EasiCrawl: A Sleep-aware Schedule Method for Crawling IoT Sensors

Li Meng*†, Haiming Chen*, Cui Li*
*Institute of Computing Technology, Chinese Academy of Sciences
†University of Chinese Academy of Sciences, China
{limeng, chenhaiming, lcui}@ict.ac.cn

***Abstract*—How to search the Internet of things effectively to identify available and useful low power sensors is an important issue for constructing IoT applications. Crawling the content of low power IoT sensors is a fundamental step towards building a generic IoT search engine. However, crawling IoT sensors is very different from crawling the web. In IoT systems, many low-power sensors may sleep periodically, which may result in invalid crawls and unpredictable latency. Besides, the energy consumption of crawl accesses is non-negligible due to the limited battery life. As a consequence, the traditional crawling schedule strategy is not suitable for IoT search engine anymore. In this work, we first formulated the problem of crawling periodically sleeping sensors as a constrained schedule optimization problem, with the objective of minimized latency. Then, we proposed an a sleep-aware heuristic schedule, named EasiCrawl to get the near optimal expected latency. Next, we evaluate EasiCrawl with simulations which show advantages over other greedy-based methods. Finally a case study is performed with real world data from Xively, showing that EasiCrawl has lower crawling latency than traditional web-based search scheduling, and is an effective sleeping-aware schedule method for crawling IoT sensors.**

I. Introduction

A large number of sensors have been deployed to collect environmental information and connected to the Internet, which assists the human society to sense and control the physical world in an intelligent way. The extended Internet with sensors are known as the Internet of Things (IoT). With the booming of public access sensors in IoT, it is an inevitable trend to develop a generic search engine for IoT users and application builders to find different kinds of IoT sensors as required. A generic IoT search engine can be used in a variety of scenes to improve the possibility of sharing sensors with others. In this way, the redundancy of sensor deployment and facilitate construction of different IoT applications can be significantly reduced. More specifically, in smart city applications, public accessible sensors may be found in an ad hoc manner to provide predications relevant to traffic jam and queuing status. Smart home applications need searchable sensors and devices to detect human activities and provide real-time sensing of the indoor environment. In recent future, unmanned systems like automatic driving is likely to be heavily dependent on information gathered from nearby sensors or other IoT sensors, which also needs effective sensor search.

Existing works show that IoT search engine can be realized by traditional web search engines with the help of standardized protocols and semantic descriptions of sensors[1]. Here, the descriptions crawled by IoT search engines include both static contents and dynamic records generated by the sensor which is described by a dynamic description method. There are many research group are currently working on the standard of dynamic describing methods, such as CoRE[2] from IETF and SensorML[3] from the OGC. With the description methods, IoT search system can be designed with technicals in traditional web search. Fig.1 shows the architecture of such an IoT search system. It consists of events, sensors and a IoT search engine. In this search system, it is supposed that all sensors support IP protocol and are treated as web pages whose content is automatically updating. IoT crawlers crawl these descriptions files, and copy them into a local repository at the server side. The repository is an organized storage, with which the user queries about IoT sensors can be accomplished. It is assumed that the users only interest in the events monitored by the corresponding sensors, which arrives periodically or just randomly. With the help of semantic IoT method, IoT sensors first capture the event stream and then continuously store these events within their description files. IoT search engine regularly accesses these IoT descriptions and updates the related items in the repository accordingly.

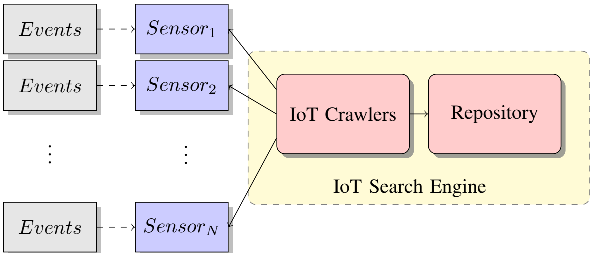

Fig. 1: Architecture of IoT Search System

However, even with dynamic description, the design of IoT search engine is not a trivial task. The main challenge is induced from the instinctive characteristics of IoT sensors. Firstly and the main concern of this paper, IoT sensors behave differently with their particular sleep mode, whereas the freshness of descriptions crawled is a critical measurement of search. If a sensor is sleeping, it can neither capture events nor be visited by IoT crawlers. If information are not collected in a right schedule, the freshness of query result will be impacted significantly. Secondly, most of IoT sensors are low-power device, where the energy consumption is a critical factor. An unnecessary high frequency of visiting a sensor may quickly exhaust sensor's energy. In this paper, we propose a new server-side scheduling method, named EasiCrawl, which can proceed sleep-aware search by taking these characteristics of IoT sensors into consideration.

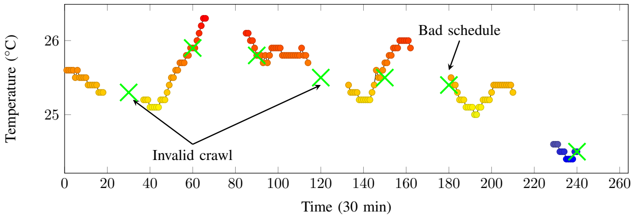

Fig. 2: Readings of a Temperature Sensor in a Smart Home Application

We use a clip of real world data to illustrate how the sensors' sleeping behaviors impact the crawl quality of IoT search system. Fig.2 is a data sequence from a temperature sensor in a smart home application, which acts as a sensor for air-conditioner control. The quick variations or break of threshold are taken as **interesting event**, which is logged into the description file immediately by the sensor. In addition, we suppose that when users are not at home, this sensor related control system will go into sleep mode for energy efficiency. We marked all these interesting events out in the figure as circles. Traditional scheduler uses a periodical crawl method to retrieve sensors' description files, as Fig.2 illustrates by green cross. In this case, this crawling schedule perform not so well, due to lacking awareness of sensors' sleep behavior. For example, it will make invalid accesses to sleeping sensor, like those showed at with pointer in Fig.2. Also, those missed readings will lead to a delay across the whole sensor' sleeping cycle. At time 210, the latency summation of events from previous working cycle may accumulated to a significantly high level.

