Towards Efficient Real-Time Decision Support at the Edge

Kyoung Don Kang∗
kang@binghamton.edu
State University of New York at Binghamton
Binghamton, New York

ABSTRACT
In the emerging Internet of Things (IoT) paradigm, the need for real-time decision support is increasing fast to support key applications, such as smart health, disaster recovery, or battlefield monitoring. In these applications, it is essential to efficiently retrieve fresh sensor data and process them in a timely manner to aid in decision making. However, related work is relatively scarce. In this paper, we formulate the problem of real-time decision support and explore a suite of scheduling methodologies to efficiently retrieve and process sensor data for real-time decision support subject to timing and data freshness constraints at the network edge and discuss open research issues.

CCS CONCEPTS
• Computer systems organization → Embedded and cyber-physical systems; Real-time systems.

KEYWORDS
Internet of Things, Web of Things, Real-Time Decision Support, Task Deadlines, Data Freshness

1 INTRODUCTION
The Internet of Things (IoT) consists of a vast number of things (sensors and actuators), gateways, and the cloud. Sensors monitor the real-world status, while gateways orchestrate them and connects them with the cloud [7]. The Web of Things (WoT) makes real-time sensor data available through the prevalent and familiar web interface. A main objective of IoT and WoT is to monitor the real-world states, analyze them, and control actuators in real-time to support important applications, such as smart health, transportation, and energy.

Cloud-based modern stream processing engines, such as Apache Storm [18], Flink [6], and Spark Streaming [17] and cloud-based IoT frameworks, e.g., AWS IoT [1] and Google Cloud IoT [8], basically assume data are transferred to and analyzed in the cloud. By leveraging abundant resources in the cloud, these stream processing engines and IoT frameworks support high throughput. They, however, suffer from long latency to transfer all data from IoT devices to the cloud for analytics and transmit control signals, if any, from the cloud to the actuators. As a result, many decision support deadlines can be missed and sensor data/control signals may become stale. The problem can be exacerbated if billions of sensors stream data to cloud-based data analytics frameworks, potentially creating bottlenecks in the backhaul network. Further, network connections to the cloud may be unavailable or intermittent in certain IoT applications, e.g., disaster recovery or battlefield monitoring. Thus, they are inadequate for time-critical real-time decision making to support, for example, healthcare, transportation management, and disaster recovery.

A plausible alternative is for an IoT gateway or micro edge servers, e.g., a cluster of Raspberry Pi nodes or mini-ITX servers [16], to analyze real-time data collected from the sensors they manage for real-time decision support, while meeting timing and data freshness constraints of real-time decision tasks. The IoT gateway communicates with embedded sensors and actuators using heterogeneous (wireless/wired) communication channels and protocols, while edge servers process the retrieved fresh sensor data in a timely manner even when no network connection to the cloud is available.
to support, for example, disaster recovery or battlefield monitoring. Thus, efficient real-time data retrievals and analysis using micro-edge servers (stationary or portable) have several merits: (1) the latency and bandwidth consumption for data transfers between sensors and the cloud can be significantly reduced and (2) security/privacy concerns could be alleviated by analyzing sensitive data locally rather than uploading all data to the cloud.

In this regard, several novel work has recently proposed to support efficient real-time sensor data retrievals [10–12] and stream processing in edge devices [5, 7]. However, the former only focuses on efficient retrievals of real-time sensor data. In contrast, the latter assumes data continuously arrive at the stream processing engine and focuses on efficient processing of given data streams in a single edge node without considering efficient retrievals of real-time sensor data. We observe that neither of them is sufficient in that both timely retrievals and processing of real-time sensor data are required. A naive approach that simply combines existing approaches to real-time data retrievals and stream processing may not succeed, since the timeliness and data freshness requirements of real-time decision tasks may conflict with each other. More frequent updates of sensor data may enhance the freshness of the sensor data but may incur more energy/network bandwidth consumption and deadline misses due to too much data to transfer and process. However, wrong decisions could be made based on stale data, if sensor data are updated too infrequently. To tackle these challenges, it is required to formally define deadlines and data freshness constraints of real-time decision tasks, while striking a balance between them to retrieve just enough data with minimal redundancy and effectively schedule real-time decision tasks and their data retrievals. Therefore, in this paper, we define the timeliness and freshness requirements, design an overarching system framework, and sketch holistic scheduling methodologies for cost-effective retrievals and processing of real-time sensor data subject to timing and data freshness constraints.

