evaluate the viability of different courses of action involved in decision-making. By accounting for models of decisions and sources, the system carries out the required information collection and transmission in a more efficient and timely manner to support decision making.

A. Exploiting Decision Structure: An Illustrative Example

A key innovation of the decision-driven system lies in a novel query interface that allows applications to express decision needs in a manner that helps the resource management components properly priorite data acquisition. Specifically, a query may specify a logical expression that describes the decision structure. This expression specifies the predicates that need to be evaluated for the corresponding choice (of a course of action) to be made. In this model, there are no limits on the types of queries that can be expressed as long as they can be represented by Boolean expressions over predicates that the underlying sensors can supply evidence to evaluate.

There are many possible ways that such expressions could be obtained. In many applications, especially those involving liability or those where human teams must operate efficiently under adverse or dangerous conditions, a well-prescribed operation workflow is usually followed. The workflow specifies how individuals should act, under which conditions a given course of action is acceptable, and what checks must be done before emparking on an action. Training manuals, rules of engagement, doctrine, standard operating procedures, and similar documents describe these workflows, essentially documenting acceptable decision structures. Decision logic could also be learned by mining datasets that describe conditions observed and decisions taken on them by an authority. Such an approach, for example, may be used to reverse-engineer strategy used by an expert or by an adversary. Finally, in some cases, decision logic could be algorithmically derived. For example, in a vehicular navigation application, the driver will generally seek a route that satifies some machine-checkable property, such as a condition on expected commute time, quality of route, or length of commute. Hence, the logic for the decision on route from alternatives on a given map is known. An interesting research question is: given the logical decision structure (i.e., the graph of logical predicates to be evaluated to arrive at a course of action), how best to deliver the requisite information?

Let us look at a toy example to help make the picture more concrete. Suppose after an earthquake that hits our smart city, there is a shortage of air support, and an emergency medical team needs to transport a severely injured person from an origin site to a nearby medical center for surgery. There are two possible routes to take: One composed of segments A–B–C, and the other of segments D–E–F. We need to make sure that the chosen route is in good enough condition for our vehicle to pass, so we want to retrieve pictures from deployed roadside cameras in order to verify the road conditions and aid our decision-making on which route to take. Our route-finding query can be naturally represented by the logical disjunctive norm form $(viable(A) \wedge viable(B) \wedge viable(C)) \vee (viable(D) \wedge viable(E) \wedge viable(F))$, where $viable(X)$ represents the predicate "route segment X is viable". This expression signifies that at least all segments of one route need to be viable for the transport to occur. In this example, if road segments A, B and C all turn out to be in good condition, then the first route is viable, and there is no need to continue retrieving pictures for road segments D, E, and F. Similarly, if a picture of segment A shows that it is badly damaged, we can skip examining segments B and C, as this route isn't going to work anyway. Instead, we can move on to explore segments D, E, and F.

As is evident from this toy example, exploiting decision structure (represented by the Boolean expression) enables us to take inspiration from heuristics for short-circuiting the evaluation of logical expressions to schedule the acquisition of evidence. Specifically, we can acquire evidence in an order that statistically lowers expected system resource consumption needed to find a viable course of action. By incorporating additional meta-data (e.g., retrieval cost of each picture, data validity intervals, and the probability of each road segment being in good or bad condition), we can compute retrieval schedules that better optimize delivery resources expended to reach decisions. This optimization, indeed, is the main research challenge in the decision-driven execution paradigm.

This optimization must consider both physical and cyber models. On one hand, models of the underlying physical phenomena are needed to correctly compute inputs such as data validity intervals (how long can one consider measurements of given physical variables fresh), and environmental conditions (e.g., probablities that some measurements not yet acquired will fall into a range that invalidates versus supports a predicate). One the other hand, models of computing and communication resources are needed to understand how much bandwidth and compute power are available for data collection from the physical world.

The latter models can be obtained from network and other resource monitoring. The former are more difficult to obtain. They can be learned over time or derived from the physical nature of the phenomena in question. For example, temperature does not change very quickly. Hence, the validity interval of a temperature measurement could be of the order of large fractions of an hour. On the other hand, state during an active emergency, such as a burning building, can change on the order of minutes. Hence, its validity interval is much shorter. It is also possible for external events to invalidate freshness of variables. For example, the existence of a resource, such a bridge across a river, can be assumed to hold with a very large validity interval. However, a large earthquake or a military air-raid may invalidate such past observations, making them effectively stale and in need of being re-acquired from sensors. The same applies to learned probabilities of conditions. The probability of traffic congestion on some freeway at 11pm on a Monday night might be known. However, a condition, such a nearby large concert that ends around the same time, can invalidate it. In general, a combination of past contextual knowledge, current observations, and invalidations will be needed to operationalize the physical models.

Lowering the data acquisition costs of decisions involves carrying out an optimal collection strategy given the resources available and the underlying physical models, such that a measure of decision correctness is maximized, while cost is minimized. If some contextual information needed for the models is not known, the optimization may proceed without it, but the quality of solutions will be lower, generally entailing a less than optimal resource cost. The sensitivity of decision cost to the quality of models supplied is itself an interesting research problem.

B. System Abstractions and Components

The decision-driven execution system represents the physical world by a set of labels (names of Boolean variables). These labels can be used in expressions of decision logic structures. The system maintains tuples of ($label$, $type$, $value$), where $label$ is just an identifier (i.e., variable name), the $type$ specifies the semantic type of the label (for example, "road condition"), and $value$ could be *true*, *false*, or *unknown*. The system can be easily extended to more general types (other than Boolean). More general discrete variables can be implicitly represented by sets of labels, one label for each allowed value of the variable, with the restriction that only one of these can be true at a time. Continuous variables can be supported as long as actions are predicated on some thresholds defined on these variables. For example, the decision to turn the lights on in a smart room can be predicated on the value of an optical sensor measurement dropping below a threshold. This is a Boolean condition whose evaluation result can be stored in a variable labeled, say, Dim. The pool of labels itself can be dynamic. New applications can add new labels (and new categories of labels) to the pool and specify sensing modalities needed to determine label values. For instance, in the routing example above, the predicate $viable(X)$ can be represented by the label $viableX$, denoting a Boolean variable of value *true* (if the route segment is viable) or *false* (if it is not). The route selection decision is associated with labels $viableA$, $viableB$, ..., $viableF$.