The main contributions of this work are: (1) Although crawler scheduling is well studied, to the best of our knowledge, this paper is the first to discuss crawling of IoT consisting of sensors with sleep character and energy constraints. (2) An iterative optimization approach is proposed for crawling sensors with different sleep pattern. The approach includes a dynamic programming method as a subroutine to compute the optimal crawl plan for single sensors. A fast greedy method is also proposed as an complementary. (3) We evaluate EasiCrawl with simulation and real world data crawled from the IoT platform Xively[4].

The rest of the paper is organized as follows. We formalize the optimization problem in section III, then introduce EasiCrawl in section IV. Simulations are performed in section V and a case study is introduced in section VI.

II. Related Work

There are a lot of off-the-shelf technologies that can be used for IoT search systems. We have investigated these network configure protocols used in LAN to automate discover domestic devices, like UPnP[5] and Bonjour[6]. Although these device discovery mechanisms are widely used, they are actually not suitable for IoT applications because of the following reasons. Firstly, these auto-configuration protocols are designed for devices with sustained power supply, while IoT sensors usually face strict energy limitations. On the other hand, more densed searches for sensors in the sleep mode are not affordable. In addition, these protocols use static description method, which can not define the real-time state of IoT sensors. As a result, these protocols can not be widely used in IoT search system.

Besides the off-the-shelf technologies, a lot of new research works on IoT search are conducted to describe dynamic states of sensors. IoT search systems with different architectures have been proposed. Based on how the sensor information is gathered, these architectures can mainly be summarized into two categories, namely push-based and pull-based search systems respectively. Push-based systems try to collect and aggregate raw data for sensor content search. Systems like TinyDB[7] and IoT feeds[8] process structured command to directly query sensor data streams. In this kind of IoT systems, the redundancy of raw sensor data may causes problems in energy and data transitions. Alternatively, most IoT search systems use pull-based search architecture instead. A pull-based architecture uses semantic description, i.e. XML, to describe IoT sensors. Sensors update their description based on data collected, making their behavior very similar to that of a web page. This property makes searching IoT with a traditional web search engine possible. For example, SPITFIRE[9] uses crawlers and modern indexing methods, along with semantic IoT technology for sensor descriptions. Previous works of semantic IoT[10] and the propose of CoRE[2] are devoted to standardizing IoT sensor description languages and building a web-style IoT search system. The description files can be accessed by simply retrieving files in the ./well-know directory used in the CoRE framework. Although the communication over-head of pull-based architecture is relatively heavy, the scalability make this search architecture very promising.

We also investigated the crawl scheduling methods, which is a well studied topic in the field of information retrial. Previous works[11][12][13] analyzed how the crawler strategy affects the freshness of the repository of web pages maintained by web crawlers, which determines the query hit rate and whether this system can return a up-to-date result. This work are based on the assumptions that all the web pages can be accessed at anytime. However, in an IoT search system, the assumption does not hold any longer. The sleep character of sensors may significantly impact the latency of the crawl of information. In our work, the sleep character and energy constraints are both taken into consideration. To the best of our knowledge, no research has yet been done to study the crawl schedule problem of IoT search system.

III. Formulation

In this section, we will formulate the crawl schedule problem, and introduce all the symbols used in the following paper. The assumptions followed are set to simplify sensors' working model. Firstly, events that users care arrive at each IoT sensor in a Poisson manner, where the time interval between any two events are in exponential distribution. Secondly, when sensors are in sleep mode, they can not be accessed by crawlers, so no new events can be captured or stored at that time. Thirdly, sensors' sleep plan is previously known. At last, the length of work or sleep cycle does not need to be equal.

EasiCrawl is invoked periodically with a cycle of T. T is called the **time range** of EasiCrawl in the following paper. We assume there are totally N sensors that need to be crawled. A discretization method is applied for model simplicity, where a finite numbers of time points can be used as candidates for crawling. We set up a discrete time table with a **discretization step length** ϵ, each crawl must be preformed at one of these candidate time points. Our goal is to find an appropriate **schedule plan** Φ so that the expected latency of all N IoT sensors is minimized. The schedule plan consists of two parts. The first part is the **arrangement of crawls** $\hat{n}$ for each sensor i, denoted as $\hat{n} = n_1, n_2, \ldots, n_N$. We use the symbol $\Phi(\hat{n})$ to indicate the schedule plan with arrangement $\hat{n}$ for short. The second part is the specific **crawl plan** for each sensors, which is a list of time points for crawling. We denote this list as ϕ_i for convenience. In order to model the sleep pattern of each sensor, we use a list of time slots to represent the working duty cycle of sensors, which is called **sleep plan** for brief. The discretization process takes the sleep plan of each sensor as input, and generate discretized time points for each sensor, where time points of sleeping period of sensors are not included. Sometimes the importance of different sensors are distinct, so for each sensor i, we define a factor ω_i as the expectation weight. Here we use the notation $E_L(n_i)$ as the minimal **expected latency** that can be gained by sensor i with totally n_i chances to crawl. The final object function $E_L(\Phi(\hat{n}))$ is the sum of expected latency of all sensors which is denoted as $\sum_{i=1}^{N} \omega_i E_L(n_i)$.

