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

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

***Abstract*—The search of IoT(Internet of things) is an important way for people to exploit useful sensors. In this paper, we study the problem of crawling the content of low power IoT sensors, which is a fundamental step towards building a generic IoT search engine. The crawl of IoT sensors is very different from that of Web. In IoT applications, many low-power sensors may sleep periodically, causing invalid crawls and unpredictable latency. Also, in IoT systems, the energy consumption of crawl access becomes non-negligible. With these sensors widely used in IoT systems, traditional crawling schedule strategy is not suitable. In this paper, a new sleep-aware schedule method, EasiCrawl, is proposed to solve the scheduling problem of crawling periodically sleep IoT sensors. We first formulated this as a constraint optimization problem, using the expectation of latency to measure the realtime performance. With this formalization, we developed an iteratively improving method, which can get the near optimal latency expectation. At last, EasiCrawl is evaluated with simulations. A case study is also performed with real world data from xively.com, showing that EasiCrawl is more suitable for IoT search than traditional Web-based search scheduling.**

I. Introduction

Searching of IoT(Internet of Things) is a basic step for users and applications to find different kinds of sensors and IoT sensors. A generic IoT search engine can be used in variety scenes, which can improve the possibility of sharing sensors with others, which can significantly reduce the redundancy of sensor deploying. Also, different applications composition method like IFTTT[1] can be easily implied with the help of IoT search engine. More specifically, in smart city applications, public sensors can be found in 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 human-centric sensing, which calls for a realtime sensor discovery mechanism. In recent future, unmanned systems like automatic driving systems and quad-copter delivery systems are likely to be heavily dependent on information gathered from nearby sensors or other IoT sensors, which also needs IoT search.

However, even with the help of standardized protocols and semantic descriptions of sensors, the design of IoT search engine is not a trivial task. The main problem lay in the internal instinct of IoT. Firstly, IoT sensors is not connected by their content, which is precisely what user cares, link-based crawl method is infeasible for IoT search engine. Secondly, IoT sensors, whose content can be highly dynamic, behaves differently from Web page, many of them may enter sleep mode periodically, meanwhile the realtime performance of search is a critical measurement. In this paper, we propose

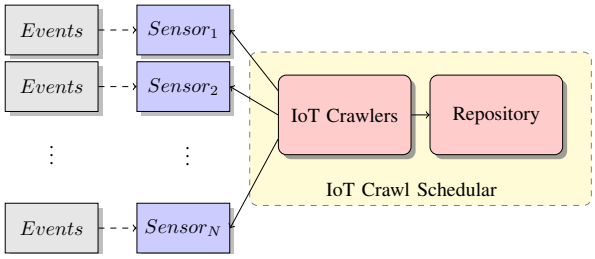

Fig. 1: Architecture of Pull-based IoT Search System

EasiCrawl, which is devoted to solve the sleep-aware IoT crawl scheduling problem.

The built of a generic IoT search engine is technically practical with researches using semantics[2]. The conception "description" is an abstraction of the state of a sensor, which means descriptions crawled by IoT search engines include not only static contents, but also dynamic records generated by the sensor itself. Indeed, many research group are working on dynamic describing methods, such as CoRE[3] from IETF and SensorML[4] from OGC.

We are building an IoT search system. Consider about the stability and standardization advantages, we use the pull-based architecture as the basic framework to study IoT crawl scheduling problem. The architecture of this pull-based IoT search system is illustrated by Fig.1. In a pull-based IoT search system, we treat sensors as Web pages, which can automatically update their description files. We suppose every sensors in this system supports IP protocol. IoT crawlers crawl different sensors for these descriptions files, and copy these into a local copy at server end, called repository. The 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 capture the event stream, and continuously store these events, and update their description files correspondingly. In this manner, the description file reveal the realtime state of sensors. IoT search engine regularly access the IoT descriptions, then updates the related repository. CoAP with CoRE resource description method can be used as standardized protocol for sensors and devices, and the description messages are listed in the ./well-know file used in the CoRE framework.