To determine the value of a label (e.g., whether conditions of a road segment make it a viable candidate), evidence must be collected. An example of such evidence might be a picture of the corresponding road segment. We call such evidence items *evidence objects* or simply *data objects*, where it is clear from context that the data in question offers evidence needed to evaluate a logical predicate in the decision structure.

Evidence objects are data objects needed for deciding the value of labels. Entities that examine evidence in order to determine the value of a label are called, in our architecture, *annotators*. For example, an annotator could be a human analyst receiving a picture of route segment A, and setting the corresponding label, $viableA$, to *true* or *false*, accordingly. Alternatively, an annotator could be a machine vision algorithm performing the same function. In general, annotators should advertise the type of evidence objects they accept as input, and the types of labels they can accordingly compute. Clearly, the same object can be used to evaluate several different labels. For example, a picture of an intersection can be used to evaluate physical road conditions. However, it can also be used to detect specific objects such as individual vehicles, license plates, or pedestrians, or used to estimate values such as length of traffic backup, traffic speed, or congestion level.

Another key component of the decision-driven resource management paradigm is the data sources. Sources that originate data, such as sensors, must advertise the type of data they generate and the label names that their data objects help resolve. For example, a source might offer pictorial evidence of road conditions. Such a source would advertise both its data type (say, JPEG pictures) and the specific geographic locale covered. In the route discovery example, this source would need to be paired with an annotator that can accept pictures as input and determine viability of road segments within that geographic locale.

Finally, an important component is network storage or caches. The decision on mapping data and computation to network nodes in a distributed execution environment is a classical problem in distributed computing systems. This problem must be solved in the context of a decision driven execution as well. Content (both data objects and annotation labels) should be cached at nodes closer to consumers who might need these objects and labels for their decision-making. Similarly, annotators will need to execute on nodes that are close to consumers needing the annotations. The placement of data and computational modules in the network to minimize decision cost remains an open problem.

The aforementioned architecture effectively changes the query paradigm from specifying *what* objects to retrieve to specifying *why* they are needed; that is to say, how they fit in the logic used to make a decision. This shift is thanks to sharing the structure describing the query's decision logic. Evidence objects are needed to resolve predicates named by labels in that decision logic. The architecture allows the network to be much smarter when answering a query. Being aware of the logical decision structure, the resource management system can allocate resources to seek evidence that helps evaluate the decision expression at the lowest cost. As alluded to in the introduction, we can take inspiration from literature on optimizing the evaluation of logical expressions to determine which labels should be evaluated first and which sources should be contacted for the corresponding evidence. In turn, this determination informs resource allocation, such as policies for scheduling/queuing of object retrieval requests, policies for caching of results, and choices governing invocation of annotators.

C. A Walk Through the Execution of a Decision Query

Putting it all together, when a user makes a decision query, at a high level, query resolution works as follows. The system first determines the set of predicates (i.e., labels) that is associated with the query from the underlying Boolean expression that describes the decision logic. This is the set of labels whose values need to be resolved. The query source then needs to determine the set of sources with relevant evidence objects. If multiple sources offer redundant evidence, some arbitration is needed to determine who to contact. A scheduling algorithm must decide on the order in which evidence objects must be retrieved to evaluate the different labels.

The system must manage caching. Say, the query source decides to resolve the value of the label, $viableX$. If the label has already been evaluated in the recent past (because of a prior query), its evaluation may be cached in the network, in which case the resolved value can be found and returned. This is the cheapest scenario. Otherwise, if the evidence object needed to evaluate the predicate has been recently requested (but the corresponding label not evaluated), the requested object may be cached. Such might be the case, for example, when the object was requested to evaluate a different predicate. The cached object needs to be sent to the right annotator to determine the label value relevant to the current query. Otherwise, if the objects is not cached or is stale, the query should be propagated to a source that has fresh relevant objects. The relevant object is then shipped to an annotator that decides label values. Both the object and the computed new labels are cached in the network with a freshness interval that specifies their validity for future use. Next, we outline the research challenges that must be addressed in realizing this architecture.

III. Decision-driven Resource Management: Optimizing Retrieval Cost

Initial work on decision-driven resource management was recently published in the context of centralized systems [3], [4]. It needs to be extended to a more general decision model and to distributed resource management. Consider a workload model, where tasks consume resources to make decisions, each represented by a logic expression in disjunctive normal form (OR of ANDs). Let $\{a_i\}$ denote the set of alternative courses of action for the ith decision, and $\{b_{i_j}\}$ denote the jth Boolean condition needed to determine the value of a_i. Therefore, a query q takes the general form:

$$q = \underbrace{(b_{0_0} \wedge b_{0_1} \wedge \ldots)}_{a_0} \vee \underbrace{(b_{1_0} \wedge b_{1_1} \wedge \ldots)}_{a_1} \vee \ldots .$$

The first challenge lies in designing algorithms that optimize the cost of retrieving evidence objects needed to resolve the decision query. In the simplest model, the query is resolved when a single viable course of action is found. Other more nuanced models may be possible. For example, a query could be resolved when a viable course of action is found for which additional conditions apply that may be represented by another logical expression structure ANDed with the original graph.

A. Minimizing Retrieval Cost by Short-ciruiting

Associated with each condition b_{i_j} may be several pieces of metadata. Examples include (i) retrieval cost C_{i_j} (e.g., data bandwidth consumed), (ii) estimated retrieval latency l_{i_j}, (iii) success probability p_{i_j} (i.e., probability of evaluating to *true*), and (iv) data validity interval d_{i_j} (i.e., how long the data object remains fresh). The question becomes: how to orchestrate the retrieval such that the query is resolved at minimum cost?

