


Article

Athena: Towards Decision-Centric Anticipatory Sensor Information Delivery [†]

Jongdeog Lee ¹, Kelvin Marcus ², Tarek Abdelzaher ^{1,*}, Md Tanvir A. Amin ¹ , Amotz Bar-Noy ³, William Dron ⁴, Ramesh Govindan ⁵, Reginald Hobbs ², Shaohan Hu ¹, Jung-Eun Kim ¹, Lui Sha ¹, Shuochao Yao ¹ and Yiran Zhao ¹

¹ University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA; jlee700@illinois.edu (J.L.); tanviralamin@gmail.com (M.T.A.A.); shu17@illinois.edu (S.H.); jekim314@illinois.edu (J.-E.K.); lrs@illinois.edu (L.S.); yao9@illinois.edu (S.Y.); zhao97@illinois.edu (Y.Z.)

² U.S. Army Research Laboratory, Adelphi, MD 20783, USA; kelvin.m.marcus.civ@mail.mil (K.M.); reginald.l.hobbs2.civ@mail.mil (R.H.)

³ The City University of New York, New York, NY 10016, USA; amotz@sci.brooklyn.cuny.edu

⁴ Raytheon BBN Technologies, Cambridge, MA 02138, USA; wdron@bbn.com

⁵ University of Southern California, Los Angeles, CA 90089, USA; ramesh@usc.edu

* Correspondence: zaher@illinois.edu

† This paper is an extended version of Abdelzaher, T.; Amin, T.A.; Bar-Noy, A.; Dron, W.; Govindan, R.; Hobbs, R.; Hu, S.; Kim, J.-E.; Yao, S.; Zhao, Y. Decision-driven Execution: A Distributed Resource Management Paradigm for the Age of IoT. In Proceedings 37th IEEE International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; IEEE: Piscataway, NJ, USA, 2017.

Received: 4 September 2017; Accepted: 8 January 2018; Published: 15 January 2018

Abstract: The paper introduces a new direction in quality-of-service-aware networked sensing that designs communication protocols and scheduling policies for data delivery that are optimized specifically for decision needs. The work complements present decision monitoring and support tools and falls in the larger framework of decision-driven resource management. A hallmark of the new protocols is that they are aware of the inference structure used to arrive at decisions (from logical predicates), as well as the data (and data quality) that need to be furnished to successfully evaluate the unknowns on which these decisions are based. Such protocols can therefore anticipate and deliver precisely the right data, at the right level of quality, from the right sources, at the right time, to enable valid and timely decisions at minimum cost to the underlying network. This paper presents the decision model used and the protocol design philosophy, reviews the key recent results and describes a novel system, called Athena, that is the first to embody the aforementioned data delivery paradigm. Evaluation results are presented that compare the performance of decision-centric anticipatory information delivery to several baselines, demonstrating its various advantages in terms of decision timeliness, validity and network resources used. The paper concludes with a discussion of remaining future challenges in this emerging area.

Keywords: decision-centric Quality of Service (QoS); information delivery for the Internet of Things (IoT); real-time scheduling

1. Introduction

The concepts of decision-driven resource management, decision-centric information monitoring and decision support systems have been studied extensively in prior literature. As discussed later in the Related Work section, the system presented in this paper complements work on existing decision-centric monitoring and support tools and augments the general category of

decision-driven resource management frameworks. It is distinguished by adapting decision-driven resource management to the realm of runtime network data management in the context of distributed sensing and Internet of Things (IoT) applications. This is as opposed to topics such as managing software architecture, human resources, financial assets or physical organizations. The application of decision-driven resource management to runtime networking leads to novel challenges and research questions in the design of network communication protocols and communication scheduling policies, presented in this paper.

This paper describes a novel model and system for anticipatory information delivery in mission-centric applications, where communication protocols and scheduling policies for data delivery are optimized specifically for meeting decision needs. We envision an environment such as an IoT-enabled smart city [1,2], where a cyber-physical system [3,4] is deployed to assist a team in performing a first-response mission in the aftermath of a natural disaster. Mission execution is affected by answers to a set of questions that first-responders might ask; for example: What are the affected areas? What is the extent of damage in each area? Who are the affected individuals? Where are the survivors? Are there injuries? Who is in most need of help? These and similar questions drive the decisions of the mission commander in allocating human and physical resources to different rescue efforts. A computing system familiar with similar missions can anticipate the types of decisions the commander must make, as well as the nature of mission-relevant information that affects these decisions. The system can then collect such information ahead of time and update it as the situation changes. In recent work, this information delivery paradigm was called decision-driven execution [5]. It re-thinks scheduling and communication to consider application models that are more pertinent to mission-driven quality-of-service-aware cyber-physical systems. We review this recent work, describe its embodiment in a system called Athena and present significantly expanded evaluation results that describe its advantages over a set of baselines.

Decision-centric anticipatory information delivery is motivated by visions of an instrumentation-rich world where a large number of sensors and other information sources may supply data relevant to an application [6,7]. There might not exist an a priori structure (such as fixed control loops) that statically dictates where each sensor should send its data. Rather, information sources will generally be connected to shared media, allowing flexible reconfiguration to support different (both one-off and repetitive) tasks. A challenge is to configure data collection from appropriate sources in order to meet specific application information needs while minimizing cost. 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 (decision) task that needs data from that sensor. In other words, resource consumption is decision-driven.

At a higher level, one can view decision-centric anticipatory information delivery as a way for data fusion systems to exert initiative in anticipating and offering mission-relevant information to mission commanders. For the system to exert initiative, it must know the nature of the mission (i.e., mission-relevant decisions that need to be made) and the unknowns that need to be determined in making mission-relevant decisions. The paradigm offers an exciting foundation for rethinking resource management as a process of arbitrating the acquisition and movement of data among processing components to meet decision needs. Such arbitration is guided by a novel set of questions: 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 the 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 paradigm must optimize communication, storage and scheduling to meet these constraints. In this paper, we describe the implementation of one such system, called Athena, and evaluate its performance.

Decision-centric anticipatory sensor information delivery is an interesting cyber-physical problem. The data of interest typically come from sensors and as such captures aspect of the physical state of the world. Since the 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, 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 the 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).

An abbreviated version of this work was presented at ICDCS 2017 [5]. This paper reviews the earlier work, places the effort in the broader category of decision-driven resource management frameworks, offers additional detail on the algorithms and implementation and significantly expands the evaluation results. Section 2 presents related work, placing decision-centric anticipatory information delivery in the broader context of decision monitoring and support tools and decision-driven resource management. Section 3 describes the new decision-driven system architecture for managing runtime data communication to optimize sensor information delivery. Section 4 overviews the resulting communication and scheduling challenges. Section 5 elaborates challenges in real-time scheduling that arise in the decision-driven context. Section 6 details networking challenges. The Athena implementation and evaluation are described in Sections 7 and 8, respectively. A brief discussion of other remaining research questions is presented in Section 9. The paper concludes with Section 10.

2. Related Work

Our work is broadly related to the concepts of decision-driven resource management, decision-centric information monitoring and decision support systems that have been studied extensively in prior literature in multiple domains. A significant amount work on decision-centric management revolves around managing software architecture decisions [8–10], product design [11], business processes [12], business intelligence [13] and physical organizations [14] (including human resources and financial assets). In contrast, we are the first to focus on decision-driven management of runtime communication and scheduling in networks that deliver sensory information. Below, we elaborate on this distinction, comparing our work more clearly to the general directions in current literature, including decision-centric management, decision-support tools and decision-driven information monitoring tools.

