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*—Search of the Internet of things is an important way for people to exploit useful sensors. Crawling the content of low power IoT sensors is a fundamental step towards building a generic IoT search engine. However, the crawl of IoT sensors is very different from that of Web. In IoT systems, many low-power sensors may sleep periodically, which results in invalid crawls and unpredictable latency. Besides, the energy consumption of crawl access becomes non-negligible. As a consequence, the traditional crawling schedule strategy is not suitable for IoT search engine. We first formulated the problem of crawling periodically sleeping sensors as a constrained schedule optimization problem, with the objective of minimized latency. Then, we developed an a heuristic schedule, named EasiCrawl to get the near optimal expected latency. At last, EasiCrawl is evaluated with simulations which show advantages over other greedy-based methods. A case study is also performed with real world data from Xively, showing that EasiCrawl has lower crawling latency and energy consumption than traditional Web-based search scheduling.**

I. Introduction

A large number of sensors have been deployed and connected to the Internet to collect physical information, which is provided for the human society to sense and control the physical space in an intelligent way. The extended Internet with sensors are known as the Internet of Things (IoT). With booming number of sensors in IoT, it is inevitable to develop a generic search engine for users and applications to find different kinds of IoT sensors. So, it can be expected that a generic IoT search engine can be used in variety scenes to improve the possibility of sharing sensors with others. In such a way, search of IoT can significantly reduce the redundancy of sensor deployment and facilitate construction of different IoT applications using method like IFTTT[1]. More specifically, in smart city applications, public sensors can be found in an ad hoc manner to provide predicates as traffic jam and queuing status. Smart home applications need searchable sensors and devices to detect human activities and provide realtime sensing of the indoor environment, which calls for sensor discovery mechanism. In recent future, unmanned systems like automatic driving and quad-copter delivery are likely to be heavily dependent on information gathered from nearby sensors or other IoT sensors, which also needs IoT search.

The IoT search engine is technically realizable with the help of standardized protocols and semantic descriptions of sensors[2]. Specifically speaking, the descriptions crawled by IoT search engines include not only static contents, but also dynamic records generated by the sensor. The realtime state of a sensor is described by a dynamic description method. Indeed, many research group are working on dynamic describing methods, such as CoRE[3] from IETF and SensorML[4] from OGC. With the description methods, IoT search system can be designed with technicals in traditional Web-based search. Fig.1 gives the architecture of an IoT search system. This IoT search system consists of the sensors on the left and a IoT search engine on the right. In this search system, we suppose every sensors support IP protocol. Sensors are treated as Web pages, where they can automatically update their description files. IoT crawlers crawl different sensors for these descriptions files, and copy these into a local repository at server side. This repository is an organized storage, with ~~whom~~ the user queries about IoT sensors can be accomplished. We assume the user only interest in the events monitored by corresponded 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.

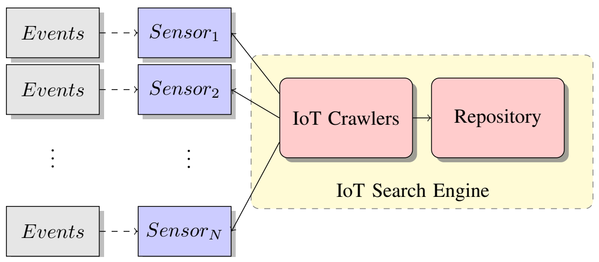

Fig. 1: Architecture of IoT Search System

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. [illegible]ly and the most importantly in this paper, IoT sensors behaves differently in sleep mode, whereas freshness of the response 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 timing, the freshness of query result will be impacted significantly. We use a clip of real world data to illustrate how the instinct of sensors' sleeping impact the crawl quality of IoT search system. Fig.2 is a data stream from a temperature sensor located in a smart home environment, who acts as a sensor for air-conditioner control. The quick variations and violation of threshold are taken as **interesting event**, which is logged into the description file immediately by the sensor. Suppose when users are not at home, these sensors will go into sleep mode. We marked all these interesting events out in the figure. The traditional