Sequential retrieval of evidence objects gives the most opportunity to take advantage of the decision logic structure to short-circuit and prune unnecessary retrievals in view of previously retrieved objects. Simply put, when handling an AND,

$$a_i = b_{i_0} \wedge b_{i_1} \wedge b_{i_2} \wedge \ldots ,$$

we want to start with the *most efficient* b_{i_j} and proceed downwards. Here, "most efficient" means *highest short-circuit probability per unit cost*

$$\frac{1 - p_{i_j}}{C_{i_j}}.$$

Imagine a particular course of action whose viability depends on just two conditions, h and k, that require retrieving and examining a 4 MB and a 5 MB audio clip, respectively. It has been estimated (e.g., from historic data or domain expert knowledge) that condition h has a 60% probability of being true, whereas k has a 20% probability. In this case, we would want to evaluate k first, as it has a higher short-circuiting probability per unit bandwidth consumption. Intuitively, this is because it is more likely to be false, thereby producing a result that obviates retrieval and evaluation of the remaining ANDed primitives. More precisely:

$$\underbrace{\frac{1 - 0.2}{5}}_{0.16} > \underbrace{\frac{1 - 0.6}{4}}_{0.1}.$$

Hence, this evaluation order leads to a lower expected total bandwidth consumption compared to the other way around (i.e., evaluating h before k)

$$\underbrace{5 + 0.2 \times 4}_{5.8} < \underbrace{4 + 0.6 \times 5}_{7}.$$

Similarly, for the handling an OR in the logic structure:

$$q = a_0 \vee a_1 \vee a_2 \vee \ldots,$$

we start processing the a_i with the highest short-circuiting probability per unit cost; in this case, one that has the highest probability of evaluating to true.

Conditions in the physical world can change over time. Therefore, it is important that, at the time a decision is made, all pieces of information involved must still be fresh. Otherwise, decisions will be made based on (partially) stale information. A greedy algorithm has been proposed [3], where all data object requests are first ordered according to their validity intervals (longest first) to meet data expiration constraints, then rearrangements are incrementally added, according to objects' short-circuiting probabilities per unit cost, to reduce the total expected retrieval cost.

The approach is heuristic and does not have a known approximation ratio. Near optimal algorithms should be investigated. Unlike early work that considers object retrieval over a single channel, it is interesting to extend the formulation to consider more general network topologies. Importantly, this retrieval order is influenced by models of the physical world that determine how fast physical state changes, and thus how often it needs to be sampled. Such models will be incorporated into the optimization to refine expressions of short-circuit probability. Specifically, whether or not a retrieved object short-circuits an expression depends not only on the value of the corresponding predicate evaluation, but also on when the evaluation was carried out. Stale evaluation results are not useful. Hence, the optimization must be cognizant of timing constraints derived from physical models of the underlying measured phenomena.

B. Minimizing Retrieval Cost by Optimizing Coverage

Another interesting question in minimizing the cost of object retrieval lies in selecting the sources from which objects should be retrieved, as well as the annotators needed to compute predicate values from the supplied evidence. Three interesting challenges arise in the context of this optimization.

First, in general, multiple sources may offer evidence objects that help evaluate the same or overlapping subsets of predicates needed for resolving a decision query. Some evidence objects may lead to evaluating multiple predictates at once. In our running example of route finding, a single picture from an appropriate camera can help evaluate conditions on multiple nearby road segments at once, if all such segments are in the camera's field of view. Hence, to determine the most appropriate sources to retrieve evidence from, one must solve a source selection problem. This problem can be cast as one of coverage. It is desired to cover all evidence needed for making the decision using the least-cost subset of sources. Variations of this problem will be investigated in the proposed work.

Second, an interesting novel factor in our resource management model is the existence of annotators. Not only do we need to collect evidence objects, but also we want to use them to determine specific predicate values. As mentioned earlier, an annotator could be a human, in which case one must consider the cost of delivering the collected evidence to that human for annotation. Alternatively, the annotator could be a machine. When the annotator is the query source, all evidence must simply be shipped to that source for both annotations and decision making. In this case, we assume success at resolving the query as long as all evidence objects can be shipped by the decision deadline and remain fresh at that deadline. When the annotator is a piece of software, we other challenges arise. For example, where in the distributed system should that software be located to minimize decision cost? Besides considerations of network cost, how to account for processing factors such as load balancing on the annotators?

Finally, there is the issue of confidentiality and trust. A user might not trust the accuracy of specific annotators or might not wish to send specific evidence objects to them for confidentiality reasons. Such additional constraints will be incorporated into the optimization algorithm. To address trust, the label values computed by different annotators will be signed by the annotator. Such signatures can be used to determine if a particular cached label meets the trust requirements of the source. Similarly, labels can note which objects the annotator used to make their annotation decision. That way, trust becomes pairwise between the annotator and the source. If an annotator requires multiple pieces of data to solve a predicate, then all are stored in the label. In JSON, one can think of the following label format:

```
{
 "label":"viableX",
 "type":"road condition",
 "value":true,
 "annotator":"/BBN/boston/bldg9/photo_analysis_v2.39",
 "sources": ["/city/marketplace/south/noon/camera1",
            "/city/marketplace/north/dawn/camera5"]
}
```

IV. Real-time Decision-driven Scheduling

The architecture described in the previous section inspires opportunities to develop a new type of real-time scheduling theory, we call *decision-driven scheduling*. The objective of a decision-driven scheduling algorithm is to schedule the retrieval of data (evidence) objects needed for current decision queries.

The retrieval schedule must obey several constraints. First, decisions must be carried out by their respective deadlines, leading to *deadline constraints*. Second, at the time that a decision is made, the data or labels it is based on must be fresh. Since retrieved evidence eventually becomes stale due to changes in the physical state of the underlying phenomena, the latter requirement leads to *data validity constraints*. We say that each retrieved object has a *validity interval* after which it is no longer guaranteed to be accurate. Decisions must be made while the objects they need are within their validity intervals.

A. Initial Results

Simple versions of the above problem have been solved in recent work [1], [2]. For example, consider the basic case of a single task (i.e, decision query) deciding the viability of a single course of action, where the underlying data objects are retrieved over a single resource bottleneck. For simplicity, let the source also be the data annotator. Hence, the system must simply deliver all evidence objects to the source by the decision deadline.