Much of today's decision-centric management frameworks are studied in disciplines such as business management [12,13] and organizational design [14]. They refer to human-centric processes that empower decision-makers, as opposed to runtime algorithms such as communication protocols and scheduling policies in computing machines. A category that comes closer to computer science in this context is the category of managing software architecture design [8–10]. Architectural decisions affect software performance outcomes. Hence, much work is focused on understanding the dependencies between decisions and outcomes, offering tools to enable decisions that lead to improved outcomes, such as improved performance [10], improved security threat mitigation [15,16] and improved defense [17]. For example, decision-driven architecture reviews [10] offer opportunities to assess the suitability of architectural decisions to design purposes early on the development cycle. Tools for decision-driven business performance management [18] document the relation between performance metrics on the one hand and decisions/sub-decisions on the other, allowing one to

explore the decision space by modifying decisions and tracking their impact on performance metrics. These approaches are intended for use to inform design early on in the software development cycle (i.e., before deployment). In contrast, we are interested in communication protocols and scheduling policies that offer runtime support.

A somewhat closer tool category for our work is the category of decision support systems [19]. These systems use models of the world to allow decision-makers to play “what-if” scenarios and explore the consequences of their decisions on the modeled domain of interest. Hence, for example, given an adequate model of the political, military, economic, social and topological terrain features in some foreign state, a decision-maker might estimate the potential consequences of a certain military operation. Various planning tools have also been proposed that allow decision-makers to compose detailed plans of action and contingencies based on domain models [20]. Work reported in this paper does not constitute a decision-support tool in the above sense. The software described in this work does not have models of the world and is unable to compose plans. Rather, it helps users to retrieve information objects in a manner that facilitates meeting decision needs.

The concept of decision-centric information monitoring comes closest to our work. A decision-centric information monitoring system [21] answers the following key question: which variables must one monitor in order to make an informed decision? The answer often comes from modeling the decision-making process [13,22]. An interesting trade-off is involved. Monitoring too many variables may overwhelm the decision-maker, whereas monitoring too few may impair the quality of decisions. This work is synergistic with ours in that we assume that the problem of determining the relevant variables (or unknowns) has already been solved. Cast in the context of a cyber-physical systems in which unknowns that determine the viability of each potential course of action have been identified, our work addresses the complementary problem of contacting the sources (e.g., sensors) that would deliver information on these unknowns over a network while minimizing delivery cost. This network communication and scheduling policy challenge is orthogonal to the manner in which relevance of specific unknowns was determined in the first place.

Recent work has made initial progress at solving the above cyber-physical network resource management problem [5,23–25]. The work bears resemblance to prior database research that considered explicit data access transactions and required a degree of data freshness [26–29]. It is new in tying such retrieval policies explicitly to the decision needs, leading to new research challenges and solutions as described below.

3. A Decision-Driven System Architecture

A data fusion system that supports decision-centric anticipatory information delivery may simultaneously retrieve data for multiple missions. An example mission might be to track a specified target (such as a robbery getaway vehicle or an escaped fugitive) in a city using deployed sensors and security cameras or to rescue a group of individuals trapped in the aftermath of a natural disaster. Execution of the mission starts with collecting information relevant to the mission. We assume that such collection occurs over a tactical network of limited resources, which is often the case in such missions as disaster response (where much infrastructure has been destroyed, leading to resource shortages). The primary purpose of the data collection system is to conserve the resources expended on data collection while at the same time collecting enough data to support the mission commander or decision-maker. In the context of a mission, the commander or decision-maker needs information to make decisions. In a rescue mission, an example decision might be: Which is evacuation path to follow in order to rescue the disaster survivors? In a tracking mission, a decision might be: Which sensors are to be turned on in order to track the target? An interesting trade-off is involved. Namely, collecting more information may overwhelm the underlying resource-constrained network, leading to delays that may negatively impact the ability of the user to obtain information on time and hence their ability to make timely decisions. In contrast, collecting inadequate information might impede one’s ability to make a correct decision. For example, if one does not collect enough information on the health

conditions of various roads in the aftermath of a large natural disaster, one might attempt to evacuate the survivors over a road that ends up being blocked or otherwise unsuitable. The key is therefore to collect just the right amount of information to make a decision. With that in mind, several levels of indirection allow the system to optimize information delivery for the respective missions, as follows:

- The decision working set: For each mission, the system maintains the set of most common decisions to be made, called the decision working set. For example, in a rescue mission, decisions may need to be made on the best evacuation route for each survivor. The set of all such mission-relevant decisions constitutes the working set.
- The decision model: In deciding on a course of action, the commander or decision-maker must consider several relevant variables that impact the decision outcome. For a trivial every-day example, in deciding what to wear in the morning, one might consider weather conditions (such as temperature and precipitation). These variables constitute the unknowns that need to be determined for a decision to be made. Hence, for each decision in the working set mentioned above, the system must know the relevant unknowns, as well as how these unknowns impact decision outcome. We call it the decision model. The choice of decision model itself gives rise to interesting research questions that warrant further attention. In the simplest model, decisions are viewed as choices of a course of action among multiple alternatives [5,25]. More on the decision model will be mentioned later in the paper. The viability of each individual alternative depends on the satisfaction of several predicates. Making a choice can therefore 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 wear a T-shirt; else, I will wear a sweater”.
- The unknowns: Consider a predicate such as “if it is (i) sunny and (ii) warm”. Evaluating such a predicate requires determination of the value of one or more unknowns. In the example quoted above, the unknowns are parameters of local weather that determine whether or not it is sunny and whether or not it is warm. The decision model specifies the unknowns whose value needs to be determined in order to decide on the viability of each course of action.
- The evidence data objects (or simply, data objects): Determination of the unknowns entails the acquisition of corresponding evidence. Data objects such as images, videos or sound clips, generated by appropriate sensors, can supply the needed evidence. For example, a picture taken several minutes ago at the location of interest, showing that it is sunny, would constitute evidence that “sunny” can be evaluated as “true”.
- The sources: 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. The system must choose a source such that timely and relevant information is provided at low cost. This gives rise to appropriate source selection and data delivery scheduling protocols.

Figure 1 is a conceptual view of the architecture, where missions are broken into decision working sets, decisions are associated with logical expressions in appropriate unknowns and the unknowns are linked to sources that can supply the relevant data objects (or evidence).

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.

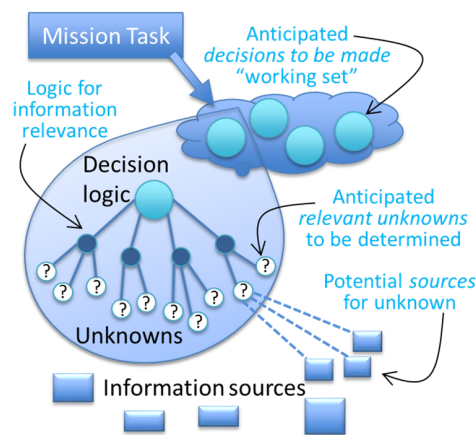


Figure 1. A conceptual architecture for anticipatory information delivery.

Athena is a recently developed system that implements the above architecture. An illustration of Athena system components is shown in Figure 2. For completeness, we list all components of the overall system below. Later in the paper, we shall focus only on data communication and scheduling policies and the components that impact them. Figure 2 presents the overall architecture. The following components are depicted:

- **User I/O:** This is the user interface component that allows entry of decision queries and missions. An instance may run on every Athena-enabled node in the distributed system. The interface passes user queries to the rest of the system. It may also originate anticipated queries based on the mission’s decision working set.
- **Application semantic translator:** This component determines the unknowns needed for a particular decision. These unknowns are mission specific. A mission-specific library implements the application semantic translator. In the current system, libraries have been implemented for tracking missions and rescue missions, as a proof of concept.
- **Semantic store:** This component maps individual unknowns to sources who may have evidence objects that determine the value of that unknown. The semantic store may be replicated. Sources would send metadata about objects they have to the semantic stores that allow the latter to match unknowns and sources.
- **Source selector:** It is often the case that multiple sources have redundant information and hence can help resolve the same unknown. It would be wasteful to collect data from all these redundant sources. Instead, the source selector determines which subset of data sources to contact.
- **Logical query resolution engine:** This engine determines the order in which evidence objects are to be acquired from their sources to ensure constraints such as data freshness and decision deadlines.
- **Information collection and dissemination engine:** This engine offers the mechanisms for data collection, including mechanisms for prefetching potentially relevant data at a lower priority. This component will be described later in the implementation section in more detail.