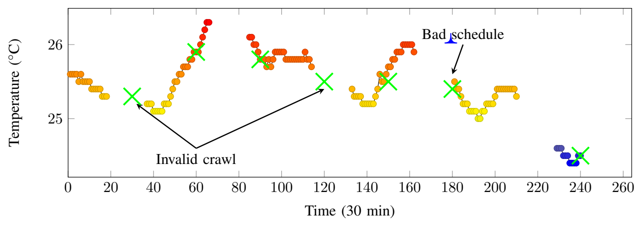

Fig. 2: Temperature Sensor in a Smart Home Environment

crawl scheduling method perform badly in this case, ~~because~~ lacking awareness of sensors' sleep. Traditional scheduler will use a periodicall~~y~~ crawl method to retrieve sensors' description files~~, causing~~ invalid access to sleeping ones, like what is showed at time 30 in Fig.2. Bad crawl ~~time cause by~~ delay across the whole sensor' sleeping cycle, at time 210, may ~~also~~ significantly ~~increase latency~~. 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. Thirdly, IoT sensors is not hyper-linked by their content, so link-based crawl method is infeasible for IoT search engine. In this paper, ~~we aim to accelerate IoT search by taking the second and third characteristic of sensors into considerations.~~ We propose a new server-side scheduling method, EasiCrawl, which can plan crawling process ~~with a guarantee of low latency when taking sensors' sleep behavior into consideration.~~

This paper has three contributions. (1) Although crawler scheduling is a well studied problem, to our ~~best~~ knowledge, our paper is the first to discuss crawling problem ~~whose target is~~ sensors with sleep behavior and energy constraints. (2) An iterative optimization approach ~~EasiCrawl~~ is proposed for crawler scheduling under different circumstances. A dynamic programming method ~~is put forward~~ as a subroutine to compute the optimal crawl plan for single sensors, ~~while~~ a fast greedy method ~~is design~~ as an complementary. (3) We evaluate EasiCrawl with simulation and real world data crawled from the IoT platform Xively[5]. ~~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 are introduced in section VI.

II. Related Work

There are lot of off-the-shelf technologies that can be used for IoT search systems. We investigated these network configure protocols used in LAN to automate domestic devices, ~~which have device discovery mechanisms~~ like UPnP[6] and Bonjour[7]. ~~While~~ widely used, they are actually not suitable for IoT applications, mainly because of ~~the~~ power. These auto-configuration protocols are designed for devices with continuous power supply. ~~With low power~~ sensors which may periodically enter sleep ~~mode, these protocol may fail and need to be modified. Also,~~ the device description language used by these protocol are relatively weak, which can not define the realtime state of IoT sensors. As a result, these protocols can not be widely used in IoT search system.

Beside off-the-shelf technologies, a lot of new research of IoT search are conducted recently. Many IoT search systems with different architectures have been proposed. ~~Due~~ to how the sensor information is gathered, these architectures can mainly be summarized into two categories, push-based and pull-based. In a push-based system, the integrity of data is ~~a~~ prior. Sensors periodically send~~s~~ their data and servers store the data once received. ~~Some of the~~ search requests of push-based systems are running on continuous datastreams~~, while other system collect data from sensors, and then~~ perform query directly in databases using SQL. Technologies like TinyDB[8] and IoT feeds[9] take IoT sensors as datasources, and process structured command to query the data stream generated by these sensors. This kind of IoT systems are easy to be implemented, but the redundancy of raw sensor data may causes problems in energy and data transitions. In many applications, not all data is need to be sent and uploaded to server, and this problem seems inevitable when sensors don't know exactly what the user want. The other side may be the solution to the requirement match problem. Pull-based architecture use semantic description, i.e. XML, to describe IoT sensors. Sensors update their description based on data collected, which make the behavior of ~~whom~~ very similar to a web page. This property makes search IoT with a traditional Web search engine possible. For example, Spitfire[10] use crawlers and modern indexing methods, along with semantic IoT technology as IoT sensor descriptions. Previous works of semantic IoT[11] and the propose of CoRE[3] devoted to standardize IoT sensor description languages~~, which indicated the possibility of~~ building a web-style IoT search system. The description files can be accessed by simply retrieving the ./well-know files used in the CoRE framework. Although the communication over-head of pull-based architecture is relatively heavy, the scalability make pull-based IoT search architecture very promising. EasiCrawl are implemented on a pull-based IoT search architecture similar to Spitfire.