In addition, this problem follows two constraints. As previously described, the crawl server has constrained server load and these IoT sensors' energy budgeted to be accessed are limited. Hence, for the server-side, we set a constraint that the crawlers from the server can at most commit θ crawls during the entire time range $[0, T]$. During the time range T, the access times n_i for sensor i is limited to be less than γ_i times. With the object function and constraints analyzed above, we get the following formula. The final output of EasiCrawl is crawl plan Φ, which contains the optimal crawl arrangement $\hat{n}$ and crawl plan ϕ_i for each sensor i.

$$\begin{array}{ll} \min_{\Phi} & E_L(\Phi(\hat{n})) \\ \text{s.t.} & \sum_{i=1}^{N} n_i \leq \theta \\ & n_i \leq \gamma_i \\ & i = 1, \ldots, N \end{array} \qquad (1)$$

The generalized formulation (1) does not require specific sensor model or sleep plan. Different sensor models can be used for this formulation, as long as the relationship between n_i and $E_L(n_i)$ can be determined. Later we will show that, this problem can be solved in polynomial time, given the fact $E_L(n_i)$ of each sensor are convex functions.

IV. Crawling IoT Sensors with EasiCrawl

We first give out the framework of EasiCrawl, which iteratively improves our solution to achieve a latency optimal scheduling plan. Then technical details of subroutines of EasiCrawl are introduced, including the schedule method for periodic or none-periodic sleep sensors. At last, we discuss the strategy for sensors whose sleep plan is previously unknown.

A. Method Framework

The main idea of EasiCrawl is to change crawl arrangements $\hat{n}$ according to the gradient of $E_L(\Phi(\hat{n}))$. This arrangement is done in an iterative manner. Firstly, an original arrangement is made in proportion to an estimated latency, which is the value of sensors' importance ω multiple length of working time. Then, EasiCrawl starts to improve the object function $E_L(\Phi(\hat{n}))$ step by step. In each iteration, EasiCrawl searches for a best change in arrangements which can deduce the expected latency most quickly. The step length for changes is denoted as ϵ. With these analyses, the minimization problem can be transformed into the following two sub-problems. First, how to rearranging crawl numbers $\hat{n}$ for each sensor under two constraints? Second, what's the best crawling plan for each sensor? Once n_i is chosen, the optimal crawl strategy for a single sensor is determined. To iteratively improve the global result, we use an approach to change a pair of n at the same time. We start to search sensors for a pair i, j, where i, j are both arrangements. If $n_i \leftarrow n_i - \epsilon$ and $n_j \leftarrow n_j + \epsilon$ will cause a largest improvement to the object function compared to the other pairs, then changes of i and j are conformed. This process will be continued until none improvement can be made.

Algorithm 1 EasiCrawl Method Framework

```
Input: $\mathbf{T}$, $N$, $\theta$, $\epsilon$
Output: $\phi$
 1: $\hat{n} \leftarrow EvenlyDivide(\theta), LastExpect \leftarrow 0$
 2: while $E_L(\Phi(\hat{n})) - LastExpect > \epsilon$ do
 3:     $BestChange \leftarrow 0, i' \leftarrow 0, j' \leftarrow 0$
 4:     for each sensor pair $i, j \in$ sensors do
 5:         $\hat{n}' \leftarrow n_\epsilon, ..., n_i + 1, ..., n_j - \epsilon, ..., n_N$
 6:         $c \leftarrow E_L(\Phi(\hat{n})) - E_L(\Phi(\hat{n'}))$
 7:         if $c > BestChange$ then
 8:             $BestChange \leftarrow c, i' \leftarrow i, j' \leftarrow j$
 9:         end if
10:     end for
11:     $\hat{n} \leftarrow n_1, ..., n_{i'} + \epsilon, ..., n_{j'} - \epsilon, ..., n_N$
12: end while
13: return $\Phi$
```

The above pseudo code 1 explains the workflow of EasiCrawl. Some input parameters are omitted for brief, namely weights of each sensor and discretization step length. In step 1, method $EvenlyDivide$ initialize the crawl arrangements $\hat{n}$. Schedule plan Φ is iteratively improved during steps 2 to 11. EasiCrawl search for the best change from Step 3 to 10, and then replaced Φ with a better one.

Here, we discuss the optimality of $E_L(\Phi(\hat{n}))$. Indeed, the correctness of EasiCrawl bases on the fact that the expected latency function for each sensor $E_L(n_i)$ are convex. If $E_L(n_i)$ can be proved convex, the convexity of the object function $E_L(\Phi(\hat{n}))$, which is the sum of $E_L(n_i)$, can be proved. Notice that if $E_L(n_i)$ and the constrains are convex, **problem** (1) **is a convex optimization problem**, in which iterative methods always lead to optimal solutions. For the non-sleep and some other periodic sleeping situation, $E_L(n_i)$ is convex, which is proved in following paper. For the sensors with different length of working cycles, there is no theory guarantee of convexity. However we still trend to believe the convexity hold for most of this situations, because of the intuitions that, with more crawls, the increase of new ones will result in less contributions per each one, which is actually convexity. A supportive result is showed in the evaluation section V about simulation with different total crawl limits, when sum of crawls are much

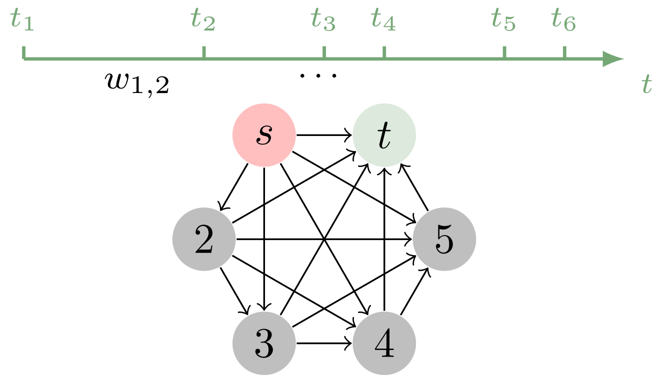

Fig. 3: Example of Non-periodic Hibernate Problem Transformation

smaller than that of work cycles. In addition, the computation of $E_h[G(t)]$ and $E_f(t_s, t)$, which are both expected latency of a single senor, is discussed in the appendix.

