Journal of Parallel and Distributed Computing 167 (2022) 97–108

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Goal-driven scheduling model in edge computing for smart city applications

Yongho Kim [a,*], Seongha Park [a], Sean Shahkarami [b], Rajesh Sankaran [a,c], Nicola Ferrier [a,b], Pete Beckman [a,c]

[a] Argonne National Laboratory, Lemont, IL, USA
[b] University of Chicago, Chicago, IL, USA
[c] Northwestern University, Evanston, IL, USA

ARTICLE INFO

Article history:
Received 24 August 2021
Received in revised form 18 February 2022
Accepted 23 April 2022
Available online 5 May 2022

Keywords:
Goal-driven scheduling
Context-aware scheduling
Edge computing
Logical reasoning

ABSTRACT

A formidable challenge in scheduling user applications lies in collecting and representing the user's goals and requirements. We introduce a "science goal" as a mechanism for users to define scientific objectives and conditions of interest. To provide an abstraction to run applications on an ensemble of edge computing nodes, we implement a two-layered scheduler—cloud and edge scheduler. In this scheduling model, the users submit their goals to the cloud scheduler. These goals are conveyed to the appropriate nodes based on a variety of constraints including geographical area, resource availability, node capabilities, and applicability. The edge scheduler, with complete understanding of the current conditions, assumes the responsibility for executing the applications on the nodes so that the users' science goals are met. This paper provides a framework for the two-layered scheduling model for goal-driven edge computing and motivates and informs its architecture through a case study.

1. Introduction

Sensing and computing capabilities deployed in cities can enable an urban-scale measurement capability to support diverse scientific and operational objectives. Expanding on traditional sensor networks, remote programmability provides a means for obtaining urban measurements for various scientific and policy goals [9,29] through the introduction of artificial intelligence (AI) and machine learning (ML) algorithms that can be fine-tuned at the "edge" (of the network). For example, traffic congestion in the streets causing air pollution and noise, unauthorized drone flights, and street flooding can be measured by capturing and analyzing data from the same set of deployed sensors, providing a form of software-defined sensing. Moving the sensed data to a cloud infrastructure offers unlimited computational power for processing. However, it has the potential shortcomings of requiring high network bandwidth, continuous connectivity for real-time processing, and high latency to respond to rapid changes in the environment; and it may also raise privacy concerns. The ability to compute "at the edge" places the computation close to the source of the data and allows in situ processing and sending only the necessary information over the network, overcoming these challenges. For example, a user application could potentially transfer a count of detected cars to the cloud without sending the raw images that might show the driver's face and vehicle identity (color, features, and plate). Additionally, in edge computing, user applications are able to change their computing behavior in near-real time based on the current perceived environmental *context*. This ability improves the quality of service while possibly reducing resource utilization by skipping computation under unfavorable or uninteresting circumstances and conditions.

Computing at the edge does, however, face some fundamental limitations. First, space and power constraints dictate that computing at the edge is always limited in comparison with the potentially limitless resources that can be provisioned in the cloud. Second, field-deployed edge devices are typically connected to the cloud infrastructure through wired and wireless Internet that vary widely in quality and consistency. These challenges necessitate a job scheduler to ensure that user applications can share the limited computing and network resources and can be consistently scheduled and executed in spite of unreliable network connectivity.

Applications are often executed at the edge to serve a scientific goal. In traditional high-performance computing (HPC) and cloud computing environments, user applications are bundled into a "job" that represents the tasks and actions. These jobs contain metainformation on how to run user applications and which HPC

* Corresponding author.
E-mail address: yongho.kim@anl.gov (Y. Kim).

https://doi.org/10.1016/j.jpdc.2022.04.024

nodes to use, as well as information regarding resource and temporal (e.g., deadline) requirements. Unlike HPC or cloud computing, in edge computing input data is often locally sourced from sensors. The edge task needs to define sensor instruments as input sources along with contextual information about the input. This can be expressed as a condition that illustrates a situation in which interesting events have occurred or will occur (e.g., rain gauge ticks when raining, traffic congestion in rush hours, sky image during daytime). However, the current approaches in mapping contextual information into the system components that support the collection of such information lack sophistication [2,23]. Moreover, understanding system functionality for describing a job request is a nontrivial task for users. To support submission of user jobs in edge computing, we define a *science goal* that captures the job description, including objectives, contexts of interest, and system and application-specific requirements. The ontology and edge node metainformation supported by the system help users detail their science goal and define triggers for running user applications.

To support user science goals in edge computing, we propose a two-layered goal-driven scheduling model that takes into account both *resource* and *context*. The cloud scheduler, one part of the scheduling model, accepts user jobs and generates science goals to distribute them to edge computing nodes. In this process, the cloud scheduler oversees capabilities and availability of edge nodes and performs the macro-level decision on where and when user science goals are served. It is also responsible for informing users about the status of their science goals. The edge scheduler, the other part of the scheduling model, is run on each edge node and makes micro-level decisions on what to run at any given time in order to accomplish given science goals. The main objective of the edge scheduler is to capture circumstances of interest at that location (i.e., the context) and allocate resources to applications corresponding to the circumstances. It is important to note that the authors in [34] state, "Context-aware computing has proven to be successful in understanding sensor data.". Indeed, context-aware computing has been employed in the Internet of Things (IoT) [16,18,23] and sensor networks [1,33] to reason about the current context of local environment. The edge scheduler must understand the current context and determine what to run. To meet these demands, we have developed a goal-driven scheduling model for edge computing to be used in smart city applications. The major contributions of this work are as follows:

- A context representation in the proposed scheduling model that refines the current context using sensors attached to edge computing nodes and evaluates event-driven and event-condition-action rules using new knowledge generated from raw sensor data. This supports both "intelligence" and "architecture"—two of the main characteristics of the context-aware IoT described in [34].
- A scheduling model that defines the structure of a science goal. Science rules that are logically expressed trigger scheduling process whenever they change.
- A case study showing the use of the proposed model for a science example. This provides a demonstration of how scientists can specify their science goals using our scheduling model.

The rest of the paper is organized as follows. Section 2 describes relevant literature on edge computing, context awareness, and scheduling on edge computing environment. Section 3 illustrates the proposed scheduling model in detail. Section 4 describes the logic of scheduling process in the proposed model. Section 5 demonstrates a case study conducted to validate how the proposed scheduling model works for a scientific study and discusses the future research points. Section 6 concludes the work.

2. Background and related work

In this section we introduce the existing technologies and software platforms that were developed to support edge computing. We also describe what aspects have been considered when scheduling and why and how the concept of context-awareness can help schedule applications appropriately at the edge.

2.1. Edge computing environment and platforms

In recent years, the edge computing environment has rapidly evolved along with the advancements in both hardware and software. Single-board computers (SBCs) equipped with powerful CPUs, GPUs, and TPUs[1] have created a new class in the computation spectrum, ranging from HPC-level computation down to microprocessor-level computation. Because SBCs were designed to be light-weight and low-power, they have been actively deployed and used in sensor networks [11], IoT [22], healthcare [32], and teaching [47]. Nvidia's Jetson devices [31] host a powerful GPU within the device and are capable of heavy computations such as running ML models for inference [28]. However, they still are unable to run multiple heavy computations at the same time, mainly because the required amount of memory for running those ML models can easily exceed the memory capacity of the devices. In [24], the authors indirectly showed how much memory ML models would require in order to run on SBCs for inference.[2] Software advancements support deploying user applications on this diversity of hardware. The technologies of containerization and container orchestration tools such as Kubernetes [8] have been dominantly used in deployment. Since those technologies allow scaling, managing a massive number of devices in IoT and sensor networks becomes easier.

The project Smarter [15], initiated by researchers in ARM, provides a platform to deploy user applications on edge computing nodes that interact with IoT endpoints including sensors and actuators. An instance of a Kubernetes cluster runs in the cloud and governs all Kubelet node agents running on each edge computing node. User applications submitted to the cloud's Kubernetes master node are fetched to Kubelet node agents that can run the user applications. Because the master node runs in the cloud, disconnected Kubelet node agents from their master node do not serve any job until they get reconnected.

Rainbow [17] is an architecture that allows the deployment of smart city applications in an edge computing environment. In Rainbow, physical systems such as sensors and actuators are abstracted as a virtual object. Virtual objects are then exposed to the computing nodes in an edge network. Since a network can have multiple nodes, a given job is taken by any node that has the capability to do so. Rainbow had no central scheduler because the intention was to bring "emergence"—accomplishing local objectives by individuals leads to accomplishing a global objective that is unknown to the individuals.

Foggy [38] provided a platform that orchestrates workloads for computing nodes that exist in both the cloud and edge. Jobs that specify edge-related requirements such as particular sensors, regions of interest, and limitations on network traffic were scheduled to the corresponding edge computing nodes that were available to fulfill the requirements. This requirement-aware scheduling greatly improved the quality of service while ensuring data privacy. However, similar to Smarter, edge nodes should be available from the master node in the cloud in order for the master node to schedule jobs.