However, the sleep behavior of sensors and possible energy constraints makes it critical to choose crawling frequency and strategies for IoT crawlers. EasiCrawl is an appropriate strategy which takes the above requirements into consideration. It's a scheduling strategy implemented at the server side as a role of IoT Crawl Scheduler. Our design principle is to minimize the information latency during a period while meet the energy constraint of IoT sensors, given sensor sleeping schedule and the frequency of event. Here we clarify that, if a sensor is sleeping, we indicate that all its communication and event detection function are disabled. The insight of the solution lies in the convexity of the function between crawl times and latency for a single sensor. If convexity of the object function holds, the minimization problem can be solved by the follow two sub problems: (1) How to decrease the object function by rearranging crawl times for each sensors during a fix length of time? (2) What's the best scheduling strategy for each sensor? Luckily, the convexity can proof with some constraints. EasiCrawl is implemented in a iterative way based on these analyze. For each steps, EasiCrawl compute the minimum sum of latency that can achieved by the current crawler arrangement, and then rearrange to see if the result can be improved. Our paper is organized as follows.

This paper has three contributions. (1) Although crawler scheduling problem is well studied, to our best knowledge, our paper is the first to discuss crawling problem in IoT system. (2) A practical approach EasiCrawl for crawler scheduling under different circumstances is proposed for this model, which is evaluated in simulation. (3) We studied the real world data crawled from the IoT platform xively.com, which indicates the feasibility of our methods. We formalize the optimization problem in section II, then introduce EasiCrawl in section III. Simulations and a case study are preformed in section IV.

II. Related Work

Crawlers Scheduling is a well studied topic in the field of Web search. Previous works[5][6][7] 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. However, In a IoT search system, the realtime requirements can be more strict while some inherent properties of IoT like the sleep of IoT sensors and resource constraints, make IoT crawler's strategy very different from crawlers in a traditional Web search engine.

The schedule strategy has intimate connections with IoT search architectures. Many different architectures of IoT search have been proposed, which can mainly be divided into two different architectures, push-based or pull-based. In a push-based system, the realtime performance is a prior. Sensor data are periodically sent and stored in servers, and search request are actually running on continuous datastreams. Many practical applications collect data from sensors, and 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 architecture are easy to implement, but can be clumsy sometime. In many applications, not all data is need to be sent and uploaded to server, and this problem seems inevitable because it's almost impossible for sensors to know exactly what the user want. On 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, Dyser[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. Although the communication over-head of pull-based architecture is relatively heavy, and when lacking of prior information, most crawlers schedule methods are actually myopic, the scalability make pull-based IoT search architecture very promising.

We also look into technologies used in LAN to automate home devices, which may also be used in IoT applications. For example, UPnP[12], Bonjour[13] and other prevalent network configure protocols. But they are not perfectly suitable for IoT applications, mainly because the device description language can not define the realtime state of IoT sensors. Also, most of this configure protocol can only be used in LAN environment, with limit amount of devices.

III. Formulation and Measurements

EasiCrawl are implemented on a pull-based IoT search architecture, where any events captured by IoT sensors are stored in their semantic descriptions. Crawlers periodically visit sensors to copied the description to the server. The follow assumptions are set to simplify sensors' working model. (1) Events that users care arrived at each IoT sensor in a Poisson manner, where the time interval between any two event follows exponential distribution. (2) In the sensors are in sleep mode, they can be accessed by crawlers, but no new events can be captured. (3) User only cares about the event latency, which is the object function for optimization. (4) Sensor cannot be accessed too frequently. (5) During time range T, the server has a limited crawl time.

EasiCrawl is periodically work schedule strategy, which is called in a cycle whose length is T. We call T as the time range of EasiCrawl in the following paper. Suppose there are totally N sensors needs to be re-crawled, while crawler can perform at most M times of crawls during the period $[0, T]$. Our goal is to find a schedule strategy ϕ, so the latency expectation of all N IoT sensors is minimized. For the model simplicity, a discretization method is implied. We set up a discrete time table $\mathbf{T}$ with a discretization step of ϵ, where each crawl must preformed at the time point in the time table. Crawl schedule $\phi_{i,j} \in \{0, 1\}$, and $\phi_{i,j} = 1$ means sensor i will be crawled at time j, where time $j \in \mathbf{T}$. The total access count of all N sensors during $[0, T]$ can be represented as $\|\phi\|_1$. For each sensor i, the total crawl time is marked as $\sum_{i=1}^{N} \phi_{i,k}$. Considering the difference between sensors, we define ω_i as the expectation weight of sensor i. The latency expectation, which is the final object function, can be evaluated as $\|\omega_i E(\phi)\|_1$, where ω_i is the weight parameter for the latency of sensor i.