Crawlers Scheduling is a well studied topic in the field of Web search. Previous works[12][13][14] analyzed how the crawler strategy affect 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. But in a IoT search system, the sleep behavior of sensors may significantly impact the latency of the crawl of information. ~~Also,~~ to our ~~best~~ knowledge, no research ~~is find~~ to study the crawl schedule with energy constrained crawl ~~targets. These two problem make IoT crawler's strategy very different from crawlers in a traditional Web search engine.~~

III. Formulation

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

EasiCrawl is invoked periodically during time period 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 ~~implied~~ for model simplicity, and finite numbers of time points can be used for crawl. We set up a discrete time table with a step length ε, each crawl must be preformed at one of these time point in the time table. Our goal is to find a property **schedule plan** Φ so that the ~~latency expectation~~ of all N IoT sensors is minimized. The schedule plan consists of two part. The first part is the **arrangement of crawls** $\hat{n}$ for each sensor i, ~~written~~ as $\hat{n} = n_1, n_2, \ldots, n_N$. We also use the symbol $\Phi(\hat{n})$ to indicate the schedule plan with arrangement $\hat{n}$. 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 sensors are different, so we define a factor ω_i as the expectation weight of sensor i. Here we use the notation $E_L(n_i)$ as the minimal **latency expectation** 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 latency expectation of all sensors which is ~~extended~~ as $\sum_{i=1}^{N} \omega_i E_L(n_i)$.

In addition, this problem follows two constraints. As previously described, ~~server load~~ and ~~IoT sensors'~~ energy consumption are ~~considered~~. During the time range T, the access time n_i for sensor i is limited to be ~~accessed~~ less than γ_i times. For the server-side, we set a constraint that the crawlers from the server can at most commit θ crawls during the time range T. ~~This models the largest bandwidth of server-side.~~ With the object function and constraints, we ~~arrived at~~ the following formula~~tion~~. 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.

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

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 of n_i and $E_L(n_i)$ can be determined.

IV. Crawling IoT Sensors with EasiCrawl

We first give out method framework of EasiCrawl, which iteratively improve our solution to achieve a latency optimal scheduling plan. Then technical details of subroutines of EasiCrawl is 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 arrange crawl chances $\hat{n}$ to the sensors according to their performance in reduction of latency expectation. This arrangement is done in an iterative manner. Firstly, an original arrangement is made in proportion to an estimated latency. ~~This estimation~~ 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 latency expectation most quickly. The step length for change is denoted as ϵ, which is 1 by default. With these analysis, the minimization problem can be transformed into the following two sub-problems. First, how to rearranging crawl numbers for each sensor under two constraints? Second, what's the best crawling plan for each sensor? Once the count of crawls 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 $i \leftarrow i-1$ and $j \leftarrow j+1$ 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: 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:         $\hat{n}' \leftarrow n_1, ..., n_i + 1, ..., n_j - 1, ..., n_N$
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:    $\hat{n} \leftarrow n_1, ..., n_{i'} + 1, ..., n_{j'} - 1, ..., n_N$
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 discretiz~~ation~~ 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 explain why this method leads to optimal schedule. Indeed, the correctness of EasiCrawl bases on the fact that the latency expectation 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 the constrains are convex, **problem** (1) **is indeed a convex optimization problem**, in which iterative method always points to a optimal solution. Next, we ~~show~~ that $E_L(n)$ is convex for sensors with sleep behavior. More specification of sensor's sleep plan is needed for this proof. First, we use $E_f(t_s, t)$ to represent the latency expectation of a time period, where t_s and t are the start time and end time. If the crawl plan are located at $t_1, t_2, ..., t_n$, then latency expectation of a single sensor can be re~~wrote~~ as $E_L(n) = \sum_{i=1}^{n-1} E_f(t_i, t_{i+1})$. We use $E_h[G(t)]$ as an alias for $E_f(t_s, t)$ for simplify. The convex of latency expectation $E_h[G(t)]$ is proved in the appendix. Time periods $[t_i, t_i + 1]$ shares a same expectation function $E_h[G(t)]$. The convexity of $E_L(n) = nE_h[G(t/n)]$ can be proofed by the convex preserve operations[15]. In this way the convexity of $E_L(n)$ can be proved. Our result in simulations also shows a supportive result for these analysis. In addition, the computation of $E_h[G(t)]$ and $E_f(t_s, t)$, which are both latency expectation 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 latency expectation $E_L(n_i)$.