The rest of this paper is organized as follows. In Section 2, we provide background and formulate a problem statement, while giving an overview of the overall system structure. In Section 3, we discuss scheduling algorithms to efficiently retrieve and process real-time sensor data in an edge node. In Section 4, open research issues are discussed. Finally, Section 5 concludes the paper and discusses future work.

2 BACKGROUND AND PROBLEM FORMULATION

In this section, we give illustrative examples to motivate our work, discuss a real-time task model for decision support, describe timing and data freshness requirements of real-time decision support tasks, and formulate the research problem.

2.1 Examples of Real-Time Decision Tasks and a System Overview

For example, let us consider a real-time decision support scenario in a smart city equipped with cameras and other sensors deployed along major roads [11]. If a fire alarm is triggered in a building or a wearable device detects an urgent health problem of the wearer, the real-time data engine in a nearby emergency vehicle needs to find the best route to the building or home by retrieving and analyzing data from sensors, e.g., cameras and bridge health sensors, available along alternative routes. Certain roads or bridges could be unavailable after a disaster such as an earthquake. Or, crowds may move around and block different roads at different times for demonstrations or riots in a politically unstable region [10]. In such applications, tasks that retrieve real-time data from sensors and process them for decision making should complete within the deadlines using fresh sensor data representing the current real-world status.

Figure 1 gives an overview of retrievals and analyses of real-time sensor data for decision support. A client request is submitted when an event of interest occurs or a user explicitly submits a decision query. The sensor lookup module in Figure 1 provides directory service: it knows which sources can provide the data needed by client requests and provides sensor selection service [2]. Further, it stores the data freshness constraints of sensors. Once relevant sensors are selected and their freshness constraints are looked up, the sensor data retriever derives a sensor data retrieval plan to efficiently fetch the fresh data. When the sensor data retriever acquires all the fresh sensor data demanded by a decision support task, the task is inserted into the priority-based ready queue of the real-time data engine that processes real-time decision tasks using fresh data. Based on the results, the real-time data engine provides recommendations for decision making. For example, it provides the best route recommendation for an emergency vehicle to take to the place of the event or provides triage decisions to properly prioritize the emergency treatment of victims in a disaster area based on their medical conditions. In this paper, we mainly focus on efficient retrievals and processing of real-time sensor data—the red boxes in Figure 1.
2.2 Real-Time Task Model and Timeliness and Data Freshness Constraints

In this paper, task arrival patterns are unknown a priori, since a real-time decision task is submitted to the real-time data engine upon a relevant event, e.g., a fire or medical emergency, or submitted by a user at anytime as discussed before. A real-time task, \( \tau_i \), is associated with the deadline, \( D_i \), for a timely decision support. \( \tau_i \) retrieves a set of data objects, \( S_i \), for decision support. \( O_{i,j} \in S_i \) is a sensor data object retrieved and analyzed by \( \tau_i \). We consider that sensors are normally off (in a low-power mode) to save limited resources, e.g., the network bandwidth or energy, but wake up and provide data upon request instead of periodically feeding data regardless whether there is a demand or not [10–12].

For data freshness management, we adopt the notion of data temporal consistency, i.e., freshness, originally devised in real-time databases. It determines whether a data object is fresh or not based on its validity interval [15]. A data object \( O_{i,j} \) accessed by \( \tau_i \) is considered fresh, if \( t \leq t_v(O_{i,j}) + V I(O_{i,j}) \) where \( t \) is the current time, \( t_v(O_{i,j}) \) is the timestamp taken when the real-time data engine successfully receives the data, and \( V I(O_{i,j}) \) is the validity interval of \( O_{i,j} \). Note that different sensors, e.g., bridge health sensors and cameras, can have different validity intervals. As a result, \( O_{i,j} \) with the relatively short validity interval may have to be updated multiple times until all the other data needed by \( \tau_i \) are retrieved, wasting precious resources. Once all fresh data are received, the decision task that process them is added to the global queue of the real-time data engine in priority order determined by, for example, deadlines, data freshness constraints, or arrival times depending on the task scheduling policy. The real-time data engine executes the task at the head of the queue using the retrieved sensor data.