In addition, this problem must follow several constraints. Server load and IoT sensors' energy consumption are two main limitations. These constraints can be formalized directly: during any period $[0, T]$, the access time C_i for sensor i is limited to be accessed less than γ_i times. For the server load, our settings are similar to the formal research[6], where the crawlers from the server can not commit more that θ crawls during time period$[0, T]$. With the object function and constraints, we get the following optimization problem:

$$\begin{array}{ll} \min_{\phi} & \|\omega_i E(\phi)\|_1 \\ \text{s.t.} & \|\phi\|_1 \leq \theta \\ & \sum_{\forall t \in \mathbf{T}} \phi_{i,t} \leq \gamma_i \\ & i = 1, \ldots, N \end{array} \tag{1}$$

Problem (1) is a generalized formulation, which does not content specific sensor model and sleep behavior of IoT sensors. Due to different requirements between IoT applications, it's not rational to use an universal model to formalize the sleep pattern. For many sensors, the sleep pattern is unknown, while others may have been per-arranged. In the follow section, different working model of sensor is discussed.

The latency talked in this paper is defined as the expectation of the duration from when the events are captured by sensors to the time when crawler copy these events to the server side repository. In other words, the latency is opposite to the freshness of all the events user get from the repository. In this paper, we mainly study the measurements of latency expectation when the sleep pattern is previously known. To measure the latency expectation at a certain time t, we define $E_f(t, t_s)$ as the expectation when crawler crawls the IoT sensor at time t, and t_s is the last time this sensor is crawled. In addition, we use the annotation $G(t)$ as latency expectation for crawling at time t.

Computing latency expectation of a continuously working sensor is a well-studied problem[5], [6]. This value can be deducted by summing up checking the conditional expectations of different number of events happened during the work time. For sleep enabled case, instead of using the complex close form expression to calculate latency expectation, we developed an relaxed measurement, which neglect the exponential part, value of which is close to 0 when t is relatively large. Fortunately, the latency expectation of a sleep enabled sensor is convex function of t. The details of latency measurement are presented in the appendices.

IV. Crawling IoT Sensors with EasiCrawl

We first give out our method to iteratively improve our solution, in order to achieve the optimal scheduling strategy, then introduce technical details of these subroutines, 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 distribute crawl chances to sensors according to their performance in reduction of latency expectation. EasiCrawl use an iterative method to improve the final schedule plan step by step. In each iteration, it compute the optimal or near optimal schedule according to the sensor's future sleep plan, and try to redistribute the arrangement for a better global latency expectation.

Suppose the latency expectation of sensor i is $E_L^i(n)$, given the number of crawls n_i. The object function of formulation 1 can be rewrite as $\sum_{i=1}^{N} \omega_i E_L^i(n_i)$. The total crawl time constraint can be taken as $\sum_{i=1}^{N} n_i < M$ and the energy constraint of single sensor can be rewrite as $n_i < \gamma_i$. Once crawl times n_i is chosen, the optimal crawl strategy for a single sensor is determined. Here, every function of $E_L(n)$ is non-decreasing. To iteratively improve the global result, we design an approach which is similar to SMO[14]. First we dispatch the M crawl time to N sensors according to ω_i. Then 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. The above is the main idea of EasiCrawl.

In addition, the feasibility of this method is based on a preliminary that $E_L(n)$ are convex. To proof this, since crawl times n is an discrete value, we can just illustrated the latency decrease of adding a new crawl to a sensor is less than that of the previous addition. Here we assume the latency expectation function $E_h[G(t)]$ is convex, which is proofed in the appendix. It's easy to show the convexity of $E_L(n)$ if the t is continuous, where the convexity of $E_L(n) = nE_h[G(t/n)]$ can be proofed by the convex preserve operations. However, the time nodes t used in the follow method is discretized, and the sleep plan of sensors are actually not continuous. In this sense, the convexity of $E_L(n)$ are verified in the evaluation section by simulation, where the result is quite supportive.