B. Dynamic Programming Method for Optimal Crawl

In a real world IoT system, sensors' sleep plan make them difficult to crawl in a fix interval. Actually, the best crawl plan may be very different from a periodic crawl, especially when the sleep plan is not periodic. With the previous discretization process, we have a chance to enumerate all the possible crawl plans. The following dynamic programming(DP) method take ~~an~~ advantage of memorized search, which caches the solution of sub-problem to accelerate the enumeration process.

We use **timeline** to ~~refer~~ the discretized time table for a single sensor, which consist 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 used to formalize this problem. E are the edges of the graph, and V are the vertex. 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, because the last vertex must be chosen. Every points standing for the working cycle as graph vertex is written as v_i. 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 latency expectation gain of a crawl at time t_j whose previous crawl is at t_i. ~~Our problem~~ is, 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 ~~method~~ 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 T^2 times. So the time complexity of iterating process is $O(m^2 n)$, in which nm sub-problem are

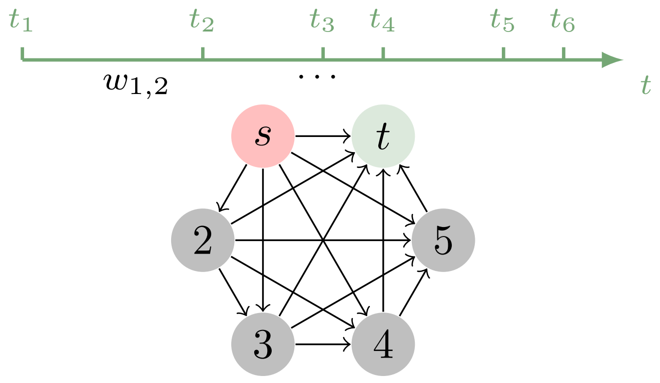

Fig. 3: Example of Non-periodic Hibernate Problem Transformation

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: $t_1, \ldots, t_n$
1: $f(v_i, n') \leftarrow \infty$, $p(v_i, n') \leftarrow 0$
2: for $n' = 1$ to $n-1$ do
3:     for $i = 2$ to $m$ do
4:         $f(v_i, n') \leftarrow \min_{k=1}^{i-1}\{f(v_{i-k}, n'-1) + w_{i,i-k}\}$
5:         $p(v_i, n') \leftarrow$ minimum $v_{i-k}$
6:     end for
7: end for
8: $t_n \leftarrow T$, $t_i \leftarrow$back trace from $p(v_i, n)$ for $t_i$
9: return $t_1, \ldots, t_n$
```

C. Greedy Method for Time efficiency

Previous DP method is a generic way to compute the accuracy crawl plan. Although it can return a optimal solution, the computation complexity is relatively high. Most of the time, there is no need to compute a optimal plan, especially when the computation resource is intensive. With more insights, we find crawl planning of a single sensor can be boosted in some special case. In this section, we introduce a greedy crawl planning method, which is a perfect approximation.

We start from the inner property of crawl planning for a single sensor. This problem actually can be formalized by ~~expression~~ 3. Here $E_f(t_i, t_{i+1})$ are the expectation function for a time period.

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

The result of this problem is actually $E_L(n)$, the minimal latency that can be achieved with n crawls. If both $E_f(c, t_i)$ and $E_f(t_i, c)$ are convex, where c is a constant, then this problem is a convex optimization problem, which can be solved using greedy method. In the follow analysis, we looked into two special situation, and explained why greedy method can not achieve the optimal solution in the most 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 $E_h[G(t)]$ [15] with induction. Here we donate $E_h[G(t)]$ as $f(t)$ for convenience. We start from the situation where only two crawling are allowed. As a result, the latency expectation 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 $E_h[G(n)] = nE[G(\frac{T}{n})]$. According to the result from appendix, $E_h[G(n)]$ has an close-form expression as $T(T\lambda/n - 1 + e^{-T\lambda/n})$. With this function, the minimum expectation can be calculated directly.

Then we illustrated that this intuition holds for the situation when sensor work periodically, with some constraints. Some points in the timeline can not be accessed, 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.

Algorithm 3 Fast Greedy Crawl Method