In this scenario [1], one needs to retrieve N data objects $O_1, \ldots, O_N$ from corresponding sensors, $S_1, ..., S_N$. The sensors might normally be off. Once activated, they sample their environment periodically, at period I_i, equal to the validity interval of the sensor measurement. Delivering a measurement from sensor S_i (i.e., object O_i) takes bandwidth C_i. Let t_i denote the activation time of sensor S_i, which is also the time its data is sampled. Subsequent samples will occur at times $t_{ik} = t_i + kI_i$ (where k is an integer). Once all needed data objects are retrieved from all N sensors via the communication medium, the decision can proceed. At that time, the sensors are deactivated. Let the time instant at which all decision data has been fetched be denoted by F. We shall henceforth call it the *decision time*. We require that $F \leq D$, where D is the decision deadline.

Let the optimal retrieval policy be one that chooses activation times, t_i, such that cost is minimized. In the absence of short-circuit opportunities, each object must be retrieved at least once. Hence, the optimal cost is:

$$Cost_{opt} = \sum_{1 \leq i \leq N} C_i \tag{1}$$

It occurs when no sensor is sampled twice. Let a feasible retrieval schedule be one that satisfies the decision deadline. Recent work has shown that *if any feasible retrieval schedule exists, then a feasible retrieval schedule exists with a cost exactly equal to $Cost_{opt}$*. This is because, at the time the decision is made in any schedule, only one sample from each sensor is within its validity interval. For purposes of that one decision, other previous samples from the same sensor need not have been retrieved, as they would not be used.

The above suggests that the aforementioned schedulability problem can be cast as one of finding sensor activation times t_i and a retrieval order such that decision cost is exactly $Cost_{opt}$. If no solution is found, the problem is unschedulable. Accordingly, the *optimal retrieval scheduling policy* as one that finds a retrieval order that meets the two constraints below, whenever any other policy does:

Data freshness: $t_i + I_i \geq F \;\; (\forall i, 1 \leq i \leq N)$,
Decision deadline: $t + D \geq F$,

where the decision query arrives at time t. The freshness constraint above ensures cost minimality. If it is violated, a second sample is taken from the sensor, which makes the cost non-optimal. These can also be represented together as:

$$\min\big(\min_{1 \leq i \leq N}(t_i + I_i),\;\; t + D\big) \geq F$$

Prior work [1] shows that the optimal solution to the above problem for a single decision query and a single communication channel is the *Least Volatile object First* (LVF). In this policy, the object with the longest validity interval is retrieved first over the shared channel. The same work also determined an optimal retrieval policy for multiple independent decision queries, assuming that each query is deciding the viability of a single course of action, the sets of evidence objects needed for the different queries are non-overlapping, and there is a single resource bottleneck over which objects are retrieved. The work proved that the optimal retrieval policy falls in the category of *hierarchical scheduling*, where non-overlapping priority bands are first assigned to different decision queries, then objects needed for a given query are prioritized within its band. In other words, priority assignment is hierarchical. First, query-level priorities can be decided. Second, within a query, a sequence for object retrieval can be defined.

The optimal algorithm, as shown in prior work [1], assigns the highest priority to the query with *the smallest value of the minimum of its object validity expiration times and its decision deadline*. Within a query, it retrieves object pertinent to that query in the *Least Volatile object First* (LVF) order.

B. Remaining Challenges

The above work has several limitations. First, while it does consider multiple decision queries, they are assumed not to overlap in the sets of data objects they need. Second, the decision for each query involves evaluation of validity of only a single course of action. Hence, there is no disjunction in the decision model. Short-circuit opportunities are not considered. Finally, all objects needed for making the respective decisions are assumed to be retrieved over a single channel, essentially reducing the problem to one of single-resource scheduling. To establish a general theory of decision-driven scheduling these limitations need to be removed. This leads to several avenues of investigation:

- *Non-independent queries:* It is important to consider the case where some queries overlap in needed data objects. In this case, retrieving each object once is not optimal anymore. That is because, if an object is shared by multiple queries, there is a possibility that the same data object can be reused. Such reuse can reduce total cost. At present, the optimal solution to this problem is unknown. Algorithms with near optimal performance are needed. They should be further extended to account for more complex decision models (i.e., multiple courses of action) and short-circuit opportunities.
- *Noisy sensor data:* The challenge here is to adapt prior algorithms to the case where sensor data is not clean. Hence, it might not be enough to retrieve a single piece of evidence to evaluate some label. Rather, multiple pieces may be needed to corroborate the computed label to a specified degree of confidence. The need for such corroboration has implications on source selection and data retrieval schedules. Requirements for confidence in computed predicate values, in the presence of noisy data, lend themselves nicely to the formulation of new scheduling problems, where

the right amount of evidence must be retrieved to guarantees a level of confidence in decision results. Annotators, in this scenario, may need to examine multiple pieces of evidence (e.g., multiple pictures) to determine the value of a particular label (e.g., whether a route segment is viable). Once the label value is determined, annotators can offer feedback on the quality of individual inputs used. For example, they may mark a given picture (and hence, its source) as not useful. Such feedback can accumulate to gradually build profiles for reliability of sources. In turn, these profiles may be considered in future source selection problems to avoid bad sources or seek sufficient corroboration such that a required level of confidence in results is attained. The problem gets more complicated by considering reliability of annotators. A bad annotator could offer false feedback that improperly influences the reliability profile of a source. Hence, individual query originators may develop different profiles for the same data sources, depending on which annotators they trust.

- *Event-triggered decision-making:* In many scenarios, the need for decision-making itself will be triggered by sensor values. For example, the firing of a motion sensor inside a warehouse after hours may trigger a decision task to determine the identity of the intruder. Other decisions may need to be done periodically. The scheduling problems described above can thus be augmented by analysis that takes into account decision triggers, offering a better model of expected future workload, such as periodicity, or specific contexts in which the decision query will arrive.

V. Network Challenges

In a distributed system where decision tasks can originate at different nodes and where evidence needed to make a decision may be distributed, it is important to address the underlying networking challenges. Specifically, how do we find sources who have evidence pertaining to the decision? Where to cache objects as they are retrieved from those sources? When objects are processed by annotators to generate values for one or more labels, where should these values be stored? Answers to these questions are needed in the context of three mechanisms, below.