[1] Tensor processing units for running neural network models.

[2] The study showed the required amount of memory for training on SBC devices. With a batch size of 1, one can estimate how much memory the model would require for inference.

Waggle [6], Array of Things (AoT) [10], and SAGE [7] are projects designed to help scientists understand environmental impacts, including those caused by activities in cities [41]. The AoT project deployed more than 100 Waggle computing nodes in the city of Chicago. Each node was equipped with a sensor pod; and many had two cameras, one facing the street and the other pointed up to view the sky. Raw sensor readings from the sensor pod and cameras were processed inside the AoT nodes, and the processed results were sent back to the cloud. The majority of the computations performed in AoT nodes were designed to be light-weight and accommodated by resources that the nodes offered. However, this design highlighted the need for the work presented here as heavier ML computations are presented.

The job scheduler in most of the existing platforms described above exists in the cloud and controls execution flow of jobs from the cloud. This mechanism is not reliable if jobs are not scheduled on edge nodes just because the nodes may frequently experience weak network signals and temporally lose the ability to talk to the cloud. The proposed model allows the edge scheduler to controls execution flow of jobs independently from the cloud scheduler, making job execution decoupled from node management. Additionally, the existing platforms are node, i.e. resource, oriented such that they deliver jobs whenever and wherever the platform can. Our scheduling model is job, i.e. goal, oriented and delivers jobs whenever the job wants to run. This makes a big difference when jobs need to be executed at the right timing to capture events of interest.

2.2. Context representation and reasoning

Representing context information plays a special role in enabling context-aware computing on edge computing nodes. In [39], the authors state: "One significant aspect of this emerging mode of computing [which indicates context-aware computing] is the constantly changing execution environment." Such environment changes affect the quality of service and validity of data that user applications produce. This aspect should be captured and understood in order for the edge scheduler to successfully meet the requirements of user applications and achieve their science goals.

As described in [34], context may be represented in many ways, including key-value, markup scheme, graphical modeling (e.g., UML and object role modeling), object-based modeling, and logic-based modeling. Key-value and markup scheme are simple ways to store information and designate relationships between entities, but their expressiveness is limited for complex reasoning. Object-based models represent relationships between objects and attributes of objects, but they lack supporting operations needed to generate new information from stored information. Logic-based models have been especially highlighted because they support logical operations for complex reasoning. In logic-based models, relating literals with logical operations creates a fact or rule of a context, and multiples of these can be used to build up a higher level of context. For example, a context of taking an umbrella when raining can be expressed as $Raining \wedge Has(I, Umbrella) \Rightarrow Take(I, Umbrella)$.

In context-aware computing, rules are used to produce different run-time behaviors of user applications that depend on context. Event condition action (ECA) is one form of rules to enable reactive functionality [3]. In [4], an XML-style representation of ECA was proposed to support defining rules as well as operations on the rules. Performing operations on rules along with perceived events helps identify the next action. Rules should be constantly evaluated at runtime to reflect real-time changes such as city road routing for avoiding traffic congestion [42].

A context reasoner is an inference engine that estimates the current context by extracting contextual information using rules and facts. As the authors in [35] note, context reasoning should happen at the edge because response time upon environmental changes is slower in the cloud. This inference engine first takes sensor inputs from the local environment and maps them into user-defined variables used in rules, in order to construct a logical representation of the perception. Such mapping is domain-specific and defined by users. When defining the mapping, however, it is beneficial to follow an ontology such that the mapping is reusable for other domains [44]. After mapping is done, the reasoner begins responding to various types of queries: *why*, *why not*, and even *what if* [26]. For those queries that ask why and why not, an implicit estimation of the current situation is logically entailed from rules and facts.

2.3. Scheduling in edge computing

Scheduling is a process of making decisions about which job to execute on which computing node. Jobs submitted to a scheduler are queued based on characteristics of the job—required amount and type of resource, their execution time, priority, and so on. Among many scheduling strategies, one of the traditional approaches is to schedule jobs as they come, namely, first come, first served. A resource manager then takes queued jobs from the scheduler and allocates resources to the jobs. While jobs are being executed, the resource manager is constantly monitoring them to know when to reclaim the allocated resource and move on to the next queued job.

Many aspects are considered when scheduling jobs in an edge computing environment. Since computing nodes are physically distributed and connected over the wireless network, data transmission between nodes is expensive and can be a bottleneck when bandwidth is limited. In order to reduce the network burden, latency-sensitive jobs (e.g., real-time video processing) are scheduled to the closest computing node that is capable of running the job [40,46]. In mobile IoT, the computation power of edge nodes is limited. Scheduling in that area takes job priority into account: higher-priority jobs are immediately processed by the node closest to the input, and lower-priority jobs are offloaded to other edge computing nodes [14]. Context awareness also helps find the best edge node to offload jobs. As mobile edge nodes freely enter and leave regions, node status as well as network latency and failure rate is considered for choosing the best node to offload jobs [12,19].

Energy consumption is another factor to consider when scheduling jobs in mobile IoT and edge computing. Edge computing nodes utilizing modern CPUs and microprocessors are able to adjust their computation performance using dynamic voltage frequency scaling [21,45] and task offloading to optimize energy consumption for the job [36] or for the device [30]. For some edge nodes that are powered by renewable energy sources, saving energy is more important than satisfying quality of service [43].

Future edge computing platforms will need more intelligence and autonomy at the edge to support data-driven and domain-aware applications. At the same time, such edge platform should provide a way of conceptualizing data and domain-specific requirements for running these applications on edge nodes. To support context-aware scheduling in edge computing, we propose a method that will run a context reasoner that constantly reasons about the environment and evaluates those requirements for running applications.

3. Goal-driven scheduling model

In this section we describe our goal-driven scheduling model. We call user applications a "plugin." We envision a scenario where multiple edge computing nodes are deployed in a city and make

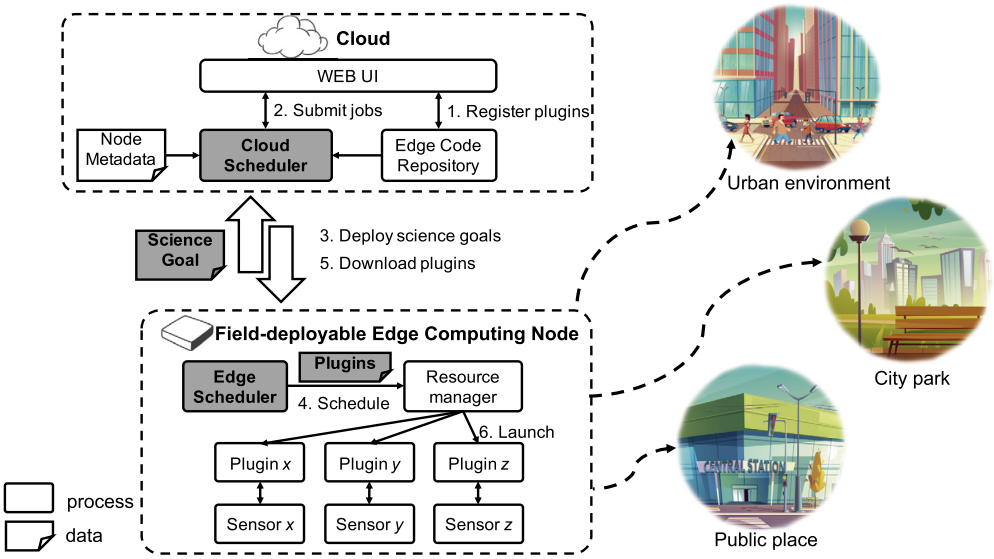


Fig. 1. Overview of the scheduling model. The grayed boxes are the main components of the proposed scheduling model. The numbered steps represent the process of scheduling. The graphics in the figure are from Freepik.com.

a wireless connection to the cloud. The deployed nodes are capable of running computations as well as interacting with sensor instruments, which measure the local environment in the city. Users (e.g., scientists) seek to run plugins on the nodes to produce meaningful information that helps answer their science questions. Fig. 1 illustrates an overview of the components utilized in the proposed scheduling model. The model offers two distinct schedulers to support isolation and autonomy within edge computing nodes: a cloud scheduler and an edge scheduler. Because the proposed model is goal-oriented, the main focus of the scheduling is to deliver jobs, that is, goals, by scheduling plugins. The following subsections describe the numbered steps shown in Fig. 1, along with the components associated with each step.