2.3 Problem Statement

Ideally, all real-time decision tasks should be completed within their deadlines using fresh data, while minimizing the consumption of resources, e.g., the network bandwidth, CPU cycles, and energy. In reality, however, it is very challenging to achieve the goal, since tasks may arrive at any time and need to retrieve multiple data items with different validity intervals and process them subject to timing and freshness constraints. If the challenges are not addressed, timing and data freshness constraints can be violated significantly, damaging the effectiveness and validity of real-time decisions. Potential consequences may include mere inconvenience, decreased safety, or even losses of life. Therefore, we explore effective approaches to scheduling sensor data retrievals and decision support tasks that analyze the data, respecting timing and data freshness constraints. Our objective is to meet timing and data freshness constraints if there is any schedule that achieves it, while minimizing repeated sensor data updates and processing within a task and across tasks that share common sensor data to significantly reduce the resource consumption due to repeated retrievals and processing of fresh sensor data shared by multiple tasks.

3 Efficient Scheduling of Real-Time Decision Tasks

In this section, we first discuss how to schedule real-time data retrievals, how to allow decision tasks to effectively share input/intermediate data, and how to let the processor cores efficiently share workloads.

3.1 Efficient Scheduling of Sensor Data Retrievals

Kim et al. [11, 12] present a novel scheduling algorithm, called EDEF-LVF (Earliest Deadline or Expiration First–Least Volatile First). When a decision task \( \tau_i \) needs to retrieve the data in \( S_i \) and process them within the deadline \( D_i \), EDEF first computes \( V I_{i,\min} = \min\{VI(O_{i,1}), \ldots, VI(O_{i,|S_i|})\} \) and then \( \gamma_i = \min(D_i, V I_{i,\min}) \). When there are \( N > 1 \) decision tasks, EDEF schedules \( \tau_i \) first if \( \gamma_i = \min(\gamma_1, \ldots, \gamma_N) \); that is, EDEF schedules the task with the earliest deadline or validity interval expiration. When no data is shared between decision tasks, EDEF is proven to be optimal in that if any feasible order exists to schedule a set of decision tasks, EDEF can always schedule the task set, meeting the task deadlines and data freshness constraints [11, 12].

When the task \( \tau_i \) with the earliest deadline or validity interval expiration is run, the data needed by \( \tau_i \) is retrieved in the least volatile first (LVF) order; that is, the data object with the longest validity interval is retrieved first. It is proven that LVF is optimal for the retrieval of data objects within each task, \( \tau_i \), in that LVF retrieves all data in \( S_i \) just once and meet their freshness constraints if there is any schedule that does it [11, 12].

Therefore, EDEF-LVF that combines EDEF and LVF to schedule real-time decision tasks and data retrievals in each task, respectively, is optimal on the condition that every task accesses a disjoint set of data without sharing any data with any other task. Assuming no data sharing, however, is very pessimistic and may result in waste of resources, because data access skew is prevalent and hot data can be accessed much more frequently by many tasks [20]. For example, many tasks may query the traffic speed or availability of a busy road after a disaster. In such cases, repeatedly retrieving the same data although they are still fresh increase the network bandwidth and energy consumption of normally-off sensors. Although several heuristics are proposed in [11] to address this issue, none of them is optimal or near optimal in the presence of any data sharing. The main challenge is...
that it is very difficult to accurately predict when the next task that reuses a sensor data object updated by a previous task will finish and whether it will finish before the validity interval of the reused data expires.

To shed light on the problem, we introduce a new concept of **extensible validity intervals**. Initially, the extensible validity interval of a sensor data object is equal to the validity interval; however, the real-time data engine extends it if consecutive sensor readings remain similar. For example, subsequent temperature readings in a smart home or surveillance images in a high security area in a building, can be similar with little difference. A significant body of work has been done to minimize redundant sensor data transmissions [3]. For example, a source (sensor) and sink can share a statistical model to predict sensor data values in the near future. The sensor transmits data to the sink and the sink updates the model based on the new data only if the new sensor data value deviates from the model by more than the pre-specified threshold and, therefore, it is unpredictable in the sink.

To leverage model-driven sensing for efficient real-time decision support based on fresh sensor data, we propose to **extend the validity interval of the data object to the infinity and treat the data as a stored data item whose value does not change over time unless an external entity, e.g., a user or sensor in the example above, explicitly updates it**. When a new version of a sensor data object is needed, the sink, i.e., the real-time data engine, does not have to explicitly fetch it from the sensor, since the sensor will send a new sensor data object when the model cannot predict it with enough accuracy due to a considerable change of the sensor data value, e.g., an unexpected temperature hike. In this way, common sensor data can be shared among multiple decision tasks that need the data with no repeated retrievals without losing noteworthy information. After a task updates a data object, subsequent tasks that access the same data do not have to retrieve it as far as the data remains fresh based on its extensible validity interval. As a result, more decision tasks can be processed in a timely manner using fresh data based on the predictive model. Although the idea is simple, to the best of our knowledge, no previous work has considered the formal definition of extensible validity intervals and applied it to efficiently schedule real-time decision tasks.