A user’s query (actual or anticipated) is accepted and translated into the corresponding logical expression (predicates over relevant unknowns) by the Application Semantic Translator, which then uses a nearby Semantic Store to identify the set of nodes that may have evidence objects to determine the unknown. Taking this information, the Logical Query Resolution Engine then uses the Source Selector to choose some subset of these sources to contact. This subset aims to minimize redundancy of the contacted node set, as well as delivery cost. Data requests are then scheduled in an order that takes into account decision deadlines and data freshness constraints. These requests are handed over to the Information Collection & Dissemination Engine, which accesses the underlying sensing and communication stacks to handle all incoming and outgoing requests, data objects and resolved

predicate label values, with proper book-keeping and information updates to its various internal components, as discussed in detail later.

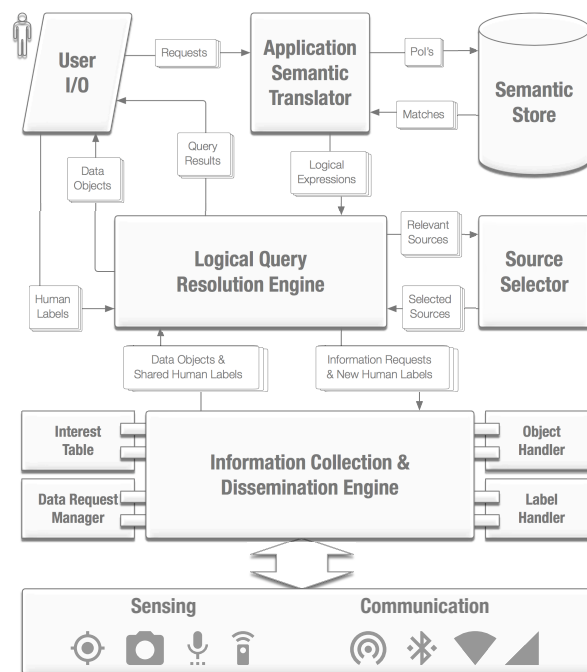


Figure 2. Athena’s architectural design, as well as how information propagates through different system components.

3.1. 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 prioritize 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 embarking 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 satisfies 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. Consider a scenario from emergency response [30]. 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 is not 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., probabilities that some measurements not yet acquired will fall into a range that invalidates versus supports a predicate). On 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 11 p.m. on a Monday night might be known. However, a condition, such as a nearby large concert that ends around the same time, can invalidate it. In general, a combination of past contextual knowledge, current observations, and invalidation 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.

3.2. 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.

3.3. 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, *viabileX*. 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.

4. Decision-Driven Resource Management: Optimizing Retrieval Cost

Initial work on decision-driven resource management was recently published in the context of centralized systems [25,31]. 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 *i*-th decision, and $\{b_{ij}\}$ denote the *j*-th Boolean condition needed to determine the value of a_i . Therefore, a query *q* takes the general form:

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

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.

4.1. Minimizing Retrieval Cost by Short-Circuiting

Associated with each condition b_{ij} may be several pieces of metadata. Examples include (i) retrieval cost C_{ij} (e.g., data bandwidth consumed), (ii) estimated retrieval latency l_{ij} , (iii) success probability p_{ij} (i.e., probability of evaluating to true), and (iv) data validity interval d_{ij} (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,

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

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:

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

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 [25], 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. The complete algorithm pseudo-code is shown in Algorithm 1.

Algorithm 1 Retrieval schedule for dynamic query resolution.

Input: A query's deadline requirement d_q , its candidate courses of action $\{a_i\}$, and acceptable parallel retrieval level r . For each constituting condition t_{ij} for a particular a_i , its corresponding

- evaluation (retrieval) costs c_{ij} ,
- retrieval latencies l_{ij} ,
- success probabilities p_{ij} , and
- freshness (validity) interval v_{ij} .

Finally, the subset of conditions that have been cached O_q , in descending order of $\frac{1-p_{ij}}{c_{ij}}$.

Output: Query resolution result

```

1: failure_counter ← 0
2:  $L_a \leftarrow \{a_i\}$  sorted in descending order of  $\frac{p_i}{c_i}$ 
3: for  $a_i$  in  $L_a$  do
4:    $Q_c \leftarrow \emptyset, Q_d \leftarrow$  longest valid. interval first order,  $S_p \leftarrow \emptyset$ 
5:    $L \leftarrow \{t_{ij}\}$  sorted in descending order of  $\frac{1-p_{ij}}{c_{ij}}$ 
6:   while  $Q_d \neq \emptyset$  do
7:     for  $t_i$  in  $L$  do
8:       if moving  $t_i$  from  $Q_d$  to the end of  $Q_c$  does not increase freshness violation degree then
9:          $Q_d \leftarrow Q_d \setminus \langle t_i \rangle, Q_c \leftarrow Q_c \cup \langle t_i \rangle$ 
10:        break
11:       end if
12:     end for
13:   end while
14:   while  $|Q_c| > 0$  do
15:      $t_e \leftarrow$  end element of  $Q_c$ 
16:      $Q_c \leftarrow Q_c \setminus \langle t_e \rangle, S_p \leftarrow S_p \cup \{t_e\}$ 
17:     if  $Q_c + S_p$  satisfies validity intervals then
18:       for  $t^o$  in  $O_q$  do
19:          $d_{t^o} \leftarrow t^o$ 's absolute validity deadline
20:         if  $Q_c + S_p \setminus \langle t^o \rangle$  satisfies  $\min(d_q, d_{t^o})$  then
21:            $Q_c \leftarrow Q_c \setminus \langle t^o \rangle, S_p \leftarrow S_p \setminus \langle t^o \rangle$ 
22:         else if a shortest tail,  $T_{Q_c}$  of  $Q_c$ 's can be moved to  $S_p$  to satisfy validity intervals AND  $|S_p| \leq r$ 
23:            $Q_c \leftarrow Q_c \setminus T_{Q_c}, S_p \leftarrow S_p \cup T_{Q_c}$ 
24:           update  $d_q$ 
25:         end if
26:       end for
27:       Process  $a_i$  with retrieval schedule  $Q_c + S_p$ 
28:       if  $a_i$  succeeds then
29:         return  $a_i$  as an successful result
30:       else
31:         increment failure_counter by 1
32:       break
33:     end if
34:   end while
35: end for
36: end for
37: if failure_counter ==  $|L_a|$  then
38:   return request resolves to failure
39: else
40:   signal validity interval cannot be satisfied
41: end if