3.1. Plugin registration

A plugin is a packaged code block that takes inputs and produces results from the inputs. Because plugins are usually deployed in remote nodes, any human interface or visualization is not required from the plugins. Many plugins implement AI and pretrained ML models to process inputs. As discussed earlier, edge computing nodes may not be able to run multiples of such plugins at the same time because of memory limitations. In some cases, plugins can achieve variable levels of quality (e.g. accuracy, confidence) in their results based on the resources used. The ability to adjust the plugin based on availability of resources (e.g., memory, flops) enables plugins to run on many different computing platforms. To control the quality levels, plugins accept configurations consisting of one or more parameters that determine target performance. For example, a plugin may take a configuration specifying that it should take high-resolution images and process 10 images per second. In programming, the configuration can be expressed as "*resolution=high* and *framerate=10*." If the plugin is assigned the specified resources, it will produce an outcome at the desired quality.

Plugins that are run on our edge computing nodes need to be built and registered in our edge code repository (ECR), shown in Fig. 1. This allows the system operator to securely manage plugins and prevent any plugins that attempt to compromise the system. All plugins to be registered are first containerized by tools such as Docker [27]. Then, containerized plugins run on ECR-managed computing nodes to profile their performance. Let $\vec{c} =< c_1, c_2, ..., c_h >$ a vector of configured parameters of a plugin, $\vec{r} =< r_1, r_2, ..., r_i >$ a vector of required resources for a $\vec{c}$, and $\vec{m} =< m_1, m_2, ..., m_j >$ a vector of performance metrics corresponding to a combination of $\vec{c}$ and $\vec{r}$. A $\vec{m}$ can then be expressed as a function of $\vec{c}$ and $\vec{r}$.

$$\vec{m} = f(\vec{c}, \vec{r}) \tag{1}$$

An interpretation of the expression is that a plugin needs to be configured as $\vec{c}$ and assigned the resources $\vec{r}$ in order to perform at $\vec{m}$. A process of profiling is then measuring $\vec{r}$s and $\vec{m}$s for a given set of $\vec{c}$s—all possible combinations of configurable parameters of the plugin. At the end of the process, it generates a knob table K that encodes the profiling results as follows,

$$\begin{aligned} K &= \{k_1, k_2, ..., k_j\} \\ k_j &= \{\vec{m}_j, \vec{c}_h, \vec{r}_i\}, \end{aligned} \tag{2}$$

where the size of K equals the number of $\vec{m}$. Table 1 shows a performance table with arbitrary values as an example. Each row in the table represents a k_j. This table of premeasured performance will help the edge scheduler be informed, prior to scheduling, about the amount of resources required to run the plugin for performing at the desired quality of service.

3.2. Submitting jobs and generating science goals via a cloud scheduler

Scientists studying cities want to observe, capture, and analyze a variety of phenomena from the environment. To support these science activities, the proposed model allows them to submit job requests to the system. Fig. 2 shows the components of the cloud scheduler and their data flow. Because users do not have direct control of edge computing nodes for running their plugins, they need to specify a complete description of the job upon submission to the cloud scheduler. A job request specifies the following:

- **Plugins**: plugins that the user wants to run
- **Nodes**: target edge nodes on which the plugins run
- **Plugin Configurations**: sets of $mc = \{\vec{m}, \vec{c}\}$ selected from given K^p for plugin p that satisfy the user requirement
- **Science Rules**: rules expressed in first-order logic that state start and stop conditions of the plugins and that help evaluate the conditions

Table 1
Example of a performance table.

Performance metric $\vec{m}$		Configuration $\vec{c}$		Resource $\vec{r}$	
Framerate m_1	Quality m_2	Resolution c_1	Process rate c_2	CPU r_1	Memory r_2
10	25	high	10	100	1000
5	25	high	5	80	1000
10	13	low	10	70	500
5	13	low	5	40	500

Table 2
List of checks for validation of a job request. All checks must be passed in order for the job request to be accepted by the cloud scheduler.

Check	Description	Evaluation
Plugin	Plugins exist in ECR	$\forall p \in P_{job} : p \in P_{ECR}$
Plugin Performance	Plugins run at mc	$\forall p \in P_{job} : k_{mc} \in K^p$
Node	Nodes support the plugins	$\forall n \in N_{job}, \forall p \in P_{job} : n_{arch} \in p_{arch}$
Node Performance	Nodes run plugins at mc	$\forall n \in N_{job}, \forall p \in P_{job} : k^p_{mc} \leq k^n_{total}$
Science Rules	Rules are valid in their syntax	$\forall s \in S_{job} : syntax(s)$
Terms	Terms in Science rules follow the ontology	$\forall t \in T_{job} : t \in Ontology$
Hardware	Nodes support hardware requirement	$\forall n \in N_{job}, \forall p \in P_{job} : p_{hw} \in n_{hw}$

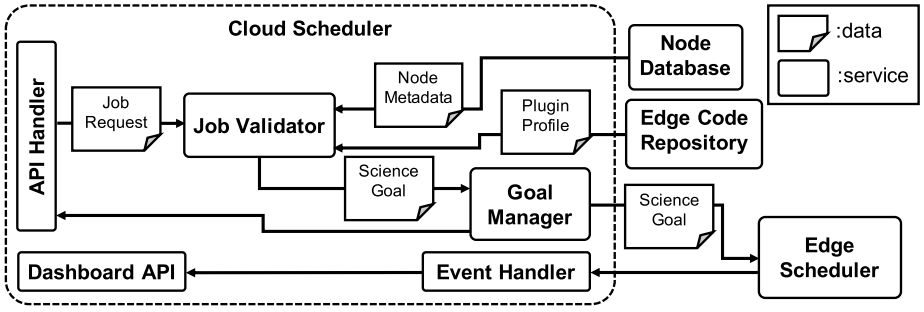


Fig. 2. Illustration of the cloud scheduler.

- **Success Criteria**: statements indicating when the job is considered as accomplished

The cloud scheduler assists users with all the information needed to complete job requests. ECR lists user plugins along with their performance table. The node database shown in Fig. 2 is managed by the system operator and stores metainformation about edge nodes deployed in the field. Such metainformation includes the node's geographical location, a list of equipped sensor instruments, and hardware specification of computing devices in the node. This helps users identify their target nodes. The node database is updated whenever any change in node metainformation is reported to the database (e.g., changes made within edge nodes are updated when the node becomes online). Keeping the database up to date is important for scheduling because some computing devices of edge nodes support dynamic voltage and frequency scaling to reduce energy consumption, which affects performance of the device. Nvidia's single-board computers also support different power modes that adjust the number of cores, frequency, and clock speed of CPU and GPU. The cloud scheduler also provides users a way to formulate user-specified science rules and success criteria using the system-provided ontology. These expressions will be constantly verified by the edge scheduler at runtime. The ontology consists of terms that are understandable by edge nodes and supports logical and numerical operations. Users also can define and use terms that are derived from user plugins as long as the terms support such operations and are logically sound.

Once users submit a job request, the cloud scheduler validates the job request to ensure that the job is executable by target nodes. This validation performs a list of checks, shown in Table 2, on the job request. Selected plugins are checked to see whether their container can run on the target nodes and whether hardware requirements of the plugins are satisfied by the nodes. The plugins are also checked to see whether the required resources to run the plugins at mc are supported. Let k^n_{total} be a knob element specifying the total amount of resource for node n with both $\vec{m}$ and $\vec{c}$ set to $\vec{0}$. We then examine $k^p_{mc} \leq k^n_{total}$ for each plugin p in order to verify that the plugin is configurable to $\vec{c} \in mc$ in terms of resources. Additionally, science rules described in the job request are checked: its syntax and terms used in the science rules are validated if they follow the ontology. Any job request that fails to pass those checks will not be accepted by the cloud scheduler. Once passed, information about the job request is combined with metadata of the target nodes and plugin information to generate a science goal. A science goal is a structured plain text that describes how and when plugins begin or stop running on devices of the target nodes. Note that for each plugin in a science goal the performance metrics that the plugin offers are pruned to get only the performance metrics supported by the devices of the target nodes. As a last step of the process, the generated science goal is staged into the goal manager in the cloud and later distributed to edge schedulers of the target nodes.