A. Hierarchical Semantic Naming and Indexing

Since decision-driven resource management is centered around data retrieval, it seems natural that some form of information-centric networking can be implemented to facilitate routing queries and finding matching objects [5], [6]. In information-centric networks, such as NDN [7], data, not machines, are the primary named entity on the network. The network adopts hierarchical data names, instead of hierarchical IP addresses. In this paradigm, consumers send low-level queries, called *interest packets*, specifying a data name or name prefix. Routing tables directly store information on how to route interests to nodes who previously advertized having data matching a name prefix. Hence, interests are routed directly to nodes that have matching data. The data then traverses the reverse path of the interest to return to the query originator.

Adaptations of the information-centric networking ideas can furnish the underlying framework for routing queries to sources in the decision-driven execution architecture. In an NDN-like implementation, evidence objects, labels, and annotators all have public names in an overall name space. Nodes possessing those objects advertize their names. Nearby routers who receive those advertizements update their tables such that interests in the given names are correctly forwarded to nodes that have matching objects. Since labels encode the semantics of the underlying variables, we call the resulting scheme hierarchical semantic indexing.

In designing hierarchical name spaces (where names are like UNIX paths), of specific interest is to develop naming schemes where more similar objects have names that share longer prefixes. This naming scheme will allow the network do clever object substitutions, when approximate matches are acceptable. For example, when a query arrives for an object `/city/marketplace/south/noon/camera1/`,
if retrieving this object is impossible or costly, the network may automatically substitute it with, say, `/city/marketplace/south/noon/camera2/`.
This is because the large shared name prefix signifies that the latter object is very similar to the former (e.g., a view of the same scene from a different angle). Hence, it is a valid substitution when approximate answers are allowed. This mechanism may lead to substantial resource savings and more graceful degradation with overload. In fact, it may offer a new foundation for network congestion control, where requirements on the degree of acceptable approximation are relaxed as a way to combat congestion and tightened again when congestion subsides.

B. Information-maximizing Publish-Subscribe

Building on the aforementioned hierarchical semantic indexing, it becomes possible to develop network resource management protocols that maximize information flow from sensors to decision tasks. The importance of delivering a piece of information is not an absolute number, but rather depends on other information delivered. For example, sending a picture of a bridge that shows that it was damaged in a recent earthquake offers important information the first time. However, sending 10 pictures of that same bridge in the same condition does not offer 10-times more information. Indeed, the utility of delivered information is *sub-additive*. This observation has two important implications; namely:

- Data triage cannot be accurately accomplished by assigning static priorities to data packets, as the importance of one piece of information may depend on other information in transit.
- Data triage cannot be accurately accomplished at the data source, as the source may be unaware of other sources supplying similar information.

The above two points argue for implementing data triage in the network. An information-utility-maximizing network must perform data triage at network nodes to maximize the delivered (sub-additive) information utility in the face of overload. Our premise is that a network that explicitly supports hierarchical names for data objects (as opposed to hierarchical IP addresses for machines) can directly maximize and significantly improve delivered information utility. In a well-organized hierarchical naming scheme, objects with hierarchical names that share a longer prefix are generally closer together in some logical similarity space. Assuming that items closer together in that space share more information in common, distances between them, such as the length of the shared name prefix, can be leveraged to assess redundancy in sub-additive utility maximization. Since content names are known to the network, fast greedy sub-additive utility maximization algorithms can be implemented on links and caches. For example, the network can refrain from forwarding partially redundant objects across bottlenecks; it can cache more dissimilar content, and can return approximate matches when exact information is not available. The above intuition suggests that naming data instead of hosts lays a foundation for information utility maximization and for improving network overload performance.

C. Support for Different Task Criticality

Importantly, network resource management mechanisms must support tasks of different criticality. In a network that directly understands content names, it is easy to implement different content handling policies that depend on the content itself. Some parts of the name space can be considered more critical than others. Objects published (i.e., signed) by an authorized entity in that part of the name space can thus receive preferential treatment. There objects, for example, can be exempt from the aforementioned approximation mechanisms for congestion control. They can also recieve priority for caching and forwarding. The integration of such preferential treatment mechanisms with the scheduling problem formulation described earlier is itself an interesting research problem.

VI. Implementation

To perform a proof-of-concept validation, we implemented a distributed system, called Athena, that embodies the decision-driven execution paradigm.

A. Query Requests

In this implementation, a user can issue query request(s) at any Athena node, using a *Query_Init* call. At each node, upon user-query initiation, Athena translates the query into the corresponding Boolean expression over predicates, and starts carrying out necessary predictate (label) evaluation. This processing is done in the context of *Query_Recv*. The component reacts to received queries (either initiated locally or propagated from neighbor nodes) by carrying out the following execution steps: (i) add the new query to the set of queries currently being processed by the node, (ii) determine the set of sources with relevant data objects using a semantic lookup service [8], [9], (iii) compute the optimal source subset using a source selection algorithm [10], (iv) send the Boolean expression of the query to neighbors and (v) use a decision-driven scheduler to compute an optimal object retrieval order according to the current set of queries. Requests for those objects that are slated for retrieval are then put in a queue, called the *fetch queue*. Note how, in this architecture, a node can receive the Boolean expression of a query from step (iii) above before actually receiving requests for retrieving specific objects. This offers an opportunity to prefetch objects not yet requested. A node receiving a query Boolean expression from neighbor nodes will try prefetching data objects for these remote queries, so these objects are ready when requested. Such object requests are put in a *prefetch* queue. The prefetch queue is only processed *in the background*. In other words, it is processed only when the fetch queue is empty. When a queue is processed, an object *Request_Send* function is used to request data objects in the fetch/prefetch queue from the next-hop neighbors.

B. Data Object Requests

As a query is decomposed into a set of data object requests, each corresponding to a specific label to be resolved. These requests are then sent through the network towards their data source nodes. Each node maintains an *Interest Table* that keeps track of *which data objects* have been requested by *which sources* for *what queries*. The interest table helps nodes keep track of upstream requests and avoid passing along unnecessary duplicate data object requests downstream.