```

4.2. 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 predicates 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"]
}
```

5. 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.

As mentioned earlier, two types of timing constraints must be enforced by the retrieval schedule. First, decision deadlines must be obeyed (i.e., deadline constraints). Second, data furnished for a decision must be fresh (i.e., data validity constraints). Each retrieved data object has a validity interval

within which it remains fresh. A decision is valid only if it is made based on objects that remain within their validity intervals.

5.1. Initial Results

Simple versions of the above problem have been solved in recent work [23,24]. 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 [23], let there be N objects, named O_1, \dots, O_N , that have information relevant to the decision. These objects are retrieved from sources identified by the source selection algorithm. Let us denote these sensors by S_1, \dots, S_N , respectively. Let us further consider that these sources monitor their environment on demand. Hence, when source, S_i , is contacted, it observes its environment and constructs an appropriate observation object, O_i (e.g., takes a picture), then sends it in response to the received request. Let us assume that O_i has a validity period I_i and size C_i . Let source S_i be contacted at time t_i .

The system collects the relevant data from all selected sources then a decision is made. Let time F denote the time when all data needed for the decision has been fetched, such that a decision can be made. We require that $F \leq D$, where D is a deadline. If all N objects are retrieved, the decision cost is given by:

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

An interesting question is: does there exist a feasible object retrieval schedule? In other words, is it possible to fetch all N objects such that the decision deadline is met (to make sure the decision is timely) and such that each object remains fresh by the time fetching is complete? The latter ensures that the decision is valid (because it was made based on unexpired data). This occurs when no sensor is sampled twice. Let a feasible retrieval schedule be one that satisfies the decision deadline. An optimal retrieval scheduling policy is one that finds a feasible retrieval order whenever one exists, thereby simultaneously satisfying both freshness and deadline constraints:

$$\textbf{Data freshness: } t_i + I_i \geq F \quad (\forall i, 1 \leq i \leq N),$$

$$\textbf{Decision deadline: } t + D \geq F,$$

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:

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

Prior work [23] shows that the Least Volatile object First (LVF) object retrieval policy is optimal in the case of a single decision query where data are retrieved over a single bottleneck resource. The policy fetches the object with the longest validity interval first. The optimal policy for multiple independent decision queries is more involved, but has been derived for a simple decision model [23].

5.2. 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 guarantee 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.

6. 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.

6.1. 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 [32,33]. In information-centric networks, such as NDN [34], 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 advertised 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 advertise their names. Nearby routers who receive those advertisements 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.

6.2. 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.

6.3. 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. These objects, for example, can be exempt from the aforementioned approximation mechanisms for congestion control. They can also receive 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.

7. Implementation

To perform a proof-of-concept validation, we implemented a distributed system, called Athena, that embodies the decision-driven execution paradigm. At present, Athena is implemented mostly in Python, with some parts in Java, and C++. Athena is hosted within the Dynamically-Allocated Virtual Clustering (DAVC) management environment [35], available from our collaborators at the US Army Research Labs. DAVC offers virtual containers that allow easy integration of physical and virtual nodes in the same environment, and thus straightforward migration from emulation to deployment. DAVC currently runs on an Ubuntu server (a 64-bit machine) in emulation mode, and has dependencies on NTP (for time management) and NFS (for file management). The Athena server implements three main data structures, local/remote query logs, fetch/prefetch queues, and the interest table. They are asynchronously protected and shared among the functional component threads. A node's main event loop simply waits on a TCP socket for incoming messages, and dispatches received messages, according to their headers, to spawn the corresponding functional threads. We simulate a network by running the actual communication protocol stack in a separate process per emulated node. Each emulated network node is thus uniquely identified by its IP:PORT pair. Below, we describe in more detail the functional details of different Athena threads.

7.1. 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 predicate (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 [36,37], (iii) compute the optimal source subset using a source selection algorithm [38], (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.

7.2. 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.

7.3. 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 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.

7.4. 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 Figure 3, an example of requests and data flows for a particular query.

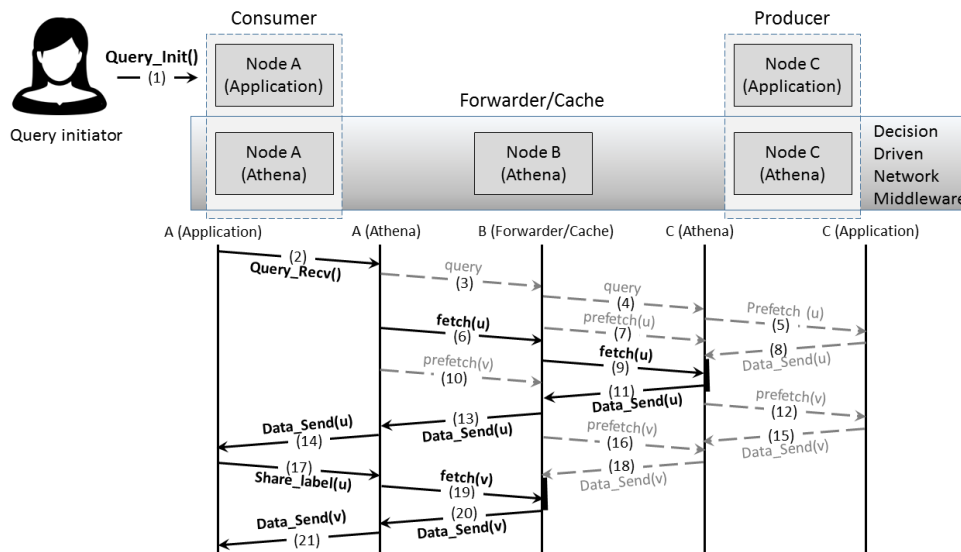


Figure 3. A visualization showing the flow of requests and data as nodes in Athena work together to resolve a query. In this example, the user uses `Query_Init()` to create and issue a query at Node A. The query in our example involves two data objects, *u* and *v*. Node A calls `Query_Recv()` locally to start the processing. The query is propagated through the network (Edges 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 (Edges 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.

8. Evaluation

In our prototype Athena implementation, a single process—a multithreaded Athena server—was instantiated per emulated node. The server’s main event loop simply waited on a TCP socket for incoming messages, and dispatched received messages, according to their headers, spawning the corresponding functional threads as needed. A communication server implemented the desired protocol stack. The application was compiled together with the Athena server. Each emulated network node was uniquely identified by its IP:PORT pair, referring to the server’s machine IP and port address associated with the emulated node.