3.3. Running science goals through the edge scheduler

Edge schedulers running on each edge node are notified by the cloud scheduler of any update in the goal manager, such as creation, modification, and completion of science goals. The edge schedulers pull updated science goals from the cloud scheduler into a local pool to manage them. Fig. 3 illustrates the scheduling components as well as the service components utilized by the

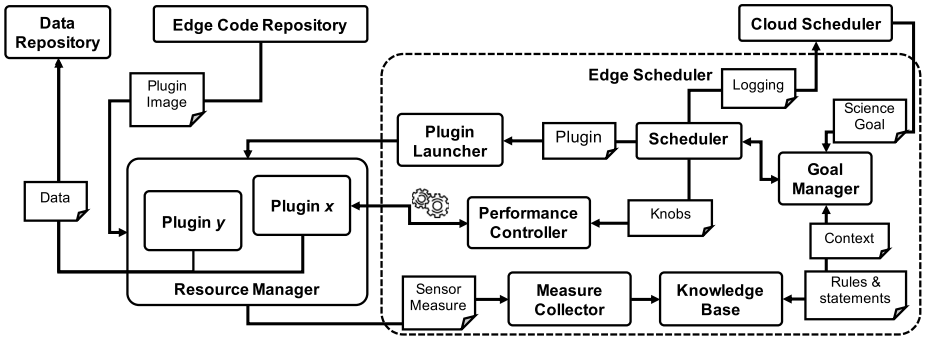


Fig. 3. Illustration of the edge scheduler.

edge scheduler. The pulled science goals are then activated: science rules from the science goals are registered in the knowledge base (KB), and their logic is constantly evaluated by using measurements from the environment. Additionally, the resource manager communicates with ECR to download plugins—program in a container—described in the science goal. When a state change in the science rules is triggered by local measures, the edge scheduler determines whether or not the corresponding plugin is scheduled via the plugin launcher. The plugin launcher is a simple interfacing component for talking to the resource manager. The resource manager then launches the container within a locally managed cluster. Launched plugins get resources allocated by the resource manager and start producing information meaningful to their science during the run. Produced information is reported to a data repository located in the cloud. Because the data pipeline is common across the system components, such information is often locally used as a measure to help the knowledge base infer more complex logic on science rules.

Launched plugins are initially configured to perform at their best—$max(k^p_{mc})$. With the nature of multitenancy, however, the system may not provide enough resources to support such a configuration. The resource manager has the role of allocating resources to plugins; however, it does not start a plugin if the requested resource in k^p_{mc} for the plugin p is greater than the currently available resource amount. The performance controller shown in Fig. 3 allows finer control on this resource allocation in order to maximize both the number of active plugins and the resource utilization. The performance table for each plugin encoded in its science goal provides a list of "knobs," in other words, configurations, to adjust the current configuration of the plugin.

All decisions made by the edge scheduler and corresponding events are monitored and logged. Logs are sent back to the cloud periodically for users and operators to understand how the node behaves. Because such events and decisions are logically explainable, analyzing logs gives the ability to backtrack why certain decisions were made and recognize any unwanted decisions made by the edge scheduler. This allows users later to update the science rules for their plugin to avoid any undesired decisions. On the other hand, the cloud scheduler is also constantly monitoring the logs to check whether any success criteria of a science goal are met. When a science goal is accomplished by the edge nodes, the cloud scheduler updates the science goal as accomplished such that the edge schedulers that have been serving the goal can drop it.

4. Context-aware goal-driven edge scheduling

This section details the logic of the edge scheduler that (de)selects plugins for scheduling by evaluating science rules at runtime. This allows the timely scheduling of plugins in response to events of interest.

4.1. Context-aware triggering from science rules

The proposed context-aware reasoning begins by sensing the environment. Sensor measures are a key-value pair that represents the state of something. For example, {'$env.temperature.outside$' : 37} expressed in JSON format represents 37 degrees of the current temperature *outside*, the place understood and agreed by the user. Units of the values are generally defined in the ontology shared between users and the system components; for this example the unit is understood to be Celsius. One also can extend the format to accommodate a timestamp indicating when the value was measured. Sensor measures are generated from both sensor instruments (via sensor plugins) and user plugins.

Science rules in a science goal are an IF-THEN statement where the IF part is a (propositional) statement with (symbolic) variables and the THEN part is a propositional variable that becomes valid if the IF part is logically true. Those rules are derived from combining scientific facts and desires toward accomplishing their science goal. One aspect of running plugins is to help fulfill missing information to unblock the process of reasoning on the logic. Arithmetic science rules are the science rules directly associated with sensor measures. For example, an arithmetic science rule "$env.temperature.outside > 35.6 \Rightarrow Hot(Outside)$" uses the outside temperature to infer $Hot(Outside)$. Note that the symbol $\Rightarrow$ indicates an implication. Logical science rules, the other type of science rules, follow the same structure without using sensor measures in their expression (e.g., $Hot(Outside) \Rightarrow Run(Airconditioner)$). In particular, this Run symbol becomes a trigger for the edge scheduler to run the plugin referred in the symbol. The work in this paper utilized forward-chaining algorithm [37] to infer any propositional variable, that is, a plugin, with particular triggering symbols—Run and $Stop$. The forward-chaining algorithm supports multilevel logical science rules as it explores through logical chains between science rules.

Algorithm 1 shows the process of inference on science rules upon receiving a sensor measure. The first for-loop in the algorithm determines which science goal is affected by the received sensor measure; that is, if the sensor measure makes any arithmetic science rule of a goal valid, then the goal is affected by the measure. With this arithmetic science rule as an example, a measure "$env.temperature.outside$" should exceed 35.6 to affect the science goal. The second double for-loop runs the forward-chaining algorithm on science rules of each affected science goal. If any plugin in those science goals is transitioned between Run to $Stop$, indicating that status of the plugin has changed, then the plugin name and its new state are outputted. The KB in the edge scheduler runs this algorithm to output a list of plugins subject to be rescheduled. The algorithm can be as complex as $O(n^2)$ in Big-O notation. In practice, however, n is small—we do not assume hundreds of science goals with hundreds of science rules running on

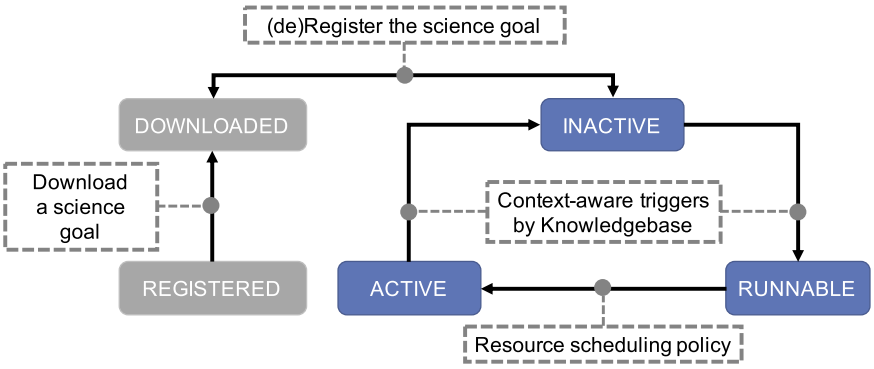


Fig. 4. Life cycle of a plugin. Solid boxes represent a state, and dashed boxes are an action that makes a transition between states. The states shown in a blue box indicate that they are being considered by the edge scheduler.

Algorithm 1: Evaluation of science rules upon receiving a sensor measure.

Input: A sensor measure s
Output: List of plugins P triggered and changed by the measure s
$sg_{triggered} \leftarrow \emptyset$; // a list of triggered science goals
forall $sr \leftarrow$ *arithmetic science rules in KB* **do**
  if $s \in sr_{LHS}$ **then** // sr_{LHS} is IF part
    if $sr_{LHS}(s) \Rightarrow sr_{RHS}$ **then** // sr_{RHS} is THEN part
      if $sg_{sr} \notin sg_{triggered}$ **then** // sg_{sr} is the science goal of sr
        $sg_{triggered}$.insert(sg_{sr});
      end
    end
  end
end
forall $sg \leftarrow sg_{triggered}$ **do**
  forall $p \leftarrow FC(sg_{rules})$ *on Run and Stop* **do** // FC is the forward-chaning algorithm
    if $p_{state} \neq p^{sg}_{state}$ **then** // p^{sg}_{state} is the current state of p in sg
      P.insert(p, p_{state}); // p_{state} indicates either *Run* or *Stop* triggered by s
    end
  end
end

an edge node at any given time. The forward-chaining algorithm itself has linear complexity in the number of science rules.

With the context-aware logic embedded inside the edge scheduler users can design their science rules to capture events of their interest and run their plugins accordingly. An arithmetic science rule that has a term "*sys.true*" in its left-hand side can be used to trigger running a plugin without needing to obtain any sensor measure since the term is always valid. This is useful to spin up the first set of plugins for a science goal. Then, the next-tier plugins can be triggered by sensor measures that the first set of plugins produce. Users can also define an event of their interest by combining science rules with appropriate sensor measures. For example, an arithmetic science rule "$env.tick.raingauge.avg > 3 \Rightarrow Raining(Now)$" defines when it is considered raining. Note that the constant 3 in the rule can be defined differently by users as their event of interest can be defined differently (e.g., one may consider 5 averaged ticks in a minute to indicate the event of raining). Logical science rules behave as a logic of how users want the system to schedule their plugins. Continuing the rain gauge example, a logical science rule "$Raining(Now) \Rightarrow Run(FloodDetector)$" takes the result "$Raining(Now)$" from the arithmetic rule and implements the step on what the user wants to run in case of raining. The inference on this logic may go even further by having another science rule "$Raining(Now) \wedge Daytime(Now) \Rightarrow Run(TrafficFlowEstimator)$"; the propositional variable $Daytime(Now)$ may come from an arithmetic science rule expressed as "$env.time.sunrise \leq env.time.now \leq env.time.sunset \Rightarrow Daytime(Now)$." Because science rules are described in first-order predicate logic, it is intuitive enough for a human to define them.