Each node also serves as a data cache, storing data objects that pass through, so new requests for a piece of data object that is already cached can be served faster. When a forwarder node already has a cached copy of a piece of data, it needs to decide as to whether or not this cached copy is still *fresh* enough to serve an incoming request for this piece of data. If yes, then the forwarder would just respond to this request by returning the cached object, otherwise it would pass along the request towards the actual source for a fresh copy.

Specifically, a *Request_Recv* is called upon receiving an object request from a neighbor. The request is first bookmarked in the interest table. Then, if the object is not available locally, the request is forwarded (using *Request_Send*) closer to the data source node if the request was a fetch (prefetch requests are not forwarded).

Above, we just discussed how data object requests are handled by Athena nodes. Next, we will look at how Athena handles the transmission of the actual requested data content, either from actual data source nodes or intermediate nodes upon cache hits, back towards the requesters.

C. Data Object Replies

Requested data objects (e.g. a picture, an audio clip, etc) are sent back to corresponding requesters in the similar hop-by-hop fashion as that of the requests themselves. Each data object, as it is being passed through intermediate forwarder nodes, is cached along the way. Cached data objects will decay over time, and eventually expire as they reach their freshness deadlines (age out of their validity intervals). In terms of functional interfaces, each Athena node implements the following two functions: *Data_Send* is used to send requested data object content back towards the original requesters; and *Data_Recv* is invoked upon receiving a piece of requested data object, which is then matched against all entries in the interest table. If the current node is the original query requester node, the data object is presented to the user for the label value, which is in turn used to update the query. Otherwise, the object will be forwarded to the next hop towards the original requester.

One important note here is that in Athena, a *raw data object* needs to be sent from the source back to the requester only when the predicate evaluation (labeling) has to be done by the

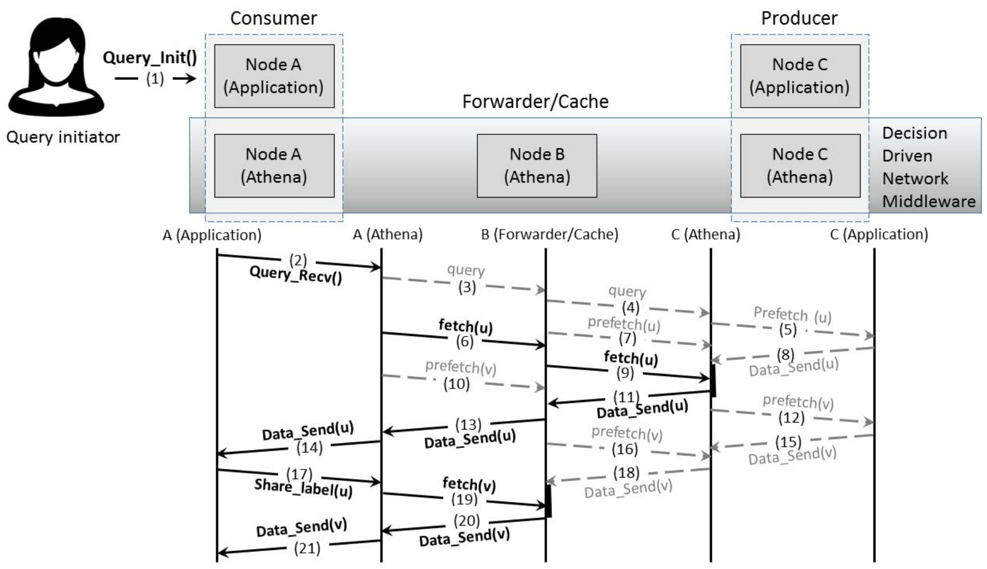

Fig. 1. A visualization showing the flow of requests and data as nodes in Athena work together to resolve a query. In this example, the user uses `Query_Init()` to create and issue a query at Node A. The query in our example involves two data objects, u and v. Node A calls `Query_Recv()` locally to start the processing. The query is propagated through the network (edge 3 and 4), reaching Node B and C. Upon receiving the query, Node C attempts prefetching, first for data object u in this particular example. Since Node C is the data source for u, it sends u back towards the requester node (edge 8, 11, 13, and 14), during which, Node A's fetch request meets the returned data at Node C (edge 9). Upon receiving u at Node A, the user examines the data, makes a judgment regarding the corresponding condition state of the query. This state label provided by the human decision maker is then propagated back into the network (edge 17). The handling for data object v follows a similar pattern, for which the fetch request has a cache hit at the forwarder node B, without reaching the actual source node C, due to prefetch requests. In the figure, grey arrows and requests represent those processed in the *background*; namely pre-fetches and their responses. Solid black arrows and requests are those processed in the foreground; namely actual object fetch requests and their responses.

requesting source. For example, after an earthquake, a user is using Athena to look for a safe route to a nearby medical camp. In doing so, Athena retrieves road-side pictures along possible routes for the user to examine. This judgment call—looking at a picture and recognizing it as a safe or unsafe road segment—is put in the hands of the user (the human decision maker) at the original query requester node. Alternatively, predicate evaluation could be made by machines automatically (e.g., using computer vision techniques to label images). If a qualified evaluator is found at a node for a given predicate, the predicate can be evaluated when the evidence object reaches that node. If the source of the query specified that the signature of this evaluator is acceptable, only the predicate evaluation is propagated the remaining way to the source (as opposed to the evidence object). In the implementation, we restrict predicate evaluators to sources of the query.

D. Label Caching

As requested data objects arrive, the query source can then examine the objects and use their own judgment to assign label values to the objects for the particular query task. These labels are injected back into the network, such that future data requests might potentially be served by the semantic labels rather than actual data objects, which depends on whether the requests need to evaluate the same predicates, and what trust relations exist among the different entities (e.g. Alice might choose not to trust Bob's judgment, and thus would insist on getting the actual data object when a matched label from Bob already exists). As such human labels are propagated from the evaluator nodes back into the network towards the data source nodes, they are cached along the way, and can be checked against the interest tables and, upon matches, used locally to update query expressions, and forwarded to the data requesters. Compared to sending actual data objects, sharing and utilizing these labels can lead to several orders of magnitude resource savings for the particular requests.