Since we have not had the opportunity to test our system in an actual post-disaster environment, we adopted a set of simulation-based experiments featuring a post-disaster route-finding 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-reconnaissance tasks. For simplicity, we consider a Manhattan-like map, where road segments have a grid-like layout. The EMANE-Shim network emulator [39,40] was used to handle all data object transmissions among Athena server processes.

More specifically, we divided the experimental region into a Manhattan grid given by an 8×8 road segment network, with around 30 Athenanodes 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 emulator is configured with 1 Mbps node-to-node connections. Each route-finding query consists of 5 candidate routes that are computed and randomly selected from the underlying road segment network. Additionally, each node is issued three concurrent queries. With these generic parameter settings, we next discuss each set of the experiments, where the corresponding particular parameter settings will be specified. For collecting results, each data point is produced by repeating the particular randomized experiment 10 times. As resource constraints is our main optimization goal, we use resource consumption (network bandwidth usage) as the main evaluation metric, unless otherwise specified in particular sets of experiments. Table 1 summarizes the key settings.

Table 1. Key emulation parameter settings (unless individual experiments state otherwise).

Network nodes	30
Data object size	100 KB–1 MB
Link bandwidth	1 Mbps node-to-node
Viable options per decision	5
Queries per node per experiment	3
Number of experiments per graph point	10

For the information retrieval schedule, we experimented with multiple baselines, besides our own algorithm, Algorithm 1, introduced in Section 4. All compared algorithms are listed 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. An object is relevant if it provides evidence regarding at least one of the unknowns specified in the decision logic. Note that, comprehensive retrieval does not try to minimize potential redundancy in retrieved evidence, nor does it optimize retrieval order based on considerations such as freshness constraints, deadlines, and cost.
- Selected sources (slt): This is one step beyond the above cmp baseline, where data source selection is performed to minimize redundancy in retrieved evidence. Specifically a coverage problem is solved to obtain the candidate set of data objects to be retrieved that allow evaluating all predicates in the underlying decision logic. Thus, if two cameras have overlapping views of the same road segment, source selection will typically choose only one to retrieve data from. We borrow a state of the art source selection algorithm [38] 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): Our scheduling algorithm, as discussed in Section 4, except that values of 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. Therefore, after a piece of retrieved evidence is annotated with a label, this label information 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 return this label value rather than (requesting and) returning the actual data object.

With these above five different information retrieval scheduling schemes, we carry experiments to study Athena's behavior along the following different dimensions.

- Environment dynamics: We experiment with different levels of environment dynamics, where different portions of data objects are considered to be of fast/slow-changing nature (i.e., having short/long validity intervals).
- Query issuance pattern: We experiment with how queries are issued to nodes—more specifically, whether all queries are issued to only a few nodes or a large number of them.
- Query complexity: For each query, we experiment with varying number of candidate routes, and different number of road segments per route. These correspond to the number of courses of action that are OR'ed together, and the number of predicates AND'ed to establish viability of each course, respectively.
- Query interest distribution: We experiment with two scenarios, namely whether all inquiries are focused on a small hotspot or spanning the entire global region.
- Query locality: We experiment with how “localized” queries are. Basically, a localized query inquires about a node's immediate surrounding area, whereas a more diverse query may ask about data objects on the far end of the network.
- Network topology: We generally use randomly generated network topologies for our experiments. However, we also experiment with two other specific network topologies, namely linear and star shaped, to see how different patterns of network connectivity might affect system behavior.

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, which is what one might expect when fast wired infrastructure has been destroyed by a disaster, resulting in slower ad hoc links. Each route-finding query consists of five candidate routes that are computed and randomly selected from the underlying road segment network. With these generic parameter settings, we next discuss each set of the experiments, where the corresponding particular parameter settings will be specified. For collecting results, each data point is produced by repeating the particular randomized experiment 10 times.

First and foremost, as our Athena information management system is designed for situation assessment and decision-making under dynamic post-disaster environments, we look at how its query resolution capability is affected by different levels of environment dynamics. In our experiment, data objects generally belong to two different categories, namely slow changing and fast changing. For example, a blockage on a major highway might get cleared within hours, but a damaged bridge likely will take days/weeks or even longer to repair. In this set of experiments, we explore how different mixtures of slow and fast changing objects affect the performance of each of the information retrieval schemes. The results are shown in Figure 4. As seen, at all levels of environment dynamics, our data-validity aware information retrieval schemes are able to successfully resolve most, if not all, queries (i.e., perform them on time and based on fresh data), whereas the baseline methods struggle even with a relatively low level of environment dynamics (an incorrectly resolved query, for our purposes is one where the decision missed the deadline or was based on data that passed their validity interval). This is due to their failure to take into account the deadline and data validity information when scheduling retrievals, which then leads to data expirations and refetches or deadline misses. This not only increases bandwidth consumption, but also prolongs query resolution process, potentially causing more data to expire.