In the following paper, we start to design subroutines for the computation of $E_L(n_i)$. At first, A dynamic programming method is put forward for an optimal solution, but the time complexity is somewhat unacceptable for large datasets. So, we proposed a greedy method as a trade-off for time efficiency. All these two methods are used to compute the crawl plan ϕ of a single sensor with sleep behavior, which leads to a optimal expected latency $E_L(n_i)$.

B. Dynamic Programming Method for Optimal Crawl

With the previous discretization process, we have a chance to enumerate all the possible crawl plans, both for the continuously working sensors and ones with sleep behavior. The following dynamic programming(DP) method take advantage of memorized search, which caches the solution of sub-problem to accelerate the enumeration process.

We use **timeline** to indicate the discretized time table for a single sensor, which consists of the working cycle and the sleeping cycle. The crawl plan can actually be formed as a route planning problem given the following transformation. Here a directed acyclic graph(DAG) model $G(E, V)$ is built to formalize this problem. E are the edges of the graph, and V are the vertexes. Every points standing for the working cycle as graph vertex is written as v_i. Suppose there are m vertexes and T time points in the timeline. Corresponding, there are exactly T vertexes in this graph, from number 1 to T. We define n as the count of crawls used, which indicates there are $n-1$ vertex allowed to pass by, because the last vertex must be chosen. For each pair of time points t_i and t_j, if $t_i < t_j$, a directed edge $e_{i,j}$ from v_i to v_j, is added to the graph. A weight value w_{ij} is set to $e_{i,j}$, whose value is $E_f(t_j, t_i)$. $E_f(t_j, t_i)$ is actually the expected latency gain of a crawl at time t_j whose previous crawl is at t_i. The problem of minimized-latency for a single sensor is now transformed into that, how to find a path $e_{1,i}, \ldots, e_{j,T}$ from v_1 to v_m, which take exactly n steps, that minimum the sum of all the weights alongside. This transform process can be illustrated by Fig.3.

With an acceptable scale of time range T and crawl time n, this problem can be solved by DP shown in Algorithm 2. Solution state with a vertex v_i and the available crawl time n' left is an sub-problem. The optimal expectation can be computed by adding weight $w_{i,j}$ with all the expectation known in sub-problems. The recursive relationship can be summarized with expression (2).

$$f(v_i, n') = \begin{cases} \min_{k=1}^{i-1} & \{f(v_{i-k}, n'-1) + w_{i,i-k}\} \\ & \text{if } i \in [1, m-1] \\ 0 & \text{if } n' = 0 \text{ and } i = 1 \\ \infty & \text{otherwise} \end{cases} \tag{2}$$

$f(v_i, n')$ is the sum of weight of a path that start from v_1 and end with v_i, with cost exactly n steps. In each step, we go over the timeline for n^2 times. So the time complexity of iterating process is $O(m^2 n)$, in which nm sub-problem are computed and each computation will cost m operations. The backtrack method uses $p(v_i, n)$ to get the optimal strategy, which takes at most n steps. As a result, the overall time complexity is still $O(m^2 n)$.

Algorithm 2 DP Method for Optimal Crawl

```
Input: $G(V, E)$, $n$
Output: $t_1, \ldots, t_n$
$f(v_i, n') \leftarrow \infty$, $p(v_i, n') \leftarrow 0$
for $n' = 1$ to $n - 1$ do
    for $i = 2$ to $m$ do
        $f(v_i, n') \leftarrow \min_{k=1}^{i-1}\{f(v_{i-k}, n'-1) + w_{i,i-k}\}$
        $p(v_i, n') \leftarrow$ minimum $v_{i-k}$
    end for
end for
$t_n \leftarrow T$, $t_i \leftarrow$back trace from $p(v_i, n)$ for $t_i$
return $t_1, \ldots, t_n$
```

C. Greedy Method for Time efficiency

Previous DP method is a generic way to compute the optimal crawl plan. Although it returns an optimal solution, the computation complexity of this DP method is relatively high. Most of the time, there is no need to compute an optimal crawling plan, especially when the computation resource is scarce. In the following paper, we show that the schedule procedure of a single sensor can be boosted using greedy method in some special case.

The optimal schedule method for a single sensor with n crawls can actually be formalized by an optimization problem 3. Here $E_f(t_i, t_{i+1})$ are the expectation function for time period $[t_i, t_{i+1}]$, given the fact $E_f(0, t)$ is convex.

$$\begin{array}{ll} \min_{\mathbf{t}} & \sum_{i=1}^{n} E_f(t_i, t_i + 1) \\ \text{s.t.} & t_i < t_{i+1} \\ & i = 1, \ldots, n-1 \end{array} \tag{3}$$

If both $E_f(c, t_i)$ and $E_f(t_i, c)$ are convex, where c is a constant, then this problem can be solved with greedy method. This assumption holds when the sensor is working continuously or when the duty cycle of sensor are the same. In the follow analysis, we looked into these two situation, and explained why greedy method can not achieve the optimal solution in the general case.

In the first case, we assume the sensor works continuously. An intuition to solve this problem is to evenly distributed the crawling action across the timeline. We prove this method actually leads to an optimal value.

Lemma 1: The optimal crawl plan of an continuously working sensor is to scatter the crawl action over the timeline $t \in [0, T]$ as evenly as possible, where T is a constant.