4.2. Scheduling plugins

Fig. 4 shows how plugins are downloaded, triggered, and activated. Because plugins registered in ECR can range from megabytes to gigabytes in size, the edge scheduler needs to download them prior to scheduling. Once the edge scheduler registers science goals, plugins associated with the science goals are set to the state "inactive." The context-awareness logic makes the plugins triggered by science rules transition to "runnable" state. The plugins with this state move into the computing cluster that the resource manager governs. They then become "active" when the resource manager allocates resources to them. Plugins in the cluster remain in the "runnable" state if no resource has been allocated to them. Plugins with the "active" state, currently running in the cluster, transition to "inactive" state when their *Stop* rule becomes valid in the context-awareness logic. For example, the flood detector plugin is dropped from the computing cluster if it is not currently raining, in other words, $NotRaining(Now) \Rightarrow Stop(FloodDetector)$. Note that using a negation $\neg$ is prohibited for an implication in forward chaining; the sentence must be either a definite or Horn clause [37]. Once a science goal is deregistered upon accomplishment by the cloud scheduler, all plugins associated with the science goal transition back to "downloaded" state, and all science rules of the science goal are dropped from the KB.

The transition from "runnable" to "active" state as shown in Fig. 4 is driven from different resource scheduling policies that involve resource allocation. Because plugins run to capture and analyze under a context, they need to be activated as soon as possible to capture and analyze the environment in time. Some plugins may conflict on resources including access to sensor instruments. However, it is out of scope in this paper to investigate which policy works the best for this goal-driven scheduling. Instead, we assume that the system relies on the default scheduling policy supported by the resource manager.

5. Case study and discussion

We designed and implemented the scheduling model to support various scientific studies that require context-aware in situ data processing at the edge. To validate the functionality and effectiveness of the proposed scheduling model, we present a case study that demonstrates the edge scheduler on a realistic use case—traffic flow analysis. Traffic flow analysis is used to understand how citizens contribute to daily traffic congestion and the

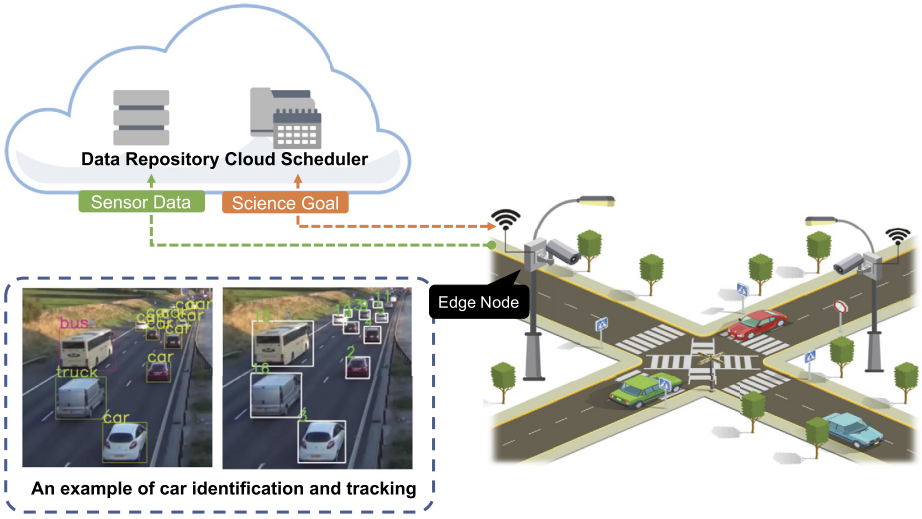


Fig. 5. Illustration of our case study. The edge computing nodes with a camera looking at an intersection run three user plugins to serve the science goal of traffic flow analysis. The illustration on the right is sourced from Freepik.com.

Table 3
Specification of the plugins used in the case study.

Plugin	Description	Output sensor measure
CarCounter	Detect number of cars in the scene	env.counter.car
TrafficFlowEstimator	Estimate averaged speed of cars in the scene	env.speed.car.avg
ImageSampler	Sample still images	*Not available*
Datetime	Report system time	sys.time.hour sys.time.sunrise.hour sys.time.sunset.hour

relationship between such congestion and the current transportation infrastructure in the city. Edge computing along with machine learning algorithms has greatly improved the quality of service and real-time processing capability for monitoring traffic state [5]. Data collection is still a difficult task, however, because it faces challenges from limited network bandwidth, image labeling [25], and preserving of privacy on collected data [20]. The platforms used for the many traffic studies are not general enough to be reused for other science, and they require significant effort to maintain the sensor network platform. We believe the proposed scheduling model with deployment of advanced edge devices could make a traffic-monitoring platform intelligent enough to accommodate multiple scientific studies served on the same platform.

In this case study we set up a scenario where a scientist wants to conduct a study on urban traffic flow. The city has multiple edge computing nodes already deployed near streets and city parks, and each node is equipped with an edge device and a camera looking at an intersection of city roads (see Fig. 5). The scientist as a user is willing to receive some data about the number of cars passing the intersections and their dynamics (their average speed when passing), as well as some sampled still images of the intersections when the traffic is slow. The user is provided two ML plugins and one non-ML plugin already registered in the edge code repository: *CarCounter*, *TrafficFlowEstimator*, and *ImageSampler*. Additionally, a 1-system plugin named *Datetime* is already running on the edge nodes. Table 3 shows a description of the plugins. Note that the *ImageSampler* plugin does not produce any sensor measure but does produce still images captured from the camera, which will then be transmitted and stored in the data repository.

Given the sensor measures from the plugins, the user now starts designing the desired behaviors of the system using the concept of science goal. Fig. 6 shows a state diagram of the desired logic. The user wants to run the *CarCounter* plugin only in the daytime and run *TrafficFlowEstimator* only in a dense traffic situation. Dense traffic is defined by a certain number of cars recognized from the scene at one time—five cars in our case study. The user also wants to run *ImageSampler* when the average speed of the traffic is slower than 48 km per hour. The plugins stop running whenever the counterpart of those conditions becomes logically true. Given the desired behaviors and sensor measures, a job submitted to the cloud scheduler would look as follows:

- **Plugin**: [CarCounter, TrafficFlowEstimator, ImageSampler]
- **Node**: [Node1 (at Main street), Node2 (at X street and Y avenue)]
- **Minimum Performance**: [CarCounter: (YoloV3, Resnet50), TrafficFlowEstimator: (KalmanV1)]
- **Science Rules**:
  - (A) $sys.time.sunrise.hour \leq sys.time.hour \leq sys.time.sunset.hour \Rightarrow Daytime(Now)$
  - (A) $sys.time.sunrise.hour > sys.time.hour \vee sys.time.hour > sys.time.sunset.hour \Rightarrow Nighttime(Now)$
  - (L) $Daytime(Now) \Rightarrow Run(CarCounter)$
  - (L) $Nighttime(Now) \Rightarrow Stop(CarCounter)$
  - (A) $env.count.car > 5 \Rightarrow Traffic(Dense)$
  - (A) $env.count.car \leq 5 \Rightarrow Traffic(Light)$
  - (L) $Traffic(Dense) \Rightarrow Run(TrafficFlowEstimator)$
  - (L) $Traffic(Light) \Rightarrow Stop(TrafficFlowEstimator)$
  - (A) $env.speed.car.avg < 48 \Rightarrow Traffic(Slow)$
  - (A) $env.speed.car.avg >= 48 \Rightarrow Traffic(Normal)$
  - (L) $Traffic(Slow) \wedge Run(TrafficFlowEstimator) \Rightarrow Run(ImageSampler)$
  - (L) $Traffic(Normal) \Rightarrow Stop(ImageSampler)$
  - (L) $Stop(TrafficFlowEstimator) \Rightarrow Stop(ImageSampler)$
- **Success Criteria**:
  - $sys.time.year > 2021$

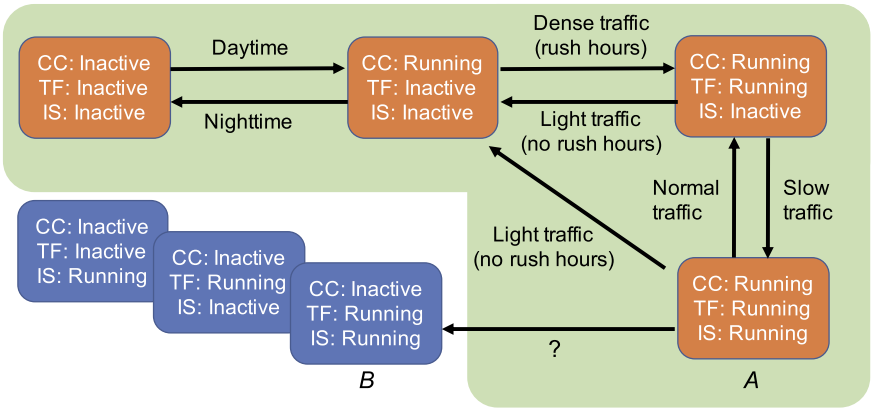


Fig. 6. States of the plugins in the case study and desired transitions between the states. CC, TF, and IS stand respectively for CarCounter, TrafficFlowEstimator, and ImageSampler. The transition from state A to B may happen unintentionally.