The actual network bandwidth consumption comparisons of all schemes are shown in Figure 5. We already saw from Figure 4 that the various baseline schemes fall short in terms of query resolution ratio; here we observe that they additionally consume more network bandwidth. Comprehensive

retrieval scheduling incurs the highest amount of network traffic, as it neither is careful about avoiding redundant data object retrieval, nor tries to follow a meaningful 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. None of the above schemes take into consideration environment dynamics. Therefore, they tend to result in more information expiration and refetching, leading to extraneous bandwidth usage. This additional usage is effectively minimized/avoided by our scheduling strategy, which leads to a considerable decrease in network bandwidth consumption. Additionally, when opportunistic label sharing (lvfl) is enabled in Athena, more significant bandwidth savings are observed, since labels are transmitted instead of actual data objects when possible.

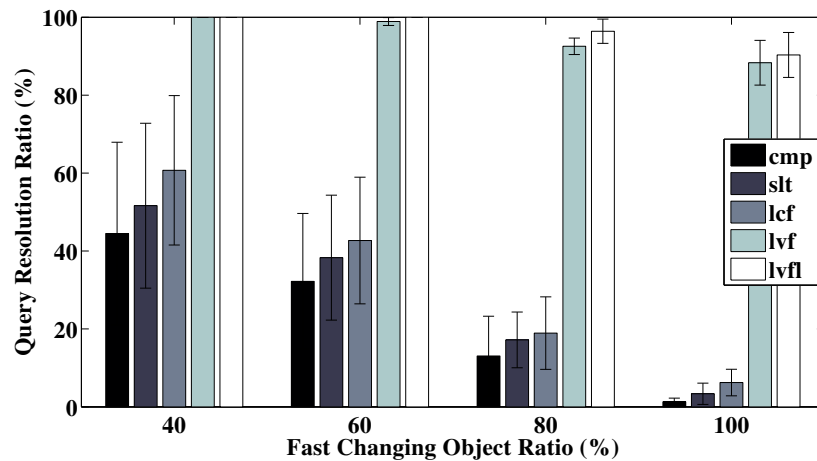


Figure 4. Query resolution ratio at varying levels of environment dynamics (ratio of fast changing objects). cmp, comprehensive retrieval; slt, selected sources; lcf, lowest-cost-first; lvf, longest validity first; lvfl, longest validity first with label sharing.

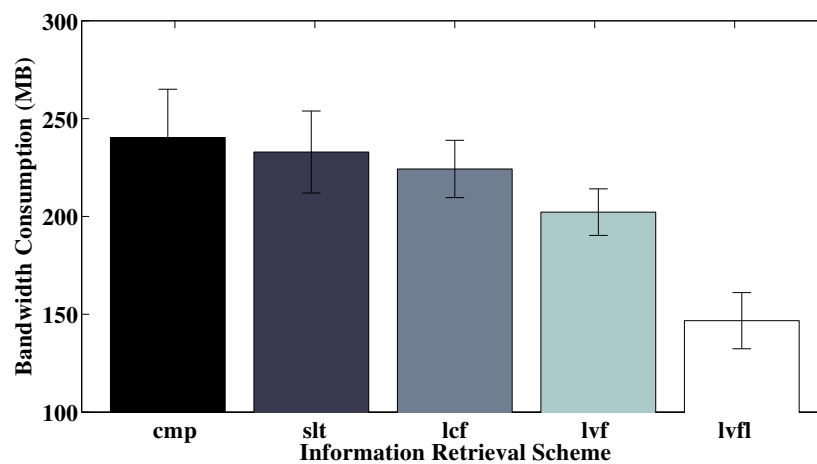


Figure 5. Total network bandwidth consumption comparison of all schedule schemes (with 40% fast changing objects).

Next, we take a look at how query issuance patterns affect system performance. This essentially means how many nodes of the entire network users are issuing queries. If the number is 1 (which is less likely for realistic scenarios), this essentially is equivalent to a centralized application where all users issue all queries at a central node. Under more realistic settings, however, this number will be high as users all around the region would potentially need to request information at each of their own respective locations. We experiment with issuing queries by a single, a pair, half, as well as all nodes. The results are shown in Figure 6. First of all, we observe that the more centralized the query issuance

pattern is, the lower the network bandwidth consumption. This makes sense because, given the same number of random queries, having fewer query nodes means higher chance of cache hits for both data objects and shared labels. We also observe that, as query issuance pattern shifts from centralized towards distributed, our information retrieval schemes lead to better improvements over baseline methods as well as consistently stable performance on query resolution ratio.

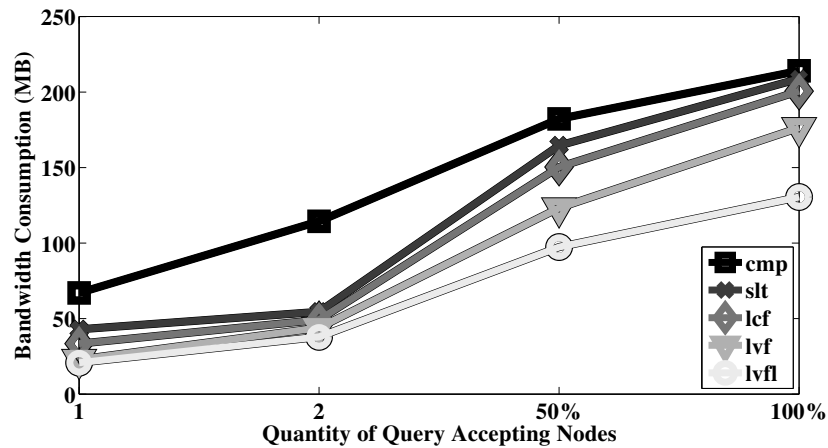


Figure 6. Different query issuance patterns (i.e., varying number of nodes to which a fixed number of queries are issued to).

Next, we look at how query complexity affects system performance. As each query is represented in its logical form, an OR of ANDs, we can vary the number of ANDs under the OR, as well as the number of tests that needs to be performed for each AND in our route finding scenario. This naturally corresponds to the number of candidate routes for each query, and the number of road segments for each candidate route. As shown in Figure 7, increasing the number of routes per query leads to higher network bandwidth consumption for all information retrieval schemes. It is worth noting that, our decision-aware scheduling algorithms (lvf, lvfl) again lead to a slower bandwidth increase compared to other baseline methods, thanks to their ability to exploit queries’ internal structure and prune logical evaluations, which would otherwise lead to unnecessary network traffic.

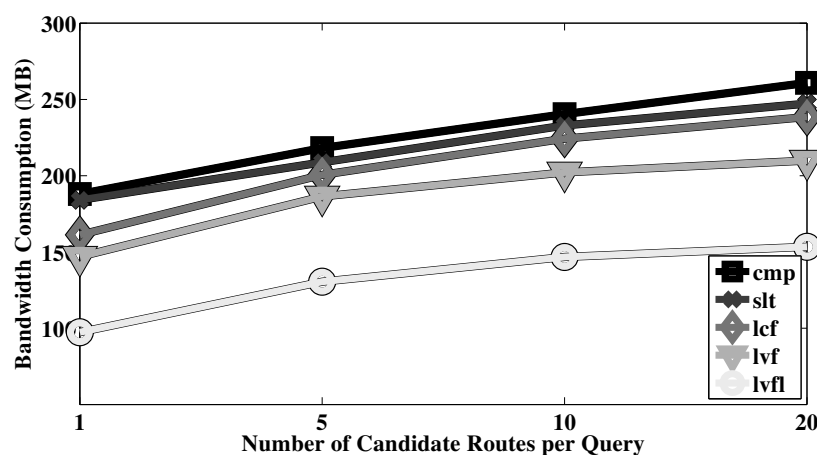


Figure 7. Varying number of candidate routes per query.

Figure 8 shows the experiment results with varying route lengths, where routes are categorized into 5 different length percentiles. As seen, the general relative comparison in terms of network bandwidth consumption remains the same as that of previous experiments. However, we notice an interesting convex shape rather than a monotonic trend (i.e., longer route lengths lead to higher

network bandwidth consumption) that some might have expected. The reason lies in the fact that the higher the percentile, the lower the number of route choices there are—i.e., we can easily find in the road network a large number of different short routes, but there might be few options for extremely long ones. This lack of route choices then leads to a higher number of repeated road segments in the queries, which in turn leads to higher cache hit rates and thus fewer transmissions of data objects from their original source nodes. Therefore, this set of experiments also illustrates how queries’ interest distributions affect system performance; namely, all other conditions being equal, the more concentrated the queries’ interests are, the lower the system bandwidth consumption is. This is due to higher cache hit rates for data objects as well as shared labels values.

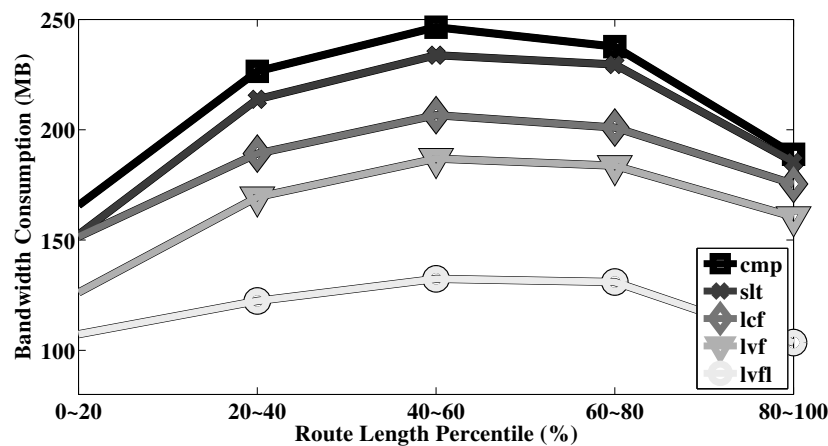


Figure 8. Varying length ranges of queries’ intended routes.

Next, we examine how queries’ locality can affect system performance. Here, locality refers to how close a query’s interests are from the node where it is issued. A nearby query is likely only expressing interests in the vicinity of its issuance node, whereas a faraway query might inquire about data objects located at the far end of the network. Experimental results on network bandwidth consumption are shown in Figure 9. We again omit reiterating the comparison between the different information retrieval scheduling schemes as it is similar to that of previous experiments. We do want to point out the non-monotonicity of bandwidth comparison as we move from nearby to random and then to faraway queries. First of all, when each query is only interested in its close vicinity, the query itself is often of low complexity (i.e., containing fewer candidate routes, and fewer road segments per route), and few data objects need to travel long paths to reach their requesters. Thus, the overall network bandwidth usage remains low. None of these mentioned characteristics still hold when queries’ interests shift from nearby to randomly covering the entire region, causing much higher traffic in the network. Then finally, when we further shift queries’ interests to be only focusing on faraway objects, we have actually limited the object interest candidate pool, causing queries to overlap more on their relevant data object set. This too leads to higher cache hit rate for both data objects and human labels, similar to what we observed in Figure 8.