Proof: This proposition can be proved by the convexity of $E_h[G(t)]$ [14] by induction. Here we donate $E_h[G(t)]$ as $f(t)$ for convenience. We start from the situation where only two crawling chance are allowed. As a result, the expected latency gain during $[0, T]$ is marked as $F(0, T)$. $F(0, T)$ is divided into $f(t)$ and $f(T - t)$ with a time point $t \in [0, T]$. With the convexity of $f(t)$, we have $2f(T/2) < f(T/2-\epsilon)+f(T/2+\epsilon)$ for $\forall \epsilon > 0$. By plugging in $F(T)$, we get $F(T/2) < F(T/2+\epsilon)$ for $\forall \epsilon > 0$. This indicates $T/2$ is the minimum point of $F(t)$, which means the evenly crawl is the best strategy for the situation with only 2 crawls. Now suppose there are n crawling actions, among which first $n-1$ actions evenly divide the timeline. For last two pieces of crawl intervals divided by the n-th crawl, we can choose the evenly one as a new strategy, which leads to a larger $f(t)$. ■

With Lemma 1, we can get the minimal gain of latency for different crawl time n. If the sensor works without any sleep, the relationship between crawl time and $E[\phi(n)]$ can be computed with $E_h[G(n)] = nE[G(\frac{T}{n})]$.

Then we illustrated that this intuition holds for the situation when the sensor works periodically, with some constraints. Some points in the timeline can not be accessed, so the expectation function in this situation is actually piecewise. Here we use T^w and T^s as the length of working and sleep cycle respectively. If sensor works periodically, the intact cycle is marked as T^c, where $T^c = T^w + T^s$. The intuition to solve problem is to arrange the crawls at the end of a work cycle. We show this intuition is indeed an optimal strategy when the crawl interval is an integral multiple of duty cycle T^c.

Lemma 2: For periodically sleeping sensors, suppose the sensor's duty cycles has exactly an integral multiple of numbers of crawls, indexed from 1 to m. The optimal strategy is to make n times of crawl at the end of a working cycle, whose index i satisfies $i \bmod (m/n) = 0$.

Proof: We still use induction for this proof. For the most simple case, when number of crawls $n = 2$, the first crawl locates at time t' and second one at the end. This strategy is optimal due to any increase of t' will cause a gain in the object function by $E_f(0, t' + T^w) - E_f(0, t')$, and any decrease of t' will raise the object function due to the evenly divide. For the case when $n > 2$, the problem can be constructed by add t'/T^c cycles, where if Lemma 2 holds for n, it must holds for $n + 1$. ■

Based on previous analysis, we put forward a greedy method, which tries to divide the crawls as evenly as possible. The result can be computed in one pass, so the complexity of this approach is $O(n)$, where n is the size of crawl times.

At last, we discuss the most general situation. When the length of working cycles are not the same, or the number of duty cycle is not divisible by m, or the length of duty cycles are different, the previous intuitions no longer holds. In these situations, the greedy method for searching crawl plan can not lead to a optimal value. It's because the expectation function with two variables $E_f(t_s, t)$ is not convex when t is fixed. As a consequence, local minimum may exists in the result, which makes the achieved by greedy method possibly not the optimal one. Even though, we add this non-optimal method in our final implementation, because of its good performance in real world datesets.

Algorithm 3 Fast Greedy Crawl Method

```
Input: $n$, $T$, $T^w$, $T^s$
Output: $t_1, \ldots, t_n$
$T^c \leftarrow T^w + T^s$
$m \leftarrow T/(T^c n)$, $r \leftarrow (T/T^c) \bmod n$
for $i = 1$ to $m$ do
    $t_{m-i+1} \leftarrow T - T^c i + T^w$
    if $i \neq 0 \wedge r \neq 0$ then
        $t_{m-i-1} \leftarrow t_{m-i+1} - T^c$
        $r \leftarrow r - 1$
    end if
end for
return $t_1, \ldots, t_n$
```

V. Simulation and Evaluation

In this section, we will test the performance of EasiCrawl under different settings and input parameters.

A. Settings

Expected latency which are previously used as a optimization objective here is taken as a metric of measurement. Performance of EasiCrawl is measured with different settings of following parameters, including sensor scales N, total crawl limit θ, discretization step length ϵ, and time range T of scheduling. We use a randomized method to generate the sleep plan for each sensor, and we used a Poisson process generator to generate the simulation events. For comparison, we implement a random schedule method, called RandomCrawl. RandomCrawl use the same method as EasiCrawl to initialize the arrangement of crawls to different sensors. Then, for each sensor, the crawl plan are set randomly, the last timestamp are chosen to ensure all the events are captured. All these simulations are implemented on a PC with MATLAB.

B. Results

We first study the impact of **sensor scale** N. We mainly test the latency of dynamic programming schedule method of single sensor in this simulation. All the sleep plan of sensors are previously known in advance. The total crawl limit θ is set as the multiple of sensor scale, varies from 10 to 200. The Poisson parameter λ of each sensor is randomly chosen from 0 to 1. The schedule time range T is set as 200 time units. In each improvement iteration, the crawl time arrangement n_i for different sensors are changed by the step length of 1. For each sensor scale, we repeated the simulation for 5 times, and compute the average, minimum and maximum value of expected latency. Simulation result is plotted in Fig. 4. In the result, EasiCrawl surpass the RandomCrawl by 2 times