In this paper, sensors are classified into 3 types, which can be solved mainly by 2 kinds of method. We proposed a dynamic programming method for the optimal scheduling, but find the time complexity is somewhat unacceptable. So, a greedy method is designed as a complementary. We also discuss the situation when the sensors' sleep plan is unknown in the following part.

B. If Sleep Plan is Unknown

To our best knowledge, for the unknown sleep pattern case, if there are no previous knowledge for the sleep behavior, the detection and scheduling can be formed as an multi-armed bandit problem, which is known to be NP-hard. If there are no latent pattern for sleep behavior, then there will be not optimal strategy expect for periodically sensor querying, which is definitely an inefficiency schedule method. In EasiCrawl, we implement a crawl strategy which try to crawl the sensors as evenly as possible.

C. Sleep Plan Known, But Non-periodic

In a real world IoT system, sleep plan of sensors are likely to be non-periodic. Also the crawling interval maybe much larger than T_h. If the length of working cycles are neglected and represented as time points, then problem of searching optimal crawling can be transformed to a search problem, which can be solved by dynamic programming.

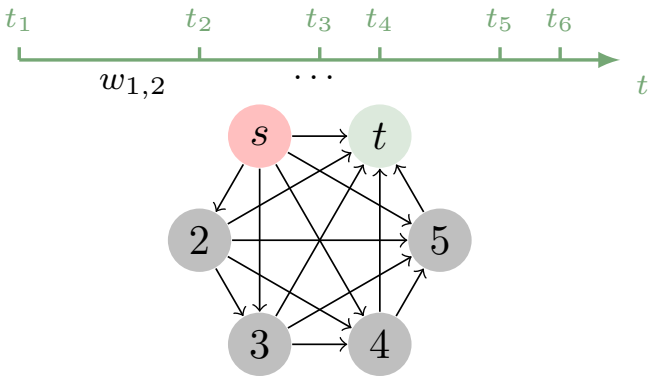

Fig. 2: Example of Non-periodic Hibernate Problem Transformation

Here a directed acyclic graph(DAG) model $G(E, V)$ is used to formalize the non-periodic hibernate problem. E are the edges of the graph, and V are the vertex. We suppose there are m vertex. If there are T' time points in the timeline which stand for the non-periodic working cycle. Corresponding, there are exactly T' vertex in this graph, from number 1 to number 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. Now we take every points which stand for the working cycle as graph vertex, written as v_i. We defined that, 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, where t_i and t_j are all . A weight value w_{ij} is set to $e_{i,j}$, whose value is $E_f(t_j, t_i)$, the 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 along. This transform process can be illustrated by Fig. 2.

With an acceptable scale of time range T and crawl time n, this problem can be solved by dynamic programming. The solution state with a vertex v_i and the available crawl time n' left can be used as an sub-problem. Noticing the optimal expectation can be computed by adding weight $w_{i,j}$ with all the expectation known in sub-problems. The recursive relationship is summarized as follows:

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

$f(v_i, n')$ 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 The computation complexity of iterating process is $O(T^2 n)$, in which nT sub-problem are computed and each computation will cost n 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 complexity is still $O(T^2 n)$

Algorithm 1 Latency Minimum Non-periodic Crawl Method

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** $t_1, \ldots, t_n$

D. Periodic Sleep Sensors

Previous introduced method is a generic method for any sensor whose sleep plan is known. Although it's proven a optimal solution, the computation complexity is relatively high. So next, we introduce a crawling strategy for single sensor under the situation that sensors' sleep plan is periodic. In the following paper, we first focused on the problem of scheduling crawls for non-sleep sensors, which has a direct connection with the periodic sleep case. In the situation when IoT sensors continuously work, an intuition is to evenly distributed the crawling action across the crawling schedule. Here a proof of this proposition is given.

Lemma 1: In a continuous situation where no sensor hibernates, the optimal strategy is to scatter the crawl action over the timeline $t \in [0, T]$ as evenly as possible, where T is a constant.