Finally, we take a brief look at how different network topologies affect system performance. In addition to randomly generated topologies, we also experiment with linear and star shape networks as two vastly different topologies in terms of network diameter. Results are shown in Figure 10. As seen, networks where nodes are linearly connected result in significantly higher bandwidth consumption. This is understandable because of the excessively long paths data packets need to travel to reach their destinations. The higher probability of data validity expiration caused by long retrieval latencies overshadows the benefits of cache hits along the long paths. The star shape networks, on the other hand, have short transmission paths, but the network bandwidth consumption is not much lower than that of linear networks, due to the existence of a bottleneck at their star centers, which lead to network

congestion. This, in turn, causes more data to expire and hence excessive refetches. It is promising to see that in general random network topologies, Athena, with our retrieval schedule schemes, gives good performance.

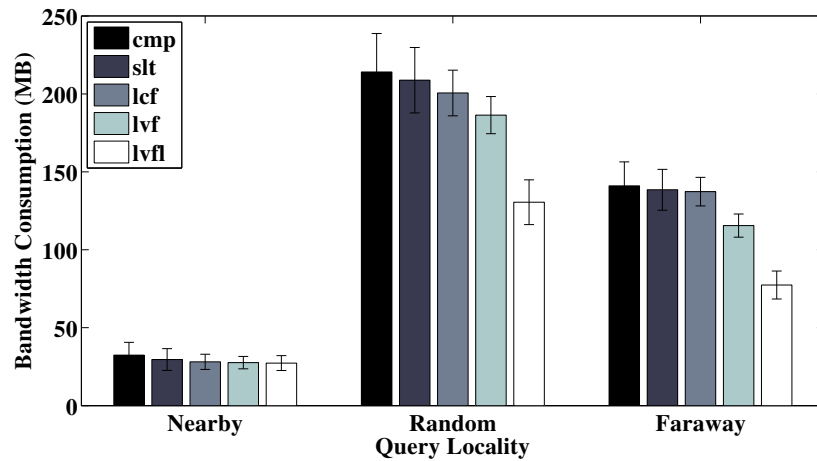


Figure 9. Different levels of queries' localities.

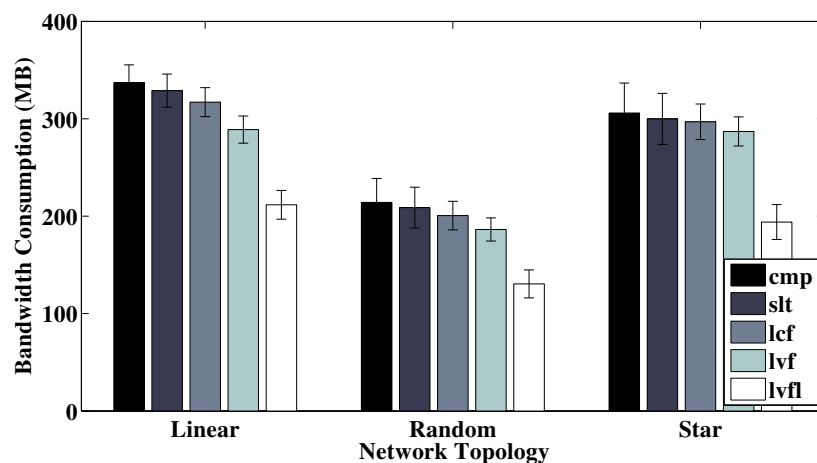


Figure 10. Different network topologies.

9. 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 strategies of individuals and groups. Such learned knowledge can, in turn, be applied to optimize the 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 structure of the flowchart 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 the current state, but also on an 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. The design of such applications on top of decision-driven execution systems remains an open research challenge.

10. Conclusions

In this paper, we outlined a novel paradigm for distributed management of communication and scheduling resources, where 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 communication resource management layer in order to enable more efficient acquisition of data that simultaneously increase decision timeliness and lower decision cost, while improving decision quality. We implemented these ideas within a framework called Athena. The framework was evaluated from two perspectives; namely, resource (i.e., communication bandwidth) consumption and ability to make valid and timely decisions. It was shown that the framework increases the latter while reducing the former. The approach, therefore, holds promise in jointly meeting timeliness, cost and accuracy goals. The paper constitutes a first step on a lengthy road. Many challenges remain that are delegated to future work. For example, more attention is needed to modeling uncertainty. Often, the exact decisions to be made in the context of a mission are not formulated precisely ahead of time; the unknowns relevant to them might not be clear; and the sources who have information on these unknowns might not be identified. The relevance of individual sources may further depend on the context, which itself might not be precisely known. More attention is also needed in modeling cost. A large object from a sensor with a faster connection might be less costly to retrieve than a smaller object from a sensor with a very slow link. Since connection speed may vary dynamically, cost itself might not be exactly known or subject to change. Furthermore, the decision models themselves can vary. While we assumed a model given by AND-OR trees (of predicates) over relevant unknowns, more involved multi-layer models are possible and should be considered in future work. The confluence of the above challenges in uncertainty estimation, cost metrics, context measurement and adaptation to various runtime dynamics and decision models significantly complicates the formulation of scheduling problems and the attainment of solutions with clean optimality properties. Hence, much work is needed on the algorithmic and theory side to understand various (near-)optimality and impossibility results in this area. At last, the notions of anticipation embedded in the work suggest a fundamental trade-off between robustness and optimality. Basing decisions on more aggressive predictions (or anticipation models) that make stronger assumptions regarding future states can improve results when these assumptions hold, but fail more severely when assumptions are violated. In contrast, less detailed predictions (that make fewer assumptions

regarding the future) tend to hedge bets against the unknown, leading to improved robustness with respect to uncertainty. However, they also result in a somewhat impaired average performance. The authors are currently addressing some of the above challenges with a particular focus on exploring the aforementioned tension between robustness and optimality in the context of decision-centric information delivery.

Acknowledgments: Research reported in this paper was sponsored in part by the Army Research Laboratory under Cooperative Agreements W911NF-09-2-0053 and W911NF-17-2-0196, and in part by NSF under Grants CNS 16-18627, CNS 13-02563, 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 herein.

Author Contributions: Jongdeog Lee and Kelvin Marcus designed and developed the current version of Athena, with advice from Tarek Abdelzaher, Amotz Bar-Noy, Ramesh Govindan and Reginald Hobbs; Tanvir Amin significantly contributed to implementation efforts; Shaohan Hu and William Dron developed a previous version of the system; Jung-Eun Kim and Lui Sha contributed real-time decision-driven scheduling extensions; Shuocho Yao and Yiran Zhao worked on various optimizations.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; nor in the decision to publish the results.

References

- Zanella, A.; Bui, N.; Castellani, A.; Vangelista, L.; Zorzi, M. Internet of Things for Smart Cities. *IEEE Internet Things J.* **2014**, *1*, 22–32.
- Khatoun, R.; Zeadally, S. Smart cities: Concepts, architectures, research opportunities. *Commun. ACM* **2016**, *59*, 46–57.
- Bordel, B.; Alcarria, R.; Robles, T.; Martín, D. Cyber–physical systems: Extending pervasive sensing from control theory to the Internet of Things. *Pervasive Mob. Comput.* **2017**, *40*, 156–184.
- De, S.; Zhou, Y.; Larizgoitia Abad, I.; Moessner, K. Cyber–Physical–Social Frameworks for Urban Big Data Systems: A Survey. *Appl. Sci.* **2017**, *7*, 1017.
- Abdelzaher, T.; Amin, M.T.A.; Bar-Noy, A.; Dron, W.; Govindan, R.; Hobbs, R.; Hu, S.; Kim, J.E.; Lee, J.; Marcus, K.; et al. Decision-Driven Execution: A Distributed Resource Management Paradigm for the Age of IoT. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 1825–1835.
- Srivastava, M.; Abdelzaher, T.; Szymanski, B. Human-centric sensing. *Philos. Trans. R. Soc. A* **2012**, *370*, 176–197.
- Stankovic, J.A. Research directions for the internet of things. *IEEE Internet Things J.* **2014**, *1*, 3–9.
- Bu, W.; Tang, A.; Han, J. An analysis of decision-centric architectural design approaches. In Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge, Vancouver, BC, Canada, 16 May 2009; pp. 33–40.
- Cui, X.; Sun, Y.; Mei, H. Towards Automated Solution Synthesis and Rationale Capture in Decision-Centric Architecture Design. In Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), Vancouver, BC, Canada, 18–21 February 2008; pp. 221–230.
- Van Heesch, U.; Eloranta, V.P.; Avgeriou, P.; Koskimies, K.; Harrison, N. Decision-Centric Architecture Reviews. *IEEE Softw.* **2014**, *31*, 69–76.