The science rules reflect the logic, shown in Fig. 6, for running and stopping the plugins. The dot-separated words in the science rules are variables defined in the ontology, and the variables match with the output sensor measures defined in Table 3. For example, $sys.time.sunrise.hour$ stores an integer number indicating the hour that the sun rises in the area, and $env.count.car$ represents the number of detected cars. One of the benefits of having these variables is that they are general enough to be applied to different contexts; for example, the same sunrise variable can be applied to multiple nodes in different locations without changing its meaning. (A) and (L) denote arithmetic and logical science rules, respectively. The success criteria describe that the job is accomplished if the year 2022 comes; that is, the job is valid until the end of 2021.

We note that the user defined only the transitions of interest, and thus the cloud scheduler might need to check whether the science goal transitions into undesired states. For example, day-long traffic congestion might occur in the city, which would keep both *TrafficFlowEstimator* and *ImageSampler* running because the number of detected cars would be greater than five for the whole day. According to the science rules, however, *CarCounter* stops running at night. Because *TrafficFlowEstimator* relies on the sensor measure from *CarCounter*, the plugin cannot be properly triggered at night. This will lead the system to the undesired transition from the state A to B in Fig. 6. This may be resolved simply by adding an additional science rule $Stop(CarCounter) \Rightarrow Stop(TrafficFlowEstimator)$, which stops *TrafficFlowEstimator* at night and also stops *ImageSampler* in a cascade. Adding such a science rule may require the user to monitor the run-time behavior of the edge scheduler on handling the user's science goal because the user may not foresee the transition when designing the science goal. The user is responsible for changing the science goal as needed. A model-checking technique [13] is one way of checking states of science goals prior to scheduling. The technique checks whether given specifications for a science goal are valid; if not, it provides a counterexample of any invalid specifications. In this example, a specification $\exists S_B$, where S_B is the state B, can be checked for finding any state change leading to the transition to the state B. A formal logic check for science goals is out of scope in this work and will be left for future research.

To demonstrate this case study, we set up a machine simulating a cloud computing environment that hosted the cloud scheduler and a virtual machine inside the host machine acting as an edge node named "Node1." Both cloud and edge schedulers were connected via the virtual network. Kubernetes was used as a resource manager on the edge node, launching and terminating plugins. The three user plugins were all made up in a way that they produce sensor measures as programmed. The *CarCounter* plugin outputted its sensor measure ranged randomly from 10 to 13 for hours in 7 to 10 and 16 to 19, and the measure went down between 0 and 3 for the rest of the hours, in order to simulate daily rush hours in a city. The *TrafficFlowEstimator* plugin behaved inversely because dense traffic usually means low speed of the traffic; it produced higher numbers when the car count measure was lower, and vice versa. After the job was submitted to the cloud scheduler via an API call, a science goal was generated and distributed to the "Node1" node. Then, the edge scheduler running inside the node loaded the science rules and started reasoning about the current context using sensor measures being reported. The *Datetime* plugin was specially designed to accelerate the simulation time such that it took 1 minute to drive 1 hour in the simulation; thus it took 24 minutes to simulate a day.

Fig. 7 illustrates the reported sensor measures and corresponding state changes of the 3 user plugins in the node during the run. The plugins were launched and terminated as intended by the science goal. The sensor measures in the figure show the implication on how the edge scheduler received those measures and used them to validate the science rules, which then triggered the plugins via the resource manager. In this work we used the default scheduling policy that allocated resources to whichever plugin came to the resource manager first. Because the container images of the plugins were downloaded in the edge node before the simulation began, it took only a few seconds for the resource manager to initialize the plugins for launching.

The simulation showed that the KB inside the edge scheduler was using the last reported sensor measures when inferring on the science rules. This approach may be undesirable when having time-sensitive science rules that imply the temporal aspect indirectly. For example, the science rule $env.count.car > 5 \Rightarrow Traffic(Dense)$ implies that the traffic is "now" dense (by checking whether the current car counter is greater than 5). If the taken sensor measure was produced hours ago, this rule may not give the correct context, depending on the actual intention by the user. One may suggest making the rule explicit for its temporal property as $TrafficDense(Now)$, but that may make the science goal much more complicated. Instead, using a time-series database and query languages that support such a temporal property may be a reasonable approach because science rules are derived from sensor measures. For example, a querying form[3] $env.count.car[1m]$ may give a cumulative number of cars recognized in the last minute.

The proposed scheduling model has left many research questions. Scheduling policy is one of the questions because it affects how science goals are delivered on edge nodes. This needs to be

[3] The form is referred from time-series database query languages such as Prometheus.

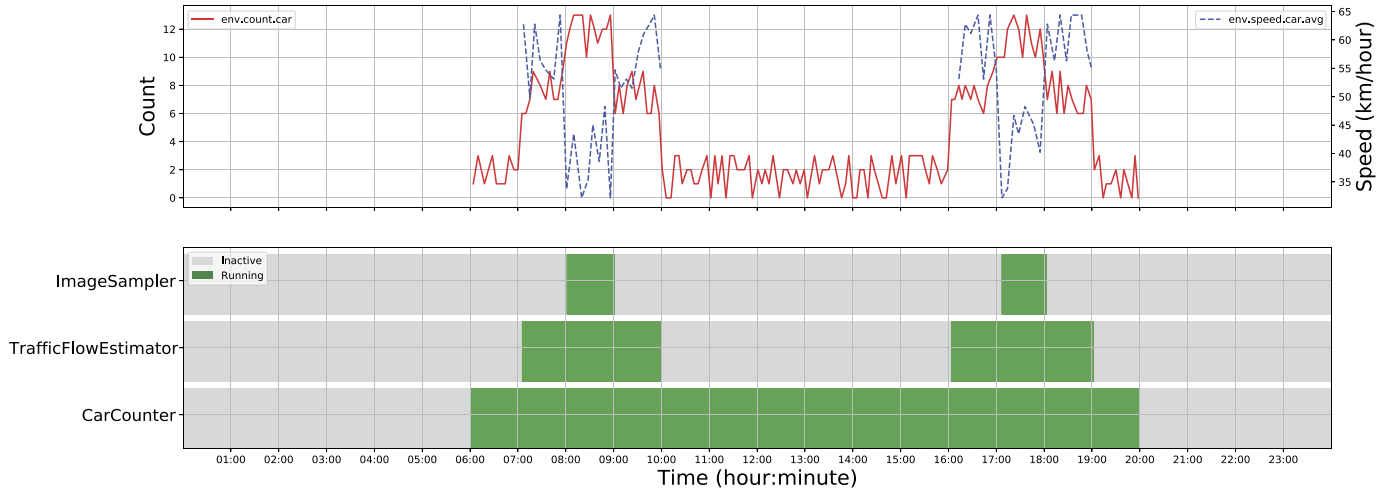


Fig. 7. Scheduling result for the case study. The edge scheduler ran the three user plugins to serve the science goal of traffic flow analysis. The graph shows a day's worth of scheduling.

investigated along with characteristics of user jobs. For example, the *ImageSampler* plugin needed to be running as soon as the system noticed a dense traffic. If the plugin was launched much later—even a few minutes since the traffic may change rather quickly—capturing an image of the dense traffic would not be possible. In addition to the scheduling policy problem, establishing priorities between conflicting science goals is another topic to study. We believe that in general one should not compare priorities between science goals because each goal represents a user's job. However, rarity of phenomena of interest may be a good factor to use because it may give higher priority to science goals seeking rarer events. Because the edge scheduler schedules plugins based on triggers made by science rules at run-time, it was assumed that any scheduled plugins will need to be run as soon as possible. But, having a tolerance time window—deadline—specified in science goal will make the scheduler more flexible on scheduling the plugin along with other plugins being scheduled. Lastly, the ontology expressing sensor measures of user plugins and its semantic representation should be universally agreed upon as the proposed model urges users to pick plugins for their science. The ontology may need to support arithmetic and temporal operations to support more expressions on sensor measures. This will allow more elaborate science rules that represent the intention of a science goal more precisely. An edge platform using the proposed scheduling model should form an ontology representing the default sensors such as meteorological sensors, cameras, and microphone that the platform provides and promote it as a foundation for edge applications, i.e. plugins, to add their data terms into the ontology. Users then should be able to use the terms in the ontology to design execution flow of plugins for their science goal. Nevertheless, we will need to start looking at more use cases requiring edge computing capability and understand their job execution flow and requirements.