```
Input: $n$, $T$, $T^w$, $T^s$
Output: $t_1, \dots, t_n$
 1: $T^c \leftarrow T^w + T^s$
 2: $m \leftarrow T/(T^c n)$, $r \leftarrow (T/T^c) \bmod n$
 3: for $i = 1$ to $m$ do
 4:     $t_{m-i+1} \leftarrow T - T^c i + T^w$
 5:     if $i \neq 0 \wedge r \neq 0$ then
 6:         $t_{m-i-1} \leftarrow t_{m-i+1} - T^c$
 7:         $r \leftarrow r - 1$
 8:     end if
 9: end for
10: return $t_1, \dots, t_n$
```

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 solution ~~generated~~ 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.

V. Simulation and Evaluation

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

A. Settings

Latency expectation which are previously used as a optimization objective ~~are~~ taken as ~~the~~ measurement. Sensor scales N, total crawl limit θ, discretization step length ϵ, and time range T of scheduling ~~are tested during the following simulations. As the input,~~ 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. We ~~use~~ a random schedule method, called RandomCrawl, ~~as a comparison.~~ 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 simulation 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. 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

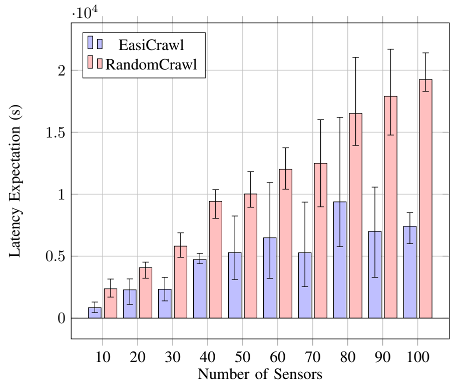

Fig. 4: Latency Expectation With Different Sensor Scales

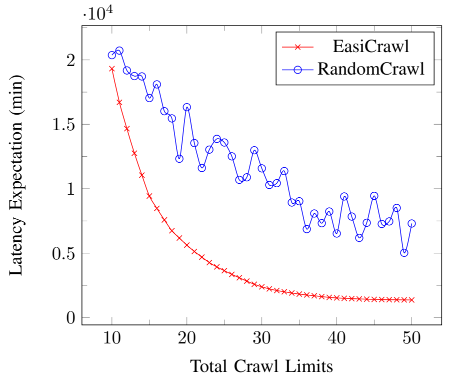

Fig. 5: Latency Expectation With Different Crawls

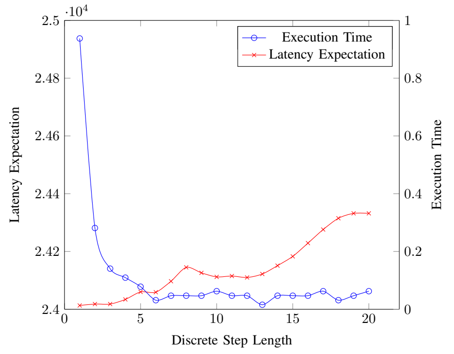

Fig. 6: Latency Expectation With Different Discretization Step

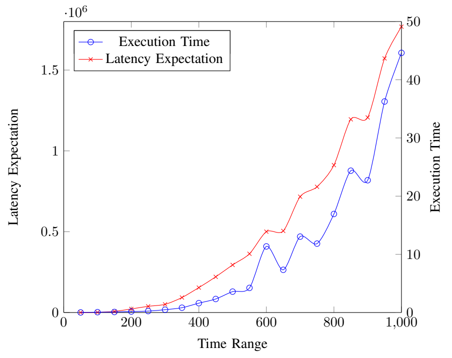

Fig. 7: Latency Expectation With Different Total Time Range

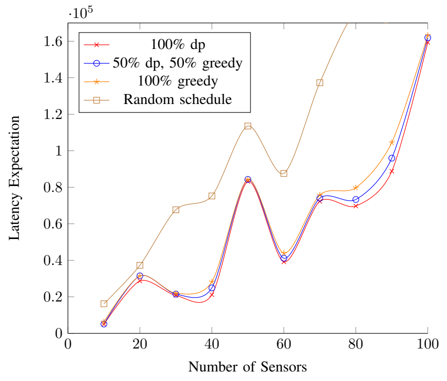

Fig. 8: Latency Expectation With Different Plan Methods

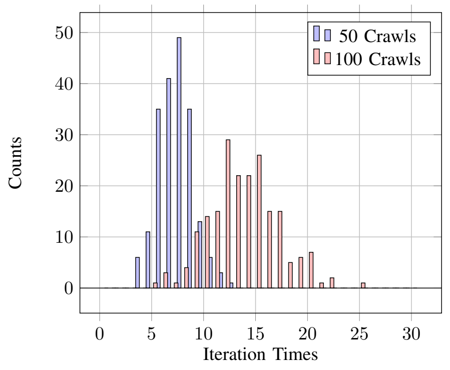

Fig. 9: Converge Speed under Different Crawl Times

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 latency expectations. Simulation result is plot in Fig. 4. In the result, EasiCrawl surpass the RandomCrawl by 2 times in reducing the latency expectation, even take error bar into consideration. This is mainly because 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 of 2, where minute is the 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 latency expectation 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 nodes which can used to crawl is large, leading to a higher computation cost.

Time range T directly affect 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, while the other half use greedy based method. The third group use~~d~~ 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 latency expectation become larger. When all the sensors use greedy method for scheduling, the latency expectation is the worst, but still much better than that of RandomCrawl.

The **speed of convergence** directly affect 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 unit. 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 grow bigger, a larger mean value is expected.

VI. Case Study With Xively

Xively is a platform which integrates IoT sensors. With thousands of sensors that can be simply accessed with through ip protocols, Xively become a ideal platform for our crawl method. We built a mini IoT search system to test our crawl scheduling method EasiCrawl, where we continuously monitored several datastream of 3 airqualityegg[16] sensors located in London and Tokyo as an example for our case study. Actually, accessing the sensors is accomplished by query the Xively database which hold the up-to-date sensor information. This makes no