To help better visualize how the various discussed components work together, we show, in Fig. 1, an example of requests and data flows for a particular query.

VII. Evaluation

We emulate a network of Athena nodes, each running the actual implementation code discussed above in a separate process per emulated node. Each node is uniquely identified by its IP:PORT pair. We adopted a set of simulation-based experiments featuring a post-disaster route assessment scenario, where Athena is deployed in a disaster-hit region and is used by people in the region to carry out situation assessment and route-finding tasks. For simplicity, we consider a Manhattan-like map, where road segments have a grid-like layout. The EMANE-Shim network emulator [11], [12] was used to handle all data object transmissions.

We experimented with multiple data retrieval protocols as follows:

- *Comprehensive retrieval (cmp)*: As a first baseline, we include a simple algorithm where all relevant data objects for each query are considered for retrieval.

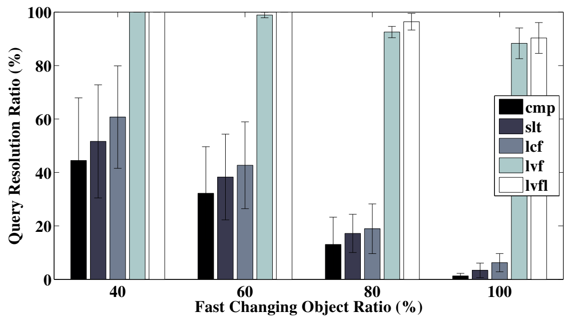

Fig. 2. Query resolution ratio at varying levels of environment dynamics (ratio of fast changing objects).

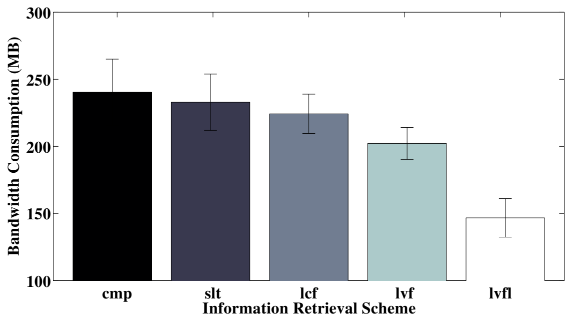

Fig. 3. Total network bandwidth consumption comparison of all schedule schemes (with 40% fast changing objects).

- *Selected sources (slt)*: This is one step beyond the above cmp baseline, where data source selection is performed to minimize the candidate set of data objects to be retrieved to cover all predicates in queries (e.g., if two cameras are overlapping on an road segment that we are interested in, we then can carry out source selection to help determine which one we use, but if two different roads are considered, we retrieve objects that cover both roads). We borrow a state of the art source selection algorithm [10] and use it in our implementation and experiments.
- *Lowest Cost Source First (lcf)*: This scheme takes the above *selected* source nodes, and sorts them according to their data object retrieval costs (i.e., data object size), prioritizing objects with lower costs.
- *Variational Longest Validity First (lvf)*: A decision-driven scheduling algorithm, where resolved labels are *not* propagated into the network for future reuse.
- *Variational Longest Validity First with Label Sharing (lvfl)*: Our scheduling algorithm, *with* label sharing enabled. After a label value is computed, it is propagated back into the network towards the corresponding data source node. Thus, any node along the path that intercepts a future request for this data object can potentially return this label value rather than (requesting and) returning the actual data object.

We divide the experimental region into a Manhattan grid given by an 8×8 road segment network, with around 30 Athena nodes deployed on these segments, where each node's data can be used to examine the node's immediate surrounding segments. Data objects range from 100 KByte to around 1 MByte, roughly corresponding to what we might expect from pictures taken by roadside cameras. The network simulator is configured with 1 Mbps node-to-node connections. Each route-finding query consists of five candidate routes that are computed and randomly selected from the underlying road segment network. Each node issues three concurrent queries. Each data point is produced by repeating the particular randomized experiment 10 times.

In our experiments, data objects belong to two different categories, namely slow changing and fast changing. The ratio of fast changing objects to the total number of objects is a quantification of the level of environmental dynamics. First, we explore how different mixtures of slow and fast changing objects affect the ability of the system to (successfully) resolve decision queries under each of the information retrieval schemes. A decision query is said to be successfully resolved if the system manages to supply enough fresh data in response to a query, such that a decision can be reached by the query deadline, according to the decision logic. The results are shown in Fig. 2. As seen, at all levels of environment dynamics, decision-driven information retrieval schemes are able to successfully resolve most, if not all, queries, whereas baseline methods struggle. This is due to their failure to take into account the data validity constraints when scheduling data retrieval, which then leads to data expirations and refetches. This not only increases bandwidth consumption, but also prolongs the query resolution process, potentially causing more data to expire.

The actual network bandwidth consumption comparisons of all schemes are shown in Fig. 3. We already saw from Fig. 2 that the various baseline schemes fall way short in terms of query resolution ratio. Here we observe that they additionally consume more network bandwidth. Comprehensive retrieval incurs the highest amount of network traffic, as it is neither careful about avoiding redundant data object retrieval, nor does it try to optimize the order when fetching data. Network bandwidth consumption marginally decreases as we include source selection (slt) and then follow a lowest-cost-first (lcf) data retrieval schedule. Our decision-driven scheduling strategy leads to a considerable decrease in network bandwidth consumption. Moreover, when opportunistic label sharing (lvfl) is enabled in Athena, more significant bandwidth savings are observed, as expected, since labels are transmitted instead of actual data objects, when possible.

This very preliminary evaluation serves as an initial proof of concept that decision-driven execution improves application ability to make timely decisions while at the same time reducing cost. More evaluation is needed in more realistic settings to offer more confidence in the proposed approach.