6. Conclusion

Smart city infrastructures use distributed sensor systems and IoT. Edge computing provides the opportunity for customized and on-demand services by deploying user applications on distributed computing nodes. Interacting with edge computing systems often is difficult for users because of the complexity and dynamics of the system. Users would benefit from a system that enables them to describe what they need without extensive computer language skills. In this paper we proposed a two-layered scheduling model to ease the use of the system and isolate edge computing nodes from direct user interaction. Such isolation ensures that the schedulers behave logically and their result can be logically tracked. The concept of "science goal" fills the gap between user job description and system specific parameters, and the goal-oriented scheduling model handles the event-driven nature of user applications on edge nodes. Those applications are proactively sought by the edge scheduler whenever they need to run, based on the current context obtained from the local environment. The context-awareness used in this paper allows users to define states of interest and use the scheduler to capture them.

The case study showed a complete example inspired from existing studies in urban traffic. It indicated how one can specify a job for the science and how the job gets interpreted and served by the system. We admit that constructing science goals and applying them in real-case scenarios require some learning practice and in-depth understanding about input/output of sensors and plugins. For example, the user in the scenario described in Section 5 needed to be aware of how *CarCounter* and *TrafficFlowEstimator* behave and what resolution and view of the camera was needed to produce the car counts from the *CarCounter* plugin. This meta information about sensors and plugins should have been given to the user ahead of job submission. Additionally, if the user was not familiar to using predicate logic, there might be some learning practice involved on how to use the syntax to define science rules. By understanding the mechanism of the scheduling model, users should be able to describe their job more precisely, capturing the full system behavior at the edge.

CRediT authorship contribution statement

Yongho Kim: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft. **Seongha Park:** Data curation, Formal analysis, Methodology, Validation. **Sean Shahkarami:** Investigation, Methodology, Software, Writing – review & editing. **Rajesh Sankaran:** Conceptualization, Methodology, Validation, Writing – review & editing. **Nicola Ferrier:** Resources, Validation, Writing – original draft, Writing – review & editing. **Pete Beckman:** Conceptualization, Funding acquisition, Project administration, Resources, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This material is based upon work supported in part by U.S. Department of Energy, Office of Science, under contract DE-AC02-06CHI1357, design work was supported by National Science Foundation's Mid-Scale Research Infrastructure grant, NSF-OAC-1935984, development work was supported by U.S. Department of Energy, National Nuclear Security Administration Office of Defense Nuclear Nonproliferation (NA-22), under award AN20-DAWN-PD3Ja, and analysis work was supported by Exelon Corporation through CRADA T03-PH01-PT1397.

References

[1] E. Aguirre, S. Led, P. Lopez-Iturri, L. Azpilicueta, L. Serrano, F. Falcone, Implementation of context aware e-health environments based on social sensor networks, Sensors 16 (3) (2016) 310, https://doi.org/10.3390/s16030310.

[2] U. Alegre, J.C. Augusto, T. Clark, Engineering context-aware systems and applications: a survey, J. Syst. Softw. 117 (2016) 55–83, https://doi.org/10.1016/j.jss.2016.02.010.

[3] J.J. Alferes, F. Banti, A. Brogi, An event-condition-action logic programming language, in: D. Hutchison, T. Kanade, J. Kittler, J.M. Kleinberg, F. Mattern, J.C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M.Y. Vardi, G. Weikum, M. Fisher, W. van der Hoek, B. Konev, A. Lisitsa (Eds.), Logics in Artificial Intelligence, vol. 4160, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 29–42.

[4] J. Bailey, G. Papamarkos, A. Poulovassilis, P.T. Wood, An Event-Condition-Action Language for XML, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 223–248.

[5] J. Barthélemy, N. Verstaevel, H. Forehead, P. Perez, Edge-computing video analytics for real-time traffic monitoring in a smart city, Sensors 19 (9) (2019) 2048, https://doi.org/10.3390/s19092048.

[6] P. Beckman, R. Sankaran, C. Catlett, N. Ferrier, R. Jacob, M. Papka, Waggle: an open sensor platform for edge computing, in: 2016 IEEE SENSORS, IEEE, Orlando, FL, USA, 2016, pp. 1–3.

[7] P. Beckman, C. Catlett, I. Altintas, E. Kelly, S. Collis, Mid-Scale RI-1: Sage: A Software-Defined Sensor Network (NSF OAC 1935984) (Oct. 2019).

[8] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, Borg, Omega, and Kubernetes: lessons learned from three container-management systems over a decade, Queue 14 (1) (2016) 70–93, https://doi.org/10.1145/2898442.2898444.

[9] C. Catlett, P. Beckman, N. Ferrier, H. Nusbaum, M.E. Papka, M.G. Berman, R. Sankaran, Measuring cities with software-defined sensors, J. Soc. Comput. 1 (1) (2020) 14–27, https://doi.org/10.23919/JSC.2020.0003.

[10] C.E. Catlett, P.H. Beckman, R. Sankaran, K.K. Galvin, Array of things: a scientific research instrument in the public way: platform design and early lessons learned, in: Proceedings of the 2nd International Workshop on Science of Smart City Operations and Platforms Engineering, SCOPE '17, ACM Press, Pittsburgh, Pennsylvania, 2017, pp. 26–33.

[11] M. Chibuye, J. Phiri, A remote sensor network using android things and cloud computing for the food reserve agency in Zambia, Int. J. Adv. Comput. Sci. Appl. 8 (11) (2017), https://doi.org/10.14569/IJACSA.2017.081150.

[12] M. Chima Ogbuachi, C. Gore, A. Reale, P. Suskovics, B. Kovacs, Context-aware K8S scheduler for real time distributed 5G edge computing applications, in: 2019 International Conference on Software, Telecommunications and Computer Networks, SoftCOM, IEEE, Split, Croatia, 2019, pp. 1–6.

[13] E.M. Clarke, O. Grumberg, D. Kroening, D. Peled, H. Veith, Model Checking, 2nd edition, The Cyber-Physical Systems Series, The MIT Press, Cambridge, Massachusetts, 2018.

[14] J. Feng, Z. Liu, C. Wu, Y. Ji, Mobile edge computing for the internet of vehicles: offloading framework and job scheduling, IEEE Veh. Technol. Mag. 14 (1) (2019) 28–36, https://doi.org/10.1109/MVT.2018.2879647.

[15] A. Ferreira, E. Van Hensbergen, C. Adeniyi-Jones, E. Grimely-Evans, J. Minor, M. Nutter, L. Peña, K. Agarwal, J. Hermes, {SMARTER}: experiences with cloud native on the edge, in: 3rd {USENIX} Workshop on Hot Topics in Edge Computing, HotEdge 20, 2020.

[16] D. Gil, A. Ferrández, H. Mora-Mora, J. Peral, Internet of things: a review of surveys based on context aware intelligent services, Sensors 16 (7) (2016) 1069, https://doi.org/10.3390/s16071069.

[17] A. Giordano, G. Spezzano, A. Vinci, Smart agents and fog computing for smart city applications, in: E. Alba, F. Chicano, G. Luque (Eds.), Smart Cities, vol. 9704, Springer International Publishing, Cham, 2016, pp. 137–146.

[18] S.P. Gochhayat, P. Kaliyar, M. Conti, P. Tiwari, V. Prasath, D. Gupta, A. Khanna, LISA: lightweight context-aware IoT service architecture, J. Clean. Prod. 212 (2019) 1345–1356, https://doi.org/10.1016/j.jclepro.2018.12.096.

[19] B. Gu, Y. Chen, H. Liao, Z. Zhou, D. Zhang, A distributed and context-aware task assignment mechanism for collaborative mobile edge computing, Sensors 18 (8) (2018) 2423, https://doi.org/10.3390/s18082423.

[20] J. Handscombe, H.Q. Yu, Low-cost and data anonymised city traffic flow data collection to support intelligent traffic system, Sensors 19 (2) (2019) 347, https://doi.org/10.3390/s19020347.

[21] P. Huang, P. Kumar, G. Giannopoulou, L. Thiele, Energy efficient DVFS scheduling for mixed-criticality systems, in: Proceedings of the 14th International Conference on Embedded Software, EMSOFT '14, ACM Press, New Delhi, India, 2014, pp. 1–10.

[22] U. Jennehag, S. Forsstrom, F. Fiordigigli, Low delay video streaming on the internet of things using Raspberry Pi, Electronics 5 (4) (2016) 60, https://doi.org/10.3390/electronics5030060.