Proof: The evenly crawl proposition here can be proved by the convexity of $E_h[G(t)]$ [15]. Here we donate $E_h[G(t)]$ as $f(t)$ for convenience. We start the proof with a simple situation where only two crawling are allowed. As a result, the whole expectation gain during $[0, T]$, which is marked as $F(0, T)$, is divided into two part, $f(t)$ and $f(T-t)$. With the convexity of $f(t)$, we get $2f(T/2) < f(T/2-\epsilon) + f(T/2+\epsilon)$ for $\forall \epsilon > 0$. By substituting $F(T) = f(t) + f(T-t)$ into it, we get $F(T/2) < F(T/2+\epsilon)$ for $\forall \epsilon > 0$, indicating $T/2$ is the minimum point of $F(t)$. This conclusion can be generalized to a normal situation where multiply crawling are committed with the following induction. Now suppose there are n crawling action, among which first $n-1$ action evenly divide the timeline. The time length of n-th crawling action does not equal to the adjacent one's, and we claimed this strategy is optimal. For any adjacent pair divided by a single crawling action, if the division is not evenly, we can choose the evenly one as a new strategy, which can lead to a larger $f(t)$ according to the proposition above, which is a contradiction. ■

With Lemma 1, we can get the exact increment of expectation when different crawling times are implied to a same sensor. This can be used to minimize the overall expectation of repository latency, which formalized as $\|\phi\mathbf{T}\|_1$ above. Here the relationship between crawling time and $E[\phi(n)]$ can be computed simply as follows, where $\phi(n)$ stands for the optimal strategy with n crawls, and T is the time period within consideration.

$$E[\phi(n)] = nE[G(\frac{T}{n})] \tag{3}$$

Function (3) can be rewrite to an close-form expression as $T(T\lambda/n - 1 + e^{-T\lambda/n})$, which is convex. with this function, the minimum expectation can be calculated directly.

Here we also study the relationship between n and the minimum latency expectation, but try to be more pragmatic. With the optimal strategy $\phi_i(n)$ for sensor i which is only related to the crawling time n, we can substitute functions (9) and (5) into (1) to get the latency expectation of non-hibernation version. But with hibernation, at some time points the sensors can not be accessed, resulting in a complex piecewise expectation function, which makes the relationship between n and $\phi_i(n)$ unclear and completely theoretical approach likely impossible. Here we focus on the situation when $T_w < T_h$ and $n < T/\Delta T$. Applications which hold $T_w \leq T_h$ is rare, while if $n < T/\Delta T$ is just the same as the push situation we talked before. A concrete formulation extends from 1 is as follows, where $\mathbf{t}$ is the set of target time sequence that a crawl action can be scheduled to, and $k \leq 0$. Here an intuition of the 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 ΔT. We uses the same symbols as the previous.

Lemma 2: If the time interval of an adjacent pair of crawls at t and $t + \Delta t$ is an integral multiple of duty cycle, then the optimal crawl time t follows $t \bmod \Delta T = T_w$.

Proof: The difference between expectations with two different time intervals, which have the different offset and have the same length multiple of duty cycle, can be written as $E_f(t, t_s) - E_f(t + \Delta t, t_s + \Delta t)$. The expansion of $E_f(t, t_s)$ is $E_h[G(t'')] - E[G(t)''] + E[G(t - t_s)]$, notice that $E_h[G(t'')]$ has a lower change rate compared with $E[G(t)]$, whose coefficient is $T_w/\Delta T$ instead of 1. This indicates that the previous difference is a non-increasing function, which the optimal is the minimum when $\Delta t = T_w$. ■

The solution of the general situation is very similar to the previous when both t and $t + \Delta t$ locate in working cycles. However, when crawl $t + \Delta t$ locate in a hibernate cycle the expectation will become 0. Also, the result will become opposite when t locate in a hibernate cycle while $t + \Delta t$ locate in a working one. In this situation, the expectation will become larger with more time of work cycle included into the expectation area. With these observations, we can design a strategy that in some circumstances, the optimal solution can be retrieved.