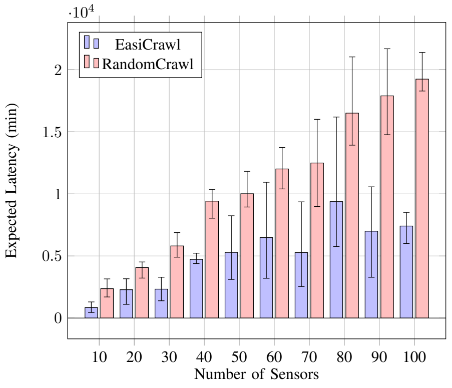

Fig. 4: Expected latency With Different Sensor Scales

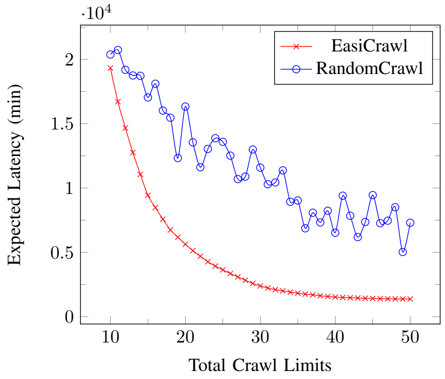

Fig. 5: Expected latency With Different Crawls

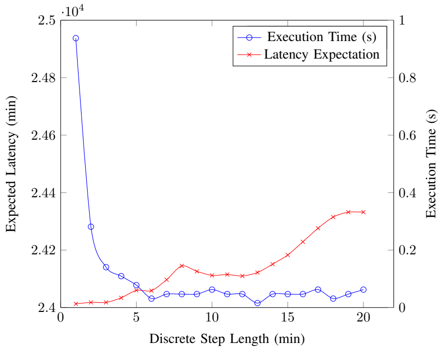

Fig. 6: Expected latency With Different Discretization Step

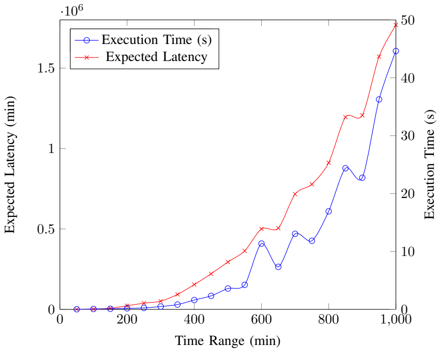

Fig. 7: Expected latency With Different Total Time Range

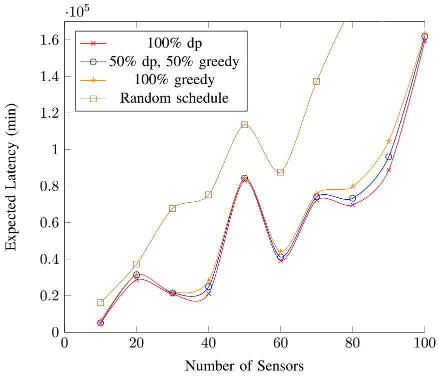

Fig. 8: Expected latency With Different Plan Methods

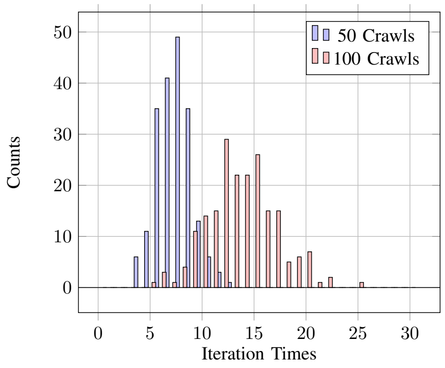

Fig. 9: Converge Speed under Different Crawl Times

in reducing the expected latency, even take error bar into consideration. This is mainly because of the theoretical optimal property of EasiCrawl when given a sequence of discretized time nodes.

The total number of crawls θ reveals the limit of the IoT search engine. In this test, EasiCrawl worked on sensor scale of 10, crawling time range of 200, and discrete step length ϵ of 2, where minute is the time unit. We use RandomCrawl as a reference, and test 40 different crawl limits on EasiCrawl and RandomCrawl, ranging from 10 to 50, which show a result as Fig.5. The result first show the advantages of EasiCrawl against RandomCrawl. More important, we notice there's a submodular relationship between total crawls and the latency reduction, which means the gain of server side crawl ability pay less when limits grow larger. This is important for us to find a proper trade-off between server cost and schedule efficiency.

Time table of sleep plans should be discretized with a **discretization step length** ϵ before starting scheduling. The choice of discretization step length is a trade off between the computation time and final accuracy. We use a simulation with 10 sensors and a time range T of 500, to test the relationship between time complexity and expectation accuracy. The total crawl count θ is set to 100 times. We carried on 20 groups of tests, where the discretization step length ϵ ranges from 1 to 20. Fig.6 shows a positive result to our analysis of the dynamic programming crawl method used in EasiCrawl. While expected latency increase about 400 units, the computation time decreased from 1 seconds to below 0.1 seconds. This result fits the theoretical analysis, where for scheduling each sensor, the time complexity is $O(m^2n)$, n is the crawl limit and m is the number of time points. When ϵ is relatively small, the number m of total time candidates which can be used to crawl is large, leading to a higher computation cost.

Time range T directly affects the executing time and running frequency of EasiCrawl. We use the same experiment settings as that of discretization test. 10 simulated sensors are used, the total crawl sum θ is 30. For time range T varies from 50 to 1000, a new time table is generated in each iteration. T has a similar effect as the discretization step ϵ. The computation time showed in Fig.7 is very close to the analysis result, which quadratic relationship between T and running time can be inferred. The expectation shares a similar shape because the distance between the distribution and the end time t can also be approximated as a quadratic function.

Two totally different **plan methods**, the DP method and the greedy one, are used in EasiCrawl. In the previous evaluations, only the DP methods are used for finding the best schedule for each sensors. In this test, both of the methods are tested. Tests are performed for 10 times with different sensor scale varies from 10 to 100. 4 different group of sensors are used for each test. Sensors in the first group are all scheduled with DP method. In the second group, half of the sensors are using DP, while the other half use greedy based method. The third group uses pure greedy based method. We also use the RandomCrawl method as a reference, which is the fourth group. Result are showed in Fig.8. As our expect, DP method performs the best in latency reduction. With the share of greedy method grows, the expected latency becomes larger. When all the sensors use greedy method for scheduling, the expected latency is the worst, but still much better than that of RandomCrawl.