- Panchal, J.H.; Gero Fernández, M.; Paredis, C.J.; Allen, J.K.; Mistree, F. A modular decision-centric approach for reusable design processes. *Concurr. Eng.* **2009**, *17*, 5–19.
- Paschke, A. A semantic rule and event driven approach for agile decision-centric business process management. In *Towards a Service-Based Internet*; Springer: New York, NY, USA, 2011; pp. 254–267.
- Feng, X.; Richards, G.; Raheemi, B. The road to decision-centric business intelligence. In Proceedings of the IEEE International Conference on Business Intelligence and Financial Engineering (BIFE'09), Beijing, China, 24–26 July 2009; pp. 514–518.
- Blenko, M.W.; Mankins, M.C.; Rogers, P. The decision-driven organization. *Harv. Bus. Rev.* **2010**, *88*, 54–62.

15. Pecharich, J.; Stathatos, S.; Wright, B.; Viswanathan, A.; Tan, K. Mission-Centric Cyber Security Assessment of Critical Systems. In *AIAA SPACE 2016*; Aerospace Research Central: Reston, VA, USA, 2016; p. 5603.
16. Albanese, M.; Jajodia, S.; Jhavar, R.; Piuri, V. Securing Mission-Centric Operations in the Cloud. In *Secure Cloud Computing*; Jajodia, S., Kant, K., Samarati, P., Singhal, A., Swarup, V., Wang, C., Eds.; Springer: New York, NY, USA, 2014; pp. 239–259.
17. Jajodia, S.; Noel, S.; Kalapa, P.; Albanese, M.; Williams, J. Cauldron mission-centric cyber situational awareness with defense in depth. In *Proceedings of the Military Communications Conference (MILCOM 2011)*, Baltimore, MD, USA, 7–10 November 2011, pp. 1339–1344.
18. Decision-Driven Resource Management Patent. Available online: <http://www.google.com/patents/US20150142726> (accessed on 10 November 2017).
19. Liu, S.; Zaraté, P. Knowledge Based Decision Support Systems: A Survey on Technologies and Application Domains. In *Group Decision and Negotiation. A Process-Oriented View, Proceedings of the Joint INFORMS-GDN and EWG-DSS International Conference, GDN 2014, Toulouse, France, 10–13 June 2014*; Zaraté, P., Kersten, G.E., Hernández, J.E., Eds.; Springer: Cham, Switzerland, 2014; pp. 62–72.
20. Vadlamudi, S.G.; Chakraborti, T.; Zhang, Y.; Kambhampati, S. Proactive Decision Support using Automated Planning. *arXiv* **2016**, arXiv:1606.07841.
21. Seligman, L.; Lehner, P.; Smith, K.; Elsaesser, C.; Mattox, D. Decision-centric information monitoring. *J. Intell. Inf. Syst.* **2000**, *14*, 29–50.
22. Pourshahid, A.; Richards, G.; Amyot, D. Toward a Goal-Oriented, Business Intelligence Decision-Making Framework. In *E-Technologies: Transformation in a Connected World: 5th International Conference, MCETECH 2011, Les Diablerets, Switzerland, 23–26 January 2011*; Revised Selected Papers; Babin, G., Stanoevska-Slabeva, K., Kropf, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 100–115.
23. Kim, J.E.; Abdelzaher, T.; Sha, L.; Bar-Noy, A.; Hobbs, R. Sporadic Decision-centric Data Scheduling with Normally-Off Sensors. In *Proceedings of the IEEE International Real-Time Systems Symposium (RTSS)*, Porto, Portugal, 29 November–2 December 2016.
24. Kim, J.E.; Abdelzaher, T.; Sha, L.; Bar-Noy, A.; Hobbs, R.; Dron, W. On Maximizing Quality of Information for the Internet of Things: A Real-time Scheduling Perspective (Invited). In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Daegu, Korea, 17–19 August 2016.
25. Hu, S.; Yao, S.; Jin, H.; Zhao, Y.; Hu, Y.; Liu, X.; Naghibolhosseini, N.; Li, S.; Kapoor, A.; Dron, W.; et al. Data Acquisition for Real-Time Decision-Making under Freshness Constraints. In *Proceedings of the IEEE International Real-Time Systems Symposium (RTSS)*, San Antonio, TX, USA, 1–4 December 2015.
26. Adelberg, B.; Garcia-Molina, H.; Kao, B. Applying Update Streams in a Soft Real-time Database System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, CA, USA, 22–25 May 1995.
27. Kang, K.D.; Son, S.; Stankovic, J.; Abdelzaher, T. A QoS-sensitive approach for timeliness and freshness guarantees in real-time databases. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, Vienna, Austria, 19–21 June 2002.
28. Kang, K.D.; Son, S.H.; Stankovic, J.A. Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases. *IEEE Trans. Knowl. Data Eng.* **2004**, *16*, 1200–1216.
29. Xiong, M.; Wang, Q.; Ramamritham, K. On earliest deadline first scheduling for temporal consistency maintenance. *Real-Time Syst.* **2008**, *40*, 208–237.
30. Tufekci, S.; Wallace, W.A. The Emerging Area Of Emergency Management And Engineering. *IEEE Trans. Eng. Manag.* **1998**, *45*, 103–105.
31. Hu, S.; Li, S.; Yao, S.; Su, L.; Govindan, R.; Hobbs, R.; Abdelzaher, T. On Exploiting Logical Dependencies for Minimizing Additive Cost Metrics in Resource-Limited Crowdsensing. In *Proceedings of the IEEE International Conference on Distributed Computing in Sensor Networks (DCoSS)*, Fortaleza, Brazil, 10–12 June 2015.
32. Wang, S.; Abdelzaher, T.; Gajendran, S.; Herga, A.; Kulkarni, S.; Li, S.; Liu, H.; Suresh, C.; Sreenath, A.; Wang, H.; et al. 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 (DCOSS '14)*, Marina Del Rey, CA, USA, 26–28 May 2014; IEEE Computer Society: Washington, DC, USA, 2014; pp. 92–100.

33. Lee, J.; Kapoor, A.; Amin, M.T.A.; Wang, Z.; Zhang, Z.; Goyal, R.; Abdelzaher, T. InfoMax: An Information Maximizing Transport Layer Protocol for Named Data Networks. In Proceedings of the 2015 24th International Conference on Computer Communication and Networks (ICCCN), Las Vegas, NV, USA, 3–6 August 2015; pp. 1–10.
34. Zhang, L.; Estrin, D.; Burke, J.; Jacobson, V.; Thornton, J.D.; Zhang, S.B.; Dmitri, G.T.K.C.; Massey, K.D.; Papadopoulos, C.; Lan, T.A.; et al. *Named Data Networking (NDN) d NDN-0001*; NSF FIA Kickoff Meeting: Arlington, VA, USA, 2010.
35. Marcus, K.; Cannata, J. Dynamically allocated virtual clustering management system. In Proceedings of the SPIE Proceedings on Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IV, Baltimore, MD, USA, 22 May 2013; Volume 8742.
36. Ra, M.R.; Liu, B.; La Porta, T.F.; Govindan, R. Medusa: A programming framework for crowd-sensing applications. In Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys), Low Wood Bay, Lake District, UK, 25–29 June 2012.
37. Jiang, Y.; Xu, X.; Terlecky, P.; Abdelzaher, T.; Bar-Noy, A.; Govindan, R. MediaScope: Selective On-demand Media Retrieval from Mobile Devices. In Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Philadelphia, PA, USA, 8–11 April 2013.
38. Bar-Noy, A.; Johnson, M.P.; Naghibolhosseini, N.; Rawitz, D.; Shamoun, S. The Price of Incorrectly Aggregating Coverage Values in Sensor Selection. In Proceedings of the IEEE International Conference on Distributed Computing in Sensor Networks (DCoSS), Fortaleza, Brazil, 10–12 June 2015.
39. US Naval Research Lab Networks and Communication Systems Branch. *Extendable Mobile Ad-hoc Network Emulator (EMANE)*; U.S. Naval Research Lab: SW Washington, WA, USA, 2016.
40. Dron, W.; Leung, A.; Hancock, J.; Aguirre, M.; Thapa.; Walsh, R. *CORE Shim Design Document*; NS-CTA Technical Report; Network Science CTA: Cambridge, MA, USA, 2014.



© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).