### 3.2 Efficient Scheduling of Decision Tasks

When a task \( T_j \) has successfully retrieved \( n \leq |S| \) fresh sensor data that it needs for real-time decision support, the task is ready to analyze the data. Given the limited resources, an important question is how to schedule the real-time data analysis tasks to significantly reduce any redundant computations and memory accesses. To this end, it is desirable for a real-time data engine to **allow tasks to share common input/intermediate data and computations** to significantly reduce repeated accesses and analyses of common data. Further, it is desirable to **load balance among the embedded processor cores** in the edge device on which the real-time data engine runs to substantially reduce deadline misses due to load imbalance. However, this is a challenging problem, since load balancing is an NP-complete problem and potential data dependencies as well as freshness constraints of data possibly shared across tasks can further complicate the problem.

#### 3.2.1 Data Sharing for Efficient Processing of Decision Tasks

In an information system, multiple queries often access common data, e.g., traffic data of busy intersections or severe weather data. In fact, large data access skews are prevalent [20]. For example, the well-known 80/20 rule indicates that 20% of data are accessed for 80% of accesses. However, a direct application of this approach may result in many deadline misses, because real-time tasks should be delayed for aggregation. For instance, Lang et al. [13] intentionally delay database queries to combine them. Their approach increased the average response time by 43%, even though it decreased the energy consumption by up to 54%.

To aggregate as many decision tasks that access common data without jeopardizing the timeliness and share as many fresh data as possible, we schedule the task with the highest priority (e.g., the earliest deadline or expiration task) first, while **scanning the ready queue sorted in non-ascending priority order backwards to merge tasks with similar data needs together in the background and assign them to the same core to enhance caching and reduce contention for the bus and memory controller among the cores**. In this way, we aim to significantly reduce duplicate data accesses and analyses without disrupting high priority decision tasks at or near the head of the global queue in the real-time data engine.

In Figure 2, for example, tasks arrive at the global earliest deadline first (GEDF) queue. Note that our approach for merging real-time decision tasks is not tied to a specific scheduling
Towards Efficient Real-Time Decision Support at the Edge

SEC 2019, November 7–9, 2019, Arlington, VA, USA

algorithm but applicable to any priority-driven scheduling principle for efficient processing and retrievals of real-time sensor data. For example, instead of GEDF, the EDEF and EDEF-LVF algorithms [11, 12], or those algorithms extended to leverage extensible validity intervals as discussed in Section 3.1 can be further enhanced to take advantage of our task aggregation technique. Thorough analytic and empirical comparisons of their performance and further improvements are reserved for future work.

3.2.2 Load Sharing. Tasks assigned to one core may suffer from many deadline misses if the core gets overloaded. According to the queuing theory [9], the queuing delay increases super-linearly as the load increases and becomes unbounded when the system gets saturated. Also, in our recent work [21], we have empirically observed that the deadline miss ratio increases abruptly when real-time systems get overloaded. This is a serious issue especially in edge devices with limited resources.

In this paper, we propose to extend work stealing [4] to mitigate load imbalance and enhance the timeliness in a real-time data engine running in a multicore edge device. For example, an idle core can steal the task with the earliest deadline or validity expiration from another core to enhance the overall timeliness and reduce potential head of line blocking that may result in cascading deadline misses. In our approach, an idle core steals one or more tasks from a busy core, if it expects that it can complete them within the deadlines based on queueing disciplines. A core i sets its busy bit if the difference, δ_i, between the processing rate, \( \mu_i \), and task arrival rate, \( \lambda_i \), of core i is less than a specified headroom, θ; that is, \( \delta_i = \mu_i - \lambda_i \leq \theta \). When there are m cores in the edge node, an idle core j steals one or more task from core i (i ≠ j) if \( \delta_i = \min(\delta_1, \ldots, \delta_m) \) and core j is expected to meet the deadlines of the migrated tasks with a high likelihood after the potential work stealing. In this paper, idle core i attempts to steal a subset of such tasks if its predicted tail latency, e.g., 99.9 percentile queueing latency, is shorter than their deadlines after accepting them.