The **speed of convergence** directly affects the efficiency of the total computation cost. Here we set up a test with two different total crawl times θ, to verify the iteratively optimization method. The time range T here is chose to be 400 time units. We separately perform this test, first use a total crawl limit θ of 50, then 100. The sleep plan of sensors and other parameters are the same. The total iterations used before reaching convergence is collected. Fig.9 is the final statistics, where the iteration steps for different experiment setting is close to Gaussian distribution. Apparently, when the number of total crawls grows bigger, a larger mean value is expected.

VI. Case Study With Xively

Here we will show the performance of EasiCrawl in reducing the crawling latency. We are using real world dataset gathered from Xively. The latency of each event equals the time of the following crawl minus the time of its occurrence.

Xively is a platform which integrates IoT sensors. With thousands of sensors that can be simply accessed through ip protocols, Xively becomes a ideal platform for testing our crawl scheduling method. We built a prototype IoT search engine with EasiCrawl, where we continuously monitored 12 datastreams of 3 airqualityegg[15] sensors located in London and 3 sensors from Tokyo as our crawling targets. The only problem here is that most raw sensors on Xively don't support semantic senor description. So we add a describing layer, which translate raw data streams to event sequences, by generating a log for interesting events from the raw data. EasiCrawl schedule the crawler based on the event frequency from logs and the sleep plans of different sensors. For example, a sudden raise in temperature or violation of radiation threshold are taken as events, which will be stored as an item of sensor's description files. The exceeding of density threshold for CO, NO_2 are logged as dangerous events. The quick raise in temperature or quick changes in humidity may be caused by the miss-use of electrical equipments, which will also be recorded. In order to test the schedule performance with non-periodic sensors, we only log temperature and humidity events when people are at home, which is 6 p.m. to 9 a.m..

In this case, the parameter for Poisson process of the arriving event sequence are estimated periodically before EasiCrawl is invoked. Fig.10 is the raw sensor data of 4 different data streams, including CO, Humidity, NO_2, along with the corresponding scheduling result. The circle marks are the events recorded which users may care. The triangle marks are the actual crawls. For each crawl, the latency is calculated by adding up every interval between events and next crawl, which is illustrated by the humidity part in Fig.10. EasiCrawl can always hold a low latency level, which is 2 to 3 times lower than that of RandomCrawl. In this particular case, the average latency of the RandomCrawl is about 23 hours, while for EasiCrawl, the time is reduced to 13 hours.

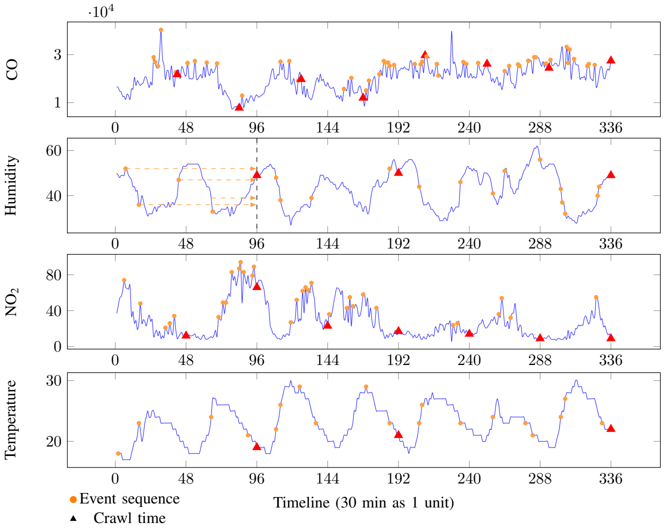

Fig. 10: Crawl Plan Generated by EasiCrawl for Data from Xively

VII. Conclusion

In this paper, we introduce EasiCrawl, a crawl scheduling strategy which runs an iteratively improving method. EasiCrawl can be used with sensors which have sleep behavior, which is ideal to be used in IoT search system. Effectiveness of EasiCrawl is also showed in simulations and a case study with Xively. In the future work, EasiCrawl are needed to be improved in two aspects. Firstly, an online scheduling version of EasiCrawl is needed, instead of running a periodically batch task. Secondly, a more sophisticated method should be designed for the situation when sensors' sleep plans are unknown.

Acknowledgment

This paper is supported by the International S&T Cooperation Program of China (ISTCP) under Grant No. 2013DFA10690, the "Strategic Priority Research Program" of the Chinese Academy of Sciences under Grant No. XDA06010403, and the National Natural Science Foundation of China (NSFC) under Grant No.61100180.

References

[1] Dennis Pfisterer and Kay Römer. SPITFIRE: toward a semantic web of things. *IEEE Communications Magazine*, (November):40–48, 2011.

[2] CoRE Working Group. Constrained RESTful Environments (CoRE) Link Format. *IETF CoRE*, pages 1–22, 2012.

[3] Mike Botts and Alexandre Robin. Opengis sensor model language (sensorml) implementation specification. *OpenGIS Implementation Specification OGC*, pages 07–000, 2007.

[4] Xively. https://xively.com/, 2015. [Online].

[5] UPnP Forum. http://www.upnp.org/. [Online].

[6] Apple Bonjour. https://www.apple.com/hk/en/support/bonjour/, 2002. [Online].

[7] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[8] Kamin Whitehouse, Feng Zhao, and Jie Liu. Semantic streams: A framework for composable semantic interpretation of sensor data. In *EWSN*, pages 5–20, 2006.

[9] Dennis Pfisterer and Kay Römer. SPITFIRE: toward a semantic web of things. *IEEE Communications Magazine*, (November):40–48, 2011.

[10] Michael Compton, Payam Barnaghi, Luis Bermudez, and et al. The SSN ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*, 17:25–32, 2012.

[11] Junghoo Cho and Hector Garcia-Molina. Synchronizing a database to improve freshness. *SIGMOD*, 29:117–128, 2000.

[12] J. L. Wolf, M. S. Squillante, P. S. Yu, and et al. Optimal crawling strategies for web search engines. *WWW*, page 136, 2002.