VIII. Discussion

The bulk of this paper focused on challenges in decision-driven execution that are more directly aligned with distributed computing. However, the paradigm offers interesting research opportunities in other related areas as well. For example, the paradigm offer a mechanism for networks to *learn* about their users and the physical environment, then exploit such learned knowledge for optimizing decision-making. The decision model used by the network can itself be refined over time as the system observes the decision-makers' information requests, records their decisions, and takes note of the underlying context as measured by the multitude of sensors connected to the system. Subsequent mining of such data can lead to progressive refinement of decision-making models and to increasingly accurate reverse-engineering of decision stratgeies of individuals and groups. Such learned knowledge can, in turn, be applied to optimize cost of future decision-making. The system, being connected to sensors, can also derive its own models of physical phenomena over time using any of an array of well-known estimation-theoretic approaches. These models can inform settings of various elements of object metadata, such as validity intervals of different types of measurements and probability distributions of particular observed quantities.

While much of the discussion in this paper focused on using the structure of a single decision query to anticipate future object fetch requests, it is also possible to apply pattern mining techniques to identify common decision query *sequences* and thus anticipate not only current but also future decision needs. This is possible because users, in many cases, adhere to prescribed workflows dictated by their training, standard operating procedures, or doctrine. The workflow is a flowchart of decision points, each conditioned on certain variables or inputs. Since the structue of the flow chart is known, so are the possible sequences of decision points. One can therefore anticipate future decisions given current decision queries. Anticipating future information needs can break traditional delay-throughput constraints: anticipating what information is needed next, as suggested by mission workflow, gives the system more time to acquire it before it is actually used.

Finally, observe that decisions can be conditioned not only on current state but also on *anticipated* state. For example, a decision on where to intercept a fleeing criminal will depend on predictions of where the criminal goes next. This information may be inferred indirectly from current measurements. Hence, decision-driven execution lends itself nicely to increasing the efficacy of missions involving a significant *anticipatory* or prediction component, as it offers the mechanisms needed to furnish evidence supporting the different hypotheses or predictions of future actions of agents in the physical environment. The system can therefore empower applications involving intelligent adversaries, such as military operations or national security applications. Design of such applications on top of decision-driven execution systems remains an open research challenge.

IX. Conclusions

In this paper, we outlined a novel paradigm for distributed execution, were all resource consumption is driven by information needs of decision making. The hallmark of the paradigm lies in exporting the logical inference structure of decision making to the underlying resource management layer in order to enable more efficient acquisition of data that simultaneously increases decision timeliness, lowers decision cost, while improving decision quality. Preliminary evaluation suggests that the approach holds promise in meeting its timeliness, cost, and accuracy goals.

Acknowledgements

Research reported in this paper was sponsored in part by the Army Research Laboratory under Cooperative Agreement W911NF-09-2-0053 and in part by NSF under grants CNS 16-18627, CNS 13-45266 and CNS 13-20209. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, NSF, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] J.-E. Kim, T. Abdelzaher, L. Sha, A. Bar-Noy, and R. Hobbs, "Sporadic Decision-centric Data Scheduling with Normally-off Sensors," in *Proc. of IEEE Int'l Real-Time Systems Symposium (RTSS)*, Dec. 2016.

[2] J.-E. Kim, T. Abdelzaher, L. Sha, A. Bar-Noy, R. Hobbs, and W. Dron, "On Maximizing Quality of Information for the Internet of Things: A Real-time Scheduling Perspective (Invited)," in *Proc. of IEEE Int'l Conference on Embedded and Real-Time Computing Systems and Applications*, Aug. 2016.

[3] S. Hu, S. Yao, H. Jin, Y. Zhao, Y. Hu, X. Liu, N. Naghibolhosseini, S. Li, A. Kapoor, W. Dron, L. Su, A. Bar-Noy, P. Szekely, R. Govindan, R. Hobbs, and T. F. Abdelzaher, "Data acquisition for real-time decision-making under freshness constraints." in *RTSS*, 2015.

[4] S. Hu, S. Li, S. Yao, L. Su, R. Govindan, R. Hobbs, and T. Abdelzaher, "On exploiting logical dependencies for minimizing additive cost metrics in resource-limited crowdsensing," in *IEEE International Conference on Distributed Computing in Sensor Networks (DCoSS)*, 2015.

[5] S. Wang, T. Abdelzaher, S. Gajendran, A. Herga, S. Kulkarni, S. Li, H. Liu, C. Suresh, A. Sreenath, H. Wang, W. Dron, A. Leung, R. Govindan, and J. Hancock, "The information funnel: Exploiting named data for information-maximizing data collection," in *Proceedings of the 2014 IEEE International Conference on Distributed Computing in Sensor Systems*, ser. DCOSS '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 92–100. [Online]. Available: http://dx.doi.org/10.1109/DCOSS.2014.32

[6] J. Lee, A. Kapoor, M. T. A. Amin, Z. Wang, Z. Zhang, R. Goyal, and T. Abdelzaher, "Infomax: An information maximizing transport layer protocol for named data networks," in *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, Aug 2015, pp. 1–10.

[7] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. T. D. K, S. B. Zhang, G. T. K. C. Dmitri, K. D. Massey, C. Papadopoulos, T. A. Lan, W. P. Crowley, L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, K. Claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh, "Named data networking (ndn) project ndn-0001," 2010.

[8] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan, "Medusa: A programming framework for crowd-sensing applications," in *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012.

[9] Y. Jiang, X. Xu, P. Terlecky, T. Abdelzaher, A. Bar-Noy, and R. Govindan, "Mediascope: Selective on-demand media retrieval from mobile devices," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2013.

[10] A. Bar-Noy, M. P. Johnson, N. Naghibolhosseini, D. Rawitz, and S. Shamoun, "The price of incorrectly aggregating coverage values in sensor selection," in *IEEE International Conference on Distributed Computing in Sensor Networks (DCoSS)*, 2015.

[11] US Naval Research Lab Networks and Communication Systems Branch, "Extendable mobile ad-hoc network emulator (emane)," http://www.nrl.navy.mil/itd/ncs/products/emane, 2016.

[12] W. Dron, A. Leung, J. Hancock, M. Aguirre, Thapa, and R. Walsh, "Core shim design document," in *NS-CTA Technical Report*, 2014.