[23] C. Kamienski, M. Jentsch, M. Eisenhauer, J. Kiljander, E. Ferrera, P. Rosengren, J. Thestrup, E. Souto, W.S. Andrade, D. Sadok, Application development for the internet of things: a context-aware mixed criticality systems development platform, Comput. Commun. 104 (2017) 1–16, https://doi.org/10.1016/j.comcom.2016.09.014.

[24] N. Kukreja, A. Shilova, O. Beaumont, J. Huckelheim, N. Ferrier, P. Hovland, G. Gorman, Training on the edge: the why and the how, in: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, IEEE, Rio de Janeiro, Brazil, 2019, pp. 899–903.

[25] J. Lee, S. Roh, J. Shin, K. Sohn, Image-based learning to measure the space mean speed on a stretch of road without the need to tag images with labels, Sensors 19 (5) (2019) 1227, https://doi.org/10.3390/s19051227.

[26] B.Y. Lim, A.K. Dey, Toolkit to support intelligibility in context-aware applications, in: Proceedings of the 12th ACM International Conference on Ubiquitous Computing, ACM, Copenhagen, Denmark, 2010, pp. 13–22.

[27] D. Merkel, Docker: lightweight Linux containers for consistent development and deployment, Linux J. 2014 (239) (2014) 2.

[28] S. Mittal, A survey on optimized implementation of deep learning models on the NVIDIA Jetson platform, J. Syst. Archit. 97 (2019) 428–442, https://doi.org/10.1016/j.sysarc.2019.01.011.

[29] M. Naphade, S. Wang, D.C. Anastasiu, Z. Tang, M.-C. Chang, X. Yang, L. Zheng, A. Sharma, R. Chellappa, P. Chakraborty, The 4th AI city challenge, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2020, pp. 2665–2674.

[30] Z. Ning, J. Huang, X. Wang, J.J.P.C. Rodrigues, L. Guo, Mobile edge computing-enabled internet of vehicles: toward energy-efficient scheduling, IEEE Netw. 33 (5) (2019) 198–205, https://doi.org/10.1109/MNET.2019.1800309.

[31] Nvidia, Nvidia's Jetson Devices, https://www.nvidia.com/en-us/autonomous-machines/. (Accessed 15 February 2022).

[32] P. Pace, G. Aloi, R. Gravina, G. Caliciuri, G. Fortino, A. Liotta, An edge-based architecture to support efficient applications for healthcare industry 4.0, IEEE Trans. Ind. Inform. 15 (1) (2019) 481–489, https://doi.org/10.1109/TII.2018.2843169.

[33] C. Perera, A. Zaslavsky, P. Christen, M. Compton, D. Georgakopoulos, Context-aware sensor search, selection and ranking model for internet of things middleware, in: 2013 IEEE 14th International Conference on Mobile Data Management, IEEE, Milan, Italy, 2013, pp. 314–322.

[34] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos, Context aware computing for the internet of things: a survey, IEEE Commun. Surv. Tutor. 16 (1) (2014) 414–454, https://doi.org/10.1109/SURV.2013.042313.00197.

[35] H. Rahman, R. Rahmani, T. Kanter, Multi-modal context-aware reasoner (CAN) at the edge of IoT, Proc. Comput. Sci. 109 (2017) 335–342, https://doi.org/10.1016/j.procs.2017.05.360.

[36] N.B. Rizvandi, J. Taheri, A.Y. Zomaya, Y.C. Lee, Linear combinations of DVFS-enabled processor frequencies to modify the energy-aware scheduling algorithms, in: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE, Melbourne, Australia, 2010, pp. 388–397.

[37] S.J. Russell, P. Norvig, E. Davis, Artificial Intelligence: A Modern Approach, 3rd edition, Prentice Hall Series in Artificial Intelligence, Prentice Hall, Upper Saddle River, 2010.

[38] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini, S. Cretti, Foggy: a platform for workload orchestration in a fog computing environment, in: 2017 IEEE International Conference on Cloud Computing Technology and Science, CloudCom, IEEE, Hong Kong, 2017, pp. 231–234.

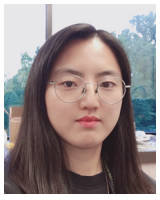
[39] B. Schilit, N. Adams, R. Want, Context-aware computing applications, in: 1994 First Workshop on Mobile Computing Systems and Applications, IEEE, Santa Cruz, California, USA, 1994, pp. 85–90.

[40] V. Scoca, A. Aral, I. Brandic, R. De Nicola, R.B. Uriarte, Scheduling latency-sensitive applications in edge computing, in: Proceedings of the 8th International Conference on Cloud Computing and Services Science, SCITEPRESS – Science and Technology Publications, Funchal, Madeira, Portugal, 2018, pp. 158–168.

[41] M.P. Silva, A. Sharma, M. Budhathoki, R. Jain, C.E. Catlett, Neighborhood scale heat mitigation strategies using array of things (AoT) data in Chicago, in: AGU Fall Meeting Abstracts 2018, 2018, PA21D-0986.

[42] H. Tawfik, A. Nagar, O. Anya, A context-driven approach to route planning, in: D. Hutchison, T. Kanade, J. Kittler, J.M. Kleinberg, F. Mattern, J.C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M.Y. Vardi, G. Weikum, M. Bubak, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science, ICCS 2008, vol. 5102, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 622–629.
[43] A. Toor, S. ul Islam, N. Sohail, A. Akhunzada, J. Boudjadar, H.A. Khattak, I.U. Din, J.J. Rodrigues, Energy and performance aware fog computing: a case of DVFS and green renewable energy, Future Gener. Comput. Syst. 101 (2019) 1112–1121, https://doi.org/10.1016/j.future.2019.07.010.
[44] X.H. Wang, D.Q. Zhang, T. Gu, H.K. Pung, Ontology based context modeling and reasoning using OWL, in: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, PERCOMW '04, IEEE Computer Society, 2004, p. 18.
[45] Y.H. Yassin, M. Jahre, P.G. Kjeldsberg, S. Aunet, F. Catthoor, Fast and accurate edge computing energy modeling and DVFS implementation in GEM5 using system call emulation mode, J. Signal Process. Syst. 93 (1) (2021) 33–48, https://doi.org/10.1007/s11265-020-01544-z.
[46] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, Q. Li, LAVEA: latency-aware video analytics on edge computing platform, in: Proceedings of the Second ACM/IEEE Symposium on Edge Computing, ACM, San Jose, California, 2017, pp. 1–13.
[47] X. Zhong, Y. Liang, Raspberry Pi: an effective vehicle in teaching the internet of things in computer science and engineering, Electronics 5 (4) (2016) 56, https://doi.org/10.3390/electronics5030056.

Yongho Kim is a Postdoctoral Appointee at Argonne National Laboratory with a joint appointment at the Northwestern University/Argonne Institute for Science and Engineering. His current research interests include job scheduling and orchestration in edge computing, autonomous systems, and artificial intelligence at the edge. Kim received his Ph.D. in Computer Information and Technology from Purdue University.



Seongha Park is a Postdoctoral Appointee at Argonne National Laboratory and the Northwestern University/Argonne Institute for Science and Engineering. Her current research interests include computer vision, machine learning, and artificial intelligence at the edge. Park received her Ph.D. in Computer Information and Technology from Purdue University.

Sean Shahkarami has been a software engineer on the Waggle project at Argonne National Lab since 2016. His interests are in providing users with great tools, building robust systems and continuously improving the software development process.

Rajesh Sankaran is a Staff Researcher at the Mathematics and Computer Science Division at Argonne National Laboratory, Lemont, Illinois, NAISE Fellow at the Northwestern University, Evanston, Illinois, and Senior CASE Fellow at the University of Chicago, Chicago, Illinois. His current research interests are in applications of embedded sensing and edge-computing in the Environmental, Urban, Infrastructure, and Weather/Climate researcher across the world. Sankaran received his Ph.D. degree in Electrical and Computer Engineering from Louisiana State University. He is a member of the IEEE and the ACM. Contact him at rajesh@anl.gov.

Nicola Ferrier is a Senior Computer Scientist at Argonne National Laboratory with a scientist-at-large appointment at University of Chicago's Consortium for Science and Engineering. Her current research interests include artificial intelligence at the edge, smart sensing, and intelligent systems. Her primary focus is on computer vision and audio-based sensing. She is the deputy director for the NSF SAGE project building a national infrastructure for software-defined sensors and artificial intelligence at the edge. Ferrier received her Ph.D. in computer science from Harvard University.

Pete Beckman is a Distinguished Fellow at Argonne National Laboratory and the Co-Director of the Northwestern University/Argonne Institute for Science and Engineering. His current research interests include artificial intelligence at the edge, smart sensing, distributed sensor networks, and extreme-scale operating systems. He is the PI for the NSF SAGE project building a national infrastructure for software-defined sensors and artificial intelligence at the edge. Beckman received his Ph.D. in computer science from Indiana University.