Algorithm 2 Heuristic Method of Latency Minimum Periodic Crawl

Input: n, T, T_w, T_h
Output: $t_1, \ldots, t_n$

```
m ← T/(ΔTn), r ← (T/ΔT) mod n
for i = 1 to m do
    t_{m-i+1} ← T − ΔTi + T_w
    if i ≠ 0 ∧ r ≠ 0 then
        t_{m-i-1} ← t_{m-i+1} − ΔT
        r ← r − 1
    end if
end for
return t_1, ..., t_n
```

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.

V. Simulation and Evaluation

In the evaluation section, we first test the performance of EasiCrawl under different parameters, and confirm the effectiveness with real events sequence. Latency expectation which are previously used as a optimization object are taken as measurement. Then, we carry out a simulation comparing efficiency and computation complexity of 3 different crawl methods implemented in EasiCrawl. Here the latency sum of generated events and executing duration are considered as measurements. Sensor scales N, total crawl limit θ, discretization step length ϵ, and time range T of scheduling are tested during the following simulations. We use a random schedule method, called RandomCrawl, to illustrate the effectiveness of our method. 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 event are captured. In addition, the time unit of these simulation are all minutes, and we take 1 min as the minimum unit that can be used for scheduling. All these simulation are implemented on a PC with MATLAB.

A. Sensor Scales N

We first study the impact of sensor scale. We mainly test the dynamic programming schedule method of single sensor in this simulation. All the sleep plan of sensors are previously known and non-periodic, actually we use a randomized method to generate the sleep plan for each sensor. 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 latency expectations. Simulation result is plot in Fig. 3.

Simulation shows a promising result. The latency expectation of EasiCrawl is far below the random one, even take error bar into consideration. This is mainly because the theoretical optimal property of EasiCrawl when given a sequence of discretized time nodes.

B. Total Crawl Times θ

The total number of crawls which the server can at most start is an important constraint for EasiCrawl. 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.4. The result first show the advantages of EasiCrawl against RandomCrawl. More important, we notice there's a submodular trends for total crawls, which means the gain of server side crawl ability pay less when limits grow larger, and this result is theoretically reasonable. This suggest us to find a proper trade-off of server cost and schedule efficiency.