different for us than directly accessing a real sensor. One problem here is that most raw sensors on Xively don't support semantic senor description functions. So we add a describing layer, which translate raw data streams to event sequence, by generating a log for the abnormal events in the raw data. For example, a sudden raise in temperature or violation of radiation limit are taken as events, which will be stored as a item of description. We consider the high level of CO, NO_2, are danger events, and the quick raise in temperature or quick changes in humidity may be caused by the miss use of electrical equipments, which should be also recorded. We also consider the temperature changes in the night time not interesting, to test the effectiveness of scheduling non-periodic sensors.

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 event captured and recorded which users may care. The triangle marks are the actual crawls. EasiCrawl can always hold a low latency level, which is 2 to 3 times lower than that of RandomCrawl.

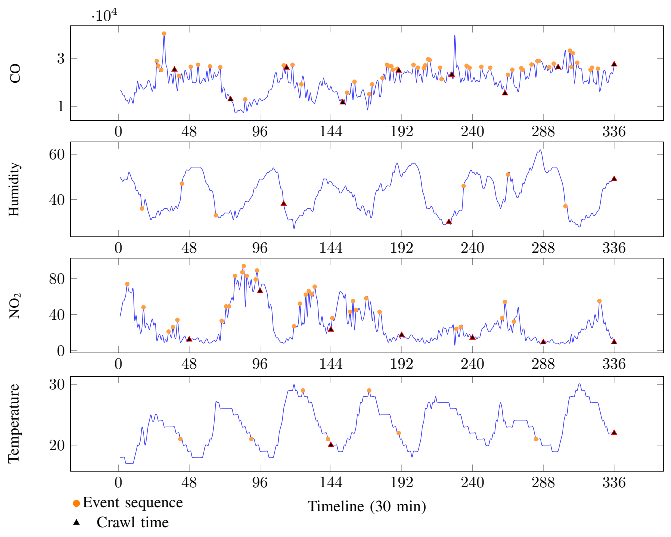

Fig. 10: Crawl Plan Generated by EasiCrawl for Data from Xively

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 sleep periodically, which is ideal to be used in IoT search system. Effectiveness of EasiCrawl is also showed in simulations and a case study with Xively. The drawbacks of EasiCrawl lies in the blindness when crawling sensors whose sleep plan is unknown. So, automatic parameter choosing mechanism is urgently needed. Besides further works should be done to enhance the description layer in our prototype.

Acknowledgment

The authors would like to thank...

References

[1] IFTTT. https://ifttt.com/. [Online].

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

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

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

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

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

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

[8] 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.

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

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

[11] 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.

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

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

[14] 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.

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

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

Appendix A
Analysis of Latency Expectation

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 latency expectation $E_L[G(t)]$ 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 latency expectation 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:

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

$P(N(t) = n)$ 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).

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

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 latency expectation.

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

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.

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

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 latency expectation. 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 latency expectation 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.

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

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