[13] James R. Challenger, Paul Dantzig, Arun Iyengar, and et al. Efficiently serving dynamic data at highly accessed web sites. *IEEE/ACM Transactions on Networking*, 12(2):233–246, 2004.

[14] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[15] Air quality egg. http://airqualityegg.com/, 2015. [Online].

APPENDIX A
ANALYSIS OF EXPECTED LATENCY

A. *Expectation of Sensor with Sleep Behavior*

In this section, we first introduce a simple way for deducing $E_f(t_s, t)$ when sensor works without sleep. Then we discuss the computation of expected latency $E_L[G(t)]$ where sensor has sleep behavior. At last we prove $E_L[G(t)]$ is a convex function of t.

We suppose the IoT sensor crawled start working at time 0 for convenience. The total expected latency when crawl at time t can be considered as an random process $G(t)$. Our goal can be rewritten to compute the expectation $G(t)$, written as $E[G(t)]$. Notice $E[G(t)]$ can be divide into different parts due to the count $N(t)$ of events which happen during $[0, t)$, then $E[G(t)]$ can be written as:

$$E[G(t)] = \sum_{n=1}^{\infty} E[G(t)|N(t) = n]P(N(t) = n) \quad (4)$$

$P(N(t) = n)$ is the probability of having n events during $[0, t)$, which follows Poisson distribution, where $P(N(t) = n) = e^{-\lambda t}(\lambda t)^n/n!$ holds. $E[G(t)|N(t) = n]$ can be resolved as $\frac{n}{\lambda}(t\lambda + e^{-\lambda t_m} - 1)$. t_m stands for the happened time of the m-th event. By taking $E[G(t)|N(t) = n]$ into (4), we get the close form solution of the expectation as equation (5).

$$E[G(t)] = (e^{-t\lambda} - 1 + t\lambda)t \quad (5)$$

For sensors with sleep behavior, we suppose sensor starts to work at time t_s, and begin to work with a sleep plan. Here we mark T_i^c as the i-th working cycle. $T_i^c = T_i^w + T_i^s$, where T_i^w is the i-th working cycle and T^w is the i-th sleeping cycle. It's hard to directly deduce an expectation for a Poisson process whose timeline is break up. However, with equation (5), we can compute the $E_f(t, t_s)$ by subtract the expectation cause by the previous crawls from $E[G(t)]$. In this recursive way, the expectation can be calculated. Here $E_h[G(t)]$ stands for the expectation to crawl a sleep enabled sensor at time t, $E[G(t)]$ still stands for the expectation if this particular sensor doesn't sleep. Start time $t_s = 0$, $E_f(t_s, t)$ can be written as $E_h[G(t)]$. Suppose time t locates in an working cycle, written as $t \in [t_s + \sum_{i=1}^{k} T_i^c, t_s + \sum_{i=1}^{k} T_i^c + T_i^w]$, where $k \in \mathbb{N}$. This assumption is based on the fact that, any crawl at a sleeping cycle can be bring forward to last work cycle for a better expected latency.

$$\begin{aligned} E_h[G(t)] = {} & E[G(t - T^L(K))] + \\ & \sum_{i=1}^{K} (E[G(t - T^L(K - i))] - \\ & E[G(t - T^L(K - i) - T_{K-i}^w)]) \\ & = (t - T^L(K))^2\lambda + \sum_{i=1}^{K} (2t - T_{K-i}^w) T_{K-i}^w \lambda \quad (6) \end{aligned}$$

Here $T^L(K) = \sum_{j=1}^{K} T_j^c$, which is time duration of first K cycles, and K is the count of cycle T^C. For any time t in a sensor working cycle, by summing those expectations before time t, the overall expectation of latency can be achieved when take sensor sleep cycle into consideration. The first factor of (6) is the expectation of last working cycle, and the sum is the expectation of the previous working cycle.

Expression (6) is a simplified form of $E_h[G(t)]$. We use this simplified version to show $E_h[G(t)]$ is indeed convex. Here we try to compute the one-order derivative on t.

$$\frac{\partial E_h[G(t)]}{\partial k} = \begin{cases} c_i + \gamma_i t & \text{if } t \in [T_i^L, T_i^L + T^w] \\ c_i & \text{if } t \in [T_i^L + T^w, T_{i+1}^L] \end{cases} \quad (7)$$

The result is a continuous linear piecewise function, where c_i and γ_i are constraints. The meaning of expression (7) is actually the gain speed of expected latency. With expression (7), the second order derivation $\nabla^2 E_h[G(t)] \geq 0$ in its domain, and the convexity of function (6) is proved.

B. *Expectation of Arbitrary Period*

Previous analysis computes the expectation with the start time at 0. The problem of computing $E_f(t, t_s)$ if $t \neq 0$, can be solved given the previous analysis. This problem can be divided into two sub-problem by value of t_s. Assume that $t_s \in [0, T_1^c]$, other case can be transformed to this by shifting t_s and t. If t_s is in a sleeping cycle, then expectation can just be calculated from the next work cycle. Otherwise, if t_s is in a working cycle, the expectation can be estimate by $E_h[G(t)]$ which start at 0 subtract the expected latency cause by the events arriving at time $[0, t_s]$.

We define $t' = t - (t_s \bmod T_1^c)$ and $t'' = t - T_1^c$ for convenience, where T_1^c is the length of first cycle. Here the minuend means the expectation at time point t of latency during time period $[0, t_s]$. If t_s is at a hibernate cycle, then the result expectation is just as the one which t_s equals next duty cycle, which is $E_h[G(t')]$, which can be written as follows.

$$E_f(t, t_s) = \begin{cases} E_h[G(t')] \text{ if } t_s \in [T_1^w, T_1^c) \\ E_h[G(t'')] - E[G(t'')] + E[G(t - t_s)] \\ \text{otherwise} \end{cases} \quad (8)$$

Combine expression (6) and (8), the expected latency of periodic sleep sensors can be computed.