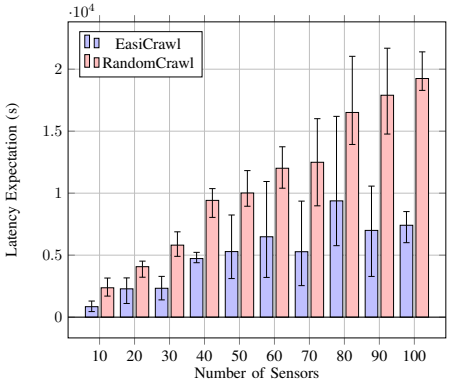

Fig. 3: Latency Expectation With Different Sensor Scales

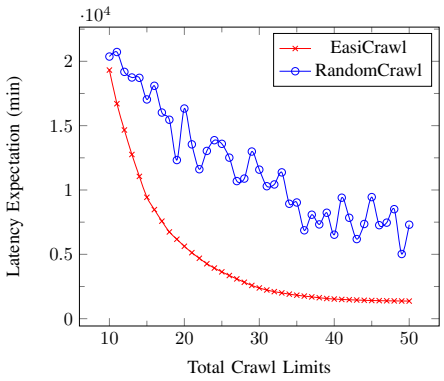

Fig. 4: Latency Expectation With Different Crawls

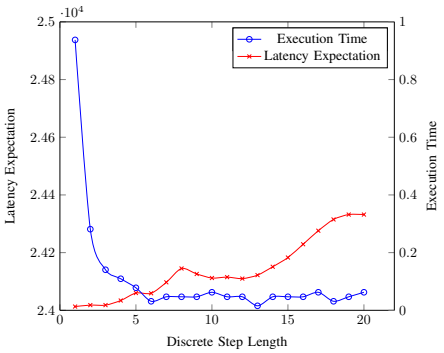

Fig. 5: Latency Expectation With Different Discretization Step

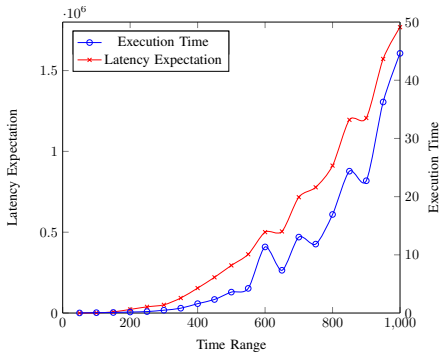

Fig. 6: Latency Expectation With Different Total Time Range

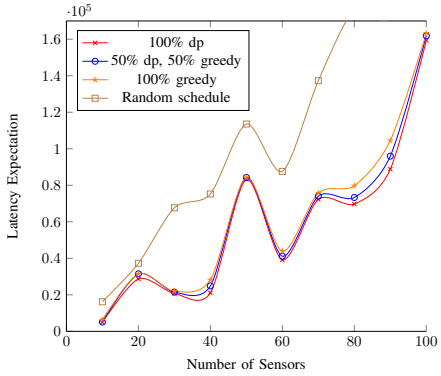

Fig. 7: Latency Expectation With Different Sensor Types

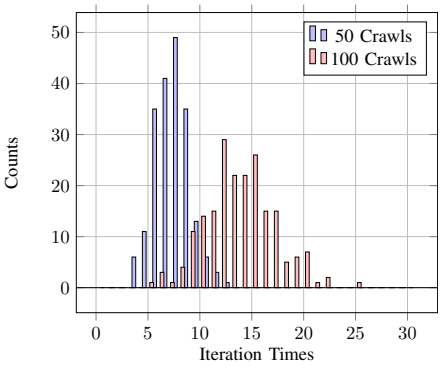

Fig. 8: Converge Speed under Different Crawl Times

C. Discretization Step Length ϵ

Time table of sleep plans should be discretized before starting scheduling. There 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. All the sensors are set to be scheduled with dynamic programming based method. The total crawl count θ is set to 100 times. We carried on 20 groups of test, where the discretization step length ϵ ranges from 1 to 20.

Fig.5 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 $O(T^2n)$, n is the crawl limit. When ϵ is relatively small, the number M of total time nodes which can used to crawl is large, which leads to a higher computation cost.

D. Time Range T

We use the same experiment settings as previous. 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. The time range T which EasiCrawl schedules on has a similar effect as the discretization step ϵ. The computation time showed in Fig.6 is very close to the analysis result, which is quadratic. The expectation share a similar shape because the distance between the distribution and the end time t can also be estimated as a quadratic function.

E. Senors' Types

In the previous evaluations, the DP(dynamic programming) methods are used for finding the best schedule for each sensors. In this test, both the greedy method and dynamic programming are both 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 used pure greedy based method. We also use the RandomCrawl method as a reference, which is the fourth group.

Result are showed in Fig.7. As our expect, DP method performs the best. 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.

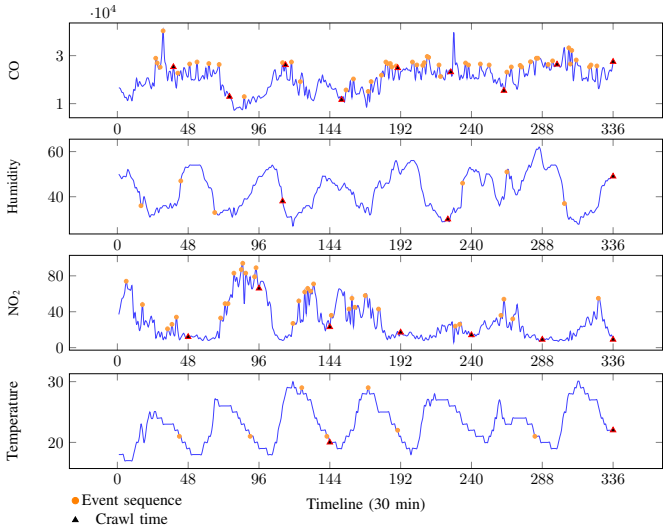

Fig. 9: Crawl Plan Generated by EasiCrawl for Data from Xively

F. Speed of Convergence

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 optimize 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.8 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 which can simply accessed with 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. Practically, what we mean by accessing the sensors is actually 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. A sudden raise in temperature or violation of radiation limit are taken as events, and will be stored as a item of description, which is the target to crawl.