Let us assume that tasks arrive under a Poisson process with an arrival rate \( \lambda \) at the single global queue in the real-time data engine. Core i processes the assigned tasks with rate \( \mu_i \), with the mean service time is \( T_i = \frac{1}{\mu_i} \). When the rate of task arrivals at core i is \( \lambda_i \), the load is \( \rho_i = \frac{\lambda_i}{\mu_i} \). When core i intends to steal n ≥ 1 tasks from a busy core, it recomputes \( \rho_i \) by tentatively increasing \( \lambda_i \) by n. Given that, the estimated \( x^{th} \) percentile, e.g., 99.9-percentile, wait time of core i is:

\[
T(x) = \frac{T_i}{1 - \rho_i} \ln \left( \frac{100C}{100 - x} \right)
\]

according to the M/M/1 queuing model where C is Erlang’s C formula that computes the probability a new task must wait in the queue. The C values are publicly available as a table that can be stored in memory for a fast look-up. In principle, our approach could be applied to heterogeneous processors, e.g., ARM big.LITTLE processors, since our approach only needs the task arrival rate at core i (\( \lambda_i \)) and its processing rate (\( \mu_i \)) measured per sampling period, e.g., 1s.

4 OPEN RESEARCH ISSUES

There are many open research issues towards efficient real-time decision support at the network edge. A summary of a few research issues closely related to our work discussed in this paper follows.

- **Parallel data retrievals**: In this paper, we mainly considered real-time data retrievals over a single channel. Alternatively, parallel retrievals of multiple data needed by a decision task using different network paths or wireless links could enhance the timeliness and freshness, if done properly. For example, more and more IoT devices support multiple radios that can be leveraged to transmit/receive multiple sensor data simultaneously. Optimal partitioning of sensor data retrievals over multiple channels, however, can be reduced to the bin packing problem, which is NP-hard. In addition, different radios may have different throughput and energy efficiency. Some radios on a single device cannot be used at the same time, since they share the same hardware modem [14]. In-depth research is required to efficiently utilize parallel data retrievals for real-time decision support.

- **Scale-up using general purpose graphics processing units (GPUs)**: In many applications, GPUs boost performance due to the extensive parallel computation capacity and high memory bandwidth. In an edge node, enhancing the timeliness of real-time decision tasks in an edge device using the GPU (if equipped) is an attractive option to consider. A challenge is the communication overhead of the PCIe bus between the CPU and GPU, which is an order of magnitude slower than the GPU device memory. To amortize the overhead, real-time sensor data should be batched; however, batching may incur violations of task timing and data freshness constraints due to the latency. A viable approach is to use the on-chip GPU that is on the same die as the CPU. However, this approach also faces several challenges. First, the integrated GPU has to share the DRAM with the CPU, losing the advantage of the high bandwidth provided by the GPU memory (e.g., GDDR5 memory). In the integrated CPU-GPU system, efficiently utilizing the memory bandwidth shared between the CPU and GPU is crucial. Also, minimizing...
the overhead of the CPU-GPU synchronization and cache coherency is important.

- **Scale-out using a cluster of edge devices/nodes:** In [16], a portable edge cluster is built using commodity hardware that can be used, for example, in an emergency vehicle to support sophisticated real-time tasks for disaster recovery, e.g., image processing and natural language translation. An open issue is how the nodes in the cluster can effectively share workloads that may vary from time to time. Another important issue is how to opportunistically leverage the resources of any other edge nodes in the vicinity, e.g., smartphones or other edge clusters in different emergency vehicles. It is essential to support reliable real-time performance, while edge devices join and leave due to mobility or failure.

- **Situation-aware security/privacy:** Edge computing has good potential to enhance security and privacy. For example, sensitive health data can be anonymized before being uploaded to a cloud. In [19], a live video analytics framework denatures video streams by selectively blurring faces at the full frame rate for privacy. When there is a medical emergency or a missing child, however, there should be an overriding mechanism to access the information of the corresponding person. Care should be taken though, since it can be misused by an attacker to steal private information. Walking on the fine line, while facilitating real-time decision support using fresh data, is a challenging yet important research issue that deserves more attention from the IoT and security communities.

## 5 CONCLUSIONS AND FUTURE WORK

For real-time decision support in IoT, efficient retrievals and processing of real-time sensor data subject to timing and data freshness constraints are important. In this paper, we first introduce a real-time task model and the notion of extensible validity intervals. Second, we leverage extensible validity intervals for cost-effective real-time data retrievals. Third, we devise data and load sharing techniques in the real-time data engine running in a resource-constrained edge node. In addition, we discuss important open research issues in Section 4. In the future, we will perform in-depth research of these issues and thoroughly evaluate their effectiveness empirically as well as analytically.

## ACKNOWLEDGMENTS

This work was supported, in part, by NSF grant CNS-1526932.

## REFERENCES