In this case, the parameter for Poission process of the arriving event sequence are estimated previously. Fig.9 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. Here, 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 the miss use of electrical equipments, which should be also recorded. We also consider the temperature changes not interesting, only to test the effectiveness of scheduling non-periodic sensors. 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.

VII. Conclusion

In this paper, we introduce EasiCrawl, a crawl scheduling strategy which runs an iteratively improving method. EasiCrawl can be used with different sensors, 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. Also the fixed step length for improve arrangements, which is not a effect way for optimizing, needs to be improved.

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

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

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

[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] BM Elahi, Kay Römer, and B Ostermaier. Sensor ranking: A primitive for efficient content-based sensor search. *IPSN*, pages 217–228, 2009.

[11] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, and Kerry Taylor. The SSN ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*, 17:25–32, 2012.

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

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

[14] J Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. pages 1–21, 1998.

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

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

APPENDIX A
ANALYSIS OF LATENCY EXPECTATION

A. Expectation Without Sleep Mode

Although previous work[5] give out a similar result, we introduce a simpler way for deducing $E_f(t_s, t)$ when sensor works without sleep. For simplification, suppose the IoT sensor crawled start working at time 0. The total freshness latency when crawl at time t can be considered as an random variable $G(t)$, so 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 of events U_i 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, holding $P(N(t) = n) = e^{-\lambda t}(\lambda t)^n/n!$. $E[G(t)|N(t) = n]$ 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. Taking (4) , we get the close form solution of the expectation as equation (5).

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

B. Expectation With Sleep Mode

For the sleep situation, 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 $E_s[G(t)]$ 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 $E_s[G(t)]$.Suppose time t locates in an working period, 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.

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

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 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 too complex for practical use, so we figure out a way to simplify this. We assume $t\lambda$ is relatively large, which means in a work cycle multiply events are trends to be captured. Notice that if $t \to \infty$, $e^{-t\lambda} - 1 + t\lambda \to 0$, which means expectation (5) can be approximated by a quadratic expression $t^2\lambda$. With this observation, the expectation gain of a previous work cycle can be approximated using $E[G(t)] - E[G(t - \delta)] = \delta(2t - \delta)\lambda$, which is a linear function of t. Here δ is the length of working cycle. This indicates that latency expectation of a sleep enabled sensor can actually be approximated by a piecewise linear function. According to equation (6), each segment of the approximate function is the sum of different non-decrease linear expressions. By taking approximation of $E[G(t)] - E[G(t - \delta)]$ into (6), an approximation of $E_h[G(t)]$ is achieved as the follows.

$$E_h[G(t)] = (t - T^L(K))^2\lambda + \sum_{i=1}^{K} (2t - T_{K-i}^w)T_{K-i}^w\lambda \tag{7}$$

C. Convexity of $E_h[G(t)]$

Expression(7) is a simplified form of $E_h[G(t)]$, and it's direct to show this expression is continuous and derivable. Here we try to compute the one-order derivative on t.

$$\frac{\partial E_h[G(t)]}{\partial k} = \begin{cases} c_i + \gamma_i t & \text{if } t \in [T_i^L, T_i^L + T^w] \\ c_i & \text{if } t \in [T_i^L + T^w, T_{i+1}^L] \end{cases} \tag{8}$$

The result is a continuous linear piecewise function, where c_i and γ_i are constraints. The meaning of expression (8) is actually the gain speed of latency expectation. With expression (8), the second order derivation $\nabla^2 E_h[G(t)] \geq 0$ in its domain, and the convexity of function (7) is proofed[15].

D. 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 be divided into two situation according to the 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.

$$E_f(t, t_s) = \begin{cases} E_h[G(t')] \text{ if } t_s \in [T_1^w, T_1^c) \\ E_h[G(t'')] - E[G(t'')] + E[G(t - t_s)] \\ \text{otherwise} \end{cases} \tag{9}$$

Combine expression (6) and (9), the expectation of latency of periodic sleep sensors can be computed.